LF-009

履歴記憶反復深化A*探索 –マルチプルアライメントへの適用–
Memory History IDA* – Applying to Multiple Sequence Alignment –

村田 裕章 * 越野 亮 * 木村 春彦 †
Hiroaki Murata Makoto Koshino Haruhiko Kimura

1. はじめに

タンパク質のアミノ酸配列を解析するとき，まず最初に配列を観察しやすいようにアミノ酸の順序を変えずに，アミノ酸とアミノ酸の間にギャップと呼ばれるものを挿入し配列の並べ換えを行う．この操作をアライメントと言い，特にN本のタンパク質配列を一度にアライメントする方法をマルチプルアライメントと言う．マルチプルアライメントの結果から保存領域が同定され，未知の機能や構造を既知情報から予測することが可能となる．

アライメントの操作は生物のアミノ酸配列を解析する際，一番最初に行うため，最も重要な操作と言われている．したがって，アライメントの結果次第では解析結果も非常に異なった結果となることがある．そのため，遺伝子を統計的にあるいは情報論的に取り扱うときには，アライメント結果が厳密でないと解析結果の信頼性が問われることになる．

マルチプルアライメントは最短経路問題に定式化でき，池田等 [2] は人工知能の効率的な最適解探索アルゴリズムであるA*アルゴリズムをマルチプルアライメントに適用した．また，ヒューリスティック値に重みを付け近似解を効率的に求める手法も提案している．しかしながら，依然として膨大な計算時間とメモリが必要である．

一方，線形記憶量アルゴリズムであるIDA*（反復深化A*探索）では再訪を回避することができず，現実的な時間では4配列までしか解くことができなかった．その後，三浦等はIDA*において，再訪を回避するため生成された状態を確率的に記憶する確率的記憶節点方式 (SNC) を提案し，7配列まで解くことに成功した [4]．

本論文では，A*の$CLOSED$リストをIDA*に導入することで，効率的に再訪を回避する履歴記憶反復深化A*探索（MHIDA*:Memory History IDA*）を提案し，A*探索に比べて速度向上が見られたことを示す．

2. マルチプルアライメントの定式化

タンパク質の構成要素であるアミノ酸には20種類あり，それらを表現するためにそれぞれ異なるアルファベットが割り当てられている．マルチプルアライメントは，複数の配列の類似する部分を縦に揃えて並べる操作である．例として，ヒト，ウサギ，ジャガイモのアミノ酸配列のマルチプルアライメントを図1に示す．

d本のマルチプルアライメントはd次元の束上の最短経路を求める問題として次のように定式化される [1]．整列させる配列の本数をd本，それぞれの配列をS_k，k番目の配列の長さをL_kとする．長さL_kのd本の配列を，d次元空間の束$L(S_1, \cdots, S_d)$に対応させる．この束はd本の配列の直積を作ることで得られる．問題空間はd次元の束として表すことができる．各節点は状態と呼ばれ，特に全ての配列の始めの文字に対応する状態を初期状態v_0，全ての配列の終りの文字に対応する状態を目標状態Gと呼ぶ．初期状態と目標状態を結ぶ経路は配列を整列させた結果に対応する．図2は図1で示したアライメントの問題空間を表す．なお，太線の経路はアライメント後の配列に対応している．

<アライメント前>
GDVEKGKKIFIMKCSQCH (ヒト)
GDAERGKKLFESRAAQCH (ウサギ)
ASFNEAPPGNPKAGEKIFKTKCACH (ジャガイモ)
<アライメント後>
-G--DV---E-K-GKKIFIMKCSQCH
-G--DA---E-R-GKKLFESRAAQCH
ASFNEAPPGNPKAGEKIFKTKCA-CH

図 1: 3本のマルチプルアライメントの例

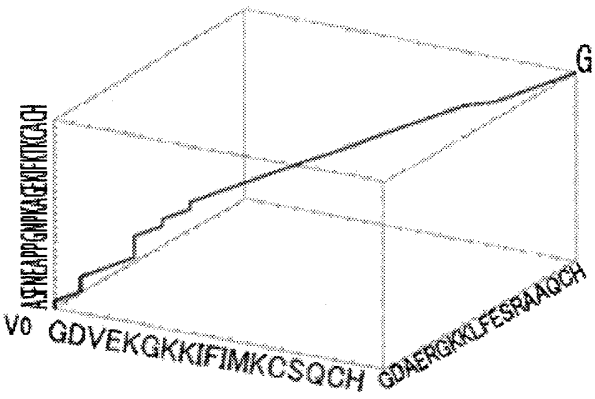

図 2: 3本のマルチプルアライメントの問題空間

3. 従来手法

3.1 A*アルゴリズム

A*アルゴリズムでは状態を展開することの好ましさを評価する評価関数が与えられ，最も良さそうな状態を優先的に探索する手法である．状態vの評価関数を$f(v) = g(v) + h(v)$で表す．ただし，$g(v)$は経路コストを呼ばれ，初期状態v_0から状態vまでのコストの和を表し，$h(v)$ヒューリスティックと呼ばれ，状態sから目標状態Gまでの経路コストの推測値を表す．A*アルゴリズムでは，$OPEN$と$CLOSED$と呼ばれる2つのリストを用いる．生成された状態は一旦$OPEN$リストに保存され，展開された状態は$CLOSED$リストに移される．したがって，それらのリストをチェックすることでA*アルゴリズムは状態の再生成を避けることができる．

*石川工業高等専門学校, INCT
†金沢大学, Kanazawa University

マルチプルアライメントにおけるヒューリスティック探索では，経路コストとしてその経路が表している配列のスコアを用いる．配列のスコアはアミノ酸の類似評価尺度を用いて計算され，類似評価尺度として PAM が使われることが多い‡．一般的に PAM には負の数や零が含まれるため，それらの和であるスコアも負の数になる可能性がある．しかし，A*アルゴリズムは，コストを全て正としてコストの合計が最小になる経路を求める手法である．そのため，コストの正負を反転させて，全て 1 以上になるようにコストの “かさ上げ” を行う．マルチプルアライメントにおいて，目標状態 G までの経路コストの推測値であるヒューリスティックは次のように与えられる [2]．k 番目の配列 S_k を $S_k = s_{k1}, s_{k2}, ..., s_{kL_k}$ (配列長 L_k) とするとき，状態 $v = (v_1, ..., v_d)$ におけるヒューリスティック $h(v)$ を以下のようになる．ただし，$h^*(v_i, v_j)$ は i 番目と j 番目の配列間における最適なヒューリスティック値であり，二次元の動的計画法を用いて求める．

$$h(v) = \sum_{i,j=1, i \neq j}^{d} h^*(v_i, v_j)$$

3.2 反復深化 A*探索

メモリ量を線形オーダーに抑える手法として IDA*（反復深化 A*探索）がある．この手法は深さ優先探索において，探索する深さに制限を設け，解が見つからなければ制限を緩和し，さらに深い場所にある状態を探索する手法である．しかし，現在探索している経路上の状態しか記憶しないため，同じ状態を再び展開する再訪を回避することができず，マルチプルアライメントでは探索に膨大な計算時間を要する．これは，マルチプルアライメントの問題空間が束上の構造をしているため，同じ状態への経路が無数に存在するためである．図 3 に IDA*のアルゴリズムを示す．なお，本論文では従来の IDA*のアルゴリズムとは違い，再帰処理を使わず繰り返し処理を用いるなど，提案手法と比較しやすいようにアルゴリズムを変更してある．説明に際して以下の記号を用いる．
$OPEN$：生成された状態を記憶するリスト．再帰処理を繰り返し処理で行うためのリスト．
$TEMP$：生成された状態を一時的に記憶するリスト．
θ：深さ優先探索における深さ制限値．
θ_2：次の繰り返しで用いられる深さ制限値．

3.3 MREC

IDA*において再訪を回避する手法として MREC[5] がある．MREC は可能な限り状態を生成順に記憶する手法であり，IDA*を一般化した手法である．メモリ内に記憶できる最大の状態数を記憶可能状態数§ $M = 0$ のとき MREC は IDA*と完全に一致する．MREC は再訪を回避するために状態 v をそのヒューリスティック値 $h(v)$ と共に記憶する．また，全ての子状態 v' が展開されると，状態 v のヒューリスティック値 $h(v)$ は $min\{c(v,v')+h(v')\}$ に更新される．これにより，より正確なヒューリスティック値を求めることができ，再訪を回避することができる．図 4 に MREC のアルゴリズムを示す．なお，N は記憶している状態数を表している．

‡本論文では，PAM250 をアミノ酸の類似評価尺度として用いる．
§[5] では最大記憶節点数と呼んでいるが，「記憶された最大の状態数」と区別するために記憶可能状態数と呼ぶ．

```
1.  Search(v0)
2.    θ = h(v0)
3.    repeat 目標状態が見つかるまで
4.      OPEN に初期状態を入れる.
5.      θ=IDA*Search(v0, θ)
6.    end repeat
7.  IDA*Search(v, θ)
8.    θ2 = ∞
9.    if v が目標状態なら then 終了
10.   repeat OPEN ≠ φ の間
11.     OPEN の末尾を取り出し，v とする.
12.     for v のそれぞれの子状態 v' に対して do
13.       f(v') = g(v) + c(v,v') + h(v') を計算し，TEMP
14.       に入れる．ただし，次の場合は TEMP に入れない.
15.       (1)f(v') > θ
16.       if θ < f(v') < θ2 then θ2 = f(v')
17.     end do
18.     TEMP を OPEN に追加する.
19.     TEMP を空にする.
20.   end repeat
21.   return θ2
```

図 3: IDA*アルゴリズム

3.4 関連研究

MREC の改良手法として状態を確率的に記憶する SNC[4] があるが，メモリに制限がない場合は A*と同じ動作をする．また，A*の改良手法として段階的節点展開法 (PEA* : Partial Expansion A*)[6] がある．この手法は，A*において計算時間を多少増加させるかわりに，大幅にメモリ量を削減する手法である．

4. 提案手法：履歴記憶反復深化 A*探索

本論文では，IDA*において効率的に再訪を回避するために，A*で用いる $CLOSED$ リストを導入する．この手法を履歴記憶反復深化 A*探索（MHIDA*:Memory History IDA*）と呼ぶ．図 5 に MHIDA*のアルゴリズムを示す．

MHIDA*では IDA*のアルゴリズムに 6 行目，13 行目，18 行目，21 行目の前半部の 4 つの処理を追加した．MHIDA*では A*探索と同様に，展開された状態とその状態の評価値を $CLOSED$ リストに保存する（**13** 行目）．状態を生成する度に $CLOSED$ リスト内を調べ，新しく生成された状態の評価値が，$CLOSED$ リスト内にある状態の評価値より悪い（大きい）場合は，その状態を展開しないようにしている（**18** 行目）．これは，同じ状態のヒューリスティック値は同じであることから，新しく見つかった経路が，それ以前に発見されている最短経路より短い場合のみ，再び展開しているとも言える．MHIDA*はこのような条件を設けることで A*と同様に最適解を求めることができる．

```
1.  Search(v0)
2.    θ = h(v0)
3.    repeat 目標状態が見つかるまで
4.      OPEN に初期状態を入れる.
5.      θ=MRECSearch(v0,θ)
6.    end repeat
7.  MRECSearch(v,θ)
8.    θ2 = ∞
9.    if v が目標状態なら then 終了
10.   repeat OPEN ≠ φ の間
11.     OPEN の末尾を取り出し，v とする.
12.     for v のそれぞれの子状態 v' に対して do
13.       f(v') = g(v) + c(v,v') + h(v') を計算し，TEMP
14.       に入れる．ただし，次の場合は TEMP に入れない.
15.       (1)f(v') > θ
16.       if θ < f(v') < θ2 then θ2 = f(v')
17.       if N < M then v'を h(v') と共に記憶する.
18.     end do
19.     TEMP を OPEN に追加する.
20.     TEMP を空にする.
21.     h(v) を min{c(v,v') + g(v')} に更新する.
22.   end repeat
23.   return θ2
```

図 4: MREC アルゴリズム

```
1.  Search(v0)
2.    θ = h(v0)
3.    repeat 目標状態が見つかるまで
4.      OPEN に初期状態を入れる.
5.      θ=MHIDA*Search(v0,θ)
6.      CLOSED を空にする.
7.    end repeat
8.  MHIDA*Search(v,θ)
9.    θ2 = ∞
10.   if v が目標状態なら then 終了
11.   repeat OPEN ≠ φ の間
12.     LIST の末尾を取り出し，v とする.
13.     v を f(v) とともに CLOSED に記憶する.
14.     for v のそれぞれの子状態 v' に対して do
15.       f(v') = g(v) + c(v,v') + h(v') を計算し，TEMP
16.       に入れる．ただし，次の場合は TEMP に入れない.
17.       (1)f(v') > θ
18.       (2)v' が CLOSED 内にあり，新しい f(v') の方が悪い
19.       if θ < f(v') < θ2 then θ2 = f(v')
20.     end do
21.     TEMP を降順にソートし，OPEN に追加する.
22.     TEMP を空にする.
23.   end repeat
24.   return θ2
```

図 5: MHIDA*のアルゴリズム

ところで，MHIDA*も IDA*と同様に反復深化アルゴリズムであるため，ある深さ制限の下で探索を行い，解が見つからない場合は，深さ制限値を大きくして再び初期状態から探索を始める．このとき，前回の深さ制限値の下で探索した状態が $CLOSED$ リスト内に残っていると初期状態から生成される状態は，すべて展開されず探索は失敗する．そのため，深さ制限値が更新される度に $CLOSED$ リストを空にする必要がある（**6** 行目）．また IDA*では生成された順に状態を展開しているが，MHIDA*では評価値の良い状態から展開している（**21** 行目の前半部）．そのため，IDA*より早く目標状態に辿り着くことができる．

MHIDA*が MREC と異なる点は次の 2 つである．

(1) メモリの使用方法

MREC では状態とヒューリスティック値が記憶され，ヒューリスティック値の計算時に用いられる．また，全ての子状態が生成されたときに子状態のヒューリスティック値を用いて親状態のヒューリスティック値を更新している．これにより，より正確なヒューリスティック値を得ることができるため，無駄な探索が抑えられ再訪を回避することができる．これに対し，MHIDA*は状態と評価値を記憶し，状態が生成されるときに展開済みでないか調べるために用いられる．生成された状態が展開済みの場合はその状態を再び展開しないようにすることで無駄に何度も展開する状態を減らし，再訪を回避を回避している．しかし，最初に展開された経路がその状態までの最短経路である保障はないため，一度しか展開しないようにすると最適性は失われる．そのため展開済みであっても，経路が以前に展開されたときより良い場合はその状態を展開する．これにより MHIDA*は最適性を保ち，それ故に評価値も一緒に記憶する．なお，ヒューリスティック値は状態にのみ依存するため同じ状態において評価値を比較することは，経路のコストを比較することと同値となる．

(2) 展開順序

MREC は生成された順番に状態を展開する．これに対し MHIDA*は評価値の良い状態から展開している．これにより MHIDA*は無駄な探索を省くことができる．

5. 性能評価実験

提案手法である MHIDA*の有用性を確認するために性能評価実験として，[2] で用いられている 10 本のアミノ酸配列を用いて 4 本から 8 本のマルチプルアライメントを行った．比較対象は IDA*と A*，MREC とし，PEA*と SNC については比較実験を行わない．なぜなら，PEA*は A*と比較し，メモリ量を減らすことができるが計算時間は大きくなると報告されているためである [6]．また本論文では高速化を目的としているために，記憶可能状態数 $M = \infty$ とする．このとき，SNC が記憶している状態数は記憶可能状態数に達することがないため MREC の効率を上回ることができない [4]．そのため，MREC との比較で十分であると考えられるためである．

各手法の実行時間，最大記憶状態数，ステップ数，バックトラック回数を表 1 に示す．記憶状態数とは $OPEN$

リスト内の状態数と *CLOSED* リスト内の状態数の和であり，最大記憶状態数とは記憶状態数の最大値である．ただし，MREC においては，*CLOSED* リスト内の状態数を，状態とそのヒューリスティック値が記憶されているリストの数とする．なお，解けなかったところは – で表してある．

$d = 7$ において MHIDA*は A*に比べ，ステップ数は約 300 倍多いが，実行時間では約 5.5 倍速くなっている．これは MHIDA*が 1 ステップに要する時間が，A*に比べて少ないためだと考えられる．A*は *OPEN* リスト内で評価値の最も良い状態を次のステップで展開する．そのため，*OPEN* リスト内の状態数が多くなると展開する状態を探す時間が多くなる．これに対し，MHIDA*では新しく生成された状態の中で評価値の最も良い状態を展開する．新しく生成される状態数は *OPEN* リスト内の状態数に比べると非常に少ないため，展開する状態を探す時間は A*より少ない¶．このことから，MHIDA*が 1 ステップに要する時間は A*に比べて少ないため，ステップ数が多くなっても，実行時間は少なくなったと考えられる．表 2 に A*と MREC，MHIDA*の 1 ステップあたりの時間を示す．$d = 7$ において，A*は MHIDA*の約 1600 倍の時間を 1 ステップに要していることが分かる．また，最大記憶状態数については，$d = 7$ において MHIDA*は A*の約 3.5%となり，大幅にメモリ量を削減されていることが分かる．

次に IDA*と比較すると，$d = 4$ において実行時間では約 6200 倍速くなり，ステップ数も 7500 倍少なくなっている．これは，MHIDA*が効率的に再訪を回避できるためだと考えられる．一方，最大記憶状態数は，約 1.4 倍多くなっている．これは，*CLOSED* リストに展開した状態を記憶するためだと考えられる．

MREC と比較すると，$d = 6$ において実行時間では約 75 倍速くなったが，最大記憶状態数は約 10%増加した．MHIDA*の実行時間が速くなった要因として次の二つが考えられる．一つ目は状態の展開順番である．MREC は生成順に状態を展開するが，MHIDA*は評価値の良い状態から展開することである．これにより無駄な探索を省くことができる．二つ目は再訪の回避方法である．MREC は，メモリを用いることで，より正確なヒューリスティック値を得ることができ，再訪を回避している．これに対し，MHIDA*は展開済みであるか調べるために用いて，同じ状態を展開しないようにすることで再訪を回避している．これにより，無駄な再訪だけでなく，再展開も防ぐことができる．なお，MHIDA*のステップ数が MREC のステップ数より少なくなる要因として，これらを挙げることができる．次に，最大記憶状態数が増加した要因について考える．MHIDA*は評価値の良い状態から展開するために，バックトラックが MREC に比べあまり発生しない．そのため，OPENList には一度の多くの状態が記憶されてしまい，最大記憶状態数は MREC より多くなると考えられる．

最後に MHIDA*は MREC より多くのメモリ量を使用しているにもかかわらず，8 本のマルチプルアライメントを解くことができた．これにより，MHIDA*が効率的に再訪を回避することができ，マルチプルアライメントに有効な手法であることが確認された．

¶全ての手法においてクイックソートを用いて展開する状態を探している．

表 1: マルチプルアライメントの結果

	d	4	5	6	7	8
Score		7691	12287	18074	24880	33092
Time[s]	IDA*	6061.75	-	-	-	-
	A*	1.39	21.73	154.12	5754.96	-
	MREC	92.71	3594.18	6149.32	-	-
	MHIDA*	0.98	19.10	82.07	1037.59	20388.9
# of states	IDA*	462	-	-	-	-
	A*	7484	19218	46845	213280	-
	MREC	587	1103	1621	-	-
	MHIDA*	656	1207	1781	7467	44906
# of steps	IDA*	126156610	-	-	-	-
	A*	591	1081	1568	6549	-
	MREC	1293703	24957355	19838451	-	-
	MHIDA*	16746	166095	345272	1920324	10862806
# of back track	IDA*	4388478	-	-	-	-
	MREC	46409	1027822	1832902	-	-
	MHIDA*	1777	28638	105681	900966	7590773

表 2: 1 ステップあたりにかかった時間 [ms]

d	4	5	6	7
A*	2.35	20.10	98.29	878.75
MREC	0.07	0.14	0.31	-
MHIDA*	0.09	0.17	0.35	0.80

6. まとめ

IDA*に A*の *CLOSED* リストを導入にすることで，効率的に再訪を回避する手法として MHIDA*を提案した．MHIDA*では状態の生成時に *CLOSED* リストを調べ，同じ状態を再び展開しないようにしている．しかし，最適解を求めるために，より良い経路が見つかった場合は再び展開するようにしている．性能評価実験として行ったマルチプルアライメントでは，$d = 7$ において，A*の約 5.5 倍の速度向上がみられ，最大記憶節点数は約 3.5%に抑えられた．IDA*と比較すると $d = 4$ において，約 6200 倍の速度向上がみられ，MREC との比較では $d = 7$ において，約 75 倍の速度向上がみられた．また，SNC では解けなかった 8 本のマルチプルアライメントが解けたことを示し，マルチプルアライメントでの MHIDA*の有用性が明らかにした．

参考文献

[1] H.Carrillo and D.Lipman, "The multiple sequence alignment problem in biology". SIAM Journal Applied Mathematic, vol.48, pp.1073–1083, 1988.

[2] T.Ikeda and H.Imai,"Enhanced A* algorithms for multiple alignmen: optimal alignments for several sequence and k-opt approximate alignment for large cases", Theoritical Computer Science, vol.210, pp.341–374, 1999.

[3] R.E.Korf, "Linear-space best-first search", Artificial Intelligence, vol.62, pp.41–78, 1993.

[4] 三浦 輝久, 石田 亨, "記憶制約下における探索のための確率的節点記憶方式", 電子情報通信学会論文誌, vol.J80-D-II, No. 9, pp.2438–2445, 1997.

[5] A.K.Sen and A.Begchi, "Fast Recursive formulations for best-first search that allow controlled use of memory", IJCAI–89, pp.297–302,1989

[6] T.Yoshizumi, T.Miura and T.Ishida ,"A* with Partial Expansion for large branching factor problems", AAAI–2000, pp.923–929, 2000.