勉強会レジュメ
-アプリケーション-

片桐孝洋

平成7年 5月 17日

1 はじめに

並列処理の対象となるべき問題は, 数多く挙げられるであろう. 多くの分野でコンピュータの処理を必要とするような膨大な計算を必要とする問題がある. それゆえ, 並列処理により現存のコンピュータの性能を向上させることができないかと考えられた.

しかしながら, 処理の方式によっては不向きなものがあり, さらには並列処理をしてもそれに応じた性能が向上しないこともありうるであろう.

勉強を通じて, それら諸問題を理解していけるように努力したい.

2 タンパク質列のアラインメント問題

2.1 問題の定式化

タンパク質を構成するアミノ酸配列を解析し, タンパク質の立体構造や機能を推定する研究が盛んになされている. 近年ではDNA配列からタンパク質のアミノ酸配列が高速かつ大量に決定されるようになり, その結果得られる膨大なデータの解析・管理のため計算機による配列解析が必須となりつつある.

アミノ酸配列の解析手法の中に重要なものの一つに配列間の類似性を調べる相同性解析がある. この解析法の基本的な操作は, 複数のアミノ酸配列の類似する部分を同じ列に揃えて並べ合わせる操作であり, これを アラインメント という.

計算機でアラインメントを求める問題はN次元探索空間の最適経路問題に定式化できる. いま例として AHIE, ADHE の2本のアミノ酸配列をアラインメントする場合を考える.

このとき, 図中の丸をノード, 矢をアークとして, 矢にその類似性を表すスコア (類似性が高い程大きい) を付けると, 重みつき有向グラフとみなせる。

以上のようにこの問題空間を定義すると, 最適なアラインメントを求める問題はこのネットワーク上の左上のノードを出発点とし, 右下のノードを終着点とした最大経路問題に対応する.

2.2 マルチプルアラインメントの手法

3 本以上の配列のアラインメントを特にマルチプルアラインメント (Multiple Sequence Alignment) という.n 本のマルチプルアラインメントも 2 本の時と同様に考えると, 原理的には n 次元超空間中の最適経路問題に帰着する. しかしながらこの方法では膨大な計算時間を必要とするため現実的でない. それゆえいろいろな方法が提唱されているが, それらについてすべて説明することは目的でないから説明を割愛してひとつの方法についてのみ言及する.

反復改善法

反復改善法は 1991 年に Berger と Munson によって提唱された手法であり, 次のような処理を行なう

1. 初期配列をランダムに 2 つのグループに分割する.
2. 分割されたグループ間で グループ間アラインメント を行なう.
3. その結果得られたアラインメントを次の反復の初期配列とする.
4. この操作をスコアが収束するまで反復する.

この方法には幾つかの欠点がある.

欠点 1: 乱数を用いて逐次的に改善していくため, 得られるアラインメントが乱数系列に大きく依存し, 実行の度に得られる解が変わってしまう.

欠点 2: どの分割を選んだ時でもスコアが改善されるわけでもなく,1 本と n-1 本のように偏りがある分割したときにスコアが改善される場合がおおいことがある. 一方配列の本数が増えれば分割の数が増えて, スコア改善に寄与しない分割を選択する確率が増し, 結果として収束時間が増す.

これら欠点は分割方法すべてに対して並列にグループ間アラインメントを行ないもっとも良い初期アラインメントを選択すれば解決される. さらに反復のたびに必ず評価値が変わるために収束が早くなる. また乱数を用いていないため解が安定になる. これを 並列反復改善法 という.

限定分割法

すべて可能な分割方法にたいして処理を行なうとすれば, 列を n 本とするときその分割法は $2^{n-1}-1$ 通りあるから配列の数の増加に対して指数オーダで増加することは明らかである. よって扱える本数に制限がある. そこでこのグループ分割を一定の範囲に限定する. これは先述のとうり偏りのあるグループ分割が改善に寄与する場合が多いことを利用したものである.

2.3 並列反復法の高速化手法

ここでは並列反復改善法をさらに高速化する手法を述べる.

2.3.1　動的計画法の並列化

最初の方法てして, グループ間アラインメントを求める手段として動的計画法が考えられる. 動的計画法とはアラインメントの際, 最適なパスを決めるために以前の情報をメモリなどに一時記憶し, 実行のたびにそれを参照して問題解決を行なう手法である. この処理を複数のプロセッサで並列実行することを考える.

動的計画法によるアラインメント

動的計画法 (Dynamic Programing 法, 以下 DP と略す) は, アルゴリズムの各ステップで決定を行なう特徴を持つ問題に対して, 動的に決定を行なうものである. どうして動的に決定するかというと, 決定的に行なう場合しらみ潰しに探索し, 結果として計算量が爆発的に増大して, 解を求めることが現実的に不可能になることがある. よってこの決定を, 解が求まるであろう決定をおこない, 余分な実行を抑制する. その結果として問題のサイズ n に対して多項式時間で解ける問題になる.(多項式帰着性). アラインメント問題は結局ネットワークのスコア最大経路問題に帰着することは述べた.

DP により 2 次元ネットワークの最適経路を求める方法は次のとうりになる.

[アルゴリズム]

1. 現在のノードに至る直前に通過する可能性のあるノード全てを考え, 各々を通過してきた場合の最適スコアを計算する. すなわち, スタートノードからそれら各ノードまでの最短経路のスコアと, そこから当該ノードへのアークのスコアを加算する.
2. その中から最良のものを選びだし, その値を現在のノードへの最短経路のスコアとする. 直前にどのノードから選択したかという情報も付け加える.
3. 現在のノードがゴールノードでなければ 1. へ戻る.
4. ゴールノードであれば, そこから順に, どのノードから選択したかという情報を辿っていくことで最短経路を得る.

さて, この DP アルゴリズムは当該ノードの左, 上, 左上の 3 つのノードが計算済みであれば計算できる. よって左上から行優先で計算すればよい. このとき, 並列化を行なうためには次の図のような分割方法が考えられる. また各分割方法に対してつぎのように分類される.

グループ 1...(a)(b)(c) この分割方法では, 各プロセッサは前のプロセッサが計算が終るまでデータが得られず, 実行できない. よって並列実行するには不適である.

グループ 2...(d)(e)(f) プロセッサ P_i は一行分の処理を終えると, その結果を右どなりのプロセッサ P_{i+1} に送信する. P_{i+1} はそのデータを受信すると, その行の処理ができるので P_i の次の行とオーバーラップして処理を進めることができる.
このとき分割の大きさが小さいと, 処理よりも通信の時間がかかり処理時間が余分にかかってしまう恐れがある.

その他...(g) DP はゴールノードから始めても同じことである. よってスタートとゴール

の両方から並列に探索して図の真中の◎ で示したノードについて各パスを加算し, 最適なものを採用する.

さて, 逐次処理と並列処理との実行時間の評価であるが次のようになる.

l:配列長　t:1 ノード当たりの処理時間
p:並列実装台数　r:受信プロセッサのオーバヘッド
d:通信時間　s:送信プロセッサのオーバヘッド

とし,T_1 を逐次処理を行なった場合の全体の実行時間,T_p を p 台のプロセッサに分割して並列処理を行なった場合の処理時間とすれば

$$\begin{aligned}
T_1 &= (1\ \text{ノード当たりの処理時間}) \times (\text{全ノード数}) \\
&= tl^2 \\
T_2 &= (\text{行の処理時間} + \text{送信}) \times (1\ \text{から}\ l-1\ \text{行}) + (n\ \text{行の処理時間}) + \\
&\quad (\text{列の処理時間} + \text{通信時間} + \text{受信}) \times (p_2\text{から}\ p_p) \\
&= \left(\frac{tl}{p} + s\right)(l-1) + \frac{tl}{l} + \left(\frac{tl}{p} + d + r\right)(p-1)
\end{aligned}$$

よってその比は

$$\begin{aligned}
\frac{T_p}{T_1} &= \frac{(\frac{tl}{p} + s)(l-1) + \frac{tl}{l} + (\frac{tl}{p} + d + r)(p-1)}{tl^2} \\
&\to \frac{1}{p}(l \to \infty)
\end{aligned}$$

となり, 配列の長さがある程度おおきいとき, すなわち問題がおおきいときにはプロセッサ台数に比例した性能がでることが期待される.

2.3.2　A^* アルゴリズムの適用

第二の方法としてグループ間アラインメントを動的計画法のような盲目的探索ではなく, より解に近付くように探索するヒューリスティック探索である A^* アルゴリズムを採用して無駄な探索空間を削減して高速化する方法が考えられる. A^* アルゴリズムは最小なノード展開で解に到達することが証明されている.

A^* の評価関数 ヒューリスティック探索を実現するため, 展開する候補のノードから解があると思われるノードを決定するため, 評価関数を導入する.

$$g(n) \quad : \quad \text{初期節点から節点}\ n\ \text{までのコスト}$$

$$\begin{aligned} h(n) &\quad : \quad \text{推定関数} \\ f(n) &\quad = \quad g(n)+h(n) \end{aligned}$$

この推定関数の値が小さい節点から展開が行なわれる.

[アルゴリズム]
s:初期節点
OPEN:未展開接点の集合
CLOSED:展開済みの節点の集合

```
1.        $f(s)$ を計算.
          OPEN ← { s }
          CLOSE ← { }
2.        OPEN = { } なら探索は失敗.
3.        OPEN のなかから $f(n)$ が最小の節点 $n$ を選択.
          OPEN ← OPEN - { $n$ }
          CLOSED ← CLOSED ∪ { $n$ }
          $n$ が目標節点なら探索は成功.
          (解は節点 $n$ から $s$ へのポインタを辿ることによって得られる.)
4.        $n$ を展開.
          各 $n' \in$ について以下の処理を行なう.
    (a)   $n' \notin$ OPEN ∪ CLOSED の場合:
    i.    $n'$ から $n$ にポインタをはる.
    ii.   $f(n')$ を計算. OPEN ← OPEN ∪ { $n'$ }
    (b)   $n' \in$ OPEN の場合:
          $f(n')$ を再計算. 新たに計算した $f(n')$ の方が小さければ, 以下を行なう.
    i.    $n'$ から $n$ にポインタをはる.
    (c)   $n' \in$ CLOSED の場合:
          $f(n')$ を再計算. 新たに計算した $f(n')$ の方が小さければ, 以下を行なう.
    i.    $n'$ から $n$ にポインタをはる.
    ii.   OPEN ← OPEN ∪ { $n'$ }
    iii.  CLOSED ← CLOSED - { $n'$ }
```

A^* の性質
ここでは A^* の性質を述べる.

[適格性 (admissibility)]:
推定関数 h が適格, すなわち $\forall n$, $h(n) \leq h^*(n)$ であれば, 無限グラフの問題空間において, 初期節点から目標節点に至る経路が存在すれば A^* は必ず最適解をみつける.

証明

$g(x) > f^*(s)$ なる任意の節点 x が展開される前に, 目標節点に到達することを示す.

1. $s = n_0, n_1, \cdots, n_i, n_{i+1}, \cdots, n_k = t$ を目標節点に至る最短経路とし, 既に $n_0, \cdots, n_i$ が展開されているとする.
仮定の適格性より, $h(n_{i+1}) \leq h^*(n_{i+1})$. また, $n_0, \cdots n_i$ がすべて展開されているので $g(n_{i+1}) = g^*(n_{i+1})$. 従って,

$$\begin{aligned} f(n_{i+1}) &= g(n_{i+1}) + h(n_{i+1}) \leq g^*(n_{i+1}) + h^*(n_{i+1}) \\ &= f^*(s) < g(x) \leq g(x) + h(x) = f(x) \end{aligned}$$

ゆえに n_{i+1} が x より先に展開される.
2. 上記を繰り返し適用すると, 節点 x を展開する前に目標節点 n_k に到達する. (証明終り)

[推定関数の単調性 (monotonicity)]:
推定関数が単調, すなわち初期節点からの経路に沿って推定関数の値が決して減少しないとき, A^* による節点の展開は, その節点に向けての最短経路上でのみおこなわれる. すなわち CLOSED 節点は再展開されない.

クループ間アラインメントへの応用

問題空間の変換 DP による解法では, 「スコア」を与え, その「スコア最大経路」の問題に帰着させた. そこで A^* による探索では, アークにコストが与えてあり, 「コスト最小経路」問題に帰着することを示す.

まず最初に A^* の評価関数は負にならないため, 現スコア S に下駄ばき c をさせ $-S+c > 0$ とすることを思いつくであろう。しかしながら, この増分は経路に依存して, もとの探索空間と等価にならなくなる. よって, コストがすべて正になるように変換することを考える.

いまネットワークの開始点を $n(0,0)$ とし, ゴールノードを $n(l,l)$ と書くこととする. いま斜めのアークの数を D とすると, $n(i,j)$ でのホップ数を以下のように定義する.

$$H(i,j) = i + j - D$$

これに, 斜め方向のアークを 2hop とする $n(i,j)$ での最適経路のホップ数を $H'(i,j)$ は経路によらず 常に i+j で一定となる. よって 1hop あたり$+c$ のかさあげすれは, 探索空間はもとのものと等価になる. これを利用して次の補題を証明する. ただし, 任意の経路 P のスコアを $Score(P)$, 同じ経路の等価変換された問題空間のコストを $Cost(P)$ とする.

補題
もとの問題空間における任意の経路 P のスコアを $Score(P)$, 同じ経路の新しく設定された問題空間のコストを $Cost(P)$ とする. ある 2 点間の任意の 2 つの経路 P_1, P_2 について

$$Score(P_1) < Score(P_2) \Leftrightarrow Cost(P_1) > Cost(P_2)$$

証明

先ほどの方法より

$$\begin{aligned} Cost(P_1) &= -Score(P_1) + H'(P_1) \times c \qquad (1) \\ Cost(P_2) &= -Score(P_2) + H'(P_2) \times c \qquad (2) \end{aligned}$$

上述のように2点間の経路のhop数は経路によらず一定であったから

$$H'(P_1) = H'(P_2)$$

(1)(2)式を差引き, 上式を代入して

$$Cost(P_1) - Cost(P_2) = Score(P_2) - Score(P_1)$$

ゆえに

$$Score(P_1) < Score(P_2) \Leftrightarrow Cost(P_1) > Cost(P_2)$$ （証明終り）

ヒューリスティック関数の設定

A^* アルゴリズムはその推定関数 h をどう設定するかによって大きく依存する. すなわち実行に対して

1. 最適経路発見の保証
2. 経路発見に際して拡張されるノード数
3. h の計算に必要なコスト

である. ここでは

$$h^*(n) = \sum_{m=j+1}^{l} \min_{k\in\{x|i+1\le x\le l\}} C(m,k) + P' \times |i-j| \le h(n)$$

$C_{i,j} : n(i-1, j-1)$ と $n(i,j)$ とを結ぶアークに与えられたコスト, P' : ギャップペナルティ

と述べるだけに留まる.

2.4 性能評価と考察

表1にプロセッサ数と処理時間, 速度向上比, 効率の関係を示す.

- 問題がおおきくなるほど, 並列処理の効果がでる

[DP の台数効果]

図16 に DP の並列化による速度向上比を示す.

- 並列の台数効果が比較的出ている

配列長が大きいほど粒度が大きくなるため, うまくプロセッサ間通信によるレイテンシが隠蔽されている. 並列化の手法では DP のほうが向いているアルゴリズムであることがわかる.

[A^* アルゴリズムの探索効率]

図17 は横軸は得られたアラインメントのカラム当たりの平均スコアである. この値がおおきいほど類似していることを表す.

- 類似度が高いほど探索空間が削減でき, 効果的である.

比較的, 類似度が低い配列でも, 全体の２５%程度で探索が可能だが, データに依存することは明らかである.

3 おわりに

並列化の効果があるのは, 反復法などのように, データの流れが決まっているアルゴリズムの方が性能がでやすい. というのは, プロセッサ間のデータの受渡しに関する通信時間が削減されるからである. また, 実際プロセッサを割り当てるとき分割の方法が不適格であれは, まったく並列処理ができないこともある.

一方, 問題自体の性質も考慮すべきである. 場合によっては, 並列処理しなくてもアルゴリズムの変更により処理速度が向上する場合もありうるだろう.

4 参考文献

[1] 荒木 志帆 並列反復改善法によるタンパク質配列のアラインメント (動的計画法の並列化と A^* アルゴリズムの適用). 京都大学卒業論文. 1993.

[2] A.Barr and E.A.Feigenbaum,$TheHandbookofArtificalIntelligence$, Willam Kaufmann,Inc,1981.

[3] 野寺 隆志. 楽々 LaTeX. 共立出版株式会社

5　アプリケーション資料