Vol.2010-MPS-81 No.20
Vol.2010-BIO-23 No.20
2010/12/17

FM-indexを用いた高速な配列相同性検索ツールの開発

鈴木 脩司[†1]　石田 貴士[†1]　秋山 泰[†1]

近年，DNA 配列等の配列決定技術の向上により高速に配列データを得ることが可能となった．これにより DNA 配列及びタンパク質配列のデータベースのデータ量が爆発的に増加している．このため大量の配列データに対して巨大な DB への相同性検索を行う機会が多くなってきてる．しかし，大規模なデータを用いた相同性検索では，BLAST など従来のツールでは解析が間に合わないという問題がある．本研究では Suffix Array を用いてクエリのインデックスを，FM-index を用いて DB のインデックスを構築し，これらのインデックスを用いてミスマッチをある程度許して短い領域で高いスコアとなる部分を見つけ，その部分の周辺をアラインメントするアルゴリズムを提案した．その結果，従来用いられてきた BLAST 以上の精度を保ったまま，約 10 倍の高速化を達成した．

Development of a fast homology search tool based on FM-index

Shuji Suzuki,[†1] Takashi Ishida[†1]
and Yutaka Akiyama[†1]

In recent years, a lot of biological sequence data can be determined easily and the size of DNA/protein sequence databases is increasing explosively because of the improvement of sequencing technologies. However, such a huge sequence data causes a problem that even general homology search analyses by using BLAST become difficult in terms of the computation cost. Therefore, we designed a new homology search algorithm that finds alignment candidates based on the suffix array of queries and the FM-index of a database. As results, the proposed method achieved about 10-fold speed up than BLAST.

†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

1. はじめに

類似した配列を持つタンパク質の間では，その構造や機能も同様によく似ている．そのためタンパク質の機能や構造，進化についての情報を得るため，そのタンパク質配列間で相同性検索を用いて解析を行うことが多々ある．しかし，近年，DNA 配列やタンパク質配列の配列決定技術の向上により高速に配列データを得ることが可能となった．これにより DNA 配列やタンパク質配列のデータベースのデータ量が爆発的に増加している．このため大量の配列データに対して巨大な DB への相同性検索を行う機会が多くなってきている．最適アラインメントを計算する際には動的計画法を用いた Smith-Waterman アルゴリズム[1] の実装である SSEARCH[2] 等を利用することが望ましい．しかし，SSEARCH は低速であるため，大規模なデータに対して用いることは少ない．このため，高速に相同性検索が可能な近似的手法である FASTA[3] や BLAST[4),5] が広く使われてきたが，最新のシーケンサーで得られる大規模なデータを用いた相同性検索では，BLAST を用いても解析が間に合わないという問題がある．

本研究では近年大規模な DNA 配列のマッピングに用いられている技術を応用して配列相同性検索を高速に行う手法を提案し，その実装を行った．

2. 手　法

本研究で提案する手法では，検索対象となるクエリと DB 間で局所的にスコアの高くなる位置を探索し，その後，探索で見つけた位置を中心にギャップなども考慮しながら伸長を行い高いスコアとなる配列を検索する．提案する手法は DNA 配列同士の比較にも，タンパク質配列同士の比較にも利用できるものである．しかし本稿ではタンパク質配列の比較のみを扱うものとする．

BLAST では最初の探索の部分では固定長の部分文字列とその部分文字列に対して数文字置換した近傍の文字列を探索しているが，固定長であると柔軟性が十分でないため，感度を保証するために近傍の文字列を列挙する際の閾値を低くしなければならない．このため，本研究では任意の長さの探索を行える Suffix Array[6] 系のインデックスを用いて高い閾値を用いても感度を落とさずに探索を行えるようにした．Suffix Array と FM-index[7] の詳細を以降に述べる．

2.1 Suffix Array

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

Vol.2010-MPS-81 No.20
Vol.2010-BIO-23 No.20
2010/12/17

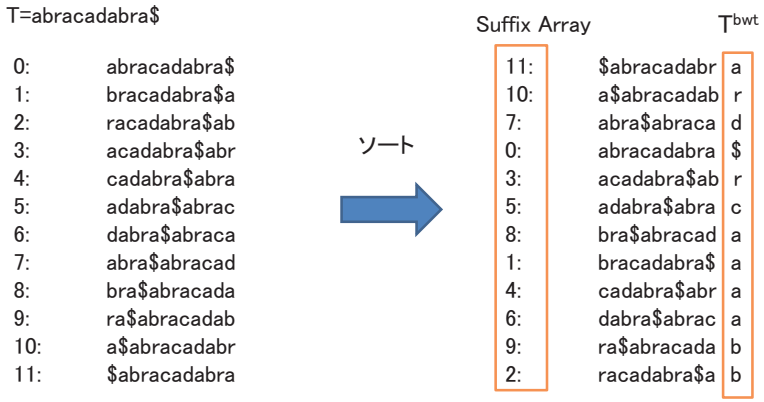

図 1 Suffix Array と BWT の例
Fig. 1 An example of Suffix Array and BWT

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．ただし，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 FM-index

FM-index は Burrows-Wheeler Transform[8](BWT) を用いた self-index の一つである．FM-index は Suffix Array と同じように任意の長さの文字列を検索できるが，必要なメモリが Suffix Array よりも少ないため，Suffix Array の代わりに用いられることが多い．この FM-index に使われている BWT は可逆変換の一種で，主にデータ圧縮の前処理として用いられてきた．

文字列 T の Suffix Array を SA とし，T の BWT によって変換した文字列を T^{bwt} とする．ここで T^{bwt} は $SA[i] = 0$ のとき，$T^{bwt}[i] = \$$，その他のとき $T^{bwt}[i] = T[SA[i] - 1]$ となる．図 1 に BWT で変換する例を示す．FM-index では T の Suffix Array を用いずにソートされた接尾辞中で接尾辞の先頭に文字列 P が最初に出現する位置 sp と最後に出現する位置 ep を検索することができる．

$FM\text{-}indexSearch(P[1..m], T^{bwt}[1...n])$

1: $i \Leftarrow m$
2: $sp \Leftarrow 1$
3: $ep \Leftarrow n$
4: **while** $sp \leq ep$**and** $(i \geq 1)$ **do**
5: $\quad c \Leftarrow P[i]$
6: $\quad sp \Leftarrow C[c] + rank_c(T^{bwt}, sp - 1) + 1$
7: $\quad ep \Leftarrow C[c] + rank_c(T^{bwt}, ep)$
8: $\quad i \Leftarrow i - 1$
9: **end while**
10: **if** $ep < sp$ **then**
11: $\quad$ **return** P は出現しない
12: **else**
13: $\quad$ **return** sp, ep
14: **end if**

図 2 FM-index での検索
Fig. 2 Search using FM-index

図 2 に FM-index を用いた文字列 $T[1...n]$ 中に文字列 $P[1...m]$ の出現する $SA[sp, ep]$ を検索する擬似コードを示す．ここで，$C[c]$ は $\{\ \$\ , 1, ..., c-1\}$ の文字の出現数が保存されている．また，$rank_c(T^{bwt}, i)$ は $T^{bwt}[1, i]$ 中で c の出現数を返す関数である．

FM-index において $rank_c(T^{bwt}, i)$ を少ないメモリで計算する手法は多く提案されている[9]．このため FM-index はメモリの消費が少なく，また $O(|P|)$ で P がソートされた接尾辞中で接尾辞の先頭に出現する範囲を検索できる．しかし，P が T 中のどの位置で出現したのかを高速に計算するために Suffix Array の一部を保持しておく必要がある．

2.3 クエリと DB の Suffix Array 系のインデックスを用いた最適アラインメントの候補位置の探索

ここでは説明のためにクエリと DB ともに Suffix Array によってインデックスを構築するものとする．実際に実装した方法については 2.4 で説明する．クエリ，DB 内すべてのタンパク質配列中に出現しない文字 # を配列の区切り文字とし，配列と配列の間にこの区切

Vol.2010-MPS-81 No.20
Vol.2010-BIO-23 No.20
2010/12/17

り文字を挿入して連結していく．ここで # と $ は違うものであるため区別して扱う．この連結配列を用いて Suffix Array(SA_q) を構築する．DB も同様にして連結配列を作成して Suffix Array(SA_d) を構築する．この Suffix Array を用いてスコアが T_s 以上になるクエリと DB の位置のペアを探索する．**図 3** にクエリと DB の Suffix Array を用いた最適アラインメントの候補位置の探索する擬似コードを示す．この探索は FM-index のように Suffix Array 系のインデックスならばほぼ同様に実行することが可能である．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) 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，D は $maxScore$ との差をどれだけ許すかを示す値，$S[c, c']$ は文字 c, c' に対するスコアマトリックスの値である．

先に述べたように BLAST では固定長の部分文字列とその部分文字列に対して数文字置換した近傍の文字列を用いて探索を行うが，固定長だと感度を良くするために近傍の文字列を列挙する際の閾値を低くしなければならない．このため，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 系のインデックス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．こうすることで，一致率の高い文字列では短い文字列でヒットとし，また一致率の低いものは長い文字列でヒットとす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 DB の両方で Suffix Array 系のインデックス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ことができる．

2.4 提案する相同性検索の手法

Suffix Array を用いた最適アラインメントの候補位置の探索による相同性検索の手法について説明する．予め，DB 中の配列で連結配列を作成し，インデックスを構築する．DB は連結配列長が長いためインデックスには多くのメモリが必要となる．このため DB 側は少ないメモリで構築可能な FM-index を用いた．FM-index は wavelet tree[10] を用いて構築した．ただし，DB のインデックスに FM-index を使うため，連結配列を逆順にしたものを用いて FM-index を構築する．こうすることで探索するパターンを逆順に探索する必要がなくなる．そして相同性検索を行うクエリをファイルから読み込み後，DB と同様に連結配列を作成し，その連結配列を用いて Suffix Array を構築する．大量のクエリを処理しなければならず，メモリの消費を抑えたい場合はクエリ側も FM-index を用いるべきだが，3 で用いた程度のクエリならば十分メモリに収まるサイズであるため，Suffix Array を用いた．そして，DB のインデックスとクエリのインデックスを用いて最適アラインメントの候補位置の探索を行う．その後，最適アラインメントの候補位置の探索でヒットした位置を中心にして Ungapped Extention を行う．Ungapped Extention は BLAST と同じようにスコアが低下し始めたら，伸長を停止しその時のスコアが T_g を超えた候補のみ，候補位置を中心に

```
Search(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:             Search(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 DB の Suffix Array を用いた最適アラインメントの候補位置の探索
Fig. 3 Search for candidate positions of optimum alignment using Suffix Array for Queries and DB

Vol.2010-MPS-81 No.20
Vol.2010-BIO-23 No.20
2010/12/17

Gapped Extention を行う．Gapped Extention も BLAST と同じように，スコアが低下し始めたら，伸長を停止し，Gapped Extention のスコアを候補位置のスコアとする．その後，クエリ毎にヒット位置をまとめ，スコアが高いものを相同性検索の結果として出力する．

3. 実験結果

本研究で提案した相同性検索ツールの検索精度と計算速度を測る実験を行った．実験に使用した CPU は Intel(R) Core(TM) i7 CPU 975 (3.3GHz)，メモリは 12GB，OS は CentOS 5.4 である．比較に使用した BLAST は NCBI BLAST(version 2.2.22) である．使用した BLAST のオプションはスコア行列に BLOSUM62，open gap と extend gap はそれぞれ 11，1，そしてフィルタを使用しない，また 1 スレッドで計算を実行するようにした．具体的に使用した BLAST のオプションは “-p blastp -m 8 -b 10 -G 11 -E 1 -g T -F F -a 1 -M BLOSUM62” である．また，今回用いた提案手法のパラメータはいくつかのパラメータの値を試したうち最適だった $T_s = 30, L_{max} = 8, D = 7$ を用いた．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．以下に実験の手順と結果について述べる．

3.1 使用データ

本研究では検索精度比較用のクエリデータとして，NCBI の Web サイトから 2010 年 10 月 28 日にダウンロードしたタンパク質立体構造 DB のタンパク質配列である pdbaa にランダムで置換，挿入，削除の操作を行って製作した配列 1 万本を使用した．このクエリの全配列の合計長は約 200 万残基となっている．また，検索精度比較用の DB にはクエリで使用した pdbaa をそのまま使用した．pdbaa は約 5 万本，全配列の合計長は約 110 万残基で構成されている．

一方，速度比較用のクエリデータとして，pdbaa からランダムに 10，100，1,000，10,000 本を選び，検索精度用のクエリと同様に置換，挿入，削除の操作を行ったものを使用した．速度比較用の DB にはサイズの小さい DB として pdbaa とサイズの大きい DB として NCBI の Web サイトから 2010 年 10 月 28 日にダウンロードした nr-aa を使用した．nr-aa には約 120 万本，全配列の合計長は約 40 億残基で構成されている．検索精度比較用に大規模な nr-aa を使わずにサイズの小さい pdbaa のみを利用した理由は，正解を定義するための SSEARCH による計算に時間が掛かり過ぎ，測定できなかったためである．

3.2 検索精度の比較

検索精度を比較するために，BLAST と提案手法でクエリ毎にアラインメントの正確さを評価した．正解には厳密な動的計画法を行う SSEARCH の結果を用いて，両手法の結果に SSEARCH が出力するスコアの最も高いクエリと DB 内の配列のペアが含まれる割合によって比較した．ただし，アラインメントの詳細な正確性に関しては考慮していない．SSEARCH の最もスコアが高い結果の E-value 毎のクエリ数の分布を**図 4** に示す．図 4 より，E-value が 1.0×10^{-12} 以下 (左端) と 1 以上 (右端) の部分のクエリの本数が約一桁程多いが，他の部分の本数はほぼ同じことが分かる．このため検索精度を測るテストデータとしてはバランスが取れており適切だと考えられる．次に E-value を変化させたときの BLAST と提案手法の正解が含まれる割合を**図 5** に示す．結果として，提案手法は E-value が 1 以上では精度が落ちているものの，その他の広い領域では BLAST よりも高い精度があることがわかった．

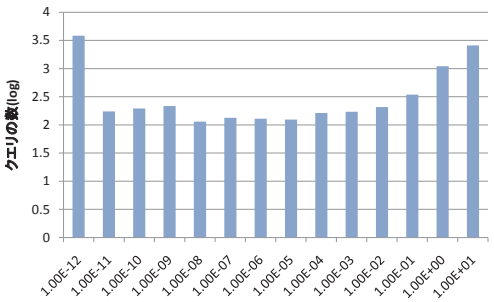

図 4 1 万本のクエリの検索精度用 DB に対する E-value 毎の本数の分布
Fig. 4 Number of queries vs. E-value for the DB for Case sensitive matching(10,000 queries)

3.3 計算速度の比較

計算速度を比較するために，クエリの本数を変えて DB を pdbaa または nr-aa とした場合の BLAST と提案手法で計算時間を比較した．BLAST と提案手法の計算時間に関する表を**表 1**，**表 2** に示す．また，BLAST の計算速度を 1 とした場合の計算速度向上比のグラフを**図 6** に示す．結果として DB に nr-aa を用いた場合は BLAST に比べて最大約 10 倍の高速化を達成した．

3.4 考　　察

3.4.1 検索精度について

提案手法は BLAST と比較すると E-value が極端に高い場合を除き，広い範囲で検索精度が良くなっている．これは提案手法はある程度ならミスマッチが入っていても候補にし

Vol.2010-MPS-81 No.20
Vol.2010-BIO-23 No.20
2010/12/17

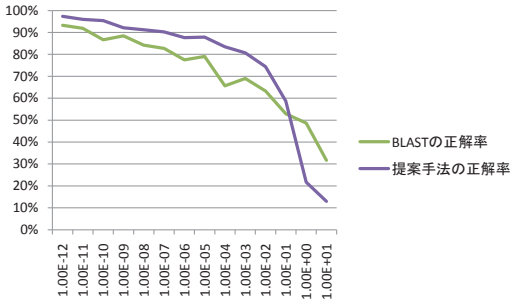

図 5 E-value を変化させた場合の正解が含まれる割合（％）
Fig. 5 Correct alignment rate for each E-value（％）

表 1 pdbaa を用いた場合の計算時間 (sec)
Table 1 Calculation time using pdbaa(sec)

クエリの本数	BLAST	提案手法	速度向上比
10	1	1	1.0
100	14	8	1.8
1,000	149	76	2.0
10,000	1,545	689	2.2

表 2 nr-aa を用いた場合の計算時間 (sec)
Table 2 Calculation time using nr-aa(sec)

クエリの本数	BLAST	提案手法	速度向上比
10	391	291	1.3
100	3,617	620	5.8
1,000	37,064	3,799	9.8
10,000	385,787	51,756	7.5

ているため，BLAST では拾いきれない候補を拾うことができるためと考えられる．また，E-value が高い位置では BLAST よりも精度が悪くなっているのは，提案手法は局所的にある程度スコアが高くならないと候補を拾うことができないため，局所的にスコアが高くならないアラインメントの候補をうまく拾うことができず精度が落ちたと考えられる．しかし，実際は E-value が高いヒットは偶然でも得られる可能性があるため，通常生物学の解析では 1.0×10^{-3} 程度までしか利用されない．このため，実用的には提案手法で十分な精度が得

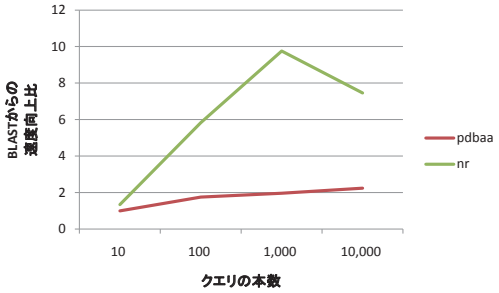

図 6 計算速度向上比（BLAST を 1 とした場合）
Fig. 6 Speed up ratio (Compared to BLAST)

られていると考えられる．

3.4.2 計算速度について

クエリの本数が少ない場合は BLAST との速度向上比が小さいが，クエリの本数が増加するにつれて徐々に速度向上比が大きくなる．これはクエリの本数が少ない時には I/O に多くの時間が割かれているためである．このため，ある程度クエリをまとめて処理した方が効率が良いことがわかる．また，DB のサイズが小さい pdbaa よりもサイズが大きな nr-aa の方が速度向上比が大きくなっている．このため，本手法は DB のサイズが大きい場合ほど，効率よく計算できるといえる．また，DB に nr-aa を用いた際，クエリが 1 万本の時には速度向上比がやや小さくなっている．これは最後の Gapped Extention にかかる時間の割合が増加しているためである．

4. 結　　論

4.1 本研究の成果

本研究ではクエリと DB の両方でインデックスを構築し高速にアラインメントの候補を検索する手法を提案し，その手法の実装を行った．実装を行った新システムでは E-value の極端に高い領域では BLAST と比べると若干精度が落ちるものの，その他の広い領域では同等以上の精度が得られ，実用上，十分な精度を持っていることが示された．また，計算時間において DB に nr-aa を用いた場合は BLAST に比べて最大約 10 倍の高速化を達成した．

4.2 今後の課題

今後，DNA 配列等の配列決定技術の向上により大量なデータを得ることが可能となり，

Vol.2010-MPS-81 No.20
Vol.2010-BIO-23 No.20
2010/12/17

さらなるアラインメントの高速化が必要と考えられる．このため，CPU による並列化や GPU を用いた並列化などにより高速に計算する必要がある．

参 考 文 献

1) Smith TF, Waterman MS: “Identification of Common Molecular Subsequence”, *Journal of Molecular Biology*, 147(1):195-197(1981).
2) Pearson WR., Searching protein sequence libraries: “comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms”, *Genomics*, 3:635-650(1991).
3) Pearson WR, Lipman DJ: “Improved tools for biological sequence comparison”, *Proceedings of the National Academy of Sciences of the United States of America*, 85(8):2444-2448(1988).
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(1988).
6) 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).
7) Ferragina P, Manzini G: “Opportunistic data structures with applications”, *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, 390-398(2000).
8) Burrows M, Wheeler DJ: “A block-sorting lossless data compression algorithm”, *Systems Research*, Research R:24-42(1994).
9) Navarro G, Makinen V: “Compressed full-text indexes”, *ACM Computing Surveys*, 39(1):2(2007).
10) Grossi R, Gupta A, Vitter JS: “High-order entropy-compressed text indexes”, *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, 841-850(2003).