3H1-02

改良重み付き反復深化A*アルゴリズムのマルチプルアライメントへの適用

Appied Improved Weighted Iterative Deepening A* Algorithm to Multiple Sequence Alignment

村田　裕章[*1]　　越野　亮[*1]
Hiroaki Murata　　Makoto Koshino

[*1]石川工業高等専門学校
Ishikawa National College of Technology

The mulitiple sequence alignment can be formulated as the shortest path problem and heuristic search as A* can be applied to this problem. A* and IDA* are applied to this problem in previous work. Unfortunately, A* requires a lot of computation time and memory. IDA* could align only four sequence at most. After that, Stochastic Node Caching(SNC) and A* with Partial Expansion(PEA*) are proposed. SNC, which improved IDA*, caches nodes selectively based on fix probabilliy for reducing the number of revisits. SNC was able to align seven sequence. PEA*, which improved A*, is to store only necessary node for finding an optimal solusion, PEA* was able to align eight sequence. In this paper, we propose two algorithms for mulitiple sequence alignment. First algorithm is called I–IDA*, which is introduced $CLOSED$ list to IDA*. $CLOSED$ list can reduce the number of revisits in IDA*. We show that I–IDA* can align seven sequence 5.5 times as fast as A* and uses only 3.5% of the stored node required A*. Second algorithm is called I–WIDA*, which is based on Weighted IDA*. We also show that I–WIDA* can align nine sequence 2800 times as fast as A*.

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対応させ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．たとえば，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．

従来研究として，池田等 [Ikeda 99] 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*）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 [三浦 97].

本論文では，A*の$CLOSED$ リストを IDA*に導入することで，再訪を回避する改良反復深化 A*探索（I–IDA*:Improved–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．また，I–IDA*において評価関数に重みを導入した改良重み付き反復深化 A*探索 (I–WIDA*：Improved – Weighted IDA*) を提案し，重み付き A*探索（WA*）に比べて大幅な速度向上が見られたことを示す．

連絡先: 越野　亮，石川工業高等専門学校，石川県河北郡津幡町字北中条タ 1，TEL:076-288-8132，FAX:076-288-8146，e-mail:koshino@shikawa-nct.ac.jp

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．以下にヒト，ウサギ，ジャガイモのアミノ酸配列のアライメントを示す．

```
        <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
```

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 [Carrillo 88]．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と結ぶ経路は配列を整列させた結果に対応する．図 1 は上で示したアライメントの問題空間を表す．なお，太線はアライメント後の配列に対応している．

3. 従来手法

3.1 最適解探索手法：A*アルゴリズム

A*探索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

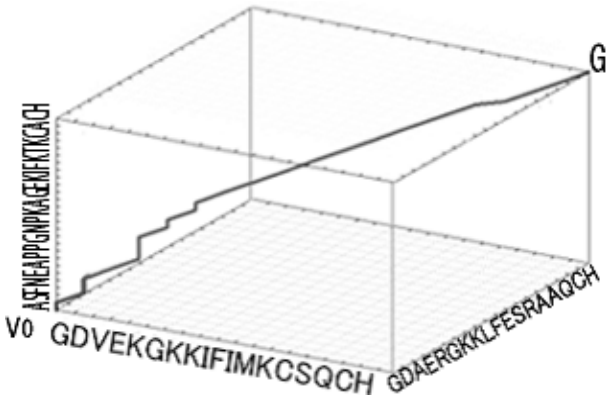

図 1: 3本のマルチプルアライメントの問題空間

ある．状態 v の評価関数を $f(v) = g(v)+h(v)$ で表す．ただし，$g(v)$ は初期状態 v_0 から状態 v までの経路コストの和 $c(v_0, v)$ であり，$h(v)$ は各状態 s から目標状態 G までのコストの推測値である．A*アルゴリズムでは，$OPEN, CLOSED$ 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* は状態の再生成を避けることができる．

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[*1]．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，コストは全て正とし，コストの合計が最小になる経路を求める手法であるため，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 [Ikeda 99]．k 番目の配列 S_k を $S_k = s_{k1}, s_{k2}, ..., s_{kL_k}$ とするとき配列長 (L_k)，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 最適解探索手法：反復深化 A*アルゴリズム

メモリ量を線形オーダーに抑える手法として反復深化 A*アルゴリズム (IDA*) 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 2 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．

```
1.  Search(v_0)
2.    θ = h(v_0)
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(v_0, θ)
6.    end repeat
7.  IDA*Search(v, θ)
8.    θ_2 = ∞
9.    if v が目標状態なら then 終了
10.   repeat OPEN ≠ ø 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 に入れる.
14.       ただし，次の場合は TEMP に入れない.
15.       (1)f(v') > θ
16.       if θ < f(v') < θ_2 then θ_2 = 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 θ_2
```

図 2: IDA*アルゴリズム

$OPEN$：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 最適解探索手法：関連研究

IDA*において再訪を回避する手法として MREC[Sen 89] がある．使用するメモリ量に制限を設け，メモリがある間は A*探索を行い，メモリがなくなると，$OPEN$ リスト内の状態を初期状態として IDA*を行う手法である．使用できるメモリがない場合は IDA*と同じ動作となり，メモリに制限がない場合は A*と同じ動作をする．MREC の改良手法として SNC[三浦 97] 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，段階的節点展開法 (PEA* : Partial Expansion A*)[Yoshizumi 00] 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．

本論文では，マルチプルアライメントを高速に解くことを目的とするためメモリ量に制限を加えない．そのため，MREC と SNC は A*と同じ動作となる．また，PEA*は A*より計算時間が多い．したがって，MREC，SNC，PEA*は提案手法との比較を行わない．

3.4 近似解探索手法

最適解探索手法において，評価関数に重み k (≥ 1) を導入し，状態 v の評価関数を $f(v) = g(v)+k\cdot h(v)$ にすることで，最適解を求める保証はなくなるが高速に近似解を求めることができる．A*に重み (Weight) を付けた手法を WA*（重み付き A*探索）[*2]，IDA*に重みを付けた手法を WIDA*（重み付き IDA*）[Korf 93] と呼ぶ．

*1 本論文では，PAM250 をアミノ酸の類似評価尺度として用いる．

*2 [Ikeda 99] では，マルチプルアライメントにおいて，A*に重みを導入した手法を A アルゴリズムと呼んでいるが，ここでは WA*と呼ぶ．

```
1.  Search(v_0)
2.    θ = h(v_0)
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=I–IDA*Search(v_0, θ)
6.      CLOSED を空にする.
7.    end repeat
8.  I–IDA*Search(v, θ)
9.    θ_2 = ∞
10.   if v が目標状態なら then 終了
11.   repeat OPEN ≠ ø 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 に入れる.
16.       ただし，次の場合は TEMP に入れない.
17.       (1)f(v') > θ
18.       (2)v' が CLOSED 内に存在し，新しい f(v') の方が悪い
19.       if θ < f(v') < θ_2 then θ_2 = 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 θ_2
```

図 3: I–IDA*のアルゴリズム

4. 提案手法

4.1 I–IDA*（最適解探索手法）

本論文で提案する改良反復深化 A*探索（I–IDA*:Improved – IDA*）は効率的に再訪を回避できるように IDA*を改良した手法である．主な改良点は A*で用いられる $CLOSED$ リストを導入した点である．図 3 に I–IDA*のアルゴリズムを示す．I–IDA*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．

I–IDA*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 $CLOSED$ リストに保存し**（13 行目）**，状態が生成される度に $CLOSED$ 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，再び探索を行っているとも言える．ところで，I–IDA*は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，$CLOSED$ リスト内に，前回探索したときの状態が残っていると初期状態から生成された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．これに対し，I–IDA*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．

4.2 I–WIDA*（近似解探索手法）

次に，I–IDA*を近似解探索手法に改良した改良重み付き反復深化 A*探索（I–WIDA*:Improved – WIDA*）を提案する．この手法は，I–IDA*において評価関数に重み k を導入し，評価関数 $f(v)$ を $f(v) = g(v) + k \cdot h(v)$ とした手法である．k の値が 1 の場合，I–WIDA*は I–IDA*と同じである．この手法では，最適解を求める保証はないが，高速に近似解を求めることができる．

5. 性能評価実験

性能評価実験として [Ikeda 99] で用いられている 10 本のアミノ酸配列を用いて実験を行った．

5.1 最適解探索手法の比較実験

提案手法である I–IDA*の有用性を確認するために，IDA*, A*，I–IDA*の三つの手法を用いて 4 本から 7 本のマルチプルアライメントを行った．各手法の実行時間，最大記憶節点数，ステップ数を表 1 に示す．なお，IDA*においては 5 本以上のマルチプルアライメントは解くことができなかった．また，記憶節点数とは $OPEN$ リスト内の状態数と $CLOSED$ リスト内の状態数の和であり，最大記憶節点数とは記憶節点数の最大値である．$d = 7$ において I–IDA*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，I–IDA*の 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，I–IDA*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*3．このことから，I–IDA*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 I-IDA*の 1 ステップあたりの時間を示す．$d = 7$ において，A*は I–IDA*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，最大記憶節点数については，$d = 7$ において I–IDA*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．

表 1: マルチプルアライメント（最適解）の結果

	d	4	5	6	7
スコア		7691	12287	18074	24880
時間 [s]	IDA*	6061.75	-	-	-
	A*	1.39	21.73	154.12	5754.96
	I–IDA*	0.98	19.10	82.07	1037.59
節点数	IDA*	607	-	-	-
	A*	7484	19218	46845	213280
	I–IDA*	656	1207	1781	7467
Step	IDA*	126156610	-	-	-
	A*	591	1081	1568	6549
	I–IDA*	16746	166095	345272	1920324

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
I–IDA*	0.06	0.12	0.24	0.54

*3 全ての手法においてクイックソートを用いて状態を探している．

5.2 近似解探索手法の比較実験

次に，提案した近似解探索手法である I–WIDA*の有用性を確認するために，7 本以上のマルチプルアライメントを行った．各手法の実行時間，最大記憶節点数，ステップ数，スコア，バックトラックの回数を表 3 に示す．重み k は [Ikeda 99] と同様に 1.01 とした．なお，バックトラックは WA*では発生しない．$d = 9$ において，I–WIDA*は WIDA*に比べ，実行時間では約 36 倍速く，スコアも約 2.6%良くなっている．これは，WIDA*が生成された順番に状態を展開することに対し，I–WIDA*は評価値の良い状態から展開するためだと考えられる．I–WIDA*は評価値の良い状態から展開することで，WIDA*に比べ，バックトラックの回数が減り余計な経路を探索しない．これにより，ステップ数が減少し，実行時間が少なくと考えられる．また，評価値の良い状態から展開することで I–WIDA*は WIDA*に比べ，良い解を見つけることができたと考えられる．

一方，WA*と比べると I–WIDA*は実行時間では約 2800 倍速くなっているが，スコアは約 0.7%悪くなっている．これは，WA*が幅優先的に探索を行うことに対し，I–WIDA*が深さ優先的に探索を行うためだと考えられる．WA*は幅優先的に探索を行うために，1 ステップにかかる時間が I–WIDA*に比べ多い．さらに，I–WIDA*のステップ数は WA*に比べて少ないことから I–WIDA*の実行時間が少なくなったと考えられる．

表 3: マルチプルアライメント（近似解）の結果

	d	7	8	9
時間 [s]	WIDA*	0.58	57.45	106.69
	WA*	142.16	1051.54	8242.48
	I–WIDA*	0.49	1.15	2.98
節点数	WIDA*	564	1265	1959
	WA*	61009	142304	342115
	I–WIDA*	39893	82391	166940
Step	WIDA*	1250	61414	40740
	WA*	548	795	1193
	I–WIDA*	460	461	461
スコア	WIDA*	23892	31848	41595
	WA*	24869	33086	43000
	I–WIDA*	24851	33034	42690

5.3 重み k による I–WIDA*の性能の変化

次に，I–WIDA*において，重み k の値を変更することにより，どのくらい，実行時間やメモリ量が変化するか検証するために，7 本のマルチプルアライメントを用いて実験を行った．各手法の実行時間，最大記憶節点数，ステップ数，スコア，バックトラックの回数を表 4 に示す．なお，バックトラックは WA*では発生しない．k の値が大きくなるに従い，WIDA*と WA*の最大記憶節点数は減少しているが，I–WIDA*は増加している．これは，バックトラックが発生していないことと一度に生成される状態数が増加することが原因であると考えられる．展開されなかった状態は $OPEN$ リストに記憶されるため，バックトラックが発生せずに探索を行うと，$OPEN$ リスト内の状態数は多くなる．また，k の値に比例し深さ制限値も大きくなるため，一度に生成される状態数が増える．これより，$OPEN$ リスト内の状態数が増加するため，I–WIDA*の最大記憶節点数は増加すると考えられる．

また，実行時間においても，WIDA*と WA*においてはステップ数と同様に少なくなっているが，I–WIDA*ではステップ数とは逆に多くなっている．これは，ステップ数が減少しても，一度に生成される状態数が増加するために，展開する状態を決めるための時間が増加したためだと考えられる．

表 4: 重み k に値による各手法の振る舞い

	k	1	1.002	1.005	1.01
時間 [s]	WIDA*	-	1.82	2.09	0.58
	WA*	5754.96	281.56	189.98	142.16
	I–WIDA*	1037.59	0.27	0.33	0.49
節点数	WIDA*	-	784	691	564
	WA*	213280	72152	65419	61009
	I–WIDA*	7467	747	14471	39893
Step	WIDA*	-	4207	4847	1250
	WA*	6549	800	637	548
	I–WIDA*	1920324	500	463	460
スコア	WIDA*	-	24820	24405	23892
	WA*	24880	24880	24880	24869
	I–WIDA*	24880	24874	24851	24851
バック トラック	WIDA*	-	1048	1033	241
	I–WIDA*	900966	16	3	2

6. まとめ

IDA*に A*の $CLOSED$ リストを導入にすることで，再訪を回避する手法として I–IDA*を提案した．性能評価実験として行った 7 本のマルチプルアライメントでは，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 4 本のマルチプルアライメントにおいて，約 6200 倍の速度向上がみられた．

さらに，I–IDA*に重みを導入した I-WIDA*を提案した．性能評価実験として行った 7 本のマルチプルアライメントにおいて，重み $k = 1.002$ としたとき WA*の約 1000 倍の速度向上がみられ，最大記憶節点数は約 1 %に抑えられた．WIDA*と比較すると約 6.7 倍の速度向上がみられた．また，$k = 1.01$ のとき I–WIDA*は 9 本のマルチプルアライメントを約 3 秒で解くことができた．これは，IDA*の約 36 倍であり WA*の約 2800 倍であった．

参考文献

[Carrillo 88] H.Carrillo and D.Lipman, “The multiple sequence alignment problem in biology”. SIAM Journal Applied Mathematic, vol.48, pp.1073–1083, 1988.

[Ikeda 99] 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.

[Korf 93] R.E.Korf, “Linear-space best-first search”, Artificial Intelligence, vol.62, pp.41–78, 1993.

[三浦 97] 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- ，No. 9，pp.2438–2445，1997.

[Sen 89] 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

[Yoshizumi 00] T.Yoshizumi, T.Miura and T.Ishida ,“A* with Partial Expansion for large branching factor problems”, AAAI–2000, pp.923–929, 2000.