最大フロー問題における全最小カット特定の一方法

Rep. Fac. Sci., Kagoshima Univ.,
No. 49, pp. 1–9 (2016)

最大フロー問題における全最小カット特定の一方法

An Identification Method of All Minimum-Cuts in the Maximum-Flow Problem

新森修一[1)]・西田美博[2)]
Shuichi SHINMORI[1)], Yoshihiro NISHIDA[2)]

Abstract: In this paper, it is a purpose to identify all Minimum-cuts using the solution method of Maximum-flow problem of the graph theory. Several methods and algorithms to search for the Minimum-cut has been contrived by itself already. Based on these results, we checked a relation between the edges and minimum cuts in detail, and propose about the method to specify all Minimum-cuts which aimed at the increased operation when solving the biggest flow problem. We can get the edges included in the Minimum-cut, the pair of edges which become the Minimum-cut efficiently by using this method.

Keyword: Maximum-flow problem, Minimum-cut, Graph theory.

1. はじめに

線形計画問題の一つに最大フロー問題がある。これは始点となるある場所から終点となる別の場所へ流す資源量（フロー量）の最大値とその経路を求める問題であり，場所と経路を点（頂点）と線（辺）で表したグラフとして置き換えて考えることで解くことができる。また，グラフにはカットという概念が存在していて，最大フロー最小カット定理と呼ばれる定理から，グラフの最小カットは最大フローと密接な関係があることが知られている。このことから，最小カットは最大フロー問題においても重要な概念であると言える。

最小カットに関して，グラフの持つ幾つかの最小カットを求めるアルゴリズムは既に多くの分野において研究されているが，全ての最小カットを求めるアルゴリズムについては，それらに比べて未だ研究が進んでいない部分も多い。その一方で，全ての最小カットに含まれている辺が見つかれば，その辺1本の容量を増加させるだけで効率的にグラフの最大フローを増加させることができ，このような性質を利用してグラフの最大フロー問題における各辺の重要度を定義することも可能となり，全ての最小カットを特定する問題は最大フロー問題に関する応用の分野において応用範囲が拡大すると考えられる。そこで本論文では，与えられた最大フロー問題に対して，グラフの持つ全ての最小カットを導出する方法について考察している。

以下，2節ではグラフ理論における最大フロー問題についての基本的な説明，3節では最小カットの持つ性質，特に，最小カット辺の容量と最大フローの関係などについて述べる。4節では3節で述べた性質を踏まえた最小カット全てを求めるアルゴリズムの提案，5節では論文のまとめと今後の課題について触れている。

2. 最大フロー問題

最大フロー問題とは，グラフ理論における線形計画法の1つである。この問題の目的は，始点と終点が

1）鹿児島大学大学院理工学研究科数理情報科専攻
Graduate School of Science Engineering, Kagoshima University, Kagoshima 890-0065, Japan
2）株式会社イノス

決められたグラフにおいて，各辺に定められた容量制限を超えないことを条件にして，終点へ流せるフローの最大値を求めることである。

一方，グラフにおけるカットとは，グラフ上の特定の辺の組の中で，それらを取り除いた際に元のグラフが2つの非連結な集合 A と B に分割されるような辺の集合のことである。特に，分割後のグラフにおいて集合 A に始点 s が，もう片方の集合 B に終点 t が含まれるような分割になっている時，そのカットはグラフ上の s-t カットであると言う。また，カットとなる組について，各辺の容量の総和をそのカットにおける容量と言い，特に，グラフに関する s-t カットの中で容量が最小のものを最小カットと言う。

最大フローと最小カットには，「グラフの最大フロー時の容量が，グラフの持つ最小カットの容量と一致する」という関係があることが知られていて，このことはグラフ理論における最大フロー最小カット定理と呼ばれている。

与えられたグラフの最大フローを特定する方法については，増加操作という手法を用いて全体のフローを少しずつ増加させることでフローの最大値を求める Ford-Fulkerson アルゴリズム [1,2] や，それをさらに改良して実行時間を短縮させた Edmonds-Karp アルゴリズム [3] といったものが知られている。増加操作の具体的な手順は，まず始点から終点へ辺を辿って到達可能な道筋（増加パス）を見つけ，そのパス上の各辺に対して流せるフロー量の中で最小の値を計算し，その値と同じだけパス上の全ての辺にフローを流す，という流れになる。このとき，パス上の各辺の中で流せるフロー量が最小の値を持つような辺のことを増加パスにおけるボトルネック辺と言う。ボトルネック辺は増加操作によってフローを流すことで辺に流せるフロー量が0となり，このようにもうフローを流せない辺のことを辺が飽和していると言う。飽和した辺は以降の増加パスで選ばれることは無いため，この事実が増加操作の繰返しによって最大フローを求められる根拠の1つになっている。

3. 最小カットの性質

本論文においては，増加パスでボトルネックとなる辺と最小カットに含まれる辺との関係や，増加操作によるフロー増加量の合計と最小カットの容量との関係等に着目しており，この節ではそれらの具体的な内容をまとめている。

3.1 増加パスにおけるボトルネック辺と最小カット辺との関係

最小カットに含まれる辺はグラフ全体におけるボトルネックであるため，増加操作においてどんな順番でパスを選ぼうとも最大フロー時には必ず飽和している。一方で，増加操作の際に増加パスにおけるボトルネック辺は全て飽和することになるが，そこでボトルネックとなる辺はパスの選択方法によって変わることも考えられるため，ボトルネック辺として選ばれて飽和した辺が必ず最小カットに含まれているとは限らない。

3.2 最小カット辺の容量増減と最大フローとの関係

グラフ上の辺と最小カットとの関係は以下の3パターンに分類できる。

①辺が全ての最小カットに含まれていない場合

②辺が全ての最小カットに含まれている場合

③辺がある最小カットには含まれているが，別の最小カットには含まれていない場合

これらの例を図示したものが図1である。図中，矢印がある注目している辺であり，破線が対象としてるグラフにおける最小カットを表している。例えば，①では注目してる辺がすべての最小カットに含まれていないことを表している。

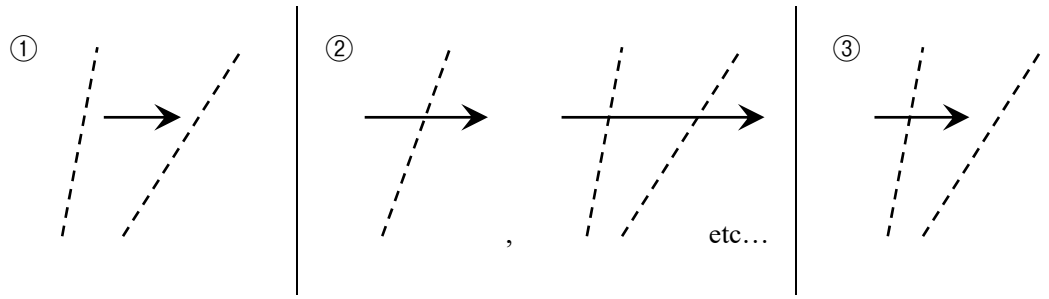

図1　グラフ上での辺と最小カットの関係の例（矢印が辺，破線が最小カット）

任意の辺がこの中のどのパターンに属しているかを調べるためには，最小カット辺の容量と最大フローの関係が利用できる。そして，この方法を用いることでボトルネック辺の中から最小カット辺を特定することも可能となる。

まず，最小カットに含まれる辺はグラフ全体におけるボトルネックであるため，その辺の容量が変わるとグラフにおける最大フロー量も変化する。具体的には，最小カットに含まれる辺の容量を減らすと，全体としての制約もそれだけ厳しくなるため最大フローも減少する。これを踏まえ，今回提案するアルゴリズムでは，各辺の容量を減少させた上で再度最大フローを計算し，その結果が減少前と変化しているかどうかで，その辺が最小カットに含まれているかどうかの特定を行っている。

一方で，辺の容量を増加させて同様の判定方法を行った場合には，その辺が最小カットに含まれているかどうかの判断が常にできるわけではない。例えば，もし「その辺は最小カットに含まれているが，その辺を含まない他の最小カットが存在する」ならば，その辺の容量を増加させることでその辺が含まれている最小カットの容量は増加しても，その辺が含まれていない最小カットに関しては容量が変化しないため，全体の最大フローが変化することもない。また，もし「その辺はそもそも最小カットに含まれていない」ならば，辺容量を増加させても最大フローが変化しないのは当然である。よって，最大フローが維持された場合の原因が「その辺は最小カットに含まれているが，その辺を含まない他の最小カットが存在する」ためなのか，「その辺はそもそも最小カットに含まれていない」ためなのかの判断をすることができない。

ここで述べた辺容量の増減と最大フローとの関係について，具体的にまとめたものが次の表1である。表の1列目はある特定の辺に注目し，その辺容量を減少させた場合と増加させた場合を表し，その結果，1行目の最大フローが維持されたか変化したかの場合に区分している。この4つの区分に対し，辺が上の①から③のどのパターンに属するかをまとめた表である。

表1　辺容量増減時の最大フロー考察から判別可能な最小カットとの関係

辺容量＼最大フロー	維持	変化
減少－－	パターン① 辺は全ての最小カットに含まれていない	パターン② or ③ 辺を含む最小カットが存在する （辺を含まない最小カットが存在するかは不明）
増加＋＋	パターン① or ③ 辺を含まない最小カットが存在する （辺を含む最小カットが存在するかは不明）	パターン② 辺は全ての最小カットに含まれる

辺容量の増加だけでは最小カットに含まれるかどうかの判定は行えないが，この方法によって分かる情報もある。この方法で最大フローが増加した場合には，全体としての制約がそれだけ緩くなったことが原因であり，その辺の容量は全ての最小カットに影響を与えている。つまり，全ての最小カットに含まれていると言えるため，グラフ上でも重要度の高い辺であることが確認できる。

以上のことから，図2のように辺容量の増減操作を行うことによって，その辺と最小カットとの間にある関係を調べることができる。図2では，正方形で囲っている「減・増」は辺容量の減少，増加を表し，矢印に添えてある「維・減・増」は，最大フローの維持（変化なし），減少，増加をそれぞれ表している。その結果，①から③のどのパターンになるかを示している。また，図の右側は，①から③のパターンに判別される様子を表している。今回は最小カットに含まれているかどうかという情報だけが必要であるので，アルゴリズム上では辺容量を減らすだけで良いが，各辺と最小カットとの関係やグラフにおける辺の重要度を確認する際には，辺容量の増減を組み合わせた判定方法も1つの手段になると考えられる。

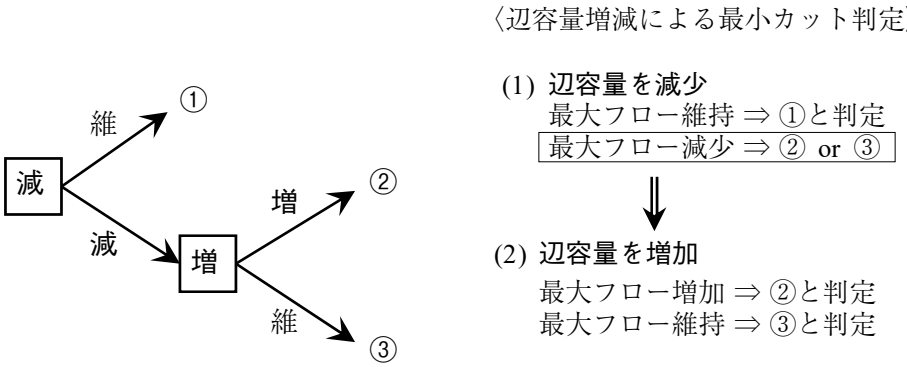

図2　辺容量増減後の最大フローに応じた最小カット判定方法

実際には，この関係を利用するだけで最小カットに含まれるかどうかの判別は可能だが，全ての辺に対して容量増減操作を適用するのは効率が悪いため，今回提案する方法では 3.1 で述べたボトルネック辺による絞り込みを先に行うようにしている。

3.3　増加パスと最小カットとの関係

最大フロー最小カット定理より，最小カットに含まれる辺の容量和は最大フローと一致する。したがって，最小カットを求める場合には，容量和が最大フローと一致するように最小カット辺を組み合わせる必要がある。最大フローを求める方法の1つとして，増加操作を繰返しながらフロー量を増加させていく方法を述べたが，各増加操作における増加パスは *s-t* パスであることから，パス上のどこかで必ず最小カットを通過することとなる。つまり，各増加パスは少なくとも1本以上の最小カット辺を含んでいると言える。一方で，全ての増加操作におけるフロー増加量の和は最大フローの流量と一致するため，各増加パスから1本ずつ最小カット辺を選択することで，辺容量の総和が最大フローと一致するような最小カット辺の組を自然に導くことができる。

ここで，1つの最小カット辺が複数の増加パスで選択されている場合の処理について考える。最小カット辺が含まれているパスではボトルネック辺として常にその最小カット辺が選ばれているというわけではなく，ある増加パスで最小カット辺が選択されてその容量が減った後で別の増加パスのボトルネック辺となる場合も考えられる。その際，上述した方法でパターンを列挙しようとすると，異なるパスから同じ辺を選択することになってしまう。そのため，重複する辺を選択している増加パスについて見てみると，最小カット辺以外の辺のみがボトルネックとなっている（＝全てのボトルネック辺が最小カット辺でない）ことが分かる。すなわち，最小カット辺以外の辺のみがボトルネックとなっているような増加パスからは辺の組を選ばないようにすれば，組の中で辺を重複して選択する問題は解決できる。

実際，もし『最小カット辺以外の辺がボトルネックとなっているような増加パスに含まれる最小カット辺がそのパス以外で選択されていない』場合を仮定すると，そのパスに含まれる最小カット辺は全ての増加操作が終わった後でも飽和していないため，最小カット辺であるという仮定に反する。よって，そのような最小カット辺はそのパス以外でも選択されていて，なおかつ最小カット辺は必ずどこかの増加パス上でボトルネック辺として現れることから，最小カット辺以外の辺のみがボトルネックとなっているような

増加パスからの辺選択を無視することによって，選択漏れも重複選択もない辺の組を作ることができる。

3.4 辺容量の総和が最大フローと一致する組み合わせパターン全てを求める方法

上で説明した辺の組の列挙を出力する方法を考えるために，以下の具体的な状況における組み合わせについて見ていく。増加パス番号とそのパスに含まれている最小カットに属する辺番号を，増加パス1は { 1, 2, 3 }，増加パス2 は { 4, 5, 6, 7 }，増加パス3は { 8, 9 } とする。

増加パスのフロー増加量の和は最大フローと一致するため，各増加パスから1本ずつ辺を選択すればその辺の組は自然と辺の容量和が最大フローと一致する組み合わせとなる。アルゴリズムとして実装する際は，このデータを元にして図3の右のように各パターンを列挙する必要があるが，今回は各パターンを1つ1つ求めるのではなく，各パターンの i 番目に選択される辺がどこなのかを全パターンに適用して，1列ごとに特定する方法でパターンを求める。具体的には，図3で言えば，1行目（1組目のパターン）を求め，2行目を求め，… とするのではなく，全パターンの1列目を求め，2列目を求め，… という動きとなる。

パターン番号	各パスの選択辺番号		
	パス１	パス２	パス３
パターン１	1	4	8
パターン２			9
パターン３		5	8
パターン４			9
パターン５		6	8
パターン６			9
パターン７		7	8
パターン８			9
パターン９	2	4	8
パターン１０			9
パターン１１		5	8
パターン１２			9
パターン１３		6	8
パターン１４			9
パターン１５		7	8
パターン１６			9
パターン１７	3	4	8
パターン１８			9
パターン１９		5	8
パターン２０			9
パターン２１		6	8
パターン２２			9
パターン２３		7	8
パターン２４			9

パターン番号	各パスの選択辺番号		
	パス１	パス２	パス３
パターン１	1	4	8
パターン２	1	4	9
パターン３	1	5	8
パターン４	1	5	9
パターン５	1	6	8
パターン６	1	6	9
パターン７	1	7	8
パターン８	1	7	9
パターン９	2	4	8
パターン１０	2	4	9
パターン１１	2	5	8
パターン１２	2	5	9
パターン１３	2	6	8
パターン１４	2	6	9
パターン１５	2	7	8
パターン１６	2	7	9
パターン１７	3	4	8
パターン１８	3	4	9
パターン１９	3	5	8
パターン２０	3	5	9
パターン２１	3	6	8
パターン２２	3	6	9
パターン２３	3	7	8
パターン２４	3	7	9

図3 データを元にした各パターン数の表示例
（左が樹形図，右が実際の組み合わせ）

表2 列数と最小カットパターンとの関係

列数	1	2	3
繰返し選択回数	8	2	1
周期	24	8	2
最小カット辺数	3	4	2

ここで，表2を参考にしながら以下の2点と列数との関係に着目しながら考えていく。

①繰返し選択回数：1周期のうちに1辺が何回連続で選択されるのか（表中の濃灰色）
②周期：何パターンで1周期となるか（表中の薄灰色）

図3で示した左の樹形図より，（i 番目の列の周期）は（i 番目の列の繰返し選択回数）×（i 番目のパスが含む最小カット辺数）で求めることができるため，考えるべきなのは各列において1周期中に1つのものを何回繰返し選択するかということになる。

まず，今回の選択方法ではパスから同じものを1つ1つ選んでいき，最後尾のパスだけ違うものを選ぶ，というように選択項目をずらしていくことによって全パターンを網羅する形になっているため，最後尾では同じものを複数繰返し選択する必要は無く，繰返し回数1回だけである。

一方，最後尾の1つ手前では，最後尾の列のみが異なる組を求めるために最後尾の周期数と同じだけの繰返し選択回数が必要となっている。同様にして，それ以前では1つ先の周期数と同じだけの繰返し選択回数が必要となることが分かる。

以上のことから，（i 番目の列の繰返し選択回数）＝（i+1番目の列の周期数）として求めることができ，アルゴリズムとして実装する際にはこのように列数に関する性質を応用したパターン列挙法が有効であることが分かる。例えば，図3において，1番目の列の繰返し選択回数8を展開すると次のようになる。

（1番目の列の繰返し選択回数：8）
＝（2番目の列の周期数：8）
＝（2番目の列の繰返し選択回数：2）×（2番目のパスの含む最小カット辺数：4）
＝（3番目の列の周期数：2）×（2番目のパスの含む最小カット辺数：4）
＝（3番目の列の繰返し選択回数：1）×（3番目のパスの含む最小カット辺数：2）
×（2番目のパスの含む最小カット辺数：4）

この例のように，繰返し選択回数はそれ以降のパスが含む最小カット辺数の積として帰納的に求めることができるため，以下のようにして計算することができる。

（i 番目の列の繰返し選択回数）＝（i+1番目以降のパスが含む最小カット辺数の積）
（i 番目の列の周期）＝（i 番目の列の繰返し選択回数）×（i 番目のパスが含む最小カット辺数）
＝（i 番目以降のパスが含む最小カット辺数の積）

4. 全ての最小カットを特定するアルゴリズム

今回提案する最小カット特定方法は，「2頂点間の最大フローを求める際に用いた増加パス」と「最大フロー時のボトルネック辺」に着目したものであり，具体的には，以下で示す最小カットの持つ特徴を踏まえて，パスのボトルネック辺から最小カットに含まれる辺を導き，それらの辺の組を各増加パスから選択することによって最小カットとなる組み合わせを求める，という流れになる。

この節では，前節で述べた性質を踏まえて最大フローから最小カットを求める処理手順とアルゴリズムとしての流れを示す。

4.1 最小カットを求める手順

全ての最小カットを求める大まかな手順は以下の５段階に分けられる。

1つ目の操作は，与えられたグラフに対して2頂点間の最大フローを求めることである。この操作は，2節で述べた最大フローを求める Edmonds-Karp アルゴリズムによって求めることができる。

2つ目の操作は，最大フロー時の各パスにおけるボトルネック辺を求めることである。3.1で述べた事実より，最大フロー時に飽和している辺がボトルネック辺であることから，この操作は最大フローが求まった時点での各辺のフロー量を各々の辺容量と比較することで求めることができる。

3つ目の操作は，ボトルネック辺の中から最小カットに含まれる辺を求めることである。3.2で述べた最小カット辺の容量増減と最大フローとの関係より，各ボトルネック辺の辺容量を1減少させて最大フローを求めた時，1の結果と比較して最大フローが減少した辺が最小カットに含まれる辺であると判定できる。

4つ目の操作は，和が最大フローと一致するような最小カット辺の組み合わせを求めることである。ここでは，3.4で述べたパターン列挙方法を元にして各増加パスから1本ずつ最小カット辺を選んで組み合わせることで，容量和が最大フローと一致するような最小カット辺の全ての組を求めることができる。

最後の操作は，4で求めた組の中でカットとなるものを求めることである。この操作は，4で求めた各組に対して，それらに含まれる全ての辺を元のグラフから取り除いた時，*s-t* パスが存在しない組があれば定義からカットであると判定ができる。

以上，5つの操作によって最終的に得られた辺の組が2頂点間の最小カットである。

4.2 最小カットを求めるアルゴリズム

4.1の処理手順をアルゴリズムとして書き表したものが以下のものになる。

〈全ての最小カットを求めるアルゴリズム〉

1 与えられたグラフに対して最大フローを求める。
2 最大フロー時のボトルネック辺を求める。
　For　全ての辺に以下を適用する。
　　If　最大フロー時に流れているフロー量 = 辺の容量　Then
　　・その辺はボトルネック辺である。
3 最小カットに含まれる辺を求める。
　For　全てのボトルネック辺に対して以下を適用する。
　①もとのグラフにおける辺容量を減少させる。
　②辺容量減少後の最大フローを求める。
　③ If　辺容量減少後の最大フロー量 ＜ 1で求めた最大フロー量　Then
　　・その辺は最小カットに含まれる辺である。
4 最小カットとなる可能性のある辺の組み合わせを求める。
　①各増加パスに含まれる最小カット辺数の積によって，選択辺の繰返し回数と周期を求める。
　②各増加パスに含まれる最小カット辺数の積によって，全体の組み合わせ数を求める。
　③ For　全ての増加パスに対して以下を適用する。
　　For　パターン数だけ以下を繰返し適用する。
　　・①の数値を元にしてパターン数に応じた辺を選択することで組み合わせを求める。
5 4で求めた組の中でカットとなるものを求める。
　For　4で求めた全ての組に対して以下を適用する。
　① For　その組に含まれている全ての辺に以下を適用する。
　　・辺容量を0にする（辺の削除）。
　②組に該当する辺を削除した後の最大フローを求める。
　③ If　最大フローが0　Then
　　・その組はカット（辺の組を削除後に *s-t* パスが存在しないため）である。
6 5で求まった組をグラフの最小カットとして出力する。

なお，紙面の都合上省略するが，以上のアルゴリズムをプログラミング言語 C によりプログラムを作成し，頂点数5から8，辺数8から17程度の幾つかの具体的なグラフに対して適用した結果，すべての最小カットの辺の組み合わせを列挙可能なことを確認している。

4.3 提案するアルゴリズムの時間計算量

上で提案したアルゴリズムの時間計算量について考える。与えられたグラフに対して，*n* を頂点数，*m*

を辺数，p を最大フロー時の増加パス数（= 増加操作の回数）とすると，①～⑤の各手順における時間計算量はそれぞれ，

$$O(mn),\quad O(m),\quad O(m^2n),\quad O(p\times 1.44^m),\quad O(mn\times 1.44^m)$$

となり，アルゴリズム全体としての時間計算量は$O(mn\times 1.44^m)$となる。

全体の時間計算量を考える上でネックとなっているのは，⑤の「求めた各組がカットとなっているかどうかを判定する」という処理の部分である。ここでは，④で求めた組の数だけ最大フローを求める操作を繰返しているため，組み合わせ数に依存する形で時間計算量が増えてしまっている。

また，これとは別のアルゴリズムとして「全てのカットを求めてその中で容量和が最小のものを探すことによって，グラフの持つ全ての最小カットを求める」という総当たり法が知られており，それらの計算時間は$O\left(mn^3\log\left(\frac{n^2}{m}\right)\right)$となっている。この結果を見ると，今回提案しているアルゴリズムの方が計算時間は遅くなってしまっているが，実際の計算においては④で得られる組み合わせパターン数は各パスに含まれるボトルネック辺の数に依存し，1つのパス上に多くのボトルネック辺が含まれる状況というのはまれであると言えるため，平均計算時間に関してはより短いものになると考えられる。

5. まとめ

本論文は，最大フローを求める際の増加パスに含まれるボトルネック辺と増加パスとの関係に着目し，各増加パスに含まれる最小カット辺の組み合わせを求めることで，与えられたグラフに対する全ての最小カットを得ることができるアルゴリズムを提案している。このアルゴリズムは従来のアルゴリズムに比べて計算時間が悪化してしまっているものの，最小カット全てを得ることができるものであり，この実行結果はグラフにおける各辺の重要度等を調べる上で役立つことも分かった。

一方，最小カットを求めるアルゴリズムは最大計算時間が指数時間となってしまっているため改良の余地が残されている。今後の課題としては，今回提案したボトルネック辺から全ての最小カットを求めるための方法を工夫する，あるいはボトルネック辺以外の性質に着目して全ての最小カットを求められるような方法を新たに考案することで計算時間の向上を図ることなどが挙げられる。

参考文献

[1] L.R.Ford and D.R.Fulkerson, Maxmal Flow Through a Network, Canadian Journal of Mathematics, Vol.8, pp.399–404, 1956.

[2] L.R.Ford and D.R.Fulkerson, A simple algorithm for finding maxmal network flows and an application to the Hitchcock problem, Canadian Journal of Mathematics, Vol.9, pp.210–218, 1957.

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

[4] E.L.Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York-London, 1976.

[5] Jon Kleniberg, Éva Tardos 著，浅野他訳，アルゴリズムデザイン，共立出版，2008.

[6] B. コルテ，J. フィーゲン著，浅野他訳，組み合わせ最適化（理論とアルゴリズム)』シュプリンガー・フェアラーク東京，2005.

[7] T.H. コルメン，C.E. ライザーソン，R.L. リベスト著，浅野他訳，アルゴリズムイントロダクション第2巻アルゴリズムの設計と解析手法，近代科学社，1995.

[8] R. ディーステル著，根上他訳，グラフ理論，シュプリンガー・ジャパン，2000.

[9] 恵羅博，土屋守正，増補改訂版 グラフ理論，産業図書，2010.

[10] 藤重悟，グラフ・ネットワーク・組合わせ論，共立出版，2002.

[11] 伊里正夫，小林隆，ネットワーク理論，日科技連出版社，1979.

[12] 奥村晴彦，アルゴリズム辞典，技術評論社，1991.
[13] 佐藤公男，原理がわかる工学選書 グラフ理論入門―C 言語によるプログラミングと応用問題―，日刊工業新聞社，1999.