計算機アーキテクチャ 110－6
設 計 自 動 化 73－6
（1995. 1. 19）

二分決定グラフを用いたテクノロジマッピングの位相最適化手法について

松永 裕介

（株）富士通研究所

〒211 川崎市中原区上小田中 1015

[概要] テクノロジマッピング処理には広く tree covering アルゴリズムが用いられているが、このアルゴリズムではマッピングを行なう前に入力部分の極性を固定する必要があり、全体として最適となる位相（極性）の割り当てを考慮することができない。本稿では入力部分の極性に応じて値の変わるコスト関数を導入することによって、位相最適化問題をコスト関数の最小化問題に定式化を行なう。そのコスト関数は、枝に重みを持つ二分決定グラフを用いることによって、非明示的に処理される。ベンチマークデータを用いた実験で 1 割程度、面積が減少することを確認した。

On Phase Optimization in Technology Mapping Using Binary Decision Diagrams

Yusuke Matsunaga

Fujitsu Laboratories LTD.

1015 Kamikodanaka, Nakahara-Ku, Kasawaki 211

[abstract] Though tree covering is an efficient algorithm for technology mapping, phase assignments on tree boundaries are not taken into consideration. This paper describes a new formulation of phase optimization problem. Cost function representing area according to each phase assignment is introduced, and tree covering algorithm is modified to handle that cost function. Edge-Valued Binary Decision Diagram is used to represent the function implicitly. Experimental results show that proposed method reduces about 10% area on average.

1 はじめに

論理式からゲートアレイやスタンダードセルなどの実際のゲート回路を自動的に作り出す論理合成の処理は大きく2つの段階に分けることができる。第一段階は、使用するセルライブラリ等のテクノロジ情報には非依存なレベルで回路の最適化を行なうもので、その後に実際のゲート／セルの情報を考慮に入れた合成／最適化を行なう処理が続く。論理合成処理をこのように分割することによって、前半のテクノロジ非依存なレベルの回路合成／最適化の処理の自由度が大きくなり、より大域的な最適化が行なえるようになっている。

反面、後半のテクノロジ依存のレベルでの合成／最適化処理（テクノロジマッピングと呼ぶ）に関してはそれほど統一的な枠組が完成しているとは言えない。最初に試みられた手法は LSS[1] や SOCRATES[2] のような局所変換を行なうルールベースシステムを用いたものであった。これらの手法の長所は、どのようなタイプのセルや複雑なコスト関数に対しても適当なルールを用いることで対応できるという柔軟性であるが、その反面、この点は新たなタイプのセルやコスト関数を導入する毎にルールを作らなければならないという短所にもなっている。また、これらの局所的変換では絶対的な最適性に対する大域的な視野を持つことが難しいので、最適化の性能自体にも限界があることが指摘されている。最近の動向からするとルールベースシステムを用いる手法はあまり有効ではないと言える[1]。

Keutzer はテクノロジ・マッピングの問題が DAG covering（サイクルを含まない有効グラフに対する被覆問題）として定式化できることを示した [4]。しかし、DAG covering はたとえそのグラフの節点の次数（出入りしている枝の数）が2に抑えられたとしても $\mathcal{NP}$ 困難問題であるので、実用規模の回路のマッピングにこの DAG covering を直接、適用することはできない。そこで彼は回路の中の fanout-free region（ただ一つの節点のみに出力している節点が作る木状の部分グラフ）に着目し、その木状の部分グラフに対しては線形時間で最適解を得ることのできるアルゴリズム（tree covering）を提案した。このアルゴリズムはダイナミック・プログラミングを応用したもので、ある節点における最適なマッピングがその節点の入力となっている節点の最適なマッピングから計算できるという性質を利用している。

この tree covering のアルゴリズムはその後 Detjens らの手によって改良され論理合成システム MIS に組み込まれている [5]。tree covering アルゴリズムの詳細は文献 [6] に述べられている。また Touati はこのアルゴリズムを基に回路の動作速度を最適化するテクノロジ・マッパーの開発を行なった [7]。

しかし、tree covering では回路を分割しているので、分割された範囲内のみの最適化しか行なえないという本質的な問題も生じる。本稿ではそのような問題の一つである、ファンアウト部分の位相をどのように決めるべきか、という問題に対する処理手法を提案する。以下、2章で tree covering アルゴリズムについて説明し、3章で位相最適化問題の定式化を行なう。4章でそのアルゴリズムの具体的な実装手段について述べる。

[1]ただし、多出力セルや EXOR ゲートのように tree covering ではうまく扱えないセルもあるので、ルールベースで前処理、あるいは後処理を行なう方法も提案されている [3]。

2 動的計画法を用いた tree covering の解法

ここでは DAG covering および tree covering について簡単に説明を行なう。

マッピングの対象となるのはブーリアン・ネットワーク[2]で、まず最初にそのネットワークがそれと機能的に等価な2入力 NAND（および NOT）で構成されたネットワークに置き換えられる。DAG covering ではこのネットワークの部分回路にマッチするセルを列挙し、そのマッチの集合でネットワークを最小コストで被覆する解を求めることが問題となる。

マッピング対象のネットワークを $G(V,E)$ とする。ここで、Vはネットワークの節点の集合、Eはその節点を結ぶ有向辺の集合とする。あるマッチ m に対して、そのマッチに被覆される節点の集合を $COVER(m)$、そのマッチの出力に対応する節点を $ROOT(m)$、そのマッチの入力に対応する節点の集合を $LEAF(m)$ とする。与えられたブーリアンネットワーク上でのすべてのマッチの集合を M_Gで表す。このマッチの部分集合 $M \subseteq M_G$がネットワーク G を被覆するとは、

$$\forall v \in V, \exists m \in M, v \in COVER(m) \tag{1}$$

が成り立つことを言う。マッチ m が解に含まれる時に真になる論理変数を X_mとし、節点 vを被覆するようなマッチの集合を $M_v = \{p_1^v, p_2^v, \ldots\}$ とすると、上の (1) 式を満たすマッチの集合を表す論理式は、

$$C1 = \prod_{v \in V} (X_{p_1^v} + X_{p_2^v} + \cdots) \tag{2}$$

で表される。この論理式を充足する（否定のリテラル$\overline{X_i}$がないのでこの論理式は必ず充足可能である。）変数の値の組合せのなかで、もっとも1となる変数の少ない組合せを求める問題（もしくは各々の変数ごとに設定された重みの和を最小にするような組合せを求める問題）は一般に最小被覆問題と呼ばれる。例えば、積和形論理式の簡単化問題もこの最小被覆問題に落すことができる。この問題は $\mathcal{NP}$ 問題であるが、比較的性能のよい近似手法が知られている（例えば文献 [9]）。

ところが、DAG covering の場合にはこの被覆条件だけでは十分ではない。マッチの結果として得られたセルのネットリストが実現可能 (feasible) でなければならないので、あるマッチ m が解に選ばれるのならば、その入力となる節点

[2]テクノロジ非依存の組合せ回路の表現形式。文献 [8] 参照

$u \in LEAF(m)$ を根とするような別のマッチも選ばれなければならない、という条件がさらに必要となる。

具体的には、

$$\forall m \in M, \forall u \in LEAF(m), \exists l \in M, ROOT(l) = u \quad (3)$$

となる。この条件は、節点 u を根に持つマッチの集合を $\{q_1^u, q_2^u, \ldots\}$ とすると、

$$C2 = \prod_{m \in M_G} \left(\prod_{u \in LEAF(m)} (\overline{X_m} + X_{q_1^u} + X_{q_2^u} + \cdots) \right) \quad (4)$$

という論理式で表される。以上をまとめると、DAG covering の条件となる論理式は、

$$C = C1 \wedge C2 \quad (5)$$

となる。問題は二項目の $C2$ である。ここには各和項中に1つの否定のリテラルが含まれているので、変数 X_i の値を $0 \rightarrow 1$ と変化させることによって $C2$ の値は $0 \rightarrow 1$ と $1 \rightarrow 0$ のどちらにも変化する可能性がある。このような論理関数を binate 関数と呼ぶ。逆にいかなる変数の値の変化に対して単調変化しかしない論理関数を unate 関数と呼ぶ。この関数の呼び名にちなんでこの問題は binate covering 問題とも呼ばれる。

この binate covering 問題は古くは'50 年代に不完全指定順序機械の簡単化問題を解くために研究されており、問題のサイズを小さくする簡約化アルゴリズムなどは知られているが、厳密解を求めることができるのは数十変数程度である。この DAG covering の場合には変数の数は $|M_G|$ となり[3]、低く見積もってもネットワークのサイズに比例するものと考えられるので、数千～数万ゲート規模の回路のマッピング問題をこの binate covering 問題として解くことは現実的には不可能である。ちなみに、元となるネットワークの最大ファンアウト数を2に限定してもこの問題の複雑さは変わらないことが明らかになっている。

そこで考え出されたのが、ネットワークをそのファンアウト数が1の部分グラフ (グラフ理論のいうところの tree) に分解し、その各々の部分グラフに対して最適なマッチの組合せを求める、という手法である [4]。図 1にそのようなネットワークの分解を示す。ファンアウト数が1の部分グラフを以後、ファンインブロックと呼び、ファンインブロックの間を結ぶファンアウト部分をファンアウトブロックと呼ぶことにする。

対象が DAG から tree に変わると動的計画法によってグラフのサイズに対して線形時間で最適解を求めるアルゴリズムが存在する。これは、ある節点 v を根とする部分木に対する最適解は、その部分木に含まれる節点（を根とする部分木）に対する最適解から計算できることによる。つまり、入力の節点から始めて自分を根とする部分木に対する最適解を計算してゆくことで、最終的に与えられた木全体の最適解が求められるというものである (図 2)。このネットワークの節点 v に対応した変数 C_v を用意し、その節点を根とする部分グラフを被覆する最小コストを表すものとする。外部入力に対応する節点 i のコスト C_i は 0 とする。

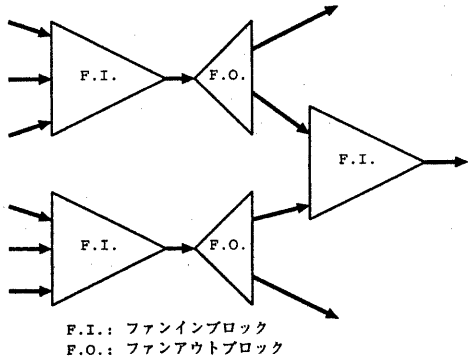

図 1: ネットワークの分解

```
tree_covering(Graph G) {
    G の各節点を入力から出力方向へトポロジカル順で整列
    for G の各節点 v に対して {
        C_v ← +∞;
        for v を根（出力）とするマッチ m に対して {
            c ← m のセル面積;
            for m の各入力に対応する節点 u に対して
                c ← c + C_u;
            C_v ← min(C_v, c);
        }
    }
}
```

図 2: tree covering のアルゴリズム

実際には、各節点は最小コストの他にそのコストを実現するマッチも保持しているので、上のアルゴリズムが終了した時点で最小被覆が得られることになる。

この tree covering はその木に対するマッピングとしては常に最適解を生成し、実用的にも高速なアルゴリズムであるが、回路全体を tree 状に分割することによって、いくつかの最適性は損なわれてしまう。具体的には、

- 分割された回路の境界を跨いだセル割り当てができない。
- 境界での位相が合わないと位相合わせのためのインバータが挿入されることになり全体では最適とならない。

という問題がある（図 3)。

[3]集合 S に対して、その要素数を $|S|$ で表す。

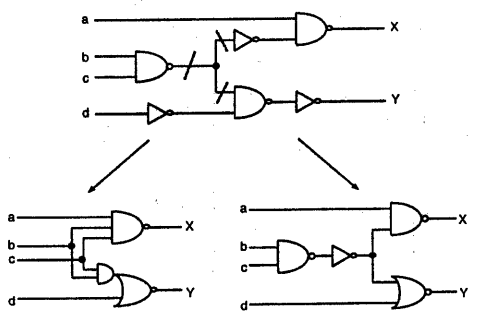

図 3: ファンアウトを跨いだセル割り当て

この図の例に対する適切な解は（セルライブラリにもよるが)、左下の回路例のように 3 入力 NAND と AND-OR-INV を一つづつ用いるものであるが、もしも、もととなったブーリアンネットワークを tree 上に分割してしまうとこのような結果を得ることはできない。さらにその分割点をどちらの極性で実現するかによって得られる回路は異なったものとなる。もしも面積を小さくしたいのであれば、右下の回路のようにもとのネットワークとは反対の極性で実現した方が良い解を得ることができる。本稿ではこのようなファンアウト部分での極性を考慮することによって、より面積の小さな回路を得るための手法について述べる。

3 位相最適化手法

位相最適化手法の最も簡単なものは、双対なテクノロジライブラリ[4]を用いると言う仮定の元で、必要となるインバータを最小化するものであるが、実はこの定式化にしても $\mathcal{NP}$ 完全問題となる。MIS[8] ではこの問題に対する 2 つの近似手法を提案している。Jain と Bryant は双対と言う制約のない一般のテクノロジライブラリに対するインバータ最小化問題を位相の制約グラフに対する 2 色の彩色問題として定式化を行なった [10]。

しかし、これらの手法は単にインバータの個数を最小にしているだけで、2 入力 NAND ゲートと 2 入力 AND ゲートを同一に考えているので、実際のセル面積を評価関数にしているわけではない。そこで本稿ではセル面積を評価関数とした位相最適化問題の定式化を行なう。

ここで述べるアプローチはおおまかには、上述の tree covering で用いるコストをスカラー値からベクトル値 (関数) に拡張し、各々の樹枝状部分回路に対して位相の組合せに対するコスト関数（セル面積）を求めてその総和を最小にする位相割当を求めるというものである。

複数のファンアウトを持つ節点で回路を分割した場合、その出力をそのままの極性で実現するのと否定した極性で実現するのでは、面積が異なってくる。しかし、これは一つのファンインブロックに閉じた話ではないので、上記の tree covering アルゴリズムでは全体的な最適性を考慮できない。そこで、図 2のコストをスカラ量から各境界での位相に応じて値の変化する関数に置き換えて、各位相の組合せにおける最適値を計算するように修正を行なう。

具体的には、各ファンアウトブロック FOT_i に対して 2 つの論理変数 P_i, Q_i を用意する。このうち、P_i は FOT_i の出力が肯定の極性で実現されている時に 0、否定で実現されている時に 1 となる。また、Q_i は FOT_i の出力に位相合わせのためにインバータが必要な時に 1 となる変数である。

つまり、

$$P_i = \begin{cases} 0 & : FOT_i\text{のソースを正極性で実現} \\ 1 & : FOT_i\text{のソースを負極性で実現} \end{cases} \quad (6)$$

$$Q_i = \begin{cases} 0 & : FOT_i\text{にインバータを挿入しない} \\ 1 & : FOT_i\text{にインバータを挿入する} \end{cases} \quad (7)$$

となる。

各節点におけるコストはこれらの変数を入力とする関数として表される (つまり、$C : \mathcal{B}^{2n} \rightarrow \mathcal{N}$、ただし n はファンアウトブロックの数、$\mathcal{N}$ は整数全体の集合とする[5])。

ファンインブロックに対して tree covering を行なう時、ファンアウトブロック FOT_i をソースとするような入力 PI_j のコストは次のように定義される。

$$C^+_{PI_j} = \begin{cases} 0 & : \text{if } P_i = 0 \vee Q_i = 1 \\ +\infty & : \text{if } P_i = 1 \wedge Q_i = 0 \end{cases} \quad (8)$$

また、PI_j の否定を実現する場合のコストは

$$C^-_{PI_j} = \begin{cases} 0 & : \text{if } P_i = 1 \vee Q_i = 1 \\ +\infty & : \text{if } P_i = 0 \wedge Q_i = 0 \end{cases} \quad (9)$$

で与えられる。

tree covering アルゴリズムに対する変更点は以下の通りである。

まず、スカラー値を扱う部分をコスト関数を扱うように拡張する必要がある。そのためにはコスト関数に対する加算と min 演算を用意すればよい。加算、min 演算ともに実際の要素に対する加算、min 演算を行なった結果を要素とするような関数をつくり出せばよい。

次に、根本（出力）の節点 v_{root} をカバーするマッチだけではなく、その否定をカバーするマッチも計算し、それぞれの最小コストをコストを $C^+_{v_{root}}, C^-v_{root}$ とする。するとこのファンインブロック FI_j のコスト関数は

$$C_{FI_j} = \begin{cases} C^+_{v_{root}} & : \text{if } P_i = 0 \\ C^-_{v_{root}} & : \text{if } P_i = 1 \end{cases} \quad (10)$$

となる。ただし、このファンインブロック FI_j の出力先のファンアウトブロックを FO_i とする。

[4] あるセルがあったら、その論理関数の双対な論理関数を実現するセルもそのライブラリに含まれているようなライブラリ。

[5] ここではセル面積が整数で与えられると仮定している。

コストはファンインブロックだけでなく、その他の部分にも設定される。

まずファンアウトブロック FO_iに対しては、

$$C_{FO_i} = \begin{cases} 0 & : \text{if } Q_i = 0 \\ \text{インバータの面積} & : \text{if } Q_i = 1 \end{cases} \quad (11)$$

のように設定される。つまり、ファンアウトのソースが実現している極性とシンクの要求している極性が異なる時 (i.e. $Q_i = 1$)、インバータの面積分のコストが余計にかかることになる。このインバータは特定のファンイン木に属しているわけではないので、ファンイン木の個別のコスト計算で考慮することはできない。

次に、外部入力に関してもコストが設定される。正確には、外部入力に関する制約をコストの形に表したもので、

$$\sum_{\forall i \in \{i | FO_i \text{のソースは外部入力}\}} C_{PI_i}$$

ただし、

$$C_{PI_i} = \begin{cases} 0 & : \text{if } P_i = 0 \\ +\infty & : \text{if } P_i = 1 \end{cases} \quad (12)$$

である。これは外部入力の極性があらかじめ決まっている[6]ことによる。

同様にして、外部出力に関する制約もコストの形に表される。

$$\sum_{\forall i \in \{i | FO_i \text{の出力先は外部出力}\}} C_{PO_i}$$

ただし、

$$C_{PO_i} = \begin{cases} 0 & : \text{if } P_i = 0 \vee Q_i = 1 \\ +\infty & : \text{if } P_i = 1 \wedge Q_i = 0 \end{cases} \quad (13)$$

以上のようにして求められたコストをすべて足し合わせれば回路全体のコストが計算できる。そのなかで最小値を与える P_i,Q_iの組合せを求めれば、それが位相最適化の答となる。

4 位相最適化アルゴリズムの実装

ここでは3章の定式化に基づいたアルゴリズムの実装についての検討を行なう。

4.1 コスト関数の表現法

コスト関数は上述の通り、$\mathcal{B}^{2n}$から整数への写像である。もちろん、一つ一つの入力値に対して陽に出力値を列挙することは現実的ではない。定義域がブール空間であるので二分決定グラフ（BDD[11]）の変種を用いることが考えられる。この関数に対して実際に必要な演算は、

加算：$C(X) = C_1(X) + C_2(X)$

min 演算：$C(X) = \min(C_1(X), C_2(X))$

最小値を取り出す演算：$x^* \leftarrow argmin_{x \in X} C(x)$

である。そこで、以下のように枝に重みが付いた二分決定グラフ（Edge Valued BDD）を提案する。このBDDは論理変数に対応するインデックスを持つ非終端ノードと、0と無限大（∞）を表す2つの終端ノードを持つ。通常のBDDと同じく、各非終端ノードはその変数を0および1に固定した時の関数を指す0-枝と1-枝を持つが、その枝は整数のラベルを持つ。そして、根本から0の終端ノードに到達するまでに通った枝のラベルの和がその経路（各変数の値の組合せに対応する）の値となる。また、入力の組合せによっては ∞ の終端節点に到達する経路も存在するが、その場合には途中のラベルに関わらず無限大となる。このようなBDDの例を図4に示す。図中で実線は1-枝を表し、破線は0-枝を表している。このBDDの表している関数は表1のようになる。

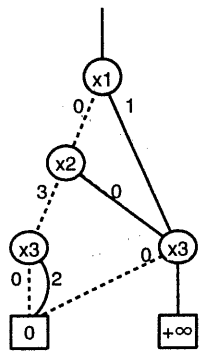

図 4: 枝に重みを持つ BDD

X_1	X_2	X_3	
0	0	0	3
0	0	1	5
0	1	0	0
0	1	1	∞
1	0	0	1
1	0	1	∞
1	1	0	1
1	1	1	∞

表 1: コスト関数の例

BDDには正規化ルールによって冗長なノードを取り除きカノニカルな表現となっているものもあるが、この枝に重みを持つBDDもいくつかのルールによってカノニカルな表現とすることが可能である。その正規化ルールは、

[6]もしも組合せ回路の入力がフリップフロップの出力ならば、肯定／否定どちらの極性も利用可能であるのでこの制約は不要となる。

1. 同一の関数を表すノードは一つにまとめる。これはBryantのROBDDと同一である。

2. 0-枝、1-枝が同一のノードを指し、かつ、同一のラベルを持っている時、そのノードを削除する。このルールも枝のラベルを比較することを除けばROBDDのルールとほとんど同一である。

3. 0-枝、1-枝のラベルをそれぞれ w_0、w_1とした時に、$w_0 < w_1$ であれば、$w_0^{new} = 0$、$w_1^{new} = w_1 - w_0$として、自分を指す枝のラベルに w_0を足す。逆も同様(図5)。この結果、枝のラベルは必ず非負となる。

4. 片側の枝が ∞ の終端ノードを指している場合には、他方の枝のラベルは0にして、その上のノードを指す枝のラベルにその値を足す(図6)。

というものである。

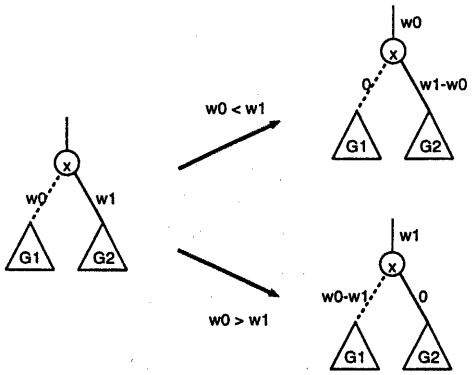

図5: 正規化ルール(3)

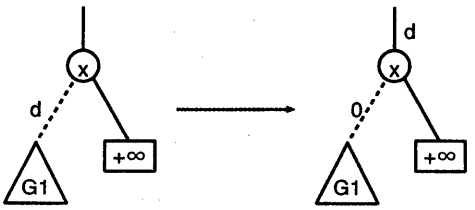

図6: 正規化ルール(4)

このように枝にラベルを導入し、正規化ルールを定義した理由は加算を効率良く行なうためである。図7の例のように互いにdisjointな変数に依存する2つのコスト関数の加算は、その変数順に従ってdisjointな変数の組を分けることが可能な場合には容易に行なうことができる。具体的には変数順の上の方の変数に依存するコスト関数の0を指す枝を他方のコスト関数を指すように変えるだけで2つのコスト関数の加算を行なうことが可能である。この場合、結果のBDDの節点数はもとのBDDの節点数の和で抑えることができる[7]。

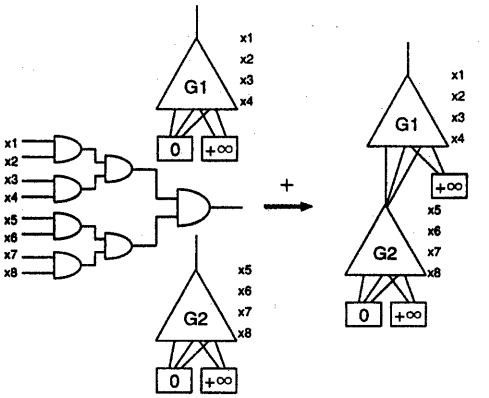

図7: 重みつきBDDの加算

さらに、あるコスト関数 f_1と f_2 の加算の結果を f_3とすると、

$$\begin{aligned}(f_1 + k_1) + (f_2 + k_2) &= (f_1 + f_2) + k_1 + k_2 \\ &= f_3 + k_1 + k_2 \end{aligned} \quad (14)$$

が成り立つので(ただし、k_1、k_2は定数とする)、2つのノードの間で行なわれた加算の結果を憶えておけば、それぞれのノードを指す枝に付いたラベルの値が変わってもその答を容易に作ることができる。これはBDD上での加算を行なうために必要な演算結果テーブルに登録すべき要素がノードだけでよく、それをさす枝のラベルは必要でないことを意味している。このことにより、加算のための演算結果テーブルをコンパクトにすることが可能であると同時に、加算の演算中に同様の演算結果が演算結果テーブルに登録されていたために実際に加算を行なわなくて済む回数が増えることになり、大きな効率化が期待できる。

また、最小値を与える入力の組合せを求めることは非常に容易である。このBDDの場合、ある節点の0-枝か1-枝の少なくともどちらか一方のラベルは0であり、他方は必ず非負なので、根から0のラベルを持つ枝をたどっていって0の終端ノードにたどり着けばその経路(に対応する入力の組合せ)が解となる。また、その最小値自体は根のノードを指す枝のラベルに等しい。

しかし、このような枝に重みのついたBDDでももう一つの演算、min演算に関しては特に有利となることはなく、他のBDDと同程度の手間が必要となる。

[7]一般に2つのBDDの間の演算結果のBDDのサイズは元のBDDのサイズの積で抑えられる。

4.2 位相最適化のための近似手法

前節で説明したBDDを用いても数百～数千ゲート規模の回路全体に対する位相最適化問題のコスト関数を表現することは難しい。そこで、いくつかの近似を行なうことが必要となる。

まず一点目はtree coveringを行なう際にBDDのノード数の爆発を抑えるためのもので、ファンインブロックの各節点に対するコスト関数を計算した後で、その時点での最小値（前に述べたように根のノードを指す枝に付くラベル値）よりも一定値以上大きな値を持つものをすべて∞に置き換えるというもので、この規定値を0に近くすれば入力が正極性・負極性のどちらも使用可能とした時の面積最小の解を求めるのと同じことになる。もちろん、その場合にはスカラー値で処理するよりもBDDを用いて処理したほうが処理の手間は増えることになる。

二点目は、全体のコスト関数として一つのBDDを構築するのではなく、各々のブロックに対するコスト関数をマージせずに保ったまま、処理を行なうというものである。具体的には、全体のコスト関数$C(P_1,Q_1,P_2,Q_2,\ldots)$が

$$C(P_1,Q_1,P_2,Q_2,\ldots)=\sum_i C_i(P_1,Q_1,P_2,Q_2,\ldots) \quad (15)$$

の形で与えられた時に実際に$C(P_1,Q_1,P_2,Q_2,\ldots)$を表すBDDを構築せずに最小値を与える変数の組合せを求めるものである。そのためには下に示すような処理を行なう。

分割処理 各々のC_iが依存している変数（サポートと呼ぶ）を求め、共通なサポートを持つ関数を一つのグループとする。

併合処理 各々のグループの中で、BDDのノード数の小さなものを選び併合可能ならば併合する（その2つの関数をC_i,C_jとすると、$C'=C_i+C_j$となるC'を計算する）。

探索処理 各々のグループの中で最小値を与える入力の組合せを求め、結果を一つにまとめる。異なるグループ間ではサポートが共通でないため、この解をまとめる時に矛盾は生じない。

まず、分割処理で行なっているように、元の関数が共通のサポートを持たない複数のグループに分割できる場合、全体の最適解は各々のグループの最適解をまとめたものに等しい。この処理はいかなる時にも有効である。もちろん、すべての関数が一つのグループに入ることもあり得る。

次の併合処理では2つのBDDを1つにすることができる場合には1つにまとめる。しかし、どの関数とどの関数を一つにするのか？どの程度まで併合を試みるのか？など、さまざまなヒューリスティックが考えられる。ここではまず最初に最もBDDのノード数の少ない関数を求め、その関数と共通のサポートを持つ関数のうち（このグループのすべての関数がこの関数と共通のサポートを持つとは限らない）最もノード数の少ない関数を選んで併合を試みている。

最後の探索処理で本当に探索を行なう。ここもいろいろとヒューリスティックが考えられるものであるが、現在はsimulated annealingのように乱数を用いて解を求めて改善する手法を用いている。

5 実験結果

前節の手法をいくつかのベンチマーク回路に適用した結果を表2に示す。元となった回路は論理合成ワークショップの多段回路のベンチマークでこれにバークレイの論理合成システムSISを用いて簡単化[8]したものを用いている。セルライブラリは富士通のCMOSゲートアレイ用のもので、結果の回路の面積評価には各々のセルに設定されているセル面積を用いている。例えばもっとも小さなインバータの面積は3.1となっている。使用計算機はSUN-4/10で、計算時間の単位は秒となっている。

表中で、方式1とあるのは、通常の方法でマッピングを行なうもので、ファンアウトブロックでの極性はその入力もとのファンインブロックの面積の小さくなる極性に応じて決められている。方式2が今回、提案した手法を用いたもので、どちらの方式の結果とも全体の面積とインバータのみの面積（カッコ内）を示している。なかにはあまり違いの見られないものや方式2の結果が悪くなっているものも見られるが全般的には1割程度、方式2のほうが小さくなっている。また、興味深いのがインバータの面積と全体の面積の関係である。ほとんどの回路ではインバータの面積の減少が全体の面積の減少と強く結び付いているが、中にはインバータの面積がほとんど変わらなかったり増えても全体の面積が減少しているものもあり、インバータ個数の削減が必ずしもそのまま全体の回路面積の削減につながらないことを意味している。

処理時間は方式1の場合に20秒から3分程度、方式2の場合には3分から6分程度であった[9]。現在用いているヒューリスティックはあまり効果的とは思えないので、より高速で効率の良い探索法を検討する必要があると思われる。

6 おわりに

テクノロジマッピングの位相最適化の手法についての検討、実験を行なった。処理時間は従来手法よりもかなりかかっているが、5%から10%程度、面積を削減している。しかし、数百～数千ゲート規模の回路に対してはこの問題を厳密に解くことは難しい。比較的速い時間でよい解を求めることのできる探索のヒューリスティックを見つけることが今後の課題である。また、複数の重みつき二分決定グラフで表されたコスト関数を最小にするような入力の組合せを求める問題は、ここで示したテクノロジマッピングの位相最適化以外にも応用が考えられるので、汎用の探索ヒューリスティックの開発が重要であると思われる。

[8] `script.rugged`を適用し、最後に`decomp -g`で細かなノードに分解した。

[9] 探索処理を3分間で打ち切っている。

表 2: 実験結果

回路名	方式 1			方式 2		
	全体の面積 (インバータの面積)		計算時間 (秒)	全体の面積 (インバータの面積)		計算時間 (秒)
9symml	530.4	(62.0)	21.0	492.6	(68.2)	243.0
C432	566.2	(105.4)	23.5	506.7	(74.4)	251.1
C499	840.2	(24.8)	26.4	845.6	(31.0)	236.8
C880	1048.6	(124.0)	28.4	1001.9	(102.3)	254.8
C1355	840.2	(24.8)	26.9	845.6	(31.0)	236.5
C1908	1030.3	(120.9)	30.5	983.2	(93.0)	246.3
C2670	1544.1	(241.8)	46.1	1356.2	(189.1)	272.8
C3540	3182.5	(368.9)	84.1	3035.2	(356.5)	356.2
C5315	3938.5	(669.6)	85.6	3729.9	(592.1)	308.6
C6288	7005.7	(1416.7)	113.7	6901.5	(1391.9)	351.7
C7552	4852.1	(821.5)	103.0	4601.4	(685.1)	316.7
apex6	2072.7	(424.7)	44.3	1813.1	(195.3)	280.2
apex7	710.5	(161.2)	18.8	619.5	(96.1)	238.8
b9	366.6	(93.0)	15.2	328.6	(65.1)	234.0
des	8870.9	(1500.4)	211.0	7758.5	(579.7)	356.9
f51m	239.3	(40.3)	13.3	243.5	(49.6)	235.3
rot	1928.9	(406.1)	38.3	1778.9	(313.1)	263.3
z4ml	103.3	(15.5)	12.5	103.3	(15.5)	222.6

参考文献

[1] J.Darringer,D.Brand,W.Joyner,and L.Trevillyan,"Lss: A system for production logic synthesis",*IBM J. Res. Develop.*,Sept. 1984.

[2] D.Gregory,K.Bartlett,A.de Geus, and G.Hachtel,"Socrates: A system for automatically synthesizing and optimizing combinational logic",in *23rd Design Automation Conference*, pp.79–85, IEEE/ACM, 1986.

[3] D.S.Kung, R.F.Damiano, T.A.Nix, and D.J.Geiger, "BDDMAP : a technology mapper based on a new covering algorithm", *29th Design Automation Conference*, pp.484–487, IEE/ACM, 1992.

[4] K. Keutzer, "DAGON:Technology Binding and Local Optimization by DAG Matching", in *Proc. 24th Design Automation conf.*, pp. 341-347, June 1987.

[5] E.Detjens, G.Gannot, R.L.Rudell, A.Sangiovanni-Vincentelli, and A.R.Wang,"Technology mapping in mis",in *International Conference on Computer-Aided Design*, pp.116–119, IEEE, Nov. 1987.

[6] R.Rudell,*Logic Synthesis for VLSI Design.* PhD thesis, U.C.Berkeley, Apr.1989. Memorandum UCB/ERL M89/49.

[7] H.J.Touati,"Performance-oriented technology mapping", U.C.Berkeley, Nov. 1990. Memorandum UCB/ERL M90/109.

[8] R.K. Brayton, R.L. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "MIS: A Multiple-Level Logic Optimization System", IEEE transactions of CAD, Vol. CAD-6, No. 6, pp.1062-1081, Nov. 1987.

[9] R.K. Brayton, C. McMullen, G.D. Hachtel, and A.L. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984

[10] A. Jain and R.E. Bryant, "Inverter Minimization in Multi-Level Logic Networks", in *International Conference on Computer-Aided Design*, pp.462–465, IEEE, Nov. 1993.

[11] R.E. Bryant, "Graph-based algorithms for boolean function manipulation", *IEEE Transactions on Computer*, C-35(12), 1986.