2003－AL－90 (1)
2003／5／23

進化木の Quarted distance の計算アルゴリズムの実装

廣川 裕　　徳山 豪
東北大学大学院　　情報科学研究科

概要 生物進化学においては種と種の関係を表す進化木が中心的な道具となっており、異なった進化木同士の違いを定量化することはとても重要な仕事である。quartet distance とは、Estabrook, McMorris, Meacham らによって提案された二つの木の間の距離尺度である。生物種のうち 4 種を選択することによって得られる部分木の組み合わせ構造を quartet topology というが、二つの根を持たない進化木の間で異なった quartet topology を取る 4 種の組み合わせの数を quartet distance と言う。本研究では Brodal 他が 2001 年に発表した quartet distance を $O(n \log^2 n)$ の時間で計算するプログラムを実装し、実験結果を示す。

Research on Algorithm Computing the Quartet Distance between Evolutional Trees

Yutaka Hirokawa　　Takeshi Tokuyama
GSIS, Tohoku University

Abstract. In evolutionary biology, evolutionary trees describing the relationship of a set of spieces are widely used, and quantifying differences between evolutional trees is a weighty task. The quartet distance is a distance measure between trees proposed by Estabrook, McMorris and Meacham. In this paper, we report experimental results on the algorithm put up by G.S.Brodal et al. in 2001, which compute the quartet distance between two unrooted evolutionary trees of n species in time $O(n \log^2 n)$.

1 はじめに

1.1 進化木の比較

生物の進化論的な結びつきを記述する際に、葉は生物の種に相当し、内部の節点は進化が異なった方向に分岐した時点に相当する木構造のグラフを用いる。このグラフを進化木という。ある生物種の集合が与えられた時、それらに対する正しい進化木はほとんどの場合不明である。そのため、遺伝子情報など、その生物種に関して得られる情報から進化木を推定するということが注目されている。以降、進化木は全ての内部節点の次数が 3 であり根を持たない木であるとする。異なったモデルと構築方法による推定を行うと、同じ生物種の集合に対しても異なった進化木を生み出すことがしばしばあり、また同じモデルと方法で生物種についての情報が異なれば、異なった進化木が生じることがある。このような違いを組織的な方法で研究するためには、明確で効率の良い方法を用いて進化木の間の違いを定量化する必要がある。

二つの進化木を比較するための一つのアプローチとして、二つの木の間にある距離尺度を定義し、それを計算することによって二つの木を比較する方法がある。これまで多くの距離尺度が提案されてきたが、この研究ではその中の一つである *quartet metric* について研究する。

1.2 quartet topology

n 種の生物の集合 S から成る進化木に対して、n 種のうちの 4 種によってもたらされるラベル付き部分木の組み合わせ構造を *quartet topology* と呼ぶ。一般的に a, b, c, d の 4 種によってもたらされる quartet topology には図 1 に示されるように 3 つのパターンが存在する。

quartet topology とは、例えばもし T における a から b, c, d それぞれへの 3 本の path の中で、b への path が最初に他の path から分離する場合は a と b をペアにする、といったように、4 種を 2 種ずつのペ

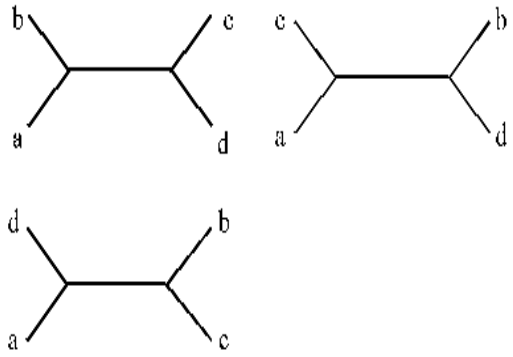

図 1: Quartet Topology の 3 つのパターン

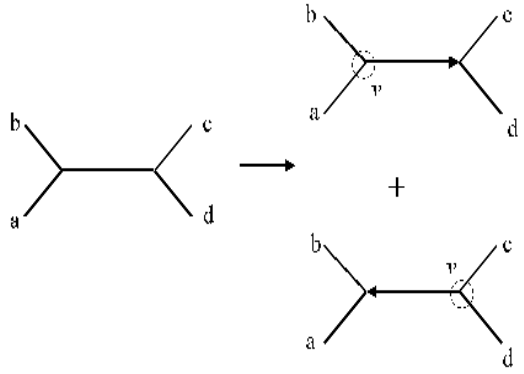

図 2: Quartet Topology の 2 つの方向付け

アにすることと同値である。

等しい生物種の集合 S から成る異なった二つの進化木 T_1、T_2 が与えられた時、この二つの木 T_1 と T_2 の間で異なった quartet topology を持つ 4 要素集合 $\{a, b, c, d\} \subseteq S$ の数を *quartet distance* と呼ぶ。S の 4 要素集合には $\binom{n}{4}$ 通りの組み合わせがあるので、全ての組み合わせを一つ一つ調べていくと quartet distance を計算するのに $O(n^4)$ もの時間がかかってしまう。本研究では Brodal らが 2001 年に発表した quartet distance を $O(n \log^2 n)$ の時間で計算するアルゴリズム [1] を、実際にプログラムを作り実装した。かなり複雑なアルゴリズムであるにもかかわらず、理論通りの高速性を持つことが示された。

2 Quartet distance

2.1 Quartet の方向付け

quartet distance は S の 4 要素集合の組み合わせの数 $\binom{n}{4}$ から T_1 と T_2 で同一の quartet topology を持つ 4 要素集合の数を引くことによって計算される。

二つの木において等しい quartet topology を見つけやすくするために、4 要素集合 $\{a, b, c, d\}$ の quartet topology の”中央の辺”を有向辺とすることによって、二つの”有向”quartet topology を考える。図 2 は一つの quartet topology から生じる二つの有向 quartet topology を示している。木 T_1 と T_2 の間で等しい有向 quartet topology の数は、方向付けされていない quartet topology の数の 2 倍となる。以下、4 要素集合の有向 quartet topology を *quartet* と呼ぶこととする。

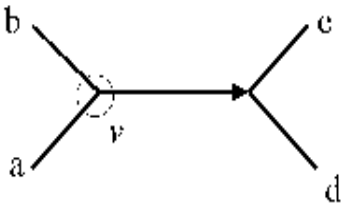

図 3: 一般的な quartet

2.2 集合 S の塗り分け

次に、quartet を T_1 の内部節点に以下のように関連付ける。図 3 に示すような $\{a, b\}$ から{c,d} へ方向付けられた一般的な quartet について考える。T_1 には a から c(または d) への path と b から c への path の交わる節点 v が唯一つだけ存在する。

このとき、図 3 の quartet を節点 v に関連づける。全ての内部節点の次数が 3 である時、葉の数が n である木の内部節点の数は $n-2$ であるので、これによって $2\binom{n}{4}$ 個ある T_1 の quartet は $n-2$ の互いに素な集合に分割される。T_1 のある内部節点 v に対して、T_1 から v と v に入射する 3 本の辺を取り除くことによって生じる三つの部分木を v に”入射する”部分木と呼ぶことにする。これは図 2.3 の A、B、C によって示される。ここで、v に関連付けられた quartet の数は以下の式で与えられる。

$$\binom{|A|}{2} \cdot |B| \cdot |C| + \binom{|B|}{2} \cdot |C| \cdot |A| + \binom{|C|}{2} \cdot |A| \cdot |B|$$

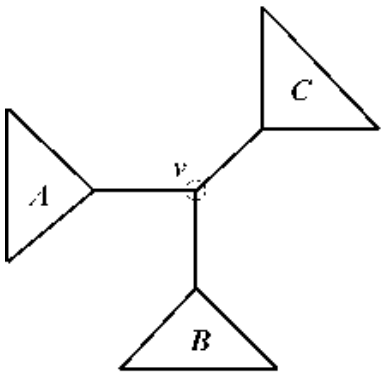

図 4: v に入射する部分木

$|T|$ は部分木 T の葉の数を示す。この式の三つの項は、図 3 の c と d がそれぞれ図 4 の部分木 A、B、C に含まれている quartet の数である。

T_1 と T_2 の間の quartet distance を計算するための戦略として、T_1 のそれぞれの内部節点 v に対して、v に関連付けられた quartet のうち T_2 においても同じ topology を持つ quartet がいくつ存在するかを計算していく。T_1 の全ての節点に対してのこの計算を合計すると、T_1 と T_2 において同一の quartet の総数が得られる。

この戦略を実行するために、生物種の集合 S の要素の $\mathcal{A}, \mathcal{B}, \mathcal{C}$ の 3 色へ塗り分け、以下の二つの定義によってこの塗り分けと quartet を関係付ける。T_1 の内部節点 v について、v に入射する三つの部分木の一つの葉のラベルが全て $\mathcal{A}$ で塗られ、他の部分木の一つの葉のラベルが全て $\mathcal{B}$ で塗られ、残った部分木の葉のラベルは全て $\mathcal{C}$ で塗られている時、S の要素は v に"従って"塗り分けられていると言う。S の要素の塗り分けと図 3 の $\{a, b\}$ から $\{c, d\}$ へ方向付けられた quartet について、a と b が異なった色で塗り分けられていて、c と d が共に残った色で塗られている時、その quartet は塗り分けに"適合する" と言う。これらの定義から次の補題が導き出される。

補題 1　T_1 の内部節点 v を考える。このとき、ある quartet Q が v に関連付けられているための必要十分条件は Q が v に従った塗り分けに適合する事である。

補題 1 より、S が T_1 において選んだ v に従って塗り分けられている時、その塗り分けに適合する T_2 の quartet は、T_1 においても同じ topology を持つ。

この塗り分けは後に述べるデータ構造によって動的に更新される。このデータ構造の主な特徴は定数時間で現在の塗り分けに適合する T_2 の quartet の数を返すことである。また、このデータ構造は生物種集合 S の n 個の要素のうち k 個の要素の色が塗り替えられた時、$O(k + k \log \frac{n}{k})$ の時間でそのデータ構造を更新するができる。このアルゴリズムは T_1 の節点 v それぞれに従って S の要素を塗り分け、T_2 においても v に関連付けられた quartet と同じ topology を持つ quartet の数を計算する。

3 Hierarchical Decomposition

3.1 component

後に記述するデータ構造に必要不可欠なのが進化木 T_2 の *hierarchical decomposition* である。全ての節点の次数が 3 以下である根を持たない木 T が与えられ、T の葉の数を n とした時、$O(\log n)$ の高さを持つ T の hierarchical decomposition を得る方法を述べる。この hierarchical decomposition は *component* の概念に基づいている。T の component C を、次に示す二つの条件のいずれかを満たす T の節点の部分集合として定義する。

1. T の節点一つから成る集合
2. 2 本以下の *external edge*(C の節点と $T - C$ の節点を結ぶ T の辺) を持つ連結した節点の部分集合。

言い換えれば、component は節点一つから成る集合か、カットがサイズ 2 以下の部分集合で定義される連結した部分集合かのどちらかである。component の"次数"は component の external edge の数である。上記の 2 番目の条件より、二つ以上の節点から成る component の次数は 2 以下であることがわかる。

T の節点（葉も含む）はそれぞれ type1 の component を構成し、type2 の component は隣接した二つの component C' と C'' の和集合として形成される。C' と C'' が $u \in C'$ かつ $v \in C''$ となるような辺 (u, v) が T に存在する時、C' と C'' は隣接していると言う。このような二つの component の和集

合を *composition* と呼ぶことにする。以上の条件より、composition は図 5 に示す 4 つのパターンのみである。節点は収縮した conponent を意味し、楕円は composition を意味する。二つの component の composition はそれぞれ二つの component を結び付けている木 T の辺と一対一に対応している。

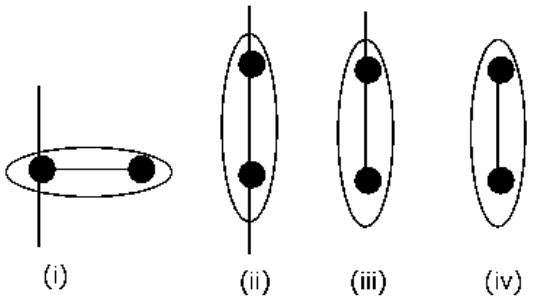

図 5: composition の 4 つのパターン

3.2 hierarchical decomposition$H(T)$の性質

根を持たない木 T の hierarchical decomposition とは、それぞれの節点が T の component を意味する根を持った二分木 $H(T)$ のことである。$H(T)$ の葉は type1 の component を意味し、これらの component と $H(T)$ の葉の間には一対一の写像関係がある。$H(T)$ の内部節点 v は、v の子が表す二つの component の composition として形成された、type2 の component を意味する。このように構成された hierarchical decomposition tree は節点の数を n 個とすると $O(\log n)$ の高さを持ち、また locally-balanced である。節点を n 個持ち根のある二分木において、全ての節点 v に対して、v を根とする部分木の高さが $c(1+\log|v|)$（$|v|$ は v を根とする部分木に含まれる節点の数）以下である時、その二分木は *c-locally-balanced* であると言う。

補題 2　次数 3 以下の節点を n 個持ち根を持たない全ての木に対して、$(1/\log\frac{12}{11})$-locally-balanced である hierarchical decomposition tree を $O(n)$ の時間で生成することができる。（$1/\log\frac{12}{11} \fallingdotseq 7.966$）

4　進化木の節点のデータ構造

T を進化木、$H(T)$ を T の hierarchical decomposition tree とする。これより、与えられた S の塗り分けに適合する T の quartet の数を定数時間で返し、与えられた塗り分けに対して $O(n)$ の時間で生成され、S の k 個の要素の色が変化した際には $O(k+k\log\frac{n}{k})$ の時間で更新される情報を $H(T)$ の節点にどのように蓄えるかを述べる。

$H(T)$ の節点各々に整数のタプル (a,b,c) と関数 F という情報を与える。$H(T)$ の節点は T の component を表している。タプルの整数 a,b,c はそれぞれ $\mathcal{A},\mathcal{B},\mathcal{C}$ に塗り分けられ、この component に含まれる T の葉にラベルづけされた要素の数であるとする。component は $k=0$〜3 本の external edge を持つ。ただし、external edge が 0 本というのは $H(T)$ の根が表す T の節点を全て含んだ component のみである。関数 F は、その conponent の external edge 各々に対して変数を三つずつ、合わせて最大 9 個の変数を持つ。external edge を一本以上持つ component に対して、これらの辺に 1〜k までの番号付けを任意に行い、$\boldsymbol{a}_i,\boldsymbol{b}_i,\boldsymbol{c}_i$ は辺 i に対応する三つの変数を示すとする。T から external edge が一本取り除かれた時、T の部分木が二つ生じるが、片方は元の component を含まない。この部分木をその external edge によって”もたらされる”部分木と呼ぶこととする。変数 $\boldsymbol{a}_i,\boldsymbol{b}_i,\boldsymbol{c}_i$ は辺 i によってもたらされる部分木の葉にラベル付けられた生物種集合 S の要素のうち、$\mathcal{A},\mathcal{B},\mathcal{C}$ それぞれに塗り分けられた要素の数である。そして、関数 F は変数 $\boldsymbol{a}_i,\boldsymbol{b}_i,\boldsymbol{c}_i(1\le i\le k)$ を引数として、その component に含まれる節点に関連付けられ、かつ与えられた塗り分けに対して適合する quartet の数を返す関数である。また、F は合計で次数 4 以下の多項式であることがわかる。

$H(T)$ の根は木 T 全体を含む component を表す。すなわち external edge を持たないので、根に保存されている関数 F は実際には定数となる。よって、与えられた S の塗り分けに適合する T の quartet の数は、$H(T)$ の根に蓄えられた情報の一部であることがわかる。

補題 3　木 $H(T)$ は $O(n)$ の時間で上記のような情報を得ることができる。

この情報は $H(T)$ の bottom-up の走査によって生

成される。まず初めに $H(T)$ の葉、すなわち T の一つの節点から成る component を表す節点に蓄えられる情報がどのように生成されるかを述べる。T の節点は次数 1 の葉か次数 3 の内部節点かのどちらかであったことを思い出して頂きたい。

$\mathcal{A},\mathcal{B},\mathcal{C}$ に塗り分けられた一つの葉から成る component に対して、タプルはそれぞれ $(1,0,0),(0,1,0),(0,0,1)$ となる。quartet は T の内部節点にのみ関連付けられ、葉に関連付けられることはないので、関数 F は 0 となる。

次数 3 の節点 u 一つから成る component に対しては、T の葉を含まないため、タプルは $(0,0,0)$ となる。関数 F は、与えられた塗り分けに適合していて T の u に関連付けられた quartet の数を計算しなければならない。$\{a,b\}$ から $\{c,d\}$ へ方向付けられた一般的な quartet がこの条件を満たすのは、c と d が component の external edge によってもたらされる三つの部分木の一つに含まれ、同じ色に塗られ、そして a と b が残った二つの部分木にそれぞれ含まれ、残った色にそれぞれ塗られている時である。辺 1 によってもたらされる部分木に c と d が含まれ、$\mathcal{A}$ で塗られた時、これを満たす quartet の数は

$$\binom{\boldsymbol{a}_1}{2}\cdot(\boldsymbol{b}_2\boldsymbol{c}_3+\boldsymbol{b}_3\boldsymbol{c}_2)$$

となる。c と d に対して部分木と色の $3\cdot 3=9$ の組み合わせを全て足し合わせると

$$\begin{aligned}
&F(\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1,\boldsymbol{a}_2,\boldsymbol{b}_2,\boldsymbol{c}_2,\boldsymbol{a}_3,\boldsymbol{b}_3,\boldsymbol{c}_3)\\
&=\tbinom{\boldsymbol{a}_1}{2}\cdot(\boldsymbol{b}_2\boldsymbol{c}_3+\boldsymbol{b}_3\boldsymbol{c}_2)+\tbinom{\boldsymbol{a}_2}{2}\cdot(\boldsymbol{b}_1\boldsymbol{c}_3+\boldsymbol{b}_3\boldsymbol{c}_1)\\
&+\tbinom{\boldsymbol{a}_3}{2}\cdot(\boldsymbol{b}_2\boldsymbol{c}_1+\boldsymbol{b}_1\boldsymbol{c}_2)\\
&+\tbinom{\boldsymbol{b}_1}{2}\cdot(\boldsymbol{a}_2\boldsymbol{c}_3+\boldsymbol{a}_3\boldsymbol{c}_2)+\tbinom{\boldsymbol{b}_2}{2}\cdot(\boldsymbol{a}_1\boldsymbol{c}_3+\boldsymbol{a}_3\boldsymbol{c}_1)\\
&+\tbinom{\boldsymbol{b}_3}{2}\cdot(\boldsymbol{a}_2\boldsymbol{c}_1+\boldsymbol{a}_1\boldsymbol{c}_2)\\
&+\tbinom{\boldsymbol{c}_1}{2}\cdot(\boldsymbol{b}_2\boldsymbol{a}_3+\boldsymbol{b}_3\boldsymbol{a}_2)+\tbinom{\boldsymbol{c}_2}{2}\cdot(\boldsymbol{b}_1\boldsymbol{a}_3+\boldsymbol{b}_3\boldsymbol{a}_1)\\
&+\tbinom{\boldsymbol{c}_3}{2}\cdot(\boldsymbol{b}_2\boldsymbol{a}_1+\boldsymbol{b}_1\boldsymbol{a}_2)
\end{aligned}$$

が得られる。

次に $H(T)$ の内部節点に保存される情報の生成法を与える。まず、二つの component C' と C'' の composition について考える。C' を表す節点に保存される情報を (a',b',c') と F'、C'' を表す節点に保存される情報を (a'',b'',c'') と F'' とすると、C' と C'' の composition C を表す節点に保存される情報は (a'+a",b'+b",c'+c") と F となる。但し、この F は composition のタイプに依存する。component が (ii) の composition である場合、component の external edge の番号付けは次のようになる。C' と C'' の 1 番目の external edge を C' と C'' を連結する辺とし、C' の 2 番目の external edge を C の 1 番目の external edge とし、C'' の 2 番目の external edge を C の 2 番目の external edge とすると C の F は下式のようになる。これとは異なった external edge の番号付けの場合でも、F' と F'' の引数の然るべき変換を行えば F を得ることができる。

$$\begin{aligned}
&F(\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1,\boldsymbol{a}_2,\boldsymbol{b}_2,\boldsymbol{c}_2)\\
&\quad=F'(\boldsymbol{a}_2+a'',\boldsymbol{b}_2+b'',\boldsymbol{c}_2+c'',\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1)\\
&\quad+F''(\boldsymbol{a}_1+a',\boldsymbol{b}_1+b',\boldsymbol{c}_1+c',\boldsymbol{a}_2,\boldsymbol{b}_2,\boldsymbol{c}_2)
\end{aligned}$$

(iii) や (iv) の composition は F の定義がより簡単になっただけで、基本的に (ii) と同じである。C'' が次数 1 の component であると仮定すると、(iii) の composition に対しては

$$\begin{aligned}
F(\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1)&=F'(a'',b'',c'',\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1)\\
&\quad+F''(\boldsymbol{a}_1+a',\boldsymbol{b}_1+b',\boldsymbol{c}_1+c')
\end{aligned}$$

を、(iv) の composition に対しては

$$F=F'(a'',b'',c'')+F''(a',b',c')$$

を得ることができる。ここで留意すべきは、(iv) の composition における F は定数であるということである。最後に、(i) の composition の F については、C' が次数 1 で C の 1 番目の辺と 2 番目の辺はそれぞれ C'' の 2 番目の辺と 3 番目の辺であると仮定すると次のようになる。

$$\begin{aligned}
&F(\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1,\boldsymbol{a}_2,\boldsymbol{b}_2,\boldsymbol{c}_2)\\
&\quad=F'(\boldsymbol{a}_1+\boldsymbol{a}_2+a'',\boldsymbol{b}_1+\boldsymbol{b}_2+b'',\boldsymbol{c}_1+\boldsymbol{c}_2+c'')\\
&\quad+F''(a',b',c',\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1,\boldsymbol{a}_2,\boldsymbol{b}_2,\boldsymbol{c}_2)
\end{aligned}$$

component に保存された関数 F の定義により帰納的に展開していくと、F は常に 4 次以下の多項式であることがわかる。次数 4 以下で 9 変数の多項式は係数を保存することによって定スペースに保存され、また二つの多項式を足し合わせる等の操作も定数時間で行うことができる。つまり、二つの component C' と C'' の composition$C$ に保存される情報は C' と C'' に保存されている情報が既知であれば定数時間で計算することができる。よって、$H(T)$ は $O(n)$ の時間で情報を蓄えることができる。

補題 4　S の要素のうち k 個の色が変化した時、$H(T)$ の情報は $O(k + k \log \frac{n}{k})$ の時間で更新される。

証明.　$H(T)$ の節点の情報は子の情報のみによる、すなわち S の要素の色が変化した時更新される必要がある $H(T)$ の情報は、その要素に対応する $H(T)$ の葉の先祖だけであることがわかった。また、$H(T)$ の節点の情報は子の情報がわかっていれば定数時間で計算することができる。$H(T)$ は $(1/\log \frac{12}{11})$-locally-balanced であるため、更新されるべき節点の数は $O(k + k \log \frac{n}{k})$ 以下となる。これは簡単な漸化式によって証明される。

合計で要する時間は更新するべき節点の数に比例するので、この補題は証明される。　□

5　計算アルゴリズム

```
Procedure Count(v)
  if v is a leaf then
    color v by the color C
    return 0
  else
    ColorLeaves(Small(v), B)
    x =NodeCount(v)
    ColorLeaves(Small(v), C)
    y =Count(Large(v))
    ColorLeaves(Small(v), A)
    z =Count(Small(v))
    return x + y + z
```

図 6: 基本アルゴリズム

このアルゴリズムはまず初めに T_1 の任意の葉を根とする。それから、T_1 の根から後順走査によって節点 v それぞれについてサイズ $|v|$ を計算し、この情報を v に保存する。但し、$|v|$ は v を根とした部分木の葉の数を示すとする。同様に S の要素を全て $\mathcal{A}$ 色で塗る。但し、根は例外的に $\mathcal{C}$ で塗る。このアルゴリズムは T_1 の根の唯一つの内部節点 v から始まり、T_1 の全ての内部節点それぞれについての計算を合計する。節点 v においてこのアルゴリズムは、まず v の二つの子の大きい方の子についてこのアルゴリズムを再帰的に呼び出して計算し、次に小さい方の子、最後に節点 v における計算結果をこれまでの合計に加える。

図 6 はこのアルゴリズムを擬似コードによって再帰的手続きCount(v) として記述したものである。Count(v) は呼び出されると T_1 の v を根とする部分木の内部節点と v についての計算結果の合計を返す。これは、初めに T_1 の根の一つだけの子を v として呼び出される。Small(v) とLarge(v) はそれぞれ v のサイズが小さい方の子、大きい方の子をである節点を返す。NodeCount(v) は節点 v についての計算結果を返す。ColorLeaves($v, \mathcal{X}$) は T_1 の v を根とする部分木の葉にラベル付けされた生物種集合 S の要素を $\mathcal{X}$ 色で塗り替える。これは v を根とする部分木の走査によって実現される。そして、このデータ構造における S の要素とそれらがラベル付けされた T_1 と T_2 の葉の間に双方向性のポインタを持たせることによって、これは $k = |v|$ とした補題 4 によって定められる計算時間で実行される。

Theorem 1　T_1 と T_2 を同一の生物種集合 S から成る根をもたない二つの進化木とし、それらの全ての内部節点の次数は 3 であるとする。その時、T_1 と T_2 の間の *quartet distance* は $O(n \log^2 n)$ の時間で計算される。

証明.　このアルゴリズムは次のような不変条件を満たす。

1. あるCount(v) の実行開始時、T_1 の v 以下の葉にラベル付けられた S の要素が全て $\mathcal{A}$ 色で塗られ、他の S の要素は全て $\mathcal{C}$ 色で塗られている。
2. あるCount(v) の実行終了時には S の全ての要素が $\mathcal{C}$ 色で塗られている。

これは、NodeCount(v) が呼び出された時、Small(v) の部分木を葉のラベルは $\mathcal{B}$ 色で塗られ、Large(v) の部分木の葉のラベルは $\mathcal{A}$ 色で塗られ、残った要素が $\mathcal{C}$ で塗られるということを示している。言い換えれば、S の要素は v に従って塗り分けられているということである。

Count(v) が呼び出された時の計算回数は補題 4 より $O(k \log n)$ となる。但し $k = |\text{Small}(v)|$ である。

ここで、この計算回数は T_1 のSmall(v) を根とした部分木のそれぞれの葉（v が葉の場合は v 自身）に $O(\log n)$ ずつの計算時間を要すると考えることができる。与えられた葉に対して、この計算回数が割り当てられるのは、その葉から根への path 上の節点 v でのみだけである。この path はSmall(v) から v への辺を通る。v のサイズは必ずSmall(v) のサイズの 2 倍以上となるので、この計算回数は高々$\log n$ 回である。従って、それぞれの葉における計算時間は $O(\log^2 n)$ となり、合計で計算時間は $O(n \log^2 n)$ となる。

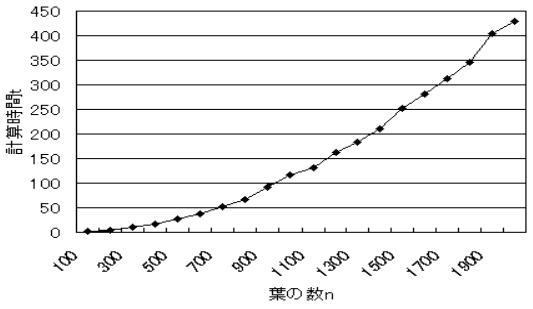

図 7: $O(n^2)$ のプログラムの計算時間

6 アルゴリズムの実装

これまで述べたアルゴリズムを LEDA を用いて実装した。

まず初めは、関数 F の再帰式をそのまま用いた quartet distance を計算するプログラムを作成した。この時注意しなければならないのは、component の external edge につける番号として、前述の定義式の番号をそのまま用いず、全ての component において、例えば根に向かう external edge を 1 番、葉に向かう external edge を 2 番とするといったように共通のルールを決めておく必要があるということである。何故なら、関数 F はどの番号の external edge の変数を何番目の引数として扱うかが固定されているので、component C' と C'' の external edge の番号と composition C の関数 F の引数の番号がきちんと対応していないと計算に狂いが生じるからである。

このプログラムを用いてランダム生成した二つの進化木の quartet distance を計算し、葉の数を $n = 100$ 〜2000 まで 100 刻みで変化させ、計算時間を測定した。しかし、このプログラムでは図 7 に示すように計算時間が $O(n^2)$ となってしまった。これは、$H(T_2)$ の根に保存された関数 F を呼び出す度、再帰的に $O(n)$ 個ある $H(T_2)$ の節点全てについて関数 F を計算してしまい、それを $O(n)$ 個ある T_1 の内部節点ごとに行うためである。

次に、$H(T_2)$ のそれぞれの節点ごとに関数 F を展開し、その 4 次以下の項全ての係数を保存し、T_1 の内部節点ごとに S の要素を塗り替え、その係数をタプル (a, b, c) と同じように子の係数から計算して更新し、$H(T_2)$ の定数項を返すことによって quartet distance を計算するプログラムを作成し、同様の実験を行った。

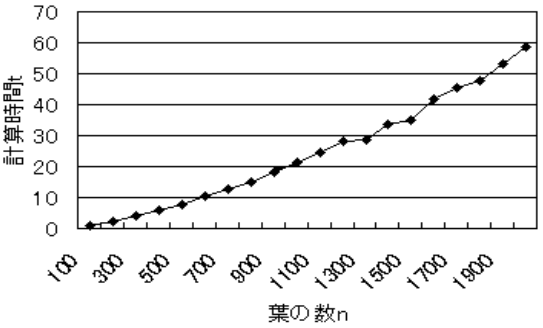

図 8: $O(n \log^2 n)$ のプログラムの計算時間

その結果、計算時間は図 8 のようになり、確かに $O(n \log^2 n)$ の計算時間でプログラムが走っていることが確認された。そして図 9 に示す通り、先程のプログラムに比べ非常に高速で動くことがわかるだろう。

7 まとめ

今後の課題としては、最近発見された quartet distance を $O(n \log n)$ の時間で計算するアルゴリズム [2] の実装、そしてラベル付けされた進化木とされていない進化木が与えられた場合、どのようにラベル付けすれば quartet distance を最小化できるかを考える問題等が挙げられる。

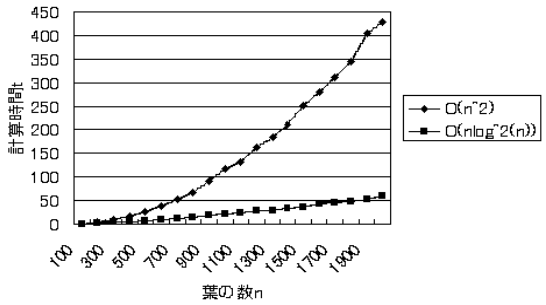

図 9: 二つのプログラムの計算時間の比較

参考文献

[1] G.S.Brodal, R.Fagerberg, and C.N.S.Pedersen. Computing the Quartet Distance between Evolutionary Trees in Time $O(n \log^2 n)$. In *Proceeding of the 12th International Symposium on Algorithms and Computation (ISAAC)*, volume 2223, of *Lecture Notes in Computer Science*, pages 731-742. Springer-Verlag, 2001.

[2] G.S.Brodal, R.Fagerberg, and C.N.S.Pedersen. Computing the Quartet Distance between evolutionary trees on time $O(n \log n)$. *Alcom-FT*, 2001

[3] 浅野哲夫, 小保方幸次.『LEDA で始める C/C++ プログラミング』, 株式会社サイエンス社, 2002