ビットパラレル手法によるアライメントアルゴリズム

馬場, 謙介
九州大学大学院システム情報科学研究院

于, 雲青
九州大学大学院システム情報科学研究院

村上, 和彰
九州大学大学院システム情報科学研究院

出版情報：情報処理学会論文誌. 数理モデル化と応用. 46 (SIG17(TOM13)), 2005-12-15. Information Processing Society of Japan
バージョン：
権利関係：

ビットパラレル手法によるアライメントアルゴリズム

An Alignment Algorithm by Bit-Parallelism*

馬場 謙介 Kensuke Baba† 于 雲青 Yunqing Yu 村上 和彰 Kazuaki Murakami

概 要

近似文字列照合問題は，二つの文字列と閾値が与えられて，片方の文字列の部分文字列のうち，もう一方の文字列との編集距離が閾値以下となるものを見つける問題である．この問題をビットパラレルと呼ばれる手法により高速に解くアルゴリズムが Myers により提案されているが，出力として編集距離だけではなくアライメントも求める場合には適用できない．本論文では，近似出現に対するアライメントについての正規形の概念を導入し，Myers のアルゴリズムを近似文字列照合に対するアライメント問題へ拡張する．

Approximate matching problem is, given two strings and a parameter, to find all substring of a string whose edit distance with the other string is at most the parameter. Myers introduced an efficient algorithm based on bit-parallelism for approximate matching problem. However, if alignment of strings is required as the answer of the problem, the algorithm can not be applied straightforwardly. In this paper, an idea of normalization of alignment is introduced and the algorithm by Myers is expanded to alignment problem.

1 序論

ヒトゲノムの解読をはじめとして，分子生物学が対象とする膨大な生命情報の理解を，文字列の解析というアプローチによって深めようとする研究が世界中で行われている．その中でも重要な概念は，二つの文字列が似ているかどうかということである．一致する文字の多い部分等の文字列間の類似性を見つけることは，相同性検索（ホモロジーサーチ）と呼ばれ，例えば，遺伝子の構造を予測したり，進化による関係を解析するために用いられる．二つの文字列が似ているかどうかを考えるための指標として，文字列間の類似度，あるいは距離を定義することが求められる場合がある．本論文では，最も基本的なものとして,「片方の文字列からもう一方の文字列を得るために，文字の編集操作（置換，挿入，削除）を最低何回行えばよいか」を表す編集距離 [13] について考える．編集距離は，編集の種類や文字の種類について重み付けを行うことによって，より一般的なものへの拡張が可能である．

編集距離を考える際,「片方の文字列中のある文字を，もう一方の文字列中のどの文字と比較するか」を表すのがアライメント [4] であり，編集距離を求める計算は，考え得る全てのアライメントに対応する編集操作の数のうち，最小のものを見つけるものである．[13] 中で，長さ m と n の文字列の間の編集距離を $O(mn)$ 時間で求めるアルゴリズムが導入されているが，本質的なアイデアは，生物学の分野で知られる動的計画法に基づくアルゴリズム [9] と同じものである [3, 8]．このアルゴリズムでは，対象となる二つの文字列のそれぞれの接頭辞についての編集距離を各要素とする行列（DP 行列）を再帰的に求めることで，計算の重複を避けている．また，DP 行列の各要素を計算する際に，再帰的な計算の一つ前の段階のどの値を元にしたかを覚えておくことで，アライメントを簡単に求めることができる．動的計画法に基づくアルゴリズムで，距離を計算する対象を二つの文字列の任意の部分文字列に拡張したもの [11] は，分子生物学の分野で多く利用されている検索システムの基礎になっている [10, 1].

近似文字列照合問題は，二つの文字列と閾値が与えられて，片方の文字列の部分文字列のうち，もう一方の文字列との編集距離が閾値以下となるものを見つける問題である．これは，二つの文字列間の編集距離を求める問題をより一般的にしたもので，動的計画法に基づくアルゴリズムの簡単な拡張により解くことがで

*An edited version of this report was published in: *IPSJ Transactions on Mathematical Modeling and its Applications, vol.46, no.SIG17(TOM13), pp.80–87, Information Processing Society of Japan, Dec, 2005.*

†Research and Development Division, Kyushu University Library, *baba@lib.kyushu-u.ac.jp*

きる．Myers [7] は，近似文字列照合問題を解くためのDP 行列の計算を，ビットパラレル [2] と呼ばれる手法により高速に行うアルゴリズムを提案した．ビットパラレル手法は，文字列間の類似度を二値の論理式の演算によって表すことで，DP 行列の成分のうち最大 w 個の値を並行して計算するものであり，このアルゴリズムの計算時間は，w をコンピュータのワード長として，$O(\lceil m/w \rceil n)$ 時間で抑えられる．(この手法に関する研究については [8] で詳しく述べられている.）しかし，ビットパラレル手法では，DP 行列の要素のうち明示的に値が求められるのは最後の行のみであり，前述の動的計画法についての手法でアライメントを求めるために必要な情報を得ることができない．アライメントを求めるための計算を新たに加えると，高速化が効果的でない場合がある．

本論文では，Myers のアルゴリズムをアライメント問題へ拡張する．これによって，近似文字列照合問題に対するアライメントを高速に求めるアルゴリズムを得る．同様のアルゴリズムとして，Hyyrö [5] は，編集操作によって異なる三種類の距離について，ビットパラレル手法によるアライメントアルゴリズムを提案している．このアルゴリズムは，ある文字列の対について距離を計算するものであり，近似文字列照合問題の場合は複数の文字列の対についての距離を考えなければならず，直接適用することはできない．また，ある文字列の対についても編集距離に対応するアライメントが複数存在し得ることから，求められるアライメントがそのうちのどのようなものかを示す必要があり，アルゴリズムの正しさも自明ではない．我々は，近似出現に対するアライメントについての正規形の概念を導入することで，アルゴリズムによって求められるアライメントを厳密に表現し，アルゴリズムの正当性を証明する．また，プログラムとして実装を行い，動的計画法に基づく単純なアライメントアルゴリズムとの計算時間の比較を行う．

2 準備

Σ を文字の有限集合とし，これを**アルファベット**と呼ぶ．Σ 上の文字列全体からなる集合を Σ^* で表し，空文字列を ε で表す．また，$\Sigma^+ = \Sigma^* - \{\varepsilon\}$ の表記を用いる．

ある文字列 s について，s の長さを $|s|$ で表し，s の i 番目の要素を s_i で表す．ただし，$1 \leq i \leq |s|$ である．文字列 $s_i s_{i+1} \cdots s_j$ を s の**部分文字列**と呼び，$s_{i,j}$ で表す．特に，$i = 1$ のとき**接頭辞**，$j = |s|$ のとき**接尾辞**と呼ぶ．便宜上，$j < i$ のとき $s_{i,j} = \varepsilon$ とする．また，$s^{-1} = s_{|s|} s_{|s|-1} \cdots s_1$ とする．

二つの文字列間の**編集距離**とは，片方の文字列から他方の文字列を得るために必要な編集操作のうち最小の回数である．ただし，編集操作としては文字の「挿入」,「削除」及び「置換」を許すものとする．二つの文字列 $p, t \in \Sigma^+$ の間の編集距離を $d(p, t)$ で表す．例えば，下のように，文字列「entry」から「empty」を得るためには，少なくとも 3 回の編集操作を行わねばならないので，二つの文字列間の編集距離は 3 である．

一致	置換	挿入	一致	削除	一致
e	n		t	r	y
e	m	p	t		y

二つの文字列 $p, t \in \Sigma^+$ 及び，ある閾値 $\delta < |p|$ が与えられて，t の部分文字列 t' が p の**近似出現**であるとは，$d(p, t') \leq \delta$ であることである．**近似文字列照合問題**とは，テキスト t 中のパターン p の近似出現を全て見つける問題である．

編集転写とはアルファベット $\{I, D, R, M\}$ 上の文字列であり，各文字は，それぞれ，三つの編集操作と「一致」を表している．つまり，ある文字列に対する編集転写は，対応する編集操作を順番に行うことで，その文字列を他の文字列に転写するものである．例えば，下のように，前述の編集距離の例に対応する編集転写は $MRIMDM$ である．

M	R	I	M	D	M
e	n		t	r	y
e	m	p	t		y

3 近似出現に対するアライメント

近似文字列照合問題は，二つの文字列間の編集距離を求める問題をより一般的にしたものである．単純に編集距離を求める場合には，実際の応用として編集距離の値が重要であることが多いが，近似文字列照合問題の場合，出現位置やアライメントと呼ばれる文字同士の対応が必要なことがある．本節では，アライメントを編集転写によって表し，近似文字列照合問題に対するアライメントを定義する．

近似文字列照合問題に対するアライメントは自明ではない．その理由のひとつは，ある近似出現に対応する編集転写が複数存在し得ることである．例えば，前節の例に対応する編集転写は，$MRIMDM$ の他に

$MIRMDM$ や $MRRRM$ もある．ここで，編集転写の正規形を以下のように定義する．

定義 1 文字列 $p \in \Sigma^+$ から $t \in \Sigma^+$ への正規編集転写とは，p と t の間の編集距離に対応する編集転写のうち，$I < R < D < M$ の順による辞書式順序で最大のものである．

近似文字列照合問題に対するアライメントが自明でないもう一つの理由は，近似文字列照合問題の解となる近似出現が複数あり得ることである．近似出現の定義に従うと，互いに重なり合わない出現の他に，厳密には以下のような出現の区別がある．t の部分文字列 t' が，閾値 δ についての p の近似出現であるとき，t' を部分文字列として真に含み，$|t''| - |t'| \leq \delta - d(p, t')$ である t の部分文字列 t'' は p の近似出現である．また，t' の部分文字列として真に含まれ，$|t'| - |t''| \leq \delta - d(p, t')$ である t の部分文字列 t'' も p の近似出現である．このような出現の区別は，一見冗長に思えるが，単純な方法でこれらを同一視すると，閾値が大きい場合，本来区別すべき出現についても同じとみなしてしまう恐れがある．例えば，$t = abababa, p = aba, \delta = 2$ のとき，p の近似出現として $t_{1,3}$ と $t_{5,7}$ を同一視してよいかどうかは，その応用に依存するべきである．

本論文では，近似出現の長さに着目し，各位置ごとの近似出現の代表の概念を導入する．

定義 2 ある位置についての代表近似出現とは，その位置から始まる近似出現のうち，対応する編集距離が最も小さく，長さが最も短いものである．

この近似出現の代表は，編集転写の正規形の導入に用いた順序によって決定することができる．これは，一致を表す M を最も優先して選んだ編集転写が編集距離に対応することと，ある文字列からの編集距離が等しい文字列は，正規編集転写中の D の数が多く，I の数が少ないほど短くなることからわかる．

補題 1 ある位置についての代表近似出現に対する正規編集転写は，その位置から始まり，対応する編集距離が最小となる近似出現に対する正規編集転写のうち，$I < R < D < M$ の順による辞書式順序で最大のものである．

本論文では，近似文字列照合問題に対するアライメントを次のように定義する．

定義 3 近似文字列照合問題に対するアライメントアルゴリズムとは，文字列 $p, t \in \Sigma^+$ と閾値 δ が与えられて，

		e	m	p	t	y
	0	$\to 1$	$\to 2$	$\to 3$	$\to 4$	$\to 5$
e	$\downarrow$ 1	$\searrow\downarrow$ $\to 0$	$\searrow\downarrow$ $\to 1$	$\searrow\downarrow$ $\to 2$	$\searrow\downarrow$ $\to 3$	$\searrow\downarrow$ $\to 4$
n	$\downarrow$ 2	$\searrow\downarrow$ $\to 1$	$\searrow\downarrow$ $\to 1$	$\searrow\downarrow$ $\to 2$	$\searrow\downarrow$ $\to 3$	$\searrow\downarrow$ $\to 4$
t	$\downarrow$ 3	$\searrow\downarrow$ $\to 2$	$\searrow\downarrow$ $\to 2$	$\searrow\downarrow$ $\to 2$	$\searrow\downarrow$ $\to 2$	$\searrow\downarrow$ $\to 3$
r	$\downarrow$ 4	$\searrow\downarrow$ $\to 3$	$\searrow\downarrow$ $\to 3$	$\searrow\downarrow$ $\to 3$	$\searrow\downarrow$ $\to 3$	$\searrow\downarrow$ $\to 3$
y	$\downarrow$ 5	$\searrow\downarrow$ $\to 4$	$\searrow\downarrow$ $\to 4$	$\searrow\downarrow$ $\to 4$	$\searrow\downarrow$ $\to 4$	$\searrow\downarrow$ $\to 3$

図 1: 動的計画法で用いる表の例

t 中の p の全ての代表近似出現の正規編集転写を返すものである．

4 アルゴリズム

4.1 動的計画法に基づく単純なアルゴリズム

長さ m と n の二つの文字列についての編集距離は，動的計画法により $O(mn)$ 時間で求めることができる．このアルゴリズムでは，概念的に，1 のような表が用いられる．この表のうち，要素の部分は **DP 行列**と呼ばれ，p, t を二つの文字列とするとき，以下の (i, j) 成分を持つ $(m+1) \times (n+1)$ 行列として定義される．

$$C[i,j] = \min \begin{cases} C[i-1, j-1] + W(p_i, t_j), \\ C[i-1, j] + 1, \\ C[i, j-1] + 1 \end{cases}$$

ただし，$i = 0$ のとき $C[0, j] = j$，$j = 0$ のとき $C[i, 0] = i$ とする．また，$a, b \in \Sigma$ について $W(a, b)$ は $a = b$ のとき 0，$a \neq b$ のとき 1 をとるものとする．1 の例においては，$t = \text{empty}, p = \text{entry}$ である．このとき，1 の表における矢印は，縦，横，斜めのものが，それぞれ，p からみた削除，挿入，置換または一致に対応している．つまり，τ を文字列 $p_{1,i}$ から $t_{1,j}$ への編集距離に対応する編集転写とすると，

- ρ_1 を $p_{1,i-1}$ から $t_{1,j-1}$ への編集距離に対応する編集転写として，$\tau = \rho_1 M$ または $\tau = \rho_1 R$，
- ρ_2 を $p_{1,i-1}$ から $t_{1,j}$ への編集距離に対応する編集転写として，$\tau = \rho_2 D$，または，

- ρ_3 を $p_{1,i}$ から $t_{1,j-1}$ への編集距離に対応する編集転写として，$\tau = \rho_3 I$

のいずれかが成り立つので，$1 \leq i \leq m$ と $1 \leq j \leq n$ について，$C[i,j]$ は $p_{1,i}$ と $t_{1,j}$ の間の編集距離であることがわかる．

このアルゴリズムは，DP 行列の初期条件を，$i = 0$ のとき $C[0,j] = 0$，$j = 0$ のとき $C[i,0] = i$ とすることで，近似文字列照合問題へ拡張することができる．つまり，$C[i,j]$ は $p_{1,i}$ と，位置 j で終わる t の部分文字列との間の編集距離のうち最も小さいものであるから，次が明らかである．

補題 2 $C[i,j]$ は，t^{-1} 中の位置 $n-j+1$ での ${p^{-1}}_{1,i}$ の代表近似出現に対応する編集距離である．

アライメントを求める場合，DP 行列を計算した後，各成分が最大値としてどの成分からの値をとったかを辿ることで，対応する編集転写を求めることができる．これは，1 の矢印が，文字の編集操作に対応していることから容易にわかる．一般に，DP 行列からアライメントを求める作業は**トレースバック**と呼ばれる．この手法による，近似文字列照合に対するアライメントアルゴリズムを 2 に示す．このアルゴリズムにおいて，DP 行列を求めるには，$(m+1) \times (n+1)$ 個の成分について最大値を求める計算を行うので，$O(mn)$ 時間が必要である．また，トレースバックに要する時間は単純な方法でも $O(m+n)$ 時間であるから，$O(mn)$ 時間で解くことができる．

4.2 ビットパラレル手法によるアルゴリズム

Myers [7] は，近似文字列問題を解くために，ビットパラレル手法により DP 行列を計算するアルゴリズムを提案した．このアルゴリズムの計算時間は，w をコンピュータのワード長として，$O(\lceil m/w \rceil n)$ 時間で抑えられている．ここで用いられているビットパラレル手法は，文字列間の類似度を二値の論理式の演算によって表すことで，DP 行列の成分のうち最大 w 個の値を並行して計算するものである．このアルゴリズムの概要を 3 に示す．ただし，ここで用いられている記号 $|$, $\&$, $\hat{}$, $\sim$, 及び $\ll$ は，それぞれ，ビット列に対する OR，AND，XOR，NOT，及びシフトの演算を表している．

以下，アルゴリズムの詳細を述べる．

$0 \leq i \leq m$ と $1 \leq j \leq n$ に対して，

$$\Delta h[i,j] = C[i,j] - C[i,j-1]$$

Procedure Naive Alignment Algorithm

inputs: $p = p_1 \cdots p_m \in \Sigma^+$,
$t = t_1 \cdots t_n \in \Sigma^+$, δ ;
outputs: $\tau_1, \ldots, \tau_k \in \{I, D, R, M\}^+$;

for $i = 0, \ldots, m$ **do** $C[i,0] = i$;
for $j = 0, \ldots, n$ **do** $C[0,j] = 0$;
for $i = 1, \ldots, m$ **do** {
  for $j = 1, \ldots, n$ **do** {
    $C[i,j] =$
    $\min \begin{cases} C[i-1,j-1] + W({p^{-1}}_i, {t^{-1}}_j), \\ C[i-1,j] + 1, \\ C[i,j-1] + 1 \end{cases}$
  }
}

$i = m$; $k = 1$;
for j s.t. $C[i,j] \leq \delta$ **do** {
  $\tau = \varepsilon$;
  while $(i, j \geq 1)$ {
    if $({p^{-1}}_i = {t^{-1}}_j)$ {
      $i = i-1$; $j = j-1$; $\tau = \tau M$;
    } **else if** $(C[i,j] = C[i-1,j] + 1)$ {
      $i = i-1$; $\tau = \tau D$;
    } **else if** $(C[i,j] = C[i-1,j-1] + 1))$ {
      $i = i-1$; $j = j-1$; $\tau = \tau R$;
    } **else**{
      $j = j-1$; $\tau = \tau I$;
    }
  }
  $\tau_k = \tau^{-1}$; $k = k+1$;
}

図 2: 動的計画法に基づく単純なアライメントアルゴリズム

と表す．また，$1 \leq i \leq m$ と $0 \leq j \leq n$ に対して，

$$\Delta v[i,j] = C[i,j] - C[i-1,j]$$

と表す．ここで，次の命題が成り立つ事が知られている [12]．

命題 1 任意の i と j について，$\Delta v[i,j], \Delta h[i,j] \in \{-1, 0, 1\}$ である．

任意の j に対して $C[0,j] = 0$ であるから，$C[m,j]$ は第 m 行目までの隣り合う成分の差の列 $\Delta v_j = (\Delta v[0,j], \ldots, \Delta v[m,j])$ から直ちに求めることができる．この Δv_j を定数時間で計算するために，i 番目に以下の要素を持つビット列 Eq_j, Pv_j, Mv_j を用意する．

$$\begin{aligned} Eq_j[i] &= \iota(p_i = t_j) \\ Pv_j[i] &= \iota(\Delta v[i,j] = 1) \\ Mv_j[i] &= \iota(\Delta v[i,j] = -1) \end{aligned}$$

ただし，ι は引数の式が真のときに 1，偽のときに 0 を返す関数である．$Eq_j[i]$ は文字 p_i と t_j が一致するかど

Procedure Bit-Parallel Algorithm

inputs: $p = p_1 \cdots p_m \in \Sigma^+$,
$t = t_1 \cdots t_n \in \Sigma^+$, Σ ;
outputs: $C[m,0], \ldots, C[m,n]$;

for $i = 1, \ldots, m$ **do** {
  for $s \in \Sigma$ **do** {
    if $(p_i = s)$ {
      $Peq(s)[i] = 1$ **else** $Peq(s)[i] = 0$;
    }
  }
}

$P_v = 1^m$; $M_v = 0^m$; $C[m,0] = m$;
for $j = 1, \ldots, n$ **do** {
  $Eq = Peq(t_j)$; $Xv = Eq \mid Mv$;
  $Xh = (((Eq \;\&\; Pv) + Pv) \hat{}\, Pv) \mid Eq$;
  $Ph = Mv \mid \sim (Xh \mid Pv)$;
  $Mh = Pv \;\&\; Xh$;

  if $(Ph \;\&\; 10^{m-1})$ $C[m,j] = C[m-1,j] + 1$;
  else if $(Mh \;\&\; 10^{m-1})$ {
    $C[m,j] = C[m-1,j] - 1$;
  }

  $Ph \ll 1$; $Mh \ll 1$;
  $Pv = Mh \mid \sim (Xv \mid Ph)$;
  $Mv = Ph \;\&\; Xv$;
}

図 3: ビットパラレル手法により DP 行列を求めるアルゴリズム

うかを表している．$Pv_j[i]$ と $Mv_j[i]$ は，第 i 行目について，第 j 列目の要素から第 $j-1$ 列目の要素を引いたものが 1 か -1 かを表している．この値が 0 のとき，$Pv_j[i] = 0$ かつ $Mv_j[i] = 0$ である．命題 1 から，Pv_j と Mv_j によって DP 行列を表すことができることがわかる．

DP 行列の隣り合う 4 つの成分 $C[i-1,j-1], C[i,j-1], C[i-1,j], C[i,j]$ を一つのブロックとして考える．$\Delta v_{out} = \Delta v[i,j], \Delta v_{in} = \Delta v[i,j-1]$ とし，また，DP 行列の行についても列と同様の概念を用いて，$\Delta h_{out} = \Delta h[i,j], \Delta h_{in} = \Delta h[i-1,j]$ とする．このとき，

$$\Delta v_{out} = \min\{-Eq_j[i], \Delta v_{in}, \Delta h_{in}\} + (1 - \Delta h_{in})$$

および

$$\Delta h_{out} = \min\{-Eq_j[i], \Delta v_{in}, \Delta h_{in}\} + (1 - \Delta v_{in})$$

である．ここで，

$$\begin{aligned} Xv &= Eq_j[i] \vee \iota(\Delta v_{in} = -1), \\ Xh &= Eq_j[i] \vee \iota(\Delta h_{in} = -1) \end{aligned}$$

とすると，Pv 及び Mv と，同様に定義される Ph 及び Mh について次が成り立つ．

$$\begin{aligned} Pv_{out} &= Mh_{in} \vee \neg(Xv \vee Ph_{in}) \\ Mv_{out} &= Ph_{in} \wedge Xv \\ Ph_{out} &= Mv_{in} \vee \neg(Xh \vee Pv_{in}) \\ Mh_{out} &= Pv_{in} \wedge Xh \end{aligned}$$

ただし，$A \vee B$ は $A = 1$ または $B = 1$ のとき 1，その他のとき 0 を，$A \wedge B$ は $A = 1$ かつ $B = 1$ のとき 1，その他のとき 0 を，$\neg A$ は $A = 0$ のとき 1，その他のとき 0 をとるものとする．

また，p_i がどの文字であるかを参照するための表

$$Peq(s)[i] = \iota(s = p_i)$$

を作成しておく．ただし，$1 \le i \le m$ かつ $s \in \Sigma$ である．

本アルゴリズムは大きく二つの段階から構成される．初期条件により，$1 \le i \le m$ について $Pv_0[i] = 1$ かつ，$1 \le j \le n$ について $Ph_j[0] = Mh_j[0] = 0$ である．

(1) $(Ph_j[i], Mh_j[i])$ を，$(Pv_{j-1}[i], Mv_{j-1}[i])$ と

$$Xh_j[i] = \begin{cases} 1 & (Peq(t_j)[i] = 1) \\ 0 & (Mh_j[i-1] = 1) \end{cases}$$

から，$Ph_j[i] = Mv_{j-1}[i] \vee \neg(Xh_j[i] \vee Pv_{j-1}[i])$ 及び $Mh_j[i] = Pv_{j-1}[i] \wedge Xh_j[i]$ によって求める．これを用いて $C[m,j]$ の値を

$$C[m,j] = C[m,j-1] + \begin{cases} 1 & (Ph_j[m] = 1) \\ -1 & (Mh_j[m] = 1) \end{cases}$$

によって更新する．

(2) $(Pv_j[i], Mv_j[i])$ を，$(Ph_j[i], Mh_j[i])$ と

$$Xv_j[i] = \begin{cases} 1 & (Peq(t_j)[i] = 1) \\ 0 & (Mv_{j-1}[i] = 1) \end{cases}$$

から，$Pv_j[i] = Mh_j[i-1] \vee \neg(Xv_j[i] \vee Ph_j[i-1])$ 及び $Mv_j[i] = Ph_j[i-1] \wedge Xv_j[i]$ によって求める．

ここで，Xv_j は $Peq(t_j)$ と一つ前の列から得られる Mv_{j-1} から求めることができるが，Xh_j を求めるには Mh_j が必要であり，また，Mh_j を求めるには Xh_j が必要となる．そこで，Xh_j を，Pv_{j-1} と $Peq(t_j)$ から

$$\begin{aligned} &Xh_j \\ &\quad = (((Peq(t_j) \wedge Pv_{j-1}) + Pv_{j-1}) \text{ xor } Pv_{j-1}) \\ &\qquad \vee Peq(t_j) \end{aligned}$$

によって求める．ただし，A xor B は $A \neq B$ のとき 1，その他のとき 0 をとるものとする．

ある要素についての OR，AND，XOR，及び，NOT の演算は，ビット列についての演算として並行して行うことができる．DP 行列の計算の帰納的な部分は，行については段階 2 においてシフト演算によって行われ，列については段階 1 において Xh_j を求める際の $+$ の演算によって行わる．長さがワード長 w を超えないビット列についての演算は定数時間で行うことができると仮定すると，各列についての二つの段階からなる計算は $O(\lceil m/w \rceil)$ 時間で行われるので，$1 \leq j \leq n$ について $C[m,j]$ を求めるには $O(\lceil m/w \rceil n)$ 時間が必要である．また，前処理として行列 Eq の値を求めなければならないが，$Peq(s)$ を導入することで $O(|\Sigma|m)$ 時間で計算できる．

4.3 アライメントアルゴリズムへの拡張

前節での議論からもわかるとおり，アライメントとしての編集転写は DP 行列から自明に求められるわけではない．ビットパラレル手法では，DP 行列の要素のうち明示的に値が求められるのは最後の行のみであり，4.1 小節で用いた単純な方法では，アライメントを求めるために必要な情報を得ることができない．トレースバックのための計算を新たに加えると，ビットパラレル手法による高速化の効果がなくなってしまう場合がある．ここでは，前小節の Myers のアルゴリズムをアライメントアルゴリズムへ拡張する．

我々は，編集転写の候補が，前小節のアルゴリズムにおいて用いられた行列 Eq, Pv, Mv によって制限されることに注目した．

補題 3 $\tau, \rho_1, \rho_2, \rho_3$ を，それぞれ，文字列 $p_{1,i}$ から $t_{1,j}$，$p_{1,i-1}$ から $t_{1,j-1}$，$p_{1,i-1}$ から $t_{1,j}$，$p_{1,i}$ から $t_{1,j-1}$ への編集距離に対応する編集転写とする．このとき，以下が成り立つ．
(1) $Eq[i,j] = 1$ ならば，$\tau = \rho_1 M$ である．
(2) $Eq[i,j] = 0$ かつ $Pv_j[i] = 1$ ならば，$\tau = \rho_2 D$ かつ $\tau \neq \rho_1 M$ である．
(3) $Eq[i,j] = 0$ かつ $Pv_j[i] \neq 1$ かつ $Mv_{j-1}[i] \neq 1$ ならば，$\tau = \rho_1 R$ かつ $\tau \neq \rho_1 M$ かつ $\tau \neq \rho_2 D$ である．
(4) $Eq[i,j] = 0$ かつ $Pv_j[i] \neq 1$ かつ $Mv_{j-1}[i] = 1$ ならば，$\tau = \rho_3 I$ かつ $\tau \neq \rho_1 M$ かつ $\tau \neq \rho_2 D$ かつ $\tau \neq \rho_1 R$ である．

証明 $Eq[i,j] = 1$ と $\tau = \rho_1 M$ が同値であることから，(1) は明らかである．上に加えて，$Pv_j[i] = 1$ と $\tau = \rho_2 D$ が同値であることから (2) を得る．これらの場合以外は，τ は $\rho_1 R$ か $\rho_3 I$ のいずれかであるが，$Mv_{j-1}[i] \neq 1$ のとき，$C[i-1,j-1] \leq C[i,j-1]$ であるから，$\tau = \rho_1 R$ である．また，$Mv_{j-1}[i] = 1$ のとき，$C[i-1,j-1] > C[i,j-1]$ であるから，$\tau \neq \rho_1 R$ かつ $\tau = \rho_3 I$ である． □

本論文で提案するアルゴリズムは以下の通りである．まず，前小節のアルゴリズムによって DP 行列の最後の行 $C[m,j]$ $(0 \leq j \leq n)$ を求める．この際，途中の計算で用いられる Pv と Mv を保持しておく．そして，$i = m$ のとき $C[i,j] \leq \delta$ となる全ての j について，以下の手順を再帰的に繰り返してトレースバックを行う．

1 $Eq[i,j] = 1$ のとき，$i = i-1, j = j-1$ とし，$\tau = \tau M$ とする

2 それ以外のとき，

2-1 $Pv_j[i] = 1$ のとき，$i = i-1$ とし，$\tau = \tau D$ とする

2-2 それ以外のとき，

2-2-1 $Mv_{j-1}[i] \neq 1$ のとき，$i = i-1, j = j-1$ とし，$\tau = \tau R$ とする

2-2-2 それ以外のとき，$j = j-1$ とし，$\tau = \tau I$ とする

これによりアライメントへ拡張されたアルゴリズムの概要を 4 に示す．

定理 1 *Bit-Parallel Alignment Algorithm* は近似文字列照合問題に対するアライメントアルゴリズムである．

証明 補題 3 から，トレースバックのための手順によって選ばれる編集転写が正規形であることがわかる．また，補題 2 および補題 1 から，このアルゴリズムによって得られる出力は，代表近似出現に対する編集転写であることがわかる． □

5 実験

本論文で提案したビットパラレル手法によるアライメントアルゴリズム Bit-Parallel Alignment Algorithm と，動的計画法に基づく単純なアルゴリズム Naive Alignment Algorithm を C 言語で実装し，動作時間の比較実験を行った．使用した計算機は ThinkPad X31（インテル $^{(R)}$Pentium$^{(R)}$ プロセッサ 1.7GHz，2GB メモリ容量，ワード長 32）で，OS は Unix 操作システムである．入力は，アルファベットを $\Sigma = \{a, b, \ldots, z\}$ と

Procedure Bit-Parallel Alignment Algorithm

inputs: $p = p_1 \cdots p_m \in \Sigma^+$,
$t = t_1 \cdots t_n \in \Sigma^+$, δ, Σ ;
outputs: $\tau_1, \ldots, \tau_k \in \{I, D, R, M\}^+$;

Compute $C[m, 0], \ldots, C[m, n]$ by Bit-Parallel Algorithm ;

$i = m$; $k = 1$;
for j s.t. $C[i, j] \leq \delta$ **do** {
  $\tau = \varepsilon$;
  while $(i, j \geq 1)$ {
    if $(Eq[i, j] = 1)$ {
      $i = i - 1$; $j = j - 1$; $\tau = \tau M$
    } **else if** $(Pv_j[i] = 1)$ {
      $i = i - 1$; $\tau = \tau D$;
    } **else if** $(Mv_{j-1}[i] \neq 1)$ {
      $i = i - 1$; $j = j - 1$; $\tau = \tau R$;
    } **else** {
      $j = j - 1$; $\tau = \tau I$;
    }
  }
  $\tau_k = \tau^{-1}$; $k = k + 1$;
}

図 4: ビットパラレル手法によるアライメントアルゴリズム

して，Σ 上の長さ $n = 80000$ の文字列 t と，長さ $m = 5$ から 63 のランダムに生成した文字列とした．出力は，p との編集距離がある閾値以下の近似出現全てについてではなく，編集距離が最も小さいものひとつについて編集転写を返すものとした．閾値を用いる場合は，トレースバックについての同様の計算を繰り返せばよい．

実験結果を 1 に示す．ただし，各値はそれぞれ 10 回の試行の平均である．この結果から，Naive Alignment Algorithm では m が長くなるにつれて計算時間も長くなるが，Bit-Parallel Alignment Algorithm では，ワード長 $w = 32$ の前後で一定の値になっており，かつ，全体としてより高速であることがわかる．

6 結論

近似文字列照合問題に対するアライメントをビットパラレル手法によって求める高速なアルゴリズムを提案した．近似出現とアライメントについて正規形の概念を導入し，提案するアルゴリズムによってその正規形が求められることを証明した．また，プログラムとして実装を行い，単純な手法によるアルゴリズムとの計算時間の比較を行った．

実験から，提案するアルゴリズムは単純なアルゴリズムに比べて高速であることがわかったが，DP 行列の計算についての他の高速化（例えば，DP 行列の斜めの線に並んだ，互いに依存しない要素の計算の単純な並列化や，いくつかの文字列についての編集距離の表をあらかじめ持っておく手法 [6]）との比較が今後の課題である．

参考文献

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403–410, 1990.

[2] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Commun. ACM*, 35(10):74–82, 1992.

[3] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, New York, 1994.

[4] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, New York, 1997.

[5] H. Hyyrö. A note on bit-parallel alignment computation. In *Proc. Prague Stringology Conference '04 (PSC2004)*, 2004.

[6] W. J. Masek and M. S. Paterson. A faster algorithm for computing string edit distance. *J. Comput. Syst. Sci.*, 20(1):18–31, February 1980.

[7] G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, 46(3):395–415, May 1999.

[8] G. Navarro. A guided tuor to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

[9] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.

[10] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. In *Proc. Natl. Acad. Sci. USA*, volume 85, pages 2444–2448, April 1988.

表 1: 二つのアルゴリズムの計算時間の比較（単位：秒）

m	5	10	16	24	32	46	52	63
Naive Alignment Algorithm	23.0	47.0	72.0	109.0	142.0	205.0	234.0	283.0
Bit-Parallel Alignment Algorithm	6.2	6.3	6.3	6.7	6.8	8.2	8.2	8.3

[11] T. F. Smith and M. S. Waterman. Identification of common molecular subseqences. *J. Mol. Biol.*, 147:195–197, 1981.

[12] E. Ukkonen. Identification of common molecular subseqences. *J. Algorithms*, 6(1):132–137, 1985.

[13] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974.