数理解析研究所講究録
1093 巻 1999 年 99-104

可変マージ関数の否定数限定複雑さ

佐藤 貴之　　天野 一幸　　丸岡 章
Takayuki Sato　　Kazuyuki Amano　　Akira Maruoka
仙台電波工業高等専門学校　　東北大学大学院情報科学研究科
taka@info.sendai-ct.ac.jp　　{ama|maruoka}@maruoka.ecei.tohoku.ac.jp

1 はじめに

P vs. NP 問題は, 理論計算機科学の分野において解決しなければならない最重要な未解決問題である. この問題を論理回路という計算モデル上で考えたとき, Razborov[Raz] は, NP に属するクリーク関数を否定ゲートを用いない単調論理回路で計算するには, 多項式より多いゲート数が必要であることを示した. しかし, 否定ゲートを含む一般の論理回路において, ある関数を計算するのに多項式より多いゲート数が必要であることを示す糸口が全くつかめない状態であることは今も変わらない.

そこで, 一般の論理回路について P vs. NP 問題を解決する為, 直接的に否定ゲートに制限を加えない論理回路を考えるのではなく, 否定ゲートの個数を制限したときの論理回路に関する研究を行い, その時の計算量や振る舞いなどを解析した後一般の論理回路について考察するという動きがおこり, 田中と西野ら [TN],[BNT] は, 与えられた n 変数論理関数を計算する NOT ゲートの使用を $\lceil \log_2(n+1) \rceil$ 個に制限した論理回路は NOT ゲートを無制限に許した同じ関数を計算する論理回路のサイズの 2 倍と $O(n \log n)$ を加えたサイズで計算可能であることを示した. これは, 一般の論理回路における計算量を解析する時, NOT ゲートの個数をほぼ $\log n$ 個に制限した論理回路の計算量を解析すれば良いということを意味している. この結果の証明の主要な部分は, 入力列 $x_1, \ldots, x_n$ が与えられた時, これらの否定 $\bar{x}_1, \ldots, \bar{x}_n$ を出力する反転器を NOT ゲートの個数を $\log n$ 個に制限して $O(n \log n)$ サイズで構成した点である. 彼らの構成した回路は, 入力をソートする単調論理回路, ソート済みの列を反転する否定ゲートを用いた回路, ここで得られた列をもとの入力の否定列に正しく並べかえる単調論理回路の 3 つの部分よりなっている. ここで, もし, NOT ゲートの使用を $\log n$ 個に制限した時に, ソート関数を計算するサイズが $o(n \log n)$ の論理回路が存在するならば, より小さなサイズの反転器の構成につながる可能性が高いと考えられる.

ソート関数とそれに類似するマージ関数における結果として, マージ関数については, Lamagna[Lam] が, 単調論理回路でマージ関数を計算するのに $\Omega(n \log n)$ 個のゲートが必要である事を示し, ソート関数の NOT ゲートを制限しない場合については, Muller と Preparata[MP] が, 一般の論理回路で $O(n)$ 個のゲートでソート関数が計算可能であることを示した. この事からソート関数やマージ関数は否定ゲートを含まない単調論理回路と否定ゲートを含む一般の論理回路では回路計算量が異なることがわかっている. したがって, ソート関数やマージ関数を, 制限された個数の否定ゲートを用いた回路で実現するのに必要なサイズを考えることは, 否定ゲートの効果を考える上でそれ自身非常に興味深い.

そこで, 本稿ではソート関数の否定数限定複雑さについて取り上げる. 佐藤, 天野と丸岡 [SAM] は, $O(\log n)$ 個に制限したとき, 入力を k-tonic 列に制限したソート関数 (これを k-ソート関数と呼ぶ) は線形サイズの論理回路で計算できることを示した. さらに, 垂井 [Tar] の問題提示により, 本稿では否定ゲートの個数を $O(\log \log n)$ 個に制限した k-ソート関数について解析することにし, 否定ゲートの個数を $O(\log \log n)$ に制限したとき, 入力系列に同様の制限を加えることによって, 線形サイズで計算可能である事を示した.

本稿では k-ソート関数を計算する論理回路を構成する為に, 入力系列の 1 の個数の 2 進数表現の上位 $\log \log n$ 桁とその $\log \log n$ 桁では取りきれないビット 1 を含む長さ $\frac{n}{\log n}$ の 01 系列を出力させるカウンターを構成し, Muller と Preparata のデコーダーを $\log \log n$ 桁入力に適したものに構成し直した. カウンターの構成にあたり, 入力系列をいくつかのブロックに分割し, 1 の個数を数えることが可能なブロックを選び 1 の個数の 2 進数表現を正しく出力しながら 1 ブロックずつ減らしていくという方針を取ることとした. 本稿の構成は以下の通りである. まず第 2 章で基本的な定義を述べ, 第 3 章において k-ソート関数の定義を述べ, その上界を与える. 第 4 章では, それから導かれる系などについて述べる.

2 諸定義

入力端子に変数, または, 定数が割り当てられた, 2 入力 AND ゲート, 2 入力 OR ゲート, NOT ゲートからなる非周期的な回路を**論理回路**という. NOT ゲートが現れない論理回路を, 特に, **単調論理回路**という.

$w=\{w_1,\ldots,w_n\}$, $w'=\{w'_1,\ldots,w'_n\}$ に対し，任意の $i=1,\ldots,n$ で $w_i\leq w'_i$ が成立するとき $w\leq w'$ と書く．n 変数論理関数 $f:\{0,1\}^n\to\{0,1\}$ が，任意の $w\leq w'$ なる $w,w'\in\{0,1\}^n$ に対して $f(w)\leq f(w')$ を満たしているとき，f を**単調論理関数**という．任意の単調論理関数は単調論理回路で計算可能であり，かつ，単調論理回路で計算可能な関数は単調論理関数のみである．論理回路 C に含まれるゲートの個数を C の**サイズ**といい，$\mathrm{size}(C)$ と表す．論理関数 f に対して，f を計算する最小のゲート数の論理回路のゲート数を f の**複雑さ**といい，$\mathrm{size}(f)$ と表す．また，単調論理関数 f に対して，f を計算する最小のゲート数の単調論理回路のゲート数を f の**単調複雑さ**といい，$\mathrm{size}_{mon}(f)$ と表す．正整数 t と論理関数 f に対して，NOT ゲートを高々 t 個しか含まない論理回路のうち f を計算するもので最小サイズのもののゲート数を f の**否定 t 限定複雑さ**，あるいは単に否定限定複雑さといい，$\mathrm{size}_t(f)$ と表す．ただし，f が否定ゲート数 t 個以下の論理回路では計算不能の場合には，この値は定義されないものとする．0,1 系列 $u=u_1u_2\cdots u_t$ に対して，$|u|$ は u のビット長 (この場合は t である) を，$(u_1u_2\cdots u_t)$ は u を 2 進数として見た時の値，すなわち $u_12^{t-1}+u_22^{t-2}+\cdots+u_t$ を表すものとする．本論文で用いる対数 log の底は全て 2 である．

3 k-ソート関数を計算する論理回路の設計

3.1 k-ソート関数

k-ソート関数は入力を k-tonic 列に制限したソート関数である．k-tonic 列は変極点の個数により定義する．

定義 1 (変極点)
$x\in\{0,1\}^n$ において x の各ビットを先頭から順に見たとき，0 から 1, または 1 から 0 に変わる場所を **変極点** という．変極点の個数が高々 k 個の列 $x\in\{0,1\}^n$ を k-tonic **列** と呼ぶ．$x\in\{0,1\}^n$ の 1 の個数を $\sharp_1(x)$ と書き，x に含まれる変極点の個数を $\sharp_t(x)$ と書く．

本稿で取りあげるのは，ソート関数に新たに k-tonic 列の概念を導入した k-ソート関数である．k-ソート関数とは，入力が k-tonic 列の時に限り，入力の列を降順にソートして出力する関数である．厳密に定義すると，以下の通りである．

定義 2 (k-ソート関数)
$SORT_n^k:\{0,1\}^n\to\{0,1\}^n$
入力： $x_1,\ldots,x_n$.
出力： $y_1,\ldots,y_n$ は $x_1,\ldots,x_n$ が k-tonic のとき，$x_1,\ldots,x_n$ の並べかえで，$y_1\geq\cdots\geq y_n$ を満たす．
入力 $x_1,\ldots,x_n$ が k-tonic を満たさないときは，出力に何の条件も課さない．
なる関数を **k -ソート関数** と呼ぶ．

本稿の主定理は以下の通りである．

定理 1 定数 c が存在して，$size_{k^2c\log\log n}(SORT_n^k)=O(k^2n)$. □

$SORT_n^k$ を計算する論理回路を図 1 の様に構成する．
まず入力列 x を COUNT_n^k に与える．この回路では x に含まれる 1 の個数を数えて，1 の個数の 2 進数表現の上位 $(\log\log n+\log k)$ 桁の列 u と，その 2 進数表現で取りきれなかった 1 を含む長さ $\frac{n}{\log n}$ の 01 系列 s を出力する．次に，COUNT_n^k の出力を，Muller と Preparata[MP] の手法の様に直接デコーダーに接続する事は出来ないため，一旦 COUNT_n^k の出力 u と s を REG_n^k に入力し 1 の個数の 2 進数表現の上位 $\log\log n$ 桁 v と長さ $\frac{n}{\log n}$ ビットの列 t に調整する必要がある．そして，REG_n^k の出力 v と t を DECODE_n に入力することによって，ソート済みの出力列 y を生成することができる．

以下混同の恐れの無い場合には，論理関数とこれを計算する回路を同じ記号で表す．図 1 中の $COUNT_n^k$, REG_n^k, $DECODE_n$ の厳密な定義は以下の通りである．

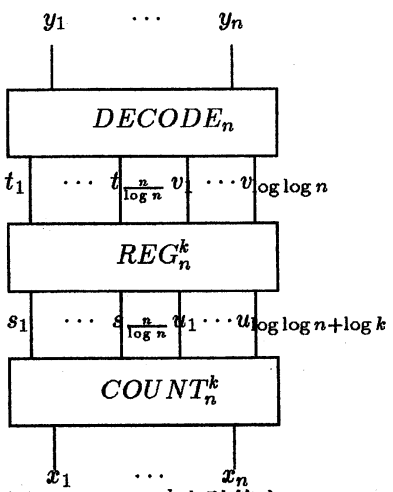

図 1 : $SORT_n^k$ を計算する論理回路

Input : 長さ n の k-tonic 列 x
Outputs : 長さ $\frac{n}{\log n}$ のビット列 s と，長さ $\log\log n+\log k$ のビット列 u
begin
　$u=0$; x^1=x;
　for $i:=1$ to $\log\log n$ **do begin**
　　$\mathrm{COUNT}^i(x^i)$;
　　$u=u+2^{1-i}c^i$;
　end
　$s=x^{\log\log n+1}$;
end.

図 2: COUNT_n^k を計算するアルゴリズム

定義 3 (kカウント関数)
$COUNT_n^k:\{0,1\}^n\to\{0,1\}^{\frac{n}{\log n}+\log\log n+\log k}$
入力： $x=x_1,\ldots,x_n$
出力： x が $k-$ tonic のとき，出力列 s と 2 進表現された 0,1 系列 $u=u_1,\ldots,u_{\log\log n}$ は，$\sharp_1(x)=\sharp_1(s)+(u_1u_2\cdots u_{\log\log n+\log k}0\cdots0)$ を満たす．ただし，x が k-tonic でないときは出力を制限しない．
この関数を kカウント関数といい，この関数を計算する論理回路をカウンターと呼ぶ．

定義 4 (kレギュレート関数)
$REG_n^k:\{0,1\}^{\frac{n}{\log n}+\log\log n+\log k}\to\{0,1\}^{\frac{n}{\log n}+\log\log n}$
入力： 長さ $\frac{n}{\log n}$ の 0,1 系列 s と，長さ $\log\log n+\log k$ の 0,1 系列 u
出力： 長さ $\frac{n}{\log n}$ の 0,1 系列 t と，長さ $\log\log n$ の 0,1 系列 v
ただし，$t\in1^*0^*$，かつ，$\sharp_1(t)+(v_1v_2\cdots v_{\log\log n}0\cdots0)=\sharp_1(s)+(u_1u_2\cdots u_{\log\log n+\log k}0\cdots0)$ を満たす．
この関数を kレギュレート関数といい，この関数を計算する論理回路をレギュレーターという．

定義 5 (デコード関数)
$DECODE_n:\{0,1\}^{\frac{n}{\log n}+\log\log n}\to\{0,1\}^n$
入力：長さ $\frac{n}{\log n}$ の列 $t\in1^*0^*$ と $v=v_1,v_2,\ldots,v_{\log\log n}$
出力：$y=y_1,y_2,\ldots,y_n$ は $y_1\geq\cdots\geq y_n$ を満たし，かつ，$\sharp_1(t)+(v_1v_2\ldots v_{\log\log n}0\cdots0)=\sharp_1(y)$ を満たす．
この関数をデコード関数といい，デコード関数を計算する論理回路をデコーダーと呼ぶ．

3.2 カウンターの設計

k カウント関数については以下が成立する．

定理 2 c :定数が存在して $\mathrm{size}_{k^2c\log\log n}(COUNT_n^k)=O(k^2n)$. □

証明 (概略)
COUNT_n^k を計算するアルゴリズムを図 2から図 4に示す．定理の条件を満たす論理回路は，このアルゴリズムを自然な形で論理回路上に実装することにより得られる．
COUNT_n^k(図 2) を計算するには COUNT^i(図 3) を $\log\log n$ 回再帰的に計算する．COUNT^iの出力列の長さは入力列の長さの半分になることに留意すると，最終的に $\log\log n$ 回再帰をかけることにより列の長さは $\frac{n}{\log n}$ となりこれが定義 3中の s にあたる．その COUNT^iで得られた $\log k+1$ 桁の 1 の個数の 2 進表現を 1 桁ずつずらして加え，出力した $\log\log n+\log k$ ビットの列が u となる．
COUNT^iはその内部で $\mathrm{COUNT}^{i,j}$(図 4) を k回実行する．ここで $\mathrm{COUNT}^{i,j}$を計算するとき求める $\mathrm{ExistClean}^{i,j}$ の状態によって COUNT^iの出力列も異なってくる．COUNT^iの入力列の長さを N として，$\mathrm{ExistClean}^{i,j}$とは，長さ

Input : 長さ $n/2^{i-1}$の k-tonic 列 x^i
Outputs : 長さ $n/2^i$の k-tonic 列 x^{i+1}と，長さ $\log k+1$ の列 c^i
ただし，$\sharp_1(x^i)=\sharp_1(x^{i+1})+(c^i0\cdots0)$.

```
procedure COUNT^i(x^i)
begin
    c^i = 0; x^{i,1} = x^i;
    for j := 1 to k do begin
        COUNT^{i,j}(x^{i,j}, CP^j);
        if ExistClean^{i,j} = 1 then begin
            if j = k then begin
                x^{i+1} = P;
                c^i = CP^{k+1};
            end
            else x^{i,j+1} = P;
        end
        else begin
            x^{i+1} = Q;
            c^i = CQ;
            return;
        end
    end
end.
```

図 3: COUNT^iを計算するアルゴリズム

を$\frac{N}{2k}$のブロックに分割した時, 先頭の k個のブロックの中の変極点があるかを調べる関数である. $\mathrm{ExistClean}^{i,j}=1$ であれば変極点が 1 つもないブロックが存在し, $\mathrm{ExistClean}^{i,j}=0$ であれば (入力が k-tonic 列であることより) 分割した全てのブロックの変極点の個数が高々1 個であること (この列を bitonic 列を呼ぶ) が容易にわかる. しかも, この 2 つの状況以外は起こり得ないことに注意する. $\mathrm{ExistClean}^{i,j}=1$ の時は, 変極点の存在しないブロックを入力列から削除し, その代わり入力列の 1 の個数を保存するため 1 ビット用意し, それを 1 の個数の 2 進数表現にあたる c^iに加える. 削除後の列が図 3においては P にあたる. よって 1 の個数を正しく保存しながら入力列の長さを減らしていく事が可能となる. それを高々k回帰納的に繰り返すことにより合計 kブロックを削除することができるので入力系列の長さを COUNT^iの入力の半分の長さにして出力することができる.

また $\mathrm{ExistClean}^{i,j}=0$ の時は,bitonic selection を実行する. bitonic selection とは, 長さ $2l$の列 $z=z_1,\ldots,z_{2l}$ に対し, $a=1,\ldots,l$について z_aと z_{a+l}を比較し大きい方のビットを左に置くように交換することをいう. bitonic selection の性質として,zが bitonic 列のとき, zに bitonic selection を実行した列を z'と置くと, z'の左半分が 1 かまたは z'の右半分が 0 になることがわかる. すなわち各ブロックが bitonic 列のときは bitonic selection を行うことでブロックの左半分か右半分を 1 の個数を保存しながら削除することが可能であることを意味する. その各ブロックの 1 の個数を保存する為に準備したビットを全て加えることによって得られた数が図 3の CQ である. また各ブロックが全て半分の長さまで削除することが出来るから一気に削除を COUNT^iの入力の半分の長さまで行った列が図 3においては Q に相当する.

図 2から 4までを論理回路で実現した時, NOT ゲートは図 2中の u の $\log\log n$ 回の計算, 図 3中の $\mathrm{Clean}^{i,j,a}$の $k^2\log\log n$ 回の計算, 図 4の OPEA を実行するか OPEB を実行するかの $k^2\log\log n$ 回の分岐,OPEA,OPEB においてのブロックの削除, カウンタの計算 (共に $k^2\log\log n$ 回) で使用されている. このゲート数を合計すると COUNT_n^kで使用されている NOT ゲートの個数は $O(k^2\log\log n)$ であることがわかる. また, 合計のゲートの個数も $O(k^2n)$ であることがわかる. □

Input : 長さ $\frac{2k+1-j}{2k}N$ の列 $J^{i,j}$ と，長さ $\log k+1$ ビットのカウンタ CP^j
Outputs : 長さ $\frac{2k-j}{2k}N$ の列 $P^{i,j}$ と，長さ $\log k+1$ ビットのカウンタ CP^{j+1}，
長さ $\frac{1}{2}N$ の列 $Q^{i,j}$ と，長さ $\log k+1$ ビットのカウンタ CQ
procedure $COUNT^{i,j}(J^{i,j})$
begin
$J' = \{x_1^{i,j}, \ldots, x_{2^{-i_n}}^{i,j}\}$;
$J'' = \{x^{i,j}$ から J' を除いた列 $\}$;
for $a := 1$ to k **do begin**
$J'(a) = \{J^{i,j}_{\frac{a-1}{2k}N+1}, \ldots, J^{i,j}_{\frac{a}{2k}N}\}$;
$AND^{i,j,a} = (J'(a)$ の全てのビットの AND);
$OR^{i,j,a} = (J'(a)$ の全てのビットの OR);
$Clean^{i,j,a} = (AND^{i,j,a} = OR^{i,j,a})$;
end
$ExistClean^{i,j} = \bigvee_{a=1}^{k} Clean^{i,j,a}$;
if $ExistClean^{i,j} = 1$ **then** OPEA$(x^{i,j}, CP^j)$
else OPEB$(x^{i,j}, CP^j)$;
end.

procedure OPEA$(x^{i,j}, CP^j)$
begin
for $a := 1$ to k **do begin**
if $Clean^{i,j,a} = 1$ **then begin**
$P' = \{J'(a)$ 以外の $J'\}$;
$P = P' \circ J''$;
$CP^{j+1} = CP^j + ((J'(a)$ の先頭のビット$)$;
end
end
end.

procedure OPEB$(x^{i,j}, CP^j)$
begin
$Q' = \emptyset$;
$CQ = 2CP^j$;
for $a := 1$ to $2(k-j+1)$ **do begin**
$Q'(a) = \{x^{i,j}_{\frac{a-1}{2k}N+1}, \cdots, x^{i,j}_{\frac{a}{2k}N}\}$;
$L(a) = (Q'(a)$ に対して bitonic selection を実行したもの$)$;
$\mathrm{Half}^{L(a)} = \bigwedge_{b=1}^{\frac{N}{4k}} x^{i,j}_{\frac{a-1}{2k}N+b}$;
if $\mathrm{Half}^{L(a)} = 0$ **then** $Q'(a) = L(a)$ の左半分
else $Q'(a) = L(a)$ の右半分;
$Q' = Q' \circ Q'(a)$;
$CQ = CQ + \mathrm{Half}^{L(a)}$;
end
end.

図 4: COUNT^{i,j} を計算するアルゴリズム

NOT ゲートの個数	MERGE	$SORT_n^k$	SORT
0	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
$c \log \log n$	$\Theta(n)^*$	$O(k^2 n)^*$	?
$c \log n$	$\Theta(n)$	$O(kn)$	?
n	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$

表 1: SORT と MERGE のサイズの対応表

レギュレータ, デコーダーについては以下の事を容易に示すことができる.

補題 1 ある定数 c が存在して, $\mathrm{size}_{c \log \log n}(REG_n^k) = O(n)$. □

定理 3 $\mathrm{size}_{mon}(DECODE_n) = O(n)$. □

ゆえに, 定理 2,3, 補題 1 より, 定理 1 を満たす論理回路 $SORT_n^k$ が構成できる.

4 おわりに

定理 1 を証明したことにより, 次の系が得られる.

系 1 k が定数のときある定数 c が存在して, $\mathrm{size}_{c \log \log n}(SORT_n^k) = O(n)$. □

これは NOT ゲートの個数を $O(\log \log n)$ に制限したとき, 定数 -ソート関数は $O(n)$ 個のゲートで構成可能であることを意味する.

以上マージ関数とソート関数の否定数限定複雑さに関して得られた結果を表 1に示す. *は本稿で得られた結果である.

使用できる NOT ゲートの個数を $c \log n$ 個に限定したとき, ソート関数が線形サイズで計算可能かどうかについては依然未解決である. しかし,NOT ゲートの個数の制限を佐藤, 天野と丸岡 [SAM] の結果より厳しくしたときの結果を出したことにより,NOT ゲートの個数を $c \log n$ 個に制限した時 k-ソート関数は線形サイズで計算可能か否かという問題が解決される日が近いと考えている.

謝辞

最後に, 本稿を執筆するにあたり, 貴重な御意見をくださった電気通信大学の垂井淳先生に心より感謝いたします.

参考文献

[AKS] M. Ajtai, J. Komós and E. Szemerédi, "An $O(n \log n)$ sorting network", *Proc. 15th STOC*, pp. 1–9, 1983.

[BNT] R. Beals, T. Nishino and K. Tanaka, "More on the Complexity of Negation-Limited Circuits", *Proc. 27th STOC*, pp. 585–595, 1995.

[Lam] E. A. Lamagna, "The Complexity of Monotone Networks for Certain Bilinear Forms, Routing Problems,Sorting,and Merging", *IEEE Trans. of Comput.*, Vol. C–28, No. 10, pp. 773–782, 1979.

[MP] D. E. Muller and F. P. Preparata, "Bounds to Complexities of Networks for Sorting and Switching", *J. of ACM*, Vol. 22, pp. 195–201, 1975.

[Raz] A.A. Razborov, "Lower Bounds on the Monotone Complexity of Some Boolean Functions", *Soviet Math. Dokl.*, Vol. 281, pp. 798–801, 1985.

[SAM] 佐藤 貴之, 天野 一幸, 丸岡 章, "連数限定入力に対する否定数限定ソーティング回路", 信学技報, COMP97–120, Vol. 97, No. 628, 1998

[TN] K. Tanaka and T. Nishino, "On the Complexity of Negation-Limited Boolean Networks", *Proc. 26th STOC*, pp. 38–47, 1994.

[Tar] J. Tarui, *Personal Communication*, 1998.