

Prüfer点列に基づいた全域木の汎用的列举法

片岡靖司

平成31年3月

Prüfer 点列に基づいた全域木の汎用的列挙法

片岡 靖詞*

(平成 30 年 7 月 24 日受付；平成 30 年 11 月 2 日受理)

A Versatile Algorithm for Enumerating All Spanning Trees Based on Prüfer Sequence

By Seiji KATAOKA*

We propose an algorithm that enumerates all spanning trees based on Prüfer sequence. This algorithm progresses by taking edges as leaf-edges and is controlled only by checking a label on each vertex. Somewhat primitive and old-fashioned as the algorithm seems to be, it can be applied to sparse graphs, directed graphs, or graphs with various constraints. Because of forming trees from leaf-edges, this algorithm works without caring about any complicated data-structural discussions such as finding cycle, cut, bridge, contracting, shrinking and so on. This algorithm is so versatile that we can enumerate some restricted trees. Furthermore, we can also use the enumerating system itself as a framework of branch-and-bound method.

Keywords: Spanning Tree, Prüfer Sequence, Enumeration, Versatility

1. はじめに

全域木はグラフ理論の中でも最も基本的な構造のひとつであり、与えられたグラフから諸条件を満足する全域木を列挙する問題は、組合せ数学における理論的な議論⁸⁾だけでなく、ネットワークデザイン、コンピュータ・ネットワークの構築、化学式構造の分析などの応用例もあり、さまざまな分野で必要とされている。Rosen¹⁵⁾や Wu¹⁸⁾では、求める解の形状が全域木となるような数理的問題をいくつか紹介している。それらの問題には、枝の向きが重要であったり、ハブ・スポークシステムのような特殊な形状が要求されたり、化学式のようにある点が必ず葉であるとか、ある点の次数が必ず 4 でなければならないなど条件が付加される。これらの諸条件が数理的にモデル化できるものであれば、ソルバの活用で解決できる問題も少なくない。しかし、枝の採否が諸条件の組合せ的な状況にしたがうような、数理的にモデル化し難いものになると、何らかの列挙法に頼らざるを得ない場合もある。

全域木を総列挙するアルゴリズムは、多くの研究者によって興味が持たれ、古典的なもの^{6, 10, 14)}から、計算時間あるいは実行スペースのオーダの縮小を目的として洗練されたもの^{7, 11, 16)}まで多くの研究がある。こ

れらの列挙アルゴリズムのアイデアは大きく 2 種に分類できる^{9, 17)}。ひとつは、バックトラック的に部分木を拡張していくもの (BT 型) である。もうひとつは、全域木の基本的な性質 — 全域木に 1 本枝を加えると閉路ができ、その閉路から 1 本の枝を除くことによって別の全域木に移移できる — に基づいてすべての全域木を局所探索的に変遷させていくもの (LS 型) である。それらの過程で生成される全域木に重複がないように、また部分木の拡張や全域木の変遷が効率的に行われるように工夫を凝らすことがアルゴリズムの中核となる。

主なアルゴリズムの特徴などを表 1 にまとめる。ここで、 V は点の数、 E は枝の数、 N は全域木の総数である。表 1 から分かるように、バックトラック法によるものは時代的に古いものであるが、有向・無向のどちらでも比較的柔軟に扱える。しかし、全域木の総数 N の枝数倍あるいは点数倍の計算時間を要し、さらには橋の判定などを膨大な回数分実施する必要があるため、実装するためにはデータ構造上の工夫も必須であった。そして、多くのアルゴリズムの研究がそうであるように、木の列挙も計算時間や実行スペースの理論的縮小化が研究目的となり、手法的には木の変遷型が主流となる。しかし、扱えるグラフが無向に限ら

* 防衛大学校 情報工学科 教授 (内線 2270)

表 1 全域木列挙アルゴリズムの特徴
Table 1 Features of algorithms for enumerating all spanning trees

文献	手法	適用	特徴
Read-Tarjan ¹⁴⁾	BT 型	有向・無向	時間 $O(EN + V + E)$, スペース $O(V + E)$ のアルゴリズム。部分木の点とそれ以外との間の橋の判定を必要とする。
Gabow-Myers ⁶⁾	BT 型	有向・無向	Read-Tarjan 計算時間を $O(VN + V + E)$ に短縮。
Kapoor-Ramesh ⁷⁾	LS 型	無向	1 度の枝交換で変遷できる木を核とし、核となる木のみ列挙することで、時間 $O(N + V + E)$, スペース $O(VE)$ のアルゴリズムを実現。
Shioura-Tamura ¹⁶⁾	LS 型	無向	逆探索 ³⁾ を利用し、Kapoor-Ramesh のアルゴリズムをより簡明化。
Nievergelt-Deo-Marzetta ¹¹⁾	LS 型	無向	fat 型・skinny 型などグラフの形状に特徴を持つ場合を考察。

れ、木を陽に全部列挙するのではなく、1 回の枝の交換で到達できる木集合を 1 グループとしてまとめることにより、時間やスペースの縮小を達成している。また逆探索³⁾が可能なように制限を設けるなどの限定性を持つ。そして、理論的には優れたアルゴリズムにはなるものの、諸条件を満足する木の列挙には対応が難しく、実用性や汎用性・簡便性からは離反していく。

本論文で提案する全域木の列挙法は、木を構成する枝を 1 本ずつ確定させながら進行するバックトラック型であり、アルゴリズムに要する時間やスペースの観点からは古典的なものである。しかし、全域木の葉の方から順次確定していくことに大きな特徴があり、これまでの列挙法には例を見ない。葉の方から確定していくため、固定する枝での閉路のチェックや枝の縮約を必要とせず、生成される全域木の重複性チェックの問題も起こらない。そして列挙の進行過程がそのまま、全域木を解の形状に持つような問題に対して、分枝限定法の枠組みとして使用することができる。また、有向グラフにも適用でき、並列計算などによって負荷を分散できることなど、実装上の諸問題にも対応できる。

本論文の列挙アルゴリズムは、Prüfer 点列¹²⁾の考え方に基づいている。Prüfer 点列は、多くの書籍にも紹介されているが、完全グラフにおける全域木の数を示すために使われる程度に過ぎず、これを用いて何らかの問題を解いたり、あるいは近似解法に使われたなどの例は見あたらない。それどころか、後述するように“近似解法には向いていない”といった主張までがある。本研究は、これまで列挙アルゴリズムの考え方としては注目されてこなかった Prüfer 点列に着目したうえでも、この点列の新たな可能性を示している。

本論文では、まず第 2 節において Prüfer 点列について説明し、その特徴を述べた後、第 3 節で列挙アル

ゴリズムを紹介する。Prüfer 点列の考え方に基づいてはいるが、実際には各点に与えたラベルを参照しているだけである。第 4 節では、提案する列挙法の適用幅の広さを示すために、枝疎グラフ・有向グラフにも使えることを示す。また初期状態を適切に与えることにより、特殊な形状の全域木のみ列挙したり、列挙木そのものが分枝限定法の枠組みとして使えることなども示す。

2. Prüfer 点列

Cayley の公式:⁵⁾ n 点の完全グラフ K_n における全域木の総数は n^{n-2} である。

この公式にはいくつかの証明が知られている²⁾。その代表的なものが Prüfer¹²⁾によるものであり、1 から n の点番号による長さ $n-2$ ($n \geq 2$) の点列と各全域木とを 1 対 1 に対応させるものである。後の議論に欠かせないため、ここで簡単に“全域木から Prüfer 点列”、“Prüfer 点列から全域木”への変換アルゴリズムを紹介する。

全域木から Prüfer 点列

各点には 1 から n の番号がつけられているものとする。木 T において、葉 (leaf) とは次数が 1 の点である。葉は木 T のある枝と対応しており、その枝の葉とは反対側の点を節 (joint) と呼ぶことにする。ここでは、葉と節を点番号の上に、それぞれ $\wedge$ と $\sim$ というマークを付けて区別する。

Prüfer 点列はある木における節番号の並びである。最小番号の葉を見つけ、その節となる点番号を Prüfer 点列に追加し、その葉と枝とを木 T から除去する。これを最後に 2 つの点が残るまで続ける。Algorithm T.to.PS() は、その手順を示したものであり、 $V(T)$ は現在の木における点番号の集合。 $\deg(i)$ は点 i の次数。PS[] は Prüfer 点列を示す配列である。

図 1 の例を用いて、全域木から Prüfer 点列を得る手順を示す。最初に度数 1 で最小番号である 3 が葉として選ばれ、2 が対応する節である。選ばれた枝 (3, 2) に対し、点 2 を $PS[1]$ に代入し、点 3 とそれに接続する枝を木から除く。次に葉となるのは 4 であり対応する節 7 を $PS[2]$ とする。同様にして (6, 5) が選ばれ、 $PS[3]=5$ 。さらに (5, 1)、最後に (1, 2) が選ばれ、その節を $PS[]$ に追加し、最終的に $PS[]=(2, 7, 5, 1, 2)$ となる。

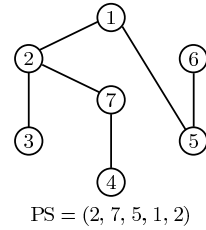


図 1 7 点の全域木と Prüfer 点列
Fig.1 A 7-node spanning tree and the Prüfer sequence

Algorithm T_to_PS()

```

k := 1;
while(k ≤ n - 2)
    /* Vertex i is the leaf having the minimum number. */
    i := min{i | i ∈ V(T), deg(i) = 1};
    /* Add the corresponding joint j ((i, j) ∈ T)
       to the Prüfer sequence. */
    PS[k] := j;
    T := T \ {(i, j)};
    k := k + 1;
end
    
```

Prüfer 点列から全域木

逆方向が Algorithm PS_to_T() である。これは Algorithm T_to_PS() で選ばれた枝を同じ順序で 1 本ずつ選ぶものである。L を葉の候補となる点集合。k 番めに選ばれる葉は、L の要素の中で k から n - 2 番めの Prüfer 点列に入っていない最小番号を持つ点である。対応する節は、Prüfer 点列の k 番めの点 $PS[k]$ である。これを L の要素が最後に 2 つ残るまで続ける。最後に L に残った 2 点間の枝を木に加えればよい。

図 1 の例題を用いて、 $PS[]$ から全域木に復元できることを示す。最初に $PS[v]$ ($v = 1, 2, 3, 4, 5$) に入っていない点で最小番号の 3 が葉として選ばれ、対応する節が $PS[1]$ の 2 となる。次に $PS[v]$ ($v = 2, 3, 4, 5$) にない点で、まだ葉として選ばれていない最小番号である 4 が葉として選ばれ、対応する節は $PS[2]$ の 7 である。同様にして $PS[v]$ ($v = 3, 4, 5$) にない点で、まだ葉として選ばれていない最小番号は 6 であり、節は $PS[3]=5$ となる。さらに (5, 1)、(1, 2) が選ばれ、最後に L に残った 2 点を結ぶ (2, 7) を木に追加して終了する。

Prüfer 点列は節になる点を入れていくので、元々の全域木において葉であった点は、Prüfer 点列には入らないことがわかる。また、節になる点を追加していくことから、Prüfer 点列を見ると元々の全域木にお

ける各点の次数も読み取ることができる。すなわち、Prüfer 点列に選ばれた点の回数プラス 1 が、その点の次数である。図 1 を見ると、度数 3 の点 2 が 2 回、度数 2 の点 1, 5, 7 が 1 回現れ、また葉である 3, 4, 6 は 1 度も現れていない。さらには、Prüfer 点列を見ると、木の形状もある程度想像がつく。Prüfer 点列に現れた番号がすべて異なっているときはハミルトニアン・パスになり、両端の点は Prüfer 点列に入っていない点である。また、すべての点が同じときは、その点を中心とする星形である。また、限られたいくつかの点番号だけが何度も現れているときは、ハブ・スポークシステムのような形状をしていることもわかる。

Algorithm PS_to_T()

```

k := 1; L := V; T := ∅;
while(k ≤ n - 2)
    /* The leaf i is the minimum number vertex
       that is in L and not in PS. */
    i := min{i | i ∈ L \ (∪_{v=k}^{n-2} {PS[v]})};
    /* The corresponding joint j is PS[k]. */
    j := PS[k];
    T := T ∪ {(i, j)};
    L := L \ {i};
    k := k + 1;
end
/* Add the last edge, where the two vertices are in L. */
T := T ∪ {(i, j)} (i, j ∈ L);
    
```

3. Prüfer 点列に基づいた全域木の総列挙アルゴリズム

Prüfer 点列は、特に次数あるいは次数から予想できる木の形状などについて有用な情報を含んでいるにもかかわらず、それが何らかの最適化問題を解く手段

```

#include<stdio.h>
#include<stdlib.h>
#define N 15
int n, PS[N], TreeNum = 0,
int main()
{
    int i;
    int InitVlabel[N];
    void AllSpanningTrees();

    printf("Key in the number of vertices. \n");
    scanf("%d",&n);

    for(i = 1; i <= n; i++) InitVlabel[i] = 0;

    AllSpanningTrees(1, 0, InitVlabel);

    printf("The number of spanning trees is %d.\n",TreeNum);
}
/*****
The Algorithm to enumerate all spanning trees
k is the depth of enumerating tree.
Vlabel is the label on each vertex.
Lsum is the total amount of Vlabel except -1(leaves).
*****/
void AllSpanningTrees(int k, int Lsum, int *Vlabel)
{
    int i, j, ell;
    int NextVlabel[N];

    /* Completion of a Prufer sequence */
    if(k > n-2){
        TreeNum++;
        printf("the %dth Prufer sequence ",TreeNum);
        for(i = 1; i <= n-2; i++) printf("%d",PS[i]);
        printf("\n");
        return;
    }

    for(i = 1; i < n; i++){ /* leaf loop */ A
        /* Leaf must be Vlabel[i] == 0 */ B
        if(Vlabel[i] != 0) continue;
        Vlabel[i] = -1; /* i is a leaf. */

        for(j = 1; j <= n; j++){ /* joint loop */ C
            /* Skip the leaves. */
            if(Vlabel[j] == -1) continue;
            /* Checking the rest PS space. */ D
            if(Vlabel[j] == 0 && k+Lsum > n-2) continue;

            /* Taking an edge (i,j) */
            printf("the %dth edge (leaf:%d-joint:%d)\n",k,i,j);
            /* Add j to the Prufer sequence. */
            PS[k] = j;

            /* Conveying labels to the next depth */
            for(ell = 1; ell <= n; ell++)
                NextVlabel[ell] = Vlabel[ell];
            /* Adjust the NextVlabel[] and Lsum */ E
            if(Vlabel[j] > 0){
                NextVlabel[j]--;
                ell = 1;
            }
            else ell = 0;

            AllSpanningTrees(k+1, Lsum-ell, NextVlabel);
        }
    }
    /* i cannot be the first found leaf. */ F
    if(k+Lsum > n-2) return;
    /* i will not be the first found leaf. */ G
    Vlabel[i] = 1;
    Lsum++;
}
}

```

図 2 AllSpanningTrees() の C 言語によるソースコード
Fig. 2 AllSpanningTrees(): the source code in C language

として使われたり，近似解法の構築に寄与したという研究を見かけることがない。ひとつには，Prüfer 点列の局所的な変化あるいは交換は，しばしば全域木の形状に大きく変化をもたらすことになるからである。Raild-Julstrom¹³⁾は次数制限付き最小木問題において，“Prüfer sequence is not useful for local search strategies such as genetic algorithms.”とまで述べている。

Prüfer 点列を利用して，ただ全域木を総列挙するだけであれば， $(11 \cdots 1)$ から $(nn \cdots n)$ までの点列に対し，アルゴリズム PS_to_T() を適用して全域木に変換するというのが最も原始的な方法であろう。しかし，この方法は完全グラフに対してのみ適用可能であり，枝疎なグラフに適用すれば，枝の有無を判定するルーチンを膨大な回数呼ばなければならない。また，もしある最適化問題が対象としているグラフが完全グラフであったとしても，各レベルで n 分枝し，レベルの高さが $n-2$ であるような列挙木を分枝限

定法の枠組みとして使うことは困難である。なぜなら，葉 i （およびそれに接続する枝）を選ぶ方法は， $\min\{i | i \in L \setminus (\bigcup_{v:=-k}^{n-2} \{PS[v]\})\}$ なので，Prüfer 点列の最後の要素， $PS[n-2]$ までわかっていなければならない。そして，列挙木の初期段階のレベルにおいては，途中までしか Prüfer 点列が決まっていない。このため，次に分枝すべき葉（およびそれに接続する枝）を選ぶことができない。

このような欠点はあるものの，Prüfer 点列や Algorithm T_to_SP() あるいは PS_to_T() をより詳しく観察すると，次のような性質があることがわかる。

- 性質 1: k 回目の繰り返しで点 i を葉として選ぶと，点 i は k から $n-2$ 番めの Prüfer 点列に現れることはない。
- 性質 2: k 回目の繰り返しで点 i を葉として選ぶと，点 i より小さい番号で，まだ葉に選ばれていない点は， k から $n-2$ 番めの Prüfer 点列に少

なくとも 1 度以上現れる。

これらの性質に基づくことで、葉と節を 1 つずつ確定させながら、全ての Prüfer 点列、つまり全ての全域木を列挙するアルゴリズムを構築することができる。このアルゴリズムの本質的なアイデアは、各点に“あと何回以上 Prüfer 点列に現れる予定か”を示すラベルをつけるだけである。初期ラベルは、どの点も 0 回以上 Prüfer 点列に現れる可能性があるため、すべて 0 である。また、いったん葉として選ばれた点は、その後 Prüfer 点列に現れることはないため、区別するためにラベルの値を -1 に設定する。列挙中に見ているのは、このラベルだけであり、Prüfer 点列の残り枠数 (k 本めの枝を選ぶときに、 $n - 2 - k$ が残り枠数) よりも、この先に Prüfer 点列に現れる予定の点の数の方が多くなってしまう場合はバックトラックする。このために、葉として -1 のラベルがついた点以外のラベルの数値の合計を $Lsum$ として覚えておく。

列挙木の動きを制御しているのは、Prüfer 点列の残り枠数と、その枠に入る予定の点の数 $Lsum$ の大小チェックだけである。これだけで表立った重複のチェックもなく、図 2 のようなアルゴリズム (完全グラフ版) が提示できる。アルゴリズムは使用するプログラム言語に依存するものではないが、簡明さを伝えるためと、カットや閉路をチェックするサブルーチンなどがブラックボックス化されていないことが分かるように、そしてそのまま入力すれば誰でも使えるように、敢えて C 言語ソースコードで示す。また、このアルゴリズムで 4 点における全域木が列挙される様子を図 3 に示す。

アルゴリズムは列挙木の各レベルで 2 重ループ構造をしており、外側のループ **[A]** は葉を選ぶため、内側のループ **[C]** は節を選ぶためのものである。葉も節も、既に葉として選ばれて $Vlabel$ が -1 となっている点から選ばれることはない。また、いったん葉に選ばれた点は、以降の Prüfer 点列に現れることはないため、葉は $Vlabel$ が 0 である点からしか選ぶことができない **[B]**。また点番号が最大である点 n が葉として選ばれることはないため、外側の for 文は n を含まない $for(i = 1; i < n; i++)$ としている **[A]**。点 n が葉となる全域木は、最後に残った 2 点の枝を採るときに葉となる。また葉の選択は点番号の大小関係が意味を持つので for 文でなければならないが、内側の節を選ぶためのループは点番号に依存していないため、適当な順序でも構わない **[C]**。このことが枝疎なグラフや有向グラフでの適用を可能にしている (4.1 節)。

節は既に葉に確定している点以外であれば候補として選べるが、 $Vlabel$ が 0 の点 j を Prüfer 点列の k 番めの節として選ぶ場合には、Prüfer 点列の残り枠数 ($n - 2 - k$) にあと $Lsum$ 個の点が入らなければならないので、それが無理ならスキップする **[D]**。また、 $Vlabel$ が 1 以上の点を節として選んだ場合は、列挙木の次のレベルでは $Vlabel$ の値を 1 少なくする。同時に $Lsum$ も 1 引いておく **[E]**。

内側のループを抜けると、点 i を葉とする木の列挙が終了する。この後に列挙される全域木では、点 i は葉ではなく、Prüfer 点列に 1 度以上現れる予定の点となる。したがって点 i の $Vlabel$ を 1 とし、 $Lsum$ を 1 つ増加させるが **[G]**、もし増加させたときに Prüfer 点列の残り枠数を超えるようであれば再帰関数から抜け出す **[F]**。

このプログラム内では Prüfer 点列 $PS[]$ を構成し、列挙木が深さ $n - 1$ に達したときに出力しているが、見ているのは $Vlabel$ と Prüfer 点列の残り枠数だけなので、 $PS[]$ の存在は本質的ではない。

図 3 はこのアルゴリズムで点が 4 つの場合の列挙木を示している。深さが 4 段になっているように見えるが、アルゴリズムが 2 重ループ構造になっているため、この列挙木において中央の破線までが深さ 1、それ以降が深さ 2 になっていることに注意する。外側のループでは選ばれた葉を二重丸で示し、内側のループで節を選び枝を確定している。一部、 $Vlabel$ の内容を示しているが、葉を選ぶループによって対象が次の葉に移ると、今まで葉だった点は 1 度以上 Prüfer 点列に現れる予定になるので 1 に変わる。しかし、その点が節として選ばれ Prüfer 点列に追加されると、次の深さに入るとき $NextVlabel$ として 0 になっていることに注意する。縦の破線以降では、Prüfer 点列の残り枠数は 1 つしかないため、最後の 4 つの列挙では $Vlabel$ が 1 になっている点からしか節を選ぶことができないため、一意にその点を選んで終了する。右端に $4^{4-2} = 16$ 通りの全域木を出力しているが、点線は最後に残った 2 点間の枝である。そして、最大番号の点 4 は、必ず点線枝の一方になっていることもわかる。

4. アルゴリズムの適用

提案するアルゴリズムは、とても簡単で稚拙にさえ見える。しかし、単に完全無向グラフを対象にした全域木を総列挙するだけではなく、枝疎なグラフ、有向グラフ、次数制約に上下限が混在した特別な形状をした全域木の列挙にも対応できる。また列挙負荷の分

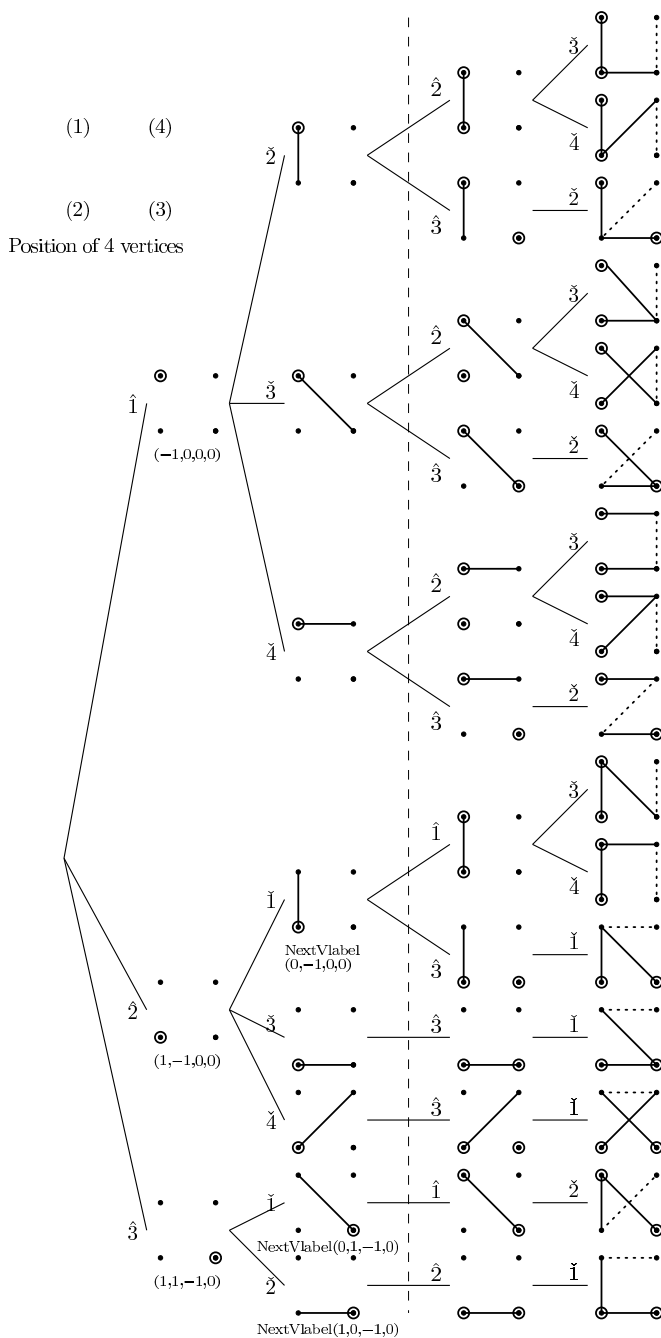


図 3 K_4 における全域木列挙の様子
Fig. 3 The diagram of AllSpanningTrees() to K_4

割,そして分枝限定法への適用など,汎用的な使い方ができる特徴がある。

4.1 枝疎なグラフおよび有向グラフ

図2および図3では完全グラフを対象に説明したが,内側の節を選ぶループ(図2の[C])は点順序には関係ないので,枝疎な場合は適当な隣接リスト構造を準備して,内側の節を選ぶループでは点 i の隣接点を走査すればよい。ただし,次の3つの点に気をつける必要がある。(1)完全グラフでは内側のループで節となる点は必ず見つかるが,枝疎なグラフでは節となる点が見つかることなく内側のループから出ることがある。その場合は内側のループを出たときに再帰関数から戻ればよい(図2の[F]の後)。(2)残次数が1となってしまう点は節になることができないので,節となる点を選ぶ際,残次数が1の点はスキップすればよい(図2の[B]の後)。(1),(2)の注意点は,このアルゴリズムで枝疎グラフを扱う際に無駄な列挙を避けるための長所であるが,(3)Prüfer 点列では,最後に2点が残る,それらを結ぶ枝が最後の枝となるが,枝疎グラフでは最後に残った2点間に枝が存在することを確認しなければならない。さらには列挙木のもっと上位段階でグラフの連結性が失われることがある。連結性が失われるとすれば葉を選ぶ時なので,葉を確定する図2の[B]の後に連結性を判定するルーチンを入れると効果的である(4.6節の2)。

有向グラフの扱いは,枝疎なグラフと同じと考えてよく,内側の節を選ぶループを適当な隣接リスト構造に置き換えればよい。このアルゴリズムの場合,有向木は葉から根に向かう方向に枝が確定していく。したがって根となる点は,点番号の一番大きな点 n になる。

4.2 次数制約

次数制約の下限 d_L は,初期のInitVlabelを設定するだけで扱うことができる。その点の初期ラベルを $d_L - 1$ とし,LsumをInitVlabelの和とすればよい。また次数制約の上限 d_U を設定したい場合,Prüfer 点列にその点が $d_U - 1$ 回より多く表れないように制限すればよい(図2の[D]の後)。また,その点を葉として走査しているときがちょうど次数上限 d_U のときは,その後に節にはならないので,葉としての走査終了後,外側のループから抜ければよい(図2の[F]の後)。

指定したい次数の合計が $n - 2$ のときは,Vlabelが0の点を節に選ぶことがないので無駄なく列挙が進行するだけでなく,さまざまな用途がある。例えば2点のInitVlabelを0,残りの点のInitVlabelを1

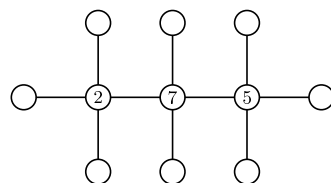


図4 化学的な特殊形状をした全域木
Fig.4 The tree like a chemical structure

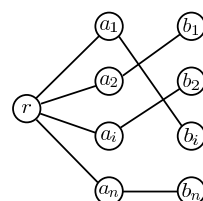


図5 2部グラフにおけるマッチング木
Fig.5 The matching-tree on a bipartite graph

とし,Lsum= $n - 2$ としてメイン関数から呼び出せば,InitVlabelを0とした2点を両端とするハミルトニアン・パスを全列挙することができる。同様に,点の数 $n = 11$,点2,5,7のInitVlabelを3,残りの点のInitVlabelを0とし,Lsum=9にした場合は,図4のような化学的な特殊形状をした全域木だけを列挙することもできる。さらには,図5のように点 r のInitVlabelを $n - 1$,点 a_i のInitVlabelを1,点 b_i のInitVlabelを0とすれば,2部グラフのマッチングの列挙も可能である。実際に効率的かどうかを度外視すれば, n 点の有向グラフに対し,ダミーの点を $n - 1$ 個加えて, s, t 以外の点にはダミーとの間を往復する枝を図6のように加えれば s から r までのハミルトニアン・パスを列挙することにより, $s-t$ パスを全列挙することも可能である。点の数は倍増するように見えるが,ダミーの点番号を n より大きなもの(点 r を最大番号)にしておけば, $s-t$ パスが確定した後の t から r のパスは一意に決まるので,点 t が節として選ばれた時点で再帰関数から戻ればよい。

4.3 計算負荷の分割

ある点の次数制約の下限を2,つまりInitVlabelの値を1に設定すれば,その点が葉になるような全域木を列挙することはない。逆に,その点を必ず葉としたいときは,InitVlabelに -1 を指定し,列挙木の深さを示す最初の引数を2としてメイン関数から呼び出してやればよい。こうすることで,その点を除いた

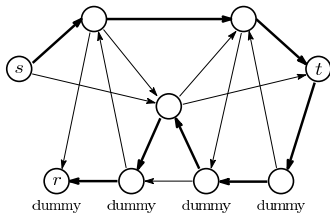


図 6 $s-t$ パスの総列挙

Fig. 6 The enumeration of $s-t$ path

$n-1$ 点における全域木を列挙でき、最終的にはその点と残りの $n-1$ 点を枝で結んでやればよい。

完全グラフでは、ある点が必ず葉となる全域木は、 $(n-1)(n-1)^{n-3} = (n-1)^{n-2}$ 通りあるので、その割合は次式のように約 4 割程度になる。

$$\frac{(n-1)^{n-2}}{n^{n-2}} = \left(1 - \frac{1}{n}\right)^{n-2} > \left(1 - \frac{1}{n}\right)^n \approx e^{-1} \approx 0.37$$

枝疎グラフでは次数に上限があるので、ある点が葉になるもの・ならないものとの割合はおおよそ半々になると評価でき、これらは独立に扱えるので、計算負荷を分割することが可能である。

4.4 分枝限定法の枠組み

本論文で提案するアルゴリズムの最も大きな特徴のひとつであり、従来のバックトラック的列挙法では考えられてこなかった特徴は、全域木の葉、それに接続する節から確定しているところにある。この特徴があるために、列挙木そのものを分枝限定法の枠組みとして捉えることができる。通常、全域木を最適解の構造に持つ問題を分枝限定法で扱うときには、ある枝を採る／採らないと固定した子問題を作成する。このとき子問題において採ると確定した枝を縮約するなど工夫するのが一般的である。また採るべき枝で閉路ができないことを確認しながら固定する枝を選ぶ必要がある。

しかしながら、本研究の列挙木を分枝限定法の枠組みに用いた場合、子問題は葉として選んだ点から出力する枝を取り除いたものとして構成されるため、縮約や閉路の確認などを必要としない。また、この子問題の中には固定する枝を含んでいないため、元問題を小さくしただけの同じ構造を継承している。そして子問題の中（図 7 の楕円内）で求めた全域木と、確定した部分木とをそのまま接続すれば、全体での全域木が構成できる。つまり、子問題において、何らかの緩和問題を解いた値に確定部分の枝重みを加えると（最小化問題では）下界値を与えるし、また、子問題で近似解

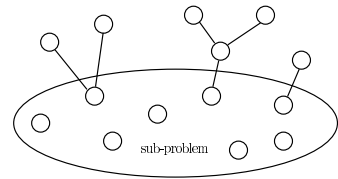


図 7 分枝限定法に用いた場合の子問題の構造

Fig. 7 The subproblem in the branch-and-bound

となる全域木を構成して、そのまま確定部分を付け加えると、その子問題における上界値を与える。

この性質は、従来のバックトラック型列挙法で確定する枝が部分木を拡張していくタイプであっても、うまく機能しない。なぜなら、従来型では、部分木を拡張する枝が、確定した部分木のどの点からも伸びていく可能性があり、図 7 の楕円内で構成した全域木と接続するだけでは閉路ができる可能性もある。このために、子問題において、元問題と同じ問題を独立して継承させることは難しい。

分枝限定法に用いる欠点としては、列挙木の各レベルにおいて多分枝しているの、ひとつの分枝を見切ることができても、まだ多くの見切りのできていない分枝が残る。一般に分枝限定法における列挙木は、多分枝方式よりも 2 分枝方式の方が計算効率が良いことが多い。また、分枝順序が点の番号に依存しているので、番号の付け方次第では良い実行可能解が列挙木の最後の方でやっと現れるということもあり得る。

多くの研究があり、効率的な解法が提案されているような問題に対してこの列挙木を分枝限定法の枠組みとして用いても、それらの研究成果を上回る結果を得ることは難しいかもしれない。しかしながら、（最小化問題の場合）有効な下界値を得る手段もなく、ほぼ全列挙に近くなってしまうような問題や、組合せ的で複雑な見切り条件が絡んでいるような問題、線形制約で簡単に見切り条件が表現できず、商用ソフトに投入するのが困難な問題などでは、効果が期待できる。

4.5 ネットワークフロー問題への適用

葉節を確定しながら列挙が進むこのアルゴリズムは、単に枝を採る・採らないを確定して全域木を構成する 0-1 計画問題だけでなく、点に需給量があり、枝に容量制限があるような問題にも適用可能である。各点に需要量・供給量が与えられて、それぞれの需給量を満足するように枝の流量を求める問題は、最適解の形状が需要点・供給点が交互に現れる木構造を持つよ

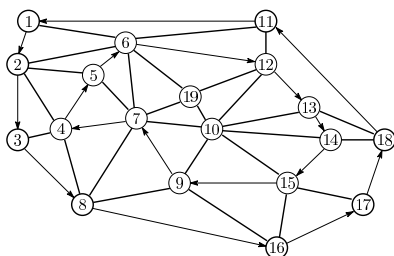


図 8 19 点, 44 枝のグラフ
Fig. 8 A graph of 19 nodes 44 arcs

う変換することが可能である¹⁾。

たとえば輸送問題に対し, このアルゴリズムを適用すると, 葉として選んだ需要点に対応する節の供給点は, 葉の需要量を満足できるだけの供給量が必要である。逆に葉として選んだ点が供給点であれば, 対応する節の需要点には, 葉の供給量をすべて送ることができる需要量を持っていなければならない。これらができない場合は列挙木を見切ることができる。

また, 輸送問題や割当問題などのように 2 部グラフで定義される数理計画問題では, 退化の問題も避けたい。ある葉の需要量をすべて節から送る, あるいは葉の供給量をすべて節に送ったとき, 残った節の供給量あるいは需要量がゼロになってしまうことがある。これは退化を意味しており, 数理計画問題では厄介な現象として, 等価とみなせる複数の解を数多く出力せざるを得ない場合がある。

提案する列挙木のアルゴリズムで, 需給量がゼロになった点を葉に選んだ場合, その葉に接続する枝はどれを選んでも流用はゼロ, つまり退化しており, 全域木としてどの枝を選んでも解としては等価である。したがって, 全域木を列挙するときには, 複数ある節の候補のうち 1 つだけ適当に選んで, その他はすべて見切ることができる。

4.6 適用事例

図 8 のグラフに対する諸設定について本研究のアルゴリズムを適用する。アルゴリズムは C 言語で記述し, DELL Precision T7500 (Intel Xeon X5680(3.3GHz) × 2) 上で実行した。

1. 図 8 を無向グラフとしてとらえた場合, 本アルゴリズムを適用することにより, 8,764,158,889 通りの全域木を列挙した。また要した時間は 167,752.6 秒である。無向グラフの場合, 全域木の総数はラプラシアン行列 (枝 (i, j) が存在するとき -1 , 存在しないときは 0 , そして各点の次

数を対角成分とした行列) から任意の 1 行 1 列除いた余因子の行列式として求められる¹⁵⁾ (キルヒホッフの Matrix-Tree 定理⁴⁾)。そして, 列挙した全域木の数が多いことは, この行列式の値と一致することで確かめられた。

2. 本アルゴリズムの欠点として, 全域木の最後に加える枝は, 残った 2 点として示される。したがって, 残った 2 点の間に枝がなければ全域木が構成できない。この状況を解決するために, 図 7 の楕円内のグラフが連結であるか否かをチェックし, 非連結であれば見切ることができる (4.1 節)。このチェックを入れたとき, 全域木の総数は 8,764,158,889 通りであり, 間違いなく総列挙できており, 実行時間は 28,448.7 秒にまで短縮できる。このような見切りができるのは, 本研究の列挙法が分枝限定法の枠組みとして最適化問題を解くためにも使えることを意味している。そのときは, 連結性だけでなく, 枝重み等により, 何らかの上下界値を活用して見切りを行う。
3. Lsum の値が $n - 1$ のときに効率がよい例として, 点 1 と点 19 の InitVlabel を 0 とし, その他を 1, Lsum = $19 - 2 = 17$ とすると, 5,280 通りのハミルトニアン・1-19 パスを 0.1 秒足らずで全列挙できる。一方, 点 7 と点 10 の次数制限を 5 以上, Lsum = 8 としたハブ・スポークシステムのような木の場合, 10,502 通りであるが, 2.0 秒ほど要した。
4. 次数に上下限が混在している例として, 図 8 の点 1, 2, 3, 8, 16, 17, 18, 11 を次数 1 つまり葉とし, その他の点は次数 2 以上, Lsum = 11 とした全域木を考える。このような問題は, 非端末木問題と呼ばれ¹⁹⁾, VLSI の設計などに応用がある。このように次数制限が厳しくなると, 139,980 通りの全域木を 0.45 秒で求めることができた。木の総数だけでみれば, ハブ・スポークシステムの 10 倍強の数があるが, 制限が複雑になるとかえって実行時間が短くなる例である。
5. 図 8 の矢線を有向枝, その他を双方向の枝としてとらえた場合, 44,513,821 通りの有向全域木があり, それらを 464.8 秒で総列挙することができた。このときの根は点番号の一番大きい点 19 である。

ここに示したものは, 本アルゴリズムの汎用性を示すほんの数例に過ぎないが, 各設定ともプログラムを微

細に修正するだけである。

5. おわりに

本論文では Prüfer 点列に基づいた全域木の総列挙アルゴリズムを提案した。Prüfer 点列そのものは古典的なものであり、広く知られているものであるが、完全グラフを対象にしたものであるため、全域木の総数を示す Cayley の公式を証明する程度にしか用いられてこなかった。その Prüfer 点列の有用性を示し、Prüfer 点列の残り枠数と点列に現れる予定の点の数の大小関係だけで制御できる簡単なアルゴリズムを提案した。簡単なアルゴリズムにもかかわらず、枝疎なグラフや有向グラフにも適用できる。また次数制約の設定には適しており、特殊な形状をした全域木やマッチング、 s - t パスの列挙にも適用できることを示した。

この列挙法の最大の特徴のひとつは、全域木の葉節を確定しながら列挙が進行するところにある。したがって、列挙木をそのまま分枝限定法の枠組みとして使うことができ、固定する枝での閉路の有無などを確認する必要もなく、しかも子問題を構成するときに縮約などの手間も要さないで、そのまま親問題の構造を継承させることができる。またネットワークフロー問題にも適用でき、そのとき起こりがちな退化現象にも対処できる。

提案するアルゴリズムにより、何らかの大規模な最適化問題を従来の研究成果よりも効率的に解くことは期待できないかもしれない。しかし、小規模であっても（最小化問題の場合）有効な下界値を求めるのが難しい問題、複雑な見切り条件が絡んでいる問題、線形制約だけでは表現が困難で商用ソフトに投入することが難しい問題などには効果を期待することができる。

参考文献

- 1) R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows*, Prentice-Hall (1993).
- 2) M. Aigner and G.M. Ziegler, *Proofs from THE BOOK*, Springer (2000).
- 3) D. Avis and K. Fukuda, "Reverse search for enumeration," *Discrete Applied Mathematics* **65** (1996), pp. 21-46.
- 4) B. Bollobás, *Modern Graph Theory*, Springer (1998).
- 5) A. Cayley, "A theorem on trees," *Quarterly Journal of Mathematics* **23** (1889), pp. 376-378.
- 6) H.N. Gabow and E.W. Myers, "Finding all spanning trees of directed and undirected graphs," *SIAM Journal on Computing* **7** (1978), pp. 280-287.
- 7) S. Kapoor and H. Ramesh, "Algorithms for enumerating all spanning trees of undirected and weighted graphs," *SIAM Journal on Computing* **24** (1995), pp. 247-265.
- 8) D.E. Knuth, *The Art of Computer Programming Volume 4 Fascicle 4 Generating All Trees - History of Combinatorial Generation*, Addison-Wesley (2006), Chap.7.2.1.6.
- 9) 久保幹雄, 田村明久, 松井知己: 「応用数理計画法ハンドブック」, 朝倉書店, (2002), pp. 886-932.
- 10) W. Mayeda and S. Seshu, "Generation of trees without duplications," *IEEE Transactions on Circuit Theory* **CT-12** (1965), pp. 181-185.
- 11) J. Nievergelt, N. Deo and A. Marzetta, "Memory-efficient enumeration of constrained spanning trees," *Information Processing Letters* **72** (1999), pp. 47-53.
- 12) H. Prüfer, "Neuer Beweis eines Satzes über Permutationen," *Archiv für Mathematik und Physik* **27** (1918), pp. 742-744.
- 13) G.R. Raild and B.A. Julstrom, "Edge-sets: an effective evolutionary coding of spanning trees," *IEEE Transactions Evolutionary Computation* **7** (2003), pp. 225-239.
- 14) R.C. Read and R.E. Tarjan, "Bounds on back-track algorithms for listing cycles, paths and spanning trees," *Networks* **15** (1975), pp. 237-252.
- 15) K.H. Rosen (eds.) *Handbook of Discrete and Combinatorial Mathematics*, CRC Press (2000).
- 16) A. Shioura and A. Tamura, "Efficiently scanning all spanning trees of an undirected graph," *Journal of the Operations Research Society of Japan* **38** (1995), pp. 331-344.
- 17) 宇野毅明: 列挙アルゴリズム, 日本オペレーションズ・リサーチ学会誌, **52** (2007), pp. 531-534.
- 18) B.Y. Wu and K.-M. Chao, *Spanning Trees and Optimization Problems*, Chapman & Hall/CRC (2004).
- 19) T. Zhang and Y. Yin, "The minimum spanning tree problem with non-terminal set," *Information Processing Letters* **112** (2012), pp. 688-690.