Vol.2016-AL-157 No.9
2016/3/6

直並列グラフの列挙

藤井　淳[1,a)]　上原　隆平[1,b)]

概要：電気・電子回路の配線問題を扱う手法の一つに，直並列グラフを回路のモデルとして用いる方法がある．このとき回路の規模を指定した場合，実現可能な回路モデルを全て知ることは有用である．直並列グラフの辺数を入力サイズとするアルゴリズムは既に知られているが，実際の配線を考慮すると，各端子間の多重配線は適当ではなく，頂点数入力に対するアルゴリズムは未だに考案されていない．本稿では，既存アルゴリズムを元に 2-tree の概念を導入し，頂点数入力による多重辺を含まない直並列グラフ列挙アルゴリズムを提案する．

キーワード：Enumeration, Reverse Search, Series-parallel graph, k-tree

Enumeration of All Series-Parallel Graphs

Atsushi Fujii[1,a)]　Ryuhei Uehara[1,b)]

Abstract: There is no algorithm that enumerates all series-parallel graphs by putting the number of vertices as a size of the seres-parallel graph. In addition, the multiple trace between arbitrary 2 terminals in an electric circuit is not considered desirable. So the algorithm that enumerates simple series-parallel graphs should be proposed. We give an algorithm to enumerate all series-parallel graphs n vertices and no multi-edge without repetition.

Keywords: Enumeration, Reverse Search, Series-parallel graph, k-tree

1. はじめに

1.1 研究の背景

頂点集合，辺集合から構成されるグラフは，様々な構造をもつシステムの特性を把握するためのモデルとして度々利用される．そして，特徴的な性質をもつグラフクラスについて、その要素をすべて列挙することは極めて有用である．あるグラフクラスについて完全なリストがあれば，それを用いて予想の反例検査を行ったり，そのグラフに対するアルゴリズムの平均的な性能を測定することに役立てることができる．

その中でも，電気回路のモデル化に良く用いられる直並列グラフの全てのリストについて知ることは，非常に興味深い問題である．辺数によって直並列グラフの規模を指定した際に，全ての直並列グラフを重複なく列挙するアルゴリズムは 2005 年に川野ら [1] によって考案された．このアルゴリズムは，逆探索法と呼ばれる手法を用いている [2]．逆探索法では，家系木と呼ばれる木状の探索空間を構築し，その家系木を根から探索する．逆探索法で効率の良い列挙を行うには，列挙する要素間の親子関係をどのように定義するかが重要になる．

1.2 研究の目的

先行研究 [1] で考案されたアルゴリズムが辺数入力に対して列挙を行っていることに対し，頂点数入力をグラフサイズとして与えたアルゴリズムは未だに考案されていない．また，実際の電気回路を考慮すると多重配線は好ましくなく，モデルとなる直並列グラフが多重辺を含まないようにするべきだと考えられる．そのために新たな制約条件や仕組みが必要となる．本研究では，頂点数入力に対して

[1] 北陸先端科学技術大学院大学
[a)] fujii@jaist.ac.jp
[b)] uehara@jaist.ac.jp

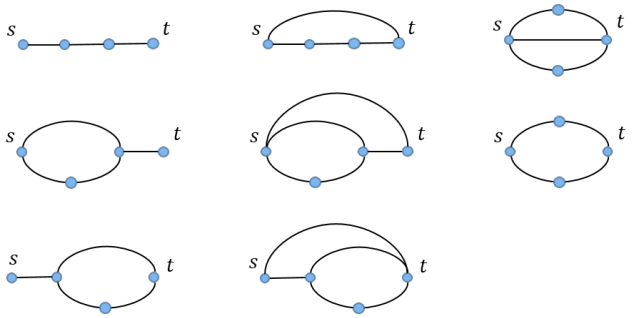


図 1.1: 多重辺を含まない頂点数 4 の全ての直並列グラフ

多重辺を含まない直並列グラフ (例：図 1.1) を全て列挙するアルゴリズムの開発を目的とする．

1.3 本稿の構成

本稿では，第 2 章で直並列グラフ等について諸々の定義を与える．次に第 3 章で，[1] で考案された直並列グラフの列挙アルゴリズムついて説明する．そして，第 4 章でこのアルゴリズムを基に，頂点数入力に対応したアルゴリズムへ拡張し，探索範囲についての制約条件と多重辺の処理について説明する．最後に第 5 章でまとめを述べる．

2. 直並列グラフと *k-tree*

この章では，文献 [3] より，直並列グラフの定義と直並列木の定義を与える．直並列木は，逆探索法において家系木構築に用いるデータ構造となる．また，4.1 章での議論のために，*k-tree* に関する定義も与える．

2.1 直並列グラフ

特別な点 s,t を terminal として持つグラフ $G(s,t)$ をターミナル付グラフという．直並列グラフは，以下のように再帰的に定義されるターミナル付グラフである．

(1) s と t のみを持ち，s と t が 1 本の辺のみで接続されたもの
(2) 複数の直並列グラフを直列結合したもの
(3) 複数の直並列グラフを並列結合したもの

次に直列結合と並列結合について説明する．k 個の直並列グラフ $G_1(s_1,t_1)$，$G_2(s_2,t_2)$，...，$G_k(s_k,t_k)$ が与えられた場合，直列結合もしくは並列結合によって新たに得られる直並列グラフ $G(s,t)$ の s，t は次のように決定される．

- 直列結合：$s=s_1, t=t_k, t_i=s_{i+1}, 1\le i\le k-1$
- 並列結合：$s=s_1=...=s_k, t=t_1=...=t_k$

図 2.1 に直並列グラフの直列結合と並列結合を示す．

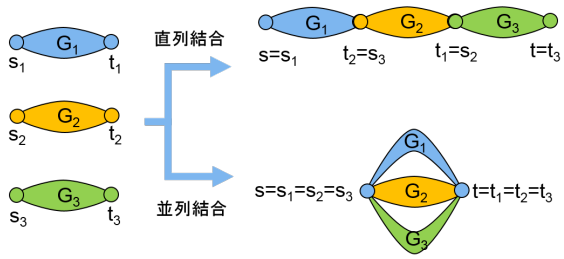


図 2.1: 直並列グラフの結合

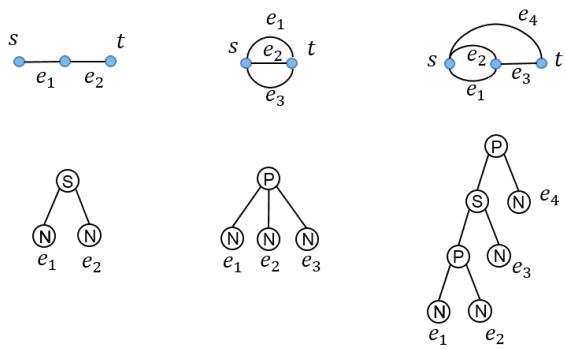


図 2.2: 直並列グラフと対応する直並列木

2.2 直並列木

直並列木の定義は次の定義 2.1 で与えられる [3].

定義 2.1 (直並列木) 直並列グラフ (G,s,t) の直並列木 $T_{(G,s,t)}$ は根付き木である．$T_{(G,s,t)}$ 内のそれぞれの頂点は S 点，P 点，もしくは葉という 3 つの属性のうちの 1 つを持つ．また，$u,v\in V(G,s,t)$ とすると，$T_{(G,s,t)}$ の頂点はラベルとして順序対 (u,v) をもつ．このラベルは直並列グラフのターミナルを表す．直並列木の全ての頂点は，唯一の直並列グラフ (G',a,b) に対応する．ここで G' は G の部分グラフであり，(a,b) は G' の頂点のラベルである．$T_{(G,s,t)}$ の根はラベル (s,t) をもち，(G,s,t) に相当する．$T_{(G,s,t)}$ の葉は G の辺に対応し，その辺の両端点をラベルとして持つ．S 点の子は順序付けされるが，P 点の子は順序付けされない．S 点によって定義される直並列グラフは，S 点の子が与えられた順序で直列結合した結果であり，P 点によって定義される直並列グラフは，P 点の子が並列結合した結果である．

図 2.2 に直並列グラフとそれに対応する直並列木を示す．直並列木の性質として，S 点の子となるのは P 点もしくは N 点，P 点の子となるのは S 点もしくは N 点であるとして一般性を失わない．そうでないときは，連続する S 点同士または P 点同士を併合することができる．

ここで，直並列木について P 点の子の順序は直並列グラ

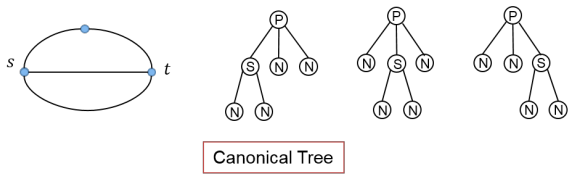


図 2.3: Canonical Tree

フの構造に影響しないため，一般にある直並列グラフに複数の直並列木が対応する．例えば，図 2.3 における直並列グラフについて，対応する直並列木は 3 つ挙げられる．今後の議論のために，直並列グラフと直並列木を 1 対 1 に対応させたい．そのために，Canonical 性の定義を与える．

定理 2.1 T を順序木とし，$V(T) = (v_1, v_2, ..., v_n)$，$dep(v)$ を v の深さとする．$L(T) = (dep(v_1), dep(v_2), ..., dep(v_n))$ とおき，T_1，T_2 を順序木とし，$L(T_1) = (a_1, a_2, ..., a_c)$，$L(T_2) = (b_1, b_2, ..., b_d)$ とする．以下の関係を満たすとき，T_1 は T_2 よりも重いという．ただし，$(i = 1, 2, ..., k-1)$ とする．

$$(a_i = b_i) \cap ((a_k > b_k) \cup (c > k - 1 = d)) \qquad (1)$$

今，それぞれの直並列グラフに対して，最も重い直並列木 H を 1 つ決めることができる．この H を Canonical な木と呼ぶ．直並列木の P 点について，Canonical 性を考慮することで，直並列グラフと直並列木を 1 対 1 に対応付けすることができる．図 2.3 の 3 つの直並列木について，$L(T)$ を調べると，一番左のものが最も重い木となり，この直並列グラフに対する Canonical な木である．

以降の議論では，特に断らない限り，扱う直並列木は全て Canonical 性を満たすものとする．

2.3 *k-tree*

4 章での議論のため，*k-tree* [3] について導入する．

定義 2.2 (*k-tree* の定義) $(k+1)$ 個の頂点からなる完全グラフは *k-tree* である．今，*k-tree* $T = (V, E)$ が与えられたとき，C を $(k+1)$ 頂点からなる T の部分完全グラフとし，$x \notin V$ とする．このとき，$T' = (V \cup \{x\}, E \cup \{cx : c \in C\})$ もまた *k-tree* である．

3. 川野らの直並列グラフ列挙アルゴリズム

この章では川野ら [1] のアルゴリズムについて説明する．逆探索法で用いるデータ構造となる直並列木について，直並列木の親子関係と，その際に必要な判定条件について説明し，川野らのアルゴリズムの疑似コードを与える．また，

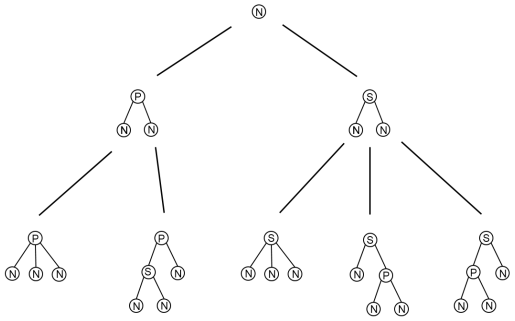


図 3.1: 家系木 F_3

川野らのアルゴリズムについて実際に実装し，可能な限り大きな値の入力によって得られた直並列グラフの総数を示すこととする．

3.1 家系木における直並列木の子生成

家系木 (例：図 3.1) の構築のために必要となるのは，直並列木同士の親子関係を厳密に決めることである．ある直並列木 T について，家系木上での親を $P(T)$，T の全ての子を $C(T)$ とする．逆探索法は，根からの探索により解を導出するため，スタックを用いることで，今の T に対して，家系木の根から親 $P(T)$ までの祖先を記憶することができる．具体的には $P(T)$ が子の 1 つである T を生成したとき，同時に自身をスタックに push する．そして，T が全ての子 $C(T)$ を探索終了したならば，スタックから $P(T)$ を pop する．

次に子 $C(T)$ の生成について説明する．まず直並列木の根が N 点である場合を考える．つまり，この直並列木は最少の直並列グラフを表している．この直並列木を T_r とすれば，T_r の 2 つの子は以下の 2 つの操作で得られる．

- N 点を S 点で置き換え，2 つの N 点を子として加える．
- N 点を P 点で置き換え，2 つの N 点を子として加える．

今，T について，Canonical な全ての子 $C(T)$ を生成するアルゴリズムがあれば，再帰的に家系木を構成できることを意味する．このアルゴリズムは川野らによって与えられた．

3.2 最終除去可能頂点

まず，直並列木同士の親子間の差異を特徴づける頂点，最終除去可能頂点について説明する．直並列木 T について，T に属する頂点が v が次の 3 つを満たすとき，v を除去不能頂点と呼ぶ．

- v は N 点 (葉) である．
- v は兄弟の中で最も右の頂点である．
- v は自身のほかに，兄弟を 1 つのみ持つ．

もし，任意の N 点が上記の条件を満たさなければ，除去

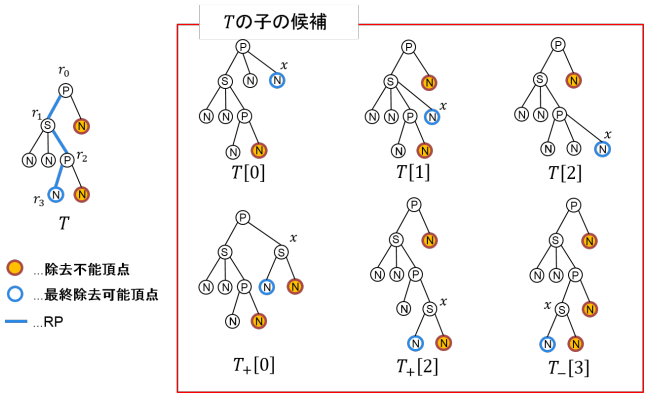


図 3.2: 子の生成

可能頂点と呼ぶ．先行順で T の最後の除去可能頂点を最終除去可能頂点と呼ぶ．

3.3 直並列木の子の生成法

T を Canonical な直並列木，r_k を T の最終除去可能頂点とする．また，$RP = (r_0, r_1, ..., r_k)$ を根 r_0 と r_k の間のパスとする (図 3.2 の T 参照)．T から 3 種類の子候補 ($T[i]$, $T_+[i]$, T_-, $0 \leq i \leq k-1$) を以下のようにして構成する (図 3.2 参照)．

(1) $T[i]$ は，r_i の最も右の子として N 点 x を追加することで得られる．$T[i]$ の最終除去可能頂点は x である．

(2) $T_+[i]$ は，r_i の右の子が除去不能な N 点であれば，S 点（もしくは P 点）x に置き換え，x に 2 つの N 点を子接続する．$T_+[i]$ の最終除去可能頂点は x の左の子となる．

(3) T_- は，r_k を S 点 (もしくは P 点)x に置き換え，x に 2 つの N 点を子接続する．T_- の最終除去可能頂点は x の左の子となる．

$T[i]$，$T_+[i]$，T_- は T と比較して N 点を 1 つ多く持つ．すなわち，家系木 F_m の深さは直並列木の N 点の子数 (直並列グラフの辺数) に対応する．

3.4 子の Canonical 性判定

$C_{can}(T) = \{T[0], ..., T[k-1]\} \cup \{T_+[0], ..., T_+[k-1]\} \cup \{T_-\}$ とする．$C_{can}(T)$ は T の子の候補であるが，$C(T) \neq C_{can}(T)$ である．これは，$T[i]$，$T_+[i]$，T_- の操作によって Canonical 性が失われる可能性があるためである．図 3.2 では，$T_+[2]$ が Canonical な直並列木ではないため，$T_+[2]$ は T の子ではない．

補題 3.1 高々 m 個の葉をもつ全ての直並列木の集合を S_m とする．$T \in S_m$，$T' \in C(T)$，r_k を T' の最終除去可能頂点，$RP = (r_0, r_1, ..., r_k)$ を根 r_0 と r_k の間のパスとする．ここで，$r_i (0 \leq i \leq k-1)$ が r_{i+1} に先行して子を持つならば，それを s_{i+1} とおく．T' が T の子となるための必要十分条件は，次の式が全ての P 点となる r_i で成立することである．

$$L(T'(S_{i+1})) \geq L(T'(r_{i+1}))$$

$C_{can}(T)$ の 1 つがアルゴリズムによって生成されたとき，補題 3.1 に基づいてそれが実際に T の子であるか調べる必要がある．これには多くの走査時間を要する． [1] では，この走査時間を改善する手法を挙げているが，本稿では割愛する．

3.5 川野らのアルゴリズムの評価

以上に述べたアルゴリズムの概要をアルゴリズム 1 にまとめる．このアルゴリズムの作業領域は $O(n)$ で抑えられ，全ての直並列グラフを列挙するのに必要な作業時間は $O(|S_m|)$ となることが [1] で示されている．

アルゴリズム 1 を C で実装し，実際に実行した．アルゴリズムの実行に用いた計算環境は表 1 の通りであり，4.3 章での実装においても同じ環境を用いた．得られた結果について表 2 にまとめる．

表 1: 計算環境

CPU	AMD A10-4655M(2.8Ghz/4core)
OS	Windows 8.1
メモリ	10.0GB
コンパイラ	gcc ver.4.8.1

Algorithm 1 川野らの直並列グラフ全列挙アルゴリズム

Input: 辺数 m ($m \geq 1$)
Output: 入力辺数をもつ全ての直並列グラフ
while 探索家系木 F 上の直並列木 T の葉が m 個以下 **do**
  if (現在の T の葉数)=(入力辺数 m) **then**
    出力．スタックから $P(T)$ を pop し，T を更新する．
  else if 現在の T の全ての子 $C(T)$ が探索済み **then**
    スタックから $P(T)$ を pop し，T を更新する．
  else
    $(T[i], T_+[i], T_-)$ によって未生成の $C_{can}(T)$ を生成する．
    if $C_{can}(T)$ が Canonical である． **then**
      $C_{can}(T)$ は $C(T)$ として確定．T を push．$C(T)$ を新しい T とする．
    else
      この $C_{can}(T)$ を破棄し，新しい $C_{can}(T)$ を生成する．
      未生成の $C_{can}(T)$ がなければ $P(T)$ を pop し，T を更新する．
    end if
  end if
end while

表 2: 川野らのアルゴリズムの実行結果

入力数：m	グラフの総数	計算時間 $[ms]$
3	5	2.3×10^3
4	15	4.0×10^3
5	48	4.9×10^3
6	165	6.3×10^3
7	590	1.5×10^4
8	2184	6.0×10^4
9	8276	2.5×10^5
10	31974	1.1×10^6
11	125394	4.4×10^6
12	497930	1.8×10^7
13	1997630	6.2×10^7

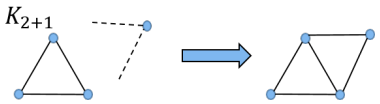
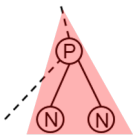
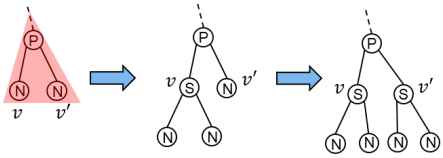


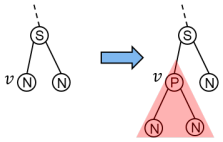
図 4.1: 2-tree の再帰的構築



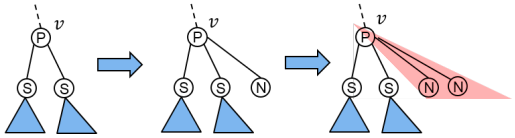
(a) 多重辺構造



(b) 多重辺構造の消滅



(c) 多重辺構造の発生



(d) 多重辺構造の発生 2

図 4.2: 多重辺構造の増減

4. 提案アルゴリズム

この章では，*partial 2-tree* を導入し，提案アルゴリズムの家系木の探索範囲を決定する．また，多重辺の処理についても述べる．

4.1 *k-tree* による探索範囲の決定

定義 2.2 より，*k-tree* は次のように再帰的に構成される．

- 完全グラフ K_{k+1} は最少の *k-tree*
- 新たな頂点 v について，*k-tree* の k 個の頂点に部分完全グラフ K_{k+1} が形成されるように接続したものも *k-tree*

単純な直並列グラフは *partial 2-tree* であることが知られている．

定理 4.1 ([4]) 直並列グラフ G は *partial 2-tree* である．

再帰的な定義から，*2-tree* は頂点数 n に対して，高々 $(2n-3)$ 本の辺を持つことがすぐに言える (図 4.1)．また直並列グラフは連結であるため，辺数は少なくとも $(n-1)$ 本ある．以上より n 頂点の単純な直並列グラフ $G(V,E)$ の辺数は次のようになる．

$$n-1 \leq |E| \leq 2n-3 \qquad (2)$$

4.2 多重辺構造の処理

ここでは多重辺の増減をどのように管理すればよいかその手法を与える．直並列グラフにおける多重辺は，直並列木構造で考えると P 点が 2 つ以上の N 点を子として持つ状態である (図 4.2(a))．子生成の際に，親となる直並列木のどのノードから新たな N 点が発生したかによって多重辺構造の発生・消滅が起こる．ここでは，生成される子は全て Canonical な性質を満たしているものとする．3.1 章で述べた子生成アルゴリズムによって，子が親からどのような変化を経て生成されたかをまとめると次のようになる．

- 内点 (S 点もしくは P 点) の一番右の子として N 点を追加する ($T[i]$ が該当)．
- 葉 (N 点) が内点 (S 点もしくは P 点) に変化し，2 つの N 点を子どもとして追加する ($T_+[i], T_-$ が該当)．

ここで，子生成過程において新たに N 点が追加された点を v とする．上記に示した v への N 点の追加のされ方と追加後の v の属性 (S もしくは P) に着目し，場合分けして考える．

(1) 内点 v に新しく N 点が追加される操作 ($T[i]$)．

(a) v が S 点である場合

この操作では直列結合が行われており，新たな多重辺が生じることがないのは自明である．この操作によって子に対応する直並列グラフでは，頂点数が 1 つ増える．

表 3: 子生成における $|V|$，$|E_{mul}|$ の増減

	v が P		v が S	
	$\|V_{chi}\| \leq 1$	$\|V_{chi}\| \geq 2$	$(\|V_{bro}\| = 0) \cup (\|V_{chi}\| \neq 2)$	$(\|V_{bro}\| \geq 1) \cap (\|V_{chi}\| = 2)$
$\|V\|$	±0	±0	+1	+1
$\|E_{mul}\|$	±0	+1	±0	−1

(b) v が P 点である場合

この操作では部分直並列グラフに対して辺を 1 つ並列結合する操作にあたり，多重辺構造が生じる可能性がある．ここで図 4.2(d) で，新たに N 点が追加される v に着目する．v に N 点が追加されても，v が既に別の N 点を子として持たなければ多重辺構造は発生しないことが分かる．すなわち，v に N 点を追加する操作の後，v が 2 つの N 点を子として持てば，多重辺構造が生じる．この操作によって子に対応する直並列グラフの頂点数は増えない．

(2) 葉 v が内点 (S 点，P 点) に変化し，2 つの N 点が子として加えられる操作 $(T_+([i], T_-)$.

(a) v が S 点に変わる場合

この操作では，並列結合関係状態にある T の葉と部分直並列木のうち，葉が直列結合をここで図 4.2(b) で，新たに N 点が追加される v と v' に着目する．v に N 点が追加されたことによってで多重辺構造が消滅していることが分かる．しかし，その後 v' に N 点が追加されてもすでに多重辺構造がないため，多重辺構造がさらに消滅することはない．そこで，この操作において多重辺構造が消滅するのは，v が自身の他に N 点の兄弟を 1 つ以上持つときである．この操作によって子に対応する直並列グラフでは，頂点数が 1 つ増える．

(b) v が P 点に変わる場合

この操作で，多重辺構造が生じることは明らかである (図 4.2(c))．この操作によって子に対応する直並列グラフの頂点数は増えない．

以上より，子生成の際に，直並列木上で新しく N 点が追加された点 v の属性 (S または P) とその親子関係・兄弟関係を観察することで多重辺構造の発生・消滅について知ることができる．v の N 点の兄弟の数を $|V_{bro}|$，v の N 点である子の数を $|V_{chi}|$，現在の頂点数，多重辺数をそれぞれ $|V|$，$|E_{mul}|$ と表記すると，子生成の際の $|V|$，$|E_{mul}|$ の増減は表 3 のように示される．

よって提案するアルゴリズムでは，探索している現在の直並列グラフの $|V|$，$|E_{mul}|$ を記憶し，入力頂点数 n として，以下の出力判定を用いる．

$$(|V| = n) \cap (|E_{mul}| = 0) \tag{3}$$

また，子生成において多重辺を 1 つ減らすのに頂点数が 1 つ増加することから以下の制約条件を設ける．

$$n - |V| \geq |E_{mul}| \tag{4}$$

Algorithm 2 提案アルゴリズム

Input: 頂点数 n $(n \geq 2)$
Output: 入力頂点数をもつ全ての直並列グラフ
$|V|$ ：現在の直並列木 T に対応する直並列グラフの頂点数
$|E_{mul}|$：現在の直並列木 T に対応する直並列グラフがもつ多重辺数
while 探索家系木 F 上の T の葉が $(2n-3)$ 個以下 **do**
  if [$(|V| = n) \wedge (|E_{mul}| = 0)$] **then**
    出力．
  else if 現在の T の全ての子 $C(T)$ が探索済み **then**
    スタック上から $P(T)$ を pop し，T を更新する．
  else if [$n - |V| < |E_{mul}|$] **then**
    スタックから $P(T)$ を pop し，T を更新する．
  else
    $(T[i], T_+[i], T_-)$ によって未生成の子候補 $C_{can}(T)$ を生成する．
    if $C_{can}(T)$ が Canonical である． **then**
      $C_{can}(T)$ は $C(T)$ として確定．T を push．$C(T)$ を新しい T とする．
      $|V|$，$|E_{mul}|$ の更新も行う．
    else
      この $C_{can}(T)$ を破棄し，新しい $C_{can}(T)$ を生成する．
      未生成の $C_{can}(T)$ がなければ $P(T)$ を pop し，T を更新する．
    end if
  end if
end while

4.3 検証

4.1 章，4.2 章で得られた制約条件を追加したアルゴリズム 2 を実際に C で実装し，得られた結果について表 4 に示す．また，本研究で実装した 2 つのアルゴリズムについて，検証を行った．(図 4.3，図 4.4)．

図 4.3 では，頂点数，辺数の各入力アルゴリズムについて，全ての直並列グラフの生成数に対する計算時間の関係を表している．図 4.3 からは，生成数に対して多項式計算時間以下で計算可能であることが予測される．また，表 2，表 4 から，入力数に対して，頂点数入力によるアルゴリズムは，川野らのアルゴリズムよりも計算時間が多くかかっているが，2 つのアルゴリズムは，生成数に比例する時間で計算可能であることが分かる．

図 4.4 では，同範囲の探索空間に対する計算時間の関係を表している．3.1 章で家系木の深さは，直並列木の N 点の個数 (直並列グラフの辺数) であることを示した．また，式 2 により，我々のアルゴリズムが探索する家系木の高さは入力 $|V(G)|$ 対して $(2|V(G)| - 3)$ となる．探索する家系

表 4: 頂点数入力によるアルゴリズムの実行結果

入力数：n	グラフの総数	計算時間 $[ms]$
3	2	1.6×10^3
4	8	4.1×10^3
5	36	8.1×10^3
6	190	5.3×10^4
7	1090	4.4×10^5
8	6910	3.4×10^6
9	49314	2.8×10^7
10	404819	2.2×10^8

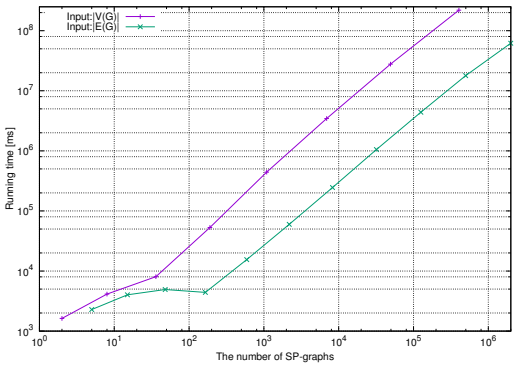


図 4.3: 直並列グラフの生成数に対する計算時間

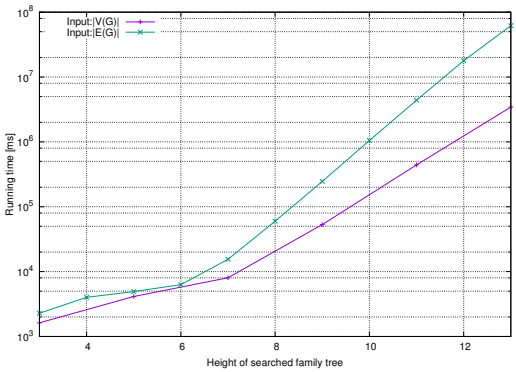


図 4.4: 探索空間に対する計算時間

木の高さを等しくした場合の，2 つのアルゴリズムの計算時間を表したものが図 4.4 である．探索する家系木の高さを等しくした場合，頂点数入力によるアルゴリズムは，川野らのアルゴリズムよりも速くなっていることが分かる．

以上のことから，生成数に対する計算時間の関係は両アルゴリズムで生成数に対し比例することが確認できた．また，探索する家系木の規模に対する計算時間で比較すると，提案アルゴリズムは，川野らのアルゴリズムよりも速くなっており，式 4 で設定した制約条件による枝刈りは有効である．

5. おわりに

本稿では、川野らによって考案された辺数入力による直並列グラフ列挙アルゴリズムを基に頂点数入力に対して単純な直並列グラフを列挙するアルゴリズムを考案した．本研究では以下のことが明らかになった．

- 辺川野らのアルゴリズムに対し，家系木探索範囲の設定と多重辺構造の処理機能を追加し，頂点数入力アルゴリズムへと拡張させた．
- 2 つのアルゴリズムを実装し，川野らのアルゴリズムでは $m = 13$ まで，提案アルゴリズムでは $n = 10$ までの該当する直並列グラフの総数を明らかにした．
- 家系木の規模，すなわち探索範囲に対しての計算時間を比較した場合，頂点数入力によるアルゴリズムでは設けた制約条件によって効率的な枝刈りができていることが確認できた．

今後の課題として以下のことが挙げられる．

- 今回提案したアルゴリズムの定量的評価を行う．
- さらに効率的なアルゴリズムのためのデータ構造を考案する．今回考案したアルゴリズムは，川野らのアルゴリズムの拡張によって得られたものであり，冗長性がある可能性がある．頂点数入力による逆探索に適したデータ構造を新たに考える必要がある．
- 同型な直並列グラフの重複を避けるアルゴリズムを提案する．

参考文献

[1] Shin-ichiro Kawano and Shin-ichi Nakano, *“Generating All Series-Parallel Graphs”*, IEICE TRANS.FUNDAMENTALS, VOL.E88-A, pp.1129-1135, 2005.

[2] David Avis and Komei Fukuda, *“Reverse search for enumeration”*, Discrete Applied Mathematics, 65, pp.21-46, 1993.

[3] Andreas Brandstadt, Vang Bang Le and Jeremy P. Spinrad, *“GRAPH CLASSES: A SURVEY”* , Philadelphia, Society for Industrial and Applied Mathematics, 2004.

[4] Hans L. Bodlaender, *“’A partial k-arboretum of graphs with bounded treewidth’*, Netherlands, Theoretical Computer Science, 209, pp.1-45, 1998.