アルゴリズム 16－8
（1990．7．17）

木の公平な重み付けの手法について

栗野 俊一　中村 憲一郎　深澤 良彰　門倉 敏夫
早稲田大学理工学部

木構造に対する重み付けの手法としてトップダウンアルゴリズムとボトムアップアルゴリズムの２つが知られており，これらは異なる結果を与える．我々はトップダウンやボトムアップの重み付けの際の重みの移動に着目し，これを一般化した比付きの重み付けを考案した．そして各々の静的な重み付けのアルゴリズムと動的な重み付けのアルゴリズムを定義し，これらの性質を比較した．比付きの重み付けアルゴリズムを用いて，ｎ個の節点を持ち，兄弟の数の最大値がｗである木に対して重み付けを行った際の計算量はＯ(ｎ＋ｗ)で，トップダウンアルゴリズムやボトムアップアルゴリズムの場合と同程度の計算量で済むことが分かった．

A METHOD OF ASSIGNING FAIR WEIGHTS TO NODES OF TREE

Shun-ichi Kurino　Ken-ichiro Nakamura　Yoshiaki Fukazawa　Toshio Kadokura

School of Science and Engineering, Waseda University
Okubo 3-4-1, Shinjuku-ku, Tokyo 169, Japan

We propose a new algorithm for assigning weights to nodes of tree. This algorithm includes top-down and bottom-up algorithms as its special cases. This algorithm is called a ratio-weighting algorithm. Our approach is based on transfers of weights from nodes of a given tree to a new node which is added. This algorithm is applied both statically and dynamically like other algorithms. The computational complexity of static ratio-weighting algorithm is O(n+w), where n is a number of nodes and w is a maximum number of brothers. This result is as same as that of static top-down and static bottom-up algorithms.

１．はじめに

木構造はものの集まりに階層関係を与える構造であり，数多くの分野で用いられている[1,2]．この中で，木の節点に対して一定の規則で重みを割り当てることで，与えられた問題を自然にモデル化できる場合がある．このような規則としては，一般にトップダウン法とボトムアップ法がよく知られている．しかし，これらの方法は応用範囲が狭く，対象とする問題を必ずしもうまく表現できないことがある．

例えば，図１．１に示すようなある会社組織において，あるセクションを課として独立させることを考える．ここで，各節点に付けられている重みは，その部や課に所属する人員や設備を何らかの方法で数値化したものとする．

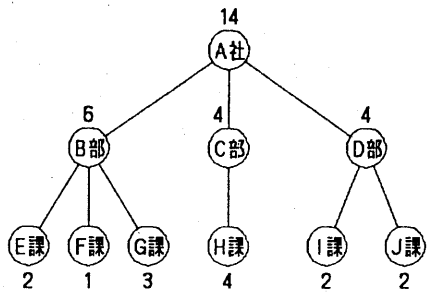
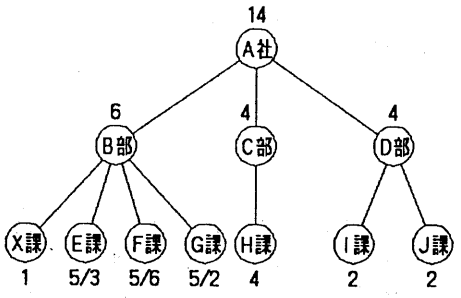
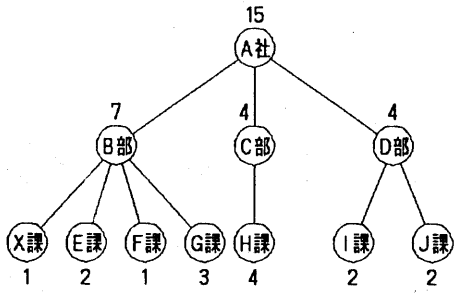


図１．１　重み付き木の例

いま，Ｂ部に新しく重み１のＸ課を設立する．Ｘ課をＢ部の中の人員や設備の移動だけで新設することは，トップダウン法に相当する．また，Ｘ課をＢ部の既存の課に全く変更を加えずに新設することは，ボトムアップ法に相当する．これらの結果を図１．２に示す．



(a) トップダウン法



(b) ボトムアップ法

図１．２　既存の重み付け法

ところが現実には，新しい課は同じ部に属する既存の課から人員や設備を移すことによって設立するとしても，新しい課が設立されると，その課独自の人員や設備が増えることが多い．したがって，会社全体の人員や設備は，１つの課を完全に新たに設立するほどではないが，若干増すことになる．このような機構をモデル化することを考える．そこで，新設するＸ課の重み１のうち，0.5は同部内の既存の課から移動し，残り0.5は新たに増加するものと仮定すると，図１．３のような結果が得られる．

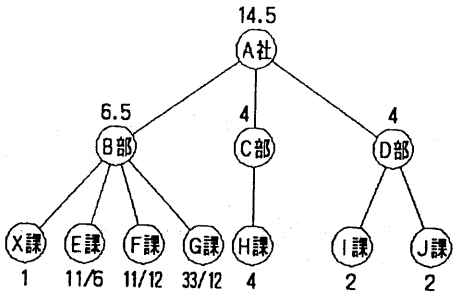


図１．３　比を用いた重み付け

ここで，上に挙げた方法を比較する．トップダウン式に重みを割り当てた時は，新たな子の影響は兄弟に伝わり，親の節点の重みは変化しない．つまり，新たな子の重みの影響は 0 : 1 の比率で親と兄弟に伝わるといえる．同様に，ボトムアップ式に重みを割り当てたときは，新たな子の影響は全て親に伝わり，兄弟の節点の重みは変化しない．つまり，新たな子の重みの影響は 1 : 0 の比率で親と兄弟に伝わる．

本研究ではこの重みの移動という性質を利用し，両者を一般化したアルゴリズムを考える．このアルゴリ

ズムは比付き重み付けアルゴリズムといい，新たな子の重みが r : (1-r) の比率で親と兄弟に伝わるものである．

また，木に重み付けをする際，完成した木に対して一度に重み付けをする方法と，木を成長させながら重み付けを繰り返す方法の２つを考える．前者を静的な重み付け，後者を動的な重み付けと呼ぶ．動的な重み付けは，上の課の新設の例などに用いた場合に，効率の点などにおいて，静的な重み付けよりも有利である．

本稿では，これらのアルゴリズムと，その対象となる木の性質との関係について述べる．

まず，静的なアルゴリズムとしてトップダウンアルゴリズムとボトムアップアルゴリズムを定義する．次に，これらを動的に行うことを考え，その方法を定義する．また，トップダウンアルゴリズムとボトムアップアルゴリズムを一般化した，比付きの重み付けを定義する．比付きの重み付けは，結果が木の成長の順序に依存する．そこで，木の成長の順序の１つとして横優先成長を定義する．そして，横優先成長の順における比付きの重み付けに等しく，しかも結果が木の成長順に依存しないような修正アルゴリズムを定義する．ただし，修正前のアルゴリズムが持つ「重みの移動」というイメージは失われる．

本稿で扱うアルゴリズム間の関係を図１．４に示す．図中の同じアルファベット同士は，同じ結果が得られることを示す．また，×印は，アルゴリズムが存在しないことを示す．

		動　的		静的
木の成長の順序		横優先	ランダム	－－
一般	トップダウン	a	a	a
	ボトムアップ	b	b	b
比付き	修正前	c	d	×
	修正後	c	c	c

図１．４　アルゴリズム間の関係

２．準備

節点と呼ばれる要素の集合に対して，階層的な(親子の)関係を与えたものを木という．節点のうちの１つを根と呼んで，他の節点と区別する．ｒを節点とし，$T_1, T_2, \cdots, T_k$が木であって，これらの根が$r_1, r_2, \cdots, r_k$であるとする．このときｒを$r_1, r_2, \cdots, r_k$のすべての親とすると，新しい木が得られる．この木ではｒが根で$T_1, T_2, \cdots, T_k$はこの根の部分木である．節点$r_1, r_2, \cdots, r_k$は節点ｒの子と呼ばれる．節点ｒを根とする部分木を$T_r(r)$，節点ｒの子の集合をｓｏｎ(ｒ)で表す．

$n_1, n_2, \cdots, n_k$が木の中の節点の列であって，$1 \leqq i < k$に対してn_iがn_{i+1}の親になっているとき，この列のことを節点n_1から節点n_kへの経路と呼ぶ．経路の長さ(距離)は，その経路に含まれる節点の数から１を引いた値である．

節点ａから節点ｂへの経路が存在するとき，ａはｂの先祖であるといい，ｂはａの子孫であるという．どの節点も自分自身の先祖であり，子孫でもある．自分自身以外の先祖や子孫を，真の先祖，真の子孫と呼ぶ．真の子孫をもたない節点を葉と呼ぶ．

木の中で，ある節点から葉への最長経路の長さを，その節点の高さと呼ぶ．木の根の高さをその木の高さという．根からある節点までの経路の長さを，その節点の深さという．

同じ親をもつ節点同士を兄弟と呼び，左を兄，右を弟とする．この順序づけを，先祖・子孫の関係のない節点に対しても，ａがｂの兄であれば，ａの子孫はすべてｂの子孫よりも左にあると拡張する．兄弟の中で最も左にある節点を長男と呼ぶ．また，ある節点のすぐ右隣にある兄弟を，その節点の次弟と呼ぶ．

図２・１のように，子の順序が違う木は，異なる木として扱う．これを順序木という．特に子の順序を無視した木は無順序木と呼ぶ．

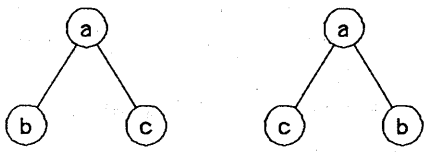


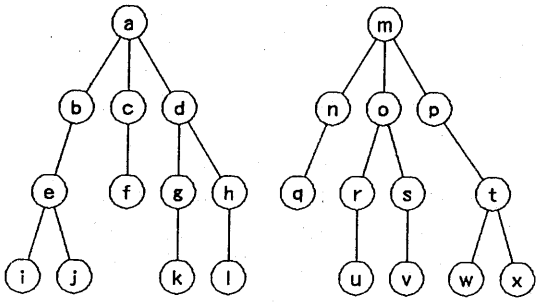
図２・１　異なる(順序)木

木の節点に一定の規則で重みを与えることがある．このとき，節点ｒの重みをｗ(ｒ)で表す．

次に，同型の概念を定義する．節点r_1，r_2を根とする２つの部分木$T_r(r_1)$，$T_r(r_2)$が同型であるとは，ｓｏｎ(r_1)からｓｏｎ(r_2)への，１対１で上への写像ｆが存在し，かつ全てのｓｏｎ(r_1)の要素ｘについて，$T_r(x)$と$T_r(f(x))$が同型になることである．ただし，単独の葉同士は同型と定義する．$T_r(r_1)$と$T_r(r_2)$が同型であることを，$T_r(r_1) \sim T_r(r_2)$と書く．同型な木の例を図２．２に示す．

同型の定義にしたがって対応した節点同士を，同型

木対応節点という．図２．２において，ｃとｎ，ｇとｓなどが同型木対応節点である．



(a)　　(b)

図２．２　同型な木の例

１つの木に属する２つの葉ℓ_1, ℓ_2に着目する．これらの葉に共通な先祖のうち，最も深い節点をｐとする．ｓｏｎ(p)を根とする部分木の集合のうち，ℓ_1を含むものとℓ_2を含むものが同型で，かつℓ_1, ℓ_2が同型木対応節点であるとき，ℓ_1とℓ_2は**対称葉**であるという．図２．２(a)において，ｉとｊ，ｋとｌなどが対称葉である．

次の性質をもつ木を，**対称重み付き木**と呼ぶ．

[定義２．１]　対称重み付き木

①各々の葉には，$0 < g < \infty$であるような重みｇが与えられており，全ての葉の重みをその最大値で割ることで正規化ができる[正規性]．

②各節点の重みは，子の節点の重みの和である[加法性]．

③対称葉は同じ重みをもつ[対称性]．

３．トップダウン，ボトムアップアルゴリズムとその一般化

３．１　静的なトップダウン，ボトムアップアルゴリズム

ある完成した木に対して重み付けを行うことを，**静的な重み付け**と呼ぶ．与えられた木に対する静的な重み付けのアルゴリズムとして，**トップダウンアルゴリズム**と**ボトムアップアルゴリズム**を次の様に定義する．

[定義３．１]　トップダウンアルゴリズム

根に１の重みを与え，全ての子は親の重みを等しく分配する．

[定義３．２]　ボトムアップアルゴリズム

全ての葉に同じ重み$(0 < g < \infty)$を付ける．親の重みは全て子の重みの和とする．

節点の数をｎとするとき，トップダウンアルゴリズムは，Ｏ(n)回の除算によって木に重み付けを行う．また，ボトムアップアルゴリズムは，Ｏ(n)回の加算のよって木に重み付けを行う．

ボトムアップアルゴリズムによって重みを付けた木を**ボトムアップ木**，トップダウンアルゴリズムによって重みを付けた木を**トップダウン木**と呼ぶ．これらの木は，対称重み付き木になる．

[定理３．１]　トップダウン木は対称重み付き木である

[定理３．２]　ボトムアップ木は対称重み付き木である

３．２　動的なトップダウン，ボトムアップアルゴリズム

完成した木に重み付けを行うことを静的な重み付けと呼んだのに対して，木を成長させながら，逐次重み付けを行っていくことを，動的な重み付けと呼ぶ．今，重みg_1の節点n_1の下に，節点n_2を子として追加する場合を考える．n_2がn_1のｉ番目の子であるとき，n_2に対する動的な重み付けの方法として，次の２つを定義する．

[定義３．３]　動的なトップダウン式の重み付け

n_2およびその全ての兄弟の重みを，

$$g = g_1 / i$$

とする．n_2の兄弟の全ての真の子孫の重みには，$(i-1)/i$を乗ずる．

[定義３．４]　動的なボトムアップ式の重み付け

n_2の重みは，g_1やｉに無関係に一定値ｇ$(0 < g < \infty)$とする．n_2の真の先祖の重みは，全てｇを加える．

動的にトップダウン式の重み付けを行った木とボトムアップ式の重み付けを行った木は，それぞれ定義より明らかにトップダウン木，ボトムアップ木になる．

３．３　分配の一般化

ボトムアップ式の重み付けでは，節点ｘの子として節点ｙを付け加えると，ｘの重みだけが変化して，ｙの兄弟の重みは変化しない．また，トップダウン式の重み付けでは，ｘの重みは変わらずに，ｙの兄弟の重みだけが変化する．言い換えると，前者では新しい節点の重みは全てその親からもらい，後者ではそれを全て兄弟からもらうと考えられる．そこで，これらの一

般化として，新たな節点の重みを，兄弟と親からそれぞれ r : (1-r) の割合(但し0≦r≦1)でもらうような重み付けの方法を考える．これを動的な比付きの重み付けと呼ぶ．

[定義３．５] 動的な比付きの重み付け
重みがg_1である節点n_1に，i番目の子として節点n_2を追加する．
この時，n_2には
$$g' = g_1/(i-1+r) \qquad (0 \leqq r \leqq 1)$$
の重みを与える．ただし，長男にはその親と同じ重みを与える．また，n_2の兄弟の重みからはそれぞれ，$(g' \times r)$を減じる．

動的に比付きの重み付けを行った木は，節点を追加する順序によって，結果が異なることがある．図３．１にその例を示す．この例は根の重みの初期値を１とし，r＝１／２とした場合を示している．

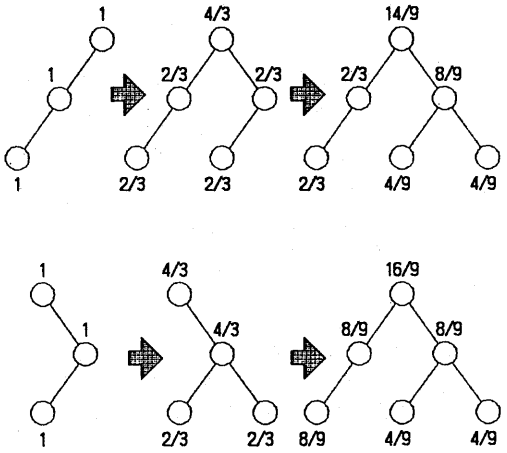


図３．１　節点の重みが成長の順序に依存する例

この不都合を解決するために，まず木の成長の順序を１つ定める．そして，その順序に基づいて比付きの重み付けを行った結果が，成長順によらず常に得られるようなアルゴリズムを考える．

４．木の成長の順序

４．１　横優先成長アルゴリズム

節点ｎの子孫のうち，最も左下にある葉ℓを，ｎの最高継承者と呼ぶ．木の生成の際，なるべく早く最高継承者を作るような順番で節点を追加していくような成長の方法を，横優先成長と呼ぶ．横優先成長の順に節点に番号を付けた木の例を図４．１に示す．

[定義４．１] 最高継承者
if 与えられた節点が子を持つ
then その節点の長男の最高継承者
else その節点自身

[定義４．２] 横優先成長
与えられた節点の全ての子について左から順に
if 子の最高継承者がまだいない
then 最高継承者を作る
与えられた節点の全ての子について
if 子が子を持つ
then 横優先成長を適用する

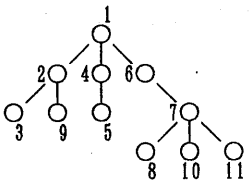


図４．１　横優先成長

横優先成長の順に木を生成しながら節点に重み付けを行う，動的な重み付けのアルゴリズムを，横優先成長アルゴリズムと呼ぶ．

[定義４．３] 横優先成長アルゴリズム
与えられた木の節点を横優先成長の順に追加し，重みを付ける．重みg_1の節点n_1の下に，i番目の子として節点n_2を追加する時，n_2の重みはg_1とiによって決定する．また，このときn_1の重みとn_2の兄弟およびその子孫の重みも変化してよいとする．

横優先成長の順では，節点を追加する時点において，その節点の兄弟の持つ子の数は高々１である．したがって，追加した節点の，兄弟への影響は，兄弟の最高継承者への影響に等しい．
横優先成長アルゴリズムに関して，次の定理が成立する．

[定理４．１] 横優先成長アルゴリズムが対称重み付き木を生成する十分条件は，節点を追加した結果全ての兄弟の重みが等しくなり，かつ正規性と加法性を保存することである

(証明) 横優先成長アルゴリズムでは，節点ｎを追加

したとき，重みが変化するのはｎの兄弟とｎの先祖だけである．このことと正規性，加法性によって，同じ重みの根から同型木を作成すると，対応する節点は全て同じ重みになる．したがって，節点の追加時にその節点と兄弟の重みが等しくなるのであれば，以下同型に成長した部分木の同型木対応節点は全て同じ重みになる．すなわち，生成される木は対称性を持つ．また，仮定よりこの木は正規性と加法性を持つので，対称重み付き木の性質を満足する．　(証明終)

横優先成長アルゴリズムを適用する際の重み付けの方法によって，トップダウン横優先成長アルゴリズム，ボトムアップ横優先成長アルゴリズム，比付き横優先成長アルゴリズムが定義できる．

[定義４．４]　トップダウン横優先成長アルゴリズム
木を横優先成長の順に成長させながら，トップダウン式の重み付けを行う．

[定義４．５]　ボトムアップ横優先成長アルゴリズム
木を横優先成長の順に成長させながら，ボトムアップ式の重み付けを行う．

[定義４．６]　比付き横優先成長アルゴリズム
木を横優先成長の順に成長させながら，比付きの重み付けを行う．

トップダウン横優先成長アルゴリズムは，定義より明らかに，Ｏ(ｎ)回の除算によってトップダウン木を生成する．同様に，ボトムアップ横優先成長アルゴリズムも，Ｏ(ｎ)回の加算によってボトムアップ木を生成する．比付き横優先成長アルゴリズムに関しては，次の定理が成立する．

[定理４．２]　比付き横優先成長アルゴリズムでは，新たに追加した節点の重みと，変化後の兄弟の重みは全て等しくなる

(証明)　追加する節点の数に関する帰納法による．
節点g_1の子の数をｎとする．

i) ｎ＝１のとき，明らかに成り立つ
ii) ｎ＝ｋの時，全ての節点の重みが等しいとする．
この時，新たに節点を増やしてｎ＝ｋ＋１にしても，全ての節点の重みが等しくなることを示す．
今，全ての節点の重みが等しくｇであるとすると，それらの親の節点g_1の重みは

$$g_1 = g \times k$$

となる．また，新たに追加するｋ＋１番目の節点の重みｇ'は

$$g' = g_1/(k+r) = g \times k/(k+r)$$

であり，兄弟の重みｇ"はそれぞれ

$$g'' = g - g' \times r/k$$

となる．よって，

$$g'' = g - g \times k/(k+r) \times r/k = g \times k/(k+r) = g'$$

したがって，全ての兄弟の重みは等しくなる．
(証明終)

なお，親の新しい重みg_1'は

$$g_1' = g' \times (k+1) = g_1/(k+r) \times (k+1) = g_1 + g_1/(k+r) \times (1-r) = g_1 + (1-r)g'$$

となり，増えた節点の重みの(１－ｒ)倍が親の重みの増加分になっている．

[定理４．３]　比付き横優先成長アルゴリズムは対称重み付き木を生成する

(証明)　定理４．１および定理４．２より明らか．

比付き横優先成長アルゴリズムは，ｒ＝０のとき動的なボトムアップの重み付けアルゴリズムに等しくなり，ｒ＝１のとき動的なトップダウンの重み付けアルゴリズムに等しくなる．すなわち，比付き横優先成長アルゴリズムは，両者の一般化となっている．

４．２　比付きの重み付けの修正

一般に，子の増加が親の節点の兄弟の重さの決め方に影響を及ばさないようなアルゴリズムによる重み付けの結果は，木の成長順序に依存しない．そこで，比付きの重み付け方法を次の様に修正する．

[定義４．７]　修正された比付きの重み付け
重みがg_1である節点n_1に，ｉ番目の子として節点n_2を追加する．この時，n_2には

$$g' = g_1 \times {}_1R_r \qquad (0 \leq r \leq 1)$$

の重みを与える．また，n_2の兄弟の重みには，

$$(i-1)/(i-1+r)$$

を乗ずる．ここで，${}_iR_r$は兄弟比(長男の最初の重みに対するｉ番目の兄弟の重みの比)を表す．

$$ {}_iR_r = (i-1)!/(i-1+r)! \qquad (0 \leqq r \leqq 1) $$

なお，ここで用いている階乗の定義は実数に拡張されている．すなわち，

$$ i! = \begin{cases} i \times (i-1)! & (i > 1 \text{の時}) \\ 1 & (i \leqq 1 \text{の時}) \end{cases} $$

である．

[定義４．８] 修正比付き横優先成長アルゴリズム
木を横優先成長の順に成長させながら，修正された比付きの重み付けを行う．

[定理４．４] 修正比付き横優先成長アルゴリズムは比付き横優先成長アルゴリズムと同じ結果を与える

(証明) 比付きの横優先成長アルゴリズムの親の重さをＧ，兄弟の重さをｇ，新たに付け加える兄弟の重さをｇ'，変化後の兄弟の重さをｇ"とする．一方，修正比付きの横優先成長アルゴリズムでの，最初の親の重み(＝長男の重み)をＨ，兄弟の重さをｈ，新たに付加える兄弟の重さをｈ'，変化後の兄弟の重さをｈ"とする．このとき，常に

$$ g" = g' = h' = h" $$

が成立する事を示せばよい．定理４．２により，

$$ g" = g' $$

は既に示してあるので，

$$ g' = h' = h" $$

を，兄弟の数に関する帰納法によって示す．

i) ｎ＝１の時

Ｇ＝Ｈ＝ｇ＝ｈより，成立

ii) ｎ＝ｋの時成立すると仮定して，ｎ＝ｋ＋１の時にも成立することを示す．仮定より，

$$ G = k \times g, \quad g = h, \quad h = H \times {}_kR_r $$

が成立する．ｋ＋１番目の兄弟の重さｇ'，ｈ'は，

$$ g' = G/(k+r) $$
$$ h' = H \times {}_{k+1}R_r $$

となる．したがって，

$$ \begin{aligned} g' &= k \times g/(k+r) \\ &= h \times k/(k+r) \\ &= H \times {}_kR_r \times k/(k+r) \\ &= H \times {}_{k+1}R_r \\ &= h' \end{aligned} $$

一方，

$$ \begin{aligned} h" &= h \times k/(k+r) \\ &= H \times {}_kR_r \times k/(k+r) \\ &= H \times {}_{k+1}R_r \\ &= h' \end{aligned} $$

よって

$$ g' = h' = h" $$

以上より，修正比付き横優先成長アルゴリズムは常に比付き横優先成長アルゴリズムと同じ結果を与える．
(証明終)

次に，修正された比付きの重み付けを用い，かつ木の成長順序を制限しないアルゴリズムを定義する．

[定義４．８] 修正比付きランダム成長アルゴリズム
木にランダムに節点を追加しながら，修正された比付きの重み付けを行う．

[定理４．５] 修正比付きランダム成長アルゴリズムは対称重み付き木を生成する

(証明) 修正された比付きの重み付けでは，新たに節点を付加するときの付けかたは，兄弟の大きさに影響されない．したがって，修正比付きランダム成長アルゴリズムによる重み付けの結果は，木の成長順序に依存せずに，修正比付きの横優先成長アルゴリズムによる結果と一致する．一方，定理４．４より，修正比付きの横優先成長アルゴリズムは比付きの横優先成長アルゴリズムと同じ結果を与えるので，修正比付きランダム成長アルゴリズムによって生成される木は対称重み付き木となる．
(証明終)

次に，修正比付きランダム成長アルゴリズムと同じ結果を与える，静的なアルゴリズムを定義する．

[定義４．９] 修正比付きアルゴリズム
根から葉ℓへの経路の上の節点での分岐数を順に

$$ \{Wi\} \qquad (1 \leqq i \leqq d) $$

とする．但しｄは葉ℓの深さである．この時，葉ℓの重みを

$$g = \prod_{i=1}^{d} w_i R_r$$

で定義する．

[定理４．７] 修正比付きアルゴリズムは修正比付きランダム成長アルゴリズムと同じ結果を与える

(証明) 追加する節点の数に関する帰納法による．節点g_1の子の数をｎとする．

1) ｎ＝１のとき，明らかに成り立つ．
2) ｎ＝ｋのとき，定理が成立しているとする．ここで，深さｆにある節点に新しい節点を追加してｎ＝ｋ＋１としたときを示す．
①追加した節点の重み
追加した節点の重みｇは

$$g = g_1 \times w_f R_r$$

である．ところが

$$g_1 = \prod_{i=1}^{f-1} w_i R_r$$

よって，

$$g = \prod_{i=1}^{f} w_i' R_r$$

となる．
②兄弟の子の葉(深さをｄとする)
兄弟の子の葉の現在の重みをｇ'とすると，新しい重みｇ"は

$$g'' = g' \times Wf / (Wf - r)$$

となる．ところが

$$g' = \prod_{i=1}^{d} w_i R_r$$

であるから，

$$g'' = \prod_{i=1}^{d} w_i R_r \times Wf / (Wf - r)$$

$$= \prod_{i=1}^{d} w_i' R_r$$

ただし，

$$Wi' = \begin{cases} Wi & (i \neq f) \\ 1 + Wi & (i = f) \end{cases}$$

(証明終)

修正比付きアルゴリズムは，ある節点の重みを決定するのに，トップダウンアルゴリズムやボトムアップアルゴリズムに比べて多くの計算が必要となる．ただしそれは，すでに計算済みの値に，別に計算する値$w_i' R_r$を乗ずるだけであり，その計算量は，実際にはＯ(ｎ)で済む．ただし，ｎは節点の数を表す．一方，$w_i R_r$の計算量はＯ(Ｗｉ)であるが，同様に過去に求めた値を再利用できる．すなわちその計算は，$w_{max} R_r$の値を計算する時の途中結果を表に保存しておき，これを利用することによって，高々１回の表の参照で済む．ここでw_{max}は兄弟の数の最大値を表す．$w_{max} R_r$の計算には，w_{max}回の乗算が必要であるから，全体ではＯ$(n + w_{max})$回の乗算が必要となる．ｎがw_{max}に比べて十分に大きい時は，計算量はＯ(ｎ)と見なしても差し支えない．

５．おわりに

本論文では，まず，対称重み付き木を定義し，これに対して，静的な重み付けと動的な重み付け，トップダウン重み付けとボトムアップ重み付けの各々を定義した．そして，これらを含み，かつ成長順に依存しないアルゴリズムとして，修正比付き(ランダム成長)アルゴリズムを提案した．

今後の拡張として以下の２点を考えている．
①本論文においては，木構造を取り扱ったが，これを有向非サイクルグラフ(ＤＡＧ)へ拡張すること．
②木の重み付けのアルゴリズムがどれだけ多くの木の形を区別できるかによって，アルゴリズムの強弱を定義すること．すなわち，対称重み付き木では，２つの木T_1，T_2が同型であるとき，対称性より，対応する葉同士の重みは同じになる．このことから逆に，T_1とT_2の対応する葉の重みが同じであるかどうかによって，これらの木の形に関する同値関係を定義できる．このように，木の重み付けのアルゴリズムは，この同値関係を定義していると考えられる．

参考文献

[1] A.V.エイホ他著，大野義夫訳「データ構造とアルゴリズム」培風館 (1987)

[2] C.ベルジュ著，伊理他訳「グラフの理論」サイエンス社 (1976)