列挙アルゴリズム

宇野　毅明

1. はじめに

列挙問題（あるいは単に列挙）とは，与えられた問題のすべての解を重複なく出力する問題である．最適化が，与えられた目的関数を最大（最小）化する解を探し，システムのある種の極みの状態を見るという，いわば直線的な解析を行うのに対し，列挙はすべての解を見つけるため，問題の構造を多面的に解析していると考えられる．

列挙問題の難しさは，解をすべて出力するという完全性と，重複した出力をしないという非重複性にある．重複回避に関しては，発見した解をすべてメモリに保持し，新たな解が見つかるたびにメモリ上の解と比較する，といった素朴な方法もあるが，メモリ効率が良くない上に手間もかかる．そのため，いかにこのような直接的な手法を使わないか，という点が議論の中心となる．

列挙アルゴリズムの構築に関しては，主に3つの手法が知られている．バックトラック法，分割法，逆探索法である．本稿では，これらの構築手法を図解することで，手法の特徴をイメージ的に理解することを目標とする．

2. バックトラック法

バックトラック法（backtrack）は，列挙すべき解の集合が，ある種の単調性をもつときに，要素を1つ追加/削除，あるいは交換などの操作によって解集合を重複なく探索し，解集合からはみ出てしまったら戻る（バックトラックする），という列挙手法である．単調性より，解集合は図1のように連結な領域となるため，黒丸を出発点とする木型の探索ルートを通り，すべての解を探索できる．

列挙する解が台集合の部分集合である場合を例にとろう．台集合をEとし，列挙すべき解の集合を$F \subseteq 2^E$とする．ここで，Fは単調性を満たす，つまり任意のFの元に対して，その部分集合はすべてFに属するものとする．この様子を図示したものが，図2の左である．これは，各部分集合を大きさごとに分類し，包含関係が成り立つところに線を引いて得られるハッセ図である．この図で，単調性が成り立つ集合族は，図で示した網掛け部分のようになる．

単調性が成り立てば，任意の解はEの要素を逐次的に加えることで得られる．しかし，場当たり的に生成したのでは，数多くの重複を出力することになる．そこで，各部分集合は，その部分集合から最大の要素を除いたものからのみ作る，というルールを加えよう．逆に言えば，部分集合Sに要素を追加する際に，Sの最大の要素よりも大きな要素しか加えない，というルールを加えることである．このルールによって，隣接関係は図2の右側のようになり，探索による重複は回避される．以下にバックトラック法のアルゴリズムを記述しよう．このアルゴリズムは，初期値としてSに空集合を代入して呼び出す．空集合はFに含まれるとする．

BackTrack（E：台集合，S：現在の解）
1. Sを出力する
2. for each i∈E, i＞（Sの最大の要素）
　If S∪{i}がFの要素 then BackTrack（E, S∪{i}）

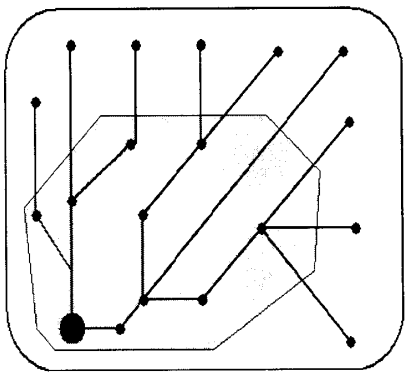

図1　バックトラック法のイメージ

うの　たけあき
国立情報学研究所
〒101-8430 千代田区一ツ橋2-1-2

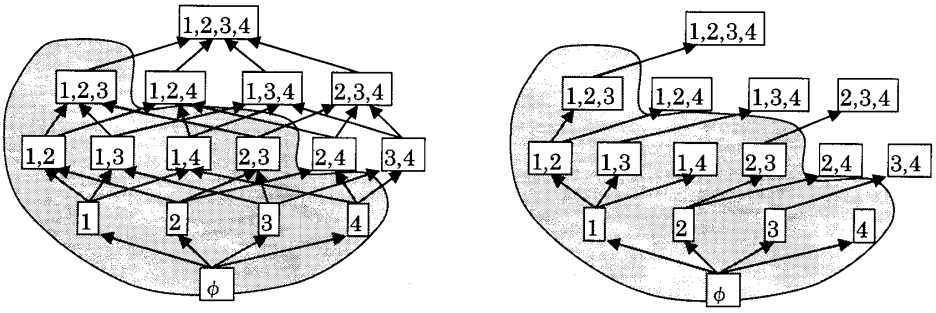

図2 左：部分集合が作るハッセ図と単調性を満たす部分集合族，右：最大となる要素のみ追加する，というルールが導く隣接関係

このアルゴリズムの反復数は解の数を超えない．そのため，このアルゴリズムは解1つにつきS∪{i}がFに属するかどうかの判定を最大|E|回行う．再帰の深さが高々|E|であることから，判定部分を除けば，このアルゴリズムの計算時間は解1つあたり入力の多項式時間となる．

3. 分割法

分割法（binary partition）は，問題の解集合をいくつかに分割し，それぞれを子問題として再帰的に解く列挙手法である．分割して得られる解集合は，要素を陽に保持するのではなく，「このグラフのパスの集合」といった問題の形で与える．分割を繰り返すと解集合は小さくなり，最終的には含まれる解が1つとなるため，それを出力する．変数の値を1つずつ固定していく分枝限定法的な視点で見るならば，分割法は，分割してできた子問題が解をもたない場合は実行を中止して引き返し，一番下のレベルまでいくと解を得ることができるアルゴリズムとみなせる（図3参照）．この手法の特徴は，分割を深さ優先的に行えばメモリが出力数に依存しないこと，および，分割の回数が解の総数を超えないので，解1つあたりの計算時間が入力の多項式時間となること，の2点にある．

例として，グラフの全張木を列挙するアルゴリズムを解説しよう[4]．無向グラフGに対して，Gのすべての頂点を含むGの部分木をGの**全張木**（spanning tree）という．ここで，与えられた無向グラフGの全張木をすべて出力する列挙問題を考える．

Gの枝eに対して，Gの全張木は「eを含むもの」と「eを含まないもの」の2つのグループに分類される．各グループの全張木を列挙する問題を子問題としたとき，両子問題を解くことができれば，それはもとの列挙問題を解いたことと等価である．

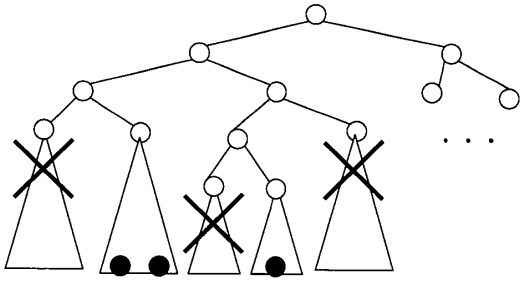

図3 分割法のイメージ：子孫（現在の問題）に解（黒丸）がなくなった反復は実行せず引き返す

「eを含まない全張木」は，Gからeを除去したグラフ（G\eとする）の全張木である．「eを含む全張木」はeを縮約して得られるグラフ（G/eとする）の全張木と1対1対応する．よって，これらのグラフに関する列挙問題を再帰的に解くことで，すべての全張木が列挙できる．このような分割を再帰的に繰り返し，グラフに全張木が1つしか含まれない，つまり入力グラフ自体が全張木となったら，分割を終了する．

全張木を2つのグループに分割する際に，空のグループと全部のグループ，という意味のない分割を行った場合，空のグループに関しては再帰呼び出しを行わない．あるいは，そのような分割を行わないように枝eを選ぶ．全張木の場合，閉路の任意の枝をeとして選べば両グループとも非空となるので，以下のアルゴリズムを得る．

SpanningTree（G：グラフ）
1. If Gが全張木を1つしか含まない then
   その全張木を出力；return
2. Gの閉路から枝eを選ぶ
3. SpanningTree（G/e）
4. SpanningTree（G\e）

4. 逆探索法

列挙アルゴリズムは，ある種の規則に従って解の空間を探索するアルゴリズムだと考えることができる．この視点から，逆に，数理的にまず探索ルートを構築し，そのルートに沿って探索を実行する，という手法が**逆探索法**（reverse search）である[1][2]．逆探索法は，ある意味で分割法やバックトラック法に比べて強力であり，しばしばこれらのアルゴリズムでは効率良く解くことのできない問題に対しても有効に働く．

逆探索法では，まず解の間に**親子**という関係を定める．そして，探索の出発点となる解以外のすべての解に対して，その親を唯一的に定義する．このとき，親子関係が巡回的にならない，つまりある解に対して，その親の親の……とたどることでその解に戻ってくることがない，という条件を満たすようにする．この条件を満たす親子関係は，すべての解を張り，探索の出発点となる解を根とする，**列挙木**とよばれる根付き木(図4) を誘導する．列挙木は解集合を張る木型の探索ルートとなる．

バックトラック法は解集合の単調性を必要とし，分割法は，問題が再帰的に解けることを必要とするが，逆探索法は，この両者を満たさないような問題，つまりバックトラック法や分割法では効率の良いアルゴリズムが設計できないような問題でも，解の隣接性を見つけるだけで列挙アルゴリズムが設計できるという点が強みである．

この列挙木を深さ優先探索すれば，すべての解を列挙できる．深さ優先探索の各反復では，現在訪れている解に対して，その子供を見つけ，各子供に対して深さ優先探索の手続きを再帰呼び出しする．これにより，列挙木をメモリ上に陽にもたずとも深さ優先探索が可能となり，使用するメモリも，子どもを列挙する部分が必要とするメモリ以外は入力の多項式で抑えられる．

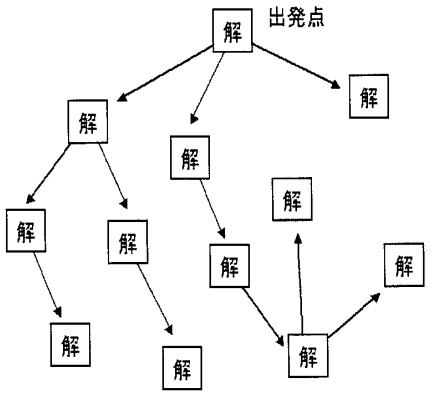

図4　解集合上の探索ルート．逆探索では，矢印の根本が親，先が子になる

逆探索法の効率は，親子関係の定義と深く関わっている．上記の条件を満たす親子関係を構築すること自体は容易である．例えば，解集合を辞書順でソートし，各解の親をその解の前にある解で定義すればよい．しかし，このように定義した親子関係では，子供を見つける作業に手間がかかることが多く，効率的なアルゴリズムの構築は望めない．

逆探索の例を**多面体の端点列挙問題**で解説しよう[1][2]．問題は，正確には，線形不等式系で与えられた多面体の，すべての実行可能基底を列挙するものである．

実行可能基底は制約式の組合せで決まる．制約式の組合せを変更すると，隣接する実行可能基底に移動できる．この際，ある種のルールに沿って，目的関数が良くなるほうに移動すると，任意の実行可能解から非巡回的に移動して最適解に到達できる．これを用いた最適化手法がシンプレックス法である．

今，適当な目的関数を選び，各実行可能基底に対してこのルールによる移動先を親と定め，親子関係を定義する．これは親を唯一的に定め，先の性質から非巡回的であるという条件も満たす．親のいない基底は，最適解である．

この親子関係が定める列挙木を探索するには，まず最適基底解を見つけ，そこから子どもを再帰的に見つければよい．可能な制約式の入れ替え（制約式を1つ入れ，1つ出す）によって移動できる実行可能基底それぞれに対して，その親を求める．親が自分と一致した場合，子どもであると確認できるので，その子どもに移動して再帰的に子孫の列挙を行う．制約式2つの組合せは高々制約式の数の2乗であるので，計算時間は1反復あたり，つまり解1つ当たり入力の多項式となる．

5. 実用上の高速化

列挙を現実問題に適用する場合，問題はそれなりに大きいが解はそれほど多くない，という場合が多い．むしろ，そのような場合でなければ列挙を使う妙味がないというものだ．このような場合，解1つあたりの計算時間は，入力の線形であっても現実的な時間では解けず，定数時間に近いことが望ましい．これは，列挙アルゴリズムの計算構造を観察することで，達成できる．

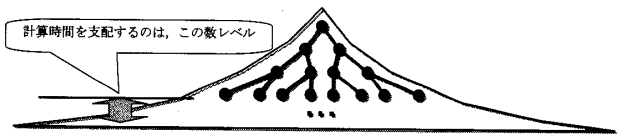

図5　典型的な，列挙アルゴリズムの計算の構造

一般に，列挙アルゴリズムは再帰型の計算構造をしている．各反復が複数の再帰呼び出しを行うため，葉に近づくに従い，反復数が指数的に増加する傾向がある．そのため計算木は，図5で示すような非常に先が広がった構造をとる．この構造を**末広がり性**（bottom-wideness）とよぼう．

末広がり性がある場合，アルゴリズムの計算時間は木の末端の数レベルに位置する反復の計算時間によって支配される．そのため，これら下のレベルにある反復の計算時間を短くできれば，その分だけ全体の計算時間が短くなる．また，下のレベルの反復は，入力した問題の中に解を少数しか含まない．そのため，すべての解に含まれない，あるいはすべてに含まれる部分を問題から除去する，という作業を各反復で行うことで，再帰的に問題を小さくでき，下のレベルでは定数に近い大きさまで減少させることができる．そのため，多くの列挙アルゴリズムでは，反復的縮約と呼ばれるこの手法を用いることで，実際の計算時間が大幅に短縮される．

6．終わりに

本稿では，列挙アルゴリズムの構築法に関するイメージ的な理解を助けるべく解説を行った．列挙アルゴリズムは近年情報学の様々な分野で使われており，OR的な問題に対しても有効であると考えられる．読者の列挙的な問題解決に本稿が資すれば幸いである．列挙アルゴリズムに関するより詳しい解説は文献[3]を，列挙アルゴリズムの実装については著者のホームページ[5]を参照されたい．

参考文献

[1] D. Avis and K. Fukuda: Reverse Search for Enumeration, Discrete Applied Mathematics, Vol. 65, p. 21, 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，p. 47, 1993.

[3] 応用数理計画ハンドブック，朝倉書店.

[4] T. Uno: A New Approach for Speeding Up Enumeration Algorithms and Its Application for Matroid Bases, Lecture Notes in Computer Science, 1627, p. 349, 1999.

[5] 宇野毅明のホームページ，http://research.nii.ac.jp/~uno/index-j.html