Vol.2012-BIO-29 No.24
2012/6/29

Suffix arrayを用いた高速な配列相同性検索の改良とエピゲノム解析への対応

鈴木 脩司[1,a)]　石田 貴士[1]　秋山 泰[1,b)]

概要：我々は以前に suffix array を用いた高速な相同性検索システムを提案したが，近年の次世代シークエンサーの進歩が目覚ましく，得られる配列データは増加しており，さらなる高速化が必要とされている．このため，本研究では従来システムの改良を試み，長さ L_{hash} のすべての文字列の suffix array の検索結果を予め計算しておき，これを保存しておく．そして，検索の際は L_{hash} 文字目までの検索には保存しておいたものを読み出すことで高速化した．また，このシステムを用いてエピゲノム解析へも対応するために，バイサルファイト処理を行った DNA 断片配列のマッピングができるように改良を行った．

キーワード：相同性検索，エピゲノム, suffix array

Speed-up of homology search tool using suffix array and it's extension for epigenomic analysis

Suzuki Shuji[1,a)]　Ishida Takashi[1]　Akiyama Yutaka[1,b)]

Abstract: We developed the system for fast homology search using suffix array. However, next generation sequencers are improving gradually and become to produce larger data than previous sequencers. Thus, we have developed a new faster system. To accelerate search using suffix array, we store the results of searching patterns whose length is less than L_{hash} and use them as caches. In addition, we enhanced our system to map bisulfite reads for epigenomics.

Keywords: homology search, epigenome, suffix array

1. はじめに

近年，次世代 DNA シークエンサーの登場により大量の DNA データが得られるようになった．次世代 DNA シークエンサーは 1 ランで数千億塩基以上が解読可能であり，これは以前の DNA シークエンサーと比べると数万倍のスループットである．しかし，この次世代 DNA シークエンサーによって得られるデータは DNA の断片配列であるため，その情報を利用するためにはマッピングなどの解析処理が必要となる．マッピングとは DNA シークエンサーによって読み取った DNA 断片配列がリファレンスゲノムのどの部分のものであったかを同定する処理であり，マッピングを行うツールとしては BWA[1], [2] や Bowtie[3] などが開発れてきた．しかし，これらのマッピング手法は高速であるが，DNA 断片配列とリファレンスゲノムとの間で数塩基程度の違いしかないことを仮定しており，遠縁の相同配列間のように，多くのギャップや置換が含まれるような曖昧な一致がある場合の検索には不向きである．

一方，土壌や腸内等に生息する複数の微生物の DNA を分離，培養を経ずに直接 DNA を読み取り，環境中に生息する微生物の遺伝子の分布を解析するメタゲノム解析というものがある．このメタゲノム解析では，環境中に含まれる微生物のすべてのゲノム配列が既知であることは稀であるため，メタゲノムのマッピングを行うには同じ種のゲ

[1] 東京工業大学 大学院情報理工学研究科 計算工学専攻
Graduate School of Information Science and Engineering, Tokyo Institute of Technology
[a)] suzuki@bi.cs.titech.ac.jp
[b)] akiyama@cs.titech.ac.jp

ノム情報を参照する必要があり，リファレンスゲノムのある単一の生物に対する通常のマッピングを行うのに比べて，マッピングの際に多くの不一致やギャップを許容する必要がある．そのため，従来開発されてきた高速なマッピング手法では検索の感度が不十分であり，メタゲノムの解析に利用するのは困難である．そういった理由から，メタゲノム解析では一般に相同性検索と呼ばれる，より高感度な手法が利用されている．また，メタゲノム解析では検索の感度を増すために DNA 配列のまま行うのではなく，多くの場合，アミノ酸配列に翻訳してから解析が行われる．DNA 配列は通常 A，T，G，C の 4 文字で表現されているのに対してアミノ酸配列は標準的な 20 文字で表現されている．さらに，DNA 配列の比較は一致か不一致の 2 状態でしか区別しないことが多いが，アミノ酸配列はアミノ酸間で性質の類似度に違いがあるため，アミノ酸毎に置換スコアが付けられた置換行列が用いられる．このため，アミノ酸配列でマッピングを行うのは DNA 配列の場合のマッピングと比べて計算はより複雑なものとなっており，困難なものとなっている．このようなタンパク質配列間での曖昧な一致も含めた配列の相同性検索では，現在では比較的高速な配列比較が可能な近似的手法の BLAST[4], [5] が利用されている [6]．しかし，次世代 DNA シークエンサーは大量の DNA 断片配列を出力するため，すべての DNA 断片配列をアミノ酸配列に翻訳してからマッピングを行うのは BLAST を用いても長い計算時間が必要となる．現在，最新の illumina 社の Hiseq 2000 という DNA シークエンサーの出力は 1 ランで合計 600G 塩基にも達し，そのデータを解析するには約 25,000CPU 日が必要になると推定される．このため，我々は以前，GPU を用いた手法 [7] や suffix array[8] という文字列検索で良く用いられているデータ構造をデータベース，クエリ両方で構築し，一致率の高い位置を高速に発見する手法を用いて，高速な配列相同性検索ツールを開発した [9].

しかし，近年の DNA シークエンサーの進歩は目覚ましく，さらに大量の配列データが得られるようになっている．このため，今後さらに高速な手法が必要となると考えられる．したがって，我々は従来手法で一番時間のかかっていた suffix array を用いた検索において検索結果の一部をあらかじめ計算しておき，メモリに保存しておくことで高速化を行った．

また，次世代 DNA シークエンサーの発展によりメタゲノム解析の他にも様々なゲノム解析が可能になりつつある．その中でも遺伝子発現のための情報とその制御機構を扱うエピゲノム解析というものがある．その中でもバイサルファイト反応を利用して DNA のどの位置がメチル化されているかという情報を次世代 DNA シークエンサーを利用することで大量に得ることができるようになっており，大きな注目を集めている．

バイサルファイト処理では，非メチル化シトシンが亜硫酸ナトリウムと反応して，ウラシルへと変換されるが，メチル化シトシンは反応しないので，原理上，全てのシトシンのメチル化状態を塩基の違いとして検出できる．これを利用することでバイサルファイト処理を施してから DNA を読み取り，出力された DNA 断片配列をマッピングして，DNA 断片配列側とリファレンスゲノム側の C と T を比較してどの位置がメチル化されているかを調べることができる．

しかし，その結果として通常のマッピングではアラインメントを計算する際には文字が一致しているという状態は同じ文字でしかありえなかったが，バイサルファイト処理した DNA 断片配列では DNA 断片配列の T はリファレンスゲノム側の C と T の二文字と一致するという状態になる．このため，通常のマッピングツールではマッピングを行うことが難しい．

現在までにバイサルファイト処理を行った DNA 断片配列をマッピングするツールとしては BS Seeker[10] や Bismark[11] などが開発されてきた．しかし，現在の DNA シークエンサーの出力する大量の DNA 断片配列に対する解析速度が不十分であり，解析には多くの時間がかかっている．

このため，我々は配列相同性検索で用いていた手法を改良して，バイサルファイト処理を行った DNA 断片配列を高速にマッピングする手法を提案し，この実装を行った．

2. Suffix array を用いた高速な配列相同性検索の改良

2.1 Suffix array

本研究で使用した suffix array の詳細について説明する．Suffix array は全文検索などに利用される検索アルゴリズムの一つであり，ある文字列 T が与えられたとき T のすべての接尾辞に添字を付け辞書順に並べ替えることによって得ることができる．**図 1** に例を示す．ここで，Σ を文字の集合とし T は Σ の要素による配列であるとする．T の末尾には配列の終端を表す $\$ \notin \Sigma$ かつ $\$ < \forall c \in \Sigma$ となる \$を付けている．この suffix array の構築には $O(|T|)$ で実行できるアルゴリズムが提案されている．また，文字列 P の完全一致の検索は最悪 $O(|P|log|T|)$ で実行できることが知られている．

2.2 配列相同性検索の概要

本研究で提案する手法の全体の手順としては我々が以前提案した手法 [9] と同一である．基本的には BLAST と同様にクエリとデータベース中の配列間で局所的にスコアの高くなる位置を探索（seed 探索）をした後，探索によって見つかった位置を中心にアラインメントの伸張（extension）を行う，seed-extension の手法を用いている．

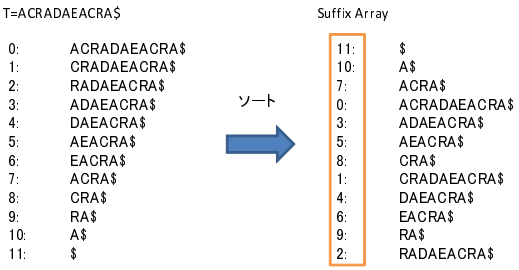

図 1 Suffix Array
Fig. 1 An example of suffix array

図 2 配列の連結
Fig. 2 Connecting sequences

手順としては予め対象となるデータベースの suffix array を構築しておく．DNA シークエンサーによって読み取られた DNA 断片配列はアミノ酸配列に翻訳し，データベースと同様に suffix array を構築する．データベースと翻訳してできたアミノ酸配列の 2 つの suffix array を用いて，局所的にスコアの高くなる位置 (seed) を探索する．そして，探索して見つかった seed を中心にアライメントの伸張を行い，スコアを計算し，相同性検索を行う．

詳細な手法を以降に述べる．

2.2.1 データベースの suffix array の構築

相同性検索先のアミノ酸配列データベースには複数のタンパク質の配列が含まれている．このアミノ酸配列毎に個々の suffix array を構築するのではなくアミノ酸配列を連結させ，単一の配列にした後，連結配列に対して suffix array を構築する．このように連結配列にすることで，データベース内の複数の配列に対してまとめて seed 探索を行うことができる．連結をする際には**図 2** のようにアミノ酸配列中には存在しない文字 # を区切り文字として，配列と配列の間にこの区切り文字を挿入して連結していく．

2.2.2 DNA 断片配列の翻訳と suffix array の構築

DNA シークエンサーによって出力された DNA 断片配列は 6 フレームで翻訳し，翻訳して出来た 6 つのアミノ酸配列をデータベース同様に連結する．この連結した配列を今後，一つのクエリとする．データベースの時と同様に，配列を連結することにより，翻訳して出来た 6 つのアミノ酸配列をまとめて seed 探索ができる．

2.2.3 クエリとデータベースの suffix array を用いたアラインメントの seed 探索

クエリとデータベースの suffix array を用いてスコアが閾値 T_s 以上になるクエリとデータベースの部分文字列のペアを探索し，部分文字列の位置を seed とする．**図 3** にクエリとデータベースの suffix array を用いたアラインメントの seed 探索をする擬似コードを示す．ここで L_{max} は探索する文字列の最大長，$SASearch(SA, P)$ はある文字列 T の suffix array(SA_T) を用いてソートされた T の接尾辞中で接尾辞の先頭に文字列 P が出現する suffix array 上での範囲 sp, ep を返す関数，$maxScore$ は w_q を元にして w_d が w_q と完全に一致した場合に得られるスコア，D は $maxScore$ と $score$ の差をどれだけ許すかを示す値，$S[c, c']$ は文字 c, c' に対する置換行列の値である．D が 1 のとき，完全一致のクエリとデータベースの部分文字列のペアのみを検索するようになる．

```
SeedSearch(w_q, w_d, maxScore, sscore)

 1: if |w_q| < L_max then
 2:   for all c ∈ Σ do
 3:     w'_q ⇐ w_q + c
 4:     sp, ep ⇐ SASearch(SA_q, w'_q)
 5:     if sp ≤ ep then
 6:       maxScore' ⇐ maxScore + S[c, c]
 7:       for all c' ∈ Σ do
 8:         w'_d ⇐ w_d + c'
 9:         sp', ep' ⇐ SASearch(SA_d, w'_d)
10:         if sp' ≤ ep' then
11:           score' ⇐ score + S[c, c']
12:           if score' ≤ T_s then
13:             sp, ep, sp'ep' を保存する
14:             return
15:           else if score' > maxScore' − D and
              score' > 0 then
16:             Searh(w'_q, w'_d, maxScore', score')
17:           end if
18:         end if
19:       end for
20:     end if
21:   end for
22: end if
```

図 3 クエリとデータベースの suffix array を用いたアラインメントの seed 探索
Fig. 3 Search for candidate positions of optimum alignment using suffix arrays for queries and database

BLAST では短い固定長の部分文字列とその部分文字列から数文字置換した近傍の部分文字列を用いて探索を行うが，固定長では感度を良くするために近傍の部分文字列を列挙する際の閾値を低くしなければならない．そうすると，探索する近傍の文字列の量が増えてしまい探索時間が増加する．しかし，本研究で用いた探索では suffix array を用いることで任意の長さの検索を行える特徴を生かし，閾値 T_s 以上の任意の長さの部分文字列を検索する．こうするこ

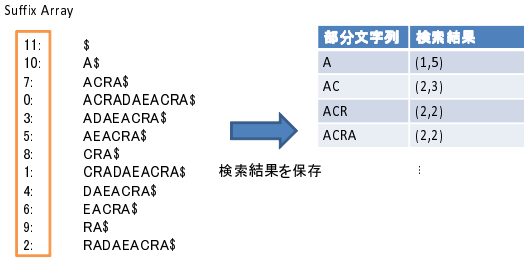

図 4 Suffix array の結果の一部を保存
Fig. 4 Storing some search results

とで，一致率の高い文字列では短い文字列が一致した時点でヒットとし，また一致率の低いものは長い文字列が一致した時点でヒットとなることでヒットする量を減らしつつ探索の感度を保つことが可能となっている．また，探索時間を短縮するためにクエリとデータベースの両方で suffix array を用いることで共通の部分文字列に関してはまとめて探索を行う．

2.3 Suffix array を用いた検索の高速化

従来手法では suffix array を用いた検索の部分で多くの時間がかかっていた．後述（2.4 節）するデータを使用した場合，約 60%が suffix array の探索であった．よって，更なる高速化のためには suffix array の検索を高速化する必要がある．

このため，本研究では検索結果の一部を保存すておくことで，この suffix array の検索の高速化を行った．データベース側の suffix array のみ，長さ L_{hash} の文字列をすべて検索し，検索結果を文字列から生成したハッシュをキーとするハッシュテーブルに結果を保存しておく．そして，suffix array を用いた検索をする際，予め計算しておいた文字列の場合はハッシュテーブルから結果を引いてくるようにした．

検索結果を保存しておく文字列の最大の長さを L_{hash} とする．連結配列の suffix array を構築後，L_{hash} 文字以内のすべての文字列に対するデータベースの検索結果を計算し，**図 4** のようにハッシュテーブルに保存する．

ハッシュテーブルを用いるため，L_{hash} 以内の文字列の検索に関しては 1 つのハッシュ値を計算する時間計算量が $O(L_{hash})$ であるとすると $O(L_{hash})$ で検索を行うことができる．

また，このハッシュテーブルを保存するために必要なメモリに関しては $O(\Sigma^{L_{hash}})$ である．実際のメモリ使用量は一つの suffix array の結果を保存するために 8byte 必要だとすると $L_{hash} = 5$ の場合は約 25Mbyte 程度であり，少ないメモリで構築することができる．

2.4 改良した配列相同性検索手法の評価実験

本研究で提案した改良手法の相同性検索の感度と計算速度を評価する実験を行った．実験に使用した計算機は TSUBAME2[12] の Thin ノードを使用した．このノードの CPU は Intel Xeon processor X5670 (6 コア，2.93GHz) を 2 つ，メモリは 54GB である．OS は SUSE Linux Enterprise Server 11 SP1，gcc は version 4.3 を使用した．性能比較には NCBI BLAST(version 2.2.24) と BLAT[13](version 34 standalone)，そして従来手法 [9] を使用した．BLAST はスコア行列に BLOSUM62，open gap penalty と extend gap penalty はそれぞれ −11，−1，そしてフィルタを使用しないようにした．具体的に使用した BLAST のオプションは “-p blastx -m 8 -b 1 -v 1 -G 11 -E 1 -g T -F F -M BLOSUM62” である．BLAT は予め DNA 断片配列をアミノ酸配列に翻訳しておき，それをクエリとして使用した．使用した BLAT のオプションは “-q=prot -t=prot -out=blast8” である．改良手法，従来手法共に seed 探索のパラメータは検索する部分文字列の長さは $L_{max} = 8$ を使用した．また，$T_s = 18, 24, 30$，$D = 1, 4, 7$ の全組み合わせを試し，感度と速度の性能を考慮し，感度重視として $(T_s = 24, D = 4)$，速度重視として $(T_s = 30, D = 1)$ を使用した．その他のパラメータの値は BLAST の値と共通のものを用いた．感度重視のパラメータの場合、検索感度は BLAST とほぼ同等となるようになっている．また，$D = 1$ のときは seed 探索の際，探す部分文字列は完全一致のペアのみとなる．このため，速度重視の場合は seed 探索で完全一致のみ探索することになる．

また，改良手法は $L_{hash} = 5$ 文字以内のすべての文字列に対するデータベースの検索結果をハッシュテーブルに保存して使用した．以下に実験の手順と結果について述べる．

2.4.1 使用データ

アミノ酸配列データベースは 2010 年 11 月時点の KEGG Genes（genes.pep）[14], [15], [16] を使用した．このデータベースはタンパク質のアミノ酸配列を 420 万本を含み，全配列の合計長さは約 2G 残基である．検索のクエリデータとしては次世代 DNA シークエンサーの出力である実データを用いた．実データとして土壌のメタゲノムを illmina 社の Genome analyzer で読み取ったデータを使用した．このデータは 1 本当たり 60〜75 塩基であり，約 7 百万本からなっている．全データを利用した場合，計算に多くの時間が必要なため，本研究では全データの内，1 万本をランダムに選択して使用した．

2.4.2 計算時間の比較

相同性検索の計算時間を比較するために，計算時間と BLAST を 1 としたときの計算速度比を比較した．

結果を表 1 に示す．まず改良手法の感度重視のパラメータについて考える．改良手法の感度重視のパラメータでは従来手法と比べて約 2.2 倍の高速化を達成している．これは従来手法が suffix array の検索に多くの計算時間を要していたが，先に検索結果を一部保存しておくことで suffix

表 1 相同性検索の計算時間 (sec.) と BLAST との計算速度比
Table 1 Computation time (sec.) and accelaretion ratio with respect to the BLAST

ツール	計算時間	BLAST からの計算速度比
従来手法 感度重視 (Ts=24,D=4)	1,662	22.0
改良手法 感度重視 (Ts=24,D=4)	768	47.7
従来手法 速度重視 (Ts=30,D=1)	311	117.8
改良手法 速度重視 (Ts=30,D=1)	156	234.9
BLAST	36,641	1.0
BLAT	799	45.9

array で検索を行う回数を減らすことができたためであると考えられる．

また，改良手法は BLAT とほぼ同じ計算速度となっている．このことから，感度重視のパラメータの場合，BLAST と同等の感度を保ったまま，BLAT とほぼ同程度の高速化を達成した．

次に改良手法の速度重視のパラメータについて考える．改良手法の速度重視のパラメータでは従来手法と比べると約 2.0 倍の高速化を達成している．また速度重視のパラメータの場合は BLAT と比較すると約 5.1 倍の高速化を達成している．

3. Suffix array を用いた高速な配列相同性検索のバイサルファイト配列への対応

通常のマッピングでは，DNA 断片配列をリファレンスゲノムの順鎖と逆鎖にマッピングするか，DNA 断片配列の順鎖と逆鎖をリファレンスゲノムの順鎖にマッピングする．このため，通常は 1 つの DNA 断片配列で 2 回のマッピングの計算を行うことになる．一方，バイサルファイト処理をした DNA 断片配列の場合，DNA 断片配列の T をリファレンスの C と T で一致すると考える．このため，バイサルファイト処理をした DNA 断片配列をペアエンドで読んだデータを用いると，DNA 断片配列中の T を C と T で一致させるようにして，DNA 断片配列の順鎖と逆鎖それぞれをリファレンスゲノムの順鎖と逆鎖へマッピングする必要がある．結果として，合計で 4 回のマッピングを行う必要がある．

3.1 バイサルファイト配列マッピングへの対応

配列相同性検索とマッピングでは許容する不一致やギャップの数が大きく違う．さらに，DNA 配列とタンパク質配列とでは出現する文字の種類が違う．このため，配列相同性検索で用いていた手法をそのままマッピングに適応すると多くの計算時間がかかる．このため，本研究では高速なマッピングを行うために suffix array を用いた seed 探索と，seed を中心とした伸張を計算する回数を減らすことで高速化した．マッピングの処理の流れは以下のとおりである．まず、予めリファレンスゲノムの順鎖と逆鎖からバイサルファイト配列マッピング用の suffix array を構築しておく．次にマッピングを行う DNA 断片配列を読み込んだ後，DNA 断片配列の順鎖と逆鎖の両方の部分文字列をリファレンスゲノムの suffix array を用いて探索する．こうして見つけた箇所を seed として，アライメントの伸張を行ない，マッピングのスコアを計算する．以下に詳細を述べる．

3.1.1 バイサルファイト処理を行った DNA 断片配列の seed 探索を行うための suffix array の構築

バイサルファイト処理を行うと DNA 断片配列中の T を C と T で一致させる必要がある．しかし，suffix array を用いた seed 探索では DNA 断片配列中の T をリファレンスゲノムの C と T で一致させると，多くの文字列について調べなくてはならなくなり，多くの計算時間が必要となる．このため，リファレンスゲノムの suffix array を構築する際に用いる配列のみ，すべての C を T にして，suffix array を構築する．こうすることで検索する際に，検索する文字列を増加させずに seed 探索を行えるようにした．

3.1.2 seed 探索の高速化

マッピングの場合，DNA 断片配列とリファレンスゲノムの間ではほぼ文字が一致しており，数文字程度の不一致，もしくはギャップしかない．DNA 断片配列とスコアが最大となるリファレンスゲノムの領域の間で不一致とギャップの数の合計が E であったとすると，DNA 断片配列を $E+1$ 個に分割すると，DNA 断片配列の分割配列のどれか 1 つはスコアが最大となるリファレンスゲノムの領域の一部と完全一致することが知られている．

しかし，マッピングでは一致、不一致、ギャップそれぞれにスコアを割り当て，類似度でマッピングのスコアを計算している．また，不一致が入る可能性はギャップが入る可能性よりも高いため，通常は不一致のコストのほうが小さくなっている．このため，一致したときのスコアを m，不一致のときのスコアを p とすると，あるスコア S 以上となり，かつ，分割してできた部分配列がスコア S 以上となる領域の一部と完全一致する場合の分割数を E_S+1 とすると E_S は以下のようになる．

$$E_S \leq \frac{Nm - S}{m - p} \tag{1}$$

よって，suffix array を用いて seed 探索を行う際，最初は分割をせずに探索を開始し，徐々に分割数を大きくしていきながら，その時点でマッピングされたスコアを元にして最大分割数を計算する．こうして計算した最大分割数まで DNA 断片配列を分割して seed 探索を行う．これにより，配列相同性検索のときよりも大幅な枝刈りが可能となっている．

3.1.3 リピート配列への対応

リピート配列等が DNA 断片配列に含まれる場合，上記

の方法だと 1 つの分割配列と完全一致する領域がリファレンスゲノムに大量に存在する場合がある．このまますべての seed でアラインメントの伸張を行うと多くの計算時間が必要となる．このため，本研究では E-value[17] を用いて 1 つの分割配列の探索で見つけた seed 数が E-value よりも大きい場合，アラインメントの伸張を行わないようにした．ただし，seed の数が 10 以下の場合はすべてアラインメントの伸張を行うようにした．こうすることでリピート配列でも高速なマッピングを行えるようにした．

3.1.4 アラインメントの伸張の計算の高速化

マッピングの場合，ギャップはほとんど入らないと考えられる．このため，多くのツールではギャップの入る最大数を制限している．このため，本研究でも同じようにギャップの入る最大数を制限した．こうすることでアラインメントの伸張を計算する際，高速な計算が可能となっている．また，DNA 断片配列の文字がすべて一致した場合，最大スコアとなる．このため，スコア S 以上でマッピングされる位置を探している場合，DNA 断片配列の最大スコアを S_{max} とすると $S_{max} - S$ 以上，不一致，またはギャップのコストが入った場合，計算を途中でやめてしまって問題がない．このようにしてアラインメントの伸張の計算の高速化を行った．

3.1.5 マッピングの並列化

DNA 断片配列についてはそれぞれ独立にマッピングを行うことができる．このため，本研究ではまず CPU コアの数に DNA 断片配列を分割し，CPU コア毎に担当した DNA 断片配列のマッピングが終わり次第，次の DNA 断片配列の処理を開始するようにし，マッピングを並列に処理できるよう実装を行った．

3.2 実験

本研究で提案した手法の精度と計算時間を評価する実験を行った．使用した計算機環境は 2.4 節と同一のものである．

3.2.1 実験データ

評価実験のリファレンスゲノムには NCBI からダウンロードしたヒトのリファレンスゲノム (Build 37) の 21 番染色体と全染色体の 2 つのリファレンスゲノムを使用した．また，実験に用いた DNA 断片配列は Chen らの研究 [10] の人工データの作成時に用いられていたメチル化率を参考にして以下のように作成した．まず，リファレンスゲノムを元にして SAMtools の wgsim を用いて 70bp の DNA 断片配列を 2M 本を作成する．そして，この DNA 断片配列に含まれる CG という部分文字列の C を 28%の確率で T に変換する．こうして作成した DNA 断片配列を使って評価実験を行った．また，提案手法は DNA 断片配列の長さの 4%までエラーを許容するように設定した．このため，70bp では 3 文字まではエラーを許容することになる．

表 2 ヒトの 21 番染色体のマッピングの実験結果
Table 2 The results for human chromosome 21

ツール	エラー率 (%)	計算時間 (sec,)	計算速度比
提案手法 (1CPU)	3.2	83.3	2.3
提案手法 (8CPU)	3.2	38.1	5.0
BS Seeker	20.0	190.5	1.0

表 3 ヒトの全染色体のマッピングの実験結果
Table 3 The results for full human chromosome

ツール	エラー率 (%)	計算時間 (sec,)	計算速度比
提案手法 (1CPU)	10.3	576.0	0.4
提案手法 (8CPU)	10.3	257.7	1.0
BS Seeker	25.5	253.1	1.0

また，ギャップは最大で 1 つまで許容し，一致，不一致，ギャップのスコアはそれぞれ 1，−3，−7 とした．また，BS Seeker は内部で Bowtie を 8CPU コア使用して実行していたため，提案手法も 8CPU コアを使うようにした場合も測定した．

3.2.2 実験の結果

実験データを用いて提案手法のマッピング精度と計算時間を測定して評価を行った．精度を評価するために各ツールの結果を SAM フォーマットに変換し，SAMtools の中の wgsim_eval.pl を用いて評価し，正しくマッピングが行えなかった割合を出した．結果は表 2，3 の通りである．

まず精度について比較する．精度については 21 番染色体と全染色体どちらの場合も BS Seeker と比較して，提案手法のほうが精度が良い結果が得られた．これは BS Seeker がデフォルトだとミスマッチを 2 つまでしか許容しないためであると考えられる．

次に，計算時間に関して比較する．21 番染色体に関しては提案手法を 1CPU コアで実行した場合，BS Seeker の約 2.3 倍の高速化を達成した．また，提案手法は 8CPU コアで実行した場合，BS Seeker の約 5.0 倍の高速化を達成した．これは見つかっているマッピング位置のスコアに応じて最大分割数を変更して，seed 探索を打ち切っているためである．また，全染色体の場合は提案手法を 1CPU コアで実行した場合，BS Seeker の約 0.4 倍と BS Seeker よりも遅くなっている．これは 21 番染色体のときと比べ，リファレンスゲノムが大きいため，I/O に多くの時間がかかっているからである．しかし，8CPU を用いた提案手法では BS Seeker と同程度の計算時間でマッピングを行うことができる．

次に提案手法の並列化効率について考える．提案手法の 1CPU コアと 8CPU コアを比較すると 21 番染色体，全染色体どちらの場合も約 2.2 倍となっている．このため，並列化効率は 0.28 と低い値になっている．これは並列化し

てないリファレンスゲノムの読み込みと結果の出力の部分に多くの時間がかかっているためである．このため，並列化効率をあげるためにはI/Oの部分の処理を速くする必要がある．

4. 結論

4.1 本研究の成果

本研究では従来手法で時間のかかっていたデータベースのsuffix arrayの検索部分を，一部予め計算しておき，保存しておくことで，実際に配列相同性検索を行う際に高速な検索を行えるようにした．結果として改良手法は従来手法より2.2倍の高速化を達成した．さらに，改良手法は計算速度を重視した，速度重視のパラメータの時には従来手法の2.0倍の高速化を達成した．

また，バイサルファイト処理を行ったDNA断片配列をマッピングをできるように改良を行いエピゲノム解析への対応を行った．その結果，21番染色体に対するマッピングの場合，BS Seekerよりも高い精度を保ちつつ，8CPUコアのとき約5.0倍の高速化を達成した．また，全染色体に対するマッピングの場合，ほぼ同じ計算時間でBS Seekerよりも高い精度を達成した．

4.2 今後の課題

近年，DNAシークエンサーの出力するDNA断片配列は徐々に伸びている．DNAシークエンサーの出力するDNA断片配列が長くなるとタンパク質のコード領域がすべてDNA断片配列に収まるようになると考えられる．このため，DNA断片配列のすべてをアミノ酸に翻訳するのではなく，オープンリーディングフレームを考慮して翻訳したほうが効率がよくなると考えられる．また，長いDNA断片配列の場合，アラインメントの伸張の部分で多くの時間がかかるようになる．このため，伸張を効率よく行うアルゴリズムが必要であると考えられる．

また，本研究ではバイサルファイト処理を行ったDNA断片配列に対するマッピングでは人工データでしか実験を行っていない．このため，実際のデータを用いて評価を行う必要があると考えられる．

参考文献

[1] Li H, Durbin R: “Fast and accurate short read alignment with Burrows-Wheeler transform”, *Bioinformatics*, 25(14):1754-1760(2009).
[2] Li H, Durbin R: “Fast and accurate long-read alignment with Burrows-Wheeler transform”, *Bioinformatics*, 26(5):589-595(2010).
[3] Langmead B, Trapnell C, Pop M, Salzberg SL: “Ultrafast and memory-efficient alignment of short DNA sequences to the human genome”,*Genome biology*, 10(3):R25(2009).
[4] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: “Basic local alignment search tool”, *Journal of Molecular Biology*, 215: 403-410(1990).
[5] Altschul SF, Madden TL, Schaffer a a, et al: “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs”, *Nucleic acids research*, 25(17):3389-3402(1998).
[6] Dalevi D, Ivanova NN, Mavromatis K, et al: “Annotation of metagenome short reads using proxygenes”, *Bioinformatics*, 24(16):7-13(2008).
[7] Suzuki S, Ishida T, Kurokawa K, Akiyama Y: “GHOSTM: A GPU-Accelerated Homology Search Tool for Metagenomics”, *PLoS ONE*, 7(5):e36060(2012) .
[8] Manber U, Myers G: “Suffix arrays: A new method for on-line string searches”, *Society for Industrial and Applied Mathematics Philadelphia*, 22(5):935-948(1990).
[9] 鈴木 脩司 , 石田 貴士 , 秋山 泰: “FM-indexを用いた高速な配列相同性検索ツールの開発”, 情報処理学会, 研究報告, 2010-BIO-23(21), 1-6(2010).
[10] Chen P-Y, Cokus SJ, Pellegrini M: “BS Seeker: precise mapping for bisulfite sequencing”. *BMC bioinformatics*, 11, 203(2010).
[11] Krueger F, Andrews SR: “Bismark: a flexible aligner and methylation caller for Bisulfite-Seq applications”. *Bioinformatics*, 27(11), 1571-1572(2011).
[12] TSUBAME2 — [GSIC] 東京工業大学学術国際情報センター.
http://www.gsic.titech.ac.jp/tsubame2
[13] Kent WJ: “BLAT–the BLAST-like alignment tool”, *Genome research*,12(4):656-64(2002).
[14] Kanehisa M, Goto S, Furumichi M, Tanabe M, Hirakawa M: “KEGG for representation and analysis of molecular networks involving diseases and drugs”, *Nucleic Acids Res*, 38, 355-360(2010).
[15] Kanehisa M, Goto S, Hattori M, Aoki-Kinoshita KF, Itoh M, Kawashima S, Katayama T, Araki M, Hirakawa M: “From genomics to chemical genomics: new developments in KEGG”,*Nucleic Acids Res*, 34, 354-357(2006).
[16] Kanehisa M, Goto S: “KEGG: Kyoto Encyclopedia of Genes and Genomes”, *Nucleic Acids Res*, 28, 27-30(2000).
[17] Karlin S, Altschul SF: “Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes”, *Proc. Natl. Acad. Sci. USA*, 87, 2264-2268(1990).