モンテカルロ木探索法を用いたテクノロジマッピングアルゴリズムについて

松永 裕介[1,a)]

概要：本稿ではモンテカルロ木探索を LUT 型 FPGA 向けテクノロジマッピングに応用したアルゴリズムについて述べる．テクノロジマッピング問題が複雑になる原因はファンアウト部分の取り扱いにある．そこで，予め回路のどの部分がファンアウト境界になるかを決めた上で既存のテクノロジマッピングのアルゴリズムである DAG Covering アルゴリズムを適用することで解空間を区切って探索する方法を考案した．このアルゴリズムとモンテカルロ木探索を組み合わせたテクノロジマッピングアルゴリズムの紹介を行う．

キーワード：論理合成，テクノロジマッピング，モンテカルロ木探索

On technology mapping alogirhtm using Monte-Carlo tree search

YUSUKE MATSUNAGA[1,a)]

Abstract: This paper presents a technology mapping algorithm for LUT-based FPGAs, which uses Monte-Calro Tree search. The main reason which makes the technology mapping problem complicated is the difficulty of handling nodes having multiple fanouts. So, a modified DAG covering algorithm which consider given fanout boundary conditions is introduced. Also, combination of the above algorithm with Monte-Calro tree search method is described.

Keywords: logic synthesis, technology mapping, Monte-Carlo tree search

1. はじめに

特定のテクノロジに依存しない論理回路をセルライブラリや FPGA デバイスに特化した論理回路に変換する処理をテクノロジマッピングと呼ぶ．テクノロジマッピングの手法としてはさまざまなものが提案されているが [3], [4], [5], [6], Tree Covering と呼ばれるアルゴリズムに基づいた手法が広く用いられる [5]．Tree Covering アルゴリズムはその名の通り，木構造の回路 (ファンアウト数が 1 の回路) に対して適用されるもので，回路規模に対して線形時間で面積最小の解を求めることができる．一方，一般の回路は DAG(directed acyclic graph) 構造を持つ (ファンアウト数が 2 以上のノードを持つ) ためそのままでは Tree Covering アルゴリズムを用いることができない．DAG Covering 問題は NP 困難であることが知られており，効率よく厳密解を得ることは難しいため Tree Covering をベースにしたヒューリスティックが用いられている．しかし，ヒューリスティックの多くはアドホックでありあまり根本的な手法に関して研究されているとは言い難い．

本稿ではモンテカルロ木探索という乱択アルゴリズム系のメタヒューリスティックをテクノロジマッピングに応用したアルゴリズムについて述べる．既存のヒューリスティックに対して制御点を追加するというアプローチで解空間を探索している．以下，2 節でテクノロジマッピング問題の定義および既存アルゴリズムの紹介を行う．つづいて 3 節でモンテカルロ木探索の簡単な説明を行い，4 で提案手法について述べる．最後に 5 節でベンチマーク回路を用いた実験結果を示す．

[1] 九州大学大学院システム情報科学研究院
〒 819–0395 福岡市西区元岡 744
[a)] matsunaga@ait.kyushu-u.ac.jp

2. テクノロジマッピング問題

テクノロジマッピングは一般の LSI を対象にしたものと FPGA を対象にしたもので分けることができる．前者は使用できる論理セルがセルライブラリとして与えられているものに限られているのに対して，後者は一定の入力数以下の論理関数なら 1 つの LUT(look-up table) を用いて実装できるという点で大きな違いがある．一方，それらをどう組み合わせて所望の性能をもった回路を合成するかというアルゴリズムに関しては両者の違いはほとんどないと言って良い [*1]　そこで，本稿では FPGA のテクノロジマッピングを例として取り上げることとする．以下の議論は容易にセルライブラリに対するテクノロジマッピングに対しても応用可能である．

FPGA にもいくつかのタイプがあるがここで LUT(look-up table) 型 FPGA を対象とする．k-LUT とは k 入力のルックアップテーブル (look-up table) のことで，k 入力以下の任意を論理関数を実現することができる．LUT 型 FPGA においては組み合わせ回路部分を構成する主要な基本論理素子となっている．

LUT 型 FPGA 向けのテクノロジマッピング問題とは，入力された論理回路と論理的に等価で k-LUT のみから構成される回路 (LUT ネットワーク) を合成する問題，と定義することができる．ただし，入力された論理回路と論理的に等価という条件はあまりにも緩すぎるので探索範囲が膨大になる．そこで実用的には入力された論理回路の構造と同様の構造を持った LUT ネットワークを合成する手法を用いることが多い．目的関数としては様々なものが考えられるがここでは LUT 数が最小となる LUT ネットワークを生成することを目的とする．それ以外の目的関数に対しても比較的容易に対応可能である．

まず，主要な概念および用語の定義を行う．

サブジェクトグラフ (sbject graph):　マッピング対象の回路を表す DAG(directed acyclic graph)．k-LUT にマッピングするためには各々のノードの入力数が k 以下である必要がある．この条件を k-フィージブル (k-feasible) と呼ぶ．もしも初期回路が k-フィージブルでない場合には入力数の多いノードを適当に分割する必要がある．

k-カット (k-cut)):　サブジェクトグラフの連結した部分グラフで以下の性質を持つもの．

- 根のノード v を持つ．
- 葉のノード集合 $U = \{u_1, u_2, \ldots, u_n\}$ を持つ．
- 各葉のノード u_i から根のノード v までの径路が存在する．u_i から v へ至るいかなる経路上にも他の葉のノード $u_j \neq u_i$ は含まれない．また，その経路上にカットの要素以外のノードも含まれない．

明らかに k-カットは 1 つの k-LUT で実現できることがわかる．つまり，サブジェクトグラフの全てのノードがいずれかのカットに含まれるようなカットの集合を求めることができれば，そのカットの集合が最終的な LUT ネットワークに対応する．ただし，結果の LUT ネットワークが回路として成り立っていなければならないので，あるカットが選ばれている場合にはその葉のノードを根とする別のカットも選ばれていなければならないという制約がある．紙面の都合上詳細は省略するが，このようなテクノロジマッピングの問題は Binate Covering 問題や 0-1 整数計画問題として定式化することができる [6]．これらはもちろん NP 困難問題であり，数千ゲート規模の回路に対するテクノロジマッピングに対する厳密解を現実的な時間で求めることは極めて難しい．そこで，いくつかのヒューリスティックが提案されている．Keutzer は木構造の回路に対するテクノロジマッピング問題を線形時間で解く Tree Covering アルゴリズムを提案している [5]．このアルゴリズムの概略をアルゴリズム 1 に示す．

```
input : サブジェクトグラフ 𝒢
input : LUT のサイズ k
output: LUT ネットワーク
𝒢 のノードを入力からのトポロジカル順に並べる．;
for v ∈ 𝒢 do
    m_v ← ∞;
    c_v ← φ;
    for c ∈ v を根とするカット集合 do
        n ← 1;
        for u ∈ c の葉のノード do
            n ← n + m_u;
        end
        if m_v > n then
            m_v ← n;
            c_v ← c;
        end
    end
end
v_o ← 𝒢 の出力のノード;
v_o の最良コストを与えているカットを選択する．;
以下同様に入力まで最良カットを選択する．;
選択されたカットを用いて結果の LUT ネットワークを構成する;
```

Algorithm 1: Tree Covering アルゴリズム

このアルゴリズムは動的計画法の考え方に基づいている．最終的に選択されるカットの根のノードかどうかにかかわらず，全てのノードに対して，そのノードを根とする部分木に対する最良のカットを記録しておく．ノード v に

[*1] 遅延モデルや消費電力モデルといった物理レベルの特性は異なっている．

対して v を根とするカット c を選択した時のコストはカット c 自身のコスト (この場合は LUT1 個なので 1) とカットの葉のノードに置ける最良コストを足し合わせたものとなる．回路全体が木構造の場合には出力における最良コストのカットを選択し，そのカットの葉になっているノードにおいても最良のコストのカットを選択する，という処理を繰り返せば回路全体に対する最良コストを与えるカット集合を選ぶことができる．

問題は回路の構造が木構造ではなく，より一般的な DAG(directed acyclic graph) 構造を持つ場合である．単純にカットの葉のノードにおけるコストを足し合わせた場合，2 つ以上のノードにファンアウトしているノードのコストが重複して足し合わされることとなり正確なコスト計算が行えない．また，ノード v を根とする部分回路の最良コストを求めてもそれが回路全体の最良コストとはならないため，動的計画法を用いることができないという問題点がある．しかし，前述のように多項式時間の厳密解法は存在しないと考えられているので，実用的なプログラムでは Tree Covering を元にしたヒューリスティックが用いられている．具体的にはアルゴリズム 1 において，葉のノード u のコストを足し合わせる処理を以下のように変更する．

$$n \leftarrow n + \frac{m_u}{|FO(u)|}$$

ここで $FO(u)$ はノード u のファンアウト先のノード集合を表す．つまり，足し合わせるコストをファンアウト数で割ることで重複して足し合わされた時のバランスを取っている．もちろん，これはただのヒューリスティックであり常に正しいコスト計算ができるわけではない．例えば図 1 の回路において，4-LUT を用いてマッピングすることを考える．回路全体を最も少ない LUT でマッピングするために

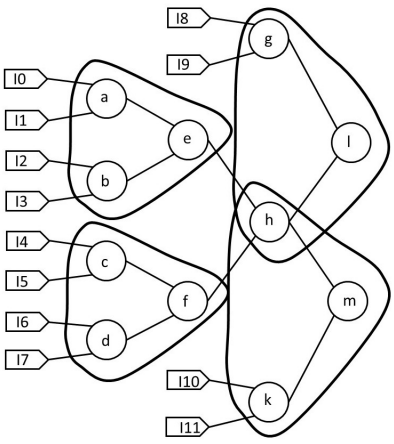

図 1 マッピングの例

はノード h を 2 つのカットで重複してカバーすることである．この場合，ノード e を根とするカット，ノード f を根とするカット，ノード l を根とするカット，ノード m を根とするカットの計 4 つのカットである．しかし，コスト計算の際にファンアウト数で割るというヒューリスティックではノード h を葉とするカットの計算の際にノード h のファンアウト数 2 が考慮されるだけで，ノード e やノード h を葉とするカットのコスト計算の際にそれらが 2 つのカットから用いられるということを考慮することができない．実際，ファンアウト数による補正を行った Tree Covering アルゴリズムではノード e およびノード f のコストは 1 であり，ノード l のコストは $1+0(I8)+0(I9)+1(e)+1(f)=3$ となり，ノード m のコストは $1+1(e)+1(f)+0(I10)+0(I11)=3$ となる．回路全体のコストは $3+3=6$ となるが実際の LUT 数は 4 である．一方，ノード h におけるコストは 3 であり，ノード l およびノード m のコスト計算においてノード h を葉とするカットを用いるとそれぞれ $\frac{3}{2}+1=2.5$ で 2 つ合わせると 5 となる．ノード h を重複してカバーする場合の見かけのコスト 6 よりも小さく見えてしまうためこちらが選択されるという問題がある．このように DAG に対するテクノロジマッピングは 2 つ以上のファンアウトを持つノードの取り扱いに問題がある．

3. モンテカルロ木探索

モンテカルロ木探索はモンテカルロ法と木の探索アルゴリズムを組み合わせたもので，大まかには，ランダムサンプリングに基づいて期待値の高いと思われる子供へ探索を行うものである．まず，最適化問題においては重要なのは今から探索しようと思っている子供の下に最適解があるかどうかであって，その子供からたどれる節点の評価値の期待値が高いかどうかと相関があるのか，という本質的な問題があるが，ここでは子供を選択するメタヒューリスティックとして期待値の高い子供を選択するものと考えておくものとする．

図 2 に一般的なモンテカルロ木探索アルゴリズムの擬似コードを示す [2].

input : 探索木の根のノード V_0
output: 最も評価値のよい探索木のノード
for 計算時間や回数制限の範囲内 do
  $V_l \leftarrow$ `TREE_POLICY`(V_0);
  $\Delta \leftarrow$ `DEFAULT_POLICY`(v_l);
  `BACKUP`(v_l, Δ);
end
return `BEST_CHILD` (V_0)

Algorithm 2: モンテカルロ木探索アルゴリズム

ここで，V_0 は探索の開始点，すなわち根のノードを表す．トップレベルのアルゴリズムはいたって単純である．与えられた制約時間もしくは制約回数に到達するまで同一の処理を繰り返す．ループの中では，TREE_POLICY() 関数で展開されている末端のノード V_l を求め，DEFAULT_POLICY()

関数でそのノードに対する評価値を計算し，BACKUP() 関数でその評価値を反映させる，という処理を行う．最後にBEST_CHILD() で最も評価値の高かったノードを返す．ただし，これはゲームの思考アルゴリズムにおいて次の手を返すためのものであるので結果として V_0 の子供のノードを返しているが，最適解を求めるためにはサンプリング中で最も評価値の高い解を記録しておきそれを返せば良い．

3.1 TREE_POLICY アルゴリズム

TREE_POLICY では木の探索および新しい子ノードの生成を行う．ここではUCT(UCB for Trees) と呼ばれるアルゴリズムについて説明する．UCT では UCB(upper confidencial bound)[1] と呼ばれる概念を用いてる．UCB は中心極限定理に基づいて現在の期待値とサンプル数から最善に見積もった期待値の上界を計算するものである．探索木を 1 段降りて行く度に以下の UCT で示される指標が最大となる子ノードを選択する [*2].

$$\mathrm{UCT} = \bar{x_j} + 2C_p\sqrt{\frac{2\ln n}{n_j}} \tag{1}$$

ここで，$\bar{x_j}$ は V_j の現在の期待値 (試行における評価値の総和を回数で割ったもの)，n は他の選択肢も含めた総試行数，n_j は選択肢 V_j における試行数である．つまり，試行回数が少ない場合にはサンプルから求めた期待値が真の期待値から離れている確率が高いと考えて試行回数に基づいた補正を行うというものである．2 項めの係数 $2C_p$ はもともとの UCB1 が期待値が $[0, 1]$ の範囲の場合を対象に考えられたものであるため，評価値がその範囲と異なる場合の補正のためのものである．

アルゴリズム 3 に TREE_POLICY アルゴリズムを示す．

```
Function TREE_POLICY(V)
    input : 探索木のノード V
    output: サンプリングを行うノード
    while V が終端設定でない限り do
        if V の子供のノードが全て展開済み then
            V ← BEST_CHILD(V);
        else
            return EXPAND (V);
        end
    end
    return V;
Function BEST_CHILD(V)
    return V の子供のうち UCT の指標が最も良いものを返す.;
Function EXPAND(V)
    V の子供のうち展開されていないノード V′ を一つ選ぶ.;
    return V′;
```

Algorithm 3: TREE_POLICY アルゴリズム

[*2] この UCT の式が UCB を用いたものである．

3.2 DEFAULT_POLICY アルゴリズム

DEFAULT_POLICY では何らかのヒューリスティックを用いて終端まで局面をすすめ，最終局面における評価値を計算する．個々の問題に対する知識やノウハウが最も必要とされる部分である．

4. 提案手法

ここでは LUT 型 FPGA 向けテクノロジマッピング問題を上記のモンテカルロ木探索を用いて解くアルゴリズムの提案を行う．ひとくちにモンテカルロ木探索を用いると言っても様々なやり方が考えられるので，次の項目について検討を行った．

- 探索の仕方: なにを探索における分岐とみなすのか．
- DEFAULT_POLICY の作り方: 未探索の部分をランダムにかつそれらしく探索するヒューリスティックを考える必要がある．
- 評価値の計算方法: 本来の問題は LUT 数を最小にすることが目的だが，UCT は式の性質上，期待値が最大のノードを選ぶ．また，問題ごとに最小 LUT 数が変化するのでそのダイナミックレンジの変化をうまく吸収できる評価値が望ましい．

以下に各項目に関して検討した内容を述べる．

4.1 探索の仕方

もっとも単純には，サブジェクトグラフの各ノードに対してどのカットを選ぶか (選ばないか)，という選択に基づく探索方法が考えられるが，これでは考慮すべき探索空間が膨大になり効率がわるい．そこで Tree Covering と DAG Covering の差異が生じる原因であるファンアウト数が 2 以上のノードに着目してみる．以降，ファンアウト数が 2 以上のノードを’**ファンアウトポイント**’と呼ぶことにする．もしも，入力となっているサブジェクトグラフ上のファンアウトポイント v がマッピング結果においてもファンアウトポイントであった場合，図 2 のようにもともとのサブジェクトグラフ G を v で分割した新たなグラフ G_v に対してテクノロジマッピングを行った結果と一致することになる．

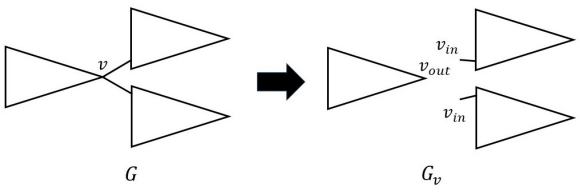

図 2 ファンアウトポイントによる分割

一方，ファンアウトポイント v がマッピング結果においてはファンアウトポイントでなかった場合，DAG Covering

のコスト計算においては v のコストはそのファンアウト数で割られているため過小評価となっていることになる．このように，サブジェクトグラフのファンアウトポイントがマッピング結果においてもファンアウトポイントになっているか否かは DAG Covering のヒューリスティックにおいては重要な役割を果たしている．以降，マッピング結果においてもファンアウトポイントになっているノードを'**境界ノード**'と呼ぶことにする．もちろん，事前にファンアウトポイントが境界ノードかどうかを判断することはできないが，ファンアウトポイント v を境界ノードとするかどうかを決めてマッピングを行うことはできる．明らかにそのどちらかに最適解が含まれることになるのでこの判断を探索における分岐と考えることができる．

次の問題はファンアウトポイント v を境界ノードとする場合としない場合のマッピングをどうするかであるが，境界ノードとする場合は前述のように非常に単純である．ただ単に v を擬似的な外部出力と外部入力とみなせば良い．つまり v へファンアウトしているノードにとって v は外部出力とみなし，v をファンインとするノードにとって v を外部入力とみなすことで図 2 の分割されたグラフに対するマッピングを考えることになる．v を境界ノードとしない場合は概念的には v をファンアウト数分複製することになる (図 1 の例を参照)．場合によっては v だけでなく v のファンイン側のノードまで複製されるかもしれないが，これは実際にマッピングを行わないとわからない．そこで実際には複製を行わずに v の各ファンアウト先のノードから v のコスト計算を行う際に v のコストをファンアウト数で割らずにそのまま使うことで擬似的に複製の効果を反映させるヒューリスティックを用いることとする．

より具体的にはアルゴリズム 1 の葉のノードのコストを足し合わせる部分を以下のように変更する．

$$n \leftarrow n + m_u \times w_u$$

ここで w_u はノード u における重み係数で u がファンアウトノードでない時には常に 1 となる．u がファンアウトノードの場合には境界ノードかどうかによって以下のように定義される．

$$w_u = \begin{cases} 1 & u \text{ が境界ノードでない時} \\ 0 & u \text{ が境界ノードの時} \\ \frac{1}{FO(u)} & u \text{ が未定の時} \end{cases}$$

この簡単な変更によって従来の DAG Covering ヒューリスティックにおいて境界ノードを考慮できるようになる．

モンテカルロ木探索全体としてはまず最初にファンアウトポイントの列挙を行い，適当な順序付けを行う [*3]．次に TREE_POLICY で新しい子ノードを作る際にファンアウトポイントを一つ取り出し，そのノードを境界ノードとする選択と境界ノードとしない選択を考えれば，全ての場合分けを考慮した探索が行えることとなる．

評価のたびにマッピングを行うので計算時間がかかると思われるかもしれないが，FPGA のテクノロジマッピングの場合，多くの時間をカット列挙に費やしており，提案手法でもカット列挙は最初に一回行うだけで良い．評価の際には各ノードにおけるコスト計算とカット選択だけを行えばよい．

4.2 DEFAULT_POLICTY の作り方

探索が境界ノードの判断に基づいて行われるので DEFAULT_POLICY においても決まっていないファンアウトポイントにおける境界ノードの判断をランダムに行えばよいように思われる．実際には，最適解において境界ノードになるかならないかはランダムに決まるものではなくサブジェクトグラフの構造から決まるものであり一様なランダムはよいヒューリスティックではない．もちろん，正解を求めるアルゴリズム存在しないので何らかのヒューリスティックを用いる必要があるが，結果にランダム性が含まれないヒューリスティックはモンテカルロ木探索では意味がない [*4]．なかなか難しい問題であり，決定版と呼べるヒューリスティックは見つかっていないが，今回は未決定のファンアウトポイントを低確率で選び境界ノードと決めたあとで (選ばれていないファンアウトポイントに関しては未決定のまま)，従来の DAG Covering アルゴリズムを適用する，という方法を用いている．

4.3 評価値の計算方法

上述のように LUT 数をそのまま評価値には用いることはできない．そこで以下のような式の値を評価値として用いている．

$$\text{Val} = \frac{U - x}{U - L}$$

ここで U は与えられたサブジェクトグラフのマッピング結果の上界，L は下界，x は実際のマッピング結果の LUT 数である．明らかにこの評価値の範囲は問題にかかわらず $[0, 1]$ となる．また，LUT 数の少ないほうがより 1 に近い値となっている．

次に上界と下界の計算方法について述べる．まず，マッピング結果の上界 U はサブジェクトグラフのノード数を用いる．つまり，もともとのノード一つに対して LUT を一つ対応させたものが上界 (上限) となる．自明な下界は知られていないので以下の 2 通りの手法で下界を計算し大きい値を用いる．

下界 1: カット c に対して内部に含まれているノード数をカットの価値 V_c と定義する．次に，ノード v に対し

[*3] 現在は入力からのトポロジカル順を用いている

[*4] 常に厳密解を求める決定的アルゴリズムなら意味がある

て v を含むカット c_i の価値 V_{c_i} の最大値 $max_i V_{c_i}$ の逆数をノード v の重み W_v とする．全てのノードに対して重みを足し合わせたもの $\sum_v W_v$ はもとのマッピング問題に対する下界となっている．
これは最適解におけるカット $c \in C$ に対して，c が含んでいるノード v の重み W_v が $\frac{1}{V_c}$ よりも等しいか小さいことから証明できる．

下界 2: 同一のカットによって同時に含まれることのない 2 つのノード v_1, v_2 を互いに独立と定義する．上記の独立関係に置ける極大独立集合 (maximal independent set) の要素数はもとのマッピング問題に対する下界となる．

どちらの下界の計算も各ノードを根とするカット列挙が行われていれば簡単に行うことができる．

5. 実験結果

提案手法の評価のためにベンチマーク回路を用いた実験を行った．ベンチマーク回路は ITC99 を用いている [*5]．使用計算機は Intel Core i7-2600 (3.40GHz)(メモリ 16GB), OS は FreeBSD 10.3-RELEASE-p7, 使用コンパイラは clang++-3.4 となっている．

比較対象として 2 節で述べた既存の DAG Covering の結果を用いている．モンテカルロ木探索はすべての例に対して 10,000 回の試行を行った．表 1 に実験結果を示す．表中の N はサブジェクトグラフのノード数, F はファンアウトポイント数を表す．従来手法，提案手法とも結果の LUT 数を示している．

ある程度提案手法によって結果が改善していることがわかる．計算時間に関しては大まかには通常のテクノロジマッピングを 10,000 回繰り返しているので従来手法よりは大幅に遅い．しかし，実際の時間の比は大規模回路では数十倍に収まっており単純に試行回数の数だけ増えているわけではない．これは前述のようにテクノロジマッピングにおいてはカットの列挙が計算時間の大部分を占めており，提案手法では各試行ごとにカットを列挙を行うのではなく，最初に一回だけカットの列挙を行っていることに起因すると考えられる．とはいえ，多大な時間をかけた割にはあまり大きな改善が得られていないので効率化およびより効果的なヒューリスティックの開発が必要と思われる．

6. おわりに

本稿ではファンアウト部分を境界ノードにするかどうかを指定して DAG Covering を行うことにより解空間を分割する手法を提案し，そのアルゴリズムをモンテカルロ木探索と組み合わせたテクノロジマッピングを開発した．ベンチマーク回路を用いた実験結果より提案手法の有効性を確認している．

表 1 実験結果

回路名	N	F	従来手法 LUT 数	従来手法 CPU(s)	提案手法 LUT 数	提案手法 CPU(s)
b14	9150	1472	2215	1.39	2192	291.03
b14_opt	6448	1450	2236	0.57	2211	172.52
b14_1	6317	1042	1673	0.58	1663	200.91
b14_1_opt	4934	1055	1666	0.29	1650	131.71
b15	8877	1479	2969	0.80	2888	240.86
b15_opt	8764	1567	3144	0.61	3122	187.06
b15_1	11825	1642	3197	1.18	3187	614.39
b15_1_opt	9456	1343	3027	0.96	2994	475.61
b17	31008	4511	9721	12.18	9639	1116.66
b17_opt	28768	4848	10049	9.55	10027	920.04
b17_1	36118	5021	9805	16.01	9795	1890.18
b17_1_opt	29254	4108	9374	11.15	9332	1469.08
b18	104580	17742	28870	266.03	28694	5208.32
b18_opt	85419	17333	28193	198.23	28167	4015.80
b18_1	98801	16952	27643	239.81	27441	5277.14
b18_1_opt	83442	17158	27845	181.84	27826	4024.06
b20	18531	2762	4419	6.56	4384	644.10
b20_opt	14500	3279	5136	2.91	5098	371.51
b20_1	13175	2181	3249	2.99	3226	464.98
b20_1_opt	12269	2885	4417	1.90	4387	310.42
b21	18966	2793	4490	6.17	4468	627.06
b21_opt	14530	3279	5058	2.65	5021	363.48
b21_1	13266	2055	3323	2.84	3299	441.77
b21_1_opt	11562	2584	4097	1.62	4069	278.38
b22	27599	4086	6566	19.53	6522	927.79
b22_opt	20590	4714	7225	5.66	7185	527.02
b22_1	19899	3248	4913	6.87	4888	689.31
b22_1_opt	17685	4126	6136	4.17	6110	467.01

参考文献

[1] Auer, P., Cesa-Bianchi, N. and Fischer, P.: Finite-time analysis of the multiarmed bandit problem, *Mach. Learn.*, Vol. 47, No. 2, pp. 235–256 (2002).

[2] Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S.: A Survey of Monte Carlo Tree Search Methods, *Computational Intelligence and AI in Games, IEEE Transactions on*, Vol. 4, No. 1, pp. 1–43 (online), DOI: 10.1109/TCIAIG.2012.2186810 (2012).

[3] Cong, J. and Ding, Y.: FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs, *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, Vol. 13, No. 1, pp. 1–12 (online), DOI: 10.1109/43.273754 (1994).

[4] Darringer, J., Brand, D., Gerbi, D., Joyner, W. and Trevillyan, L.: LSS: A System for Production Logic Synthesis, *IBM Journal of Ressearch and Development*, Vol. vol. 28, No. no. 5, pp. 537–545 (1984).

[5] Keutzer, K.: DAGON: Technology Binding and Local Optimization by DAG Matching, *Proceedings of 24th Design Automation Conference*, pp. 341–347 (1987).

[6] Rudell, R.: Logic Synthesis for VLSI Design, PhD Thesis, U. C. Berkeley (1989).

[*5] 結果の LUT 数が 1000 以上の回路のみを載せている．