

3 アドレス QDD マシン用コードの最適アルゴリズムについて

福山 泰介[†] 笹尾 勤[†] 松浦 宗寛[†]

[†]九州工業大学大学院 情報工学研究院 〒 820-8502 福岡県飯塚市大字川津 680-4

あらまし 論理シミュレーションの高速化を目的として、4 値の決定グラフ QDD を用いる。本論文では、QDD をデータ構造とする 3 アドレス方式の QDD マシン用コードを、 $O(N^2)$ (N は QDD の節点数) の計算時間で最適化するアルゴリズムを提案する。種々の多出力ベンチマーク関数に対し、アルゴリズム適用時の処理時間を測定した。

キーワード QDD, ブランチング・プログラム・マシン, QDD マシン

An Optimal Algorithm for 3-address QDD Machine Code

Taisuke FUKUYAMA[†], Tsutomu SASAO[†], and Munehiro MATSUURA[†]

[†] Department of Computer Science and Electronics, Kyushu Institute of Technology
Kawazu 680-4, Iizuka, Fukuoka, 820-8502 Japan

Abstract To speed up the evaluation of logic functions, we use quaternary decision diagrams (QDDs). A 3-address QDD machine is a branching program machine which has 3-address QDD branch instructions. In this paper, we show a method to minimize a 3-address QDD machine code in time complexity of $O(N^2)$, where N is the number of QDD nodes. We minimized codes for 3-address QDD machines representing various benchmark functions. Experimental results are shown.

Key words QDD, Branching Program Machine, QDD Machine

1. はじめに

近年、デジタルシステムの大規模化に伴い、論理シミュレーションに費やす時間が増加している。論理シミュレーションを実行する従来の方法として、汎用プロセッサとソフトウェアを用いる方法、FPGA (Field Programmable Gate Array) 上に実装する方法がある。前者は実現は容易だが低速であり、後者は設計に手間がかかるという問題がある。そこで、論理シミュレーションを高速にするため、QDD(Quaternary Decision Diagram) 用 ブランチング・プログラム・マシン (QDD マシン) を用いる。QDD とは、BDD(Binary Decision Diagram) の各変数を 2 個ずつまとめて平均ステップ数を削減した決定グラフである。本論文では、3 アドレス方式の QDD マシンについて、そのコードを最適化する手法を提案する。

第 2 章では、多出力論理関数の表現法の 1 つである MTBDD と、4 値の決定グラフである QDD について述べる。第 3 章では、QDD マシンとコード最適化について述べる。第 4 章では、3 アドレス QDD マシン用コードを最適化するアルゴリズムについて述べる。第 5 章では、種々のベンチマーク関数に対して、アルゴリズムを適用した結果を示し、考察を述べる。第 6 章では、まとめと今後の課題について述べる。

2. 多出力論理関数の表現

[定義 2.1] MTBDD

任意の n 入力 m 出力論理関数は MTBDD (Multi-Terminal Binary Decision Diagram) で表現できる。MTBDD は終端節点に m ビットの出力ベクトルを持つ BDD であり、高々 n 回のメモリアクセスで多出力を同時に評価できる。以降、本論文で取り扱う決定グラフにおいては、MT を省略する。図 1 に、MTBDD の例を示す。

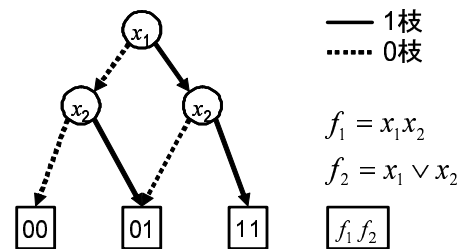


図 1 MTBDD の例

[定義 2.2] 分割と超変数

論理変数を $X = (x_1, x_2, \dots, x_n)$ とする。 X の変数の集合を $\{X\}$ で表す。 $\{X\} = \{X_1\} \cup \{X_2\} \cup \dots \cup \{X_u\}$ かつ

$\{X_i\} \cap \{X_j\} = \phi$ ($i \neq j$) のとき, (X_1, X_2, \dots, X_u) を X の分割, X_i を超変数という. $|X_i| = k_i$ ($i = 1, 2, \dots, u$) とすると, $k_1 + k_2 + \dots + k_u = n$ である.

BDD の評価時間削減のため, MDD (Multi-valued Decision Diagram) を用いる.

[定義 2.3] MDD(k)

MDD は, BDD の入力変数 $X = (x_1, x_2, \dots, x_n)$ を (X_1, X_2, \dots, X_u) と分割し, BDD の各変数を 2^{k_i} 値超変数でグループ化して構成した決定グラフである. $k_1 = k_2 = \dots = k_u = k$ の場合を特にホモニアス MDD(k) と呼ぶ. 本論文では以降, ホモニアス MDD(k) を扱い, これを単に MDD(k) と表記する.

論理関数を MDD(k) で表現すると, BDD の場合に比べメモリアクセス回数をしばしば $\frac{1}{k}$ まで削減できるが, 1 つの節点を表現するために必要なメモリ量は $O(2^k)$ で増加する. 図 2 に, BDD から MDD(2) への変換例を示す. 図 2 における MDD(2) では, 元の BDD の入力変数 $X = (x_1, x_2, \dots, x_8)$ を, (X_1, X_2, X_3, X_4) と分割し, 4 つの超変数 $X_1 = (x_1, x_2)$, $X_2 = (x_3, x_4)$, $X_3 = (x_5, x_6)$, $X_4 = (x_7, x_8)$ を構成している.

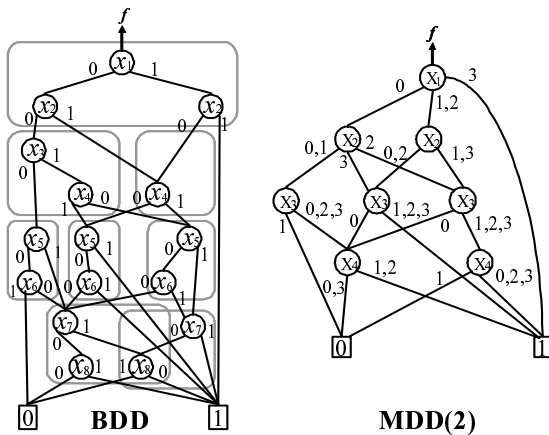


図 2 BDD から MDD(2) への変換例

[定義 2.4] QDD

MDD(2) は 1 つの非終端節点が 4 つ分岐を持つため, QDD (Quaternary Decision Diagram) と表現する.

多くのベンチマーク関数において, $k = 2$ の時 MDD(k) の総メモリ量が最小となる [3]. よって, 論理シミュレーションには, BDD よりも QDD の方が適している.

3. QDD マシンとコード最適化

3.1 ブランチング・プログラム・マシン

分岐命令と出力命令を持ち, BDD などの決定グラフをシミュレートするプロセッサをブランチング・プログラム・マシン (BPM) という [1]. BPM は, 汎用プロセッサよりアーキテクチャが単純である. また, 条件分岐を専用命令で実行するので, 回路シミュレーションやパターンマッチングなどの特定の用途では, 汎用プロセッサよりも高速である. BPM における条件分

岐命令と出力命令は, 決定グラフにおいては, それぞれ非終端節点, 終端節点に対応する.

3.2 QDD マシン

QDD 用の BPM を QDD マシンと呼ぶ. QDD マシンには, 3 アドレス QDD マシンと 4 アドレス QDD マシンがあり, 両者の主な違いは, 条件分岐命令の構成である. 図 3 に 4 アドレス QDD マシンの条件分岐命令を, 図 4 に 3 アドレス QDD マシンの条件分岐命令を示す. 図 3, 図 4 において, Branch は条件分岐命令であることを示すオPCODEであり, INDEX は超変数 X_i のインデックス i を指定する. ADDR には, 次の分岐先アドレスを格納する. また, 図 4 のプログラムコードにおける +1 は, 後述する 3 アドレス QDD マシンのアーキテクチャのプログラムカウンタ (PC) を 1 つ増加することを意味している. 4 アドレス QDD マシンでは, オペランドに 4 つの分岐先アドレスを全て格納するため, 命令長が長くなるという欠点がある. 本研究では, 4 アドレス QDD マシンよりも命令長が短い 3 アドレス QDD マシンを扱う.

Q_Branch (ADDR0, ADDR1, ADDR2, ADDR3), INDEX					
Branch	INDEX	ADDR0	ADDR1	ADDR2	ADDR3

図 3 4 アドレス QDD マシンの条件分岐命令

Q_Branch (ADDR0, ADDR1, ADDR2, +1), INDEX				
Branch	INDEX	ADDR0	ADDR1	ADDR2

図 4 3 アドレス QDD マシンの条件分岐命令

3.3 3 アドレス QDD マシン

3.3.1 アーキテクチャ

図 5 に, 3 アドレス QDD マシンのアーキテクチャを示す. INDEX で指定される超変数 X_i のインデックス i の値に応じて, 入力セレクタで, 連続する 2 つの 2 値変数を選択する. 選択された値は, マルチプレクサの制御入力信号となる. 制御入力信号 (超変数) が, $X_i = (0, 0)$ の場合は ADDR0, $X_i = (0, 1)$ の場合は ADDR1, $X_i = (1, 0)$ の場合は ADDR2 が選択される. $X_i = (1, 1)$ の場合は PC の値を 1 つ増加し, 現在の命令の次のアドレスにアクセスする.

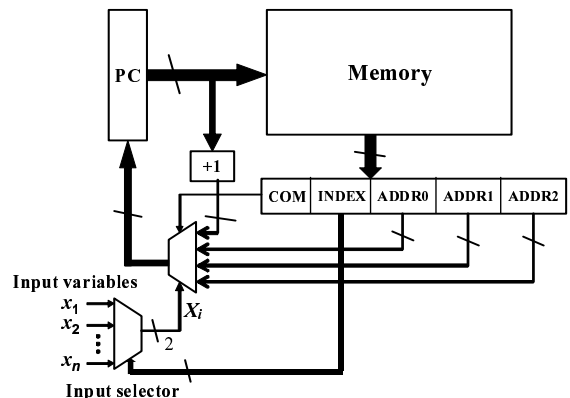


図 5 3 アドレス QDD マシンのアーキテクチャ

4.3 パス数最小化アルゴリズム

与えられた QDD について、パス数を最小化するアルゴリズムを示す。ただし M は、4.1 節で定義したマッチングである。また、初期マッチング時において、出て行く枝について M に飽和されていない節点を探索開始節点と呼ぶ。

アルゴリズム 1 : パス数最小化アルゴリズム

入力 : QDD と任意の初期マッチング M

出力 : QDD におけるパス数を最小化したパス被覆

方法 : M -増大道の探索

- (1) 探索開始節点のリストを作る。
- (2) リストの節点から開始して、入ってくる枝について M に飽和されていない節点までの M -増大道 (P) を探す (深さ優先探索)。
- (3) P が見つかった場合は、 P 上の M に属する枝は M に属さない枝に、 M に属さない枝は M に属する枝としたものを新しい M とする。
 P が見つからなかった場合は、リストの次の探索開始節点からの増大道を探す。
- (4) 全ての探索開始節点について、(2), (3) を繰り返す。
- (5) M に属する枝を辿るパスで、パス被覆を構成する。

4.4 M -増大道探索の枝刈り

パス数最小化アルゴリズムでは、 M -増大道の探索が主な処理であるため、無駄な探索 (M -増大道が見つかる可能性のない探索) の枝刈りを行い効率化を図る。

枝刈りのための準備として、配列 mark を用意する。配列 mark の要素数は、探索開始節点の出次数の合計+1 とする。探索途中に通った節点に付ける印 (now-sign) は非負整数とし、この印の値を配列 mark のインデックスに対応させる。配列 mark の各要素の値は、OK か NG のいずれかとする。

M -増大道探索時に通過する節点を区別するため、出て行く枝が探索道上にある節点を上側節点、入ってくる枝が探索道上にある節点を下側節点と定義する。また、探索開始節点に隣接する下側節点の数 (出次数) を outdeg 、そのうち、探索済み (M -増大道未発見) の下側節点の数を searched で表す。

アルゴリズム 2 : 枝刈りの手順

- (1) 初期状態では、 mark の要素は全て OK、付ける印 (now-sign) は 0 とし、QDD の全ての節点の印 (sign) は 0 とする。
- (2) 探索開始節点について
探索開始時に now-sign を +1、 searched を 0 とする。その後は順次、隣接する下側節点の先を、 M -増大道を発見するまで探索していく。
 - (2-1) M -増大道を発見した場合 ;
 M を更新し、他の探索開始節点に移る。
 - (2-2) M -増大道未発見の場合 ;
 searched を +1、 $\text{mark}[\text{now-sign}] = \text{NG}$
 - (a) ($\text{searched} < \text{outdeg} - 1$) の場合 ;
 now-sign を +1、次の下側節点の先を探索。
 - (b) ($\text{searched} = \text{outdeg} - 1$) の場合 ;
次の下側節点の先を探索し、出次数が 3 以上

の上側節点に初めて到達した時点で、
 now-sign を +1。探索はその後も継続する。

- (c) ($\text{searched} = \text{outdeg}$) の場合 ;
他の探索開始節点に移る。
- (3) 探索中に通過する上側節点について (探索開始節点以外)
 M に属さない枝で隣接する全ての下側節点について、その印 (sign) が、 $\text{mark}[\text{sign}] = \text{NG}$ を満たす場合、 M に属する枝で隣接する下側節点に sign の印を付ける。
- (4) 探索中に通過する下側節点について
 - (a) ($\text{mark}[\text{sign}] = \text{OK}$) の場合 ;
 $\text{sign} = \text{now-sign}$ とし、先に探索を進める。
 - (b) ($\text{sign} = \text{now-sign}$ or $\text{mark}[\text{sign}] = \text{NG}$) の場合 ;
それ以上先は探索しない。

4.5 最小性の証明

アルゴリズム 1 が最小パス被覆を求めることを証明するため、まず、補題 4.2, 補題 4.3, 補題 4.4 を導く。各補題における M は、出入りの異なる枝の隣接を許容したマッチングであり、パス被覆は、 M に属する枝を辿ったパスで構成する。

[補題 4.2] アルゴリズム 1 適用後の $M \Rightarrow M$ -増大道が存在しない

(証明) アルゴリズム 1 における M -増大道の探索は、全ての探索開始節点に対して、それぞれ一度だけ行う。1 つの探索開始節点についての M -増大道探索は、見つかる可能性のある全ての交互道を調べ終えるまで実行する。一度 M -増大道がないと判断した探索開始節点については、その後 M が更新されても、新たな M -増大道が現れることはない (仮に新たな M -増大道が現れたとすると、一回目の探索時に、到達できる全ての交互道を調べていないことになり矛盾)。一方、 M -増大道が見つかった探索開始節点は、 M の更新により、出て行く枝について M に飽和される。結果として、全ての探索開始節点に対して M -増大道の探索が終了した時点では、出て行く枝について M に飽和されていない上側節点からの M -増大道は存在しなくなる。 (証明終)

[補題 4.3] M -増大道が存在しない $\Rightarrow M$ が最大マッチング

(証明) 根を除く全ての非終端節点を、それぞれ入ってくる枝のみと接続する節点と、出て行く枝のみと接続する節点の 2 節点に分解し、そのグラフを $G'(V', E')$ とする。ここで、出て行く枝のみと接続する節点の集合を X 、入ってくる枝のみと接続する節点の集合を Y とすると、 $X \cup Y = V'$ 、 $X \cap Y = \phi$ となり、有向枝であることを許せば、 G' は二部グラフとなる。二部グラフ G' においては、C. ベルジュの定理 [4] により、「 M' -増大道が存在しない $\Leftrightarrow M'$ が最大マッチング」が成り立つ。ただし、 M' とは、枝の隣接を許容しない通常のマッチングである。節点を分解しても、枝については、出入りの異なる枝同士が隣接しなくなるだけで、 E と E' の要素 (枝) 間には 1 対 1 の対応関係がある。マッチングについても G と G' には、互いに 1 対 1 で対応するマッチング M, M' が存在する。よって、明らかに「 M' が最大マッチング $\Rightarrow M$ が最大マッチング」が成り立つ。また、

G における M -増大道路の場合、 M に属さない枝は、枝の向きと同じ方向に、 M に属する枝は、枝の向きと逆向きに辿るが、これは二部グラフ G' においては、 X の節点と Y の節点を交互に通過することに相当し、節点の分解の影響は受けない。よって増大道路についても対応関係が存在し、「 G' に M -増大道路が存在する \Rightarrow G に M -増大道路が存在する」が成り立つ。(証明終)

[補題 4.4] M が最大マッチング \Rightarrow 最小パス被覆である

(証明) マッチング M に属さない枝を、 M に属する枝に変えると、その枝に接続する 2 節点は同じパスに属することになる。このとき、この 2 節点は、元々は別々のパスに属している。仮に元々同じパスに属していたとすると、出て行く枝の内 M に属する枝を 2 つ以上持つ節点が存在することになり、QDD におけるマッチングの条件に反する。また、QDD は非環状の有向グラフなので、 M に属する枝が環状になることもない。つまり、 M に属する枝を 1 つ追加すると、パス数はちょうど 1 つ減る。この性質と数学的帰納法により、「パス数 + $|M| = QDD$ の節点数」という関係が成立する。与えられた QDD において、その節点数は定数なので、 $|M|$ が最大のとき、パス数は最小となる。(証明終)

補題 4.2, 4.3, 4.4 より、以下の定理 4.2 が成り立つ。

[定理 4.2] アルゴリズム 1 は、最小パス被覆を求める。

(証明) 補題 4.2, 補題 4.3, 補題 4.4 より明らか。(証明終)

4.6 アルゴリズム 1 の時間計算量

与えられた QDD の節点数を N とすると、アルゴリズム 1 の時間計算量について、定理 4.3 が成り立つ。

[定理 4.3] アルゴリズム 1 の時間計算量は、 $O(N^2)$ である。

(証明) 任意に初期マッチングをとった場合、探索開始節点数は最大で非終端節点数 N_N となる。すなわち M -増大道路の探索は最大で N_N 回なので、繰り返し回数については $O(N)$ となる。また、 M -増大道路の探索においては、 M -増大道路が見つかるまでは、見つかる可能性のある全ての範囲を調べる。同一枝の探索を回避するため、各節点に印をつけながら探索するので、一つの探索(特定の探索開始節点からの探索)において同じ枝を 2 度通ることはない。つまり最悪の場合でも全ての枝を一度だけ探索したところで、 M -増大道路があるかどうかを判明する。QDD における非終端節点の出次数は 4 であるため、総枝数は $4N_N$ となる。よって、 M -増大道路の探索の時間計算量は $O(N)$ である。以上のことから、アルゴリズム 1 の時間計算量は $O(N^2)$ となる。(証明終)

5. 実験結果

パス数最小化アルゴリズムの有効性を調べるために、種々のベンチマーク関数を表現する 3 アドレス QDD マシンのコードを最小化した。

5.1 実験環境

C 言語で記述したコード最適化プログラムを、Cygwin の gcc

でコンパイルし、Windows XP Professional SP2 環境下で実行した。CPU は Intel Core 2 Duo E6420 (2.13GHz) を用いた。

5.2 実験内容

実験は、条件分岐命令として図 6 の Branch0 のみを用いた場合 (1type) と、4 種類全てを使用可能とした場合 (4type) の 3 アドレス QDD マシンについて、入力数が 130 以下のベンチマーク関数 163 種類に対して行った。

5.2.1 節点数とコードサイズ

各ベンチマーク関数について、QDD の節点数を調べ、1type においては、コードサイズ (1type CSIZE) と無条件分岐命令数 (1type GOTO) を、4type においては、アルゴリズム 1 を適用して最適化したコードサイズ (4type CSIZE) と無条件分岐命令数 (4type GOTO) を調べた。これらの測定結果を表 1 に示す。

5.2.2 アルゴリズムの実行時間

4type の 3 アドレス QDD マシンについて、コード最適化処理の実行時間を計測した。ただし、実行時間は 100 回の計測の平均値であり、初期マッチングは空集合とした。QDD の節点数と実行時間の関係を図 11 に示す。

5.3 実験結果と考察

5.3.1 BDD と QDD の節点数

表 1 において、実験した全てのベンチマーク関数について、QDD の節点数の方が、BDD の節点数よりも小さくなっている。これは元の BDD の 2 値変数を、4 値超変数としてまとめたためであり、1 個の QDD の節点は、1~3 個の BDD の節点をグループ化したものとなる。よって、QDD の非終端節点数は、元の BDD の非終端節点数に対して最悪で同数、最高で 1/3 まで減少する。

5.3.2 アルゴリズム適用後のコードサイズ

1type と 4type の 3 アドレス QDD マシンを比較すると、4type の方がコードサイズと無条件分岐命令数が少ないことがわかる。特に、ex1010 や misj はアルゴリズム 1 を適用したことによって無条件分岐命令数を 0 にすることができている。他方、intb や shift などはアルゴリズム 1 適用後も依然として無条件分岐命令が多く残っている。misj と比較して、QDD 節点数の少ない intb の方が、無条件分岐命令が多く残っていることから分かるように、無条件分岐命令数は、QDD の節点数だけでなく、QDD の形状 (QDD の最大幅や終端節点数) や、QDD の枝の数等にも依存している。

5.3.3 アルゴリズム 1 の実行時間

図 11 において、4type の 3 アドレス QDD マシンに対してのアルゴリズム 1 の実行時間は多くのベンチマーク関数では数十 msec 以下となったが、ts10, shift, jbp などの比較的節点数の多いベンチマーク関数の場合は実行時間は長くなった。

図 11 では、QDD の節点数が増えると実行時間は増加する傾向にあるが、分布には多少ばらつきがあり、定理 4.3 で示したアルゴリズムの時間計算量 $O(N^2)$ が実験値と適合するかどうか確かめるためには、さらに多くのベンチマーク関数 (特に入力

数の大きい関数) についても実行時間を計測する必要がある。

文 献

- [1] T. Sasao, H. Nakahara, M. Matsuura, Y. Kawamura and Jon T. Butler, "A quaternary decision diagram machine and the optimization of its code," *International Symposium of Multiple-Valued Logic (ISMVL-2009)*, May 20-23, 2009.
- [2] 夜久竹夫, 植村憲治, 山神憲司, "ある種の非サイクル的有向グラフの極大パス被覆を与える線形時間アルゴリズム," *電子情報通信学会論文誌*, Vol.J 75-D-I No.10, 1992.10.
- [3] S. Nagayama, and T. Sasao, Y. Iguchi and M. Matsuura, "Area-time complexities of multi-valued decision diagrams," *IEICE Transactions on Fundamentals of Electronics*, Vol. E87-A, No.5, pp. 1020-1028, May, 2004.
- [4] 土屋守正, 恵羅博 共著, 『シリーズ 情報科学の数学 グラフ理論 初版』, 産業図書株式会社, 2000.

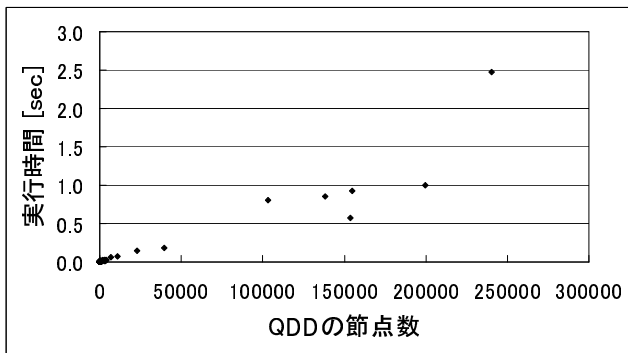


図 11 QDD の節点数と実行時間

表 1 BDD と QDD の節点数とコードサイズ

Func name	In	Out	BDD nodes	QDD nodes	1type csize	1type goto	4type csize	4type goto
apex4	9	19	749	600	639	39	601	1
b4	33	23	4237	3473	3734	261	3482	9
count	35	16	196625	153952	170337	16385	153953	1
ex1010	10	10	1548	892	949	57	892	0
in6	33	23	4325	3546	3815	269	3555	9
intb	15	7	709	401	567	166	479	78
misj	35	14	6024	4072	5297	1225	4072	0
mlp6	12	12	5270	2582	2966	384	2694	112
mlp9	18	18	329224	154803	196228	41425	163622	8819
shift	19	16	196095	138069	159701	21632	147413	9344
t1	21	23	4543	3997	4009	12	3997	0
tial	14	8	697	388	552	164	466	78
ts10	22	16	589837	240303	317238	76935	305839	65536
x7dn	66	15	307503	199444	253345	53901	229396	29952

6. まとめと今後の課題

本論文では, 3 アドレス QDD マシン用コードに含まれる冗長な無条件分岐命令の削減法として, 最小パス被覆を求める手法を提案した. このアルゴリズムを適用することにより, 本研究で実験に使用したベンチマーク関数において, 無条件分岐命令数を最小化し, コードサイズを削減できることを確認した. また, アルゴリズムの時間計算量は $O(N^2)$ となることを示した.

今後は, より入力数の大きいベンチマーク関数についての実験や, 実行時間が理論と一致するかどうかの検証実験を行う予定である. また, アルゴリズムの改良 (時間計算量の改善) を考えている.

謝辞 本研究は, 一部, 日本学術振興会・科学研究費補助金, および, 文化科学省・知的クラスター創成事業 (第二期) の補助金による. 有益な助言を頂いた中原啓貴博士, および笹尾研究室の皆様へ感謝する.