極小集合被覆を列挙する実用的高速アルゴリズム

宇野 毅明
国立情報学研究所 〒101-8430 東京都千代田区一ツ橋 2-1-2 e-mail: uno@nii.jp

抄録: 台集合 E 上の集合族 $\mathcal{F}$ に対し, $\mathcal{F}$ の部分集合で E のすべての要素を覆うものを集合被覆という. 集合の意味で極小な集合被覆を列挙する問題は, ハイパーグラフの極小横断, 極小ヒッティングセットの列挙など多数の問題と等価であり, 多くの研究が行われている. しかし, この問題が出力数多項式時間で解けるかどうかは未解決である. 本稿では, この問題に対する, 実用上高速なアルゴリズムを提案し, 計算実験によって乱数で発生させた問題に対する平均的な計算時間が出力 1 つあたり $O(|E|)$ 程度であることを確認した.

A Practical Fast Algorithm for Enumerating Minimal SetCoverings

Takeaki Uno
National Institute or Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, JAPAN
e-mail: uno@nii.jp

Abstract: For a set family $\mathcal{F}$ defined on a grand set E, a subset of $\mathcal{F}$ covering all the elements of E is called a set covering. The enumeration problem of minimal set covering is equal to the enumeration problem of hypergraph dualization, minimal hitting sets, and other other many problems, and have been studied intensively. However, the existence of an output polynomial algorithm for this problem is still open. However, there proposed an algorithm whose average computation time on computational experiments for random generated instances is output polynomial. In this paper, we propose a practical fast algorithm obtained by improving this algorithm, and by computational experiments, show that the average computation time for randomly generated instances is $O(|E|)$ per output, by computational experiments.

1 導入

$\mathcal{F} = \{F_1, ..., F_n\}$ を台集合 $E = \{e_1, ..., e_n\}$ 上の集合族とする. このとき, $\mathcal{C} \subseteq \mathcal{F}$ が, 任意の $e \in E$ に対して, ある $S \in \mathcal{C}$ が存在して $e \in S$ となるという条件を満たすとき, $\mathcal{C}$ は集合被覆であるという. 極小集合被覆とは, 他の集合被覆を真部分集合として含まない集合被覆のことをいう. 極小集合被覆列挙問題とは, $E, \mathcal{F} \subseteq 2^E$ が与えられたとき, $\mathcal{F}$ の極小集合被覆を全てちょうど 1 度ずつ出力せよ, という問題である.

この問題は, ハイパーグラフの極小横断, 極小ヒッティングセットなどの他のいろいろな問題と等価であることも知られており, 各方面からの盛んな研究が行われている. また, この問題は, #P-完全であることが知られている [4]. また, この問題に対する出力多項式時間のアルゴリズムがあるかどうかはまだ未解決である. 現在, 計算量の意味での最速なアルゴリズムは, 出力数を N とすれば, およそ $O(N^{\log N})$ 時間で終了する [3]. また平均的に高速なアルゴリズムを開発し, 計算機実験によりその速度を検証する, という研究も行われている [3].

本稿では [3] のアルゴリズムを改良し, より実際に速いアルゴリズムを提案し, 計算機実験によりその効果を検証する.

列挙アルゴリズムの構築方法の１つに, 1996 年に提案された逆探索と呼ばれるものがある [1]. 本稿では [3] のアルゴリズムの基本設計を逆探索で捉え, 2 節で, 逆探索によるアルゴリズムを解説する. 3 節でアルゴリズムの改良を示し, 4 節で計算実験の結果を示す.

2　逆探索を用いた基本アルゴリズム

この節では, 極小集合被覆列挙問題に対する, 逆探索アルゴリズムを解説する. これは, [3] のアルゴリズムと等価であり, 彼らのアルゴリズムを逆探索で解釈したものである.

まず, 逆探索の一般的な解説を行おう. 今, 列挙したい解の集合 $\mathcal{Y}$ が与えられたとする. このとき, ある $B_0 \in \mathcal{Y}$ 以外の全ての $\mathcal{Y}$ の解 B に対し, B の親 $p(B)$ を何らかのルールによって定義する. ただしこのとき,

- B 自身が B の真の祖先にならない, すなわち $B \neq p(p(...p(B)...))$

という性質を満たすようにする. ここで, この親子関係から得られるグラフ, つまり解の集合を頂点集合とし, 親子関係にある頂点間に枝を張って得られるグラフを考える. 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 B_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という解法である.

一般に列挙木のサイズは巨大であるので, 探索を行う際に列挙木をメモリに格納することはほぼ不可能である. その代わりに, 逆探索は, 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使用する. 列挙木を深さ優先探索する際に, 探索の各反復においてこのアルゴリズムを呼び出し, 見つけた子供それぞれについて再帰的に探索を行う. 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. 逆探索は１つの反復で１つ要素を出力するので, 逆探索の計算時間は, １つの出力あたり, １反復の計算量となる. 逆探索の詳細については [1] を参照されたい. [2] に同じ著者の日本語の解説もある.

次に, 極小集合被覆列挙問題に対する逆探索法の解説を行う. $\mathcal{X}$ を, $E, \mathcal{F} \subseteq 2^E$ の極小集合被覆の集合とする. $i = 1, ..., n$ に対して, $E_i = \{e_1, ..., e_i\}$ とする. 集合 $\mathcal{C} \subseteq \mathcal{F}$ が, i 集合被覆であるとは, 任意の $e \in E_i$ に対して, ある $F \in \mathcal{C}$ が存在して, $e \in F$ となることとする. 特に $\mathcal{C}$ が他の i 集合被覆に含まれないときに, $\mathcal{C}$ を i 極小集合被覆と呼ぶ. すべての i 極小集合被覆の集合を $\mathcal{X}_i$ とする. １つの集合 $\mathcal{C} \subseteq \mathcal{F}$ が二つ以上の i に対して $\mathcal{C} \in \mathcal{X}_i$ となる可能性がある. これらを区別するため, $\mathcal{X}_i$ の任意の構成要素を $(i, \mathcal{C})$ と表記する.

ここで解説する逆探索は, すべての i 極小集合被覆, つまり $\bigcup_{i=1}^{n} \mathcal{X}_i$ の全ての構成要素を列挙するアルゴリズムである. 実際に $\mathcal{X}$ の極小集合被覆のみを列挙したい場合には, 出力するものの中で, n 極小集合被覆, つまり $\mathcal{X}_n$ の極小集合被覆であるもののみ出力する.

集合被覆 $\mathcal{C}$ とその構成要素 $F \in \mathcal{C}$ に対して, $e \in \mathcal{F}$ が F のクリティカル要素であるとは, 任意の $F' \neq F, F' \in \mathcal{C}$ に対して $e \notin F'$ であることとする. また, F のクリティカル要素の集合を $Cr(F, \mathcal{C})$ と表記する. i 集合被覆 $\mathcal{C}$ が i 極小集合被覆であるための必要十分条件は, 任意の $F \in \mathcal{C}$ に対して, $Cr(F, \mathcal{C}) \cap E_i \neq \emptyset$ となることである. また, 集合 $\mathcal{C} \subseteq \mathcal{F}$ と任意の $e \in E$ に対して, $Cov(e, \mathcal{C})$ を $e \in Cr(F, \mathcal{C})$ となる $F \in \mathcal{C}$ で定義する. ただし, そのような F が存在しない場合には $Cov(e, \mathcal{C}) = nil$ とする. ２つの集合 $F \neq F', F, F' \in \mathcal{C}$ に対して, 同時に $e \in Cr(F, \mathcal{C})$ と $e \in Cr(F', \mathcal{C})$ が満たされることはないので, $Cov(e, \mathcal{C})$ は唯一的に定義されることを注意しておく.

さてここで, 逆探索が用いる親子関係の定義を行おう. $i > 1$ であるような i 極小集合被覆 $(i, \mathcal{C}) \in \mathcal{X}_i$ に対して, $(i, \mathcal{C})$ の親 $p((i, \mathcal{C}))$ を

・ $\mathcal{C}$ が $i-1$ 極小集合被覆であるならば,

$$p((i,\mathcal{C})) = \begin{cases} (i-1,\mathcal{C}) & \mathcal{C} \text{ が } i-1 \text{ 極小集合被覆であるとき} \\ (i-1,\mathcal{C} \setminus \{Cov(e,\mathcal{C})\}) & \mathcal{C} \text{ が } i-1 \text{ 極小集合被覆でないとき} \end{cases}$$

と定める. この親子関係は以下の補題を満たす.

補題 1 任意の $(i, \mathcal{C}) \in \mathcal{X}_i$ に対して, $p(\mathcal{C}) \in \mathcal{X}_{i-1}$

Proof : $\mathcal{C}$ が $i-1$ 極小集合被覆である場合は, $p((i, \mathcal{C}))$ の定義より成り立つ. $\mathcal{C}$ が $i-1$ 極小集合被覆でない場合は, ある $F \in \mathcal{C}$ が存在して,

$$Cr(F, \mathcal{C}) \cap E_{i-1} = \emptyset$$

となる. $\mathcal{C}$ は i 極小集合被覆であったので,

$$Cr(F, \mathcal{C}) \cap E_i \neq \emptyset$$

であり, よって

$$F = Cov(e_i, \mathcal{C})$$

であり, $\mathcal{C} \setminus \{Cov(e_i, \mathcal{C})\} = \mathcal{C} \setminus \{F\}$ は $i-1$ 集合被覆である. また F 以外の $F' \in \mathcal{C}$ に対しては,

$$Cr(F', \mathcal{C}) \cap E_i \neq \emptyset, e_i \notin Cr(F', \mathcal{C})$$

であるので,

$$Cr(F', \mathcal{C}) \cap E_{i-1} = \emptyset$$

となり, よって $\mathcal{C} \setminus \{Cov(e_i, \mathcal{C})\}$ は $i-1$ 極小集合被覆である. ∎

この補題より, 任意の i 極小集合被覆の親は, $i-1$ 極小集合被覆となることがわかった. よって, この親子関係は $\bigcup_{i=1}^{n} \mathcal{X}_i$ の構成要素間に定義され, かつ巡回を含まないので, 逆探索で使用される親子関係の条件を満たしている. これにより, 任意の 1 極小集合被覆に対して, その子孫を逆探索により列挙すると, すべての i 極小集合被覆を列挙できる.

さて次に, 与えられた i 極小集合被覆 $(i, \mathcal{C})$ の子どもを求める方法を説明しよう. 定義より, $(i+1, \mathcal{C}') \in \mathcal{X}_{i+1}$ が $(i, \mathcal{C})$ の子どもであれば, $\mathcal{C} = \mathcal{C}'$ かある $F \in \mathcal{F} \setminus \mathcal{C}$ に対して, $\mathcal{C} \cup \{F\} = \mathcal{C}'$ が成り立つ.

$\mathcal{C}$ が $i+1$ 極小集合被覆である場合は, 任意の $F \in \mathcal{F} \setminus \mathcal{C}$ に対して, $\mathcal{C} \subseteq \mathcal{C} \cup \{F\}$ であることから, $\mathcal{C} \cup \{F\}$ は $i+1$ 極小集合被覆にならない. $(i+1, \mathcal{C})$ の親は, 定義より,

$$p((i+1, \mathcal{C})) = (i, \mathcal{C})$$

となり, よって, $(i+1, \mathcal{C})$ が $(i, \mathcal{C})$ の唯一の子どもである.

$\mathcal{C}$ が $i+1$ 極小集合被覆でない場合は, $(i+1, \mathcal{C})$ は $(i, \mathcal{C})$ の子どもにはならない. また, 任意の $F \in \mathcal{F} \setminus \mathcal{C}$ に対して, $\mathcal{C} \cup \{F\}$ が $i+1$ 極小集合被覆であれば, 定義より,

$$p((i+1, \mathcal{C} \cup \{F\})) = (i, \mathcal{C})$$

であり, $(i+1, \mathcal{C} \cup \{F\})$ が $(i, \mathcal{C})$ の子どもとなる.

以上より, i 極小集合被覆の子どもをすべて見つける方法がわかった. この方法を用いた基本アルゴリズムを以下に記述しよう.

Algorithm 極小集合被覆列挙_INIT$(E, \mathcal{F})$
(EMSC1) **For each** $F \in \mathcal{F}, e_1 \in F$ **do**
(EMSC2) **Call** 極小集合被覆列挙_ITER$(E, \mathcal{F}, 1, \{F\})$
(EMSC3) **End for**

Algorithm 極小集合被覆列挙_ITER$(E, \mathcal{F}, i, \mathcal{C})$
(EMSC4) **If** $i = |E| + 1$ **then output** $\mathcal{C}$
(EMSC5) **Else if** $F \in \mathcal{C}, e_i \in F$ なる F が存在 **then**
　　call 極小集合被覆列挙_ITER$(E, \mathcal{F}, i + 1, \mathcal{C})$
(EMSC6) **Else for each** $F, F \in \mathcal{F} \setminus \mathcal{C}, e_i \in F$ **do**
(EMSC7) **If** $\mathcal{C} \cup \{F\}$ が $i + 1$ 極小被覆集合 **then**
　　call 極小集合被覆列挙_ITER$(E, \mathcal{F}, i + 1, \mathcal{C} \cup \{F\})$
(EMSC8) **End for**

このアルゴリズムの総反復数はすべての i 極小集合被覆数, つまり $|\bigcup_{i=1}^{n} \mathcal{X}_i|$ であるが, これは出力数 $|\mathcal{X}_n|$ に対してどの程度大きくなるか, 効果的な算定はできない. ゆえに, このアルゴリズムは出力多項式時間アルゴリズムではない.

E の要素に対する添え字付けの順序を工夫すれば, 効果的な算定が出来る可能性もあるが, 現在の所, そのようなことができるかどうかは未解決である. また, E の要素の任意の添え字付け順序に対して $|\bigcup_{i=1}^{n} \mathcal{X}_i|$ が $|\mathcal{X}_n|$ の多項式で押さえられないような問題例が存在するかどうか, という問題も未解決である. しかし, 平均的には $|\mathcal{X}_i| \leq |\mathcal{X}_{i+1}|$ という関係が成り立つであろう, と推測されるので, アルゴリズムの総反復数は出力数 $|\mathcal{X}_n|$ に対してそれほどは大きくならないであろうと考えられる.

3　基本アルゴリズムの改良

前節で示したアルゴリズムは, 単純に実装すると, 問題の大きさの増加に対して出力される解ひとつあたりの計算時間の増加が大きい. そこで本稿では, 1 反復の計算時間の減少させる改良と, 反復数を減少させる改良を提案する.

前節で示した基本アルゴリズムは, (EMSC7) で $\mathcal{C} \cup \{F\}$ が $i + 1$ 極小被覆集合であるかどうかを調べる. これを単純な方法で行うと, 各 $F' \in \mathcal{C}$ について, $(\mathcal{C} \cup \{F\}) \setminus \{F'\}$ が集合被覆であるかを $\mathrm{O}(|E| \times |\mathcal{C}|)$ 時間で調べることになり, 合計で $\mathrm{O}(|\mathcal{C}|^2|E|)$ 時間を要する. (EMSC6) のループが最大で $|\mathcal{F}|$ 回程度なので, 1 反復での最悪計算時間は $\mathrm{O}(|\mathcal{C}|^2|E| \times |\mathcal{F}|)$ 時間となる.

そこで本稿では, (EMSC7) での $\mathcal{C} \cup \{F\}$ が $i+1$ 極小被覆集合であるかどうかの判定方法に対して 2 つの改良を行う. まず第 1 の改良を説明する.

アルゴリズムが問題を入力する際に, 各 $e \in E$ について集合 $Set(e)$ を, e を含む $\mathcal{F}$ の構成要素の集合とする. また, 各 $F' \in \mathcal{C}$ に対して変数 $Cr(F')$ を, 各 $e \in E$ について変数 $Cov(e)$ を用意し, アルゴリズム実行中, 常に

$$\begin{aligned} Cr(F') &= Cr(F', \mathcal{C}) \cap E_i \\ Cov(e_j) &= Cov(e_j, \mathcal{C}) \end{aligned}$$

が満たされるようにする.

今, $\mathcal{C}$ は i 極小被覆集合であるが, $i+1$ 極小集合被覆ではないので, 各 $F \in Set(e) \setminus \mathcal{C}$ に対して $\mathcal{C} \cup \{F\}$ が $i+1$ 極小被覆集合であるための必要十分条件は,

・ 任意の $F' \in \mathcal{C}$ に対して, $Cr(F', \mathcal{C}) \cap E_i \setminus \{F\} \neq \emptyset$

である. そこで,

・ すべての $e \in F$ に対して, $Cr(Cov(e)) \neq nil$ であれば, $Cr(Cov(e))$ から e を取り除く

という作業を行い, 空集合となる $Cr(F')$ が存在するならば, またそのときに限り, $\mathcal{C} \cup \{F\}$ は $i+1$ 極小集合被覆となる. この判定方法が要する計算時間は $\mathrm{O}(|F|)$ である.

こう判定方法を導入するためには, 再帰呼び出しによる i と $\mathcal{C}$ の変化に伴う Cov と Cr の更新を行う操作が必要である. 以下でその方法とその計算時間について述べる. 再帰呼び出し終了後の変更は, 再帰呼び出しを行うさいの変更の逆の操作を行えばよいので, 同じ計算量で行えることを注意しておく.

再帰呼び出しが行われるとき, i は必ず 1 つ増える. $\mathcal{C}$ は, 変化がない場合と, ある 1 つの集合 F が追加される場合がある. まず, $\mathcal{C}$ に変化がない場合の説明を行う. この場合, Cov に関する変更は必要ない. 各 $Cr(F')$ に関しては,

$$Cr(F', \mathcal{C}) \cap E_{i+1} \neq Cr(F', \mathcal{C}) \cap E_{i+1}$$

ならば, 変更する必要がある. この条件は $e_{i+1} \in Cr(F')$ となるものについてのみ成り立つ. よって, $Cov(e_{i+1}) \neq nil$ である場合のみ, $Cr(Cov(e_{i+1}))$ に e_{i+1} を追加すればよい. この計算は $\mathrm{O}(1)$ 時間で終わる.

次に $\mathcal{C}$ に F が追加された場合の説明を行う. まず Cov であるが, これは, $Cov(e) \neq nil$ かつ $e \in F$ であるものに対して, $Cov(e) = nil$ となる. この計算は $\mathrm{O}(|F|)$ 時間で終わる. 次に 各 $Cr(F')$ であるが, これは $Cr(F', \mathcal{C}) \setminus F$ に変更される. これは, 上記の判定方法の操作と同じであるので, 計算時間は $\mathrm{O}(|F|)$ である. それと, $Cr(F)$ を新たに設定する必要があるが, $\mathcal{C}$ が i 集合被覆であることより, 必ず

$$Cr(F) = \{e_{i+1}\}$$

となる. この操作も $\mathrm{O}(1)$ 時間で終わる.

結局, 1 反復では $\mathrm{O}(|F|)$ 時間の計算を 1 回, $\mathrm{O}(|F|)$ 時間の計算を最大で $|\mathcal{F}|$ 回行うので, 第 1 の改良を行った結果, 1 反復の計算量は

$$\mathrm{O}(|F||\mathcal{F}|) = \mathrm{O}(|E||\mathcal{F}|)$$

となる.

第 2 の改良は, 各 $F \in \mathcal{F} \setminus \mathcal{C}$ に対して変数 $Cr'(F)$ を用意し, アルゴリズムの実行中, 常に

$$Cr'(F) = F \cap (\cup_{F' \in \mathcal{C}} Cr(F', \mathcal{C}))$$

とすることである. 改良 1 を施したアルゴリズムでは, (EMSC7) の判定で $\cup_{F' \in \mathcal{C}} Cr(F', \mathcal{C})$ に含まれる要素 e に関してのみ操作を行えば十分であるので, 判定にかかる計算時間を $\mathrm{O}(|F \cap (\cup_{F' \in \mathcal{C}} Cr(F', \mathcal{C}))|)$ に減少できる. 再帰呼び出しによる $\cup_{F' \in \mathcal{C}} Cr(F', \mathcal{C})$ の変化に伴い, $Cr'(F)$ の変更を行う必要があるが, これは 1 つの $F \in \mathcal{F}$ につき $\mathrm{O}(|F \cap (\cup_{F' \in \mathcal{C}} Cr(F', \mathcal{C}))|)$ 時間で行えるので, 1 反復の計算時間は $\mathrm{O}(\sum_{F \in \mathcal{F} \setminus \mathcal{C}} |F \cap (\cup_{F' \in \mathcal{C}} Cr(F', \mathcal{C}))|)$ となる.

次に, 反復回数を減らすための改良方法を説明する. このアルゴリズムが生成する反復の数は, 入力した台集合 E の要素の添え字付けの順序が変化により増減すると思われる. そこで, 各要素 $e \in E$ を含む $\mathcal{F}$ の集合の数に注目し, その大きさ順で添え字を付ける. 順序が小さい順であるならば, 反復数が減るだろうと推測される. また, 大きい順であれば, 各反復の計算時間, 特に再帰の深いレベルにある反復の計算時間が減ると期待される.

以上の改良の効果を確かめるため, 計算実験を行った. その結果は次節で解説する.

4 計算機実験

この節では計算機実験の結果を解説する. 実験を行ったコンピューターは, Pentium III 500MH を搭載したパーソナルコンピューター, OS は linux, 実装を行った言語は C である. 問題は乱数を用いて, 各要素が $\mathcal{F}$ の各集合に含まれる確率が 3/10 として発生させた.

問題は, 台集合の大きさが 50, 500, 集合族 $\mathcal{F}$ の大きさが 30, 50, 70, 100, 150, 200, 300, 500 であるものをそれぞれ 10 題ずつ, 集合族 $\mathcal{F}$ の大きさが 50, 500, 台集合の大きさが 30, 50, 70, 100, 150, 200, 300, 500 であるものをそれぞれ 10 題ずつ発生させた. それぞれの問題に対してプログラムを実行し, 解 100 万個あたりの実行時間を計測した. 出力数が 1,000,000 以上になった場合は, 1,000,000 で実行を打ち切った. 以下で, その計算結果を表にまとめる. それぞれの問題の大きさに対する, 最大・平均・最小計算時間を示した. 単位は秒である.

また, E の要素の添え字を, その要素が含まれる $\mathcal{F}$ の集合の数でソートした問題についても実験を行った. 含まれる集合の数が少ない順でソートしたものと, 多い順でソートしたものを, 台集合の大きさが 50, 集合族 $\mathcal{F}$ の大きさが 30,50,70,100,150,200,300,500 であるものをそれぞれ 10 題ずつ, また, 集合族 $\mathcal{F}$ の大きさが 50, 台集合の大きさが 30,50,70,100,150,200,300,500 であるものをそれぞれ 10 題ずつ発生させ, 解いた. これについても以下の表でまとめた. 小さな問題に比較的長い計算時間がかかっているのは, 初期設定などの時間が, 全体の計算時間に対して大きくなるからである.

表 1: $|E|$ を固定した場合の計算時間の比較

集合族の大きさ		30	50	70	100	150	200	300	500
改良 1	最大	43	30.9	29	34.8	42.8	65.9	56.2	54.2
$\|E\| = 50$	平均	29.9	26.3	25.9	27.6	30.3	40.6	39.7	39.7
	最小	23.5	19.9	22.9	24.1	22.1	26.7	25.8	21.3
改良 1	最大	932	506	411	456	525	679	833	1165
$\|E\| = 500$	平均	672.7	401.4	346.5	392.2	423.4	494.5	592.6	699.1
	最小	500	328	307	325	273	362	341	461
改良 2	最大	18.7	15.5	16.8	15.8	15.9	16.5	14.6	15.6
$\|E\| = 50$	平均	27.3	20.8	21.3	20.1	21.9	23.7	24	27.9
	最小	30.5	24.1	25.3	25.9	34.8	33.2	33.8	48.5
改良 2	最大	584	153	135	124	142	194	190	242
$\|E\| = 500$	平均	302	132.1	105.9	107.9	116.1	134.4	145.5	172.4
	最小	202	110	80.7	83	89.6	90	73.6	98.9

表 2: $|\mathcal{F}|$ を固定した場合の計算時間の比較

台集合の大きさ		30	50	70	100	150	200	300	500
改良 1	最大	17.8	28.7	40.5	59.5	110.5	175	268	512
$\|F\| = 50$	平均	14.9	24.2	35.9	51.1	83.7	125.2	217.2	389.4
	最小	12.2	18.6	31.5	44.8	65.5	101	194	296
改良 1	最大	28.1	69.3	90.7	149	211	326	572	1014
$\|F\| = 500$	平均	20	43.4	50.4	86.1	138.9	211.1	356.2	648.2
	最小	12.6	24.2	21.7	39.9	94.3	110	138	304
改良 2	最大	17.2	24.1	36.9	45.9	79.6	95	112	153
$\|F\| = 50$	平均	13.8	20.8	28.3	36.1	50.3	66.4	88.4	132.1
	最小	9.4	15.5	21	27.9	40.9	58.8	64.6	110
改良 2	最大	33.8	48.4	49.1	83.1	176.1	207	177	243
$\|F\| = 500$	平均	22.6	27.9	39.3	61.8	103.4	111.3	139.9	173
	最小	16.2	15.7	32.1	35.7	54.3	69.1	96	99

表 3: $|E|$ を固定し, 添え字をソートした場合の計算時間の比較

集合族の大きさ		30	50	70	100	150	200	300	500
改良 1	最大	19.2	20.1	23.2	27.7	38.2	44.7	51.9	64.5
$\|E\| = 50$	平均	15.8	18.2	19.6	23.1	26.8	32.2	31.7	37.9
少ない順にソート	最小	13.1	15.8	16.1	17.6	20.3	23.9	18	23.7
改良 1	最大	278	48.1	47.7	54.4	52.4	57.9	57.1	74.1
$\|E\| = 50$	平均	82.4	38.5	37.3	34.3	39.1	38.9	38.3	45.1
多い順にソート	最小	40	28.2	29.3	24.2	27.7	18.7	24.2	28.1

表 4: $|\mathcal{F}|$ を固定し, 添え字をソートした場合の計算時間の比較

台集合の大きさ		30	50	70	100	150	200	300	500
改良 1 最大	最大	12.3	20.1	26.7	39	66.6	87	131	244
$\|F\| = 50$	平均	10.7	18.2	23.3	35.6	56.4	75.9	117.5	210.7
少ない順にソート	最小	8.79	15.8	19.4	30.4	44.9	68.2	98.6	196
改良 1	最大	27.2	48.1	73.2	164	274	324	498	1339
$\|F\| = 50$	平均	21.4	38.5	59.2	101.1	175.8	244.9	422.3	944.5
多い順にソート	最小	17.7	28.2	44.4	68.5	140	189	340	720

これら結果より観察されたことを考察と共に以下で解説する.

1. 改良 1 を加えたアルゴリズムの, 解 1 つあたりの計算時間は, $|\mathcal{F}|$ を固定した場合, $O(|E|)$ より少々大きい程度であり, $|E|$ を固定した場合は $|\mathcal{F}|$ の増加に伴いゆるやかに増加する. つまり, $O((|E||\mathcal{F}|)^{\epsilon})$ ($\epsilon > 0$ は 1 よりも小さい数) 程度であろうと考えられる.

2. 改良 1 と改良 2 を加えたアルゴリズムの, 解 1 つあたりの計算時間は $|\mathcal{F}|$ を固定した場合, $O(|E|)$ より小さく, $|E|$ を固定した場合は $|\mathcal{F}|$ の増加に伴いゆるやかに増加する. つまり, $O((|E|^{1-\epsilon}|\mathcal{F}|^{\epsilon})$ 程度である. [3] では, アルゴリズムの 1 反復当たりの計算量は算定されていないが, 彼らの計算実験の結果から推測するに, 解 1 つあたりの計算時間は $O((|E|^2|\mathcal{F}|)^{\epsilon})$ 程

度であろうと推測されるので, 今回の改良により, 実験的なオーダーの減少が行われたと結論づけられる.

3. 実際の計算時間も大幅に短くなった. 問題の発生の仕方に違いがあるので, 純粋な比較は出来ないのであるが, [3] では, クロック周波数 450MHz のワークステーションを用いて, $|E| = 50, |\mathcal{F}| = 50$ 程度の問題を解くのに, 解 100 万個につき 300 秒ほどの時間をかけている. $|E| = 70, |\mathcal{F}| = 50$ になると, 解 100 万個につき 2000 秒ほどである. 彼らのプログラムは $C++$ で書かれているので, コンピューター・プログラム言語による差異により, 本稿のプログラムが有利になっているとは考えにくい.

4. 両者とも, 10 題の問題例に対して, 最大計算時間と最小計算時間は 2 倍から 3 倍程度であった. それほどおおきなばらつきはないようである.

5. 要素の添え字の並べ替えについては, 台集合の大きさ 50, 集合族の大きさ 50 の問題で, 少ない順による並び替えにより, 通常より 4 割ほど高速化が行われた. 逆に多い順による並び替えは, 計算時間が 2 倍程度に増えた. これらの速度差は, 台集合・集合族が大きくなると, 差が小さくなる. これは, 台集合・集合族が大きくなると, 各要素が含まれる $\mathcal{F}$ の集合の数にあまり差が無くなるためと思われる.

6. 要素の添え字の並び替えを行うと, 計算時間のばらつきが小さくなる. 少ない順, 多い順ともに, ばらつきが半分程度になった. 添え字の付け方の, 計算時間への関与は, 計算速度の向上だけではないようである.

5 まとめ

本稿では, 極小集合被覆 (ハイパーグラフ極小横断) 列挙問題に対する実用上高速なアルゴリズムを提案した. 計算実験により, 提案したアルゴリズムの平均的な計算時間が, 解 1 つ当たり $O(|E|)$ 程度であることを示した.

参考文献

[1] D. Avis and K. Fukuda. "Reverse Search for Enumeration," *Discrete Applied Mathematics*, 65:21–46, 1996.

[2] 福田 公明. 逆探索とその応用. 離散構造とアルゴリズム II, 近代科学社, 47–78, 1993.

[3] Dimitris J. Kavvadias, Elias C. Stavropoulos "Evaluation of an Algorithm for the Transversal Hypergraph Problem," *Algorithm Engineering*, pp. 72–84, 1999.

[4] M. L. Fredman and L. Khachiyan, "On The Complexity of Dualization of Monotone Disjunctive Normal Forms," *Journal of Algorithms,* **21**, pp. 618-628, 1996.