Vol.2022-HCI-196 No.5
2022/1/11

根付き木の空間効率の良い一方向レイアウト

三末 和男[1,a)]　中島 悠介

概要：根付き木を視覚的に表現する際には、連結図の一方向レイアウト、特に階層レイアウトが用いられることが多い。ただし、しばしば空間効率が悪いレイアウトになるという問題がある。特に同世代の子孫を水平に配置するレイアウトでは、全体形状が横長になる傾向にあり、描画領域のアスペクト比によっては、大幅に空間効率が悪くなる。この論文では、このような問題に対して開発された、一方向という条件は維持しつつも、水平線上に配置するという条件を緩めるレイアウト手法が紹介される。このレイアウト手法は、水平線ではなく、蛇行した（時には垂直な）線上にノードを配置するもので、「一方向局所蛇行レイアウト」とよばれる。レイアウトアルゴリズムとともに、空間効率および計算時間に関する評価結果が示される。

1. はじめに

「木」とは閉路のない連結グラフである。「根」とよばれるノードがひとつ定められている木を「根付き木」とよぶ。根付き木は、ファイルの管理構造や組織構造など階層的な構造の抽象的な表現として、しばしば使用される。根付き木の視覚的な表現には、連結図や領域図など様々な形態があるが、一つの代表的な形態が、ノードを点で、エッジを線で表す連結図である。隣接する、すなわちエッジでつながれた、二つのノードは親子関係にあると言い、根に近い方が「親」、逆が「子」とよばれる。親子関係の向きはエッジの向きの視覚的な表現によって表すことができる。たとえば、エッジが親から子に向っているとすれば、エッジを表す線の子側の端点に矢じりをつけることで、親子関係の向きを表すことができる。それ以外にも、親と子の位置関係で表すことも可能である。たとえば、子を親よりも下に配置することにすれば、エッジを表す線に矢じりなどつけなくても、親子関係の向きを読み取ることができる。このように親子関係の向きを、一方向の向きで表した配置を「一方向レイアウト」とよぶ。一方向レイアウトは、矢じりのようなエッジの装飾よりも、根付き木の表す、根から葉に向う（あるいは葉から根に向う）大局的な構造を把握しやすいという特徴がある。

根付き木の一方向レイアウトはしばしば空間効率が悪い。たとえば、下向きの一方向レイアウトであれば、根付き木の全体は横長の図になることが多く、縦横比を維持して、描画領域に収めるためには、全体を縮小する必要がある。

本研究の目的は、指定されたアスペクト比の描画領域に対して、根付き木を空間効率の良い図として描くことである。そのための技術的課題は二つあり、第 1 は根付き木の空間効率の良いレイアウトを設計すること、第 2 は空間効率のよいレイアウトを生成するためのアルゴリズムを構築することである。ここで言う、「空間効率」とは、レイアウトに要する面積の小ささではなく、指定された描画領域に効率良く収められることを意味する。つまり、レイアウトに要する面積が同じであっても、一方は指定された描画領域に収めるために縮小が必要であれば、縮小が必要でないものよりも空間効率は悪いとする。

根付き木のレイアウトに関してはすでに多くのアルゴリズムが開発されている [1]。一方向レイアウトの代表は、階層レイアウトとよばれるもので、ノードは、平行に配置された（描かれない）水平線上に配置される。同レベルのノード群が同じ水平線上に配置されることで、親が違っても、同じ世代のノードはひと目で把握することができる。根付き木は下の世代に進むほどノードの数が増えるため、同じ世代をひとつの水平線上に配置すると、どうしても横に長い図になってしまう。上記の技術的課題に対して筆者らが考案した手法は、一方向という条件は維持しつつも、水平線上に配置するという条件を緩めるものである。水平線ではなく、蛇行した（時には垂直な）線上に葉群を配置することから、「局所蛇行レイアウト」とよぶことにした。

2. 関連研究

根付き木の表現手法としては様々な形態が開発されている [1]。根付き木に限らず、グラフの代表的な表現手法

[1] 筑波大学
Uniersity of Tsukuba
[a)] misue@cs.tsukuba.ac.jp

は連結図[*1]であろう。ノードを点あるいは点に準じる図形で、エッジを点をつなぐ線分で表す。オイラー図をベースにする領域図も利用されており、包含規約[*2]とよばれている [2]。その代表である Treemap[3] は近年 Office ツールにも組込まれるなど一般化してきている。連結図と領域図のハイブリッドとも言える表現手法も開発されている。Elastic hierarhies は連結図とツリーマップを組み合わせることで、両方の利点を得ようとした [4]。Hi-tree は、子との関係を連結で表すノードと包含で表すノードの、2 種類を導入することで表現力を向上させた [5]。本論文では連結図に焦点を合わせる。さらに、補足的ではあるが、ハイブリッドの活用も視野に入れる。

2.1 連結図による根付き木の描画手法

連結図による根付き木の表現手法にもさまざまな種類がある。放射状レイアウトは、根を中心に配置し、深さごとにノードを同心円上に配置する手法である [6]。H-V レイアウトは、根付き木のうち、2 分木を対象とした表現手法である [7]。エッジが水平線分または垂直線分で構成されており、親子関係も左右関係（左が親）または上下関係（上が親）で表現する。H-V レイアウトは 2 分木を対象とするが、一般の根付き木の親子を上下だけでなく左右にも配置する手法としては、横転規約[*3]が開発されている [2]。

2.2 根付き木の一方向レイアウト

Wetherell ら [8] と Reingold ら [9] は、2 分木を一方向レイアウトで描画するアルゴリズムを開発した。彼らは、レイアウトに関して、エッジ同士が交差しない、任意の親子ノードについて親ノードの方が根ノードに近いという条件に加えて、以下のような美的基準を定めた。

(1) 同じ深さのノードは同一直線上に配置し、それぞれの直線は平行である
(2) 2 分木の場合、左の子ノードは親の左側に、右の子ノードは親の右側に配置する
(3) 親ノードは子ノードの中央に配置する
(4) 同じ構造の部分木は同じ形で描画し、対称な構造のものは鏡像で描画する

美的基準 (1)〜(3) は Wetherell らによって設定されたものであり、Reingold らはそれらに (4) を追加した。一方向レイアウトのなかでも、美的基準 (1) に従うものは「階層レイアウト」とよばれる。

Walker はこれらの美的基準を満たしつつ、対象を一般の根付き木に拡張した [10]。このアルゴリズムでは美的基準を満たすと同時に、描画幅を最小限に抑えることも可能にした。Buchheim らはさらに、2 乗実行時間であった Walker のアルゴリズムを、線形実行時間に改善した [11]。

2.3 空間効率の改善

根付き木の描画手法の進化系統の一つは、可読性の向上を目指して美的基準を充実させる向きに進んできた。その一方で、空間効率に着目し、美的基準を緩和する向きにも進んでいる。

Wetherell らは、先に示した美的基準に加えて、できる限り描画幅を小さくするのが良いと述べており、美的基準 (3) と描画幅最小化の優先順位を変更した 2 つのアルゴリズムを提示した。Marriott らも、美的基準 (3) を緩和することで、占有領域の横方向の圧縮を試みた [12]。van der Ploeg は美的基準 (1) を緩和し、同じ深さでも親が異なるノードは同一直線上に配置しないことを許容した [13]。これにより、高さの異なるノードがあった場合でも無用な余白が発生せず、より空間効率よく描画できるようになった。

対象が根付き木ではなくグラフであるが、視覚的表現を表示領域のアスペクトに合わせるために、連結図を大局的に蛇行させる手法も開発されている。Sonke ら [14] は線形レイアウトを対象に、Rüegg ら [15] は有向階層レイアウトを対象に、蛇行レイアウトを開発してる。

3. 問題定義

根付き木を $T = (V, E)$ のように表す。このとき V はノードの集合、E はエッジの集合であり、$E \subset V \times V$ である。根を r で表す。$r \in V$ である。$(u, v) \in E$ のとき、ノード u を v の親とよび、ノード v を u の子とよぶ。ノード $v \in V$ の位置を $p(v)$ で表す。x 座標や y 座標を表すときには、$p_x(v)$ や $p_y(v)$ のように添字を付けることにする。ノード $v \in V$ の幅と高さを、$w(v)$ と $h(v)$ で表す。

3.1 アルゴリズムの入出力

筆者らが構築を目指すアルゴリズムの入出力は以下の通りである。

入力

- 描画領域のサイズ: 描画領域は各辺が水平または垂直な長方形とし、その横と縦の長さを入力とする。
- 根付き木 $T = (V, E)$
- 各ノード $v \in V$ のサイズ ($w(v)$,$h(v)$)
- レイアウトのパラメータ（ノードの最低左右間隔 s_h、最低上下間隔 s_v）

出力

- 各ノード $v \in V$ の位置 $p(v)$
- 各エッジの配線（折れ線による表現）

3.2 空間効率

アルゴリズムが目指す空間効率を、「配置の決定された根付き木の全体を、指定された描画領域内にちょうど収め

[*1] node-link diagram
[*2] inclusion convention
[*3] tip-over convention

るための拡大率」として定義し、大きいほど空間効率が良いとする。描画領域内に収めるために拡大ではなく縮小が必要となる場合もあるが、縮小の場合は 1 以下の拡大率だとし、本論文では、拡大・縮小に関わらず「拡大率」で統一することにする。

3.3 制約条件

なお、問題を簡単にするために、ここでは、すべての分岐ノードのサイズは均一とする。また、すべての葉のサイズも均一とする。

4. 一方向局所蛇行レイアウト

空間効率を高めるために、van der Ploeg は親が異なる葉は同一水平線上に並ばなくてもよいとした [13]。筆者らは、さらに、蛇行配置を許し、親が同じ葉であっても必ずしも水平線上に並んでいなくても良いとした。図 1 に蛇行レイアウトの例を示す。図 1(a) と (f) の例では葉は蛇行はしていないが、これらも含めると、同じ根付き木を 6 種類のアスペクト比で描けることが分る。

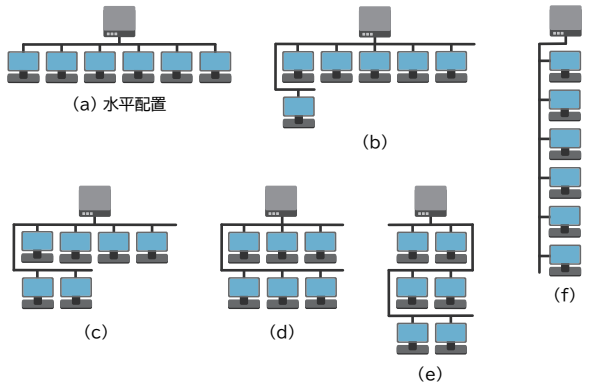


図 1 蛇行レイアウトの例

4.1 基本的な処理の流れ

一方向局所蛇行レイアウトの基本的な処理の流れは以下の通りである。

(1) 共通の親をもつ葉群をダミーノードに代表させる
(2) ダミーノードに置き換えられた葉群の相対位置を求める（求め方は後述）
(3) 葉群をちょうど含む長方形の大きさをダミーノードの大きさとする
(4) 一部の葉がダミーノードに置き換えられた根付き木を、van der Ploeg のアルゴリズムでレイアウトする
(5) ダミーノードに置き換えられた葉群の絶対位置を求める

ステップ (2) で求める葉群の相対位置は、蛇行のさせ方、つまり図 1 に示したようなバリエーションによって異なる。ここでは、指定された列数に応じて、（葉が収まる）長方形を並べることにする。図 1(a)、(b)、...、(f) は、葉群を列数 6、5、...、1 で並べたものである。

4.2 目標とするアスペクト比への適合のさせ方

共通の親をもつ葉群をそれぞれダミーノードに置き換えた根付き木の、全体のアスペクト比は、各ダミーノードの形によって異なる。つまり、全体のアスペクト比は、ダミーノードに置き換えられた葉群のレイアウト、さらに言うと、葉群のレイアウトを決める列数によって変えることができる。目標とするアスペクト比に適合させるための手法としては、まずは素朴な手法を採用した。それは、アスペクト比（幅/高さ）が最も小さい（縦長の）状態から始めて、目標とするアスペクト比 a_t 以上になるまで、徐々に葉群の列数を増すというものである（図 2 参照）。この処理を手続き的に示すと Algorithm 1 のようになる。

Algorithm 1 Local Folding Layout

Input: $T = (V, E)$ – 根付き木
Input: a_t – 目標とするアスペクト比
Output: V の全要素の位置
1: 共通の親をもつ葉群をダミーノードに代表させる
2: すべてのダミーノード内を列数 1 で配置する
3: 根付き木全体をレイアウトし、アスペクト比 a を求める
4: **while** $a < a_t$ **do**
5: 底辺が最も下にあるダミーノード d を探す
6: ダミーノード d 内の葉群の列数を 1 増やす
7: 根付き木全体をレイアウトし、アスペクト比 a を求める
8: **end while**

なお、4 行目〜8 行目の while 文の終了条件には、実際には、通常のノードが最も下にある場合や、対象葉群の列数をそれ以上増やせない場合も含める必要がある。ただし、それらの条件で while 文を抜ける場合には、目標とするアスペクト比に到達できない。

4.3 ダミーノード内の葉群の配置

ダミーノード d に代表される葉群を $(v_0, v_1, \ldots, v_{n-1})$ とする。列数 m が指定されたときには、1 行にノードを m 個ずつ指定された間隔を空けながら左から上揃えで並べる。次の行に並べる際には上の行の下部から一定の間隔を空けて、左から上揃えで配置する。

ノード v_i の相対位置は、葉のサイズを $w \times h$、ノード間の横方向の間隔を s_h、縦方向の間隔を s_v とすると、式 (1) と (2) のように表される。なお、位置の表記で p^r のように右肩に r を付けたのは、相対位置であることを示すためである。

$$p^r_x(v_i) = (w + s_h)(i \bmod m) \quad (1)$$
$$p^r_y(v_i) = (h + s_v)\lceil i/m \rceil \quad (2)$$

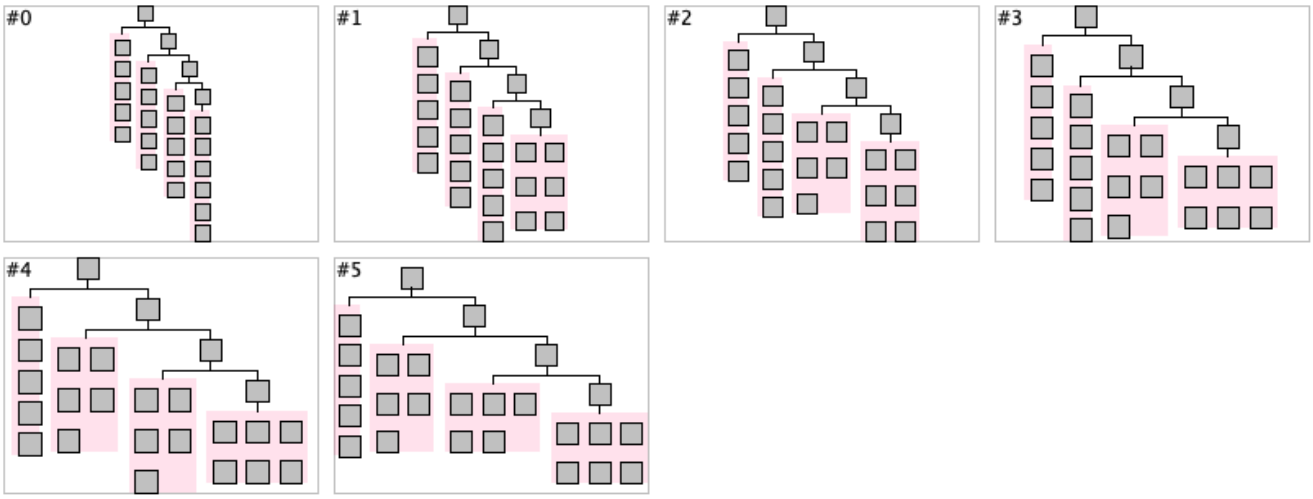


図 2　目標アスペクト比に近付ける過程の例（ピンクの長方形はダミーノード）

4.4　エッジの配線

エッジの配線には、バスネットワークの視覚的表現に使われるような、水平線と垂直線で構成する手法を採用した。図 3 は、エッジの配線形態の説明図である。

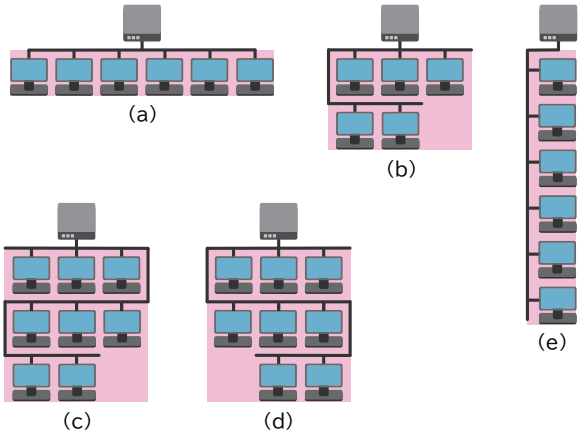


図 3　エッジの配線形態（ピンクの長方形はダミーノード）

4.4.1　葉群が 1 行の場合

まず、図 3(a) は水平配置における配線で、これは従来手法でもしばしば使用される配線手法である。(i) 親の下辺の左右の中心から下に向かう垂直線、(ii) 親と子の間に位置する水平線、(iii) 水平線から個々の子に向う垂直線から構成される。

4.4.2　葉群が複数行、複数列の場合

図 3(b)-(d) は、蛇行配線である。(ii') 親と子の間に位置する線を、水平の直線ではなく、蛇行させている。それ以外の、(i) 親の下辺の左右の中心から下に向かう垂直線と (iii) 蛇行した線から個々の子に向う垂直線は同様である。

なお、(ii') 親と子の間に位置する蛇行した線の始点が、図 3(b) では右上にあるのに対して、図 3(c) では左上にある。これは 4.3 において、最下行の配線が無駄に長くならないようにするためであり、各行の葉を左詰めで配置したことから、行数が偶数の場合には右上から、奇数の場合には左上から開始している。ただし、行数に関わらず、開始位置を（あるいは左上）に揃える方が、美的には良いと感じるかも知れない。その場合には、図 3(d) のように、行数が奇数の場合には、最下行を右詰め（あるいは左詰め）で配置すればよい。

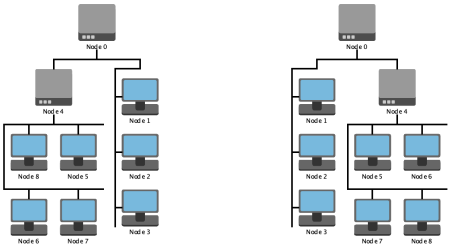
4.4.3　葉群が 1 列の場合

葉群が 1 列に並んでいる場合には、図 3(e) のように、(i) 親の下辺の左右の中心から下に向かう垂直線、(ii') 親と子の間に位置する水平線と子の左に位置する垂直線、(iii') 垂直線から個々の子に向う水平線から構成される。これは横転規約で利用される配置と同じものである。ただし、親の横方向の中心が、(ii') の垂直線からずれているため、(ii') に水平線を含める必要がある。

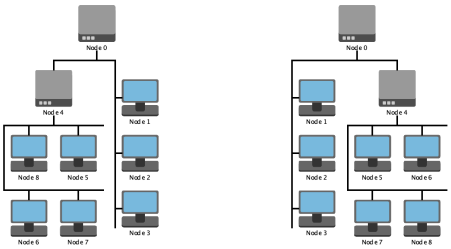
図 3(e) では子の左右の中心と親の左右の中心が揃っているが、親には他にも葉ではない子があるかも知れない。その場合には、子の左右の中心に親を配置することになり、親から下に降りる垂直線と、子の左側に位置する垂直線の横方向の位置が揃っていないことで、図 4(a) に示した例のように、無用な折れ曲りが出現してしまう。葉群が 1 列の場合でも、他に（葉でない）子がある場合には、下に下す線を葉群の左右中心ではなく左にずらすことで、図 4(b) に示した例のように、折れ曲りを 1 回にすることができる。

5.　描画例

図 5 と図 6 に描画例を示す。図 5 に示した例は、2 分木のような木であるが、葉だけはひとつの親に 20 ある。図 6 に示した例は、左右がアンバランスなカスケード状の木である。子の数はそれぞれ 5、だたし根の子は 30 である。どちらもコンピュータネットワークにおいて、スイッチが階層構造になっており、末端に PC がたくさんあるような



(a) 根から葉に降りる部分が 3 回折れ曲っている



(b) 根から葉に降りる部分が 1 回しか折れ曲っていない

図 4 エッジの折れ曲りを減らした例

構造を模したものである。

図 5 と図 6 のどちらも、描画領域の面積を均一とし、目標アスペクト比を、それぞれ、(a) 1:2、(b) 1:1、(c) 2:1 として、局所蛇行レイアウトを求めたものである。また、図 (d) は従来の水平配置の結果をアスペクト比 2:1 の描画領域に収めたものである。(a) や (b) と同じ描画領域に水平配置の結果を収めたものは、図 (d) よりもさらに縮小しただけであるため、ここでは掲載していない。

図 (a)-(d) の中で、最も横長の描画領域である、2 : 1 においても、図 (c) と図 (d) の違いは明らかである。局所蛇行レイアウトの図 (c) は、水平配置の図 (d) に比べて、大きな拡大率で描画領域に収めることができている。さらには、図 (a) と (b) でも、比較的大きな拡大率で描画領域に収まっていることが分る。図 (c) と比べても、これらの例では、図 (a) や (b) の方が拡大率が大きい場合もある。

6. 評価

性質と規模の異なる木を使用して、空間効率と計算時間を調査した。使用した木は以下の 4 種類で、それぞれ規模（ノード数）の異なる 10 個を使用した。

T1 2 分木、深さ N（$N = 2, \ldots, 11$）、ただし葉である子はそれぞれ 50

T2 N（$N = 2, \ldots, 11$）分木、深さ 4、ただし葉である子はそれぞれ 50

T3 カスケード状の木、深さ N（$N = 2, \ldots, 11$）、子の数はそれぞれ 5、だたし根の子は 100

T4 カスケード状の木、深さ 5、子の数はそれぞれ N（$N = 10, \ldots, 100$、10 刻み）、だたし根の子は 100

描画領域の面積は一定として、アスペクト比を変えたときの、空間効率（描画領域にちょうど収めるための拡大率）と計算時間を調べた。このとき、アスペクト比は $2^{a/2} : 1$（$a \in \{-4, -3, \ldots, 4\}$）の 9 段階とした。最も縦長の描画領域は 1 : 4、最も横長の描画領域は 4 : 1 である。

6.1 空間効率

図 7 は調査した木の種類毎に、アスペクト比（比の値）と拡大率の関係を示したものである。いずれも多少の上下はあるものの、おおむね水平である。ただし、図 7(c) では、大きい木において、アスペクト比が大きくなると、空間効率が低下する傾向が見える。これは、最も下にあるダミーノード内がすでに横一列になっており、これ以上縦方向を縮めることができないためである。

6.2 計算時間

指定されたアスペクト比に適合するレイアウトの求め方は、条件を満すまで、徐々に葉群の列数を増すというものであり、一種の線形探索である。計算効率を議論する段階ではないが、それでもおおまかなパフォーマンスを把握するために、現状の実装による計算時間を調査した。

プログラムは Java で実装し、Java(TM) SE Runtime Environment (build 17.0.1+12-LTS-39) で実行した。使用した計算機は、Intel(R) Core(TM) i9-9900K CPU @3.60GHz、RAM 64.0GB、Windows 10 Pro である。他のアプリケーションはできるだけ停止して実行した。揺れによる誤差を排除するため、同条件の計算をそれぞれ 1,001 回行って計算時間を計測し、中央値を使用した。

図 8 は、調査した木の種類毎に、木の規模（ノード数）と計算時間（ミリ秒）の関係を示したものである。図 8(a) や (b) に示した 2 分木では、ノード数が多くなると計算時間が小さくなるところがあり、その後は線形に増えている。これは、ある段階からは最も縦長のレイアウトでも描画領域のアスペクト比よりも横長になるため、1 回のレイアウト計算で終ることを表している。つまり、計算時間は小さいが、空間効率は低い状態になっている。図 8(c) で、ノード数が多くなると、AR:4.00 や AR:2.83 の線が頭打ちになっているのも、同様の理由である。本手法では、木の高さよりも縦方向を小さくはできない。図 8(d) はどの条件でもまだ空間効率を最大にできている。描画領域のアスペクト比を固定すると、ノード数の増加に対して計算時間もおおよそ線形に増えている。

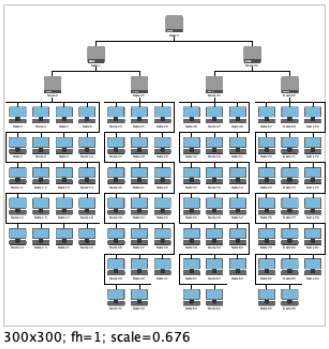
7. 制限および対策

7.1 蛇行による視覚的混雑

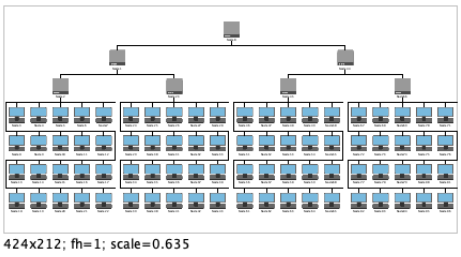
本手法は、連結図として表現することを前提として、バスネットワークに使われるような視覚的な表現を採用した。空間効率を向上させるために、描画領域に合わせてバスにあたる部分を蛇行させたが、このことは時には視覚的混雑を引き起こすかも知れない。

この問題に対する一つの解決は、葉とその親の関係の表

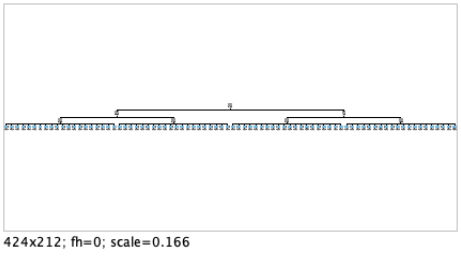
(a) 1:2



(b) 1:1

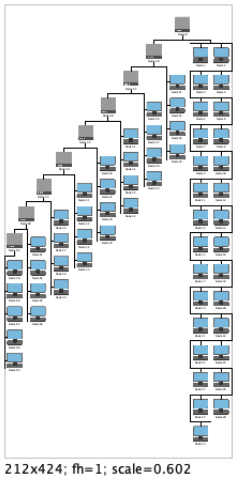


(c) 2:1

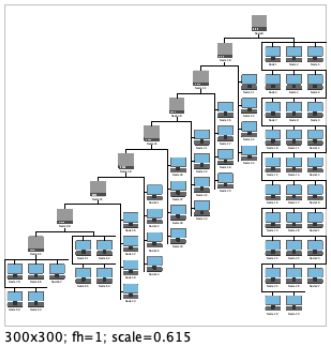


(d) 2:1（水平配置）

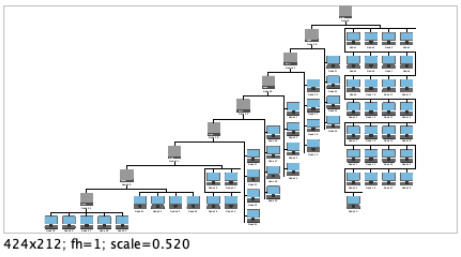
図 5 描画例 1: 2 分木、ただし葉である子の数はそれぞれ 20



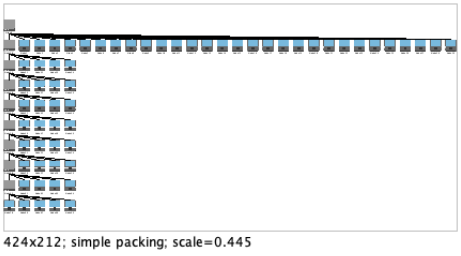
(a) 1:2



(b) 1:1



(c) 2:1



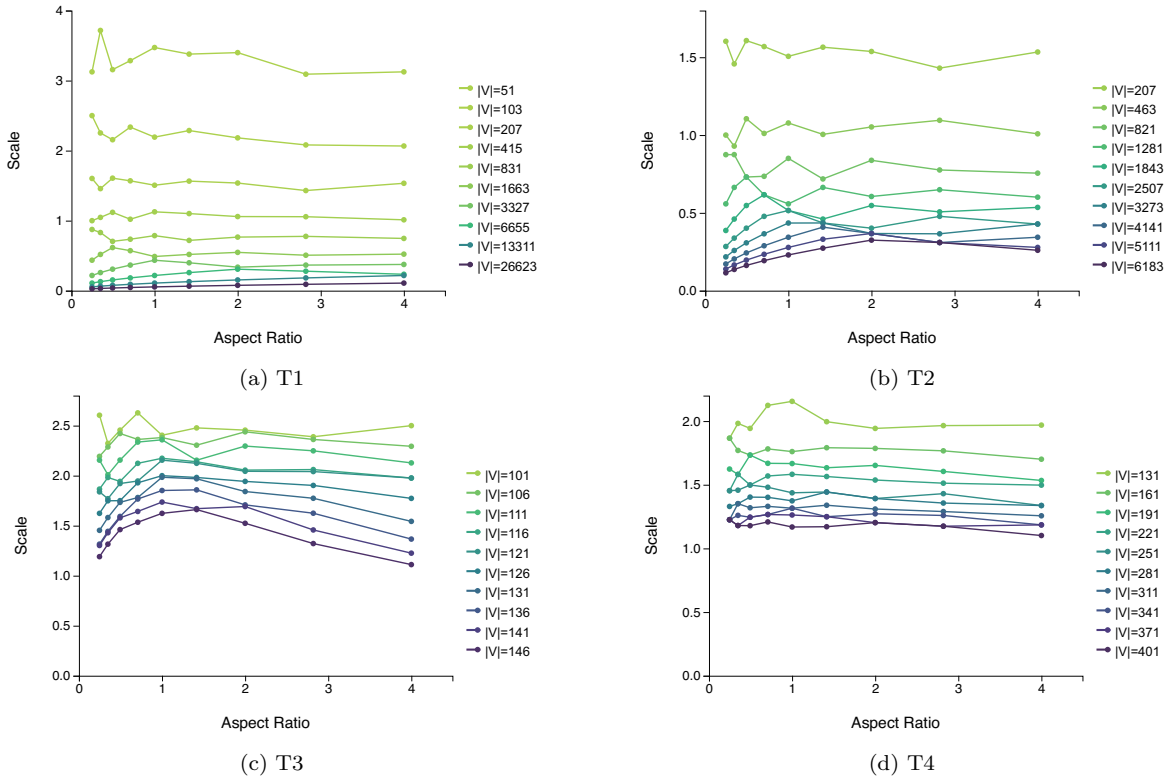
(d) 2:1（水平左詰め配置）

図 6 描画例 2: カスケード状の木、
子の数はそれぞれ 5、だたし根の子は 30

現方法を連結図から領域図に変えるものである。葉群を囲む図形としては、すでにダミーノードが求められている。このダミーノード内のエッジを線分で描かずに、ダミーノードをひとつの領域として描くことで、このような表現が実現できる。図 9(b) は、図 9(a) と同じ木を、そのようにして描いた例である。

7.2 十分に空間効率を高められないケース

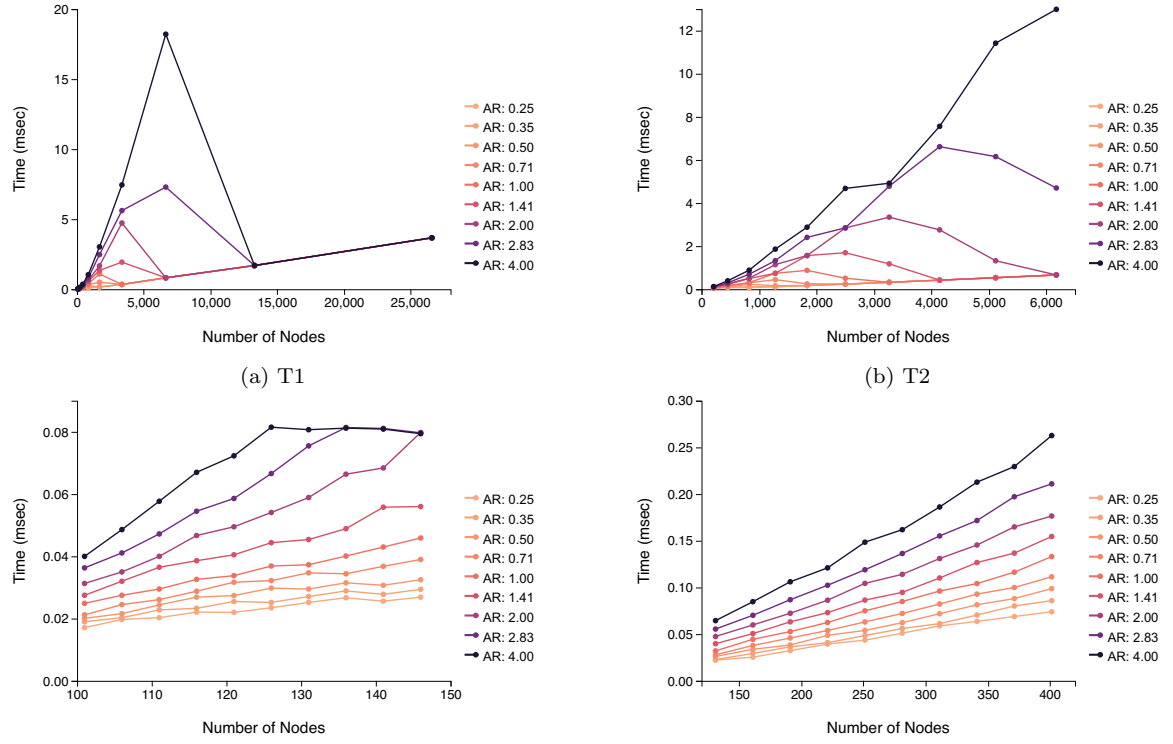
本手法は葉の兄弟が少ない木ではその効果が十分に表れない。たとえば完全 2 分木は兄弟が 2 しかないため、二つ



(a) T1　(b) T2

(c) T3　(d) T4

図 7　空間効率の評価: アスペクト比と空間効率の関係



(a) T1　(b) T2

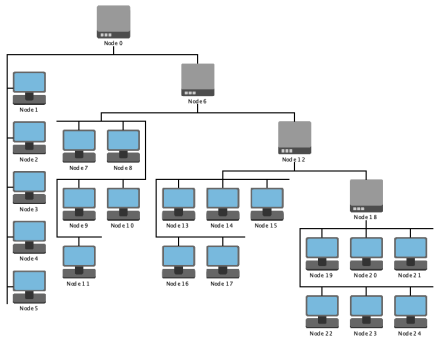
(c) T3　(d) T4

図 8　計算時間の評価: ノード数と計算時間の関係

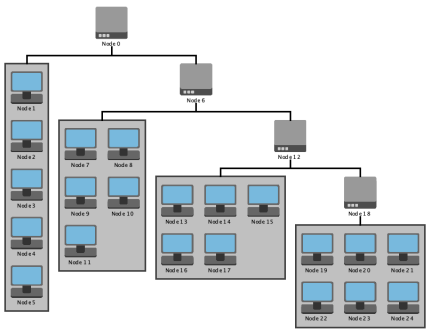
のノードを縦に配置することで、横方向の幅を半分にできるが、それでも規模によっては横長の配置になってしまう。図 10 は深さ 7 の完全 2 分木の描画例である。アスペクト比 4 : 3 の領域に収めようとすると、縦方向に余白が生じてしまう。

8. まとめと今後の課題

根付き木の描画手法として、連結図の一方向レイアウト、特に階層レイアウトが用いられることが多いが、しばしば空間効率が悪いレイアウトになるという問題がある。この



(a) 蛇行した線による親子関係の表現



(b) 長方形領域による親子関係の表現

図 9 連結線と領域による親子関係の表現の併用

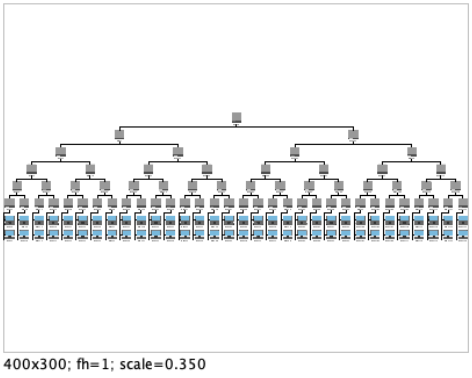


図 10 十分に空間効率を高められないケース

ような問題に対して、一方向という条件は維持しつつも、水平線上に配置するという条件を緩める手法である「一方向局所蛇行レイアウト」を開発した。このレイアウト手法は、水平線ではなく、蛇行した線上にノードを配置するものである。レイアウトアルゴリズムを示すとともに、空間効率と計算時間に関する実験結果を示した。評価対象とした木に関しては、空間効率が描画領域のアスペクト比にあまり影響を受けないことが示された。その一方で計算時間は規模に関して揺れが大きい。

今後の課題は、計算コストの改善と、さらなる空間効率の向上である。現在のアルゴリズムでは、与えられた木の最も縦長のレイアウトから始めて、徐々に横長のレイアウトに変えることで、指定されたアスペクト比に近付くレイアウトを探している。一種の線形探索であり、計算コストを下げるためには、線形でない探索方法を探す必要がある。また、現在のアルゴリズムによるレイアウトでは、7.2 節に示したような兄弟ノードの少ないケースでは、描画領域に余白が生じる。このような余白を減らせるような、さらに空間効率の高いレイアウト手法の開発が期待される。

参考文献

[1] 杉山公造. グラフ自動描画法とその応用. 計測自動制御学会, 1993.

[2] Peter Eades, Tao Lin, and Xuemin Lin. Two tree drawing conventions. *International Journal of Computational Geometry & Applications*, Vol. 3, No. 2, pp. 133–153, 1993.

[3] Ben Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, Vol. 11, No. 1, pp. 92–99, January 1992.

[4] Shengdong Zhao, M. J. McGuffin, and M. H. Chignell. Elastic hierarchies: combining treemaps and node-link diagrams. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, pp. 57–64, Oct 2005.

[5] Kim Marriott, Peter Sbarski, Tim van Gelder, Daniel Prager, and Andy Bulka. Hi-trees and their layout. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 17, No. 3, pp. 290–304, 2011.

[6] Margaret A. Bernard. On the automated drawing of graphs. In *Proceedings of the Third Caribbean Conference on Combinatorics and Computing*, pp. 43—55, 1981.

[7] P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Computational Geometry*, Vol. 2, No. 4, pp. 187–200, 1992.

[8] Charles Wetherell and Alfred Shannon. Tidy drawings of trees. *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 5, pp. 514–520, 1979.

[9] E.M. Reingold and J.S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 2, pp. 223–228, 1981.

[10] John Q. Walker II. A node-positioning algorithm for general trees. *Software: Practice and Experience*, Vol. 20, No. 7, pp. 685–705, 1990.

[11] Christoph Buchheim, Michael Jünger, and Sebastian Leipert. Drawing rooted trees in linear time. *Software: Practice and Experience*, Vol. 36, No. 6, pp. 651–665, 2006.

[12] Kim Marriott and Peter Sbarski. Compact layout of layered trees. In *Proceedings of the Thirtieth Australasian Conference on Computer Science - Volume 62*, ACSC '07, p. 7–14, AUS, 2007. Australian Computer Society, Inc.

[13] Atze van der Ploeg. Drawing non-layered tidy trees in linear time. *Softw. Pract. Exper.*, Vol. 44, No. 12, p. 1467–1484, December 2014.

[14] Willem Sonke, Kevin Verbeek, Wouter Meulemans, Eric Verbeek, and Bettina Speckmann. Optimal algorithms for compact linear layouts. In *2018 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 1–10, 2018.

[15] Ulf Rüegg and Reinhard von Hanxleden. Wrapping layered graphs. Report 1803, Department of Computer Science, Kiel University, Kiel, Germany, 2018.