1995年度日本オペレーションズ・リサーチ学会
秋季研究発表会

全張木を重さの軽い順に列挙する
Ranking all the spanning trees

01605000 東京大学 松井知己 MATSUI Tomomi
01605630 東京都立大学 * 松井泰子 MATSUI Yasuko

1. はじめに

$G=(V,E)$ を、頂点集合 V と枝集合 E からなる無向グラフとする。また $|V|=n$, $|E|=m$ とする。ここでは、グラフ G は自己閉路も平行枝も含まない連結グラフとする。ある枝の部分集合 $T\subset E$ において、グラフ (V,T) が閉路を含まない連結グラフであるとき、T はグラフ G の全張木であるという。グラフ G の全張木すべての集合を $\mathcal{T}$ とする。グラフの各枝には、枝重みが与えられているものとする。全張木の重さは、その木に含まれている枝の重みの総和と定義する。本稿では、全張木はすべて異なる重さを持っていると仮定する。この仮定は各枝の重さを摂動することによって容易に満たすことができる。重さの最も軽い全張木を最小全張木と呼ぶ。本発表では、与えられたグラフ G の全張木を、その重みの軽い順に列挙する方法について議論する。

このような問題を解く算法は、順位列挙算法と呼ばれる。最小全張木問題に対する順位列挙算法は、全張木問題にさらにいくつか制約が加わったような問題を解く際に有効である。すなわち、順位列挙算法を実行して、全張木を最も軽いものから順次列挙し（付加的な）制約を満たすような解が最初に得られた時点で算法を停止することによって、目的の解を得ることができる。

この問題に関する過去の研究としては、[1, 2] 等が存在するが、必要記憶容量に着目して計算機実験を行なった報告は（筆者の知る限り）存在していない。

2. 解法の基本構造

本稿で考慮する算法では、以下のような列挙木を仮想的に定義し、この列挙木を構築することによって、全張木を軽い順に列挙する。本稿で用いる列挙木 $(\mathcal{T},T^*,\phi)$ は次の性質を満たすものである。

(1) 列挙木のノード集合は、グラフ G の全張木全体の集合 $\mathcal{T}$ である。
(2) 列挙木のルートノード T^* の全張木は、グラフの最小全張木である。
(3) ルートノード以外のノード T において、T の親ノードは $\phi(T)$ と表され、$\phi(T)$ の重さは T の重さと同じかより軽い。

上記のような列挙木が仮想的に定義されているものとすると、以下のような一般的な順位列挙算法を構築することができる。

Algorithm A

Step 1: $\mathcal{Q}=\{T^*\}$ とする.

Step 2: $\mathcal{Q}$ が空ならば、終了.

Step 3: 集合 $\mathcal{Q}$ 中の全張木で、重さの最も軽い全張木 T を取り出し、T を出力する.

Step 4: T の子ノードを生成し $\mathcal{Q}$ に入れ、Step 2 へ戻る.

現実において全張木を順位列挙する際は、全ての全張木を列挙しなければならないことは少なく、上記の算法は適当な時点で終了させることが多い。

上記のような算法を実行した際、集合 $\mathcal{Q}$ の大きさが算法の進行に従い急速に大きくなることがしばしばある。さらに、算法の実行時間の大部分を集合 $\mathcal{Q}$ の取り扱いにかかる時間が占めてしまったり、集合 $\mathcal{Q}$ が多き過ぎて保持することが困難となる場合もある。本稿では、集合 $\mathcal{Q}$ の大きさが急速に大きくならないような列挙木について議論する。

3. 子ノードの生成

以下では、前節での性質に加え、次の性質を満たす列挙木についてのみ議論する。

(4) 親子関係にある任意の全張木の対は、丁度2本の枝が異なる。

性質 (4) を満たす全張木の対は、全張木多面体において隣接していることが知られている。またこの二つの全張木対は、一方の全張木からある枝を取り除き、そのとき生成される基本カットセットから枝を一本選び加えるという操作によって、多方の全張木が生成されるという性質を持っている。

T^* 以外の任意の全張木に対し、その親の全張木を生成することのできる多項式算法が存在するならば、任意の全張木 T に対しその子ノードを以下のように多項式時間で生成することができる。T 中の枝 e と T に含まれない枝 f の全ての対に対し、$T-e+f$ が全張木であり、かつ $T+e-f$ の親ノード

© 日本オペレーションズ・リサーチ学会. 無断複写・複製・転載を禁ず.

が T であるかを確認する。もしこれが満たされるならば、$T-e+f$ は T の子ノードである。T 中の枝 e と T に含まれない枝 f の全ての対の数はたかだか nm であることから、上記の手続きによって全ての子ノードを多項式時間で生成することができる。

4. 列挙木

本稿では、以下で4種類の列挙木を定義し、それぞれの列挙木を用いた際の Q の大きさの増加について議論する。

本節では、与えられたグラフの枝集合 E は、枝の重みの小さい順に $\{e_1, e_2, \ldots, e_m\}$ と番号付けられているとする。枝 e_j に対し、j を e_j の添字と呼び $\mathrm{Idx}(e_j)$ と書く。T^* を、辞書式順序において最小の木とする。このとき T^* が最小木となることが知られている。以下で定義する4つの列挙木では、そのルートノードを T^* とする。全張木 T に対し、$P(T) = \{f \in T \mid \exists e,\ T+e-f$は全張木, $\mathrm{Idx}(e) < \mathrm{Idx}(f)\}$ と定義する。枝 $e \in T$ に対し $\mathrm{cut}(T,e)$ を e によって定まる基本カットセットとする。

LL 列挙木: T^* と異なる任意の全張木 T に対し、その親を $T+e-f$ と定義する。ただし f は $P(T)$ の中で最も大きな添字を持つものであり、e は $\mathrm{cut}(T,f)$ 中の f より小さな添字を持つ枝の中で最も大きな添字を持つものである。

LS 列挙木: T^* と異なる任意の全張木 T に対し、その親を $T+e-f$ と定義する。ただし f は $P(T)$ の中で最も大きな添字を持つものであり、e は $\mathrm{cut}(T,f)$ の中で最も小さな添字を持つものである。

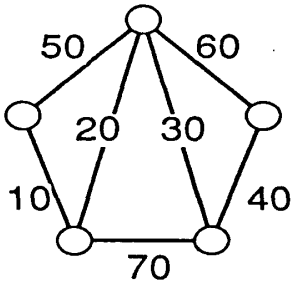
SS 列挙木: T^* と異なる任意の全張木 T に対し、その親を $T+e-f$ と定義する。ただし f は $P(T)$ の中で最も小さな添字を持つものであり、e は $\mathrm{cut}(T,f)$ の中で最も小さな添字を持つものである。

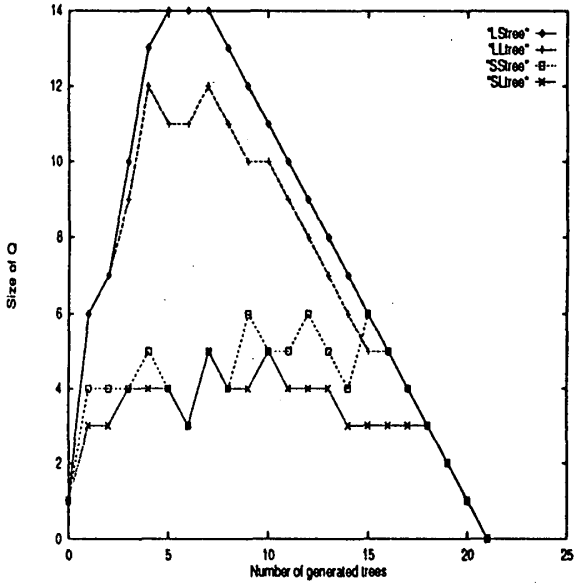
SL 列挙木: T^* と異なる任意の全張木 T に対し、その親を $T+e-f$ と定義する。ただし f は $P(T)$ の中で最も小さな添字を持つものであり、e は $\mathrm{cut}(T,f)$ 中の f より小さな添字を持つ枝の中で最も大きな添字を持つものである。

上記の4つの規則によって定まる列挙木が、前出の条件 (1) ～ (4) を満たしていることは容易に分かる。

5. 数値例

右のようなグラフにおいて、上記の4つのルールを用いた際の集合 Q の大きさを調べた。下図の横軸は生成された全張木の数であり、縦軸は各木が生成された直後の Q の大きさを表している。





上記の例では、必要記憶容量の意味で、SL 列挙木が最も良いと思われる。また、いくつかの予備的な実験においても、SL 列挙木は4つの列挙木の中でも最も良いことが確認されている。発表においては、さらに詳細な計算機実験について報告を行なう予定である。

本研究をすすめるにあたって御協力いただいた藤越慎治氏に感謝いたします。

REFERENCES

[1] S. Kapoor and H. Ramesh. Algorithms for generating all spanning trees of undirected, directed and weighted graphs. In F. Dehne, J.-R. Sack, and N. Santoro, editors, *Lecture Notes in Computer Science*, volume 519, pp. 461–472. Springer-Verlag, 1992.

[2] N. Katoh, T. Ibaraki, and H. Mine. An algorithm for finding K minimum spanning trees. *SIAM J. Computing*, 10 pp.247–255, 1981.

© 日本オペレーションズ・リサーチ学会. 無断複写・複製・転載を禁ず.