

モンテカルロ木探索を用いた 並列プレフィックス加算器合成手法

松永 多苗子^{1,a)} 松永 裕介²

概要: モンテカルロ木探索アルゴリズムは、囲碁等のゲームアルゴリズム分野で脚光を浴び、様々な分野への応用が期待されている。本稿では LSI CAD 問題の一つである並列プレフィックス加算器合成問題にモンテカルロ木探索を適用するための一手法を述べ、課題について議論する。

キーワード: モンテカルロ木探索, 組み合わせ最適化, 並列プレフィックス加算器

On Automatic Generation of Parallel Prefix Adders based on Monte-Carlo Tree Search Algorithm

MATSUNAGA TAEKO^{1,a)} MATSUNAGA YUSUKE²

Abstract: Monte-Carlo tree search algorithm succeeds especially in the field of game algorithms, and is expected to apply to many other fields. This paper proposes its application to automatic generation of parallel prefix adders in the field of CAD algorithms for LSI design, and discusses the issues on efficiency.

1. はじめに

算術演算回路は様々なアプリケーションにおいて回路全体の品質に影響を与える重要な要素である。特に 2 入力加算器は最も基本的で使用頻度の高い演算器であるため、従来より数多くの加算器アーキテクチャが提案されている [1]。中でも並列プレフィックス加算器は、先見桁上げの概念を一般化した手法により桁上げ伝搬を高速化するもので、様々な特徴をもった規則的な構造が提案されるとともに [2]、構造をあらかじめ固定するのではなく制約に応じて自動生成するアルゴリズムが提案されている [3], [4], [5], [6]。

遅延制約下の面積最小化の場合、これまでに比較的よいヒューリスティックが提案されており、特に遅延制約が比較的緩い場合には最小解が得られることも多い。しかし、

遅延制約が厳しい場合の解の品質については議論の余地が残されている。また、消費電力など、論理レベルでは扱いにくい指標や、テクノロジーを考慮に入れた遅延・面積など、より現実的な指標については、まだ十分に考慮されていない。

本稿は、この並列プレフィックス加算器合成問題に対する新しい試みとして、モンテカルロ木探索 [7] の適用を考えるものである。モンテカルロ木探索はもともとコンピュータ囲碁の思考ルーチンとして提案されたものであるが、探索木に基づいた順序的な決定を行う問題に対して有効であることが示されており、LSI の CAD 問題への適用可能性が指摘されている [8]。本稿では、より現実的な指標を用いたプレフィックス加算器合成環境の構築を念頭に、その第一段階としてモンテカルロ木探索の適用可能性についての考察を行うものである。

以下、2 節においてモンテカルロ木探索の概要を述べ、3 節において並列プレフィックス加算器問題について説明を行う。それらを踏まえて、4 節においてモンテカルロ木探索のプレフィックス加算器問題への適用について説明

¹ 日本文理大学工学部情報メディア学科
〒 870-0397 大分市一木 1727

² 九州大学大学院システム情報科学研究院

^{a)} matsunagatk@nbu.ac.jp

し、課題についての考察を行う。

2. モンテカルロ木探索

モンテカルロ木探索とは、ランダムサンプリングを用いて期待値等を計算するモンテカルロ法と、木の探索アルゴリズムを組み合わせたものである。アルゴリズム 1 にモンテカルロ木探索アルゴリズムの擬似コードを示し、図 1 に探索における 1 サイクルの動作を示す。

Algorithm 1 モンテカルロ木探索アルゴリズム

```

function MCT_SEARCH( $V_0$ )
  while (計算時間や回数制限の範囲内) do
     $V_i \leftarrow$  TREE_POLICY( $V_0$ )
     $\Delta \leftarrow$  DEFAULT_POLICY( $V_i$ )
    BACKUP( $V_i, \Delta$ )
  return BEST_CHILD( $V_0$ )

function TREE_POLICY( $V$ )
  while ( $V$  が終端接点でない) do
    if  $V$  の子ノードがすべて展開済み then
       $V \leftarrow$  BEST_CHILD( $V$ )
    else
      return EXPAND( $V$ )
  return  $V$ 

function BEST_CHILD( $V$ )
   $V$  の子ノードのうち UCT の指標が最も良いノードを返す。

function EXPAND( $V$ )
   $V$  の子ノードのうち展開されていないノード  $V'$  を一つ選ぶ。
  return  $V'$ 

function DEFAULT_POLICY( $V$ )
  while ( $V$  が終端接点でない) do
     $V$  の子供  $V'$  をランダムに選ぶ。
     $V \leftarrow V'$ 
  return ( $V$  の評価値)
  
```

図 1 において各節点は部分解に相当し、矩形の節点の一つの解に相当する。アルゴリズム記述中の V_0 は探索の開始点（探索木の根のノード）を示す。

一回の処理は、展開されている末端ノードを求め (TREE_POLICY)、そのノードに関する評価値を計算し (DEFAULT_POLICY)、その評価値を反映させる (BACKUP) 処理から構成される。この処理を与えられた計算時間や繰り返し回数制限の範囲内で繰り返し、最後に最も評価値の高かったノード返す。

展開する子ノード V_j の選択には、以下の UCT (Upper Confidence Bound fo Trees) で示される指標を用いる。

$$UCT = \bar{x}_j + 2C_p \sqrt{\frac{2 \ln n}{n_j}} \quad (1)$$

ここで、 n は他の選択枝も含めた総試行数、 n_j は V_j における試行数であり、 \bar{x}_j はその時点での V_j の期待値（試行

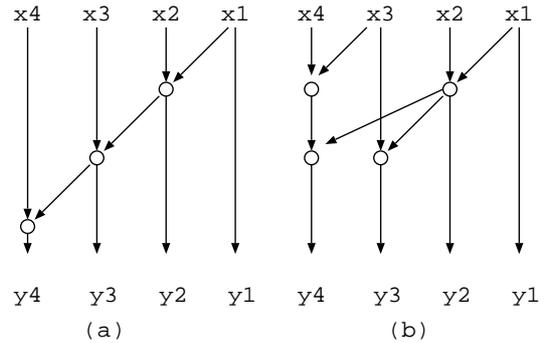


図 2 幅 4 のプレフィックスグラフの例

における評価値の平均) である。2 項目の $2C_p$ は、期待値が $[0,1]$ の範囲と異なる場合の補正のための係数である。

3. 並列プレフィックス加算器

本節では並列プレフィックス加算器の構造を考える上での基盤となるプレフィックスグラフについて説明し、それを用いて加算器合成問題を定義する。

3.1 プレフィックス計算とプレフィックスグラフ

N 個の入力 x_N, x_{N-1}, \dots, x_1 と結合則を満たす任意の演算 \circ が与えられたとき、 N 個の出力 y_i を $y_i = x_i \circ x_{i-1} \circ \dots \circ x_1$ によって計算するものをプレフィックス計算と呼ぶ。これは、 i 番目の出力 y_i は $j \leq i$ となる入力 x_j のみに依存することを意味する。

N 入力プレフィックスグラフは、 N 個の入力に対するプレフィックス計算における各演算 \circ をノードとした非巡回有向グラフ (directed acyclic graph: DAG) であり、各出力の値を計算するための演算の実行順序を表現したものである。ノードの 2 つの入力は 2 項演算 \circ の 2 つのオペランドに対応し、ノード v_1 が v_2 のオペランドであるとき、 v_1 から v_2 へのエッジが存在する。 v_1 を v_2 のファンインノード、 v_2 を v_1 のファンアウトノードと呼ぶ。

図 2 に幅 4 のプレフィックスグラフの例を示す。(a) の y_4 は、 $y_4 = x_4 \circ (x_3 \circ (x_2 \circ x_1))$ という演算順で計算され、(b) では $y_4 = (x_4 \circ x_3) \circ (x_2 \circ x_1)$ で計算されていることが分かる。プレフィックス計算は結合則が成り立つため、どちらの順序で計算しても結果は変わらない。

図 3 は、図 2(b) の各ノードを入力範囲の幅で整列したものである。 i 行 j 列のノード $v_{i,j}$ ($i \leq j$) は、 $i+1$ 個の入力に対するプレフィックス計算の中間結果を表わしている。演算の 2 つのオペランドは、連続した入力範囲に対する中間結果となっている。

例えば図 3 の $v_{3,3}$ は 2 つのファンイン $v_{1,3}, v_{1,1}$ を持ち、 $v_{3,3} = v_{1,3} \circ v_{1,1}$ という演算結果を表している。

$v_{1,3}$ は入力 2 から 3 に対する演算結果、 $v_{1,1}$ は入力 0 から 1 に対する演算結果であり、それらに演算を施した結果 $v_{3,4}$ は入力 0 から 3 に対する演算結果を表わしている。

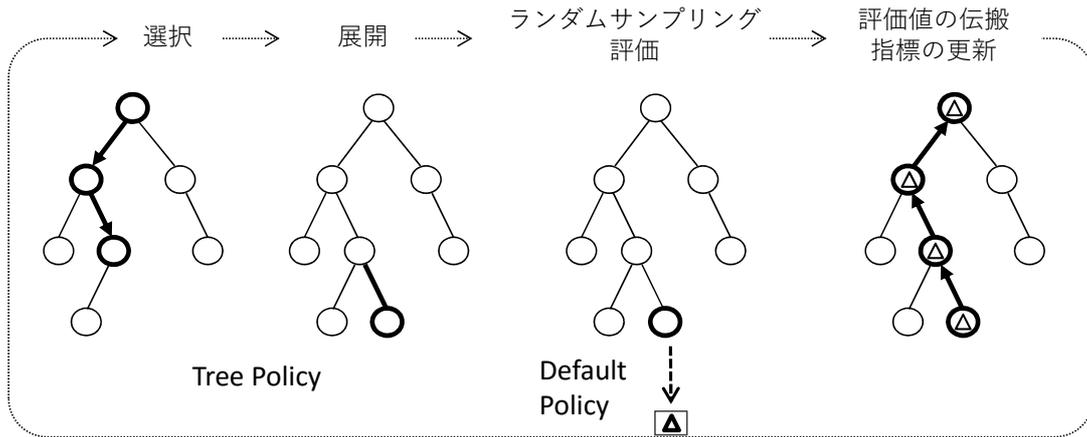


図 1 モンテカルロ木探索における処理の流れ

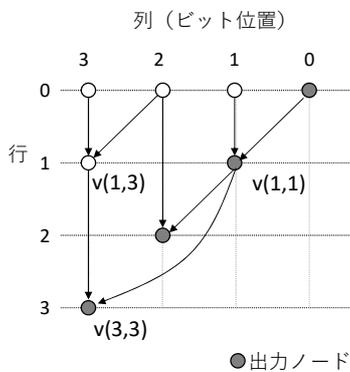


図 3 ノードを入力範囲の幅で整列したプレフィックスグラフ

3.2 プレフィックス計算としての2進加算

n ビットの2進加算の入力を $A = a_n, \dots, a_1, B = b_n, \dots, b_1$, 出力を和 $S = s_n, \dots, s_1$, 及び, キャリー出力 c_n とすると, 各ビットの和 s_i とキャリー出力 c_i は, $s_i = a_i \oplus b_i \oplus c_{i-1}$ と $c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$ で計算される. この n ビット加算は, 1 ビット信号に対する桁上げ生成/伝搬関数 (g, p) と, それを連続する複数ビットのグループに対する演算に拡張した桁上げ生成/伝搬関数 (G, P) を用いると以下のように計算できる.

- g, p の生成: $g_i = a_i \cdot b_i, p_i = a_i \oplus b_i$
- プレフィックス処理: G, P を用いて $c_i = G_{[i:1]}$ を計算

$$G_{[i:j]} = \begin{cases} g_i & \text{if } i = j \\ G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]} & \text{otherwise} \end{cases}$$

$$P_{[i:j]} = \begin{cases} p_i & \text{if } i = j \\ P_{[i:k]} \cdot P_{[k-1:j]} & \text{otherwise} \end{cases}$$

- s_i の生成: $c_i = G_{[i:1]}, s_i = p_i \oplus c_{i-1}$

並列プレフィックス加算器は上記の3つの処理を行う要素から構成される加算器である. このうちプレフィックス

処理の部分は, 演算 \circ を以下のように定義することによってプレフィックス計算とみなすことができ, その演算順序はプレフィックスグラフで表現することができる.

$$\begin{aligned} (G, P)_{[i:j]} &= (G, P)_{[i:k]} \circ (G, P)_{[k-1:j]} \\ &= (G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}, \\ &\quad P_{[i:k]} \cdot P_{[k-1:j]}) \end{aligned}$$

並列プレフィックス加算器の面積や遅延時間等の特性は, プレフィックス処理部の構成に依存するため, 加算器構成の大まかな特性はプレフィックスグラフにより表されていると言える.

並列プレフィックス加算器は多くの場合, プレフィックスグラフのノード数, 入力から出力への段数, 1つのノードから出力されるエッジの本数(ファンアウト数)を用いて特徴づけられる. 既存の自動合成アルゴリズムにおいても, これらの要素を指標として合成処理が行われるものがほとんどである. 本稿においても, モンテカルロ木探索の並列プレフィックス加算器問題への適用の第一段階としてこれらの指標を用いるものとして, 以下の問題を考えることとする.

定義 1 (並列プレフィックス加算器問題) 加算器の入力ビット幅と最大段数制約が与えられた下で, ノード数最小のプレフィックスグラフを生成する.

4. モンテカルロ木探索を用いた並列プレフィックス加算器合成

4.1 方針

まず, 解(部分解を含む)を表すプレフィックスグラフとして, 図3に示した整列されたグラフを用いるものとする. グラフにおけるノードは, 一連のビット範囲に対する G, P 演算を実現する. r 行 c 列のノードの場合, 幅が $r+1$ で最上位ビットが c である範囲の G, P を計算してい

る。最終的にすべての出力 (i 行 i 列のノード) に対して 0 から i の範囲の計算を実行している状態が一つの解となる。

モンテカルロ木探索を用いるにあたって、以下の方針を取ることにする。

- 初期状態：レベル 0 の n 個のノード ($v(0, i), 0 \leq i \leq n - 1$) からなるプレフィックスグラフ。
- 毎回の処理：プレフィックスグラフ中の 2 つのノードに対して演算を行い 1 つのノードを生成。
- 終了判定：すべての出力 ($v(i, i), 0 \leq i \leq n - 1$) がグラフに含まれること。

初期状態に含まれるノードは、各ビットに対して g_i, p_i の計算のみ行う状態を示している。探索における毎回の処理で 2 つのノードを組み合わせるということは、 $[lsb1 : msb1]$ の範囲の演算と $[lsb2 : msb2]$ ($lsb1 \leq lsb2, msb1 \leq msb2$ とする) の範囲の演算を結合して $[lsb1 : msb2]$ の範囲の演算結果を求めることに相当する。演算は連続した範囲でないといけないので、 $lsb2 = msb1 + 1^{*1}$ である必要がある。

最終状態においては、すべてのビット i において、0 から i までの連続した範囲の演算結果が生成されている (つまりプレフィックスグラフの出力ノードが生成されている)。

4.2 TREE POLICY

TREE POLICY においては、子ノードが展開済みかどうかの判断、展開済の場合はより深く探索するノードの選択、展開済み出ない場合は次に展開するノードの選択が必要となる。

- 子ノード
前述の通り、グラフ中のノード対から新たなプレフィックスノードを生成できるのは、それらがカバーする範囲が少なくとも隣り合っていることが条件となる。
- より有望なノードの選択

式 1 を用いる。最初の項は総ノード数の期待値である。本問題はノード数最小化問題であるため、第 2 項の符号はマイナスにしている。係数 C_p は評価値の取りうる範囲によって調節する必要がある。現時点では、プレフィックスグラフのノード数は最大でも $n(n - 1)/2$ (整列されたプレフィックスグラフ上の r 行 i 列の個数) であることを用いて、評価値を $[0:1]$ の範囲に補正している。

なお、この式そのものには段数の要素が入っていない。段数制約を考慮しないなら、探索するまでもなくシミュレーション実行 (リップルキャリーに相当) が最小解になってしまう。これに対して、現時点では評価値を $[0:1]$ の範囲に補正した状態で、段数制約を満たさない場合評価値を 1 にすることにより、最終的に選ばれにくくする方法をとっている。

*1 本来は 2 つの範囲に重なりがあっても構わないので、 $lsb2 \leq msb1 + 1$ であるが、簡単のため本稿では重なりを考えない。

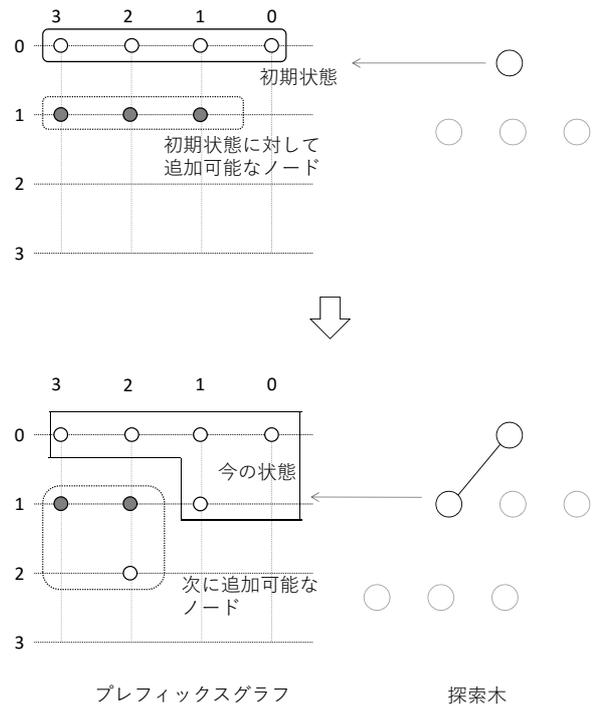


図 4 プレフィックスグラフに対するモンテカルロ木探索の例

- 未展開ノードの選択については、ランダムに行う。

4.3 DEFAULT POLICY

上記と同様に、現在プレフィックスグラフ中に存在するノード対の中で、組み合わせることのできるものをランダムに選んでグラフに追加する、ということを繰り返して、すべての出力が生成されたら終了し、その時にノード総数を返す。ただし、出力を生成する候補は優先的に選択する。

図 4 にビット数 4 の場合の探索開始時の様子を示す。初期状態は 1 ビットの演算ノード (第 0 行のノード) のみが含まれるグラフで、これが根の状態に相当する。ここに含まれる状態から生成可能なノードは、 $v_{0,0}$ と $v_{0,1}$ から生成される $v_{1,1}$, $v_{0,1}$ と $v_{0,2}$ から生成される $v_{1,2}$, および、 $v_{0,2}$ と $v_{0,3}$ から生成される $v_{1,3}$ の 3 つのノードであるため、根ノードは 3 つの子ノードをもつ。このうち例えば $v_{1,1}$ を状態に加えたとしても、左端の子ノードに対応するノード集合は $\{v_{0,0}, v_{0,1}, v_{0,2}, v_{0,3}, v_{1,1}\}$ であり、まだ含まれていない出力ノードが存在するため終端ノードではない。そこで、DEFAULT POLICY により、次に加えることができる候補ノード集合 $v_{1,2}, v_{1,3}, v_{1,1}$ からランダムにノードを選んで加える、という作業をすべての出力が含まれるまで繰り返し、その場合のノード数を返す。

5. 考察

上記の方針に基づいて予備実験を行ったところ、い

くつかの検討事項が挙がってきている。まず、DEFAULT_POLICY において単純なランダムサンプリングを行うのは、かなり効率が悪いということである。今回の定式化においては、1ビットの演算から始めて徐々に範囲を広げていく、というボトムアップのやり方をとっている。しかし、このやり方では、出力ノードがすべて生成された時点で、出力には影響を及ぼさないノードが生成されている可能性がある。評価の際に除外することは可能であるが、より効率の良い方法を検討する必要がある。

次に、[8]でも指摘されているように、探索が実際には木にならず DAG になる場合には、UCT の指標が不適切になってしまうという問題がある。プレフィックスグラフの場合にもこのような状況は存在していると考えられるが、現状では DAG 構造を無視している。効率への影響の調査や計算式の工夫など、検討課題として残っている。

さらに、現在は非常にナイーブな実装であるため小規模な実験しか行っていないが、既存の手法を見ても、ビット幅が大きくなるほど、あるいは、制約が厳しいほどアルゴリズムの差が出てくるように思われる。今後実装上の改良を行った上で、制約が厳しい状態での本手法の優位性について評価していく必要がある。

6. おわりに

本稿では、モンテカルロ木探索アルゴリズムを、並列プレフィックス加算器合成問題へ適用するための一手法を提示し、予備実験をもとに考察を行った。モンテカルロ木探索のアルゴリズムの構造はシンプルで、様々な分野への応用が期待されている。しかし、実際に具体的な問題に適用するにあたっては、効率や指標の有効性などの課題が挙がってきており、今後更に評価・検討が必要である。

並列プレフィックス加算器合成において残されている問題として、評価指標の精密化がある。プレフィックスグラフのノード数と面積とはある程度相関が高いと言える。しかし、遅延時間や消費電力に関しては、段数やファンアウト数で間接的に考慮することはできるが、テクノロジー非依存のレベルでは正確に行うことは困難である。既存の手法では対応することが難しいと思われるが、モンテカルロ木探索では解を生成して評価するため、より複雑な評価関数にも対応できる可能性があり、この点についても検討を行う予定である。

参考文献

- [1] I. Koren, *Computer Arithmetic Algorithms*. A K Peters, Ltd., 2002.
- [2] N. H. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*, chapter 10. Datapath Subsystems, pages 637–711. Addison Wesley, 2004.
- [3] R. Zimmermann, “Non-heuristic optimization and synthesis of parallel-prefix adders”, In *International Work-*

- shop on Logic and Architecture Synthesis*, pages 123–132, December 1996.
- [4] J. Liu, S. Zhou, H. Zhu, and C.-K. Cheng, “An algorithmic approach for generic parallel adders”, In *ICCAD*, pages 734–730, November 2003.
- [5] Taeko Matsunaga and Yusuke Matsunaga, “Timing-Constrained Area Minimization Algorithm for Parallel Prefix Adders”, in *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol.E90-A, no.12, pp.2770-2777, December 2007.
- [6] S. Roy, M. Choudhury, R.Puri and David Z. Pan, “Polynomial Time Algorithm for Area and Power Efficient Adder Synthesis in High-Performance Designs”, in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.35, num5, pp.820-831, May 2016.
- [7] C. Browne, E. Powley, D. Whitehouse, S. Lucas, “P.I.Cowling, P.Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis and S. Colton”, “A Survey of Monte Carlo Tree Search Methods”, in *IEEE Transactions on Computational Intelligence and AI in Games*, vol.4, no.1, pp.1-49, March 2012.
- [8] 松永裕介, “モンテカルロ木探索の CAD 問題への応用について”, 信学技法 VLD2015-46, pp.51-55, 2015.