2006－AL－104（10）
2006／1／20

最小費用流問題アルゴリズムに対する実験的性能評価

慶祐 俊文*　高藤 大介*　田岡 智志*　渡邉 敏正*

*広島大学大学院 工学研究科 情報工学専攻
〒 739-8527 東広島市鏡山一丁目 4-1
(電話) 082-424-7662 (渡邉), -7666 (田岡)
(ファクシミリ) 0824-22-7028
(電子メール) {keiyu, daisuke, taoka, watanabe}@infonets.hiroshima-u.ac.jp

[概要] 最小費用流問題とは,「有向グラフ $G(V,E)$ と各辺 $(i,j) \in E$ に対する単位フローあたりの非負整数コスト $c(i,j)$，各辺に流すフロー上界値の非負整数容量 $u(i,j)$ 及び始点から終点に流す最大フロー k が与えられたとき，流量が k でありかつ，各辺に流れるフロー $f(i,j)$ とコスト $c(i,j)$ の積の総和で求められる $z(f)$ が最小であるフロー f を求めよ」と定義される．本問題は輸送計画問題などに広く応用され，本問題を高速に解くことは重要である．本稿では，最小費用流問題における既存解法アルゴリズムの性能を計算機実験により評価を行い，密なグラフでは RELAX が，疎なグラフでは NETFLO が最も高速な解法であることを示す．

Experimental Evaluation of Minimum Cost Flow Algorithms

Toshifumi Keiyuu*,　Daisuke Takafuji*,　Satoshi Taoka*,　and　Toshimasa Watanabe*

*Graduate School of Engineering,　Hiroshima University
1-4-1,　Kagamiyama,　Higashi-Hiroshima,　739-8527 Japan
Phone : +81-82-424-7662 (Watanabe),　-7666 (Taoka)
Facsimile : +81-82-422-7028
E-mail : {keiyu,　daisuke,　taoka,　watanabe}@infonets.hiroshima-u.ac.jp

[abstract] The minimum cost flow problem is defined by "Given a directed network $G(V,E)$ with a nonnegative integer cost $c(i,j)$ and a nonnegative integer capacity $u(i,j)$ associated with every arc $(i,j) \in E$, and a specified pair of vertices (a source $s \in V$ and a sink $t \in V$) and having pair of the maximum flow value k, find a flow f of value k from s to t such that $z(x)$ is minimum, where $z(f)$ is the sum of the product $f(i,j) \bullet c(i,j)$ over all arcs (i,j) of a flow." It is well recognized that the problem has wide application, such as the transportation problem for example, and devising efficient algorithms has been investigated so far. This paper compares performance of existing seven well-known algorithms haved on results by computer experiment. It is shown that RELAX or NETFLO gives the highest performance for dense networks or sparse over, respectively, among them.

1　はじめに

最小費用流問題とは，V を点集合，E を枝集合とする単純有向グラフ $G(V,E)$ における，各辺 $(i,j) \in E$ に対する単位フローあたりの非負整数コストを $c(i,j)$，各辺に流すフロー上界値の非負整数容量を $u(i,j)$ とする．各辺 $(i,j) \in E$ に流れる流量を $f(i,j)$ とし，始点から終点に流す最大フロー k が与えられたとき，流量が k であり，各辺に流れるフロー $f(i,j)$ とコスト $c(i,j)$ の積の総和で求められる $z(x)$ が最小であるフロー f を求める問題である．

本問題は輸送計画問題などに広く応用され，本問題を高速に解くことは重要である．これまでに，[17] において，RELAX[8], NETFLO[4], CS[6], RNET[5] が実験比較されており，CS[6] が最も有用であることが示されている．

本研究では，これらの論文で比較されている RELAX[8], NETFLO[4], CS[6] に加えて，比較されていない既存解法についても性能評価を行った．実験を行った解法については，表 1 から 3 の *Implimentation* の欄に ◦ の印をつけて記す．本研究の目的は，これら 7 つの解法に対して計算機実験により性能評価を行い，どの解法が高速か示すことである．但し，NETFLO[4]，CS[6] と RELAX[8] は，公開しているプログラムを使用した．

実験結果として，疎な入力グラフでは NETFLO[4],

密な入力グラフでは RELAX[8] が高速であることがわかった．

今後の課題として，ベンチマークなど様々なデータにおける追加実験，未実装解法の実装及び，実装済の解法との性能比較などが挙げられる．

表 1: *augmentation* に基づく既存解法．但し，n は点数，m は辺数，C はコストの最大値，U は容量の最大値

文献	計算時間	Implementation
[1]	$O((nU)(m+n\log n))$	○ (SSP)
[8]	$O(nm^2CU^2)$	○ (RELAX)

表 2: *cancelling* に基づく既存解法．但し，n は点数，m は辺数，C はコストの最大値，U は容量の最大値

文献	計算時間	Implementation
[2]	$O(nm^2CU)$	○ (CC)
[4]	$O(n^2m^2\log n)$	○ (NETFLO)
[9]	$O(n^2m^2\log(nC))$	○ (MMC)
[11]	$O(nm+n^2\log n(nm\log(nC)))$	×
[12]	$O(m^2\log n(m+n\log n))$	×

表 3: *scaling* に基づく既存解法．但し，n は点数，m は辺数，C はコストの最大値，U は容量の最大値

文献	計算時間	Implementation
[3]	$O((m\log U)(m+n\log n))$	○ (CAPS)
[6]	$O(n^2m\log(nC))$	○ (CS)
[7]	$O((m\log n)(m+n\log n))$	×
[10]	$O(nm\log U\log(nC))$	×

2 諸定義

V を点集合，E を枝集合とする有向グラフ $G(V,E)$ において，$|V|=n, |E|=m$ と仮定する．始点 s と終点 t は V の特別な点である．各辺 $(i,j)\in E$ に対する単位フローあたりの非負整数コストを $c(i,j)$，各辺に流すフロー上界値の非負整数容量を $u(i,j)$ とする．$G(V,E)$ において G のフロー f とは，V の各点対に対して定義された次の性質を持つ．

1. 歪対称性：
各点対 (v,w) に対して，$f(v,w)=-f(w,v)$ である．
2. 容量条件：
各点対 (v,w) に対して，$0\le f(v,w)\le u(v,w)$ である．
3. フロー保存則：
始点 s と終点 t 以外の全ての点において，$\sum_w f(w,v)=0$ つまり，全ての $v\neq s,t\in V$ について，v に入ってくるフローの流量と v からでていくフローの流量が等しいことを示す．

$IN_f(v)=\sum_{w:(w,v)\in E} f(w,v)$，
$OUT_f(v)=\sum_{w:(v,w)\in E} f(v,w)$ とする．すると，$0=\sum_w f(w,v)$ より，$IN_f(v)=OUT_f(v)$ となる．フロー f の流量を始点 s からの流出量，$OUT_f(s)=\sum_{(s,w)\in E} f(s,w)$ とし，$|f|$ で表す．

また，3 のフロー保存則を緩めて，下記 3′ のように，各点 $v\neq s,t$ への流入量が流出量よりも多くてもよいとしたものをプリフローと呼ぶ．

3′. フロー残存則：始点 s と終点 t 以外の全ての点において，$\sum_w f(w,v)\ge 0$.

$v\in V$ におけるプリフロー f の残存量 (*excess*) $e(v)$ を $e(v)=IN_f(v)-OUT_f(v)$ とする．定義より，フローはプリフローである．プリフロー f が与えられたとき，点対 (v,w) の f に関する残余容量とは，$u_f(v,w)=u(v,w)-f(v,w)$ であり，$u_f(v,w)\ge 0$ である．このとき，$f(v,w)=u(v,w)$ なる点対 (v,w) は飽和しているという．また，G の f に関する残余グラフ $G_f=(V,E_f)$ とは，$E_f=\{(v,w)|u_f(v,w)>0\}$ と定義する．枝 $(v,w)\in E$ に対応する枝 $(w,v)\in E_f$ のコスト $c(w,v)$ は $-c(v,w)$ とする．また，全ての枝の流量がゼロであるフローをゼロフローと呼ぶ．このとき，次の定理は明らかである．

定理 **1:[**負閉路最適基準**][14]**
実行可能流 f は，その残余グラフ G_f がコスト c に関する負閉路を持たないとき，かつそのときに限り，最適 (最小費用流) である．

残余グラフ G_f における辺 (v,w) の *reduced cost* $c^\pi(v,w)$ を $c(v,w)-\pi(v)+\pi(w)$ とする．このとき，任意の π に関して，閉路 C のコストは変わらない (すなわち，$\sum_{(v,w)\in C} c(v,w)$ $=\sum_{(v,w)\in C} c^\pi(v,w)$) ので，定理 1 は，任意の *reduced cost* について成り立つ．また π をポテンシャルと呼ぶ．そして，この *reduced cost* に関して次の最適性の条件が知られている．

定理 **2:[reduced cost** 最適基準**][14]**
実行可能流 f は，ある点のポテンシャル π があって，π に関する *reduced cost* が次の条件を満たすとき，かつそのときに限り，最適である．
• 残余グラフ G_f における全ての枝 $(v,w)\in E_f$ に対して，$c^\pi\ge 0$ である．

さらに，線形計画法の’主問題と双対問題の対応する変数のどちらかは 0 である’という相補条件に関連して，次の最適性基準が得られる．

定理 **3:[**相補性最適基準**][14]**
実行可能流 f は，ある点のポテンシャル関数 π があって，全ての枝 $(v,w)\in E$ に対して，次の相補条件が満たされるとき，かつそのときに限り，最適である．
(i) もし，$c^\pi(v,w)>0$ ならば，$f(v,w)=0$ である．
(ii) もし，$c^\pi(v,w)<0$ ならば，$f(v,w)=u(v,w)$ である．
(iii) もし，$0<f(v,w)<u(v,w)$ ならば，$c^\pi(v,w)=0$ である．

この相補性最適基準は、キルター図 (図 1) によって、わかりやすく表現することができる．キルター図は、枝 (v,w) の状況を二次元座標 $(f(v,w),c^\pi(v,w))$ で表したものである．このとき，図 1 の太い実線が，相補性条件を満たす枝の状態である．

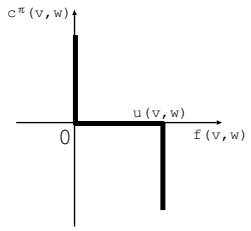

図 1: 枝 (v,w) のキルター図

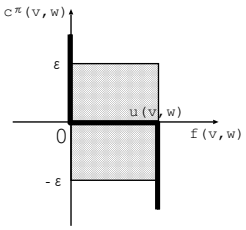

図 2: 枝 (v,w) のキルター図 ($\epsilon-$ 最適性)

次に，定理 2[14] に対応する $\epsilon-$ 最適性を定義する．但し，$\epsilon > 0$ とする．

[$\epsilon-$ 最適性][14]

実行可能流 f は，ある点のポテンシャル π があって，次の条件を満たすとき，$\epsilon-$ 最適である．

• 残余グラフ G_f における全ての枝 $(v,w) \in E_f$ に対して，$c^\pi(v,w) \geq -\epsilon$ である．

疑似フローを，容量条件を満たすがフロー保存則を必ずしも満たさないものと定義する．疑似フローに対しても，上と同様に $\epsilon-$ 最適性を定義すると，$\epsilon = 0$ のときは，定理 2[14] の *reduced cost* 最適基準になる．この $\epsilon-$ 最適性から，定理 3 の相補性最適基準に対応する次の $\epsilon-$ 相補性最適基準が得られる．

[$\epsilon-$ 相補性最適性][14]

実行可能流 f は，ある点のポテンシャル π があって，残余グラフ G_f における全ての枝 $(v,w) \in E_f$ が次の条件を満たすとき，$\epsilon-$ 最適性である．

(i) もし，$c^\pi(v,w) > \epsilon$ ならば，$f(v,w) = 0$ である．

(ii) もし，$c^\pi(v,w) > -\epsilon$ ならば，$f(v,w) = u(v,w)$ である．

(iii) もし，$0 < f(v,w) < u(v,w)$ ならば，$-\epsilon \leq c^\pi(v,w) \leq \epsilon$ である．

図 2 の網掛け部分と太い実線部分は，$\epsilon-$ 最適な枝の状態を示している．このとき，次の定理は $\epsilon-$ 最適性を利用する上で重要である．

定理 4[14]

もし，全てのコストが整数で $\epsilon < 1/n$ ならば，

$\epsilon-$ 最適なフロー f は，最小費用流である．

3 基本アイデアの説明

既存解法は，*augmentation, cancelling, scaling* の基本アイデアからなる解法に大きく分けられる．*augmentation, cancelling* においては，アイデアの中で最も基本的な解法である SSP[1]，CC[2] のアルゴリズムについて概説する．また，*scaling* においては，容量によって辺を限定して段階的に *augmentation* の操作を繰り返すアルゴリズム CAPS[3] と，最大流を求める *Preflow-Push algorithm* アルゴリズム [13] を最小費用流問題に拡張したアルゴリズム CS[6] の二つについて概説する．

最小費用流を求めるためには，(1) 実行可能流であることと (2)*reduced cost* 最適基準 (定理 2)，あるいは，相補性最適基準を満たすフローを見つける必要がある．

3.1 SSP[1]

この操作は，定理 2[14] を不変性として，最後に実行可能流を求める方法である．*augmentation* の基本アイデアは，ゼロフローとゼロポテンシャルから始めて，次々に始点からの最短パス P に沿って，フローを流し，ポテンシャルを *reduced cost* 最適基準を満たすように更新していく．また，残余グラフ $G(f)$ 上の最短パス P が与えられたとき，最短パス P の残余容量 $res_f(P)$ をその $\min\{res_f(e)|e \in P\}$ とする．

1. 任意の $i \in V$ に対し，$\pi(i) \leftarrow 0$, $f \leftarrow 0$ とする．
2. フローの値が最大フロー k になるまで以下の 3. から 5. を繰り返す．
3. 任意の $i \in V$ に対し，$\pi(i) \leftarrow \pi(i) - d(i)$ とする．但し，$d(i)$ は各辺のコストを距離と考えたときの始点から点 $i \in V$ への最短パスである．
4. 現在のフロー f の残余グラフ $G(f)$ において，各辺 (i,j) に対し，$c^\pi(i,j) \leftarrow c(i,j) - \pi(i) + \pi(j)$ とする．
5. 始点から終点への最短パス P に対し，パス上の辺のフローを x' だけ増やす．但し，$x' = \min\{k - |f|, res_f(P)\}$ である．

これによって得られたフロー k が最小費用流である．定理 2 [14] より、この解法の正当性が成り立つ．

3.2 CC[2]

この操作は，*augmentation* とは逆に，実行可能流であることを不変性として，フローのコストを下げていく方法である．*cancelling* の基本アイデアは，フロー f に関する残余グラフにおいて，辺のコスト総和が負になる有向サイクル (負閉路) がある限り，この閉路に沿ってフローを流せるだけ流し，最小費用流を求める．

1. 与えられた最大フロー k だけ始点から終点に流す．
2. 残余グラフ $G(k)$ で負閉路 C を探す．負閉路 C が存在する間，3. を繰り返す．
3. 負閉路 C に沿って $res_k(C)$ だけフローを流すことにより，その負閉路を消去する．

これにより得られたフロー k が最小費用流である．定理 1 [14] より，この解法の正当性が成り立つ．

3.3 CAPS[3]

この操作は，*augmentation* の拡張ともいえる．辺を限定して *augmentation* を行う方法である．

1. 任意の $i \in V$ に対し，$\pi(i) \leftarrow 0$, $f \leftarrow 0$, $\Delta \leftarrow 2^{\lfloor \log U \rfloor}$ とする．
2. $\Delta \geq 1$ のとき，以下の 3. から 5. を繰り返す．
3. 現在のフロー x の残余グラフ $G(x)$ において，容量が Δ より大きい辺のみで構成された残余グラフ $G(x, \Delta)$ を構成する．
4. この残余グラフ $G(x, \Delta)$ に対して SSP[1] と同様の操作を行いフローを求める．
5. 4. 終了後，Δ において同様の操作が行えるなら，4. へ，行えないなら，6. へ行く．
6. $\Delta \leftarrow \Delta/2$ とする．

これによって得られたフロー k が最小費用流である．定理 2 [14] より，この解法の正当性が成り立つ．

3.4 CS[6]

この操作は，最大流を求める *Preflow-Push algorithm*[13] の一般化であり，ϵ を段階的に減らしながら，各段階で
ϵ– 最適なフローとポテンシャルを求めて，$\epsilon = 0$ になるまで続ける操作である．

1. 任意の $i \in V$ に対し，$\pi \leftarrow 0$, $f \leftarrow 0$,$\epsilon \leftarrow C$ とする．
2. $\epsilon \geq 1$ のとき，以下の (3) から (7) を繰り返す．
3. $c^{\pi}(i,j) > 0$ ならば $f(i,j) = 0$, $c^{\pi}(i,j) < 0$ ならば $f(i,j) = u(i,j)$ とする．
4. $e(i) > 0$ となる点 (*active node*) i が存在しなくなるまで 5. と 6. を繰り返す．
5. $active\ \ node$ i を始点とし，辺のコストが $-1/2\epsilon \leq c^{\pi}(i,j) < 0$ の条件を満たす辺 (*admissible arc*)(i,j) に $\min\{e(i), u(i,j)\}$ だけフローを流す．
6. *active node* i が存在するが，*admissible arc* が存在しないとき，$\pi(i) \leftarrow \pi(i) + \epsilon/2$ とする．
7. $\epsilon = \lfloor \epsilon/2 \rfloor$ とする．

これによって得られたフロー k が最小費用流である．アルゴリズム中では $c^{\pi}(i,j) \geq -\epsilon/2$ が成り立つ [6] ので，定理 2 [6] より，この解法の正当性が成り立つ．

4 解法の分類

表 1 から 3 より，現在までのところ解法の主な基本アイデアは，*augmentation, cancelling* と *scaling* であることが知られている．

augmentation の SSP[1] は，始点から終点の最短経路を求め，その経路にフローを流すという操作を最大フロー k だけ流れるまで繰り返すアルゴリズムである．RELAX[8] は，*relaxation algorithm*[8] を使った解法である．

cancelling の CC[2] は，始点から終点に k だけ流れている残余グラフにおいて，負のコストを持った閉路（これを負閉路と呼ぶ）がある限り，この閉路に沿ってフローを流せるだけ流すという操作を繰り返すアルゴリズムである．NETFLO[4] は，この基本アイデアをシンプレックス法で解いたものである．また，この解法におけるオーダは $O(n^2m^2logn)$ であることが [15],[16] によって示されている．MMC[9] は，負閉路の消去を効率的に行うために CC[2] を改良したものである．[11] は，MMC[9] に *augmentation* のアイデアを取り入れたものである．[12] は，基本アイデアに *scaling* のアイデアを取り入れたものである．

scaling の CAPS[3] は，容量によって辺を限定して段階的に *augmentation* の操作を繰り返すアルゴリズムであり，*capacity-scaling algorithm* と呼ぶ．CS[6] は，最大流を求める *Preflow-Push algorithm* アルゴリズム [13] を最小費用流問題に拡張したアルゴリズムであり，*cost-scaling algorithm* と呼ぶ．[7] は，CAPS[3] のアルゴリズムを改良したものである．[10] は，CAPS[3] と CS[6] の考え方を両方使うアルゴリズムである．

5 実験概要と結果

5.1 実験概要

表 1 から 3 より，既存解法 SSP[1]，CC[2]，CAPS[3]，NETFLO[4]，CS[6]，RELAX[8] について計算機 (CPU:Pentium4 1.7GHz, OS:FreeBSD 5.4) 上で SSP[1]，CC[2]，CAPS[3]，CS[6] は C 言語，NETFLO[4]，RELAX[8] は FORTRAN を用いて実装した．

NETFLO[4]，CS[6] 及び RELAX[8] は，それぞれ [18]，[19]，[20] で公開されているプログラムを使用した．但し，k を最大フローの値とし，最大フロー値を求める時間は計算時間に含まない．

5.2 データ生成

総点数，総辺数，最大コスト，最大容量を入力とし，以下のアルゴリズムに従って有向グラフを生成する．

1. 点に番号 (1, 2, ... , n) を付与する．始点を 1，終点を n とし，始点には出辺，終点には入辺の

み辺を付加する．また，各辺の容量，コストはランダムに与えられた数字を付加する．

2. 1 から $n-1$ 番目の各点を出発点とし，ランダムに選んだ自分の番号より大きい点を到着点とする辺を一つずつ付加する．

3. 2 から n 番目の各点を到着点とし，ランダムに選んだ自分の番号より小さい点を出発点とする辺が存在しないならば付加する．

4. 総辺数の辺を付加するまで，1 から $n-1$ 番目の各点を出発点とし，ランダムに選んだ自分の番号より大きい点を到着点とする辺を付加する．

上記の生成方法により，次のように生成したデータを合計 2190 個用意した．但し，n を点数，m を辺数，C をコストの最大値，U を容量の最大値とする．

- 各 $n \in \{100, 200, \ldots, 900\} \cup \{1000, 2000, \ldots, 10000\}$ において，$m = 2n$, $U = 100$, $C = 100$ なるデータ 30 個
- 各 $m \in \{10^4, 2 \times 10^4, \ldots, 10 \times 10^4\}$ において，$n = 1000$，$U = 100$，$C = 100$ なるデータ 30 個
- 各 $n \in \{100, 200, 300\}$ と辺密度が 10%, 20%, ... , 90% となる m の全ての組み合わせにおいて，$U = 100$，$C = 100$ なるデータ 30 個
- 各 $n \in \{300, 400, 500\}$ において，$m = 11940$ ($n = 200$ において辺密度 60% となる辺数)，$U = 100$，$C = 100$ なるデータ 30 個
- 各 $C, U \in \{10, 100, 1000, 10000\}$ となる全ての組み合わせにおいて，$U = 100$, $C = 100$, $n = 1000$, $m = 2n$ または，辺密度が 60% となる m それぞれのデータ 30 個

但し，辺密度 d とは，頂点数 n の完全グラフの辺数 $m' = (n(n-1))/2$ に対し，$d = (m/m') \times 100(\%)$ なる値である．

5.3 結果と考察

5.3.1 疎なグラフに関する実験結果

疎なグラフ ($d < 50\%$) に対する実験結果を表 4 から 7 及び図 3 から 6 に示す．計算時間に影響を与えるパラメータ n, m, U, C のうち三つを固定し，残りの一つのパラメータ値を増減させて比較を行った．

表中は最小値/平均値/最大値を表し，単位は秒とする．'−' は 10 時間以内に全てのグラフで結果が得られなかったことを示す．

表 4: m=10000, U=100, C=100 を固定して各 $n \in \{1000, 2000, 3000, 4000\}$ それぞれ 30 個の計算時間（秒）

手法	n=1000	n=2000
	n=3000	n=4000
SSP	0.453/ 1.017/ 1.867	1.258/ 2.048/ 2.805
	1.359/ 2.605/ 4.031	2.898/ 4.137/ 5.859
CC	-	-
	-	-
CAPS	0.766/ 1.708/ 3.734	1.461/ 2.316/ 3.164
	1.586/ 2.838/ 4.398	3.258/ 4.456/ 6.156
NETFLO	0.016/ 0.021/ 0.027	0.021/ 0.029/ 0.039
	0.013/ **0.021/ 0.035**	**0.014/ 0.020/ 0.039**
CS	0.023/ 0.027/ 0.039	0.023/ 0.032/ 0.047
	0.031/ 0.038/ 0.055	0.039/ 0.049/ 0.063
RELAX	**0.007/ 0.014/ 0.021**	**0.014/ 0.023/ 0.031**
	0.012/ 0.032/ 0.077	0.024/ 0.053/ 0.102
MMC	0.187/ 0.219/ 0.304	0.220/ 0.259/ 0.362
	0.134/ 0.251/ 0.373	0.150/ 0.220/ 0.374

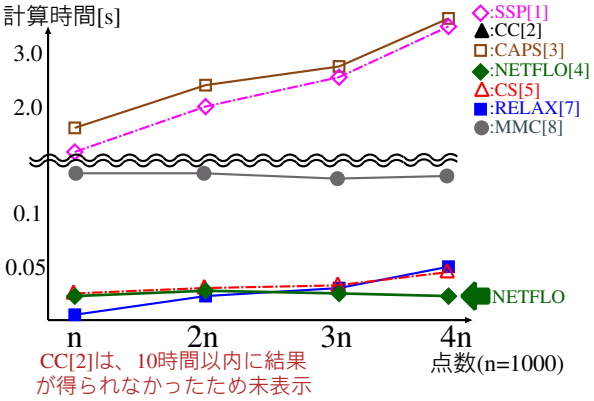

図 3: m=10000, U=100, C=100 を固定して点数 n を変化させた場合の平均計算時間

5.3.2 密なグラフに関する実験結果

密なグラフ ($d \geq 50\%$) に対する実験結果を表 8 から 11 及び図 7 から 10 に示し，疎なグラフと同様の比較を行った．

5.3.3 考察

本実験結果より以下のことがわかった．

表 5: n=1000, U=100, C=100 を固定して各 $m \in \{4000, 6000, 8000, 10000\}$ それぞれ 30 個の計算時間（秒）

手法	m=4000	m=6000
	m=8000	m=10000
SSP	0.141/ 0.391/ 0.836	0.195/ 0.697/ 1.258
	0.398/ 0.707/ 1.180	0.453/ 1.017/ 1.867
CC	176.9/ 452.2/ 1172	1627/ 3327/ 6385
	4608/ 8328/ 12337	-
CAPS	0.156/ 0.418/ 0.906	0.266/ 0.815/ 1.445
	0.555/ 0.909/ 1.453	0.766/ 1.708/ 3.734
NETFLO	0.005/ 0.008/ **0.010**	0.007/ 0.013/ 0.019
	0.010/ 0.016/ 0.024	0.016/ 0.021/ 0.027
CS	0.007/ 0.008/ **0.010**	0.011/ 0.014/ **0.018**
	0.015/ 0.019/ 0.021	0.023/ 0.027/ 0.039
RELAX	**0.003/ 0.006**/ 0.016	**0.004/ 0.008**/ 0.023
	0.006/ 0.008/ 0.010	**0.007/ 0.014/ 0.021**
MMC	0.027/ 0.041/ 0.074	0.065/ 0.100/ 0.146
	0.126/ 0.143/ 0.196	0.187/ 0.219/ 0.304

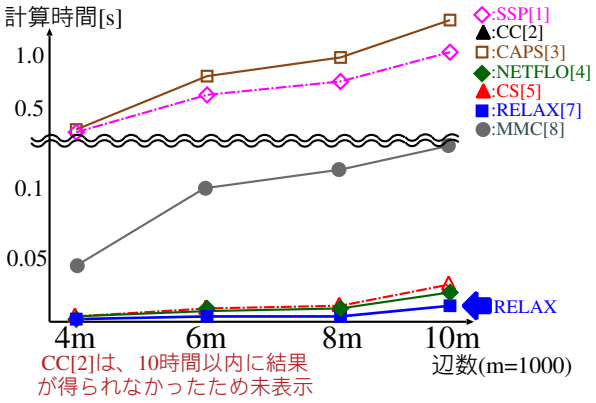

図 4: n=1000, U=100, C=100 を固定して辺数 m を変化させた場合の平均計算時間

表 6: n=1000, m=2000, C=100 を固定して各 $U \in \{10, 100, 1000, 10000\}$ それぞれ 30 個の計算時間 (秒)

手法	U=10 U=1000	U=100 U=10000
SSP	0.070/ 0.168/ 0.141	0.086/ 0.192/ 0.297
	0.094/ 0.201/ 0.430	0.086/ 0.192/ 0.266
CC	3.458/ 17.81/ 43.86	3.265/ 46.55/ 101.5
	3.179/ 35.55/ 101.7	7.046/ 45.23/ 124.6
CAPS	0.070/ 0.099/ 0.148	0.094/ 0.196/ 0.313
	0.102/ 0.253/ 0.539	0.117/ 0.196/ 0.297
NETFLO	**0.001/ 0.002/ 0.003**	0.002/ **0.003/ 0.004**
	0.002/ 0.003/ **0.005**	**0.002/ 0.003/ 0.004**
CS	0.004/ 0.005/ 0.006	0.003/ 0.005/ 0.007
	0.000/ 0.002/ 0.007	**0.000**/ 0.005/ 0.006
RELAX	0.002/ 0.004/ 0.010	**0.002**/ 0.005/ 0.007
	0.002/ 0.005/ 0.013	0.003/ 0.009/ 0.012
MMC	0.006/ 0.010/ 0.013	0.007/ 0.017/ 0.022
	0.009/ 0.016/ 0.022	0.010/ 0.016/ 0.023

(i) 疎な入力グラフでは NETFLO[4]，密な入力グラフでは RELAX[8] が高速である．

NETFLO[4] は，負閉路を除去する操作を繰り返すアルゴリズムということから，辺数が大きくなると負閉路を探すのに時間がかかり，計算時間に大きな影響を与える．そのため，疎なグラフに対しては高速であるが，密なグラフに対しては RELAX[8] の方が高速である．また，図 10 より最大費用 C= {1000, 10000} のとき密なグラフにおいて NETFLO[4] の方が RELAX[8] よりも高速になっている．原因については，厳密な理由がわかっていないため，これから考える必要がある．

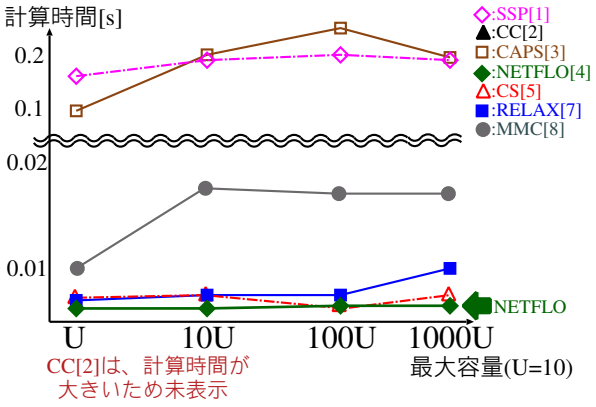

図 5: n=1000, m=2000, C=100 を固定して最大容量 U を変化させた場合の平均計算時間

表 7: n=1000, m=2000, U=100 を固定して各 $C \in \{10, 100, 1000, 10000\}$ それぞれ 30 個の計算時間 (秒)

手法	C=10 C=1000	C=100 C=10000
SSP	0.031/ 0.180/ 0.211	0.086/ 0.192/ 0.297
	0.133/ 0.203/ 0.305	0.125/ 0.199/ 0.266
CC	2.169/ 9.899/ 24.43	3.265/ 46.55/ 101.5
	11.14/ 58.03/ 156.6	9.553/ 47.44/ 103.2
CAPS	0.047/ 0.144/ 0.242	0.094/ 0.196/ 0.313
	0.148/ 0.220/ 0.281	0.141/ 0.210/ 0.281
NETFLO	**0.001/ 0.002/ 0.002**	**0.002/ 0.003/ 0.004**
	0.002/ 0.003/ 0.004	**0.002/ 0.003/ 0.004**
CS	0.002/ 0.005/ 0.006	0.003/ 0.005/ 0.007
	0.005/ 0.006/ 0.007	0.006/ 0.007/ 0.007
RELAX	**0.001/ 0.002**/ 0.003	**0.002**/ 0.005/ 0.007
	0.003/ 0.007/ 0.017	0.005/ 0.010/ 0.023
MMC	0.005/ 0.010/ 0.016	0.007/ 0.017/ 0.022
	0.013/ 0.019/ 0.030	0.008/ 0.015/ 0.024

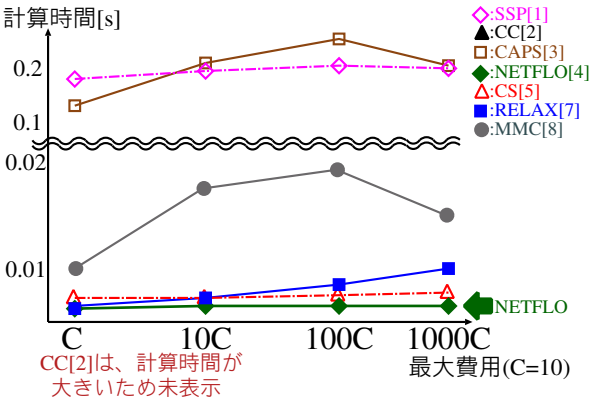

図 6: n=1000, m=2000, U=100 を固定して最大コスト C を変化させた場合の平均計算時間

(ii) SSP[1]，CAPS[3] において 4 つのパラメータの

表 8: m=11940, U=100, C=100 を固定して各 $n \in \{200, 300, 400, 500\}$ それぞれ 30 個の計算時間 (秒)

手法	n=200 n=400	n=300 n=500
SSP	2.000/ 2.563/ 3.164	1.212/ 1.656/ 2.195
	0.820/ 1.179/ 1.783	0.732/ 1.208/ 1.689
CC	-	-
	-	-
CAPS	3.766/ 10.68/ 20.23	2.531/ 5.352/ 13.11
	1.814/ 3.878/ 7.701	1.259/ 2.438/ 4.104
NETFLO	0.020/ 0.027/ 0.033	0.018/ 0.022/ 0.028
	0.016/ 0.020/ 0.030	0.016/ 0.021/ 0.024
CS	0.023/ 0.030/ 0.031	0.028/ 0.031/ 0.033
	0.024/ 0.029/ 0.034	0.028/ 0.030/ 0.032
RELAX	**0.014/ 0.018/ 0.021**	**0.011/ 0.015/ 0.019**
	0.015/ 0.017/ 0.021	**0.015/ 0.018/ 0.022**
MMC	0.248/ 0.279/ 0.313	0.266/ 0.303/ 0.348
	0.270/ 0.290/ 0.319	0.251/ 0.301/ 0.363

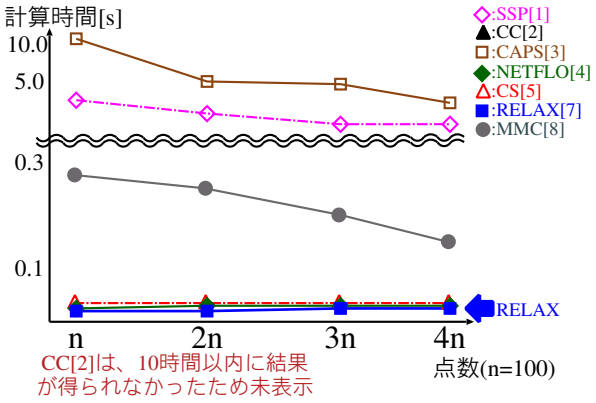

図 7: m=11940(点数 200 における辺密度 60% の辺数), U=100, C=100 を固定して点数 n を変化させた場合の平均計算時間

表 9: n=200, U=100, C=100 を固定して各辺密度が 50%, 60%, 70%, 80% それぞれ 30 個の計算時間 (秒)

手法	d=50 d=70	d=60 d=80
SSP	1.406/ 1.662/ 1.898 3.594/ 4.180/ 4.922	2.000/ 2.563/ 3.164 5.734/ 6.052/ 6.375
CC	- -	- -
CAPS	2.758/ 6.277/ 15.344 7.164/ 20.67/ 63.96	3.766/ 10.68/ 20.23 10.141/ 42.05/ 168.6
NETFLO	0.019/ 0.022/ 0.026 0.031/ 0.037/ 0.045	0.020/ 0.027/ 0.033 0.046/ 0.049/ 0.053
CS	0.023/ 0.024/ 0.031 0.031/ 0.037/ 0.039	0.023/ 0.030/ 0.031 0.039/ 0.045/ **0.047**
RELAX	**0.009/ 0.012/ 0.014** **0.016/ 0.023/ 0.029**	**0.014/ 0.018/ 0.021** **0.025/ 0.034**/ 0.061
MMC	0.206/ 0.227/ 0.259 0.280/ 0.415/ 0.490	0.248/ 0.254/ 0.313 0.334/ 0.424/ 0.682

中で点数 n を増加させた場合，疎な入力グラフでは計算時間が大きくなり，密なグラフでは小さくなる．

SSP[1] と CAPS[3] のアルゴリズムはどちらも最短パスにフローを流す操作を繰り返すアルゴリズムである．図 7 より，密なグラフにおいて点数に対して辺数が大きいとき最短パスを求める回数が多くなることから，計算時間が遅くなる．しかし，図 3 より，疎なグラフにおいては点数に対して辺数が大きいとき，計算時間が速くなる．この原因としては，予備実験により 1 つの最短パスを求める時間が短いという結果が得られたことが挙げられる．

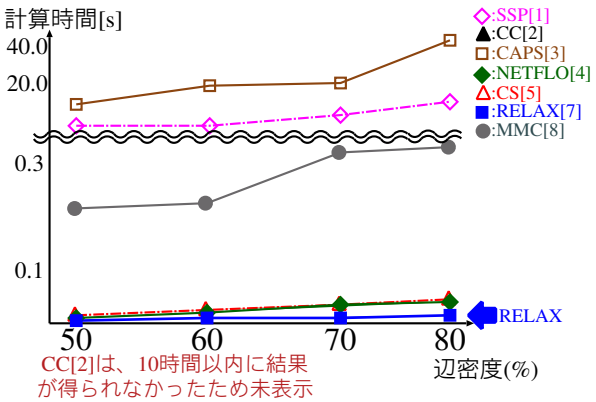

図 8: n=200, U=100, C=100 を固定して辺数 m を変化させた場合の平均計算時間

表 10: n=200, m=11940, C=100 を固定して各 $U \in \{10, 100, 1000, 10000\}$ それぞれ 30 個の計算時間 (秒)

手法	U=10 U=1000	U=100 U=10000
SSP	1.196/ 1.460/ 1.712 2.118/ 3.044/ 4.219	2.233/ 2.816/ 3.902 2.229/ 3.031/ 4.246
CC	- -	- -
CAPS	2.313/ 5.703/ 16.12 4.400/ 6.726/ 11.50	4.612/ 6.201/ 13.57 4.479/ 6.893/ 19.60
NETFLO	0.020/ 0.026/ 0.032 0.025/ 0.030/ **0.036**	0.023/ 0.030/ 0.037 0.024/ 0.033/ 0.084
CS	0.025/ 0.028/ 0.031 0.029/ 0.032/ 0.039	0.028/ 0.032/ 0.036 0.027/ 0.032/ 0.036
RELAX	**0.012/ 0.017/ 0.023** **0.013/ 0.019**/ 0.047	**0.013/ 0.019/ 0.030** **0.013/ 0.019/ 0.024**
MMC	0.183/ 0.218/ 0.231 0.338/ 0.371/ 0.422	0.237/ 0.289/ 0.353 0.416/ 0.466/ 0.527

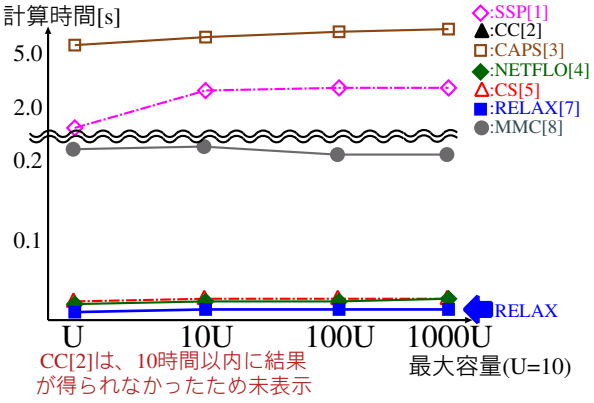

図 9: n=200, m=11940(辺密度 60%), C=100 を固定して最大容量 U を変化させた場合の平均計算時間

(iii) CS[6], MMC[9] は 4 つのパラメータの中で辺数 m を増加させた場合，最も計算時間が大きく

表 11: n=200, m=11940, U=100 を固定して各 $C \in \{10, 100, 1000, 10000\}$ それぞれ 30 個の計算時間 (秒)

手法	C=10 C=1000	C=100 C=10000
SSP	1.316/ 1.772/ 2.212 2.414/ 3.155/ 4.349	2.233/ 2.816/ 3.902 2.479/ 2.982/ 4.079
CC	- -	- -
CAPS	2.646/ 4.153/ 10.53 5.721/ 15.34/ 63.87	4.612/ 6.201/ 13.57 5.785/ 17.89/ 49.44
NETFLO	0.023/ 0.026/ 0.030 0.024/ **0.031/ 0.038**	0.023/ 0.030/ 0.037 **0.025/ 0.030/ 0.039**
CS	0.026/ 0.029/ 0.033 0.031/ 0.034/ **0.038**	0.028/ 0.032/ 0.036 0.030/ 0.035/ 0.040
RELAX	**0.007/ 0.009/ 0.014** **0.022**/ 0.033/ 0.051	**0.013/ 0.019/ 0.030** 0.028/ 0.038/ 0.052
MMC	0.178/ 0.211/ 0.237 0.265/ 0.334/ 0.461	0.237/ 0.289/ 0.353 0.223/ 0.311/ 0.365

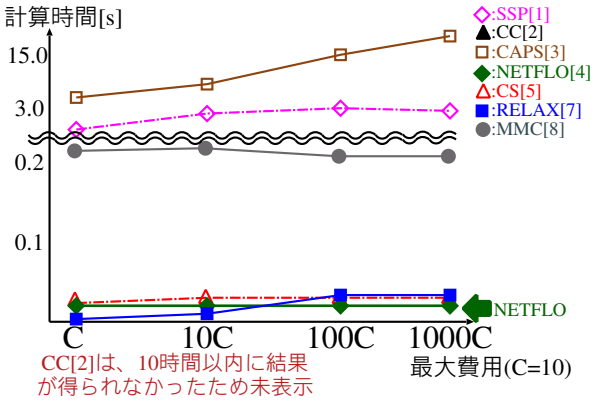

図 10: n=200, m=11940(辺密度 60%), U=100 を固定して最大コスト C を変化させた場合の平均計算時間

なる．

CS[6] はフローを流す辺を探すため，MMC[9] は負閉路を探すため，それぞれの理由でパラメータ m を増加させたとき，計算時間が大きくなる．

(iv) CC[2] の計算時間が非常に大きい．

CC[2] は，計算時間の最小値と最大値の幅が非常に大きいことからグラフによって大きな差があるといえる．今後さらに追加実験を行い，どのようなグラフに対して時間がかかっているか調べていきたい．

6 まとめと今後の課題

最小費用流問題アルゴリズムを実装し，計算機実験による性能評価を行った．計算時間に影響を与えるパラメータ n, m, U, C のうち三つを固定し，残りの一つのパラメータ値を増減させて解法の計算時間を比較した．その結果，疎な入力グラフでは NETFLO[4]，密な入力グラフでは RELAX[8] が高速であることがわかった．

今後の課題として，以下の項目を挙げる．

(1) 未実装の既存解法の実装及び計算機による比較実験

(2) 様々なデータにおける追加実験

参考文献

[1] R. Busaker and P. Gowen, "A procedure for determining minimal-cost network flow patterns," Tech. Rep. 15, ORO, 1961.

[2] M. Klein, "A primal method for minimal cost flows with application to the assignment and transportation problems," *Management Science*, vol. 14, pp. 205–220, 1967.

[3] J. Edmonds and R. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of ACM*, vol. 19, pp. 248–264, 1972.

[4] J. Kennington and R. Helgason, *Algorithms for Network Programming*, Wiley Interscience, New York, 1978.

[5] M. Grigoriadis, "An efficient implementation of the network simplex method," *Math. Prog*, vol. 26, pp. 83–111, 1986.

[6] A. Goldberg and R. Tarjan, "Solving minimum cost flow problem by successive approximation," *Proceedings of the 19th ACM Symposium on the Theory of compiting*, pp. 7–18, 1987.

[7] J. Orlin, "A faster strongly polynomial minimum cost flow algorithm," *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pp. 377–387, 1988.

[8] D. Bertsekas and P. Tseng, "Relaxation methods for minimum cost ordinary and generalized network flow problems," *Operations Research*, vol. 36, pp. 873–886, 1988.

[9] A. Goldberg and R. Tarjan, "Finding minimum-cost circulations by cancelling negative cycles," *Journal of ACM*, vol. 36, pp. 873–886, 1989.

[10] J. Orlin R. Ahuja, A. Goldberg and R. Tarjan, "Finding minimum-cost flows by double scaling," *Mathematica Programming*, vol. 53, pp. 243–266, 1992.

[11] S. Iwata M. Shigeno and S. McCormick, "Relaxed most negative cycle and most positive cut canceling algorithms for minimum cost flow," Tech. rep., University of British Columbia, Vancouver, B.C., V6T 1Z2, 1996.

[12] R. Tarjan P. Sokkalingram and J. Orlin, "New polynomial-time cycle-canceling algorithms for minimum-cost flows," *Networks.*, vol. 36, pp. 53–63, 2000.

[13] A. Goldberg and R. Tarjan, "A new approach to the maximum-flow problem," *Journal of ACM*, vol. 35, no. 4, pp. 921–940, 1988.

[14] T. Magnanti R. Ahuja and J. Orlin, *Network Flows*, Prentice Hall, 1993.

[15] J. Orlin, "A polynomial time primal network simplex algorithm for minimum cost flows," *SODA*, pp. 474–481, 1996.

[16] J. Orlin, R. Ahuja, P. Sharma and P. Sokkalingam, "A network simplex algorithm with o(n) consecutive degenerate pivots," *A journal version of this paper appeared in OR Letters 30*, pp. 141–148, 2002.

[17] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," *J. Algorithms*, vol. 22, pp. 1–29, 1997.

[18] Dimacs Implementation Challenge, `ftp://dimacs.rutgers.edu/pub/netflow/`

[19] Zuse Institute Berlin, `http://elib.zib.de/pub/Packages/mathprog/mincost/relax4.tar.Z`

[20] Andrew Goldberg's Network Optimization Library, `http://www.avglab.com/andrew/soft.html`