数理解析研究所講究録
1120巻 1999年 1-10

列挙アルゴリズムの高速化技法とその応用

A Speeding up Technique for Enumeration Algorithms and its Application for Perfect Matching

宇野 毅明
Takeaki UNO

東京工業大学経営工学専攻
〒 152-0033 東京都目黒区大岡山 2-12-1 uno@me.titech.ac.jp

摘要: 本稿では列挙アルゴリズムの高速化手法を提案する. 本稿の高速化手法は, 一般的に行われている反復の高速化を利用した手法とは違い, 列挙アルゴリズムに対する新しい時間計算量の解析技法を提案し, その解析技法でより小さな計算量の上界が得られるようなアルゴリズムの設計方法に主眼を置いている. この高速化手法を用いた列挙アルゴリズムの解析を行う場合, いくつかの数値の上界・下界を求める必要がある. 本稿ではそれらを求めるための道具立てになる性質もいくつか証明する. さらに, この高速化手法を用いて高速化された2部グラフの完全マッチングの列挙アルゴリズムもあわせて紹介する.

キーワード: 列挙アルゴリズム, 高速化手法, 時間計算量解析

1 導入 ・ 列挙問題とアルゴリズム

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, 全張木であるものを出力せよ, 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 [9], 無向グラフの全張木の列挙 [4, 7, 10, 8, 16], 有向グラフの2点を結ぶパスの列挙 [8] 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 [5] 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.

列挙アルゴリズムは一般に多くの計算時間を必要とする. ゆえに高速化は大きな課題の一つである. 過去にも高速化に関して多くの研究が行われてきたが, ほとんどの研究がある特定の列挙アルゴリズムを高速化する, という研究にとどまっている. アルゴリズムの研究を進めていくという視点からは, 列挙アルゴリズムに対する一般的な高速化, という観点での研究が重要であろう. そこで, 本稿では列挙アルゴリズムに対する一般的な高速化手法の提案を行う. あわせて, 列挙アルゴリズムの時間計算量を解析する一般的な技法も提案する. この解析技法を使って時間計算量の解析を行うことにより, 提案する高速化手法を用いた列挙アルゴリズムの時間計算量を小さくおさえることができる. この高速化手法は全ての列挙アルゴリズムを高速化できるわけではないが, 過去の研究では高速化を行えなかったようなアルゴリズムに対しても, 効果的に高速化を行うことができる. この高速化手法, 及びその実例に関しては, [14, 15, 16] も参照されたい. 以下で, この高速化手法と解析技法の解説を行うが, その前に準備として, 列挙アルゴリズムの時間計算量を算定する尺度と, 解析を行う際に使用する列挙木について説明を行う.

一般に, アルゴリズムの時間計算量は, 入力の大

きさ n に関する式で表される. 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ので, 入力の大きさだけで算定すると, 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. 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, $O(f(n)N)$ 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, 解の出力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.

列挙アルゴリズムの解説や計算量の解析を行うときに, 列挙木と呼ばれる構造がよく用いられる. 以下の節で列挙木を用いるため, ここで列挙木の簡単な解説を行っておこう. 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. 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.

次の節からは, 本稿の目的である列挙アルゴリズムの高速化手法の提案を行っていく. まず, 2節で列挙アルゴリズムの解析技法の提案を行い, また, 解析を行う際に役立つ補題をいくつか証明する. 次に, 3章で高速化手法を提案する. そして, 4章でこの高速化手法により2部グラフの完全マッチングを列挙するアルゴリズムを高速化し, その計算量を2章で紹介した補題と解析技法を使って解析する.

2 列挙アルゴリズムの計算量解析技法 ・ 計算時間の均一化

この節では, 列挙アルゴリズムの新しい解析手法について述べる. 過去の研究においても, 全張木やパス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 [7, 10, 8, 16, 7, 12, 14]. 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. 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のがこの方法のアイディアである. 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.

一般に, 再帰型, あるいは反復型屋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. なぜなら, 列挙アルゴリズムの多くは, 一般に子問題の計算時間のほうが親問題よりも小さく, また, 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からである. すなわち, この方法で計算時間を算定しても, 実際の計算時間とはかけ離れている場合が存在するのである.

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が提案されてきている. 本稿の技法も, このならし解析を使用している. 以下にその技法を紹介しよう.

列挙アルゴリズムに対し, その列挙木を考える. 列挙木の各頂点は, その頂点に対応する反復で使用した計算時間を持つものとする. ただし, 各反復の計算時間とは, その子問題の計算時間を含まないものとする. このとき, 列挙木の根に近い頂点は多くの計算時間を持ち, また, 列挙木の葉に近い頂点は少ない計算時間を持つと考えられる. そこで, 列挙木の親頂点から子供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操作を行い, 計算時間をならそうというのが本稿の技法のアイディアである. 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.

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の上界とする. 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, $T\times D(y)/(D(x)-1)$ を分配するもので, 2 つ目は y の計算時間に比例する量, すなわち $T\times T(y)/\sum_{u\in C(x)} T(u)$ を分配するものである. ここで, $C(x)$ 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 $T=T(x)+T_p(x)-\alpha\hat{T}\le\alpha\hat{T}(D(x)-1)=\alpha\hat{T}\sum_{y\in C(x)}D(y)$ 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. 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, 以下の定理を得る.

定理 1 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. また, 子孫数の下界を示す手助けとして, 以下の補題を証明しておく. $f(x)$ を, 列挙木の頂点 x に対し整数値を与える関数とする. x が葉であるとき, $f(x)=\mathrm{O}(1)$ であるとする.

補題 1 任意の頂点 x が $f(x)\le\sum_{y\in C(x)} f(y)+O(1)$ を満たすならば, $D(x)=\Omega(f(x))$ である.

Proof: $D(x)\ge 1$ であるので, 仮定より, ある定数 ϵ が存在して $f(x)=\mathrm{O}(1)$ のとき, $D(x)\ge f(x)/\epsilon$ が成り立ち, また, 任意の頂点で $f(x)\le\epsilon+\sum_{y\in C(x)} f(y)$ が成り立つ. そこで, $f(x)<k$ であるときに $D(x)\ge\epsilon f(x)$ であると仮定する. ここ

で x が $f(x)=k$ を満たすとき, 仮定より,

$$\begin{aligned} D(x) &= 1+\sum_{y\in C(x)} D(y) \\ &\geq 1+\sum_{y\in C(x)} f(y)/\epsilon \\ &\geq f(x)/\epsilon \end{aligned}$$

である. よって, 帰納法により, 任意の k について $D(x)\geq f(x)/\epsilon$ が成り立つ. ∎

補題 2 任意の頂点 x の a 人以上の子供 y が $f(x)\leq bf(y)$ を満たすならば, $D(x)=\Omega(a^{\log_b f(x)})=\Omega(f(x)^{\log_b a})$ である.

Proof: $D(x)\geq 1$ であるので, ある定数 ϵ が存在して $f(x)=$O(1) のとき, $D(x)\geq f(x)/\epsilon$ が成り立つ. そこで, $f(x)<k$ であるときに $D(x)\geq a^{\log_b f(x)}/\epsilon$ であると仮定する. ここで x が $f(x)=k$ を満たすとき, 仮定より,

$$\begin{aligned} D(x) &= 1+\sum_{y\in C(x)} D(y) \\ &\geq 1+a\epsilon f(y) \\ &\geq f(x)/\epsilon \end{aligned}$$

である. よって, 帰納法により, 任意の k について, $D(x)\geq \epsilon f(x)$ が成り立つ. ∎

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. 列挙木上で y が x の先祖であるとする.

補題 3 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 を結ぶパス上にある頂点 w が存在して, (1)$\bar{D}(w)\geq\sum_{u\in C(w)}\frac{\alpha}{\alpha+1}\bar{D}(u)$ が成り立つ. ∎

Proof: この補題を証明するために, 全ての $w\in P_{yx}\setminus y$ が (1) を満たさないと仮定しよう. 任意の y の子供 w は, $T_p(w)<(\alpha-1)\hat{T}(w)$ を満たす. ここで, ある頂点 $w\in P_{yx}\setminus y$ の親 w' が $T_p(w')\leq(\alpha-1)T(w')$ を満たすとしよう. すると, 補題の仮定より,

$$\begin{aligned} T_p(w) &= \frac{(T_p(w')+T(w')-\alpha\hat{T})D(w)}{D(w')-1} \\ &\leq \frac{\alpha T(w')D(w)}{\sum_{u\in C(w)}\bar{D}(u)} \\ &\leq \frac{\alpha\hat{T}\bar{D}D(w)}{\sum_{u\in C(w)}\bar{D}(u)} \\ &\leq \frac{\alpha\hat{T}D(w)\frac{\alpha-1}{\alpha}\sum_{u\in C(w)}\bar{D}(u)}{\sum_{u\in C(w)}\bar{D}(u)} \\ &\leq (\alpha-1)D(w)\hat{T} \end{aligned}$$

となる. ここで, $T(w)\leq\hat{T}D(w)$ であったことに注意すると, $T_p(w)\leq(\alpha-1)\hat{T}D(w)$ であることが導ける. これより, 頂点 x は, $T_p(x)+T(x)\leq\alpha\hat{T}D(x)$ を満たすことになり, これは, x が excess 頂点であるという仮定に矛盾する. ∎

補題 4 もし, 全ての頂点に対して分配ルール 2 が適用されたなら, x と y を結ぶパス上にある頂点 w が存在して, (2)$T(w)\geq\sum_{u\in C(w)}\frac{\alpha}{\alpha+1}T(u)$ が成り立つ. ∎

Proof: この補題を証明するために, 全ての $w\in P_{yx}\setminus y$ が (2) を満たさないと仮定しよう. 任意の y の子供 w は, $T_p(w)<(\alpha-1)\hat{T}D(w)$ を満たす. ここで, ある頂点 $w\in P_{yx}\setminus y$ の親 w' が $T_p(w')\leq(\alpha-1)T(w')$ を満たすとしよう. すると, 補題の仮定より,

$$\begin{aligned} T_p(w) &= \frac{(T_p(w')+T(w')-\alpha\hat{T})T(w)}{\sum_{u\in C(w')}T(u)} \\ &\leq \frac{\alpha T(w')T(w)}{\sum_{u\in C(w')}T(u)} \\ &\leq \frac{\alpha T(w)\frac{\alpha-1}{\alpha}\sum_{u\in C(w)}T(u)}{\sum_{u\in C(w')}T(u)} \\ &= (\alpha-1)T(w) \end{aligned}$$

となる. ここで, $T(w)\leq\hat{T}D(w)$ であったことに注意すると, $T_p(w)\leq(\alpha-1)\hat{T}D(w)$ であることが導ける. これより, 頂点 x は, $T_p(x)+T(x)\leq\alpha\hat{T}D(x)$ を満たすことになり, これは, x が excess 頂点であるという仮定に矛盾する. ∎

これらの性質を持つ頂点の数の上界は X^*-1 の上界でもあり, また以下の条件より比較的簡単に得ることができる. x_0 を列挙木の根とする.

補題 5 列挙木の任意の頂点 x の子孫数の下界 $\bar{D}(x)$ は, 葉の方向に単調減少, つまり, x の任意の子供 y について, $\bar{D}(x)\geq\bar{D}(y)$ が成り立つとする. C を定数として, $\bar{D}(w)$ が $4C$ より

大きいような任意の頂点 w の子供 $C(w)$ が 2 つの集合 C_1 と C_2 に分割できて, $\sum_{u\in C_i}\bar{D}(u) \geq (1/C)\bar{D}(w) - C$ を $i = 1, 2$ について満たすとき, $X^* = O(\log_{C/(C-1)}\bar{D}(x_0))$ である.

Proof : 定数 $\alpha = 2C$ とすると, $(1)\bar{D}(w) \geq \sum_{u\in C(w)}\frac{\alpha}{\alpha+1}\bar{D}(u)$ が成り立つような頂点 w では, $\frac{2C+1}{2C}\bar{D}(w) - \sum_{u\in C_1}\bar{D}(u) \geq \sum_{u\in C_2}\bar{D}(u)$ が成り立つ. よって, 補題の仮定より $\sum_{u\in C_i}\bar{D}(u) \geq (2/2C)\bar{D}(w) - C$ であるので,

$$\begin{aligned}\sum_{u\in C_2}\bar{D}(u) &\leq \frac{2C+1}{2C}\bar{D}(w) - \sum_{u\in C_1}\bar{D}(u)\\ &\leq \frac{2C+1}{2C}\bar{D}(u) - (2/C2)\bar{D}(u) + C\\ &\leq \frac{2C-1}{2C}\bar{D}(w) + \bar{D}(w)/4C\\ &= \frac{4C-1}{4C}\bar{D}(w)\end{aligned}$$

を得る. 同様にして, $\sum_{u\in C_1}\bar{D}(u) \leq \frac{4C-1}{4C}\bar{D}(w)$ を得る. よって, 補題 3 の (1) を満たすような頂点 w の子供の計算時間は, 親に比べてると少なくとも $\frac{4C-1}{4C}$ 以下である. よって, 列挙木の根から葉のパス上に (1) を満たす頂点は多くとも $\log_{4C/(4C-1)}\bar{D}(x_0)$ 個しか存在しない. よって, 補題 3 より, $X^* = \mathrm{O}(\log_{C/(C-1)}\bar{D}(x_0))$ を得る.

補題 6 列挙木の任意の頂点 x の計算時間の上界が, 葉の方向に単調減少, つまり, x の任意の子供 y について, $T(x) \geq T(y)$ が成り立つとする. C を定数として, 任意の葉 x で $T(x) = f$ であるとする. $T(w)/f$ がある定数 $4C^2$ より大きいような任意の頂点 w の子供 $C(w)$ が 2 つの集合 C_1 と C_2 に分割できて, $\sum_{u\in C_i}T(u) \geq (1/C)T(w) - Cf$ を $i = 1, 2$ について満たすとき, $X^* = O(\log_{C/(C-1)}T(x_0)/f)$ である. ただし, $T(w)/f$ がある定数 $4C^2$ より小さいとき, w の子孫数は定数であるとする.

Proof : 定数 $\alpha = 2C$ とすると, $(2)T(w) \geq \sum_{u\in C(w)}\frac{2C}{2C+1}T(u)$ が成り立つような頂点 w では, $\frac{2C+1}{2C}T(w) - \sum_{u\in C_1}T(u) \geq \sum_{u\in C_2}T(u)$ が成り立つ. よって, 補題の仮定より $\sum_{u\in C_i}T(u) \geq (2/2C)T(w) - Cf, T(w)/4C \geq Cf$ であるので,

$$\begin{aligned}\sum_{u\in C_2}T(u) &\leq \frac{2C+1}{2C}T(w) - \sum_{u\in C_1}T(u)\\ &\leq \frac{2C+1}{2C}T(w) - (2/C2)T(w) + Cf\\ &\leq \frac{2C-1}{2C}T(w) + T(w)/4C\\ &= \frac{4C-1}{4C}T(w)\end{aligned}$$

を得る. 同様にして, $\sum_{u\in C_1}T(u) \leq \frac{4C-1}{4C}T(w)$ を得る. よって, 補題 4(2) を満たすような頂点 w の子供の計算時間は, 親に比べてると少なくとも $\frac{4C-1}{4C}$ 以下である. よって, 列挙木の根から葉のパス上に (2) を満たす頂点は多くとも $\log_{4C/(4C-1)}T(x_0)/f$ 個しか存在しない. よって, 補題 4 より, $X^* = \mathrm{O}(\log_{C/(C-1)}T(x_0)/f)$ を得る. ∎

3 列挙アルゴリズムの高速化技法・縮約平衡法

この節で提案する縮約平衡法は, 前節の解析方法を利用した列挙アルゴリズム高速化手法である. 前節の解析手法は, $\hat{T}$ と X^* を使って効率よく計算量を押さえることに成功しているが, 大きな $\hat{T}$ や大きな X^* を生成する列挙アルゴリズムに対しては効果がない. そこで, 前節の解析がうまく働くように, アルゴリズムに 2 つのフェイズを加えて改良しようというのが縮約平衡法のアイディアである.

1 つ目の縮約フェイズでは, 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. 縮約フェイズを加えることにより, $\hat{T}$ を小さくすることができる. 2 つ目の平衡フェイズでは, 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が発生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, バランスをとることが出きる. バランスがとれた木では, 補題 3 や補題 4 の条件が成り立つ頂点の子供は, 大きさがある程度小さくなることが多い. ゆえに, X^* を小さな値で押さえることができる. 以下に, 縮約平衡法を利用した列挙アルゴリズムの概略を示そう.

ALGORITHM ENUMERATION (X)
Step 1: X に縮約フェイズを施す
Step 2: $k :=$ 平衡フェイズが作る子問題の数
Step 3: **For** i := 1 **to** k
Step 4: 平衡フェイズで子問題 i の入力 X_i を作る

Step 5: ENUMERATION (X_i) を呼び出す
Step 6: End for

このアルゴリズムを以下のように書きなおすと，アルゴリズム ENUMERATION の入力は全て縮約フェイズを施されていると仮定できる．このほうが解析の際に説明が簡単なことが多いので，縮約平衡法は，以下のように記述されるものとする．k が定数であり，子問題での計算時間の上界が親問題の計算時間の上界よりも大きくなければ，計算量的には，1 反復の計算時間は変化しない．

ALGORITHM ENUMERATION_INIT (X)
Step 1: X に縮約フェイズを施す
Step 2: ENUMERATION (X) を呼び出す

ALGORITHM ENUMERATION (X)
Step 1: $k :=$ 平衡フェイズが作る子問題の数
Step 2: For i := 1 **to** k
Step 3: 平衡フェイズで子問題 i の入力 X_i を作る
Step 4: X_i に縮約フェイズを施す．
Step 5: ENUMERATION (X_i) を呼び出す
Step 6: End for

この縮約平衡法を使用することにより，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，この中から完全マッチングを列挙するアルゴリズムの高速化を紹介しよう．このアルゴリズムは [3] のアルゴリズムを引用している．完全マッチング列挙アルゴリズムに関しては他にも [2, 1, 13] に研究がある．

与えられた 2 部無向グラフ $G = (V = (V_1 \cup V_2), E)$ に対して，G の完全マッチングを列挙する問題を考える．マッチングとは，互いに端点を共有しないような枝の集合のことである．この問題に対しては，分割法がうまく働く．まず，グラフの完全マッチング M を求める．これは，[6] などのアルゴリズムにより，$O(n^{1/2}m)$ で実現できる．そして，G に M 以外の完全マッチング M' が存在するかどうかを調べる．M' が存在しなかった場合は終了，そうでない場合は，分割枝 $e^* \in M \setminus M'$ を選び，問題を (1) e^* を含む完全マッチングを列挙，(2) e^* を含まない完全マッチングを列挙，する問題に分割する．ここで，$G^+(e^*)$ を G から e^* と，e^* に隣接する枝と，e^* に接続する頂点を除去したグラフとし，$G^-(e^*)$ を G から e^* を除去したグラフとする．e^* を含まない完全マッチングは $G^-(e^*)$ の完全マッチングであるので，e^* を含まない完全マッチングと $G^-(e^*)$ の完全マッチングは 1 対 1 対応する．また，$G^+(e^*)$ の完全マッチングは，e^* を含む G の完全マッチングから e^* を除去したものであるので，e^* を含む完全マッチングと $G^+(e^*)$ の完全マッチングは 1 対 1 対応する．よって，$G^+(e^*)$ と $G^-(e^*)$ を作り，両者について含まれる完全マッチングを列挙することにより，完全マッチング列挙は再帰的に解くことができる．M' は $G^-(e^*)$ の，$M \setminus \{e^*\}$ は $G^+(e^*)$ の完全マッチングになっているので，それぞれの子問題で完全マッチングを求める必要は無いことを注意しておく．

ここで G に含まれる M 以外の完全マッチングを求める方法を述べよう．G の 2 つの完全マッチング M と M' の対称差を $M \Delta M'$ とすると，G の任意の頂点に接続する $M \Delta M'$ の枝は 0 本か 2 本である．よって，$M \Delta M'$ はいくつかの互いに素な閉路の集合となる．$M \Delta M'$ を構成するそれぞれの閉路には，M と M' の枝が交互に現れる．このように，M に含まれる枝と M に含まれない枝が交互に現れるような閉路を「M に関する交互閉路」，または単に「交互閉路」と呼ぶ．上記の議論より，G が M 以外の完全マッチングを含むならば，G は M に関する交互閉路を 1 つは含む．また，M と M に関する交互閉路 C が与えられたとき，C に関して M に含まれる枝と含まれない枝を入れ替えて得られる枝集合，すなわち $M \Delta C$ は G の完全マッチングとなる．よって，M に関する交互閉路を発見することにより，G に M 以外の完全マッチングが含まれるかどうかを判定できる．交互閉路の発見は，G の枝に向き付けをして得られる有向グラフ $D(G, M)$ を使って行う．$D(G, M)$ の枝は，M に含まれれば，V_2 に尾を，V_1 に頭を持ち，そうでなければ，V_2 に頭を，V_1 に尾を持つとする．この定義から，$D(G, M)$ の有向パス，有向閉路には M の枝が交互に現れることがわかる．また，任意の G の交互閉路も $D(G, M)$ では有向閉路となる．よっ

て, $D(G,M)$ の有向閉路を見つければ, G の交互閉路を見つけることができる. $D(G,M)$ の有向閉路は, 深さ優先探索により $O(|V|+|E|)$ 時間で見つけることができるので, この分割法を使ったアルゴリズムの計算時間は 1 つの完全マッチングあたり $O(|V|+|E|)$ 時間となる. 以下に完全マッチングを列挙するアルゴリズムを記述しよう.

ALGORITHM ENUM_PMAT_INIT $(G=(V,E))$
Step 1: G の完全マッチング M を見つける
Step 2: M が存在しなかったら終了
Step 3: ENUM_PMAT (G,M) を呼び出す

ALGORITHM ENUM_PMAT (G,M)
Step 1: $D(G,M)$ の有向閉路 C を見つける
Step 2: C が存在しないなら M を出力して終了
Step 3: $M' := M \Delta M'$
Step 4: $e^* := M \setminus M'$
Step 5: ENUM_PMAT $(G^+(e^*), M \setminus e^*)$ を呼び出す
Step 6: ENUM_PMAT $(G^-(e^*), M')$ を呼び出す

さて, このアルゴリズムに縮約平衡法を適用してみよう. まず, 1 反復の計算時間を子孫数に対して小さくするために, 縮約フェイズを設計する. 問題を小さくするためには, G の中から無駄な部分を削り, 冗長な部分を縮約する.

まず最初に, 無駄な部分を削る方法を述べる. e を $D(G,M)$ の任意の有向閉路に含まれない G の枝とする. すると, e は M と任意の完全マッチング M' の対象差に含まれない. つまり, e は全ての G の完全マッチングに含まるか, または全ての G の完全マッチングに含まれない. よって e を G から除去しても G に含まれる完全マッチングの集合に変化は無い. これは, $D(G,M)$ から e を取り除いても, 任意の閉路は壊されないし, また, 新しい閉路も発生しないことから明らかであろう. よって, これらの無駄な枝を全て除去することにより, 問題の大きさを小さくすることができる. 有向閉路に含まれない枝を発見するには, 強連結成分分解のアルゴリズムが使える. 強連結成分に含まれない枝は, 任意の有向閉路に含まれない. 強連結成分分解は $O(|E|+|V|)$ 時間で実行できるので, この無駄な枝を除去する操作は $O(|E|+|V|)$ 時間で行える. グラフ G に対して, この操作を行ったグラフを $\hat{t}(G)$ と表記する.

次に冗長な部分を縮約する方法を述べる. G に, 隣接する次数 2 の頂点 u,v があるとき, これらの頂点に接続する G の枝は, $(w_1,u),(u,v),(v,w_2)$ の 3 本である. ゆえに, 任意の完全マッチング M に対して, $(w_1,u),(v,w_2) \in M$ か, $(u,v) \in M$ が成り立つ. そこで, G から $(w_1,u),(u,v),(v,w_2)$ を除去し, かわりに枝 (w_1,w_2) を加えたグラフ G' を考える. G の完全マッチング M に対し, $(u,v) \in M$ であれば $M \setminus \{(u,v)\}$ が G' の完全マッチングとなり, そうでない場合は, $M \setminus \{(w_1,u),(v,w_2)\} \cup \{(w_1,w_2)\}$ が完全マッチングとなる. 明らかにこの逆も成り立つので, G' の完全マッチングを行うことにより, G の完全マッチングの列挙を行うことができる. 隣接する次数 2 の頂点全てに対してこの操作を行うことにより, 問題を小さくできる. ただし, $w_1=v, w_2=u$ が成り立つときは, 操作後に $w_1=w_2$ となってしまうので, この操作を行わない. この操作の計算時間は, 次数 2 の頂点全てを探し出し, 隣接するものを選び, それら各々について定数個の枝の入れ替えを行うだけなので, $O(|V|)$ である. 以上, これら 2 つの操作が, 縮約フェイズの操作である. グラフ G に縮約フェイズの操作をほどこして得られるグラフを $t(G)$ と表記する.

さてここで, $t(G)$ が含む完全マッチングの数の下界を求めよう. そのために, $D(G,M)$ が含む有向閉路の数の下界を算定する. $C=(V_C,E_C)$ を, $D(G,M)$ の有向閉路 1 つからなる有向グラフとする. 今, C の有向閉路の数は, $|E_C|-|V_C|+1$ である. $D(G,M)$ は強連結であったので, $D(G,M)$ から C の枝を除いたグラフは, 枝集合が空でなければ, 有向閉路か, 両端のみが C の頂点である有向パスを含む. そこで, その中の 1 つを C に加える. すると, C には少なくとも 1 つ, 新しい有向閉路が発生する. この操作により C に加えられた枝数を a, 加えられた頂点数を b とすると, $a-b=1$ である. よって, C の有向閉路の数は, 少なくとも $|E_C|-|V_C|+1$ である. $E=E_C$ となるまでこの操作を繰り返すことにより, $D(G,M)$ の有向閉路数は少なくとも $|E|-|V|+1$ となることがわかる. $D(G,M)$ は連続する次数 2 の頂点を含むとすればそれは, 長さ 2 の有向閉路である. そこで, $D(G,M)$ から, これらの有向閉路を除去すると, 枝数は頂点数の 1.25 倍になる. 長さ 2 の有向閉路については, 枝 2 つにつき 1 つ有向閉路が存在するので, G に含

まれる完全マッチングの数は $0.2|E| = \Omega(|E|)$ よりも大きくなる. よって, グラフ G_x を入力するような任意の列挙木の頂点 x について $\bar{D}(x) = |E|/5$ とすることができる.

さて次に平衡フェイズの解説を行おう. 平衡フェイズの目標は, 子問題の大きさが小さくならないようにすることである. そこで, 子問題が小さくなってしまったときに, 問題の分割のしかたを変更して, ある程度大きな子問題に作りかえる方法を述べる. 長さ2の有向閉路からなる強連結成分が $D(G,M)$ に含まれる場合, その中の枝を分割枝 e^* として選ぶことにより, 両子問題の大きさを $|E|-2$ とする事ができる. この場合, 子問題の大きさは十分大きいので, 以下では $D(G,M)$ が長さ2の有向閉路からなる強連結成分を含まない場合の分割方法を述べる. G の2つの完全マッチング M と M' があり, $e^* \in M \setminus M'$ であるとする. 今, $D(G,M)$ は2つ以上の有向閉路を含むとする.

今, もし $D(G^+(e^*), M \setminus \{e^*\})$ 枝のうち, 9/10 以上が有向閉路に含まれない場合, $G^+(e^*)$ を入力する子問題のサイズは小さくなる. これらの枝は, $D(G,M)$ の有向閉路には含まれていたので, これらの枝を含む有向閉路は必ず e^* を含む. ここで, $\hat{D}$ を $D(G \setminus \{e^*\}, M)$ の強連結成分を縮約して得られる有向グラフとし, s を e^* の頭, t を e^* の尾とする. 仮定より, $\hat{D}$ は 少なくとも $9|E|/10$ 本の枝を含む. $\hat{D}$ に e^* を加えると全ての枝は有向閉路に含まれるようになるので, $\hat{D}$ の全ての枝は, いずれかの s-t パスに含まれることを注意しておく. T' を $\hat{D}$ の全域有向根付き木とし, $\hat{D}$ の頂点 v に対し, $m'(v)$ を, T' 上の v-t パス上の出次数が1でない頂点に尾を持つ枝の数とする. 出次数1の頂点に尾を持つ枝の総数は高々 $|V|$ 本, また, $D(G,M)$ は長さ2の有向閉路からなる強連結成分を含まないので, $|E| \geq 1.25|V|$ が成り立ち, よって, 出次数が1でない頂点に尾を持つ枝は $9|E|/10 - 8|E|/10 = |E|/10$ 本以上存在する. ここで P を s から順に $m'(v)$ が最大の頂点をたどって得られる有向 s-t パスとする. 言い換えれば, 任意の頂点 $v \in P$ に対し, 2本の枝 $(v,u_1),(v,u_2) \in L^+(v)$ に対して, $(v,u_1) \in P$ ならば $m'(u_1) \geq m'(u_2)$ が成り立つような有向 s-t パスである. ただしここで, $L^+(v)$ は v に尾を持つ枝の集合とする. ここで, T を $T' \cup P$ から P の枝と頭を共有する T' の枝を除去して得られる全域有向根付き木とする. T に対して, $m(v)$ を $m'(v)$ と同じように定める. P と T の作り方から, P の任意の頂点 v に対して $m(v) \geq m'(v)$ が成り立ち, P に含まれない任意の頂点 v に対して $m(v) \leq m'(v)$ が成り立つ. よって, P の任意の頂点 v に対し, 2本の枝 $(v,u_1),(v,u_2) \in L^+(v)$ は, $(v,u_1) \in P$ ならば $m(u_1) \geq m(u_2)$ を満たす.

ここで, $\hat{v}$ を, $m(\hat{v}) < 2|E|/30$ か, 入り次数が1でない P の頂点で, P 上での s への距離が最も近いものとする. また, $(\hat{u},\hat{v})$ を P の枝で $\hat{v}$ に頭を持つものとする. つまり, $\hat{u}$ は P 上で $\hat{v}$ の s 側に隣接する頂点である. ここで, P' を, P の部分 s-$\hat{u}$ パスとする. $m(\hat{u}) \geq 2|E|/30$ であるので, $m(\hat{v}) < 2|E|/30$ であれば $\hat{u}$ の出次数は1でないことを注意しておく. $\hat{u}$ の出次数が1でないか, $\hat{v}$ の入り次数が1でないので, M に含まれる $\hat{u}$ に接続する枝は P' 上で $\hat{u}$ に頭を持つ枝である. よって, $L^+(\hat{u})$ は M の枝を含まない. ここで, 以下の補題が成り立つ.

補題 7 ある有向パス $R = (v_1, ..., v_k)$ の任意の頂点 v_i の入り次数が1のとき, R の枝のうち少なくとも1つを含まないような有向パスの集合 $\mathcal{P}_1$ は, (v_{k-1}, v_k) を含まないパスの集合 $\mathcal{P}_2$ と一致する.

Proof : $\mathcal{P}_2 \subseteq \mathcal{P}_1$ は自明であるので, $\mathcal{P}_1 \subseteq \mathcal{P}_2$ を証明する. もし, $^\exists Q \in \mathcal{P}_1 \setminus \mathcal{P}_2$ であるとする. $e = \text{aug } \max_{(v_j,v_{j+1})}\{(v_j,v_{j+1}) \in R \setminus Q\}$ とすると, e は Q の枝と頭を共有している. これは, 仮定に反するので, $^\nexists Q \in \mathcal{P}_1 \setminus \mathcal{P}_2$ である. ∎

今, $m(\hat{v}) < 2|E|/30$ が成り立っているとしよう. このとき $L^+(\hat{u})$ の頭 v の $m(v)+1$ の総和 $\sum_{(\hat{u},v) \in L^+(\hat{u})} m(v)+1$ は $2|E|/30$ 以上になる. よって $L^+(\hat{u})$ の部分集合 F でその頭の集合 $h(F)$ が, $|E|/30 \leq \sum_{v \in h(F)} m(v) \leq 2m/30$ を満たすものが必ず存在する. $m(\hat{v}) < 2|E|/30$ でないときは, このような F が存在するとは限らない. そこで, $F = L^+(\hat{u})$ とする. さてここで, G の完全マッチングの集合を以下の2つに分割する.

(a) $M \Delta \hat{M}$ が P' と枝 $e \in F$ を含むような完全マッチング $\hat{M}$ の集合
(b) それ以外の完全マッチングの集合

F の枝を含む M に関する交互閉路は，必ず e^* と P' の枝を含む．よって，(a) は M との対称差が F の枝を含む完全マッチングの集合となり (b) は M との対称差が F の枝を含まない完全マッチングの集合となる．よって，補題 7 より，(a) は G から $\hat{u}$ に接続する F に含まれない枝を除去したグラフ G_1 の完全マッチングの集合となり，(b) は G から F の枝を除去したグラフ G_2 の完全マッチングの集合となる．前述の通り，$F \subseteq L^+(\hat{u})$ は M の枝を含まない．また，G_1 の完全マッチング M' は F の枝を含む．よって，M' と M の対象差は F の枝を含むことを注意しておく．よって，これら 2 つのグラフを作ることにより，(a) と (b) は再帰的に列挙することができる．

$D(G,M)$ の任意の枝は有向閉路に含まれるので，尾を共有する $\hat{D}$ の枝 f_1 と f_2 に対して，e^* を含み，各 f_i を含むような $D(G,M)$ の有向閉路 C_1 と C_2 が存在する．もし，C_1 と C_2 が F の枝を含むとき，$M\Delta C_1$ と $M\Delta C_2$ は (a) に含まれる．$\hat{D}$ の任意の有向 v-t パスは P' の枝を含まないので，このような有向閉路の組は必ず存在する．よって，G_1 には，$i=1,2$ に対して，f_i を含む完全マッチングと f_i を含まない完全マッチングが存在する．ゆえに，各 f_i は $\hat{t}(G_1)$ に含まれる．よって，P に含まれる頂点 $v \neq \hat{u}$ に対して，v に尾を持つ枝が 2 つ以上存在したら，それらは $\hat{t}(G_1)$ に含まれる．また，$\hat{u}$ に対して，$\hat{u}$ に尾を持つ F に含まれる枝が 2 つ以上存在したら，それらは $\hat{t}(G_1)$ に含まれる．同様にして，F の枝を含まない $D(G,M)$ の有向閉路に含まれる頂点 v に対して，v に尾を持つ異なる枝 $f_1, f_2 \notin F$ は $\hat{t}(G_2)$ に含まれる．以上より，$\hat{t}(G_1)$ には，少なくとも，$|F|=1$ のとき $\sum_{(\hat{u},v)\in F} m(v)$, $|F|>1$ のとき $|F|+\sum_{(\hat{u},v)\in F} m(v)$ 本の枝が含まれる．両者をあわせると，$\hat{t}(G_1)$ の枝数は少なくとも $|F|-1+\sum_{(\hat{u},v)\in F} m(v)$, 多くとも $|F|+\sum_{(\hat{u},v)\in F} m(v)$ となる．同様に，$\hat{t}(G_2)$ の枝数は少なくとも $m(s)-|F|-\sum_{(\hat{u},v)\in F} m(v)-1$ となる．$m(s)$ は $\hat{D}$ の次数が 2 以上の頂点に尾を持つ枝の総数と一致する．F の選び方から，$\hat{t}(G_1)$ の枝数は少なくとも $|E|/30-1$ である．$m(\hat{v}) < 2|E|/30$ であるとすると，$m(\hat{u}) \leq |L^+(\hat{u})| + \sum_{(\hat{u},v)\in L^+(\hat{u})} m(v)$ であり，$\sum_{(\hat{u},v)\in F} m(v) \leq 2|E|/30$ であったことから，$\hat{t}(G_2)$ に含まれる枝の数は，少なくとも

$$m(s) - m(\hat{u}) + |L^+(\hat{u}) \setminus F| - 1 + \sum_{(\hat{u},v)\in L^+(\hat{u})\setminus F} m(v)$$

$$\geq |E|/10 - |F| - \sum_{(\hat{u},v)\in F} m(v) - 1$$

となる．F の選び方より，$|F| + \sum_{(\hat{u},v)\in F} m(v) \leq 2|E|/30$ であることから，これは $|E|/30-2$ より大きい．また，$m(\hat{v}) \geq 2|E|/30$ であるときは，$\hat{v}$ の入り次数は 1 でなく，$\hat{D}$ の有向閉路で $\hat{v}$ を含み，$\hat{u}$ を含まないものが存在する．よって，$\hat{t}(G_2)$ には $m(\hat{v}) \geq 2|E|/30$ 本以上の枝が含まれる．

以上より，$\hat{t}(G^+(e^*))$ に $|E|/10$ 以下しか枝が含まれない場合でも，G の完全マッチングの列挙問題を，$|E|>90$ であれば，$\hat{t}(G_1), \hat{t}(G_2) \geq |E|/90$ となるグラフ G_1 と G_2 の完全マッチングの列挙問題に分割できることがわかった．$\hat{t}(G^-(e^*))$ が $|E|/10$ 以下しか枝を含まないときも，$|E|>90$ であれば，同様にして，問題を，$|E|/90$ 以上の枝を含むグラフの完全マッチング列挙問題 2 つに分割できる．この分割にかかる計算時間は $O(|E|+|V|)$ である．

さてここで，この縮約フェイズと平衡フェイズを加えたアルゴリズムの生成する列挙木に対し，補題 6 を使って X^* の値を算定しよう．グラフ $G_x=(V_x,E_x)$ を入力とする列挙木の各頂点 x のでの計算時間は $O(|E_x|+|V_x)$ である．枝の接続しない頂点はグラフから取り除けるので，これは $O(|E_x|)$ であるとしてよい．よって，ある定数 $\bar{C}$ を用いて，$T(x)=\bar{C}|E|$ とできる．$\bar{D}(x)=|E_x|/5$ であったことを考慮すると，$\hat{T}=O(1)$ となる．

$T(x)$ は x の任意の子供 y について，$T(x) \geq T(y)$ であり，任意の葉 x で $T(x)=f=O(1)$ である．また，ある定数 C に対して，$T(x)/f \leq 4C^2$ のとき，w の子孫数は定数であり，$T(x)/f \geq 4C^2$ のときは x の 2 つの子供 x_1 と x_2 は $T(x_i) \geq (1/C)T(x)-Cf$ を $i=1,2$ について満たす．よって，補題 6 より，$X^* = O(\log_{C/(C-1)} T(x_0)/f)$ である．以上の結果より，$\hat{T}=O(1)$, $X^*=O(\log|V|)$ であるので，下の定理を得る．

定理 2 2 部グラフの完全マッチングは 1 つあたり $O(\log|V|)$ 時間で列挙できる．∎

ただし，この算定はは出力にかかる時間を考慮していない．完全マッチングの出力には，通常 $O(|V|)$

の時間を必要とするが, コンパクト出力のテクニックを使うことにより, 出力を圧縮し, 出力の総量をアルゴリズムの計算時間と同じだけにすることができることを注意しておく. 詳細については, ここでは割愛させていただく.

謝辞

本研究を進めるにあたってさまざまな助言を頂きました, 茗荷谷クラブの方々に厚く感謝いたします.

参考文献

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

[2] K. Fukuda and T. Matsui, "Finding All the Minimum Cost Perfect Matchings in Bipartite Graphs," Networks **22**, pp.461-468 (1992).

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

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

[5] 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).

[6] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs," SIAM J. Comp. **2**, pp.225-231 (1973).

[7] 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).

[8] R. C. Read and R. E. Tarjan, "Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees," Networks **5**, 237-252 (1975).

[9] R. Seidel, "Small-Dimensional Linear Programming and Convex Hulls Made Easy," Discrete and Computational Geometry **6**, 423-434 (1991).

[10] 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).

[11] S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa, "A New Algorithm for Generating All the Maximum Independent Sets," SIAM J. Comp. **6**, pp.505-517 (1977).

[12] 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).

[13] 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).

[14] T. Uno, "Studies on Speeding Up Enumeration Algorithms," Doctral Thesis, Dept. Systems Science, Tokyo Institute of Technology (1998)

[15] T. Uno, "A New Approach for Speeding Up Enumeration Algorithms," Lecture Note in Computer Science **1533**, Springer-Verlag, Algorithms and Computation, 287-296 (1998).

[16] 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)