数理解析研究所講究録
第731巻 1990年 187-194

内点法の平面ネットワークフロー問題への適用

今井 浩* 田川 勇治

九州大学工学部情報工学科

アブストラクト

一般の線形計画問題に対する解法である内点法を、線形計画問題の特殊な場合である平面ネットワーク最小費用流問題に適用することについて考察し、従来、有限要素法などで知られていた離散数値計算技法 (George [2], Lipton, Rose, Tarjan [5]) が適用できることを示す。中心パスを追跡する内点法で再出発を定期的に行なう Vaidya [7] のアルゴリズムと、この技法を組み合わせることにより、容量および費用がすべて γ 以下の整数であるような n 点の平面ネットワークでの最小費用流問題が、 $O(n^{1.75}\sqrt{\log n}\log(\gamma n))$ の手間で解けることに触れる。また、効率的コレスキー分解の部分だけを用いた計算機実験の結果も示す。

1. はじめに

一般の線形計画問題に関しては、最近、内点法とよばれる単体法と同じぐらい古くからある方法が、そのいくつかの変形版が線形計画問題を多項式時間で解くということで、注目を浴びている。特に、Vaidya [7] は、内点法の従来の手法と高速行列乗算のアルゴリズムをうまく組み合わせることにより、 M 制約式、 N 変数、入力サイズ L の線形計画問題が $O((M+N)^{1.5}NL)$ 回の $O(L)$ ビットの数の演算で解けることを示している。また、Vaidya [7] は、このアルゴリズムをネットワークフロー問題に適用し、その特殊性（特に制約行列の疎構造）を利用して解析することにより、 m 辺、 n 点のネットワークで、容量および費用がすべて γ 以下の整数であるようなネットワークでの最小費用流問題が、 $O(\sqrt{m}n^2\log(\gamma n))$ の手間で解けることを示している。

本稿では、ネットワークフロー問題の中でも、平面ネットワークに関するものを考える。この平面ネットワークフロー問題に対して内点法を適用したときには、内点法の探索方向を求める際の線形方程式を解く部分で、有限要素法などで用いられていた線形計算における離散的手法（nessted dissection とよばれる）を用いることができる（今井 [3]）。この性質を Vaidya [7] の結果と組み合わせることにより、上述のようなネットワークで、ネットワークが平面である場合には、高

* 現在：東京大学理学部情報科学科

速行列乗算を用いることなく、最小費用流問題をより効率的に解くことができる。具体的には、$O(n^{1.75}\sqrt{\log n}\log(\gamma n))$ の手間で平面ネットワーク最小費用流問題を解くことができる。以下では、この結果について簡単に触れ、また、実際に格子ネットワーク上の最小費用流問題を例に計算機実験を行なった結果について述べる。

2. ネットワークフロー問題と内点法

点集合 $\{v_1, v_2, \ldots, v_n\}$ 、有向枝集合 $\{a_1, a_2, \ldots, a_m\}$ からなる連結グラフ G で、容量ベクトル $\boldsymbol{c} = (c_j)$ 、費用ベクトル $\boldsymbol{d} = (d_j)$ が与えられ(枝 a_j の容量 c_j 、費用 d_j)、流出・流入量ベクトル $\boldsymbol{b} = (b_i)$ (点 v_i で $b_i > 0$ のとき流出、 $b_i < 0$ のとき $|b_i|$ 流入) が与えられたネットワーク N を考える。グラフ G の接続行列を A とする。すなわち、 $A = (a_{ij})$ は $n \times m$ 行列で、

$$a_{ij} = \begin{cases} 1 & \text{点 } v_i \text{ から枝 } a_j \text{ が出ている} \\ -1 & \text{点 } v_i \text{ に枝 } a_j \text{ が入っている} \\ 0 & \text{それ以外のとき} \end{cases}$$

流れベクトル $\boldsymbol{x} = (x_j)$ で枝 a_j 上の流れ x_j を表わすと、このネットワーク上での最小費用流問題は、次のように表わせる。

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{d}^{\mathrm{T}}\boldsymbol{x} \\ \text{s.t.} \quad & A\boldsymbol{x} = \boldsymbol{b} \\ & 0 \le \boldsymbol{x} \le \boldsymbol{c} \end{aligned} \tag{P}$$

n 点よりなる連結グラフ G の接続行列の階数は $n-1$ である。このネットワークに 1 点人為的な点 v_0 と、その点 v_0 と各点 v_i を両方向に結ぶ人為的な有向枝各 2 本(容量 $+\infty$ で、費用も十分大きいとする) を付け加えたネットワークを考え、その接続行列の内、点 v_0 に関する行を除いた行列 $\widetilde{A}$ を用いて、次の等価な問題を考える。

$$\begin{aligned} \text{minimize} \quad & \widetilde{\boldsymbol{d}}^{\mathrm{T}}\widetilde{\boldsymbol{x}} \\ \text{s.t.} \quad & \widetilde{A}\widetilde{\boldsymbol{x}} = b \\ & 0 \le \widetilde{\boldsymbol{x}} \le \widetilde{\boldsymbol{c}} \end{aligned} \tag{P'}$$

但し、 $\widetilde{A} = (A \mid I \mid -I)$ (I は $n \times n$ 単位行列)、 $\widetilde{\boldsymbol{c}} = (c_1, \ldots, c_{m+2n})^{\mathrm{T}}$ 、 $c_j = +\infty$ $(j = m+1, \ldots, m+2n)$ 、 $\widetilde{\boldsymbol{d}} = (d_1, \ldots, d_{m+2n})^{\mathrm{T}}$ 、 $d_j = M$ $(j = m+1, \ldots, m+2n)$ 、 M は十分大きな正の数とする。行列 $\widetilde{A}$ の階数は n である。

内点法の多くのアルゴリズムでは、 $\widetilde{A}$ と $(m+2n) \times (m+2n)$ の対角行列 D を用いて

$$B = \widetilde{A}D\widetilde{A}$$

表 3.1. nested dissection によるピボット順でコレスキー分解した時の上 (下) 三角行列の非零数 (対角要素を含む)

k	20	30	40	50	60	70	80	90	100	110	120
n	400	900	1600	2500	3600	4900	6400	8100	10000	12100	14400
非零数	5008	14372	29195	50968	79789	115770	159164	211639	272009	342133	419592

で定められる $n \times n$ の対称行列 B に関する線形方程式を解く。 B の ij 要素は、元のグラフ G で点 v_i, v_j 間に枝がないときには、必ず 0 であり、枝があるときには通常 0 でない。すなわち、B の非零パタンは、グラフ G の隣接行列と通常同じであり、隣接行列で 0 のところは B で必ず 0 である。

3. 平面グラフの隣接行列と同じ非零パタンの行列のコレスキー分解

対称行列に関する線形方程式をコレスキー分解で解くときの中心的な問題の一つは、行と列の対称な入れ換えによりフィルイン (コレスキー分解の過程で行列の元々は 0 であった要素が 0 でなくなるところ) をできるだけ少なくするという離散的最適化問題である。線形計画問題を上述のようなアプローチで解くには、制約行列からまず解くべき線形方程式の行列を構成し、それを解くわけだが、ここで元の制約行列の疎構造が線形方程式の行列に反映され、さらにその疎構造がうまく少ないフィルインに結びついたとき、内点法の効率がたいへん良くなる。フィルインを最小にする問題は、NP 困難であり [8]、フィルインをある程度少なくするための発見的解法が考えられている。[1] では、代表的な二つの発見的解法を線形計画問題に適用し、比較している。一方、理論的に一般の線形計画問題を対象としたフィルイン最小化問題の議論を行なうことは、難しいと思われる。

$n \times n$ 対称行列 B に関する線形方程式を解くとき、 B の非零パタンが格子グラフ、平面グラフの隣接行列と (ある程度) 同じなら、最小フィルイン数の高々定数倍のフィルイン数で線形方程式を解く方法が知られている [2,5]。この方法によれば、 $O(n \log n)$ のフィルイン数で、 $O(n^{1.5})$ の手間で方程式が解ける。この方法の必要とする記憶容量の大きさは、フィルイン数に比例し、また行・列のピボット順を計算する部分は、 $O(n \log n)$ の手間で行える。また、この方程式をコレスキー分解で解くときには、 $n^{1.5}$ に比例する以上の演算が必要であることも示されており、その意味で最適である。

$k \times k$ の格子グラフを例にすると、このピボット順番は、まず $k \times k$ 格子をまん中の行・列によりほぼ 1/4 の大きさの格子 4 つに分割し、今選んだ行・列の点の番号を他の点よりも大きくし、4 つの部分格子各々に対して再帰的にこの手続きを適用していくことにより得られる。実際のフィルイン数を表 3.1 に示す。平面グラフの場合には、いわゆる平面グラフ分割定理 [6] を用いて同様なことを行なう。

4. 内点法を平面ネットワークフロー問題に適用したときの計算の手間

Vaidya [7] は、内点法の従来の手法と高速行列乗算のアルゴリズムをうまく組み合わせることにより、 M 制約式、 N 変数、入力サイズ L の線形計画問題が $O((M+N)^{1.5}NL)$ 回の $O(L)$ ビットの数の演算で解けることを示している。また、Vaidya [7] は、このアルゴリズムをネットワークフロー問題に適用し、その特殊性（特に制約行列の疎構造）を利用して解析することにより、 m 辺、 n 点のネットワークで、容量および費用がすべて γ 以下の整数であるようなネットワークでの最小費用流問題が、 $O(\sqrt{m}n^2\log(\gamma n))$ の手間で解けることを示している。

このアルゴリズムでは、 r 回ごとに線形方程式を完全に解き直し、その r 回の反復の過程では、ランク 1 の更新によって近似的に線形方程式を解いていく。全体で必要とする反復数は、$O(\sqrt{m}\log(\gamma n))$ である。おおざっぱにこのアルゴリズムを最小費用流問題に適用した場合の各反復の手間について述べると、線形方程式を完全に解くのに $O(m+n^{2.4})$ の手間（高速行列乗算のアルゴリズムを用いる）、1 反復あたりのランク 1 の更新に $O(nr+r^5)$ の手間がかかる（ r 回ごとに線形方程式を解く間に、それぞれ $O(r^2)$ 回のランク 1 の更新が必要となる）。従って、1 反復当たり

$$O(\frac{m+n^{2.4}}{r}+(nr+r^5))$$

の手間がかかる。 $r=n^{0.4}$ とすると、この 1 反復あたりの手間は、 $O(n^2)$ となり、これに反復数をかけると上述の最小費用流問題を解く手間が出てくる。

この内点法アルゴリズムを平面ネットワーク最小費用流問題に適用することを考える。以下ではおおざっぱな議論を行なうが、実際には細部についての詳細な議論が必要である。この場合、線形方程式を完全に解き直すのに $O(n^{1.5})$ の手間、ランク 1 の更新に $O(n\log n)$ の手間がかかる。ランク 1 の更新は、 r 回の反復の間に $O(r^2)$ 回行なわれ、それに付随して反復あたり $O(r^5)$ の手間がかかる。従って、1 反復あたりの計算にかかる手間は

$$O(\frac{n^{1.5}}{r}+rn\log n+r^5)$$

となる。ここで、 $r=n^{0.25}/\sqrt{\log n}$ とすると、1 反復あたりの手間は、 $O(n^{1.25}\log n)$ となる。全体での反復回数が、 $O(\sqrt{n}\log(\gamma n))$ であるから、次の定理を得る。

定理．平面ネットワーク最小費用流問題は、 $O(n^{1.75}\sqrt{\log n}\log(\gamma n))$ の手間で解ける。

この結果を、平面ネットワークでの多品種流問題に拡張することもできる。

5. 格子ネットワークでの計算機実験

2, 3 節の方法をプログラミングし、計算機実験を行なった（従って、4 節の再出発を行なう方法は用いていない）。対象としては、 $k\times k$ 無向格子ネットワーク (従って、点数 $n=k^2$) 上での

容量制限無しの最小費用流問題を考え、内点法としては、問題 (P′) の双対問題に対するアフィンスケーリング法 (たとえば [1]) を考えた。無向ネットワークは、各無向枝を両方向の有向枝 2 本で置き換えることにより有向ネットワークの問題に変換できる。容量制限無しとしているので、双対問題は次のように表わせる。

$$\text{maximize} \quad \boldsymbol{b}^{\mathrm{T}}\boldsymbol{y}$$

$$\text{s.t.} \quad \tilde{A}^{\mathrm{T}}\boldsymbol{y} \leq \tilde{\boldsymbol{d}}$$

初期実行可能解は、 $\boldsymbol{y} = 0$ を用いた (費用ベクトル $\tilde{\boldsymbol{d}}$ を下記のように正にしていることから、これで実行可能な内点となる)。直線探索では、制約に当たるまでの 0.99 進むという方式を取った。また、停止条件は、後述するネットワークシンプレックス法のコードとの比較もあり、各ステップの目的関数値の増加がその時点での目的関数値の 10^{-5} 未満になれば停止するというものを用いた。

費用 $\boldsymbol{d}$ 、流出・流入量ベクトル $\boldsymbol{b}$ に関しては、次の 2 つの場合を考えた。

(Case 1) 各枝の費用は、1 から 10^5 までのランダムな整数、 $k \times k$ 格子の i 行、 j 列の点の流出・流入量を i $(i \leq k/2)$ 、 $i-k$ $(i > k/2)$ とした;

(Case 2) 各枝の費用は、 $10^5 \pm 10^2$ の間の整数、 $k \times k$ 格子の i 行、 j 列の点の流出・流入量を j $(i = 1)$, i $(1 < i \leq k/2)$, $i-k$ $(k/2 < i < k)$, j $(i = k)$ とした;

また、 M は 10^3 とした。

プログラムは FORTRAN で行なった。実数計算は、すべて倍精度で行なっている。アフィンスケーリング法のプログラムでも、一部格子ネットワークに特別なデータ構造などを用いており、一方コレスキー分解の部分のプログラムにはまだ少し改良の余地がある。計算機実験に用いた計算機は、VAX8800 (Ultrix V2.3) で、f77 コマンドで -O(最適化) を指定してコンパイルした後実行した。表 5.1, 5.2 に計算機実験の結果を示す (本来ならグラフ等を用いて、よりわかりやすく示すべきであるが、ここではまだ内点法としてアフィンスケーリング法だけ扱っていることでもあり、総合的な報告は別の機会にゆずることとする)。計算機時間には、問題の入出力の時間は含まれていない。(Case 1) については、費用の乱数を 2 タイプ考え、その平均を示している。

また、同時に Kennington, Helgason [4] による一般の最小費用流問題に対するネットワークシンプレックス法のコードを [4] より入力し、問題 (P) に対して適用し、参考のため比較した。このコードは、一般の最小費用流問題に対するものであり、汎用性がある。ただ、このコードがネットワークフローの種々のコードの中で、最高速であるというわけでもない [4]。また、比較の際には、アフィンスケーリング法のプログラムでの停止条件との兼ね合いなどを詳細に考えなければいけない。従って、ここでの比較では、あくまでもネットワークの点数を増やしていったときの両コードの要する計算機時間の増加のオーダなどの点に注目すべきである。また、費用ベクトルなど

表 5.1. $k \times k$ 格子ネットワーク最小費用流問題での計算実験結果: (Case 1)

問題の大きさ		アフィンスケーリング:A		シンプレックス: S		計算時間比
k	n	反復回数	時間 (s)	反復回数	時間 (s)	S/A
20	400	10	2.39	458	0.43	0.18
30	900	11	9.44	1007	1.60	0.17
40	1600	10	21.19	1788	4.18	0.20
50	2500	11	46.46	2816	9.28	0.20
60	3600	11	82.23	4094	17.71	0.22
70	4900	11	134.63	5510	29.94	0.22
80	6400	11	203.86	7345	49.63	0.24
90	8100	11	297.60	9098	74.95	0.25
100	10000	11	415.74	11528	111.38	0.27
110	12100	11	564.49	13933	162.27	0.29
120	14400	11	737.56	16987	237.15	0.32

表 5.2. $k \times k$ 格子ネットワーク最小費用流問題での計算実験結果: (Case 2)

問題の大きさ		アフィンスケーリング:A		シンプレックス: S		計算時間比
k	n	反復回数	時間 (s)	反復回数	時間 (s)	S/A
20	400	12	2.88	942	0.98	0.34
30	900	11	9.38	2482	4.83	0.51
40	1600	11	23.12	5186	16.12	0.70
50	2500	12	50.92	9298	37.15	0.73
60	3600	12	90.73	14412	78.23	0.86
70	4900	12	149.55	21304	146.48	0.98
80	6400	12	226.77	30532	245.02	1.08
90	8100	12	330.40	38923	382.85	1.16
100	10000	12	461.52	53161	593.52	1.29
110	12100	12	627.15	66526	902.77	1.44
120	14400	12	822.51	86740	1273.25	1.55

について 2 つの場合を考えているが、この (Case 1) と (Case 2) では、ネットワークシンプレックス法のコードの方が、性能に大きな違いが生じる例となっており、これだけで一般的なことがいえるわけではない。

また、アフィンスケーリング法を C 言語でもプログラミングし、そのコードで格子ネットワーク上の最短路問題を解いた。始点を格子ネットワークの左下の点とし、終点を右上の点として、その間の最短路を求めた。枝の費用（長さ）は、1 から 100 の間のランダムな整数とした。収束などに関しては、同様の条件を用いた。2 組の乱数に対する計算時間と反復数の平均を示したものが表 5.3 である。この最短路問題の場合、最小費用流のときと違って、反復回数が問題サイズとともに増加している点が興味深い。

アフィンスケーリング法のコードは、最小費用流問題の場合、問題の大きさに関わらずほぼ一定 (10 から 12) の反復回数しか要さない（最短路問題に対してはそうでないことが観察されるが）。

表 5.3. $k \times k$ 格子ネットワーク最短路問題での計算実験結果

問題の大きさ		アフィンスケーリング:A	
k	n	反復回数	時間 (s)
10	100	10.5	0.49
20	400	12.5	4.31
30	900	12.5	15.61
40	1600	14	43.57
50	2500	15	101.71
60	3600	15.5	177.04
70	4900	16	302.12
80	6400	17	486.99
90	8100	18	764.50
100	10000	18	1077.51
110	12100	19.5	1588.43

各反復が、線形方程式を解くところで $O(n^{1.5})$ かかっていることから、全体でほぼ $n^{1.5}$ に比例した時間がかかっている (一部、コレスキー分解のプログラムでの不備な点の影響がでているが)。また、この (Case 1), (Case 2) に関してはあまり差がない。一方、ネットワークシンプレックス法のコードは、$n^{1.8\sim2}$ に比例した時間がかかっており、n が莫大となれば、この停止基準の下では、アフィンスケーリング法のコードの方がいつか早くなることが見て取れる (また、この例に関しては (Case 2) で実際にそうなっている)。停止基準については、対象としているものがネットワーク問題であるので、その性質を用いて最適性の判定を行なった上での比較が本来は望まれる。記憶領域の大きさに関しては、アフィンスケーリング法のコードは 2 節で示したフィルイン数の配列を必要とするため、かなりシンプレックス法のコードに比べて多くを必要とする。

6. おわりに

内点法をネットワークフロー問題などの特殊なクラスの線形計画問題に適用して、そのクラスの問題に対する効率のよいアルゴリズムを構成するというアプローチでは、適用した問題の性質をどれぐらい生かせるかで、アルゴリズムの性能が決まってくる。本稿では、そのようなアプローチの 1 つとして、平面ネットワークフロー問題に内点法を適用し、平面ネットワークの性質を内点法のアルゴリズムの中で十分利用できることを示した。また、計算機実験により、実際に特殊化したアルゴリズムが、既存の他の方法とある程度同じ性能を有することも示した。最近では、内点法を用いてネットワーク問題に対する高速並列アルゴリズムを構成するということも考えられている。今後とも、内点法を統一的な観点から研究していくとともに、その特殊化について調べていく必要があるだろう。

参考文献

[1] I. Adler, N. Karmarkar, M. G. C. Resende and G. Veiga: An Implemetation of Karmarkar's Algorithm for Linear Programming. Manucript, May 1986, revised June 1987.

[2] A. George: Nested Dissection of a Regular Finite Element Mesh. *SIAM Journal on Numerical Analysis*, Vol.10, No.2 (1973), pp.345–363.

[3] 今井浩: 平面ネットワーク最小費用流問題に対する内点法の高速化について. 情報処理学会アルゴリズム研究会研究報告 89-AL-9-1, 1989.

[4] J. L. Kennington and R. V. Helgason: *Algorithms for Network Programming*. John Wiley & Sons, New York, 1980.

[5] R. J. Lipton, D. J. Rose and R. E. Tarjan: Generalized Nested Dissection. *SIAM Journal on Numerical Analysis*, Vol.16, No.2 (1979), pp.346–358.

[6] R. J. Lipton and R. E. Tarjan: A Separator Theorem for Planar Graphs. *SIAM Journal on Applied Mathematics*, Vol.36, No.2 (1979), pp.177-189.

[7] P. Vaidya: Speeding-Up Linear Programming Using Fast Matrix Multiplication. *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, 1989, pp.332–337.

[8] M. Yannakakis: Computing the Minimum Fill-In is NP-Complete. *SIAM Journal on Algebraic and Discrete Methods*, Vol.2, No.1 (1981), pp.77–79.