Vol.2013-BIO-34 No.14
2013/6/28

データベースの部分文字列クラスタリングによるアミノ酸配列相同性検索の高速化

鈴木 脩司[1,a)]　石田 貴士[1,b)]　秋山 泰[1,c)]

概要：メタゲノム解析ではDNA配列をアミノ酸配列に変換して相同性検索を行うが、次世代シークエンサの登場によって得られるようになった大量のDNA断片配列の処理に多くの時間がかかるようになっている。このため、我々はあらかじめデータベースを部分文字列に分割して、類似度が高い部分文字列をまとめておき、まずその代表点に対して検索を行うことで効率よく検索する手法を開発した。

キーワード：配列相同性検索, クラスタリング，メタゲノム

Speed-up of homology search tool using clustering of subsequences in database

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

Abstract: In metagenome analysis, DNA sequences are transformed into amino acid sequences before homology search. However, next generation sequencers became to produce larger data than previous sequencers. And then it takes more time to analyze data from next generation sequencers. Thus, we have made clusters of subsequences from database before homology search and search queries from the representations in clusters to search efficiently.

Keywords: sequence homology search, sequence clustering, metagenome analysis

1. 序論

近年，次世代DNAシークエンサ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シークエンサによって得られるデータはreadと呼ばれる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っている．マッピングとはDNAシークエンサによって読み取ったreadがリファレンスゲノムのどの部分の配列なのかを同定する処理である．このマッピングを行うツールとしてはBWA[1], [2]やBowtie[3], [4]など様々なツールが開発されている．しかし，これらのマッピングの手法は高速であるが，readとリファレンスゲノムとの間で数%程度の違いしかないことを仮定している．このため，多くのギャップや置換が含まれるような遠縁の相同配列を見つけるような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．このため，メタゲノムのデータのマッピングを行うには遠縁の生物のゲ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マッピングよりも

[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)] t.ishida@bi.cs.titech.ac.jp
[c)] akiyama@cs.titech.ac.jp

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，現在のメタゲノム解析ではより高感度な配列相同性検索と呼ばれる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 配列のまま配列相同性検索を行うのではなく，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基本的に 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スコアマトリックスが用いられる．このため，アミノ酸配列でマッピングを行うことは 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，よ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の曖昧な一致も含めた配列の相同性検索では，比較的高速な配列比較が可能な近似的手法の BLAST[5], [6] が利用されている [7]．しかし，次世代 DNA シークエンサは大量の read を出力するため，read をアミノ酸配列に翻訳してからマッピングを行うことは BLAST を用いてもなお膨大な計算時間が必要となる．現在，最新の illumina 社の Hiseq 2500 という DNA シークエンサの出力は 1 ランで合計 600G 塩基にも達し，そのデータを BLAST を用いて解析するには約 25,000 CPU 日が必要になると推定される．

BLAST よりも高速な配列相同性検索を実現しているツールとして BLAT [8] がある．しかし，BLAT は BLAST よりも高速である一方，検索の感度が低いという問題がある．また，メタゲノム解析のための配列相同性検索を高速に行うために，元来，画像処理に特化した GPU (Graphics Processing Unit) を用いて配列相同性検索を行う GHOST-M[9] がある．このツールはメタゲノム解析を行う上で十分な検索感度を保ったまま，高速な計算をすることが可能だが，GPU がない環境では実行することができない．また，アミノ酸配列の文字の種類を減らす圧縮アミノ酸を用いて曖昧な検索を高速に処理することができる RAPSearch [10], [11] が 2011 年に開発され，高速な配列相同性検索を行うことができるようになった．

しかし，DNA シークエンサの進歩は凄まじく，配列相同性検索を実行する read 数は今後も増加すると考えられる．そして，DNA シークエンサの進歩により配列データの解析が進むにつれてデータベースのサイズも大きくなっている．こういった理由から，さらなる配列相同性検索の高速化が必要となっている．

BLAST などの配列相同性検索ツールの多くは seed extension という手法を用いている．この手法は seed search と extension という 2 つの処理を組み合わせて検索を行う手法である．まず，seed search はクエリとデータベースとの間で短い部分文字列 (アミノ酸配列の場合は数文字程度) の完全一致、もしくは数個のミスマッチを許して一致する部分 (seed) を転置ファイル等のインデックス構造を使用して検索する．次に extension によって seed を中心にしてアラインメントを伸長しながらアラインメントのスコアを計算する．

seed extension を用いた場合，データベースが増加してくると seed search によって多くの seed が得られる．この seed の増加により，extension の処理で多くの時間がかかる．しかし，データベースの増加により extension をしてもスコアが高くならない seed が増加してくる．このため，無駄な extension が増加するという問題が生じる．

そこで本研究では配列相同性検索の高速化のために，この seed の増加による無駄な extension の増加を抑える手法を開発した．この手法ではまず，データベースを部分文字列に分割し，類似度の高いデータベースの部分文字列同士をクラスタリングし，クエリとクラスタの代表点との類似度を元にクラスタメンバの類似度の上限値を計算する．そして，この類似度の上限値がある一定以上の値の場合，extension の処理を行うことで効果的な検索を達成した．

2. 提案手法

まず，データベースの部分文字列のクラスタリングを用いないベースとなる手法 (提案手法 1) を説明し，その後部分文字列クラスタリングを用いて拡張した手法 (提案手法 2) について説明する．

2.1 提案手法 1：ベースとなる配列相同性検索の手法

ベースとなる配列相同性検索の手法の大まかな流れは以下の通りである．

(1) ハッシュテーブルを用いた seed search の実行
(2) Seed を中心にしたギャップなしの伸長を行う ungapped extension を実行し，閾値 T_u を超えるスコアを持つ seed のみを残す
(3) 近傍にある seed を chain filter によってまとめる
(4) Seed を中心にギャップありで伸長を行う gapped extension を実行する

それぞれの処理について以降に詳しく述べる．

2.1.1 ハッシュテーブルを用いた Seed Search

BLAST の seed search では固定長の部分文字列（デフォルトでは 3 文字）の一致を 2 箇所見つけ、その部分を中心にしてアラインメントを伸長している．しかし，固定長では文字の組み合わせによって合計のスコアに大きな違いが出てくる．このため，その後のフィルタリングでフィルタアウトする seed が多くある．

この問題を解決するために，我々はあるスコアの閾値 T_{seed} を超える部分文字列の一致を seed として検索するようにした．同じように可変長の部分文字列の一致による seed search を用いた手法として [12], [13] 等があるが，seed

search で suffix array[14] を用いるためるため，seed search の際に多くのキャッシュミスが発生する．このような理由から高速な検索が難しい．

このため，本研究ではデータベースの構築の際にスコアマトリックスと閾値を予め決めておき，ハッシュテーブルによる部分文字列の一致を用いた．こうすることで配列相同性検索の際にスコアマトリックスや seed search の閾値を変更できないが，メタゲノム解析の場合，データベース構築よりも配列相同性検索のほうが多くの時間がかかるため問題にならないと考えられる．

また，ミスマッチをある程度許容するために seed search の seed の一致を検索する際には RAPSearch と同様に圧縮アミノ酸を用いた．これにより，圧縮アミノ酸による完全一致によって曖昧な文字列検索を可能にした．

2.1.2 Ungapped Extension

Seed search で得られた seed の全てに対して gapped extension を行うと多くの時間がかかる．このため BLAST と同様にしてギャップなしの伸長 (ungapped extension) を行って閾値 T_u を超えるスコアが得られる領域の seed であるかをチェックした．

Ungapped extension には BLAST と同様のパラメータによる cutoff[6] を使い，ある程度スコアが連続で低下するようならば ungapped extension の処理を打ち切るようにした．

2.1.3 Chain Filter

Ungapped extension のスコアが閾値以上の領域の seed である場合，データベースの少しずれた位置に同じように閾値を超える seed がある場合が多い．このような近傍にある seed は gapped extension でほぼ同じ結果になる．このため，gapped extension を行う前に近傍にある seed 同士をまとめる chain filter を用いた．

この chain filter では seed のクエリ側の出現位置を縦軸，データベース側の出現位置を横軸にして対角線上に並ぶ seed を 1 つにまとめるかどうかチェックする．Seed をまとめる場合の例を**図 1** に示す．

図 1(A) のように seed 同士が重なっている場合は必ず seed を 1 つにまとめる．また，図 1(B) のように seed 同士が同じ対角線上に並んでいるが離れている場合，間を ungapped extension で伸長し，途中で cutoff で打ち切られずにつながる場合 1 つにまとめるようにした．

2.1.4 Gapped Extension

Gapped extension では chain filter でまとまった seed に対してギャップありで伸長を行う．Gapped extension の際も BLAST と同様に cutoff を用いてある一定以上スコアが低下した際には伸長をうち切るようにした．

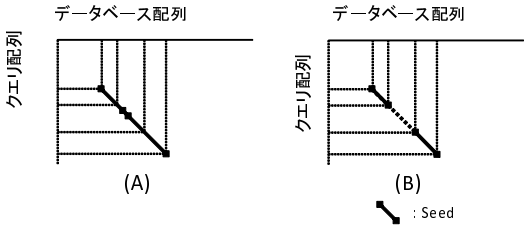

図 1 Seed をまとめる条件
Fig. 1 Conditions to reduce seeds

2.2 提案手法 2 : データベースの部分文字列のクラスタリングを用いた配列相同性検索の手法

提案手法 1 のベースとなる配列相同性検索では，検索精度を保つようにすると ungapped extension で多くの時間がかかる．これは seed search で出てくる seed が多いためである．特にデータベースが増加してくると seed search によって多くの seed が得られる．しかし，データベースの増加により extension をしても類似度が高くならない seed が増加してくる．このため，無駄な extension が増加するという問題が生じる．

この seed の増加による無駄な extension の増加を抑えるために，提案手法 2 ではデータベースを部分文字列に分割し，類似度の高いデータベースの部分文字列同士をクラスタリングし，クエリとクラスタの代表点との類似度を元にクラスタメンバの類似度の上限値を計算する．そして，この類似度の上限値がある一定以上の値の場合，extension の処理を行うようにした．

提案手法 2 ではまず，データベースの部分文字列をクラスタリングして類似度が高い部分文字列をまとめておく．

その後，以下のようにして配列相同性検索を行う．

(1) ハッシュテーブルを用いた seed search の実行
(2) クラスタメンバを持つ部分文字列に seed がヒットした場合，seed 周辺で類似度を計算し，この類似度を元にしてクラスタメンバの類似度の上限を計算してこれが閾値を超える場合，クラスタメンバの seed を生成する．
(3) Seed を中心にしたギャップなしの伸長を行う ungapped extension を実行し，閾値 T_u を超えるスコアになる seed のみを残す
(4) 近傍にある seed を chain filter によってまとめる
(5) Seed を中心にギャップありで伸長を行う gapped extension を実行する

Ungapped extension 以降の処理は提案手法 1 のベースとなる手法と同様のことを実行する．

このデータベースの部分文字列のクラスタリングにより seed search 後の ungapped extension の実行回数を減らし，高速な配列相同性検索を行うようにした．

図 2 類似度関数
Fig. 2 Similarity function

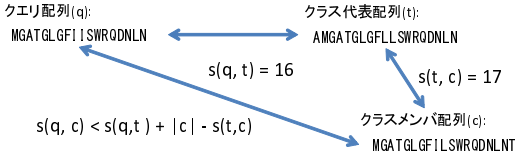

図 3 類似度の上限値
Fig. 3 The upper bound of similarity

2.2.1 類似度の上限値の計算

類似度の上限値を計算するためには，類似度の関数は三角不等式のような式を満たす必要がある．距離の関数のようにこのような性質をもつ関数は幾つかあるが，本研究では文字列間で共通する文字の数を類似度とした．例を**図 2** に示す．

この類似度を用いると類似度を計算しているクエリとなる部分文字列を q，クラスタ代表の部分文字列を t，類似度を計算する対象となるクラスタメンバの部分文字列を c とするとき，q と c の類似度 $s(q,c)$ は以下の式を満たす．

$$s(q,c) \leq s(q,t) + |c| - s(c,t) \tag{1}$$

例を**図 3** に示す．

この式を用いることで $s(q,c)$ の上限値を計算することが可能となる．

2.2.2 データベースの部分文字列の生成

データベースの部分文字列は，S_{db} 文字ずらしで L_{db} 文字の部分文字列を作る．この時，クエリの部分文字列の長さを L_q とするとき，$S_{db} = Lq, L_{db} = 3S_{db}$ となるようにした．そして seed search の際に使用するハッシュテーブルには部分文字列の S_{db} 番目以降の文字から $L_{db} - S_{db}$ までの文字列について登録するようにする．

こうすることでハッシュテーブルに重複する位置のエントリーがなくなり，かつクエリの部分文字列とデータベースのある領域との間の類似度が閾値 T_{seed} 以上となるような領域が分割されずに少なくとも 1 つ以上の部分文字列の中に含まれるようになる．

また，ある程度ミスマッチを許容するために，部分文字列は圧縮アミノ酸を用いて変換したものを使用した．

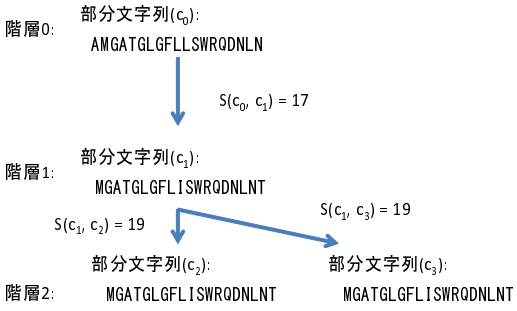

図 4 複数階層のクラスタリングによる部分文字列のクラスタリング
Fig. 4 The rank-based clustering for subsequences

2.2.3 クラスタリング手法

本研究ではクラスタリングを行う際に**図 4** のように複数階層のクラスタリングを用いた．

各階層の類似度の閾値は部分文字列の長さに比例するようにした．また i 番目の階層のクラスタメンバになる閾値の割合を t_i とする時，$i+1$ 番目の階層のクラスタメンバになる閾値 t_{i+1} は

$$t_{i+1} = \frac{1+t_i}{2} \tag{2}$$

とした．複数階層のクラスタリングを用いることで代表点となる部分文字列を増やし，より類似度が高い部分文字列のメンバとなれるようにした．

自身のクラスタ代表を探索する際には自身よりも前に出現する部分文字列のみを探索することで，部分文字列数を n とするとき，類似度の比較回数を $O(n \log n)$ となるようにし，高速化した．また，類似度の高い部分文字列を探索する際にはハッシュテーブルを使用した．

クラスタリングの際のハッシュテーブルはデータベースの部分文字列を 1 文字ずらしに K-mer と呼ばれる K 文字の固定長の部分文字列を生成し，圧縮アミノ酸を用いてハッシュキーを生成し，部分文字列の ID と K-mer の部分文字列の出現位置をハッシュテーブルに登録しておく．

こうして出来たハッシュテーブルを使い，自身の部分文字列の ID よりも前にある部分文字列の中で閾値以上の類似度となる部分文字列を見つける．ハッシュテーブルを用いた K-mer の検索では自身の K-mer を K 文字飛ばしで作り、m 個のミスマッチを許して一致する K-mer をハッシュテーブルの中から検索する．

このようにハッシュテーブルで検索をすると部分文字列の長さが L_{db} のとき，m 個のミスマッチを許容した K-mer の検索をすると，(3) より e 個以下のミスマッチで一致する部分文字列をもれなく見つけることができる．

$$e = \frac{mL_{db}}{K} - 1 \tag{3}$$

このため，K-mer の検索でもれなく最適な部分文字列の一致を見つけることができるように一番上位のクラスタの階層のメンバとなる閾値の割合 t_0 を以下のようにした．

$$t_0 = \frac{e}{L_{db}} \tag{4}$$

2.2.4 クラスタリングを用いた Seed Search

クラスタリングを用いた seed search を行う際にはクエリ側も部分文字列を作る．

その後，データベースの部分文字列のクラスタ内で重複する部分を除いたハッシュテーブルを用いて seed search を行う．

seed search 後，seed がクラスタメンバを持つクラスタ代表の部分文字列にヒットしていた場合，ungapped extension を行う前に seed の周辺で類似度を計算し，クラスタの代表部分文字列とクエリ配列間の類似度を元にしてクエリ配列とクラスタメンバの配列間の類似度の上限値がある一定以上かどうかを確認する．

クエリの長さを $|q|$，長さに対する閾値の割合を $T_{cluster}$ とすると類似度の閾値を $|q|T_{cluster}$ としてクエリの長さに比例するようにした．

このため，以下の式を満たす場合，クラスタメンバについても seed を生成するようにした．

$$|q|T_{cluster} \leq s(q,t) + |c| - s(c,t) \tag{5}$$

3. 実験

本研究で提案したベースの手法とクラスタリングを用いた手法の検索感度と計算時間を評価する実験を行った．以下に実験の詳細について述べる．

3.1 実験環境

本研究の実験に使用した計算機は TSUBAME2.0 [15] 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，コンパイラについては gcc version 4.3 を使用した．

性能の比較対象には NCBI BLAST (version 2.2.28+) と BLAT (version34 standalone)，RAPSearch (v2.16) を使用した．

BLAST はスコアマトリックスに BLOSUM62, open gap penalty と extend gap penalty はそれぞれ −11，−1，そしてフィルタと composition-based statistics[16] 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 “ -output 6 -seg no -comp_based_stats 0” である．BLAT は予め read 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．RAPSearch についてはデフォルトパラメータを使用して実験を行った．

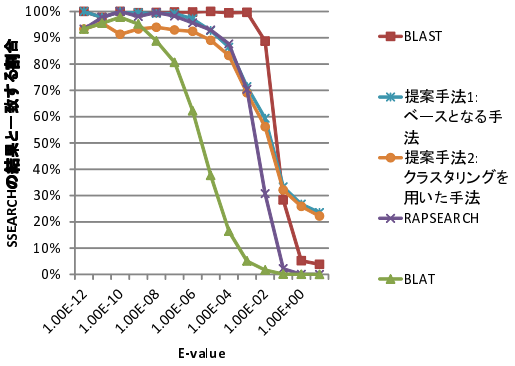

図 5 検索感度の比較
Fig. 5 Comparing of the accuracies

提案手法は，seed search のハッシュテーブルに登録する部分文字列の閾値 $T_{seed} = 39$，クラスタリングを行う部分文字列のスキップ数 $S_{db} = 16$，データベースの部分文字列の長さ $L_{db} = 48$，クエリの部分文字列の長さ $L_q = 16$，クラスタリングを行う際の階層 0 の閾値 $t_0 = 0.9$，クエリとクラスタ代表を比較した際に seed とする類似度の閾値の割合を $T_{cluster} = 0.6$ とした．また，使用した圧縮アミノ酸は RAPSearch と同様のものを用いてアミノ酸を 20 種類から 10 種類に圧縮したものを用いた．これらのパラメータは RAPSearch と比較して同程度の検索感度を持つように最適化されたものである．

アミノ酸配列データベースは 2010 年 11 月時点の KEGG Genes (genes.pep) [17], [18], [19] 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 illumina 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．

3.2 検索感度の評価実験

上記のデータを用いて提案手法の検索感度の評価実験を行った．検索感度を評価するために各ツールの検索結果を SSEARCH[20] の結果と比較し，クエリ毎に各ツールの Top スコアのデータベース配列が SSEARCH の結果の Top スコアのデータベース配列と一致している割合によって評価を行った．

結果を**図 5** に示す．図 5 は横軸が E-value[21]，縦軸が SSEARCH の Top と一致する割合を示した．結果を見ると提案手法 1 のベースとなる手法はほぼ RAPSearch と同

表 1 相同性検索の計算時間と BLAST との計算速度比
Table 1 The computation time and the acceleration ratio with respect to the BLAST

	計算時間 (sec.)	BLAST との計算速度比
提案手法 1 ベースとなる手法	274	133.6
提案手法 2 クラスタリングを用いた手法	345	105.8
BLAST	36,535	1.0
BLAT	855	42.7
RAPSearch	284	128.7

表 2 それぞれの手法のハッシュテーブルのエントリー数
Table 2 The number of entries in hash table for each methods

	ハッシュテーブルのエントリー数
提案手法 1 ベースとなる手法	2,047,436,201
提案手法 2 クラスタリングを用いた手法	1,442,778,441

程度の精度がでている．このため，メタゲノム解析に用いるために十分な精度があると考えられる．

しかし，提案手法 2 のクラスタリングを用いた手法は，提案手法 1 のベースとなる手法よりも検索感度が低い．これはクラスタとなっているデータベースの部分文字列については別途類似度の計算をしているため，その部分で閾値を超えられずに計算されなかった部分が原因であると考えられる．ただ、低下率は数%以内であるためほぼ問題ないと考えられる．

3.3 計算時間の評価実験

検索感度に用いたデータを使って提案手法の計算時間の評価実験を行った．各ツールの計算時間を**表 1** に示す．

計算時間は提案手法 1 のベースとなる手法は BLAST を 1 とした場合，約 134 倍と全ツールの中で最速であった．一方，提案手法 2 のクラスタリングを用いた手法はベースとなる手法よりも遅く約 106 倍にとなっている．

以下でこの速度低下について考察する．

3.4 クラスタリングされたことによるハッシュテーブルのエントリー数の比較

提案手法 1 のベースとなる手法と提案手法 2 のクラスタリングを用いた手法で seed search の時に使用するハッシュテーブルのエントリー数がどれくらい違うのかを比較する．それぞれの手法のハッシュテーブルのエントリー数は**表 2** の通りである．表 2 よりクラスタリングを用いることで約 70%に縮小できたことがわかる．このことからクラスタリングによりハッシュテーブルのサイズが縮小できたことがわかる．

表 3 それぞれの手法の ungapped extension の実行回数と類似度比較の回数
Table 3 The number of calls of ungapped extension and similarity checking in each methods

	Ungapped extension	類似度比較	合計回数
提案手法 1 ベースとなる手法	1,619,040,297	0	1,619,040,297
提案手法 2 クラスタリングを用いた手法	915,742,250	389,069,800	1,304,812,050

3.5 Ungapped extension と類似度比較の回数

提案手法 1 のベースとなる手法の ungapped extension の実行回数と，提案手法 2 のクラスタリングを用いた手法の ungapped extension の回数と類似度比較の回数を比較する．

それぞれの手法の ungapped extension の実行回数と類似度比較の回数について**表 3** に示す．Ungapped extension と類似度比較は厳密には実行していることは違うが，計算量はほぼ同じである．このため，クラスタリングを用いた手法はこの 2 つの合計数で比較する．このとき，提案手法 1 のベースとなる手法に比べ提案手法 2 のクララスタリングを用いた手法の場合，実行回数は約 80%に抑えられている．このため，クラスタリングを用いたほうが実行回数については少なくなっていることがわかる．

しかし, ungapped extension と類似度比較を Intel VTune Amplifier XE 2011 でプロファイリングしてみると，データベースの部分文字列のクラスタメンバの情報を取り出す分のメモリアクセスが増えたことにより計算時間が増加していた．この結果から，クラスタリングを用いることでメモリアクセスが増加するため計算時間が少なくならなかったと考えられる．

4. 結論

4.1 本研究の成果

本研究ではまずベースとなる高速な配列相同性検索の手法を提案し，それを拡張したデータベースのクラスタリングを用いた手法を提案した．提案手法 1 のベースとなる手法については RAPSearch と同程度以上の検索感度で，より高速な配列相同性検索を行うことができ，BLAST の約 134 倍の高速化を達成した．

また，提案手法 2 のデータベースの部分文字列のクラスタリングを用いた手法ではベースの手法よりも検索速度を保ったままで，ungapped extension の回数を減らすことに成功した．しかし，計算速度としてはメモリアクセス回数が増加してしまい，逆に改良前よりも低下してしまった．だが，今後，メモリアクセスを減らすことができれば，計算の回数は低下させることに成功しているため，高速な配列相同性検索が実現できる可能性がある．

4.2 今後の課題

データベースの部分文字列のクラスタリングを用いた手法を高速化するために，メモリアクセスを減らす必要がある．こうすることで ungapped extension の回数を減らすことによる高速化の恩恵が得られると考えられる．また，DNA シークエンサの進歩は目覚ましく，出力される read の長さも徐々に伸びていることから長い read に対する高速化が必要であると考えられる．

謝辞 本研究は，科研費（特別研究員奨励費 24・8766），HPCI 戦略プログラム分野 1「予測する生命科学・医療および創薬基盤」の支援を受けて行われたものである．

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] Langmead B, Salzberg SL: “Fast gapped-read alignment with Bowtie 2”, *Nature Methods*, 9(4):357–359 (2012).
[5] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: “Basic local alignment search tool”, *Journal of Molecular Biology*, 215:403–410 (1990).
[6] Altschul SF, Madden TL, Schaffer AA, *et al*: “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs”, *Nucleic Acids Research*, 25(17):3389–3402 (1998).
[7] Dalevi D, Ivanova NN, Mavromatis K, *et al*: “Annotation of metagenome short reads using proxygenes”, *Bioinformatics*, 24(16):7–13 (2008).
[8] Kent WJ: “BLAT–the BLAST-like alignment tool”, *Genome research*,12(4):656–64 (2002).
[9] Suzuki S, Ishida T, Kurokawa K, Akiyama Y: “GHOST-M: A GPU-Accelerated Homology Search Tool for Metagenomics”, *PLoS ONE*, 7(5):e36060 (2012).
[10] Ye Y, Choi JH, Tang H: “RAPSearch: a fast protein similarity search tool for short reads”, *BMC Bioinformatics*, 12(1):159 (2011).
[11] Zhao Y, Tang H, Ye Y: “RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data”, *Bioinformatics*, 28(1):125–126 (2012).
[12] 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).
[13] 鈴木 脩司 , 石田 貴士 , 秋山 泰: “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への対応””, 情報処理学会, 研究報告, 2012-BIO-29(24), 1–7 (2012).
[14] 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).
[15] 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
[16] Altschul SF, Wootton JC, Gertz EM, Agarwala R, Morgulis A, *et al*: “Protein database searches using compositionally adjusted substitution matrices”,*FEBSJ* , 272:5101–5109 (2005).
[17] 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).
[18] 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).
[19] Kanehisa M, Goto S: “KEGG: Kyoto Encyclopedia of Genes and Genomes”, *Nucleic Acids Res*, 28:27–30 (2000).
[20] Smith TF, Waterman MS: “Identification of Common Molecular Subsequence”, *Journal of Molecular Biology*, 147(1):195–197 (1981).
[21] 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).