情報処理学会論文誌　コンピューティングシステム　Vol.11　No.1　1–11 (Mar. 2018)

逆方向カット・エッジのない最小カットを求めるアルゴリズム

神保 潮[1,a)]　五島 正裕[2]

受付日 2017年7月21日, 採録日 2017年11月24日

概要：FF を用いた回路をラッチを用いた回路に変換する問題は，最小カット問題の一種に帰着する．ただしこの際，始点から終点に至るすべての道にカット・エッジをただ 1 つ含むという制約がある．既存の最小カット・アルゴリズムでは，この制約を満たすことができない．本稿では，この制約が，カットが逆方向カット・エッジを含まないことと等価であることを証明し，逆方向カット・エッジのない最小カットを見つけるアルゴリズムを提案する．このアルゴリズムにおいて最もオーダが大きい部分は既存の最大フロー・アルゴリズムであり，提案アルゴリズム全体のオーダはこれより悪化することはない．実験により，ゲート数約 3.4 万，配線数約 9.7 万程度の回路に対しても，約 375 秒の実用的な時間で最適解が求められることが分かった．

キーワード：グラフ・アルゴリズム，最小カット，最大フロー最小カット定理

An Algorithm to Find the Minimum Cut without Backward Cut Edges

Ushio Jimbo[1,a)]　Masahiro Goshima[2]

Received: July 21, 2017, Accepted: November 24, 2017

Abstract: A problem to convert a circuit with FFs into one with latches leads to a kind of minimum cut problem with a constraint that each of the paths from the start to terminal points has a single cut edge. Existing minimum cut algorithms do not satisfy this constraint. This paper proves that this constraint is equivalent to one that the cut does not have a backward cut edge, and proposes an algorithm to find the minimum cut without backward cut edges. The most complex part of the proposed algorithm is one of existing maximum flow algorithms, and the time-complexity of the whole proposed algorithm is not larger than the maximum flow algorithm used. Experimental results show that the algorithm can practically find the optimal solution for a large circuit with approximately 34 thousand gates and 97 thousand wires in approximately 375 seconds.

Keywords: graph algorithm, minimum cut, max-flow min-cut theorem

1. はじめに

同期回路における同期動作を実現する方式には，フリップ・フロップ（**FF**）を用いたもののほかに，二相の**ラッチ**を用いたものがある．前者に比べて後者は，タイミング制約が緩いという利点がある [1], [2], [3], [4] が，設計はより煩雑である．そこで，前者を入力として，後者を自動的に生成することが考えられる．

逆相ラッチ挿入問題

そのためには，**図 1**（上）から（中）に示すように，

(1) FF をラッチに変更すると同時に，

(2) FF と次の FF とに挟まれたロジックの中央に**逆相**のラッチを挿入する，

という変換を行えばよい．ただし逆相ラッチを挿入する際には，以下の制約がある：ロジック内のあらゆるパスに対して，逆相ラッチはただ 1 つ挿入されなければならない．

また，逆相ラッチの挿入位置に関しては，以下の 2 つの評価基準がある：

(1) ラッチの挿入個数は少ないほどよい．

(2) クリティカル・パスを短縮するため，挿入位置はロジッ

[1] 総合研究大学院大学複合科学研究科
School of Multidisciplinary Sciences, SOKENDAI, Chiyoda, Tokyo 101–8430, Japan
[2] 国立情報学研究所アーキテクチャ科学研究系
Systems Architecture Research Division, NII, Chiyoda, Tokyo 101–8430, Japan
[a)] ushio@nii.ac.jp

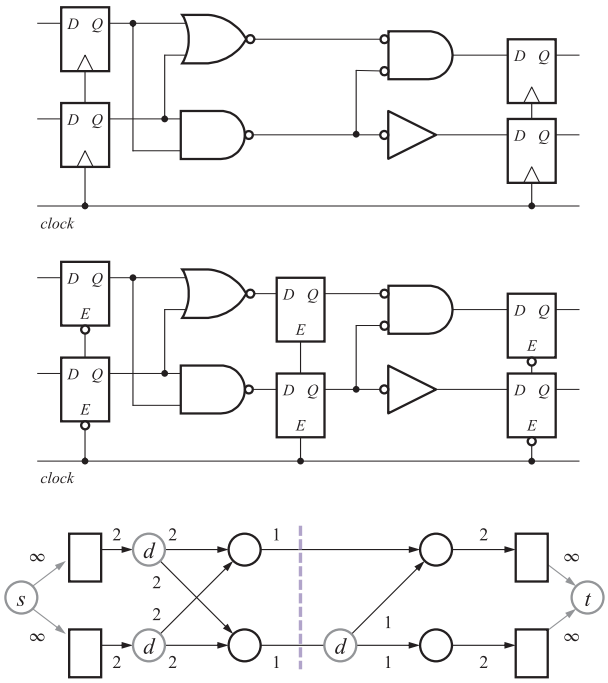

図 1　FF（上），二相ラッチを用いた回路（中）と，そのグラフ（下）
Fig. 1　Circuits with FFs (upper), with two-phase latches (middle), and their graph (lower).

ク内の各パスの遅延を等分することが望ましい．

グラフ・カット

これらの評価基準に対して最適な挿入位置を求めるにあたって，回路は，図 1（下）に示すようなグラフに写像することができる：

- ロジックの，FF やゲート等のインスタンスを頂点，ネットをエッジとする．
  インスタンスとネットには，信号の流れる向きがあるから，グラフ（エッジ）は有向となる．
- エッジのコストは，ステージ内の各パスの中央ほど低く，両端ほど高く設定する．

また，以下のようなダミーの頂点とエッジを追加する：

- 入/出力側の FF の前/後に，始点 s/終点 t を追加する．s/t と FF を結ぶエッジのコストは ∞ とし，それらがアルゴリズムに影響を与えることのないようにする．
- ネットの分岐にダミーの頂点を挿入する（図中 d）．

するとグラフは，ラッチが挿入されたエッジにおいて，s 側と t 側に二分される．このような二分割をグラフの**カット**という．2 つの分割にまたがるエッジ（この問題ではラッチが挿入されるエッジ）を**カット・エッジ**という．カット・エッジのコストの総和をカットの**サイズ**という．すると問題は，サイズ最小のカットを求める**最小カット**の問題（の一種）に帰着される．同図の例では，破線で示すカットが，サイズ $1+1=2$ で最小である．

なお，図中下側のカット・エッジのように，ネットの分岐に対して挿入したダミー d に対しては，その入力側を選ぶことで，複数の出力先をまとめて 1 個のラッチを挿入することを表現することができる．

単一カット・エッジ制約

逆相ラッチ挿入問題では，前述したように，ラッチはロジック内のあらゆるパスにただ 1 つ挿入されなければならない．この制約は，グラフの言葉では以下のようになる：

単一カット・エッジ制約　始点 s から終点 t へ至るあらゆる道上にカット・エッジが 1 つ存在する．

なお，逆相ラッチ挿入問題では，この制約を満たすカットは必ず存在する；すなわち，入力側ラッチの直後，あるいは，出力側ラッチの直前で分割したものである．特に前者の場合，逆相ラッチを挿入された回路は元の FF の回路と等価となる．

既存の最小カット・アルゴリズム

既存の最小カット・アルゴリズムは，道上のカットの数を意識せず，この問題にそのまま用いることはできない．4.6 節で示すように，実用的な回路に対しても単一カット・エッジ制約を満たさないカットが選ばれることがある．

そこで本稿では，この単一カット・エッジ制約を満たすカットのうちでサイズ最小となるものを見つけるアルゴリズムを提案する．本稿の構成は以下のとおりである：2 章で既存のアルゴリズムについてまとめた後，3 章で提案のアルゴリズムについて詳しく述べる．4 章では，逆相ラッチ挿入問題に対してアルゴリズムを実行した結果について述べる．

2. 既存のアルゴリズム

最小カットを求めるには，最大フロー最小カット定理に基づいて，最大フローを求めた結果として最小カットを求めることが一般的である．2.1 節と 2.2 節では，そのようなアルゴリズムの例として，最も基本的なフォード・ファルカーソンのアルゴリズムを概説する．

しかし，このような最大フローに基づく方法では，1 章で述べた単一カット・エッジ制約を満たすことはできない．2.3 節で紹介する無向グラフに変換する方法もまた，単一カット・エッジ制約を満たさない．2.4 節で紹介する探索による方法は，単一カット・エッジ制約を満たすことを念頭に設計されたものであるが，実行時間に問題がある．

2.1　フォード・ファルカーソンのアルゴリズム

前述のように，最小カットは，フロー・ネットワークにおける最大フローを求めた結果から求めることが一般的である．なお，フロー・ネットワークにおいては，エッジに与える重みはフローの**容量**と呼ぶが，前章におけるコストと同じと考えてよい．

そのための最大フロー・アルゴリズムとしては，**フォード・ファルカーソンのアルゴリズム**（Ford-Fulkerson algorithm）[5] が最も基本的である．

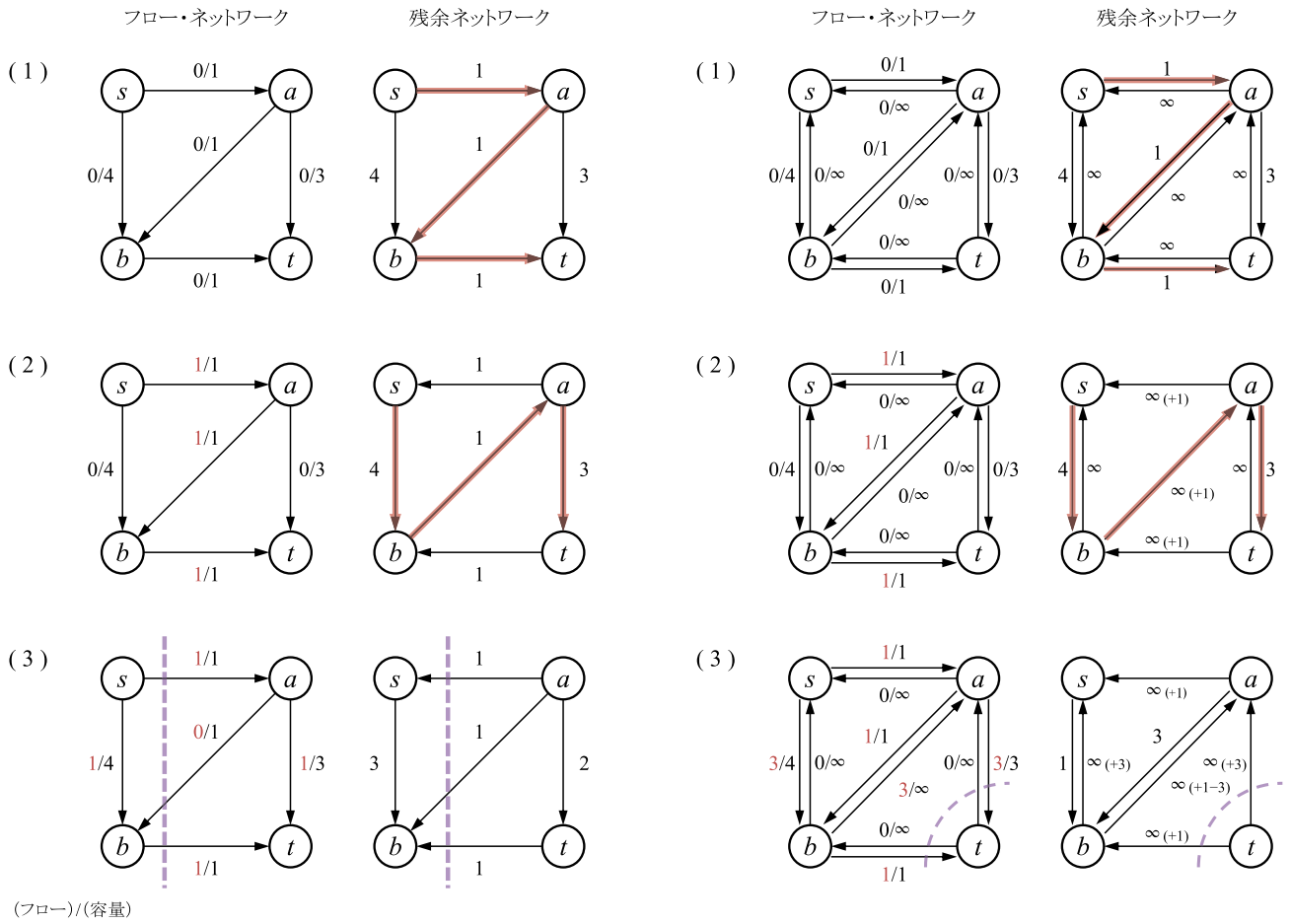

図 2　オリジナル（左）と提案（右）のフォード・ファルカーソンのアルゴリズムの動作例
Fig. 2　Behavior of Ford-Fulkerson algorithm in original (left) and proposal (right).

残余ネットワーク

フォード・ファルカーソンのアルゴリズムでは，元のフロー・ネットワークに対して**残余ネットワーク**（residual network）というネットワークを生成する．元のフロー・ネットワークにおいて容量 $c(u,v)$ のエッジ $u \to v$ にフロー $f(u,v)$ を流したとき，残余ネットワークにおける u, v 間には順方向と逆方向の 2 つの有向エッジを張る：

順方向　まだあと $c_f(u,v) = c(u,v) - f(u,v)$ だけ流せるという意味で，容量 $c_f(u,v)$ の順方向エッジ

逆方向　逆に，$c_b(u,v) = f(u,v)$ だけ減らすことができるという意味で，容量 $c_b(u,v)$ の逆方向エッジ

なお，$c_f(u,v) + c_b(u,v) = c(u,v) - f(u,v) + f(u,v) = c(u,v)$ である．

増加道

残余ネットワークにおいて s から t へと至る道を**増加道**（augmenting path）という．s から t へのフローは，増加道の容量の最小値（すなわち，増加道を形成するすべてのエッジの容量のうちの最小値）だけ，増加させることができる．

増加道に上述した逆方向エッジが含まれる場合には，そのエッジのフローを逆に減少させることになる．このおかげで，増加道をグリーディに見つけて行っても最大フローが求まるというのがフォード・ファルカーソンのアルゴリズムの要諦である．証明は，文献 [6] 等を参照されたい．

アルゴリズムの動作例

図 2（左）の例を用いてフォード・ファルカーソンのアルゴリズムの動作を説明する．同図中，(1)〜(3) の各段階において，左がフロー・ネットワーク，右がそのフロー・ネットワークに対応する残余ネットワークを示す．一般に，フロー・ネットワークにおいては，エッジに「(フロー)/(容量)」と付す．また，残余ネットワークにおいては，容量 0 となったエッジは除去する[*1]．

アルゴリズムは，以下のように進む：

(1) 初期状態では，フローは 0 とする．
　したがって右の残余ネットワークは，左のフロー・ネットワークの容量をそのまま写したものとなる．
　この残余ネットワークにおいて，s から t へ至る増加道を探す．辞書順だと，$s \to a \to b \to t$ が見つかる（太矢印）．

[*1] 主に図の見やすさのため．要は容量 0 のエッジを含む増加道を見つけないようにすればよいので，プログラムでは容量 0 のエッジとして残しておいた方が実装が容易であろう．

(2) この増加道に最大のフローを流す．この増加道を形成するエッジ $s \to a$，$a \to b$，$b \to t$ の容量はすべて 1 であるから，それぞれに容量一杯のフロー 1 を流すことになる．すると，左のフロー・ネットワークが得られる．

さらにこのフロー・ネットワークから右の残余ネットワークを得る．エッジ $s \to a$，$a \to b$，$b \to t$ のそれぞれに容量一杯のフロー 1 を流したため，残余はそれぞれ 0 となる．これらのエッジに対しては逆に，フローを 1 だけ減らすことができるという意味で，容量 1 の逆向きのエッジを張る．

この残余ネットワークにおいては，増加道 $s \to b \to a \to t$ が見つかる（同じく太矢印）．

(3) この増加道に最大容量である 1 のフローを流すと，左のフロー・ネットワークが得られる．エッジ $a \to b$ には，先ほど 1 のフローを流したが；今回，同じく 1 のフローを今度は逆向きに流したので，フローはキャンセルされて 0 に戻る．

さらにこれから右の残余ネットワークが得られる．この残余ネットワークにおいては，s から t へ至る増加道はもはや見つからないので，アルゴリズムは終了する．

最大フロー

増加道 $s \to a \to b \to t$ と $s \to b \to a \to t$（図 2 中の太矢印 2 本）によって 1 ずつ増加されたので，最大フローは合計 2 となる．

物理的なフローは，これらの 2 つの増加道の重ね合わせである；すなわち，道 $s \to a \to t$ と $s \to b \to t$ に 1 ずつ流されている．キャンセルされるので，エッジ $a \to b$ にはフローは流されない（0 である）．

2.2　最大フロー最小カット定理

フォード・ファルカーソンのアルゴリズムにおいては，最小カットは，アルゴリズム終了時の残余ネットワークにおいて，s から到達可能な頂点とそれ以外の頂点への分割として与えられる．図 2 の場合，終了時の残余ネットワーク，すなわち，(3) 右において，s から到達可能であるのは b のみであるので，最小カットは $\{s, b\}$ と $\{a, t\}$ である．

2 つの部分をまたがるカット・エッジのうち，$s \to a$ と $b \to t$ は，s 側から t 側へ向かう順方向エッジであり；$a \to b$ は逆に，t 側から s 側へ向かう逆方向エッジである．最大フロー最小カット定理の文脈においては，上述した最大フロー 2 に対する最小カットは，順方向カット・エッジの容量の総和，すなわち，$c(s,a)+c(b,t) = 1+1 = 2$ で与えられ；逆方向カット・エッジの容量，すなわち，$c(a,b) = 1$ は含めない．順方向カット・エッジのフローはそれぞれの容量一杯であり，同時に，逆方向カット・エッジのフローはすべて 0 であるとき，最大フローは実現される（最大フロー最小カット定理）．

この最小カットでは，道 $s \to a \to b \to t$ を構成する 3 つのエッジ $s \to a$，$a \to b$，$b \to t$ はすべてカット・エッジとなっており，単一カット・エッジ制約は満たされていない．

2.3　無向グラフにおける最小カット

前節までで述べたような最大フロー・アルゴリズムを用いる以外には，無向グラフ化して最小カットを求めることが考えられる．

無向グラフに対する全域的な最小カットも，Nagamochi らのアルゴリズム [7], [8] 等を用いて，効率良く求めることができる．

しかし，無向グラフにおける最小カットも，単一カット・エッジ制約を満たすとは限らない．実際，図 2 の例では，フォード・ファルカーソンのアルゴリズム場合と同じ最小カット（図 2（左）の (3)）が得られる[*2]．

2.4　探索による方法

単一カット・エッジ制約を満たすため，我々は，探索による方法を試してきた [9]．

探索木のノードは，カット・エッジ候補の集合とする．すなわち，探索木の葉において，この集合の要素がカット・エッジとなる．探索ノードの展開は，この集合にエッジを 1 つ加えることになる．展開のたびに，加えられたエッジからグラフを上流，下流へと経路探索し，経路上のエッジを候補から除外する．この除外によって，単一カット・エッジ制約は保証される．この探索木上で最良優先探索を行えば，単一カット・エッジ制約を満たすカットのうちサイズ最小のものが得られる．

しかしこの方法は，計算量の大きさと，探索空間の広さのため，実用的な時間内に結果を得られていない：

計算量の大きさ　エッジの候補からの除外は，エッジ数 E に対して $O(E)$ の時間がかかる．そのため，通常の探索問題に比べ，展開に時間がかかる．

探索空間の広さ　たとえば，始点側 FF から終点側 FF までの最短経路がエッジ 10 段あり，1 段ごとにカット・エッジの候補が 100 あるとすると，解空間は 100^{10} にもなる．探索順序を工夫して 1/100 のノードの探索で解が求まったとしても，探索ノード数は依然 100^9 もある．

実際，4 章で評価するエッジ数 10 万程度の回路に対して探索を行ったところ，2 日経過しても終了しなかった．なお，提案手法では，約 375 秒で終了する（4.4 節）．

なお，カット・エッジを加えると残りの解空間が変化するため，A^* アルゴリズムのためのヒューリスティック関数は見つけられていない．

[*2] ただし最小カットのサイズは，$c(s,b)+c(a,t) = 1+1 = 2$ ではなく，$c(a,b) = 1$ を含む 3 となる．

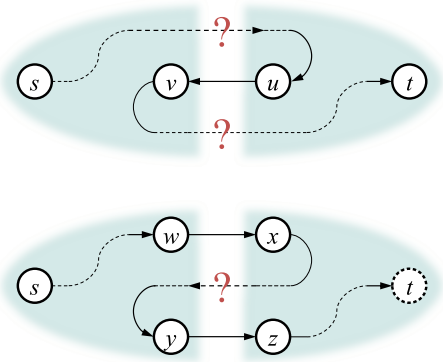

図 3　逆方向カット・エッジを含むカット
Fig. 3　Cut with a backward cut edge.

3. 提案アルゴリズム

本稿で求めるべきカットには，始点から終点へ至るすべての道にカット・エッジがただ 1 つ現れるという単一カット・エッジ制約がある．実はこの制約は，逆方向カット・エッジがないという逆方向カット・エッジなし制約と等価である．そこで本稿では，逆方向カット・エッジを含まない最小カットを求めるアルゴリズムを考える．まず，3.1 節で，単一カット・エッジ制約と逆方向カット・エッジなし制約が等価であることを証明する．その後，3.2 節と 3.3 節で，アルゴリズムの手順と動作例を示す．アルゴリズムの正しさの証明は，改めて 3.4 節で行う．提案手法の中心部分は最大フロー・アルゴリズムであり，計算量と停止性は，どの最大フロー・アルゴリズムを採用するかに依存する．3.5〜3.8 節では，エドモンズ・カープのアルゴリズムを採用した場合の計算量と停止性について説明する．

3.1 逆方向カット・エッジなし制約

定理 1. フロー・ネットワークとそのカットにおいて，以下は等価である：

単一カット・エッジ制約　始点 s から終点 t へ至るすべての道にカット・エッジがただ 1 つ現れる．

逆方向カット・エッジなし制約　逆方向カット・エッジがない，すなわち，カット・エッジはすべて順方向である．

証明.　単一カット・エッジ制約 ⇒ 逆方向カット・エッジなし制約を，背理法で証明する．s から t へ至る道にカット・エッジがただ 1 つ現れ，かつ，それが逆方向であると仮定する．その逆方向カット・エッジを $u \to v$ とする（**図 3** 上）．すると，s から u，v から t へ至る道は（順方向の）カット・エッジを含むことになり，ただ 1 つという仮定と矛盾する．

逆も，同じく背理法で証明する．カット・エッジはすべて順方向エッジであり，かつ，s から t へ至る道のうち，カット・エッジが 2 つ以上現れるものがあると仮定する．2 つの順方向カット・エッジを $w \to x$ と $y \to z$ とする（図 3 下）．すると，道 $x \to \cdots \to y$ は，逆方向カット・エッジを含むことになり，カット・エッジがすべて順方向であるという仮定と矛盾する．□

図 4　逆方向エッジの削除
Fig. 4　Elimination of a backward edge.

したがって，1 章の問題等のために，始点から終点へ至るすべての道にカット・エッジがただ 1 つ含まれるようなカットを求めるためには，逆方向カット・エッジのないカットを求めればよい．

3.2 提案アルゴリズムの手順

提案の逆方向カット・エッジを含まない最小カット・アルゴリズムは，以下のとおりである：

(1) **前処理**　元のフロー・ネットワークのすべてのエッジに対して，容量 ∞ の逆方向エッジを追加する．

(2) **最大フロー**　このフロー・ネットワークに対して，任意の最大フロー・アルゴリズムによって最小カットを求める．

ただしもちろん，最大フロー・アルゴリズムは，(1) によって変更された条件，すなわち，∞ の容量に対応したものでなければならない．逆方向エッジについては，**図 4** のようにすれば必ず削除できる．

証明は 3.4 節で行うが，直感的には，このアルゴリズムは以下のようにして逆方向カット・エッジを避ける：あるエッジを逆方向カット・エッジとして選ぼうとすると（元の順方向エッジではなく）追加された逆方向エッジの容量 ∞ がカット・サイズに加算されてしまう．したがって，有限のサイズが達成されるならば，最小カットには逆方向カット・エッジは含まれない．

3.3 動作例

本節では主に，前節における (2)，すなわち，最大フローを求める部分の動作を説明する．

動作例

図 2（右）に，提案アルゴリズムの動作を示す．対象のフロー・ネットワークは同図（左）と同一であり，（右）では (1) において容量 ∞ の逆方向エッジが追加されている．

最大フロー・アルゴリズムとしては，同図（左）と同じフォード・ファルカーソンのアルゴリズムを用いている．したがって，同図（左）/（右）では，同一の最大フロー・アルゴリズムが異なる初期フロー・ネットワークに対してどのように振る舞うかを見ることになる．アルゴリズムは以下のように進むが，実際，(2) までは，同図（左）と変わらない：

(1) 増加道 $s \to a \to b \to t$ が見つかる（太矢印）．
(2) この増加道に最大容量である 1 のフローを流すと，フロー/残余ネットワークが得られる．この残余ネットワークにおいては，増加道 $s \to b \to a \to t$ が見つかる（太矢印）．
(3) ただし，この増加道のフローは（左）とは異なる．この増加道のフローは，（左）では $b \to a$ の 1 によって制限されていた．（右）では，容量 ∞ の逆方向エッジ $b \to a$ が追加されたため，$b \to a$ ではなく，$a \to t$ の 3 によって制限されることになる．

この結果，$a \to t$ は飽和し，残余ネットワークにおいて増加道はもはや見つからず，アルゴリズムは終了する．

最大フロー最小カット

最大フローは $1+3=4$ となり，元のフロー・ネットワークにおける最大フロー 2 より大きい．これは，元のフロー・ネットワークに存在しない容量 ∞ の逆方向エッジ $b \to a$ に 2 のフローを流すことによって達成されていることに注意する必要がある．

2.1 節で述べたとおり，フォード・ファルカーソンのアルゴリズムでは，最小カットはアルゴリズム終了時の残余ネットワークにおいて s から到達可能な頂点とそれ以外の頂点への分割として与えられる．同図の場合，s から b，そして a へ到達可能であるので，最小カットは $\{s,a,b\}$ と $\{t\}$ となる．カット・エッジは，$a \to t$ と $b \to t$ であり，逆方向エッジは含まれない．また，s から t へ至るあらゆる道上でカット・エッジは 1 つである．

最小カットは，最大フローと同じ 4 である．これは，次節で証明するように，元のフロー・ネットワークにおいて逆方向カット・エッジを含まないカットのうちで最小のものである．

3.4　証明

定理 2. 提案アルゴリズムによって得られるカットは，元のフロー・ネットワークのカットの中で，逆方向カット・エッジなし制約（単一カット・エッジ制約でも等価）を満たすものがあれば，それらのうちでサイズ最小のものである．

証明. 図 5 の上の列は，それぞれ，あるフロー・ネットワークのすべてのカットを，それらのカット・サイズに従って昇順に並べたものである．図中のアイコンは，楕円は元のフロー・ネットワークを，矢印はエッジを，破線はカットを，それぞれ模式的に表している．特に，左向き矢印が破線に重なっている場合，この矢印は逆方向カット・エッジとなっている．

これらのカットに対して，仮に，提案手法の前処理を施した，すなわち，すべてのエッジに容量 ∞ の逆方向エッジを追加した場合のカット・サイズを求めよう．下の列は，上の列のカットを，この新たに求めたカット・サイズの昇順に並べ直したものである．同じカットを異なるサイズに従って並べ直しただけであるから，上/下の列のカット間には，1 対 1 の関係がある．上下を結ぶ矢印は，この 1 対 1 の関係を示している．

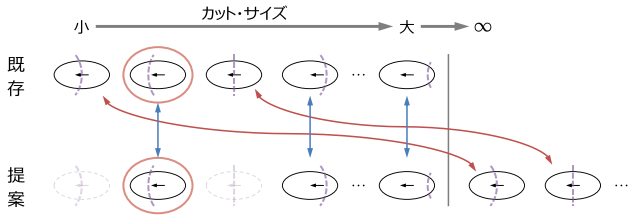

図 5　既存（上）/提案（下）によって計算されたサイズによるカットの昇順列

Fig. 5　Sequences of cuts in ascending order of their sizes computed by existing (upper)/proposed (lower) algorithms.

これらのカットの中には，元のフロー・ネットワークにおいてカット・エッジに逆方向エッジを含むものと含まないものがある．その結果，上下では以下のような順序の変化がある：

逆方向カット・エッジを含むカット　前処理を施した場合には，追加された逆方向エッジの容量 ∞ が加算されて，カット・サイズも ∞ になる．その結果，下の列では最後尾に移動することになる．

逆方向カット・エッジを含まないカット　前処理を施した場合でも，追加された逆方向エッジの ∞ の容量が加算されないため，カット・サイズは変化しない．その結果，下の列でもその位置に残されている．

前処理後に最大フロー・アルゴリズムを適用すると（その最大フロー・アルゴリズムが最小カットを見つけるならば）図中，下の列において一番左，◯を付けて示されたカットが得られる．

ここで，逆方向カット・エッジを含まないカットのみに注目しよう．前処理を施した場合でもカット・サイズは変化しないため，それらの間の順序は上下で変化しないことが分かる（上下を結ぶ直線の矢印）．

したがって，提案アルゴリズムで得られた最小カットは，上の列においては，左から 2 番目，同じく◯を付けて示されたカットと同一のものである．これは，元のフロー・ネットワークのカットの中で逆方向カット・エッジを含まないもののうち，サイズ最小のものである．　□

3.5　エドモンズ・カープのアルゴリズム

最大フロー・アルゴリズムは数多く存在する [10] が，4 章の実験ではエドモンズ・カープのアルゴリズムを用いた．本節では，このアルゴリズムについて簡単に述べ；次節では，このアルゴリズムを用いた場合の提案手法の計算量について議論する．

2.1 節で述べたフォード・ファルカーソンのアルゴリズ

ムでは，増加道の探索順序は指定されていない．図 2 の例では，辞書順とした．**エドモンズ・カープのアルゴリズム**（Edmonds-Karp algorithm）[11] は，この特に指定されていなかった増加道の探索順序を幅優先と指定したものである；すなわち，エドモンズ・カープのアルゴリズムはフォード・ファルカーソンのアルゴリズムの特殊化（a specialization）と位置付けられる．このため，フォード・ファルカーソンのアルゴリズムといっていても実際にはエドモンズ・カープのアルゴリズムが実装されている場合がある．

フォード・ファルカーソンのアルゴリズムの実行時間は，容量が整数の場合には，頂点数を V，エッジ数を E，最大フローを F とすると，$O(EF)$ である．実行時間に F が含まれるのは，最悪の場合，増加道 1 本につきフローが 1 ずつしか増加しないことによる．そのため，F が大きい場合には時間がかかることがある．同様の理由により，容量が無理数である場合には，停止しないことがある．

エドモンズ・カープのアルゴリズムでは，増加道の探索順序を幅優先とすることで，このような問題を解決している．エドモンズ・カープのアルゴリズムでは，増加道は，（重みなしとした場合の）幅優先で，すなわち，ホップ数の少ないものから順に探索される．図 2 のフロー・ネットワークに対してエドモンズ・カープのアルゴリズムを適用した場合には，2 ホップの道 $s \to a \to t$，$s \to b \to t$ が先に選ばれ，3 ホップの道 $s \to a \to b \to t$ 等は選ばれることなく終了する．

3.6　エドモンズ・カープのアルゴリズムの計算量

次節で，提案における容量 ∞ のエッジの追加がどのように影響するかについて議論する前に，本節では，エドモンズ・カープのアルゴリズムの計算量が $O(VE^2)$ であることを説明する．なお，本節での説明は一般的なもの [6] で，特に本稿の論旨に依存しないが，論理はホップ数に基づいて展開されており，容量が出現しないことに注意されたい．

以下，始点 s からある頂点へのホップ数を，単に，その頂点へのホップ数という．また，あるフロー・ネットワークにおいて，あるフローの状態 f における残余ネットワークを R_f，R_f 上の v へのホップ数を $\delta_f(v)$ とする．

定理 3. エドモンズ・カープのアルゴリズムでは，残余ネットワーク上の任意の頂点へのホップ数は，フロー増加操作によって減少しない．

証明. あるフロー増加操作の結果，ある頂点へのホップ数が減少すると仮定し，矛盾を導く．

このフロー増加操作の前/後のフロー状態を f/f' とする．この増加操作によってホップ数が減少した頂点の中で，f' でのホップ数最小のものを v とする．すなわち，$\delta_f(v) > \delta_{f'}(v)$ である．

また，$R_{f'}$ 上で，v へのホップ数最小の道上，v の手前の頂点を u とする．すなわち，$u \to v \in E_{f'}$ であり，かつ，

$$\delta_{f'}(u) = \delta_{f'}(v) - 1. \tag{1}$$

さらに，この増加操作によってホップ数が減少した頂点の中で f' でのホップ数最小のものを v としたのであるから，u へのホップ数は減少していない，すなわち，

$$\delta_f(u) \le \delta_{f'}(u). \tag{2}$$

R_f に $u \to v$ があるかどうかに分けて考える：

***R_f* に $u \to v$ がある場合**　$u \to v$ を通る道がありうるから，$\delta_f(v) \le \delta_f(u) + 1$ である．
ここで式 (2) より，$\delta_f(u)+1 \le \delta_{f'}(u)+1$．さらに式 (1) より，$\delta_{f'}(u) + 1 = \delta_{f'}(v)$．合わせて，$\delta_f(v) \le \delta_{f'}(v)$ となる．

***R_f* に $u \to v$ がない場合**　$u \to v$ は，R_f にあり，$R_{f'}$ にないから，増加操作によって逆方向のエッジ $v \to u$ のフローが増加したことになる．エドモンズ・カープのアルゴリズムはホップ数最小の道を増加道に選ぶから，R_f 上の u までのホップ数最小の道の最後のエッジは $v \to u$ である．したがって，$\delta_f(v) = \delta_f(u) - 1$．
ここで式 (2) より，$\delta_f(u) - 1 \le \delta_{f'}(u) - 1$．さらに式 (1) より，$\delta_{f'}(u) - 1 = \delta_{f'}(v) - 2$．合わせて，$\delta_f(v) \le \delta_{f'}(v) - 2$ となる．

いずれの場合も，$\delta_f(v) > \delta_{f'}(v)$ に矛盾する．したがって，このような頂点 v は存在しない．　□

定理 4. エドモンズ・カープのアルゴリズムでは，1 つのエッジが飽和する回数は $O(V)$ である．

証明. 増加道はホップ数最小の道だから，あるフロー状態を f においてエッジ $u \to v$ が初めて飽和するとすると，$\delta_f(v) = \delta_f(u) + 1$ である．飽和したので，$u \to v$ は残余ネットワークから消える．

$u \to v$ が再び別の増加道上に出現するには，その前までのいずれかのフロー状態 f' において，逆方向のエッジ $v \to u$ が増加道上に出現する必要がある．したがって，$\delta_{f'}(u) = \delta_{f'}(v) + 1$ が成立する．

一方，定理 3 により，フロー増加操作で任意の頂点までのホップ数は減少しないから，$\delta_f(v) \le \delta_{f'}(v)$ である．

以上から，$\delta_{f'}(u) = \delta_{f'}(v) + 1 \ge \delta_f(v) + 1 = \delta_f(u) + 2$ を得る．すなわち，$u \to v$ が飽和した時点から，次に飽和する時点までの間に，u へのホップ数は必ず 2 以上増える．

ホップ数は最大で $V - 1$ であるから，$u \to v$ が飽和した後にさらにたかだか $(V - 1)/2$ 回しか飽和しない．すなわち，1 つのエッジの飽和の回数は $O(V)$ である．　□

エッジのそれぞれが飽和する回数は $O(V)$ であるから，エッジすべてについてでは $O(VE)$ である．そして，増加道の探索は幅優先探索で $O(E)$ であるから，アルゴリズム

全体の計算量は $O(VE^2)$ となる．

前述したように，論理はホップ数に基づいて展開されており，容量は出現しないことが確認できる．

3.7　提案手法の計算量と停止性

本節では，提案の前処理によって追加される容量 ∞ のエッジがエドモンズ・カープのアルゴリズムの計算量に与える影響を説明する．

前節の証明では，増加操作によって 1 つ以上のエッジが飽和することが暗に仮定されている．一方，提案アルゴリズムにおいては，容量 ∞ のエッジは飽和しないため，これを自明のものとして仮定することはできない．

厳密には，逆方向カット・エッジなし制約を満たすカットが存在する場合には，以下の定理によって，前節の証明はそのまま成立する：

定理 5. 逆方向カット・エッジなし制約を満たすカットが存在するフロー・ネットワークに対しては，容量 ∞ の逆方向エッジを追加した場合であっても，1 回の増加操作で 1 つ以上のエッジが飽和する．

証明. 任意の増加道 $s \rightsquigarrow t$ に対して，逆方向カット・エッジなし制約を満たすカットを考える．この増加道 $s \rightsquigarrow t$ のエッジの中には，これらのカットのカット・エッジでもあるものがある．これらのカットは逆方向カット・エッジなし制約を満たすから，これらのエッジは順方向で，有限の容量を持つ．

増加道上に有限容量のエッジが 1 つ以上あるとき，それらの中で最小の容量に等しいフローが流され，その容量を持つエッジはすべて飽和する．

以上より，逆方向カット・エッジなし制約を満たすカットが存在するなら，1 回の増加操作で 1 つ以上のエッジが飽和することになる． □

したがって提案アルゴリズムの場合でも，前節の証明はそのまま成立し，（エドモンズ・カープのアルゴリズム部分の）計算量は $O(VE^2)$ となり，停止性は保証される．

3.8　逆方向エッジ追加による計算量の変化

3.6 節で述べたように，エドモンズ・カープのアルゴリズムの実行時間は $O(VE^2)$ である．提案では，逆方向エッジを追加する前処理によってエッジ数 E は 2 倍になるので，実行時間は $2^2 = 4$ 倍になるように思われるが，実際にはそうではない．4 章の評価では，1.5 倍程度にとどまっている．それは，以下の理由による．

エドモンズ・カープのアルゴリズムをはじめ，フォード・ファルカーソンのアルゴリズムをベースとする最大フロー・アルゴリズムでは，残余ネットワークの逆方向エッジも順方向エッジと区別なく扱われる．3.6 節で見たように，$O(VE^2)$ という実行時間は，この逆方向エッジを考慮に入れたものである．

提案における逆方向エッジは，通常なら増加操作によって随時追加されるものを，初期状態から追加するにすぎない．したがって，残余ネットワークを用いる最大フロー・アルゴリズムを用いた場合，提案手法の実行時間はオーダ上は悪化しない（4.5 節）．

4.　実験

1 章で紹介した逆相ラッチの挿入位置を求める問題を例として，提案アルゴリズムを適用した．

4.1　プログラム開発・実行環境

表 1 に，開発・実行環境をまとめる．

C#と，C#のグラフ用のライブラリ QuickGraph [12] を用いてプログラムを記述した．

3.5 節で述べたように，最大フロー・アルゴリズムとしては，エドモンズ・カープのアルゴリズムを用いた．エドモンズ・カープのアルゴリズムは，QuickGraph にも Algorithm-Extensions.MaximumFlowEdmondsKarp として含まれているが，インタフェースが他の部分と合わなかったため，今回は自前で記述したものを用いた．記述にあたっては，文献 [13] を参考にした．

4.2　実験対象

1 章で述べたように，回路を基にしたフロー・ネットワークに対して上記のプログラムを実行した．

回路としては，リプル・キャリー・アダーを用いた 32-bit カウンタと，RISC-V ISA [14] に準拠する 64-bit スカラ・プロセッサ Rocket [15] を用いた．

Vivado Design Suite 2016.3 を用いて Xilinx Artix-7 FPGA をターゲットにダウンロード可能なネットリストを入力とし，FF のデータ出力から FF のデータ入力に至る連結な部分を 1 つのステージとして切り出し [16], [17]，ステージごとに提案アルゴリズムを適用した．

カウンタは，4.6 節で結果を詳しく見るためのもので，FPGA の持つハードウェア・キャリー・チェーンを用いずに，ユーザ・ロジックで構成した．

表 1　開発・実行環境
Table 1　Development/execution environment.

CPU	Intel Core i7-4770，3.40 GHz
RAM	DDR3，PC3-12800，8 GB × 2
OS	Windows 10 Pro，Ver. 1703
C#開発環境	Visual Studio 2015
ビルド	Release（最適化あり，デバッグ出力なし）
FPGA 開発環境	Vivado Design Suite 2016.3
FPGA	Xilinx Artix-7

4.3　エッジの容量

逆相ラッチの挿入位置を求める問題に対して，エッジの容量は以下のように定めた [16], [17].

評価基準

1 章で述べたように，逆相ラッチの挿入位置に関しては，以下の 2 つの評価基準がある：

(1) ラッチの挿入個数は少ないほど良い．
(2) クリティカル・パスを短縮するため，挿入位置はロジック内の各パスの遅延を等分することが望ましい．

より正確には，(2) は以下のように修正される：

(2) パスを正確に等分することが重要性は，そのパスの長さに依存する．

すなわち，クリティカル・パスでは，パスを等分することが重要である一方；非クリティカル・パスでは，そのパスの短さに応じて，(1) 個数の優先度が高くなる．

エッジの容量

そこで，エッジの容量は，パスを等分するラッチを 1 個として，等分しないラッチを C 個分と考えることにする．たとえば，あるエッジの容量が 10 であるときには，そのエッジを選択することで全体で 11 個のラッチが削減できるなら，そのエッジのずれは許容されることになる．

評価関数

今回は，エッジの容量 C は，そのエッジを含むパスのうち最長のものに対して，以下のヒューリスティックな評価関数を用いた：

$$C(d,p) = B(p)^{10d}$$
$$B(p) = (N-n)p^M + n$$

d　クリティカル・パス長で正規化した，最も中央に近い挿入位置からのずれ．
p　クリティカル・パス長で正規化した，そのエッジを含むパスのうち最長のもの長さ．
N　クリティカル・パスにおいて，クリティカル・パス比 10% のずれがラッチ何個分に相当するか．今回は 10.
n　長さ 0 の仮想的なパスにおいて，クリティカル・パス比 10% のずれがラッチ何個分に相当するか．今回は 2.
M　非クリティカル・パスにおいて，N の大きさの効果を緩和する度合いを表すパラメータ．今回は 1.5.

長さは，ユニット遅延でも実遅延でもよい．今回は，ユニット遅延を用いた．

この関数は，以下のように，前述の評価基準を満たす．

まず $C(d,p)$ は，$B(p)$ を底として，ずれ d に対して指数関数的に増加する．

底 $B(p)$ は，そのエッジを含むパス（のうち最長のもの）の長さによって，$(n,N]$ の範囲で変化する．エッジがクリティカル・パス上にある（$p=1$）場合，$B(1) = (N-n)\times 1^M + n = N$；一方，長さ 0 の仮想的なパス上にある（$p=0$）場合，$B(0) = (N-n)\times 0^M + n = n$ となる．

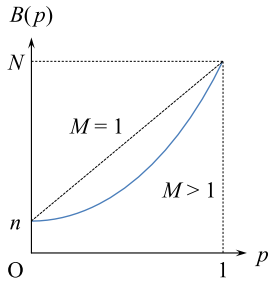

図 6　$B(p)$ のグラフ
Fig. 6　Graph of $B(p)$.

そして C は，最も中央に近い位置に挿入された場合 ($d=0$)，$C(0,p) = B(p)^0 = 1$ と，p にかかわらず 1 となる．逆に中央からずれた場合には，パスの長さによって以下のように変化する：

- クリティカル・パス上（$p=1$，$B(1)=N$）では，10% ずれた（$d=0.1$）場合，$C(0.1,1) = N^{10\times 0.1} = N$；最もずれた（$d=0.5$）場合，$C(0.5,1) = N^5$ となる．ずれ d に対して指数関数的に増加する結果，クリティカル・パスにおいては，中央からのずれが許容されにくくなる．
- 仮想的な長さ 0 のパス上（$p=0$，$B(0)=n$）では，同様に，10% ずれた場合，$C(0.1,0) = n$，最もずれた場合，$C(0.5,0) = n^5$ となる．ずれ d に対して，同じく指数関数的に増加はするものの，$n < N$ であるだけ増分は抑えられ，ずれが許容されやすくなる．

M は，非クリティカル・パスにおいて，N の大きさの効果を緩和する．**図 6** に，$B(p)$ のグラフを示す．M を 1 より大きくすると，p が 1（クリティカル・パス）よりわずかに短くなるだけで $B(p)$ が大きく減少することになる．結果，クリティカル・パスよりわずかに短いパス上において，中央からのずれが許容されやすくなり，逆により大きい N の採用を可能にする．

なお，(1) 個数と (2) 遅延の，いずれを優先すべきかは対象によって異なるため，ユーザがパラメータを適切に調整する必要がある．上記の評価関数の場合，パラメータ N，n，M のうち，一次的には N によって調整することができる．N を大きくすれば，(2) 遅延がより優先される．

4.4　実験結果

表 2 に結果を示す．同表中，ベースは，前処理を施さずにエドモンズ・カープのアルゴリズムを適用した結果である．Rocket については，最大のステージの結果のみを示す．

まず，当然のことではあるが，提案手法では逆方向カット・エッジ数が 0 となっている．

実行時間は，Rocket の場合でも，ベースの 1.5 倍程度にとどまっている．表 2 に示した Rocket のステージは，全ステージの中でも突出して巨大なもの（除算器の一部）で，その他のステージはすべてその 1/10 程度以下にすぎない．

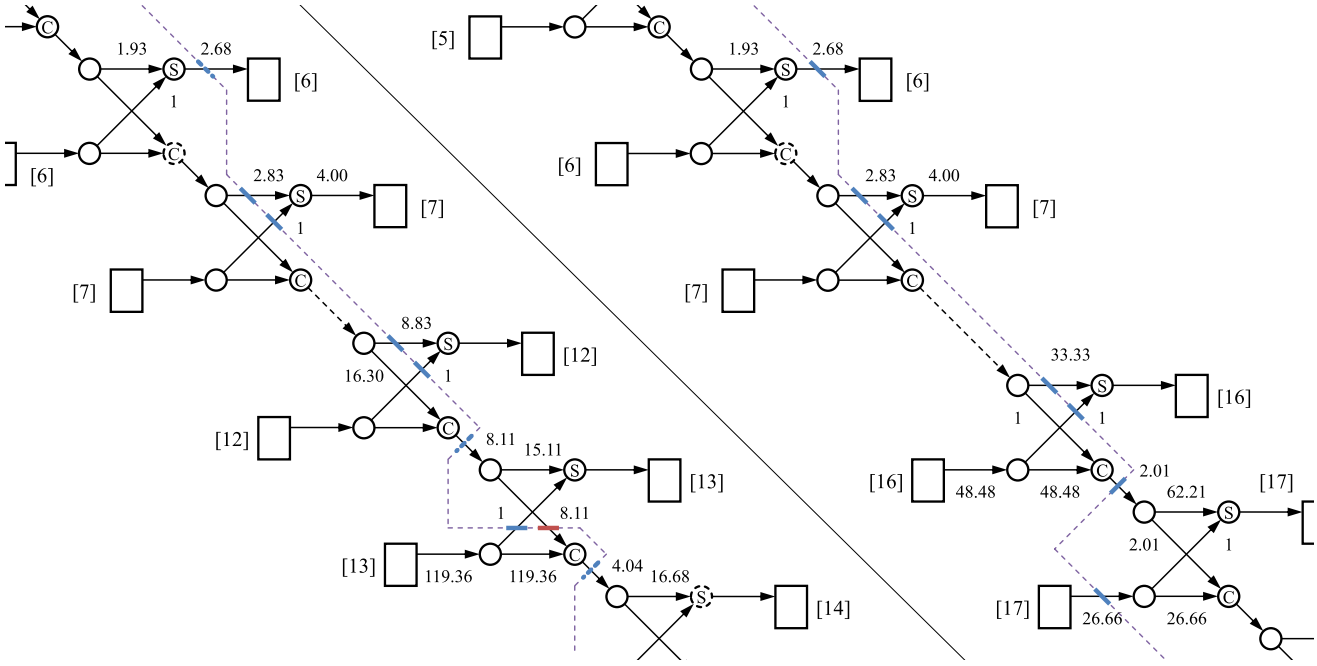

図 7　ベース（左）と提案手法（右）によって得られたカウンタのフロー・ネットワーク
Fig. 7　Flow networks of counter found by base maximum flow (left) and proposed algorithms (right).

表 2　実験結果
Table 2　Experimental results.

	カウンタ		Rocket	
	ベース	提　案	ベース	提　案
頂点数	190		34,347	
エッジ数	281		96,795	
順方向カット・エッジ	45	43	6,318	8,301
逆方向カット・エッジ	6	0	3,287	0
カット・サイズ	146.7	262.8	9,642.0	12,503.7
実行時間 [sec]	0.0143	0.0147	240.0	375.3

このような巨大なステージに対しても，約 375 秒という実用的な時間で最適解が求められることが分かる．

4.5　実行時間に関する考察

前述したように，実行時間は，Rocket の場合でもベースの 1.5 倍程度にとどまっている．3.8 節で述べたように，エドモンズ・カープのアルゴリズムの実行時間 $O(VE^2)$ は逆方向エッジを考慮に入れたもので，逆方向エッジを追加する前処理によって実行時間は $2^2 = 4$ 倍にはならない．

特にプログラムにおいては，この逆方向エッジは，必要に応じて追加/削除するのではなく，あらかじめすべてのエッジに逆方向エッジを追加したうえで，その容量を 0 に初期化することによって実現することになろう．この場合，提案手法における前処理とは，逆方向エッジを追加するのではなく，この初期値を 0 から ∞ に変更するにすぎない．実際，今回のプログラムでは，そのように実装されている [13]．

3.4 節で述べたように，提案手法ではよりサイズの大きいカットを探すことになる．そのためには，より多くの増加道を探索する必要がある．実際の実行時間の差は，このために生じると考えてよい．すなわち，ベースの最大フロー・アルゴリズムと提案手法の実行時間の差は，オーダ上のものではなく，トポロジ的には同一のフロー・ネットワークにおける容量の違いによるものである．

4.6　カウンタの詳細な結果

図 7 に，カウンタに対してベースのエドモンズ・カープのアルゴリズムと提案手法を適用して得られた回路のフロー・ネットワークを示す．同図中，四角は FF を，Ⓢと Ⓒは，部分和とキャリーを求める部分回路を，◯は複数出力のためのダミー（1 章参照）を，それぞれ表す頂点である．カウンタであるため，左右，対となる FF は，物理的には同一の FF に対応する．そして，破線は求められたカットを表す．

双方において，第 0〜6 ビットまでは，Ⓢの出力側に，カット・エッジがあるのに対して；第 7 ビット以降では，Ⓢの（出力側ではなく）入力側にカット・エッジがある．これは，下位からのキャリーの遅延が徐々に長くなり，第 7 ビット以降では，入力側に 2 つのラッチを置いた方が容量の和が小さくなるためと考えてよい．

ベースでは，第 13（図中）と 14〜18 ビット（図外），計 6 本の逆方向エッジが現れ，クリティカル・パスであるキャリー・チェーン上に複数のラッチが挿入される．

提案では，逆方向エッジは現れず，キャリー・チェーン

上には，第 16 ビットの次にラッチがただ 1 つ挿入される．

5. おわりに

FF を用いた回路からラッチを用いた回路に変換する際等には，単一カット・エッジ制約，すなわち，始点から終点に至るすべての道にカット・エッジを 1 つ含むという制約を満たすカットを見つける必要がある．

本稿では，まず，単一カット・エッジ制約が，逆方向カット・エッジなし制約，すなわち，カットが逆方向カット・エッジを含まないことと等価であることを証明した．そのうえで，逆方向カット・エッジを含まない最小カットを求めるアルゴリズムを提案し，その正しさを証明した．

このアルゴリズムにおいて最もオーダが大きい部分は既存の最大フロー・アルゴリズムであり，提案アルゴリズムのオーダはこれより悪化することはない．

現実的な回路から生成されたグラフに対して提案手法を適用したところ，巨大な回路に対しても約 375 秒という実用的な時間で最適解が求められることが分かった．

謝辞　本研究の一部は，文部科学省科学研究費補助金 No. 16H02797 による．

参考文献

[1] Harris, D.: *Skew-Tolerant Circuit Design*, Morgan Kaufmann Publishers (2001).

[2] Jimbo, U., Yamada, J., Shioya, R. and Goshima, M.: Applying Razor Flip-Flops to SRAM Read Circuits, *IEICE Trans. Electron.*, Vol.E100-C, No.3, pp.245–258 (online), DOI: 10.1587/transele.E100.C.245 (2017).

[3] 神保　潮，山田淳二，五島正裕：動的タイム・ボローイングを可能にするクロッキング方式の適用 (2017). *cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming*（*xSIG 2017*）に採択．

[4] 神保　潮, 山田淳二, 五島正裕：動的タイム・ボローイングを可能にするクロッキング方式の適用, 情報処理学会論文誌コンピューティングシステム，Vol.10, No.2, pp.1–12（オンライン），入手先 ⟨http://id.nii.ac.jp/1001/00183237/⟩ (2017).

[5] Ford, L.R. and Fulkerson, D.R.: Maximal flow through a network, *Canadian Journal of Mathematics*, pp.399–404 (online), DOI: 10.4153/CJM-1956-045-5 (1956).

[6] Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C.: *Introduction to Algorithms*, The MIT Press (2009).

[7] Nagamochi, H. and Ibaraki, T.: Computing edge-connectivity in multigraphs and capcacitated graphs, *SIAM Journal on Discrete Mathematics*, Vol.5, pp.54–66 (1992).

[8] 京都大学永持研究室：研究成果 > アルゴリズムのデモ > 最小カット問題（オンライン），入手先 ⟨http://www-or.amp.i.kyoto-u.ac.jp/demo/MINCUT.html⟩.

[9] 津坂章仁，谷川祐一，広畑壮一郎，五島正裕，坂井修一：動的タイム・ボローイングを可能にするクロッキング方式の二相ラッチ生成アルゴリズム，情報処理学会研究報告，Vol.2014-ARC-211, No.9, pp.1–10（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009808089/⟩ (2014).

[10] Goldberg, A.V. and Tarjan, R.E.: A new approach to the maximum-flow problem, *J. ACM*, Vol.35, No.4, pp.921–940 (online), DOI: 10.1145/48014.61051 (1988).

[11] Edmonds, J. and Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems, *J. ACM*, Vol.19, No.2, pp.248–264 (1972).

[12] QuickGraph, Graph Data Structures and Algorithms for .NET (online), available from ⟨http://quickgraph.codeplex.com/⟩.

[13] Sharaiha, E.: Edmonds Karp in C# (online), available from ⟨http://gist.github.com/Eyas/7520781⟩.

[14] RISC-V Foundation: RISC-V Foundation | Instruction Set Architecture (ISA) (online), available from ⟨http://riscv.org/⟩.

[15] Asanović, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser, J., Izraelevitz, A., Karandikar, S., Keller, B., Kim, D., Koenig, J., Lee, Y., Love, E., Maas, M., Magyar, A., Mao, H., Moreto, M., Ou, A., Patterson, D.A., Richards, B., Schmidt, C., Twigg, S., Vo, H. and Waterman, A.: The Rocket Chip Generator, Technical Report UCB/EECS-2016-17, EECS Dept., UCB (online), available from ⟨http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html⟩ (2016).

[16] 広畑壮一郎，神原太郎，吉田宗史，倉田成己，五島正裕，坂井修一：動的タイム・ボローイングを可能にするクロッキング方式の適用手法の予備評価，先進的計算基盤システムシンポジウム SACSIS，ポスター (2013).

[17] 吉田宗史，広畑壮一郎，倉田成己，五島正裕，坂井修一：動的タイム・ボローイングを可能にするクロッキング方式の適用手法の評価，情報処理学会研究報告，Vol.2013-ARC-206, No.6, pp.1–13（オンライン），入手先 ⟨http://id.nii.ac.jp/1001/00094541/⟩ (2013).

神保 潮　（学生会員）

1990 年生．2013 年東京大学工学部電子情報工学科卒業．2015 年東京大学情報理工学系研究科電子情報学専攻修士．同年総合研究大学院大学複合科学研究科情報学専攻進学，現在に至る．コンピューティング・システムの研究に従事．2017 年 xSIG Poster Award 受賞．

五島 正裕　（正会員）

1968 年生．1992 年京都大学工学部情報工学科卒業．1994 年同大学大学院工学研究科情報工学専攻修士課程修了．同年より日本学術振興会特別研究員．1996 年京都大学大学院工学研究科情報工学専攻博士後期課程退学，同年より同大学工学部助手．1998 年同大学大学院情報学研究科助手．博士（情報学）．2005 年東京大学情報理工学系研究科准教授．2014 年国立情報学研究所教授，現在に至る．コンピューティング・システムの研究に従事．著書に「ディジタル回路」．2001 年情報処理学会山下記念研究賞，2002 年同学会論文賞受賞．IEEE 会員．本会シニア会員．