宇野 毅明
東京工業大学 社会理工学研究科 経営工学専攻

Enumeration Algorithms and Speeding up

Takeaki UNO
Industrial Engineering and Management, Tokyo Institute of Technology

Abstract: Recently, enumeration algorithms have been developed for many problems. This paper surveys enumeration algorithms and also shows related studies, especially studies on constructing way and speeding up of enumeration algorithms. At the last of the paper, we write some open problems and further remarks in the enumeration problems.

1 列挙アルゴリズムとは

列挙問題とは, 与えられた集合 $\mathcal{F}$ の要素で, 性質 P を満たすものを全て出力する問題である. 例えば, あるグラフの部分木の集合の中で, 全張木になっているものを出力せよ, というものである. これを単に全張木の列挙と呼ぶ. 「P を満たす $\mathcal{F}$ の要素の集合 $\mathcal{F}'$」を考えれば, この問題は $\mathcal{F}'$ の要素を全て出力する問題に置き換えられる. これを $\mathcal{F}'$ の列挙と呼ぶ. 列挙問題は, 線形不等式系により与えられた多面体の頂点の列挙 [24], 無向グラフの全張木の列挙 [10, 16, 25, 23, 34], 有向グラフの 2 点を結ぶパスの列挙 [23] など, 多くの問題が研究されている. また, 文字列 A に含まれる全ての文字列 B を出力するストリングマッチング, パラメーター λ を制約や目的関数に含む最適化問題で λ を変化させたときの最適解を全て出力するパラメトリック最適化問題 [11] なども列挙問題の一種とみなせる. これら列挙問題を解くアルゴリズムを列挙アルゴリズムという. 本稿では列挙アルゴリズムの高速化に関わる研究を中心として, 列挙問題の周辺を解説する.

まず, この節では, 列挙問題の応用, 関連する研究, 過去の列挙アルゴリズムの紹介をしよう. 列挙問題は非常に基礎的な問題であるが, 応用上の問題で列挙問題に変換されるものは比較的少ない. しかし列挙アルゴリズムを使用する解法や研究は多い. 例えば, $\mathcal{F}$ の要素を一様発生させたいときには全ての要素を列挙しておけば容易であろうし, 2 目的最適化問題において, 片方の目的関数値がある程度良い解を列挙してもう片方の目的関数を良くする解を探す方法もある. 列挙アルゴリズムはグラフや多面体の性質を調べるために利用されることもある. 安定集合問題, 巡回セールスマン問題等の組み合わせ問題では, 解集合の作る多面体のファセットに関する研究が多い [21]. これらの多面体のファセットを列挙すれば新しい種類のファセットを発見する糸口となるであろう. また, 列挙問題の変形として, 全てではなくある程度の数だけ要素を出力するという研究もあり, 列生成法において解の候補を増やすため, また分枝切除法では, 最適解の下界を与える多面体を切る平面をいくつか求め, 多面体を縮小するために使われる [22].

$\mathcal{F}$ の要素に重みを定義し, 重みの小さい順に $\mathcal{F}$ の要素を出力する問題の研究もある. 特に最小な要素から小さい順に k 番目までの要素を出力する問題は k-best 問題と呼ばれる [2, 18, 9, 4]. 2 目的関数最小化で, 片方の関数値が k 番目以内の解を求め, もう片方の関数値を評価することにより, 効率的に良い解を見つけることができる. また集合族の列挙では, 要素の辞書順で列挙を行う研究もある [15]. 出力をメモリに格納して使用する場合, 辞書順であればデータの扱いは容易になる.

要素を出力する際に, 直接的な方法を使わず, 他の方法を使うことにより出力量を小さくするコンパクト出力の研究も行われている [16, 25, 28, 29]. よく用いられるコンパクト出力に, ある要素を, その前に出力した要素との差分で出力する方法がある. この方法では, 出力量は, 出力の順番や

リズムの高速化が望める．また, 列挙アルゴリズムのいくつかは, 計算そのものよりも出力に時間がかかるので, これらの高速化にも役立つ．他に, 今井浩らのグループにより, 出力を集合の直積の形や, BDD で表現することにより出力の大きさを劇的に小さくする研究も行われている [13].

グラフの性質の調査や, ネットワーク信頼性問題との関わりから, 集合の要素の総数を計算する問題も研究されている．無向・有向グラフの全張木に対しては, それらの総数を計算する多項式アルゴリズムが知られている [3]．また, 総数の計算の難しさにより問題をクラス分けする研究も進んでいる．要素の総数 N を計算するのに $\theta(N)$ の時間が必ずかかる $\#P$-complete というクラスが提案されており, 2 部グラフのマッチングの総数計算は $\#P$-complete であることが知られている．要素の総数が効率よく計算できると, それを使って要素の一様発生を行うアルゴリズムが設計できることが多い．全張木に対しては, $O(|V|^3)$ 時間で 1 つの全張木を等確率で発生させるアルゴリズムが提案されている．[3]

以上が列挙問題に関わる研究の概略である．他にも関連する研究があるが, ここでは割愛させて頂く．列挙アルゴリズム自体に対しても多くの研究がある．以下に, 過去に提案されてきた列挙アルゴリズムの一部を表で示そう．なお, 時間は出力 1 つあたりにかかる計算時間を示す．また, n はグラフの頂点数, または平面上の頂点数を示し, m はグラフの枝数, または制約式の数を表す．d は次元を表し, $l(m,d)$ は d 次元で制約数 m の線形計画を解くのにかかる時間とする．

列挙問題		著者	時間	空間
2 部グラフのマッチング	完全	Fukuda(93) [8]	$m+n$	nm
		Uno(97) [30]	n	$m+n$
	k-best	Murty(68) [20]	n^3	kn^2
		Chegireddy(87) [2]	n^3	$m+kn$
	極大	Uno(97) [30]	n	$m+n$
一般グラフの極大マッチング		Uno(97) [31]	$n \log\log n$	$m+n$
2 部グラフの辺彩色		Matsui(96) [29]	n	$m+n$
全張木	無向	Read(75) [23]	m	$m+n$
		Gabow(78) [10]	n	$m+n$
		Kapoor(92) [16]	1	nm
		Shioura(93)	1	nm
		Shioura(97) [25]	1	$m+n$
		Uno(97) [33]	1	$m+n$
	k-best	Kato(81) [18]	m	$k+m$
		Frederickson(85) [9]	$m^{1/2}$	$k+m$
		Eppstein(90) [4]	k^2	$k+m$
	有向	Gabow(78) [10]	m	$m+n$
		Kapoor(92) [16]	n	$m+n$
		Uno(96) [28]	$n^{1/2}$	$m+n$
		Uno(98) [33]	$\log^2 n$	$m+n$
閉路, パス	無向	Read(75) [23]	m	$m+n$
	有向	Johnson(75) [14]	m	$m+n$
		Read(75) [23]	m	$m+n$
有向パス	k-best	Eppstein(94) [5]	k^2	$m+n+k$

極小カット	Tsukiyama(80) [27]	$m+n$	$m+n$
２部グラフの極大安定集合	Kashiwabara(92) [17]	1	
極大安定集合	Tsukiyama(78) [26]	nm	$m+n$
	Johnson(88) [15]	nm	$m+n$
連結な頂点誘導部分グラフ	Avis(96) [1]	n	$m+n$
	Uno(98) [32]	1	$m+n$
アサイクリックグラフのトポロカルオーダー	Avis(96) [1]	n	$m+n$
平面的全張木	Avis(96) [1]	n^3	n
多面体の頂点	Seidel(91) [24]		
	Avis(96) [1]	nd	nd
アレンジメントのセル	Avis(96) [1]	$ml(m,d)$	nm
平面点集合の三角形分割	Avis(96) [1]	n	n

この表の詳細, 及び他のアルゴリズムに関する情報は, 松井泰子, 松井知己, 福田公明作成による列挙アルゴリズムのホームページを参照されたい. [19]

2　代表的な列挙アルゴリズム構築法

この節では列挙アルゴリズムの基本的な設計法について述べよう. 列挙アルゴリズムの設計手法は, 再帰的, あるいは逐次的な手法に基づいたものが多い. 特に, 前者の再帰的な設計手法が多数を占める. ここでは, その再帰的な手法として, 分割法と逆探索を紹介しよう.

まず 1 つ目として分割法 (binary partition) を紹介する. 集合 $\mathcal{F}$ に対し, 分割法は $\mathcal{F}$ を空でない 2 つの集合 $\mathcal{F}_1$, $\mathcal{F}_2$ に分割する. そして, $\mathcal{F}_1$ の列挙問題と, $\mathcal{F}_2$ の列挙問題を再帰的に解くことにより, $\mathcal{F}$ 全体の列挙を行う. 再帰的に分割を繰り返し, 問題の大きさが十分小さくなったら, しらみつぶしに要素を探し, 列挙する. イメージを掴むために、図 1 を参照されたい. $\mathcal{F}$ の分割方法としては, $\mathcal{F}$ が X 上の集合族のとき, ある分割要素 $e^* \in X$ を選び, 「e^* を含む集合の集合」と「e^* を含まない集合の集合」に分割するものがある. 分割法は非常に単純であり, かつ, 多種の問題に対して効率的なアルゴリズムを設計することができる. 例として, 無向グラフの全張木の列挙アルゴリズムを見てみよう. 詳しくは [10] などを参照されたい.

与えられた無向グラフ $G = (V, E)$ の全張木の集合を $\mathcal{F}$ とし, $\mathcal{F}$ の要素を列挙する分割法のアルゴリズムを考えよう. もし, セルフループでない閉路が G にを含まれなければ, G は全張木を 1 つしか含まない. この場合, 列挙は簡単である. そうでない場合, 閉路の中から適当に分割枝 e^* を選ぶ. そして, $\mathcal{F}$ を e^* を含む全張木の集合 $\mathcal{F}_1$ と, e^* を含まない全張木の集合 $\mathcal{F}_2$ に分割する. ここで, e^* を含まない全張木は G から e^* を取り除いたグラフ $G \setminus e^*$ の全張木になっている. また, e^* を含む全張木は, e^* を縮約して得られるグラフ G/e^* の全張木に e^* を加えたものになっている. また, これらの逆も成り立つ。 よって, $\mathcal{F}_1$ の列挙は G/e^* の全張木の列挙と等価であり, また $\mathcal{F}_2$ の列挙は $G \setminus e^*$ の全張木の列挙と等価である. G/e^* を $G \setminus e^*$ がどのようなグラフになるかは, 図 2 を参照されたい. 以上より, G の全張木は, これらの子問題を再帰的に解くことによって列挙でき, また, 子問題も同じように再帰的に解ける.

以上のように無向グラフの全張木の列挙は分割法により効率良く行うことができる. この問題に対して分割法がうまく動く理由は, 分割して得られた集合 $\mathcal{F}_1$, $\mathcal{F}_2$ が共に, あるグラフの全張木の集合になり, 子問題が親問題と同様な手法で解けるからである. 一般的にはこのようにうまく行くとは限らず, 例えばグラフ G の極大安定集合 (互いに枝で結ばれていない G の頂点集合) 列挙問題では, ある分割頂点 v^* を含む極大安定集合は, G から v^* に隣接する頂点を取り除いたグラフの極大安定集合になるが, v^* を含まない極大安定集合は, このようにうまく表すことができない. G から v^* を除いたグラフを考えても, 図 3 のように, このグラフの極大安定集合は G の極大安定

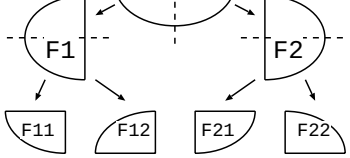

図 1：分割法の動き.
集合を再帰的に分割する.

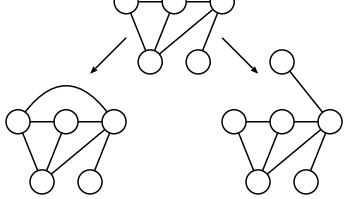

図 2：$\mathcal{F}_1$ と $\mathcal{F}_2$ の列挙問題に
対応するグラフ.

集合になるとは限らない. 効率的なアルゴリズム作成には, 集合の分割方法, 及び分割して得られた集合をいかにして表すかが重要である.

2つ目に紹介する逆探索は, 分割法よりは多少複雑で, ゆえに分割法では解けない問題を解くことができる. 逆探索の詳細については [1] を参照されたい. また, [7] に同じ著者による日本語による解説もある. 逆探索のアイディアは, 全ての要素に「その要素の親」を定義し, その親子関係から得られるグラフを探索することによって列挙を行うというものである. ある要素 e_0 以外の $\mathcal{F}$ の要素 e に対し, その親 $p(e)$ を何らかのルールによって定義する. ただしこのとき, e 自身が e の真の祖先になる, すなわち $\mathrm{p(p(...p(e)...))}$ が e となるようなことがないようにする. ここで $\mathcal{F}$ の要素を頂点とし, e_0 以外の全ての e に対して e と $p(e)$ の間に枝を張ったグラフ $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ を考える. 上記の親子関係に関する制約から, このグラフは木となる. この木を列挙木と呼ぶ. 逆探索は, 列挙木 $\mathcal{T}$ を e_0 から始めて深さ優先探索し, 各頂点に対応する要素を出力することにより列挙を行う. しかし, 一般に列挙木のサイズは巨大であるので, 探索を行う際にメモリに格納することはほぼ不可能である. そこで, 列挙木のある頂点を与えると, その子供を全て出力するアルゴリズムを作り, 探索の各反復においてそれを呼び出す. そして, 見つけた子供それぞれについて再帰的に子供を全て出力していく. これにより, メモリに列挙木を格納せずに深さ優先探索を行うことができる. 図 4 で逆探索の動きを確認されたい.

逆探索の例として, 線形不等式形で与えられる多面体の実行可能辞書を列挙するアルゴリズムを述べよう. このアルゴリズムの詳細は [1] を参照されたい. 以下, 実行可能辞書を単に辞書と表記する. 多面体の辞書から辞書に移動するためには, 単体法という良い道具がある. そこで, ある単体法 S と目的関数 c を用意し, 各辞書 D に対し, その辞書から S のピボットルールで得られる辞書 を $p(D)$ とする. 最小添え字規則などを使った単体法はサイクリングを起こさないので, この親子関係は先の条件を満たし, 最適解が e_0 に対応する. 最適解が e_0 ただ一つになるように c を設定できることを注意しておこう. ある辞書 D とその子供は高々 1 個所しか違いがない. そこで, D を 1 箇所変更してできる全ての辞書に対し, その親を作成し, D が親であるか調べると, D の子供を全て出力できる. 1 箇所変更してできる辞書の数は D の大きさの多項式に比例するので, 1 反復あたりの時間が D の多項式である効率的なアルゴリズムが作成できる.

効率良い逆探索アルゴリズムには, 良い親子関係が不可欠である. 親子関係はどのようにも定義できるが, 適当に定義した親子関係では, ある要素の子供の候補が指数個になったり, ある要素の親を得るのに指数時間かかったりする. 例えば, 列挙する要素に適当な重みを与えて順番を付け, k 番目の親は $k-1$ 番目という親子関係が定義できるが, これでは, ある子供の親を見つける, またはその逆を行うのに多大な時間がかかるであろう. 上記の例では, 単体法により得られる親子関係が, 親を簡単に計算でき, かつ, 子供の候補も少ない, ということが効率的なアルゴリズム作成の鍵になっている.

この他にも, 要素を辞書順のヒープに蓄え列挙する方法や [15], 多面体の端点を列挙する逐次追加法 [24] など, いくつかの列挙アルゴリズム構築法が提案されている. これらの中で上記の分割法と逆探索を取り上げた理由は, 一つに方法が単純でかつ応用範囲が広いこと, 二つに, 使用する記憶領域が少ないことである. ヒープを使った方法と逐次追加法は, 出力の大きさに対して指数的な

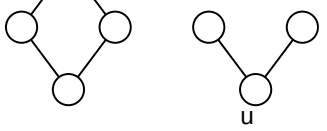

図 3: 右のグラフ $G \setminus v$ の極大安定集合 u は左のグラフ G の極大安定集合にはならない

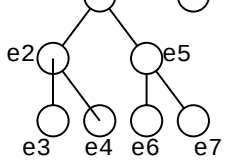

図 4: 列挙木の例. 逆探索は e_0 で子供 e_1, e_8 を発見, e_1 に移動, 子供 e_3, e_6 を発見，と進む.

記憶領域を使用する可能性があり, その意味で分割法・逆探索が効率的であろう.

3　列挙アルゴリズムの速度と高速化

この節では, 列挙アルゴリズムの高速化について触れる. 一般に高速化の研究の評価は, ある尺度で測った計算時間をどの程度減少させたかで行われる. そこでまずここでは, 列挙アルゴリズムの速度の指標について述べよう. 一般に, アルゴリズムの速度は, 入力した問題の大きさ n に対する計算時間の上界で測られる. しかし, 列挙問題の場合, 出力数 N が n の指数オーダーになることが珍しくない. そのため, この尺度では, 組み合わせをしらみつぶしに探すアルゴリズムや、出力数の小さいときにも多大な時間を使うアルゴリズムと、効率の良いと思われるアルゴリズムが同じ計算量であるということになりかねない. ゆえに, 一般的に, 列挙アルゴリズムの計算量は n と N に対する時間で算定される. 通常 N は非常に大きいので, 計算量が N の線形より大きいと非効率的である. 列挙アルゴリズムの計算量は, N に対して線形, すなわち, $\mathrm{O}(f(n)N)$ のような多項式で表されることが望ましい. ゆえに, 近年提案された列挙アルゴリズムの多くが, 出力数の線形時間で終了する. そこで以下では, 列挙アルゴリズムの速度を, 出力１つあたりの計算時間で論ずることにする. ただし, ここには初期化などの時間は含まないものとする.

一般に, 列挙アルゴリズムは多くの計算時間を必要とする. それゆえに高速化は１つの大きな課題である. 多くの列挙アルゴリズムは出力に対して線形であるので, 出力数に対しての計算量の減少は難しい. そこで, 入力に対する計算量をいかに下げるかという研究が多く行われている. 先に述べた全張木の列挙アルゴリズムは高速化の研究が盛んであり, 多くのアルゴリズムが考案されてきた [16, 25, 23, 34]. その他, グラフのパス [5], 有向根付き木 [16, 28, 32] 等の列挙アルゴリズムの高速化が研究されている. これらの高速化の多くは, データ構造の導入により行われている. これは, 列挙アルゴリズムは, 親問題と違いの少ない子問題を再帰的に解くことが多いので, 動的データ構造を使用することにより, 各反復での計算時間を減少させようというアイディアに基づいている. しかし, 一般に列挙アルゴリズムの反復での操作は比較的単純であり, 高速化の余地が少ない. ゆえにこの方法での高速化は成功例が少なく, また, この単純さが他の高速化の技術の適用をも難しくしている. それゆえに, 列挙アルゴリズムの多くには高速化の研究がなされていない. そこで, 近年では「ならし解析」(amortized analysis) の技法を用いて計算量を算定するアルゴリズムや, BDD 形式の出力を行うことで, 実用上の計算時間を減少させるアルゴリズムが提案されてきている. ここでは「ならし解析」による手法を紹介しよう.

一般に, 再帰型のアルゴリズムの計算量は, 一反復の最悪計算時間と反復の最大数の積で算定される. この方法は列挙アルゴリズムの計算量算定にも頻繁に使われているが, 列挙アルゴリズムに対しては非効率的であることが多い. 分割法や逆探索で子問題を再帰的に解く場合, 一般に子問題の計算時間のほうが親問題よりも小さい. 再帰の深さが同じである反復の個数は, 深さに対して指数関数的に増加すると考えられるので, 少量の計算時間しか要さない反復が数多く存在する可能性がある. すなわち, 実際の計算時間とはかけ離れている場合があるのである.

この問題を解決するために用いられるのが, 「ならし解析」である. ならし解析は計算時間の消費が多い反復から少ない反復へと計算時間を移動して全体をならし, ならしたあとの１反復の持つ計算時間の最大で計算量を算定するものである. この手法で計算量を解析することを前提にして

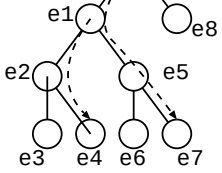

図 5: 1 つ目の分配方法の様子. e_0 から e_1, e_5, e_7 へ、e_1 から e_2, e_4 へと計算時間を分配する.

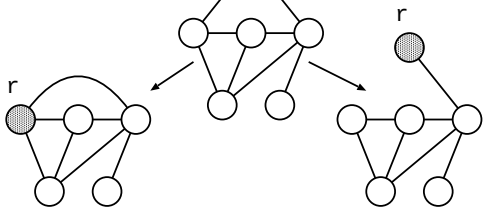

図 6: 枝 e^* を含む木を列挙するためのグラフ G/e^* と, e^* を含まない木を列挙するためのグラフ $G \setminus e^*$.

アルゴリズムを設計することにより, 計算量のより小さい上界が得られる. 有向, 無向の全張木の列挙アルゴリズムの計算量解析がこの手法を使用している [16, 25, 28].

列挙アルゴリズムの各反復がどれくらいの計算時間を要するか, また再帰の構造はどうなるかは入力に依存し, 実際の計算を行わずに, 正確な全体像を得ることは難しい. そこで, 計算時間をならす際には, ある分配のルールを定め, おおまかにならすという方法が一般的である. すなわち, 良いならし解析には良い分配方法が必要になってくる. どのような分配方法がよいかは, アルゴリズムに依存する. だが, 確かに 1 つ 1 つの列挙アルゴリズムに対して独自の分配方法を設計すれば良い解析結果が得られるだろうが, アルゴリズムの設計を簡単にし, アルゴリズム設計法の研究を進める上では, 多くの列挙アルゴリズムに対して適用できる一般的な分配方法を提案するべきであろう. ここではそのような分配方法の中から 2 つを紹介する. 詳しくは [32, 33, 34] を参照されたい.

分配方法の説明をするに当たり, 説明の道具として列挙木を導入する. 列挙木は列挙アルゴリズムと入力に対して定められる木で, アルゴリズムの再帰の構造を表現するものである. アルゴリズムの各反復に頂点を割り当て, ある反復がある反復を呼び出したときに, それらに対応する頂点間に枝を結ぶ. 再帰構造は閉路を含まないので, こうして作られたグラフは木になる. これをアルゴリズムの列挙木と呼ぶ. この列挙木は逆探索が用いる列挙木とは定義が異なるが, 同一のものと考えて差し支えない. 列挙アルゴリズムの解析は, 任意の入力が生成する列挙木いずれもが満たす性質を使い, 行われる.

ではまず 1 つ目の分配方法を説明していこう. 簡単のため, 列挙木の任意の内点は必ず 2 つの子供を持つものとする. 任意の列挙木はこの形に変形することができるので, この仮定は一般性を失わないことを注意しておく. 一般に, 列挙アルゴリズムは, 列挙木の根の周辺に対応する反復で多くの計算時間を消費し, 逆に葉の周辺の部分では少量の計算時間しか消費しない. また, 葉の周辺には, 根の周辺に比べて非常に多くの頂点が存在する. そこで, ある反復の計算時間をその子孫に分配しよう, というのが 1 つ目の分配方法のアイデアである. ただし, 各反復が自分の全ての子孫に分配すると, ある反復が多くの先祖から計算時間を受け取る可能性があり, 具合が悪い. そこで, 各反復が高々1 つの先祖からしか計算時間を受け取らないような分配方法を考えよう. 列挙木の任意の頂点 x に対し, その 2 つの子供を $c^+(x), c^-(x)$ とする. どちらの子供を $c^+(x)$ と定義するかは, 解析の際に都合のいいように行う. x が子供を持たない場合は $c^+(x), c^-(x) = \emptyset$ とする. また, $c_0^+(x) = c_0^-(x) = x$ とし, $i > 0$ に対して $c_i^+(x) = c^+(c_{i-1}^+(x)), c_i^-(x) = c^-(c_{i-1}^-(x))$ とする. $c_i^+(x)$ は, x から $+$ の子供を i 回たどって行き着く頂点である. また, x から $+$ の子供をたどって葉に到達するまでの回数, すなわち, $\min\{i-1 | c_i^+(x) = \emptyset\}$ を $H^+(x)$ とする. 同様にして $H^-(x) = \min\{i-1 | c_i^-(x) = \emptyset\}$ とする.

さてここで, 列挙木の各頂点 x に対応する反復での計算時間を $\mathrm{O}(H^+(c^-(x))T(x))$ としよう. この計算時間を各 $c_i^+(c^-(x))$ (ただし $0 < i < H^+(c^-(x))$) に $\mathrm{O}(T(x))$ ずつ分配する. 任意の頂点 x, y と任意の正整数 i, j (ただし $i < H^+(c^-(x)), j < H^+(c^-(y))$) に対して $c_i^+(c^-(x)) \neq c_j^-(c^-(y))$ であるので, 各 $c_i^+(c^-(x))$ には x のみが計算時間を分配する. 具体的な分配の様子を知るために, 図 5 を参照されたい. よって, 分配終了後, 各頂点 x は $\mathrm{O}(T(x))$ 時間を持っていることになる.

点が必ず２つの子供を持てば，１反復あたり $O(T(x))$ 時間で終了する．■

この方法を使用して良い解析結果の得られるアルゴリズムを設計するには，いかに $H^+(c^-(x))$, $H^-(c^+(x))$ の値を算定するか，また，いかに各反復の計算量 $(H^+(c^-(x))+H^-(c^+(x)))T(x)$ を小さくするかが鍵である．この解析方法を使用して，連結な頂点誘導グラフ，大きさ k の連結な頂点誘導グラフ，部分木，大きさ k の部分木の列挙アルゴリズムの高速化が行われており，また，２次元上の点集合の平面的な全張木，２次元の多角形の三角形分割を列挙するアルゴリズムの高速化が期待できる．この中から１つ，部分木を列挙するアルゴリズムを以下で紹介しよう．

与えられた無向グラフ $G=(V,E)$ と G の頂点 r に対し，r を含む G の部分木を列挙する問題を考える．この問題に対して，分割法によるアルゴリズムを構築しよう．まず，列挙を行う前の初期化として，各平行枝の組 $e_1,...,e_k$ を単一の枝 e_0 で置き換える作業を行う．e_i を含む各木は，e_0 を含む木から e_0 と e_i を入れ替えることにより得られるので，この操作により問題の本質は変化しない．初期化後，アルゴリズムはまず，r と r 以外の頂点を結ぶ枝 e^* を分割枝として選ぶ．もしそのような枝 e^* が存在しなかった場合，G の中で r 自身だけが r を含む木となるので，これを出力する．e^* が存在した場合には，元の問題を，e^* と r を含む木を列挙する問題と，e^* を含まず r を含む木を列挙する問題に分割する．e^* と r を含む木は，e^* を縮約して得られるグラフ G/e^* の，縮約してできた頂点 (この頂点も同じく r と表記する) を含む木と１対１対応する．また，e^* を含まず r を含む木は G から e^* を除去して得られるグラフ $G\setminus e^*$ の r を含む木と１対１対応する．よって，これら２つのグラフを作ることにより，元の問題は再帰的に解くことができる．図 6 に，実際に子問題を作った様子を記してある．なお，G/e^* には平行枝が発生する可能性がある．平行枝があってもアルゴリズムは正確に動くが，ならし解析の結果をよくするため，初期化と同じ操作を行い，平行枝は単一の枝で置き換えることにする．頂点 r,v の次数を $d(r),d(v)$ と表記すれば，これら２つのグラフを作成する時間は $\mathrm{O}(d(r)+d(v))$ であるので，このアルゴリズムの計算量は，単純に解析すると１つあたり $\mathrm{O}(|V|)$ となる．

このアルゴリズムの計算時間を上述の分配方法を使ったならし解析で解析してみよう．コンパクト出力を使うことにより，各部分木の出力にかかる時間を $\mathrm{O}(1)$ にできることを注意しておく．このアルゴリズムの生成する列挙木の任意の頂点 x に対し，x が入力したグラフを G_x と表記する．x での分割枝を e_x^* としたとき，x の子供で，G_x/e_x^* を入力とするものを $c^+(x)$, $G_x\setminus e_x^*$ を入力とするものを $c^-(x)$ と定義する．x での 頂点 v の次数を $d_x(v)$ とすると，e_x^* は r に接続しているので，$d_{c^-(x)}(r)=d_x(r)-1$ となる．$d_y(r)=0$ となると，その反復 y は列挙木の葉に対応する．よって，$H^-(x)=d_x(r)$ である．一方，G_x/e_x^* は e_x^* を縮約して得られるグラフなので，縮約してできた頂点 r の次数は $d_x(r)+d_v(r)-2$ となる．G_x/e_x^* は平行枝を含む可能性があるが，１本の枝に平行な枝は高々１本である．よって，すべての平行枝の組を単一枝で置き換えた後の r の次数は $d_{c^+(x)}(r)\geq(d_x(r)+d_v(r)-2)/2$ となる．よって，$H^-(c^+(x))=\mathrm{O}(d_x(r)+d_v(r))$ となり，定理 1 よりこのアルゴリズムの計算量は，１つあたり $\mathrm{O}(1)$ となる．

定理 2 無向グラフの頂点 r を含む木は，１つあたり $O(1)$ 時間で列挙できる．■

なお，上記のアルゴリズムを使用して，無向グラフ G の連結なグラフを誘導する頂点集合を列挙することができる．考えてみていただきたい．この問題に対しては逆探索アルゴリズムが提案されており [1]，１つあたりの計算量は $\mathrm{O}(|V|)$ である．

上記の通り，この分配方法は単純であり，アルゴリズムへの適用可能性の判定も比較的容易である．しかしその反面，適用のできないアルゴリズムも多数存在する．また，この解析がうまく働く

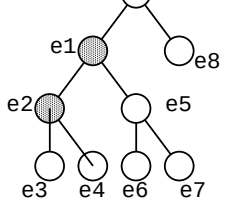

図 7: 灰色の頂点が excess 頂点. e_2 は e_2 から e_8 に, e_3 は e_4 と e_5 に計算時間を分配する.

ようなアルゴリズムの設計指針もあまり詳しいとはいえない. そこで次に, もう一つのならし解析法及び, その解析での結果をよくするようなアルゴリズム設計法を併せて紹介しよう.

2 つ目の方法のアイデアは, 親から子への計算時間の分配ルールを定め, それを列挙木の根に適用し, 次に根の子供に適用し, と分配ルールを根から子孫へ再帰的に適用し, 根の周辺にある多量の計算時間を葉の方へ分配するというものである. 親の持つ計算時間を子供に分配する際には, 子供の持つ子孫の数, または計算時間の量に比例した量を分配する. これにより, 一部の子孫に多量の計算時間が分配されるのを防ぐ. 証明等の詳しい部分は [33] を参照されたい.

さて, では次に具体的な分配ルールを説明するために, 必要な記法をいくつか定義する. まず, 定数 α を解析前に定めるパラメーターとする. $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ をある列挙アルゴリズムの生成した列挙木としよう. $\mathcal{T}$ の頂点 x に対して, $T(x)$ を x で消費された計算時間とする. また, $D(x)$ を x の子孫数とし, $\hat{T}$ を $T(x)/D(x)$ の最大値, すなわち $\max_{x\in\mathcal{V}} T(x)/D(x)$ とする. この分配方法では, 各頂点からその子孫にしか計算時間を分配しないので, $\hat{T}$ は分配後の各頂点の持つ計算時間の下界になっている. また, 頂点 y に分配ルールを適用したときに, y の子供 x が受け取る計算時間を $T_p(x)$ と表記する. 列挙木の根 x_0 では, $T_p(x_0) = 0$ とする.

さて, 以上の記法を用いて, 列挙木の頂点 x に対する分配ルールを説明しよう. x に分配ルールを適用するときは, その親はすでに分配ルールの適用を受けている. であるので, x は $T(x)+T_p(x)$ の計算時間を持っている. この中の $\alpha\hat{T}$ だけ x に残し, 残りの計算時間 $T = T(x)+T_p(x)-\alpha\hat{T}$ を x の各子供 y に以下の 2 つの規則のうちどちらかを使って分配する. 1 つ目は y の子孫数 $D(y)$ に比例する量, すなわち, $T \times D(y)/(D(x)-1)$ を分配し, 2 つ目は y の計算時間に比例する量, すなわち $T \times T(y)/\sum_{u\in C} T(u)$ を分配するものである. ここで, C は x の子供の集合とする. 2 つ目の規則では, ある子供 y に対し, y 以外の子供が y に比べて非常に小さな計算時間しか消費しない場合, y は大量の計算時間を受け取る可能性がある. そこで, $T_p(y)$ が $\alpha\hat{T}D(y)$ よりも大きくなった場合には, 1 つ目の規則で分配を行うことにする.

この分配方法を根から順に適用していくと, 列挙木の葉に近づくにしたがって, だんだんと親から受け取る計算時間が大きくなっていく可能性がある. そこで, もし x の持つ計算時間 $T(x)+T_p(x)$ が $\alpha\hat{T}D(x)$ よりも大きくなったなら, x を excess 頂点と呼び, x には分配ルールは適用しないことにする. ただし, x の子孫に対しては, 引き続き分配ルールを適用することにする. この結果, 任意の頂点 y の受け取る計算時間 $T_p(y)$ は $\alpha\hat{T}D(y)$ よりも小さくなる. これは, x が excess でなければ $T = T(x)+T_p(x)-\alpha\hat{T} \le \alpha\hat{T}(D(x)-1) = \alpha\hat{T}\sum_{x\in C} D(y)$ であることより明らかであろう. よって, 任意の頂点 x に対し, $T_p(x)+T(x) \le (\alpha+1)\hat{T}D(x)$ が導ける. また, 任意の葉 x においては, $D(x) = 1$ であるので, $T(x)+T_p(x) \le (\alpha+1)\hat{T}$ となる. よって, 分配終了後, excess 頂点以外の頂点の持つ計算時間は, すべて $(\alpha+1)\hat{T}$ を越えない.

さて, 今, excess 頂点だけが大量の計算時間を持っている. そこで, 各 excess 頂点 x の計算時間を, $(\alpha+1)\hat{T}$ だけ x にたくわえ, 残りを x の子孫に一様に分配しよう. x の持つ計算時間 $T_p(x)+T(x)$ は $(\alpha+1)\hat{T}D(x)$ よりも小さいので, x の各子孫 y が x から受け取る計算時間は $(\alpha+1)\hat{T}$ を越えない. excess 頂点からの分配の方法を詳しく見るために, 図 7 を参照されたい. ここで, X^* を, 列挙木の根から葉への任意のパスに含まれる excess 頂点の数の最大値としよう. すると, 任意の頂点は多くとも X^* 個の excess 頂点からしか計算時間を受け取らない. よって, excess 頂点からの分配終了後, 各頂点は $(\alpha+1)\hat{T}+(\alpha+1)\hat{T}X^* = \mathrm{O}(X^*\hat{T})$ の計算時間を持つ. よって,

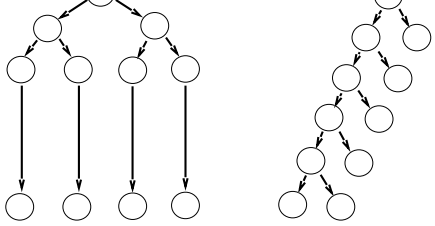

図 8: 左が例 1. 右が例 2. 頂点の計算時間は, 頂点の位置（高さ）に比例しているとみなす.

以下の定理を得る.

定理 3 列挙アルゴリズムは, 1 反復あたり $O(X^*\hat{T})$ 時間で終了する. ∎

さて, 以上で列挙アルゴリズムの計算量を X^* と $\hat{T}$ の積で押さえられることがわかった. 次に, X^* と $\hat{T}$ の効率良い上界を得る方法を紹介しよう.

$\bar{D}(x)$ を $D(x)$ の下界としよう. すると, $T(x)/\bar{D}(x)$ の最大値, すなわち $\max_{x\in\mathcal{V}} T(x)/\bar{D}(x)$ は $\hat{T}$ の上界となる. X^* を押さえるためには, 次の性質を使う. x と y を根から葉の同一のパス上にある excess 頂点とする. ただし C は w の子供の集合とする.

補題 1 もし, 全ての頂点に対して分配ルール 1 が適用されたなら, x と y を結ぶパスのある頂点 w が存在して, $\bar{D}(w) \geq \sum_{u\in C} \frac{\alpha}{\alpha+1}\bar{D}(u)$ が成り立つ. ∎

補題 2 もし, 全ての頂点に対して分配ルール 1 が適用されたなら, x と y を結ぶパスのある頂点 w が存在して, $T(y) \geq \sum_{u\in C} \frac{\alpha}{\alpha+1}T(u)$ が成り立つ. ∎

これら性質を持つ頂点の数の上界は X^* の上界でもあり, また比較的簡単に得られる.

以上により, $\hat{T}$ と X^* を効率よく押さえることができた. これで, 列挙アルゴリズムの計算量を, より小さく押さえることができる. しかし, 列挙アルゴリズムの中には, 大きな $\hat{T}$ や大きな X^* を生成するものがある. 図 8 を見ていただきたい。1 つ目の例は, ある頂点の計算時間がその子孫数に比べて大きい場合である. この場合, $\hat{T}$ が大きくなる. 2 つ目の例は, 列挙木の形がパスのようになっている場合である. この場合, 頂点の計算時間の分布により, 列挙木のほとんどの内点が excess 頂点となる可能性がある. 列挙木がこれらの部分木を含む場合, $\hat{T}$ や X^* が大きくなる可能性がある. X^* や $\hat{T}$ が大きくなれば必ずこのような部分木を含むわけではないが, これらの例は非常に典型的である. そこで, これらの悪い部分木を発生させないため, 列挙アルゴリズムの各反復に, trimming フェイズと balancing フェイズという 2 つのフェイズを付け加えよう.

1 つ目の trimming フェイズでは, 入力から無駄な部分を省き, 入力を子孫数に対して小さくする. これは 1 つ目の悪い例を避けるために付け加えるものである. 2 つ目の balancing フェイズでは, 子問題を発生させるときに子問題の大きさのバランスをとり, 1 つの大きな問題といくつかの小さな問題に分割することを防ぐものである. これにより, 列挙木がパスのようになることはなくなり, よって, 2 つ目の悪い例を避けることができる.

この 2 つのフェイズを加えることにより, 多くの列挙アルゴリズムの高速化を行うことができる. 有向グラフの有向根付き木, 2 部グラフの完全マッチング, 2 部グラフの最大マッチング, 無向グラフの 2 頂点を結ぶパス, マトロイドの基の列挙を行うアルゴリズムが高速化される. また, 無向グラフのアサイクリックな向き付けの列挙を行うアルゴリズムの高速化が期待できる. ここでは, この中から 2 頂点を結ぶパスを列挙するアルゴリズムの高速化を紹介しよう.

与えられた無向グラフ $G = (V, E)$ と G の頂点 s と t に対して, s と t を結ぶ単純なパスを列挙する問題を考える. 単純なパスとは閉路を含まないようなパスのことである. 以下ではこのようなパスを単に s-t パスと呼ぶ. この問題に対しては, 分割法がうまく働く. 分割法は, s に接続する枝 e^* を分割枝として一つ選び, s-t パスの集合を e^* を含む s-t パスの集合と, e^* を含まない s-t

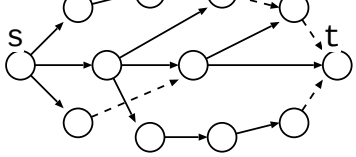

図 9: F の例. 実線の枝が F^* の枝.

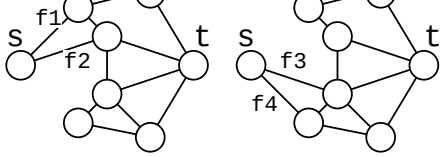

図 10: 左が f_1 か f_2 を含む s-t パス列挙のためのグラフ, 右が f_3 か f_4 を含む s-t パス列挙のためのグラフ

パスの集合に分割する. e^* を含む s-t パスは e^* をコントラクトして得られるグラフの s-t パスになり, (コントラクトして得られた頂点を s とする.) また, e^* を含まない s-t パスは e^* を除去して得られるグラフの s-t パスになっている. よって, この列挙問題は分割法により, 再帰的に解くことができる. 列挙アルゴリズムの 1 反復の計算量は $\mathrm{O}(|E|+|V|)$ となる [23].

さて次に, このアルゴリズムに trimming フェイズと balancing フェイズを加え, 高速化しよう. trimming フェイズは, 1 反復の計算時間を子孫数に対して小さくする. そのために, 全ての s-t パスに含まれる枝はコントラクトし, また, どの s-t パスにも含まれない枝は除去する. 枝 (s,t) を G に加えると, s-t パスに含まれる枝は, (s,t) と同じ 2 頂点連結成分に含まれる. これで, どの s-t パスにも含まれない枝は識別できる. これらの枝を除去した後, G を 2 枝連結成分分解すると, 全ての s-t パスに含まれる枝は橋になるので, これも識別できる. 計算時間は $\mathrm{O}(|E|+|V|)$ である.

また, グラフが次数 2 の頂点を多く含むと, 頂点数に対して s-t パスの数は少なくなる. 例えば, グラフが長い 1 本の s-t パスで構成されているような場合である. そこで, 次数 2 の頂点が存在したときにはその頂点に接続する 2 つの枝を 1 本の枝で置き換える. これにより, グラフから次数 2 の頂点を除去することができる. この計算時間は $\mathrm{O}(|V|)$ である.

次に, trimming フェイズが終わった後のグラフ G がいくつの s-t パスを含むかを調べよう. そのために, G の部分グラフに向き付けを行った有向グラフ F , 及び F に含まれる根付き木 F^* を作る. ただし, F , F^* は以下の性質を満たす. (1) G の任意の頂点及び F の任意の枝は, F 中のある有向 s-t パスに含まれる. (2) $F \setminus F^*$ の枝は全て F^* の葉に尾を持つ. このようなグラフは $\mathrm{O}(|E|+|V|)$ で作成できることを記しておく. F 及び F^* の例を図 9 に示してあるので, 参照されたい. さてここで, F^* に含まれない枝 (u,v) について, F^* 中の s から u への有向パスと, 有向枝 (u,v) と F に含まれる v から t への有向パスをあわせて得られるパスか, u と v を入れ替えて, 同じ操作をしてえられるパスが, (u,v) を含む G の s-t パス P になっている. (u,v) の, P の中で s に近いほうの頂点を (u,v) の尾とし, そうでないほうを頭とする. P の中を s から t へとたどると, 初めて現れる F^* に含まれない枝は (u,v) である. よって, この操作により, F^* に含まれない枝それぞれに, 異なる s-t パスを作ることができる. よって, G には少なくとも, $(|E|-|F^*|) = |E|-|V|+1$ 本のパスが含まれることになる. G は次数 2 以下の頂点を含まないので, $|E| \geq 1.5|V|$, よって, $D(x) \geq |E|/3$ を得る.

次に balancing フェイズを解説しよう. まず, F^* の各枝 e に対し, e の重み $w(e)$ を, 尾が F^* での e の子孫である枝の数とする. また, $M = (|E|-|V|)/3$ とし, r に接続する F^* の枝の集合を $S = \{f_1, ..., f_k\}$ とする. 上述より枝 $f \in S$ を含む s-t パスは, 少なくとも $w(f)$ 本存在する.

さて, 以上の性質を用いて, 子問題の出力数が M 以上となるような問題の分割法を, 3 通りに場合分けして説明しよう.

1. $M \leq w(f) \leq 2M$ なる枝 $f \in S$ が存在する場合. f を分割枝に選ぶと, 子問題 G/f は M 本以上の s-t パスを含み, また $w(f) \leq 2M$ より $G \setminus f$ も少なくとも M 本以上の s-t パスを含む.

2. 任意の $f \in S$ について $w(f) < M$ である場合. ある l が存在して, $M \geq \sum_{i=1}^{l} w(f_i) \geq 2M$ となる. よって, G の s-t パスの集合を, G の $f_1, ..., f_l$ を含む s-t パスの集合と, $f_{l+1}, ..., f_k$ を含む s-t パスの集合に分割し, 2 つの子問題を作成する. 前者は, G から $f_{l+1}, ..., f_k$ を取り除いて得られるグラフの s-t パスの集合であり, 後者は, G から $f_1, ..., f_l$ を取り除いて得られるグラフの s-t パスの集合である. この場合も, 両子問題は M 本以上の s-t パスを含む. 具体例として, 両子

ら, f を分割枝として問題を分割すると, 両子問題の出力数は M 以上になる. M 以下である場合は, 任意の f の s でない端点に接続する枝 g を含む s-t パスは, 必ず f を含むことが証明できる. よって, G の s-t パスは, f をコントラクトしたグラフの s-t パスと一対一対応する. そこで, f をコントラクトして, 再び上記の方法で分割を試みる. それでもある枝 f' が $w(f') > 2M$ を満たす場合は f' をコントラクトして, さらに上記の操作をくり返す.

以上の操作を行って得られる分割枝を使って問題を子問題 x_1, x_2 に分割すると, 両子問題は M 本以上のパスを出力する. よって, $D(x_i) \geq M/3 = (|E| - |V|)/3 \geq |E|/9$ を得る. 上記の操作は, 1,2,3 で 1 回分割を試みるのに $\mathrm{O}(|E|)$ の時間がかかり, 3 の繰り返し操作により, 分割を最大で $\mathrm{O}(|V|)$ 回試みる可能性がある. しかし, 繰り返しの部分を 2 分探索で行うことにより, 計算時間を $\mathrm{O}(|E|\log|V|)$ まで少なくすることができる.

さて, ここからは, このアルゴリズムの計算量を得るために, 解析を行おう. まず, α を 6 と定める. グラフ $G_x = (V_x, E_x)$ を入力とする反復 x に対して, 上記の議論より, $\bar{D}(x)$ を $|E_x|/3$ とできる. よって, $\hat{T} \leq |E|\log|V|/|E_x|/3 = \mathrm{O}(\log|V|)$ を得る. また, 問題の分割の仕方 (balancing フェイズ) より, x の子問題 y に対して, $\bar{D}(y) \geq (|E_x| - |V_x|)/9 \geq \bar{D}(x)/3$ を得る. さて, 補題 2 より, X^* は, 列挙木の根から葉へのパスの中で $\bar{D}(x) \geq \frac{7}{6}(\bar{D}(y_1) + \bar{D}(y_2))$ が成り立つ頂点の数の最大値で押さえられる. ただし, y_1 と y_2 は x の子供である. 今, x が, この式を満たすとする. このとき, $\bar{D}(y_i) \geq \bar{D}(x)/3$ より, $\bar{D}(y_i) \leq \frac{5}{6}(\bar{D}(y_1)+$ を満たす. よって, 列挙木の根から葉へのパスの中には, 多くとも $\log_{6/5}|E|$ 個しか, 上記の式を満たす頂点は存在しない. よって, $X^* = \mathrm{O}(\log|V|)$ を得る.

定理 4 無向グラフの s-t パスの列挙は, 1 つあたり $O(\log^2|V|)$ 時間で行える.

これで, 高速化についての研究の紹介は一通り終わった. 他にもいくつかの列挙アルゴリズム高速化の研究が存在するので, 興味のある方は参照されたい.

4 これからの課題

最後に, この分野に関する未解決問題と, 最近著者が考える課題を幾つか述べておこう.

未解決問題として有名なものは, 多面体の端点列挙, 及び極小集合被覆の列挙を行う出力数多項式のアルゴリズムが存在するかどうかである. 前者は, 非退化である単純な多面体に関しては, 本稿で解説した辞書列挙アルゴリズムが出力数多項式アルゴリズムになっている. 後者に関しては, 現在, 1 つあたり入力の大きさ n に対して $\mathrm{O}(n^{\log n})$ 時間のアルゴリズムが知られているが, これは多項式アルゴリズムではない. これらの問題は, 多くの研究者たちによって精力的に研究されているようだが, 現在も未解決である. 他に, マトロイドのサーキット, 有向グラフの極大部分アサイクリックグラフの列挙を行う出力数線形アルゴリズムが存在するかどうかも未解決である.

最適化問題への応用という観点からは, 局所最適解の列挙は大きな課題である. グラフの安定集合に対しては極大解の列挙アルゴリズムが提案されているが, 極大要素の列挙は難しいものが多い. 例えば巡回セールスマン問題の 2-opt 解の列挙, 充足可能性問題の極大充足解の列挙など, 未解決な問題が多い. これらの問題に対する効果的な解決法が見つかれば, 最適化問題を解く場合に, これらの局所最適解の総数が少ないような近傍を定義し, その局所最適解を列挙することにより, 問題は高速に解けるであろう. また, 多目的関数最適化問題などで, k-best 問題がよく利用されるが, これに関してもまだまだ研究が行われていない分野が多い.

次に著者の考える課題を述べよう. まずは, 本稿の主題である高速化の課題から. 過去に提案されてきた多くの列挙アルゴリズムには, 高速化の研究がない. これらの高速化が研究課題の 1 つである. また, 高速化手法, 及び計算時間の算定方法に関してもまだまだ研究は少ない. 本稿の解析手法は, ある反復の計算時間をその反復の子孫に分配することにより計算時間をならす. しかし, この解析方法では, グラフの極小カットの列挙アルゴリズムのように, 列挙木の局所に計算時間が集中するようなアルゴリズムの計算時間をうまく算定することはできない. このアルゴリズムを

法の開発が不可欠である. また, k best 問題のアルゴリズムや, 与えられた数 k 個だけ要素を出力するアルゴリズムに対しても, 高速化が研究されていない分野が多く, またそれらのアルゴリズムに対する一般的な高速化手法の提案も行われていない.

高速化が行える, または行いやすい問題のクラスの研究もされるべきであろう. 例えば, コンパクト出力が行えるかどうかと高速化は関係あるのか, また, 集合族の列挙で, 要素全体が作る 0-1 多面体での解の隣接関係と高速化はどう関わっているのかなど, 今後の研究が期待される.

これら列挙アルゴリズム設計・高速化手法の応用先としてパラメトリック最適化問題の解列挙がある. これは一般の列挙問題より比較的構造が複雑で, ゆえにアルゴリズム及びその高速化の研究は少ない. 現在提案されているアルゴリズムの多くはパラメーターを変化させて逐次的に解を求める方法 [11] , 及びそれをデータ構造を使って高速化したものである. これらの問題に対する列挙問題の手法の適用がどの程度効果的であるのかはまだ明らかにされていない.

列挙アルゴリズムから, 集合の要素数を計算する問題への応用も研究が期待される. 要素数を計算する問題は出力量が大きいという列挙問題の制約はなく, 高速化の余地が大きい. また, 同時に一様発生などの問題への応用も期待される. これらの問題がいろいろな対象に対して効率よく解ければ, 例えば, マルチスタート局所探索など, 多くの応用が期待できる.

謝辞

本稿を作成するにあたりご協力いただいた, 東海大学の松井泰子先生, 東京大学の松井知己先生, 上智大学の塩浦昭義先生に感謝いたします. また, 日頃より研究に対しての助言をいただいた, 茗荷谷ゼミのメンバーにも感謝いたします.

参考文献

[1] D. Avis and K. Fukuda, “Reverse Search for Enumeration,” Discrete Appl. Math. **65**, 21-46 (1996).

[2] C. R. Chegireddy and H. W. Hamacher, “Algorithms for Finding K-best Perfect Matchings,” Discrete Appl. Math., **18**, 155-165 (1987).

[3] C. J. Colbourn, R. P. J. Day, and L. D. Nel, “Unranking and Ranking Spanning Trees of a Graph,” Journal of Algorithms **10**, 271–286 (1989).

[4] D. Eppstein, “Finding the k Smallest Spanning Trees,” SWAT ’90 Springer Verlag Lec. Note Comp. Sci. **447**, 38-47 (1990).

[5] D. Eppstein, “Finding the k shortest paths,” 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, 20-22 November, 154-165 (1994).

[6] M. C. ER “A New Algorithm for Generating Binary Trees Using Rotations,” The Computing Journal **32**, No. 5, 470-473 (1989).

[7] K. Fukuda, “逆探索とその応用,” 離散構造とアルゴリズム II, 47-78. (1993).

[8] K. Fukuda and T. Matsui, “Finding All the Perfect Matchings in Bipartite Graphs,” Appl. Math. Lett. **7**, No. 1, 15-18 (1994).

[9] G. N. Frederickson, “Data Structures for On-Line Updating of Minimum Spanning Trees, Wtih Applications,” SIAM J. Comp. **14**, No. 4, 781-798 (1985).

[10] H. N. Gabow and E. W. Myers, “Finding All Spanning Trees of Directed and Undirected Graphs,” SIAM J. Comp. **7**, 280-287 (1978).

[11] G. Gallo, M. D. Grigoriadis and R. E. Tarjan, “A fast parametric maximum flow algorithm and applications,” SIAM Journal on Computing **18**, 30-55 (1989).

[12] H. W. Hamacher, J. C. Picard and M. Queyranne, “On Finding the K best Cuts in a Network,” Operations Research Letters, **2**, 303-305 (1984).

[13] Hiroshi Imai, Satoru Iwata, Kyoko Sekine and Kenshu Yoshida, “Combinatorial and Geometric Approaches to Counting Problems on Linear Matroids, Graphic Arrangements and Partial Orders,” Lect. Note in Comp. Sci. **1020**, 68-80 (1996).

[15] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, "On Generating All Maximal Independent Sets," Info. Processing Lett. **27**, 119-123 (1988).
[16] H. N. Kapoor and H. Ramesh, "Algorithms for Generating All Spanning Trees of Undirected, Directed and Weighted Graphs," Lecture Notes in Computer Science, Springer-Verlag, 461-472, (1992).
[17] T. Kashiwabara, S. Masuda, K. Nakajima and T. Fujisawa, "Generation of Maximum Independent Sets of a Bipartite Graph and Maximum Cliques of a Circular-Arc Graph," J. Algorithms, **13**, 161-174 (1992).
[18] N. Katou, T. Ibaraki and H. Mine, "An Algorithm for Finding K Minimum Spanning Trees," SIAM J. Comp. **10**, 247-255 (1981).
[19] Y. Matsui, T. Matsui and K. Fukuda, "Home page of Enumeration Algorithms," http://dmawww.epfl.ch/roso.mosaic/kf/enum/enum.html
[20] K. G. Murty, "An Algorithm for Ranking All the Assignments in Order of Increasing Cost," Operations Research, **16**, 682-687 (1968).
[21] M. W. Padberg, "On the Facial Structure of Set Packing Polyhedra," Mathematical Programming, **5**, 199-216 (1973).
[22] G. L. Nemhauser and G. Sigismondi, "A Strong Cutting Plane / Branch-and-Bound Algorithm for Node Packing ", Journal of Oprational Research Society (1992).
[23] R. C. Read and R. E. Tarjan, "Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees," Networks **5**, 237-252 (1975).
[24] R. Seidel, "Small-Dimensional Linear Programming and Convex Hulls Made Easy," Discrete and Computational Geometry **6**, 423-434 (1991).
[25] A. Shioura, A. Tamura and T. Uno, "An Optimal Algorithm for Scanning All Spanning Trees of Undirected Graphs, " SIAM J. Comp. **26**, No. 3, 678-692 (1997).
[26] S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa, "A New Algorithm for Generating All the Maximum Independent Sets," SIAM J. on Comput., **6**, No. 3: 505-517 (1977).
[27] S. Tsukiyama, I. Shirakawa and H. Ozaki, "An Algorithm to Enumerate All Cutsets of a Graph in Linear Time per Cutset," J. ACM., **27**, 619-632 (1980).
[28] T. Uno, "An Algorithm for Enumerating All Directed Spanning Trees in a Directed Graph," Lect. Note in Comp. Sci. **1178**, Springer-Verlag, Algorithms and Computation, 166-173 (1996).
[29] Y. Matsui and T. Uno, "A Simple and Fast Algorithm for Enumerating all Edge Colorings of a Bipartite Graph," Research Report, Dept. Math. and Comp., Tokyo Institute of Technology, Japan.
[30] T. Uno, "Algorithms for Enumerating All Perfect, Maximum and Maximal Matching s in Bipartite Graphs," Lecture Note in Computer Science **1350**, Springer-Verlag, Algorithms and Computation, 92-101 (1997).
[31] T. Uno, "A Fast Algorithm for Enumeration of Maximal Matchings in General Graphs," Research Report, Dept. Math. and Comp., Tokyo Institute of Technology, Japan.
[32] T. Uno, "Studies on Speeding Up Enumeration Algorithms," Doctral Thesis, Dept. Systems Science, Tokyo Institute of Technology (1998).
[33] T. Uno, "A New Approach for Speeding Up Enumeration Algorithms," Lecture Note in Computer Science **1533**, Springer-Verlag, Algorithms and Computation, 287-296 (1998).
[34] T. Uno, "A New Approach for Speeding Up Enumeration Algorithms and Its Application for Matroid Bases," Lecture Note in Computer Science **1627**, Springer-Verlag, Computing and Combinatorics (Proceeding of COCOON99), 349-359, (1999)