2004－DBS－134　(I)　(10)
2004／7／13

類似配列検索のための配列アラインメントアルゴリズムの高速化

吉田 孝廣† 牧之内 顕文††

配列アラインメントは，遺伝子やタンパク質の配列データベースから類似配列を検索するために用いられる，生命情報科学の分野において最も重要な技術の 1 つである．動的計画法に基づいた配列アラインメントの手法は広く使われている．もし今より高速なアルゴリズムが登場すれば，生命情報科学の可能性は広がると考えられる．

我々は動的計画法に基づいた配列アラインメントの新しいアルゴリズムを提案する．実験では，我々のアルゴリズムの性能は，後藤のアルゴリズムや Myers と Miller のアルゴリズムの性能を上回った．これらは，動的計画法に基づいたグローバルアラインメントのアルゴリズムとして最も有名なアルゴリズムである．またローカルアラインメントについても，有意義な結果が得られた．

A Fast Algorithm for Sequence DB Refinement

Takahiro Yoshida† and Akifumi Makinouchi††

The sequence alignment, which is the technique to find homology sequences from the gene and protein sequence database, is one of the most important techniques for Bioinformatics. The sequence alignment based on dynamic programming method is the most widely used method. If high-speed algorithm is offered, the application of Bioinformatics will spread further.

We propose a new algorithm of sequence alignment based on dynamic programming. In our experiments, this algorithm outperforms Gotoh's algorithm and Myers-Miller's algorithm, both of which are the most famous algorithm of global alignment based on dynamic programming. For local alignment, tests were made the result which shows usefulness of our algorithm, too.

1. はじめに

我々は生物の塩基配列やアミノ酸配列データベースのための高速類似検索アルゴリズムの研究開発を行っている．この類似検索では，空間データベース検索と同様に，フィルタリング (filtering) ステップとリファインメント (refinement) ステップをとるのが効果的である．リファインメントステップでは，索引を使って検索候補を絞り込む．リファインメントステップでは，絞り込まれた候補集合から，新に意味のえる配列を検索する．

後者では，入力問い合わせ配列と候補配列のひとつひとつとを「配列アラインメントアルゴリズム」を使ってアラインメントし，候補の中から最も検索配列に類似している (類似度の高い) 配列を選ぶ．この操作は，候補配列の数が多い場合にはかなりの時間がかかる．従って，配列アラインメントアルゴリズムの高速化は類似検索時間の短縮につながる．本研究はこの面からの研究成果のひとつである．

配列アラインメントでは，2 本の文字列を各文字ごとに照合し，一致数が最大になるように整列させる．ギャップ ("-"，ハイフンで表す) は，一致数がより大きくなるような列 (アラインメントという) を得るために文字列間に挿入される．アラインメントは，文字が一致した時やギャップを挿入した時に与える得点の合計が最大となるように導かれる．この得点の合計をアラインメントスコアという．すなわち配列アラインメントとは，最大となるアラインメントスコアが導かれるような最適なアラインメントを求めることである．

配列アラインメントは遺伝子病の原因を探るために用いられる．遺伝子学者は健常者と患者の遺伝子の配列やタンパク質の配列を比較する．もし，患者の配列から健常者の配列にはない異常部位が発見されれば，その部位が病気の原因である可能性が高い．

† 九州大学大学院システム生命科学府
Graduate School of Systems Life Sciences,
Kyushu University
†† 九州大学大学院システム情報科学研究院
Faculty of Information Science
and Electrocal Engineering,
Kyushu University

動的計画法に基づいた配列アラインメントは，タンパク質や遺伝子の配列の比較に広く用いられている．配列アラインメントの高速なアルゴリズムが開発されれば，薬学，医学，農学の分野によりよい機能を持つ機器を提供できる[8]．

この論文は，後藤のアルゴリズムと Myers と Miller のアルゴリズムに基づいた新しいアルゴリズムについて書いたものである．実験結果によると，我々のアルゴリズムはグローバルアラインメントについて，それら 2 つのアルゴリズムより高速であった．またローカルアラインメントについても，それらに劣らぬ性能を示した．

2. 関 連 研 究

動的計画法に基づいた配列アラインメントのアルゴリズムでは，メモリ上に 2 次元配列を確保することを想定している．Charter らが指摘するように[7]，アルゴリズムは 2 つの手続きに分けることができる．1 つは 2 次元配列各要素の値を計算する手続き，もう 1 つは 2 次元配列の要素をたどり，アラインメントスコアの最大値を導く最適なアラインメントを求める手続きである．$O(MN)$ のメモリ領域を使うアルゴリズムをフルマトリックスアルゴリズムという．このことが，配列上を $O(M+N)$ 回たどることで最適なアラインメントを求めることを可能にした (M, N は入力配列の長さ)．

現在，生物配列の配列アラインメントに用いられるアルゴリズムは，Needlman と Wunsch の革新的な業績に由来する[1]．Needlman と Wunsch は動的計画法を用いたグローバルアラインメントのアルゴリズムを提案した．このグローバルアラインメントとは，2 本の配列を全体的に比較し整列させる配列アラインメントのことである．

しかし，線形ギャップペナルティを用いたアルゴリズムの時間計算量が $O(MN)$ であるのに対して，アフィンギャップペナルティを用いたアルゴリズムの時間計算量は $O(M^2N)$ となる．

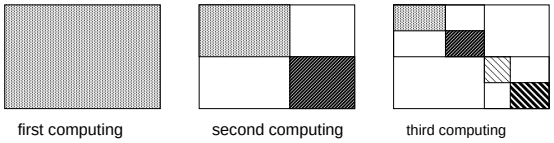

図 1 分割統治法
Fig. 1 Divide-and-conquer procedure

Hirschberg は，領域計算量 $O(MN)$ の Needlman と Wunsch のアルゴリズムを，領域計算量 $O(M+N)$ で行うアルゴリズムを提案した[2]．Hirschberg は，分割統治法を図 1 のように再帰的に用いることで，領域計算量を削減した．

Smith と Waterman は Needlman と Wunsch のアルゴリズムを，2 つの入力配列の最も類似性 (相同性) が高い部分列を探すように拡張した[3]．ここで使う類似性は，任意長の欠失や挿入を許す．このような部分列を求める配列アラインメントをローカルアラインメントという．

後藤は時間計算量 $O(MN)$ のアフィンギャップペナルティを用いたアルゴリズムを提案した[4]．2 つの入力配列の長さが M と N であるとき ($M \leq N$)，時間とメモリ領域は $M \times N$ に比例する．アルゴリズムは 2 つの段階からなる．まず $(M+1) \times (N+1)$ の値を計算して最適なアラインメントのスコアを求める．それから計算した値をたどって，最適なアラインメントを求める．

Myers と Miller は，後藤のアルゴリズムと Hirschberg のアルゴリズムを組み合わせて，領域計算量 $O(M+N)$ のグローバルアラインメントのアルゴリズムの高速化を実現した．

Huang，Hardison，Miller は，Smith と Waterman のアルゴリズムに代表されるローカルアラインメントのアルゴリズムに，Myers と Miller のアルゴリズムの技術を組み合わせた[6]．このアルゴリズムは，始めにローカルアラインメントの両端を求め，次にその領域における最適なアラインメントを Myers と Miller のアルゴリズムを用いて求める．

Charter，Schaeffer，Szafron は FastLSA(Fast Linear-Space Alignment) アルゴリズムを提案した[7]．Hirschberg のアルゴリズムは行の情報を保存することで，再帰的に計算領域を半分に分割して計算を行う．しかし，行と列両方の情報を保存し，計算領域を行方向と列方向の両方に二分割できれば，より少ない再計算で最適なアラインメントを求めることができる．計算領域を k^2 $(k \geq 2)$ の領域に再帰的に分割することで，FastLSA は実行時間を短縮した．ただし，$2 \times k \times n$ 本の境界の情報を保存するので，確保するメモリ領域は増える．$k = m$ のとき，このアルゴリズムはフルマトリックスアルゴリズムと等価なアルゴリズムになる．

数年後，Charter らは Huang らのアルゴリズムの考えを基にして FastLSA を拡張した[9]．

我々は後藤のアルゴリズムと Myers と Miller のアルゴリズムを比較して，2 つのアルゴリズムの長所を調べた．それらに基づいて，新しいアルゴリズムを提

案する．

3. アルゴリズム

我々が実験を行ったところ，Myers と Miller のアルゴリズムでは，後藤のアルゴリズムの約半分の時間で最適スコアを計算することができた．しかし，最適なアラインメントを求めるまでにかかる時間については，両者にあまり差はなかった．これは，Myers と Miller のアルゴリズムが，最適なアラインメントを求めるために，再計算を行う必要があるためである．それに対し，後藤のアルゴリズムは再計算を行う必要がない．なぜなら，後藤のアルゴリズムでは，スコア計算をする際に計算の経路を保存しておき，それを使って最適なアラインメントを求めるからである．

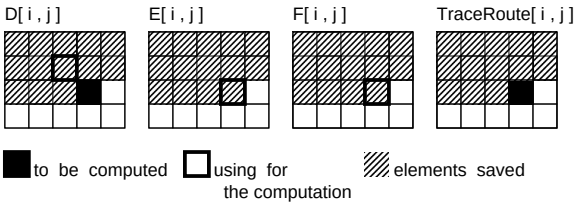

図 2 後藤のアルゴリズムによるメモリ確保
Fig. 2 The memory space allocation by Gotoh's algorithm

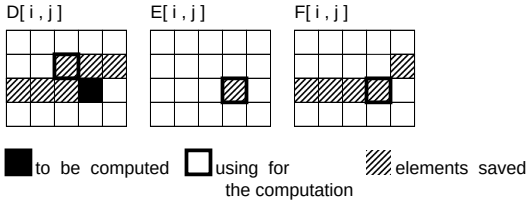

図 3 Myers と Miller のアルゴリズムによるメモリ確保
Fig. 3 The memory space allocation by Myers-Miller's algorithm

図 2 と図 3 は，2 つのアルゴリズムで，"to be computed"の値を計算している時に値が保存されているメモリ領域を示している．我々は，これら 2 つのアルゴリズムの特徴から新しいアルゴリズムのアイディアを得た．Myers と Miller のアルゴリズムでは，最適スコアを計算する間に，計算の経路を保存していないので，再計算が必要となる．後藤のアルゴリズムでは，計算の経路を保存しているので再計算する必要がない．すなわち，最適スコアを計算する間に計算の経路を保存していれば，Myers と Miller のアルゴリズムでも，再計算せずに最適なアラインメントを求めることができる．

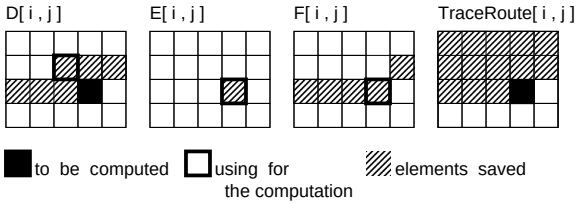

図 4 吉田・牧之内のアルゴリズムによるメモリ確保
Fig. 4 The memory space allocation by Yoshida-Makinouchi's algorithm

このことから，新しいアルゴリズムを考えた．我々はそれを吉田・牧之内のアルゴリズムと名付けた．図 4 は，吉田・牧之内のアルゴリズムで，"to be computed"の値を計算している時に値が保存されているメモリ領域を示している．

```
procedure ComputeScore(
  input : A[1..M], B[1..N]
  output : RouteTable[1..M, 1..N])
{
  D[0..N], F[0..N] : vectors;
  c, d, e, i, j : scalars;

  D[0] = 0;
  F[0] = 0;
  for j = 1 to N do
  {
    D[j] = ALPHA + BETA*(j-1);
    F[j] = ALPHA + BETA*(j-1);
  }
  for i = 1 to M do
  {
    c = D[0];
    D[0] = ALPHA + BETA*(i-1);
    e = ALPHA + BETA*(i-1);
    for j = 1 to N do
    {
      e = max{ e-BETA, D[j]-ALPHA };
      F[j] = max{ F[j]-BETA, D[j]-ALPHA };
      d = max{ c+Score(A[i], B[j]), e, F[j] };
      switch d {
        case c+score(A[i], B[j]) :
                  save the symbol "no in/del"
                                into RouteTable[i,j];
        case e :  save the symbol "insert"
                                into RouteTable[i,j];
        case F[j] :  save the symbol "delete"
                                into RouteTable[i,j];
      }
      c = D[j];
      D[j] = d;
    }
  }
}
```

図 5 最適スコアを計算する関数
Fig. 5 Procedure computing the optimal score

図5は2つの配列を比較した時の最適スコアを計算する関数 ComputeScore のプログラムコードを示している．この関数では，アフィンギャップペナルティを用いている．ALPHA は開始ギャップペナルティ，BETA は伸長ギャップペナルティを意味する．A と B は入力文字配列を意味する．この関数では，動的計画法を用いて最適スコアを計算している．この計算の間に，2次元配列 RouteTable に計算の経路を保存している．

```
procedure TraceRoute(
  input : A[1..M], B[1..N],
          RouteTable[1..M, 1..N]
  output : TraceResult[1..ResultSize])
{
  i, j : scalar

  i = M;
  j = N;
  ResultSize = 0;

  while  (i >= 1) and (j >= 1)
  {
    switch RouteTable[i, j]
    {
      case the symbol "no in/del" :
      {
         save the symbol "no in/del"
                 into TraceResult[ResultSize];
         i--;
         j--;
      }
      case the symbol "insert" :
      {
         save the symbol "insert"
                 into TraceResult[ResultSize];
         j--;
      }
      case the symbol "delete" :
      {
         save the symbol "delete"
                 into TraceResult[ResultSize];
         i--;
      }
    }
    ResultSize++;
  }
}
```

図6 最適なアラインメントとなる計算の経路をたどる関数
Fig. 6 Procedure tracing the optimal path

図6は2つの入力配列の最適なアラインメントとなる計算の経路をたどる関数 TraceRoute のプログラムコードを示している．この関数では，最適なスコアが求まるように，2次元配列 RouteTable 上の計算の経路をたどる．そして，その結果を1次元配列 TraceResult に保存する．ResultSize は TraceResult の長さである．

```
mainProcedure YoshidaMakinouchi(
  input : A[1..M], B[1..N]
  output : AlignmentResult)
{
  /* Compute the optimal score. */
  ComputeScore(A[1..M], B[1..N]);

  /* Trace the array "RouteTable". */
  TraceRoute(A[1..M], B[1..N],
             RouteTable[1..M, 1..N]);

  /* Align sequences with the vector "TraceResult". */
  Alignment(A[1..M], B[1..N],
            TraceResult[1..ResultSize]);
}
```

図7 吉田・牧之内のアルゴリズム
Fig. 7 Yoshida-Makinouchi's algorithm

図7は吉田・牧之内のアルゴリズムのメイン関数のプログラムコードである．この関数では，関数 ComputeScore で最適スコアを計算し，関数 TraceRoute で最適なアラインメントとなる計算の経路を求め，その結果を用いて関数 Alignment で最適なアラインメントを求める．

4. 性能評価

4.1 実験環境

本研究で使用したプログラムは，すべてプログラミング言語 C を用いて記述している．性能評価実験は，OS に FreeBSD4.9-release，クロック周波数 400MHz，実メモリ 256MB を搭載したデスクトップ PC を使用した．入力配列として，11 個の長さが異なるタンパク質の文字配列を用意した．それらは配列データベース SwissProt で公開されているものである[10]．

第一の実験では，後藤のアルゴリズム，Myers と Miller のアルゴリズム，そして吉田・牧之内のアルゴリズムの性能を，グローバルアラインメントを実行することで比較した．第二の実験では，後藤のアルゴリズム，Myers と Miller のアルゴリズム，そして吉田・牧之内のアルゴリズムを用いて作った3つのローカルアラインメントのプログラムの性能を比較した．Myers と Miller のアルゴリズムを基にしたローカルアラインメントのプログラムは，Huang らの論文に基づいて作成した．残りの2つのアルゴリズムを基にしたローカルアラインメントのプログラムは，Smith らの論文に基づいて作成した．

4.2 グローバルアラインメント

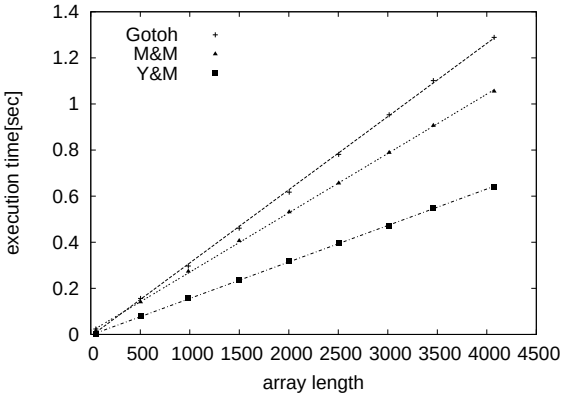

図 8 グローバルアラインメントの実行時間
Fig. 8 The execution time of global alignemt

図 8 は 3 つのアルゴリズムによるグローバルアラインメントの実行時間を比較したものである．"Gotoh" が後藤のアルゴリズム，"M&M"が Myers と Miller のアルゴリズム，"Y&M"が吉田・牧之内のアルゴリズムである．今回の実験では，2 つの入力配列のうち，一方の配列は長さが 393 の配列に固定し，もう一方の配列には長さが 51 から 4074 までの 9 本の配列を用いた．図 8 から，どのアルゴリズムの実行時間も入力配列の長さに比例することがわかる．その中でも，吉田・牧之内のアルゴリズムは，残りの 2 つのアルゴリズムより高速なアルゴリズムであることがわかる．

4.3 ローカルアラインメント

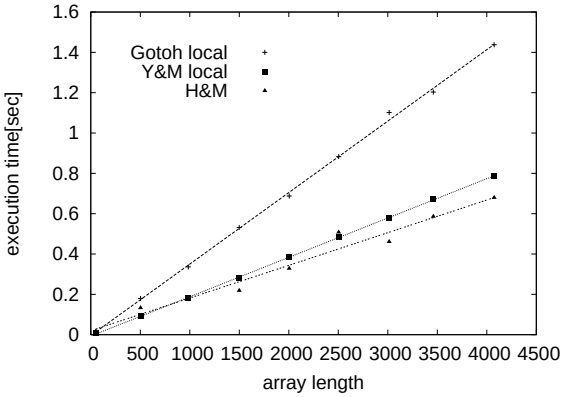

図 9 ローカルアラインメントの実行時間 (1)
Fig. 9 The execution time of local alignemt(1)

図 9 と図 10 は，3 つのアルゴリズムによるローカルアラインメントの実行時間を比較したものである．"Gotoh local"は後藤のアルゴリズムを用いた

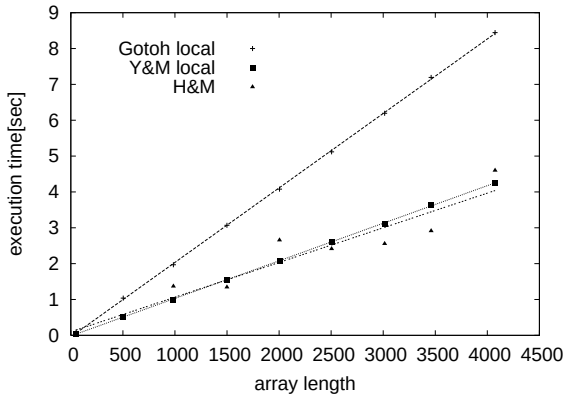

図 10 ローカルアラインメントの実行時間 (2)
Fig. 10 The execution time of local alignemt(2)

Smith らのローカルアラインメントのアルゴリズム，"H&M" は Myers と Miller のアルゴリズムを用いた Huang らのローカルアラインメントのアルゴリズム，そして"Y&M local"は吉田・牧之内のアルゴリズムを用いた Smith らのローカルアラインメントのアルゴリズムである．図 9 では，2 つの入力配列のうち一方の長さを 393 に固定し，図 10 では長さを 2142 に固定している．図 9，図 10 とももう一方の入力配列には長さが 51 から 4074 までの 9 本の配列を用いた．Huang らのアルゴリズムは他の 2 つのアルゴリズムより実行時間が短い．しかし，一部では吉田・牧之内のアルゴリズムの方が，Huang らのアルゴリズムよりも実行時間が短くなっていることがわかる．これは，ローカルアラインメントの対象となる部分の長さの違いからくると考えられる．ローカルアラインメントの対象となる部分の長さは，2 つの入力配列の相同性の高さに依存する．平均すると，Huang らのアルゴリズムと吉田・牧之内のアルゴリズムの間に差はない．両者とも後藤のアルゴリズムの性能を上回っている．

5. おわりに

吉田・牧之内のアルゴリズムは，グローバルアラインメントについては後藤のアルゴリズムと Myers と Miller のアルゴリズムの性能を上回った．吉田・牧之内のアルゴリズムは後藤のアルゴリズムよりメモリ領域を必要としない．しかし，Myers と Miller のアルゴリズムよりも大きなメモリ領域を必要とする．現在ではメモリの価格は以前よりも安くなっている．そのため，このことはあまり重要な問題ではない．もし 256MB 以上のメモリを搭載した計算機を使っていれば，吉田・牧之内のアルゴリズムを実行することがで

きる．ローカルアラインメントについても，吉田・牧之内のアルゴリズムはよい性能を示した．

謝辞 本論文の執筆に当たり，意見・助言をいただいた九州大学大学院システム情報科学研究院の金子邦彦助教授，同大学大学院システム生命科学府の稲田稔さん，同大学大学院システム情報科学府の才野大輔さんに感謝致します．

参 考 文 献

1) S.Needlman and C.Wunsch, “A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins”, hJournal of Molecular Biology, 48:pp.443-453, 1970.
2) D.S.Hirschberg, “A Linear Space Algorithm for Computing Maximal Common Subsequences”, Communications of ACM, 18(6):pp.341-343, 1975.
3) T.Smith and M.Waterman. “Identification of Common Molecular Subsequences”, Journal of Molecular Biology, 147:pp.195-197, 1981.
4) O.Gotoh, “An Improved Algorithm for Matching Biological Sequences”, Journal of Molecular Biology, 162:pp.705-708, 1982.
5) E.Myers and W.Miller, “Optimal alignments in linear space”, Computer Applications in the Biosciences (CABIOS), 4(1):pp.11-17, 1988.
6) X.Huang, R.C.Hardison and W.Miller, “A space-efficient algorithm for local similarities”, Computer Application is the Biosciences (CABIOS), 6(4):pp.373-381, 1990.
7) K.Charter, J.Schaeffer and D.Szafron, “Sequence Alignment using FastLSA”, In Proceedings of The 2000 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS’2000), pp.239-245, 2000.
8) A.Davidson, “A Fast Pruning Algorithm for Optimal Sequence Alignment”, In Proceedings of 2nd IEEE International Symposium on Bioinformatics and Bioengineering (BIBE’01), pp.49-56, 2001.
9) K.Charter, J.Schaeffer and D.Szafron, “FastLSA : A Fast, Linear-Space, Parallel and Sequential Algorithm for Sequence Alignment”, In Proceedings of 2003 International Conference on Parallel Processing, pp.48-57, 2003.
10) Swiss Prot, http://kr.expasy.org/sprot/