社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－AL－107 (8)
2006／7／3

負閉路探索手法の性能評価

石田 勉† 小平 行秀† 高橋 篤司†

† 東京工業大学 理工学研究科 集積システム専攻 〒152-8550 東京都目黒区大岡山 2-12-1-S3-58
E-mail: †{tsutomu,yukoh,atsushi}@lab.ss.titech.ac.jp

あらまし クロックの入力タイミングを制御することで高性能な回路実現を目指す準同期式回路では，回路の制約グラフに負閉路が存在するとき，クロックスケジュールをどのように定めても回路は正常に動作しない．制約グラフの各枝重みはクロック周期により変化するため，制約グラフに負閉路が存在しない最小のクロック周期が，回路が動作可能な最小クロック周期となる．回路が動作可能な最小クロック周期は回路設計において繰り返し計算されるので，効率的な負閉路探索手法が必要となる．本研究では，負閉路探索手法として知られる改良ベルマン-フォード法とビットスケーリング法，どちらが回路の制約グラフに有効であるか性能評価を行なった．実験により，改良ベルマン-フォード法はビットスケーリング法より高速に制約グラフ中の負閉路を発見することを確認した．

Performance Evaluation of Negative Cycle Detection Algorithms

Tsutomu ISHIDA†, Yukihide KOHIRA†, and Atsushi TAKAHASHI†

† Department of Communications and Integrated Systems, Tokyo Institute of Technology
Ookayama 2-12-1-S3-58, Meguro-ku, Tokyo, 152-8550 Japan
E-mail: †{tsutomu,yukoh,atsushi}@lab.ss.titech.ac.jp

Abstract In semi-synchronous circuits, clock input timings are controlled in order to improve the performance. Clock input timings of a circuit that realize the correct circuit functionality exist if and only if the constraint graph of the circuit contains no negative cycle. Since the edge weight of the constraint graph of a circuit depends on the clock period, the minimum clock period in which the constraint graph contains no negative weight cycle is the minimum feasible clock period of the circuit. Since the minimum feasible clock period of a circuit is calculated repeatedly in circuit synthesis, an efficient negative cycle detection algorithm is requested. In this paper, we evaluate two negative cycle detection algorithms, enhanced Bellman-Ford algorithm and scaling algorithm, in terms of the efficiency to the constraint graph of benchmark circuits. In experiments, we show that the enhanced Bellman-Ford algorithm is more suitable for the constraint graph of benchmark circuits than the scaling algorithm.

1. はじめに

回路中の記憶素子に同時にクロックを分配する完全同期方式は，人間が理解しやすく，また設計も容易であるため，現在の集積回路におけるクロック分配方式の主流となっている．しかし，完全同期方式の下での性能追求は限界に達しつつあり，また，その前提を維持するためのクロックバッファ数や消費電力などといったコストも非常に大きくなっている．その一方で，クロックを回路中の記憶素子に同一周期ではあるが同時に分配することを前提としない準同期方式 [1], [2] は，物理遅延を完全同期方式よりも効率良く利用できる可能性があるため，クロック周期，クロック面積，消費電力，信頼性などさまざまな指標で計られる性能を低コストで実現することが期待されている．

準同期方式の回路設計では，回路の動作条件を制約グラフで表現し，その回路の最小クロック周期を求める．あるクロック周期に対し，制約グラフに負閉路が存在しなければ回路はそのクロック周期で動作可能であるので，2分探索手法を用い，負閉路が存在しない最小のクロック周期を求める．最小クロック周期は回路設計において繰り返し計算されるので，制約グラフ中に負閉路が存在するかを調べる効率的な手法が必要となる．

グラフ中に負閉路が存在するか否かは，準同期式回路の最小クロック周期と関連するだけでなく，それ以外の様々な問題とも関連するため，いままでに負閉路探索問題に対する数多くの手法が提案されている．なお，グラフ中の枝重みの正負をすべて反転すると正閉路は負閉路に負閉路は正閉路になる．負閉路探索問題は正閉路探索問題として扱われることもあるが本質的には両者は同じである．文献 [3] では，様々な負閉路探索手法が紹介されその性能が比較されており，ビットスケーリング法

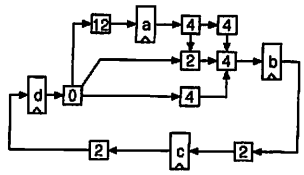

図 1　回路グラフ A

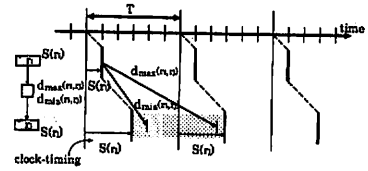

図 2　同期回路のタイミングチャート

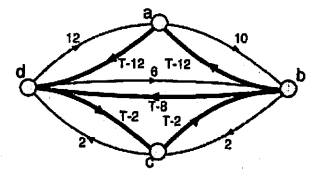

図 3　制約グラフ $H(A)$

の効率が良いと結論されている．しかし，負閉路探索手法の性能は対象とするグラフの性質にも大きく依存すると考えられる．実際，文献 [4] では，単同期式回路の最小クロック周期を求める際に回路の制約グラフに対し改良ベルマンフォード法を用いて負閉路を探索している．

本研究では，文献 [3] で効率が良いとされているビットスケーリング法と，単同期式回路の最小クロック周期を求める際に用いられている改良ベルマンフォード法の回路の制約グラフに対する性質を比較した．ベンチマーク回路を用いた実験により，改良ベルマンフォード法が回路の制約グラフに対する負閉路探索手法として相応しい手法であることを確認した．

以降，2 章では単同期式回路，3 章では負閉路探索手法について説明し，4 章で実験結果を示し，5 章で結論を述べる．

2. 準同期式回路

本稿では，回路はレジスタ，論理ゲート，それらをつなぐ配線から構成されるとし，回路をグラフ $A=(V_a,E_a)$ として表現する．V_a はグラフの点集合で，入出力ピン，レジスタ，ゲート，配線に対応する．E_a はグラフの有向枝の集合で，信号伝搬に対応する．ある点 $v\in V_a$ の重みを $d(v)$ と表し，対応する回路の要素の遅延値とする．

回路のレジスタの集合を V_r とする．明らかに，レジスタの集合は回路グラフの点の部分集合である $(V_r\subseteqq V_a)$．回路グラフの例を図 1 に示す．$\{a,b,c,d\}$ がレジスタ，点の中の数字が遅延値を表す．レジスタの遅延値も点の重みとして表現できるが，本稿では説明を簡略にするためレジスタの遅延値を 0 とする．

準同期方式では，各レジスタのクロック周期は同一であるが，クロック入力は同時に分配されるとは限らない．レジスタ r のクロックタイミングを $S(r)$ と表す．同期回路が動作するための条件は，信号が伝搬する全てのレジスタ対が以下の 2 式を満たすことある [1] (図 2)．

Setup 条件：0 クロック制約

$$S(r_i)-S(r_j)\leqq T-d_{\max}(r_i,r_j)$$

Hold 条件：二重クロック制約

$$S(r_j)-S(r_i)\leqq d_{\min}(r_i,r_j)$$

ここで，T はクロック周期であり，$d_{\max}(r_i,r_j),d_{\min}(r_i,r_j)$ はそれぞれレジスタ r_i と r_j 間の最大遅延，最小遅延である．

現在主流の回路方式である完全同期式回路では，全てのレジスタに同じタイミングで同じ周期のクロックを与えることを前提とする．したがって，最小クロック周期はレジスタ間の最大遅延以上であると判断する．一方，準同期式回路では，クロック周期が最大遅延よりも小さい場合にも全ての制約式を満たすのであれば回路は正常に動作すると正確に判断する．

それぞれのレジスタに任意のタイミングでクロックが入力できるという仮定の下で，回路 A が動作可能な最小クロック周期を $T_S(A)$ とする．以後，この $T_S(A)$ を単に回路 A の最小クロック周期と呼ぶ．

単同期方式の最小クロック周期 $T_S(A)$ は，回路 A から得られる制約グラフ $H(V_r(A),E_r(A))$ によって定まる [2]．$H(V_r(A),E_r(A))$ の点集合 $V_r(A)$ は，レジスタの集合 V_r に対応する．$H(V_r(A),E_r(A))$ の有向枝集合 $E_r(A)$ は，2 つの制約式に対応する．レジスタ r_i から r_j への有向枝は，Hold 条件に対応し，D 枝と呼ばれる．その枝の枝重み $l(r_i,r_j)=d_{\min}(r_i,r_j)$ である．レジスタ r_j から r_i への有向枝は，Setup 条件に対応し，Z 枝と呼ぶ．その枝の枝重み $l(r_j,r_i)=T-d_{\max}(r_i,r_j)$ である．以後，制約グラフ $H(V_r(A),E_r(A))$ を簡単に $H(A)$ と表記する．また，制約グラフの Z 枝はクロック周期 T の関数であり，クロック周期 $T=t$ とした制約グラフを $H(A,t)$ と表記する．グラフにおいて，枝重みの和が負の閉路を負閉路，0 の閉路を 0 閉路という．

[定理 1]（[2]）　単同期方式の最小クロック周期 $T_S(A)$ は，制約グラフ $H(A,t)$ が負閉路を持たない最小の t である．

例として図 1 に示される回路 A について考える．レジスタ間の最大遅延が 12 なので，完全同期方式の最小クロック周期は 12 となる．回路 A の制約グラフ $H(A)$ を図 3 に示す．図において，制約グラフの Z 枝，D 枝はそれぞれ，黒色，灰色で表されている．制約グラフ $H(A,9)$ を図 4 に示す．$H(A,9)$ に負閉路はなく，閉路 (a,d,b) のみ 0 閉路である．また，閉路 (a,d,b) は $H(A,t)(t<9)$ で負閉路となるので，回路 A の準同期方式の最小クロック周期 $T_S(A)$ は 9 となる．クロック周期 9 を実現した準同期式回路の例を図 5 に示す．

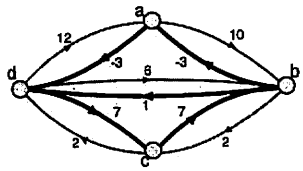

図 4　制約グラフ $H(A, 9)$

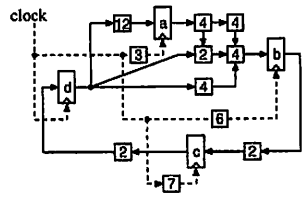

図 5　回路 A のクロック周期 9 での実現例

単同期方式の回路設計において，最小クロック周期を，線形計画法を解くことで求める手法 [1] と，制約グラフを用いて求める手法 [2], [4]〜[8] とがある．本稿では，大規模な回路に対しても適用可能な制約グラフを用いて最小クロック周期を求める手法を用いる．最小クロック周期の計算の効率は負閉路探索の効率に依存するため，効率的な負閉路探索手法が求められている．そこで，負閉路探索手法として効率が良いとされている改良ベルマン-フォード法とビットスケーリング法，2 つの手法の性能を比較し評価する．

3. 負閉路探索手法

この章では，改良ベルマン-フォード法 [4] とビットスケーリング法 [9] についてそれぞれ解説する．

3.1 改良ベルマン-フォード法

負閉路探索手法の多くはグラフ中の距離を求める手法を利用している．グラフ中の 2 点間の距離を 2 点間の最短パスの長さ (枝の重み和) と定義したとき，グラフ中に負閉路が存在しなければ 2 点間の距離を求める多項式時間のアルゴリズムが存在するが，負閉路が存在すると 2 点間の距離を求める問題は NP 困難となる．一方，2 点間の距離を点や枝の重複を許した 2 点間の最小ウォークの長さと定義すると，有限の枝重みを持つ連結グラフに対して，グラフ中に負閉路が存在しなければ 2 点間の距離は有限値となるが，負閉路が存在するとマイナス無限大となる．

枝重みがすべて非負の場合には上記の 2 つの距離の定義に違いはなく，2 点間の距離はダイクストラ法を用いて求めることができるが，負重みの枝を含む場合にはダイクストラ法を適用することはできない．ベルマン-フォード法は枝重みが負の枝を含む場合にも後者の定義で距離を求める．ベルマン-フォード法で求めた 2 点間の距離がすべて有限であればグラフ中に負閉路は存在しないことが確認でき，マイナス無限大となる場合には負閉路の存在が確認できる．

ベルマン-フォード法はフェーズ単位で距離ラベルの更新操作

```
procedure 改良ベルマン-フォード法 (G(V, E))
 1.  choose r ∈ V, Q1 ← ∅, Q2 ← ∅.     /* init */
 2.  for all v ∈ V let d(v) := ∞.     /* distance from r */
 3.  d(r) := 0, push r to Q1.
     /* update procedure (step 4-15) */
 4.  while Q1 is not empty do
 5.    u ← pop(Q1).
 6.    for all v adjacent to u in G do
 7.      if d(v) > d(u) + w(u, v) then
           /* labeling operation (step 8-10) */
 8.        d(v) := d(u) + w(u, v).
 9.        set parent pointer from u to v.
10.        push v to Q2 if v is not in Q2.
11.      endif
12.    endfor
13.    if r is in Q2 return "No".    (negative cycle exists)
14.    if reach u by following parent pointer from u
                 return "No".    (negative cycle exists)
15.  endwhile
16.  if Q2 is empty return "Yes".    (no negative cycle exists)
17.  Q1 ← Q2, Q2 ← ∅, goto step 4.
end
```

図 6　改良ベルマン-フォード法

を繰り返し，距離ラベルの更新が行なわれなくなったとき終了する．まず，最初のフェーズでは始点の距離ラベルを更新し，それ以降のフェーズでは，距離ラベルの更新されたすべての点に対して隣接点の距離ラベルの更新操作を行なう．グラフ中に負閉路が存在しなければフェーズを高々点数回繰り返すことで距離ラベルの更新は行なわれなくなりベルマン-フォード法は終了する．一方，フェーズを点数回繰り返して更新操作が終了しないとき，グラフ中には負閉路が存在し更新操作は永遠に続くことになることが分かるため，ベルマン-フォード法は終了する．グラフの点数を n，枝数を m としたとき，ベルマン-フォード法の各フェーズの計算量は $O(m)$ であり，全体の計算量は $O(mn)$ である．

ベルマン-フォード法を負閉路探索手法として適用すると，グラフ中に負閉路が存在しないとき，負閉路が存在しないことを高々点数回のフェーズの繰り返しで確認し，多くの場合，繰り返し回数は点数よりもはるかに小さく計算時間も大きくはならない．一方，グラフ中に負閉路が存在するとき，ちょうど点数回フェーズを繰り返す．実際の回路に対する制約グラフの点数は数万のオーダーになることも珍しくないため，制約グラフに負閉路が存在する場合，点数回のフェーズを繰り返すベルマン-フォード法の計算時間は非常に大きくなる．そこで，負閉路を早い段階で見つけるためベルマン-フォード法を改良した手法が，改良ベルマン-フォード法である．

改良ベルマン-フォード法は，前点ラベルを用いて負閉路を発見する．ベルマン-フォード法は，各フェーズで，前のフェーズで距離ラベルが更新された点を経由し，その隣接点に至る経路の長さが隣接点の距離ラベルより小さいとき，その隣接点の距離ラベルを更新する．改良ベルマン-フォード法では，距離

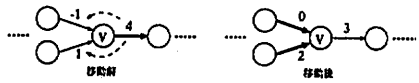

図 7　枝重み移動

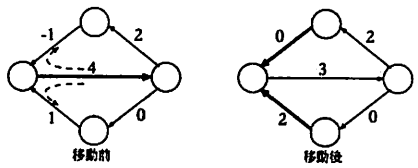

図 8　枝重み移動前後の閉路の重み和

ラベルが更新された点からその前点ラベルを辿る操作を加える．前点ラベルを辿ると始点に到達するか，もしくは自分に戻る．ある更新点から前点ラベルを辿ったとき，自分に戻ったらグラフ中に負閉路が存在することを意味するため，改良ベルマン-フォード法は負閉路を発見して終了する．一方，全ての更新点から前点ラベルを辿り，全て始点に到達したら距離ラベルの更新操作を続ける．また，始点の距離ラベルが更新される，つまり始点に負の距離がついたときもグラフ中に負閉路が存在することを意味するため，負閉路を発見して終了する．図 6 に文献 [4] にある改良ベルマン-フォード法のアルゴリズムを示す．グラフの点数を n，枝数を m，任意の 2 点間の最小トレイルの枝数の最大値を k としたとき，改良ベルマン-フォード法の各フェーズの計算量は $O(m+kn)$ であり，フェーズは高々 k 回繰り返されるので，全体の計算量は $O(km+k^2n)$ である．

3.2　ビットスケーリング法

ビットスケーリング法は，正重みを持つ枝から負重みを持つ枝に枝重みを移動させ，グラフ中の負重み枝を消せないとき，負閉路を発見する．負閉路の有無を調べたいグラフを $G=(V,E)$，G の枝重みを移動したグラフを G' とする．ここでは，ビットスケーリング法がどのような操作を経て G 中の負閉路を発見するかについて詳しく解説する．

まず，枝重みの移動方法について解説する．ある点に関する移動量 a の枝重みの移動では，その点の全ての出力枝の枝重みを a だけ減少させ，全ての入力枝の枝重みを a だけ増加させる．図 7 は点 v に対する移動量 1 の枝重み移動前と移動後の各辺の重みを示す．辺 (u,v) の枝重みを $l(u,v)$ とし，点 v に対する枝重み移動量を $s(v)$ としたとき，枝重み移動後の辺 (u,v) の枝重みは $l(u,v)-s(u)+s(v)$ となる．また，図 8 より分かるように，枝重み移動前と移動後では，閉路内の枝重みの総和は変わらない．G' の全ての枝の枝重みが 0 以上になるよう枝重みを移動させることができるとき，枝重み移動前のグラフ G 中に負閉路は存在しない．

ビットスケーリング法では，グラフ G' の全ての枝重みをビット単位の精度 (bit) で表す．bit を G' 中の負の最小枝重みの絶対値を超えない最小の 2 のべき乗としたとき，辺 (u,v) の枝重みは以下のように表される．

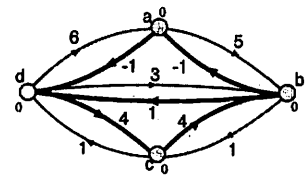

図 9　$H(A,9)$ の変換後 ($H_2(A,9)$)

```
procedure REFINE(G'_bit)
  1.  k ← the number of improvable nodes.
  2.  repeat
  3.    DECYCLE(G_p).
  4.    find S.
  5.    CUT-RELABEL(S).
  6.    k ← the number of improvable nodes.
  7.  until k = 0.
  8.  return(G'_bit).
end
```

図 10　REFINE

$$l_{bit}(u,v)=\left\lceil \frac{l(u,v)}{bit} \right\rceil$$

G' を変換したグラフを G'_{bit} とする．このとき，グラフ G'_{bit} の全ての枝の重みは -1 以上となる．前章の回路 A の制約グラフ $H(A,9)$(図 4) に対して，グラフの枝重みを変換をしたグラフ $H_2(A,9)$ を図 9 に示す．

REFINE は，枝重みが -1 以上となるように上で述べた方法で枝重みを変換したグラフ G'_{bit} を入力とし，枝重み -1 の枝を全て解消するように枝重みを移動するか，負閉路を発見して終了する．REFINE により入力グラフ G'_{bit} に負閉路が発見されたとき，枝重み変換前のグラフ G' の対応する閉路は負閉路である．枝重み -1 の枝を全て解消できたとき，G' では全ての枝の枝重みが $-bit$ より真に大きくなるように枝重みは移動されたことを意味する．bit 精度を上げ，最終的に $bit=1$ のとき REFINE が枝重み -1 の枝を全て解消できると，G' の枝重みも全て 0 以上となり，グラフ G に負閉路が存在しないことが分かる．REFINE の効率的な実装は図 10 になる．このような REFINE は計算量 $O(nm)$ で動く．

REFINE の DECYCLE(G_p) と CUT-RELABEL(S) サブルーチンを解説する．ここで，G'_{bit} の枝重みが $0,-1$ である枝の集合を E_p とし，$G_p=(V,E_p)$ を着目グラフとする．枝重みが $0,-1$ である枝の G' における枝重みは 0 以下であり，枝重みが正である枝の G' における枝重みは正である．

DECYCLE(G_p) は，まず G_p 中の強連結成分を見つける．ある強連結成分に枝重み -1 の枝が含まれるとき，G_p には負閉路が存在する．これは G' でも負閉路が存在することを意味し，ビットスケーリング法は負閉路を発見したとして終了する．全ての強連結成分が枝重み 0 の枝で構成されるとき，各強連結成分を構成する点集合をそれぞれ 1 点に縮退する．縮退により G_p は森になる．この G_p に対して CUT-RELABEL(S) を適

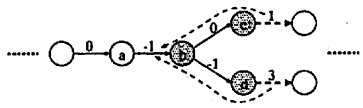

図 11 CUT-RELABEL(S) 前

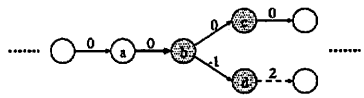

図 12 CUT-RELABEL(S) 後

```
procedure ビットスケーリング法 (G(V, E))
  1.  calculate bit.
  2.  while bit ≧ 1 do
  3.    for all a ∈ E let  l_bit(a) = ⌈l(a)/bit⌉
  4.    REFINE(G'_bit).
  5.    bit := bit/2.
  6.  endwhile
  7.  return "No".   (no negative cycle exists)
end
```

図 13 ビットスケーリング法

用する．

G_p 中で枝重みが -1 である枝の出力点を改善可能点（improvable node）とする．CUT-RELABEL は G_p 中のある改善可能点を選択し，その点から自分自身を含み，G_p 上で辿りつける全ての点の集合を S とする．CUT-RELABEL(S) は，集合 S 中の全ての点の枝重み移動量 s を 1 とし，G'_{bit} に対し枝重み移動を実行する．CUT-RELABEL に関して以下の補題が証明されている．

[補題 1]（[9]），CUT-RELABEL(S) の実行後，S によって定義されるカット上の枝重み -1 の枝の重みは 0 となり，G'_{bit} 中に枝重みが負の枝を新たに作らない．

つまり，CUT-RELABEL(S) は S 中の点に枝重み移動量 s を与えることで，グラフ G'_{bit} 中の正重みを持つ枝から負重みを持つ枝に枝重みを移動させている．例えば図 11 のグラフを考える．実線は G_p に含まれる枝，点線は含まれない枝を表す．枝 (a, b) に着目すると，$S = \{b, c, d\}$（色つきの点）となり，CUT-RELABEL(S) を実行して得られるグラフを，図 12 に示す．

以上のことをまとめると，ビットスケーリング法は図 13 の操作を行って負閉路を発見する．グラフ G の点数を n，辺数を m，負の最小枝重みの絶対値が N のとき，ビットスケーリング法のある bit での計算量は $O(nm)$ であり，それが $\log_2 N$ 回繰り返されるので，全体の計算量は $O(nm \log N)$ となる．

4. 実験結果

2 つの負閉路探索手法の効率を調べるため，Grid グラフ

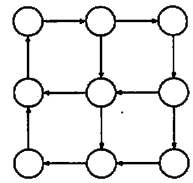

図 14 Grid グラフ（Grid3）

グラフ	負閉路の枝数	改良 BF 法 [sec]	BS 法 [sec]
Grid50	6	>0.001	0.004
	（上一段） 100	0.002	0.005
	（外周） 196	0.003	0.005
Grid70	6	>0.001	0.009
	（上一段） 140	0.006	0.010
	（外周） 276	0.009	0.010
Grid100	6	> 0.001	0.019
	（上一段） 200	0.022	0.021
	（外周） 396	0.036	0.022

表 1 負閉路探索時間（グラフサイズ，負閉路枝数）

グラフ	負閉路の要素	改良 BF 法 [sec]	BS 法 [sec]
Grid70	（1 本） −1	0.009	0.010
	（275 本） 0		
	（1 本） −6	0.009	0.048
	（5 本） 1		
	（270 本） 0		
	（1 本） −11	0.009	0.081
	（10 本） 1		
	（265 本） 0		

表 2 負閉路探索時間（負閉路枝重み）

と ISCAS89 ベンチマーク回路の制約グラフに対して，それぞれを適用する実験を行った．計算機は 3.40GHz/1024KB Intel Pentium-4 CPU，1GB RAM の PC を使用し，gcc3.5.5 の C++で実装した．

まずは，Grid グラフを用いてそれぞれの負閉路探索手法の特徴を調べる．実験で用いた Grid グラフの枝は基本的に，行方向は左向き，列方向は下向きとした．ただし，一番上の行は右向き，左端の列は上向きとした．また，Grid グラフの各枝重みは，特に指定が無い限り 1 とする．一辺の点数が n の Grid グラフを Grid$n$ で表す．図 14 に Grid3 示す．

表 1 は，Grid グラフの大きさ，負閉路を構成する枝数を変えたときの負閉路発見までの計算時間である．改良 BF 法は改良ベルマン-フォード法，BS 法はビットスケーリング法を表す．負閉路は枝重み -1 の枝一本と枝重み 0 の枝で構成し，負閉路外の枝重みは 1 とした．この実験からは，改良ベルマン-フォード法は負閉路を構成する枝数によって計算時間が変わり，ビットスケーリング法は負閉路を構成する枝数は関係なく，グラフの大きさによって計算時間が変わるということが分かる．

表 2 は，Grid70 に対し，負閉路を構成する枝数を 276 本（外周）とし，負閉路を構成する枝の枝重みを変え，負閉路探索時

回路	点数	枝数	クロック周期			OK	NG	改良 BF 法	BS 法
			(L.B.)	(Min.T)	(U.B.)			[sec]	[sec]
s3384	184	3876	147000	154000	168000	7	7	0.01	0.25
s38417	1637	68462	60000	61000	85000	8	7	0.30	6.83

表 3 最小クロック周期の計算時間

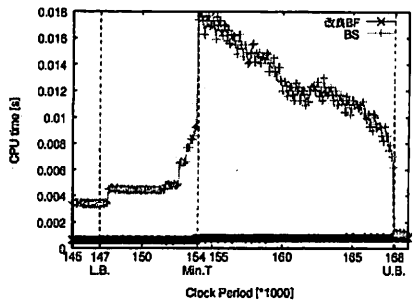

図 15 制約グラフ H (s3384) の負閉路探索時間

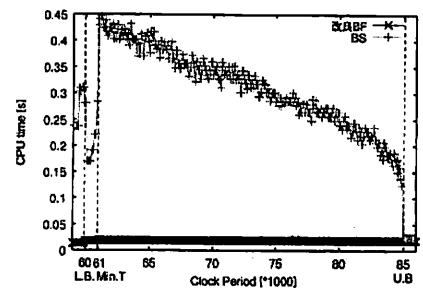

図 16 制約グラフ H (s38417) の負閉路探索の時間

間を調べた結果である．負閉路外の枝重みは 1 とした．この実験からは，改良ベルマン-フォード法の計算時間は負閉路を構成する枝の枝重みに関係なく，ビットスケーリング法は枝重みの移動回数が増えるため，負閉路を構成する枝の枝重みが計算時間に大きく影響することが分かる．

次に，ベンチマーク回路の制約グラフに対して，それぞれの負閉路探索手法を適用する．表 3 は，それぞれの負閉路探索手法が各回路の最小クロック周期を求めるまでの時間を測定した結果である．L.B. は 2 分探索法の下限，U.B. は上限，Min.T は最小クロック周期を指す．また，OK は負閉路が存在しない制約グラフを負閉路探索した回数，NG は負閉路が存在する制約グラフを負閉路探索した回数である．全ての回路において，改良ベルマン-フォード法の方が早く最小クロック周期を求めている．より詳しくクロック周期と負閉路探索時間の関係を解析するため，クロック周期 T の値を変えたときの各手法の負閉路探索の実行時間を図 15，16 に示す．クロック周期が Min.T 未満の場合に制約グラフに負閉路が存在し，Min.T 以上の場合に負閉路は存在しない．改良ベルマン-フォード法は負閉路のある無しに関わらず高速に実行されるが，ビットスケーリング法は改良ベルマン-フォード法より実行時間が大きく，特に負閉路が存在しない場合にその傾向は顕著になる．

5. まとめと今後の課題

実験により，改良ベルマン-フォード法を用いることにより回路の制約グラフ中の負閉路を早く発見できることが分かった．そのため，回路の最小クロック周期も改良ベルマン-フォード法により高速に求められる．これは，回路の制約グラフが少ない枝数から負閉路を構成していることが要因としてあげられる．また，ビットスケーリング法と改良ベルマン-フォード法で，負閉路探索の時間が大きく違うことから，制約グラフ中に大きな負重みの枝が数多く存在することも分かる．

今後，本稿で性能を評価した 2 つの負閉路探索手法以外の手法について評価すること，および新たな負閉路探索手法を開発することが課題である．特に，最小クロック周期を 2 分探索で求める際に，制約グラフの構造は変化せず枝重みのみ変化するため，前回の負閉路探索結果を利用することでより高速に負閉路を発見することを可能にし，さらに効率良く最小クロック周期を求める手法の開発を目指す．

文 献

[1] J. Fishburn, "Clock Skew Optimization," IEEE Tranc. on Computers, vol.39, no.7, pp.945–951, 1990.

[2] A. Takahashi and Y. Kajitani, "Performance and Reliability Driven Clock Scheduling of Sequential Logic Circuits," ASP-DAC'97, pp.37–43, 1997.

[3] B. Cherkassky and A. Goldberg, "Negative-Cycle Detection Algorithms," Proc. 4th European Symp. on Algorithms, pp.349–63, 1996.

[4] A. Takahashi, "Pracitical Fast Clock-Schedule Design Algorithms," 第 18 回 軽井沢システムワークショップ論文集, pp.515–520, 2005.

[5] T. Szymanski, "Computing Optimal Clock Schedules," Proc. 29th Design Automation Conference (DAC), pp.399–404, 1992.

[6] R.B. N. Shenoy and A. Sangiovanni-Vincentelli, "Graph Algorithms for Clock Schedule Optimization," Proc. International Conference on Computer-Aided-Design (ICCAD), pp.132–136, 1992.

[7] R. Deokar and S. Sapatnekar, "A Graph-Theoretic Approach to Clock Skew Optimization," Proc. International Symposium on Circuits and Systems (ISCAS), pp.407–410, 1994.

[8] J.S. C. Albrecht, B. Korte and J. Vygen, "Cycle Time and Slack Optimization for VLS-chips," Proc. International Conference on Computer-Aided-Design (ICCAD), pp.233–238, 1999.

[9] A.V. Goldberg, "Scaling Algorithms for the Shortest Paths Problem," SIAM J. Comput., vol.24, no.3, pp.494–504, 1995.