部分グラフ同型判定アルゴリズムの
FPGAを用いた実装手法

知識情報工学専攻 931709 小西幸治
指導教官 市川周一

目 次

1　はじめに

部分グラフ同型判定は，化学構造式で記述された化学物質中に指定された化学結合が存在するかどうかの判定や，シーン解析すなわちパタン認識など多くの分野に応用可能である [1]．しかし一般的に部分グラフ同型問題は NP 完全であり [2]，効率的に解くアルゴリズムは存在しない．

そこで部分グラフ同型問題をプログラム可能な論理デバイスの 1 つである FPGA(Field Programmable Gate Array) に実装し，高速化することにした．FPGA を使用することで，汎用計算機より高性能な専用アーキテクチャが実現可能となる [3]．しかも SRAM 型 FPGA は事実上，無限回の論理書き換えが可能であり，ASIC(Application Specific Integrated Circuit) 等の専用回路より簡単かつ安価に実現可能である．本研究では Lucent 社の ORCA OR2CxxA シリーズ FPGA[4] を 2 個搭載する PCI バス用再構成可能論理ボード OPERL[5] を使用し，OPERL のユーザー FPGA 1 個に部分グラフ同型問題を解くアルゴリズムを実装する．現段階では FPGA への実装は完了していないが，手法の提案および性能の見積りを行う．

以下，2 章では部分グラフ同型の定義を行う．3 章では写像と探索木の関係および探索空間削減の必要性について説明する．4 章では性能比較対象となる Ullmann のアルゴリズム [1] と，同アルゴリズムで探索空間の削減を行う refinement procedure について説明する．5 章では部分グラフ同型判定アルゴリズムを実装する ORCA OR2CxxA シリーズ FPGA について説明したあとで，同 FPGA を搭載する OPERL ボードについて触れる．6 章では提案アルゴリズムの探索空間削減法を説明し，アルゴリズムの説明を行う．その後リソース使用量の見積りと適用可能なグラフ規模について述べる．7 章では提案アルゴリズムと refinement procedure 両方式の性能比較を行う．8 章では OR2CxxA FPGA が持つ動的再構成を利用した拡張方法について述べる．

2　部分グラフ同型問題

ここではグラフ用語について簡単に説明し，部分グラフ同型 (subgraph isomorphism) と部分グラフ同型問題を定義する．

2.1　グラフ用語の説明

グラフ用語に関しては，[1] で使用されている用語を本論文でも使用する．

あるグラフを G と表現しよう．グラフ G は，頂点 (vertex) とよばれる有限個の p 要素からなる集合 $V = [v_i](1 \leq i \leq p)$ と，2 頂点の組 $v_i, v_j \in V$ で構成される辺 (edge) が有限個の q 要素からなる集合 $E = [e_j](1 \leq j \leq q)$ で表現される．一般にこれらをまとめて $G = (V,\ E)$ と記述する．以下では，他のグラフに対しても同様の記法を用いる．辺を構成する 2 頂点の順番が考慮されない，つまり辺に向きがないグラフは無向グラフ (non-directed graph) とよばれる．これに対し，辺を構成する 2 頂点の順番を考慮するグラフを有向グラフ (directed graph) という．この論文では無向グラフを扱い，同じ 2 頂点で構成される辺が複数存在しないものとする．さらに自分自身の頂点と結合している辺は扱わない．

図 1に簡単なグラフを示し，具体的に説明する．図中の● は頂点を表し，頂点間の線分は辺を表している．なお● は図 1に限って示し，これ以降の図では頂点を明示する場合以外は記さない．このグラフの頂点集合 V は a，b，c と d から構成されている．また辺集合 E は 1，2，3，4 と 5 で構成されており，実際には頂点の組 {a, b}，{a, d}，{b, c}，{b, d} と {c, d} で表現される．頂点数は 4 なので $p = 4$，辺数は 5 なので $q = 5$ である．

辺集合 E を表現する方法として隣接行列 (adjacency matrix) がある．ここでは G の隣接行列を A と表現することにする．$A = [a_{ij}]$ は $p \times p$ 個の要素からなり，無向グラフの場合は各要素 a_{ij} が以下の条件によって決定される．

$$a_{ij} \quad = \quad \begin{cases} 1, & \{i,\ j\}\ or\ \{j,\ i\} \in E \\ 0, & \{i,\ j\}\ and\ \{j,\ i\} \notin E \quad (1 \leq i,j \leq p) \end{cases}$$

図 1の頂点をそれぞれ a → 1，b → 2，c → 3，d → 4 と置きかえると (以下でも同じように表現する)，隣接行列 A は以下のようになる．

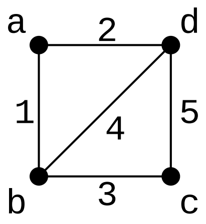


図 1: グラフ $G = (V,\ E)$

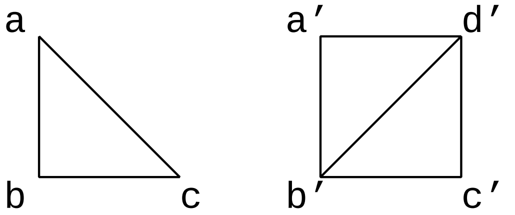


図 2: 部分グラフ同型が存在するグラフ G_α(左) と G_β

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

次に次数 (degree) の定義を行う．次数はある頂点 $v_i(1 \leq i \leq p)$ に接続している辺の数である．ここで隣接行列に注目してみる．a_{ix}が 1 ということは，片方の頂点が v_iである辺が存在することを意味している．つまり i 行目にある 1 の数を合計すれば，それが頂点 v_iの次数となる．このことから，頂点 v_iの次数を deg_iとすると以下のようになる．

$$deg_i = \sum_{x=1}^{p} a_{ix}$$

図 1で各頂点の次数は，それぞれ $deg_1 = 2$，$deg_2 = 3$，$deg_3 = 2$，$deg_4 = 3$ である．

最後に辺密度 (edge density) という用語の定義をする．辺密度とはグラフ G の隣接行列で 1 の成分の割合である．ただし対角成分は除く．これは頂点数 p の完全グラフ $K(p)$ の辺数に対して実際に存在する辺数の比ということもできる．先程例に出した隣接行列 A では，辺密度は$\frac{10}{12} = \frac{5}{6}$となる．

2.2 部分グラフ同型および部分グラフ同型問題の定義

部分グラフ同型問題は 2 個のグラフを扱う．それぞれ $G_\alpha = (V_\alpha,\ E_\alpha)$，$G_\beta = (V_\beta,\ E_\beta)$ とし，頂点数を p_α ,$p_\beta(p_\alpha \leq p_\beta)$，辺数を q_α ,$q_\beta(q_\alpha \leq q_\beta)$ とする．

この論文で部分グラフ同型とは，G_βの部分グラフと G_αが同型であることを示す．同型とは，両グラフの全頂点と全辺に対して完全な 1 対 1 対応がある場合のことである．$V \subseteq V_\beta, E \subseteq E_\beta$とすると，部分グラフ同型の条件は以下のようになる．

$$\forall i, j(\{v_{\alpha i},\ v_{\alpha j}\} \in E_\alpha \Rightarrow \{f(v_{\alpha i}),\ f(v_{\alpha j})\} \in E) \text{ となる写像 } f : V_\alpha \to V \text{が存在} \tag{1}$$

具体例をだして部分グラフ同型を説明する．図 2は部分グラフ同型が存在するグラフ G_α ,G_βの例である．部分グラフ同型なものを検出するには，(1) 式が成立するような写像 fを見つければよい．例えば $f(a) = a'$,$f(b) = b'$,$f(c) = d'$という写像は部分グラフ同型となる．この時 {a,b} は {a’,b’}，{a,c} は {a’,d’}，{b,c} は {b’,d’} に写像され (1) 式を満たしている．

3　部分グラフ同型判定アルゴリズムの概念

この章では，写像と探索木 (search tree) の関係を説明し，探索空間すなわち探索木の探索方法について触れる．その後，探索空間削減の必要性について述べる．

3.1　写像と探索木の関係

(1) 式をみると $V_\alpha \rightarrow V$ の写像 f を決めなければならないことがわかる．V_α の全要素に対して一度に f が決定されるわけではなく，$v_{\alpha 1}, v_{\alpha 2}, ..., v_{\alpha p_\alpha}$ の順番で決定され，f の選び方は $_{p_\beta}P_{p_\alpha}$ 通りある．

ここで写像を探索木上に表現することを考える．深さ $i(1 \leq i \leq p_\alpha)$ の節点に，$v_{\alpha i}$ が f によって写像された頂点 $f(v_{\alpha i})$ を示す．もちろん 1 度使用された写像を再び使用することはできない．全ての f に対してこの作業を行うと，全体として深さ p_α の探索木ができる．図 2に対する探索木は図 3のようになる．以下で “深さ d の写像” と書かれた場合は，$v_{\alpha d}$ からの写像を表す．

f は逐次的にしか決定できないため，何らかの方法で探索木を逐次探索する必要が生じる．そこで本論文では深さ優先探索 (depth-first search) を用いることにした．

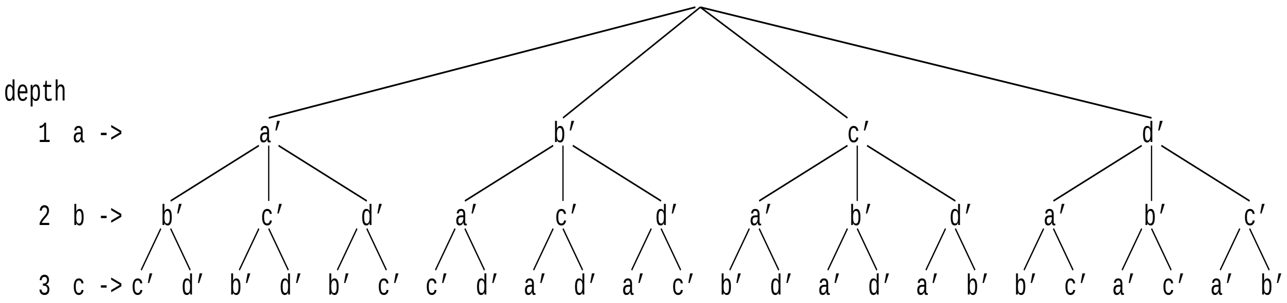


図 3: 写像と探索木

3.2　探索空間削減の必要性

先程 f の選び方は $_{p_\beta}P_{p_\alpha} = p_\beta \times (p_\beta - 1) \times \cdots \times (p_\beta - p_\alpha + 2) \times (p_\beta - p_\alpha + 1)$ 通り存在すると書いた．これは p_α と p_β が増加すると実用時間では計算不可能となることを示している．計算量を少なくするため，言い換えれば実用上計算不可能となる p_α ,p_β の値を増大させるために，調べるべき探索木の範囲 (探索空間) の削減を行う必要がある．探索空間削減は葉以外の節点において部分グラフ同型となり得ない写像を判断し，該当する節点以下を巡回しないことにより計算量を減少させることである．探索空間の削減方法に付いては後程説明する．

4　Ullmann の部分グラフ同型判定アルゴリズム

この章では Ullmann の refinement procedure[1] を用いた部分グラフ同型判定アルゴリズムを説明する．まず探索空間削減を行う refinement procedure を 4.1節で説明し，続いて探索木を巡回するアルゴリズムを 4.2節で説明する．

4.1　Refinement procedure

4.1.1　使用する記号

Refinement procedure を図 4に示す．

$B = [b_{ij}](1 \leq i, j \leq p_\beta)$ は G_β の接続行列である．B_j は配列 B の j 列目を取り出したものを表す．$M = [m_{ij}]$ は $p_\alpha \times p_\beta$ の大きさを持つ行列である．$m_{ij} = 1$ は $v_{\alpha i}$ から $v_{\beta j}$ へ写像したときに，部分グラフ同型である可能性があることを示す．深さ $d < p_\alpha$ つまり探索木中の節点で M_d は写像 f に対応する要素を 1，その他の要素を 0 にする．

```
$elim := 0$;
do
{
    $i := 1$;
    while ($i \le p_\alpha$)
    {
        $j := 1$;
        while ($j \le p_\beta$)
        {
            $sc := 2^{p_\beta - 1}$;
            if (($M_i \& sc) \ne 0$)
            {
                $h := 1$;
                while ($h \le deg_i$)
                {
                    $x := lst_h$;
                    if (($M_i \& B_j) = 0$)
                    {
                        $M_i := M_i \& NOT\ sc$;
                        $elim := elim + 1$;
                        goto P1;
                    }
                    $h := h + 1$;
                }
            }
P1:         $sc := sc/2$;
            $j := j + 1$;
        }
        if ($M_i = 0$) return(FAIL);
        $i = i + 1$;
    }
} while ($elim \ne 0$);
return(SUCCEED);
```

図 4: Refinement Procedure[1]

$v_{\alpha i}$と$v_{\beta j}$の1対1対応を探しているので，1つの部分グラフ同型を発見した場合には，Mの各行には1が1個しか存在せず，かつMの各列には最大でも1が1個しか存在しない．図4でM_iは配列Mのi列目を取り出したものを表す．

またh ,i ,jとxは整数のポインタである．$elim$はrefinement procedureを実行後，削除された候補数である．削除の方法に関しては後ほど説明する．deg_iは$v_{\alpha i}$の次数である．lstは$v_{\alpha i}$と隣接する頂点集合である．scはスキャンするために用いられる1ビットのみが1となるデータである．全ての変数について，ある行で最も右の要素や，ある列で最も下の要素が，ワード中の最下位ビット(LSB)にあたる．$\&$は論理和を表し，NOTは反転を表す(例: $1100\ \&\ 1010 = 1000$．$NOT\ 1100 = 0011$)．

なお図中の$:=$は代入を表す．例えば$d := d + 1$ならばdに$d + 1$を代入するという意味になる．

4.1.2 Refinement Procedureの探索空間削減

Refinement procedureでは，部分グラフ同型を検出するための計算量を減らすために配列M中の1を0に変化させる，つまり探索空間の削除を行う．

ここで探索木中の葉でない節点，つまり深さ$d < p_\alpha$に対応する配列Mについて考えてみる．もしMが存在する節点以下に部分グラフ同型となるものが存在しないならば，現在注目している写像m_{ij}を0にしても問題はない．

以下に$m_{ij} = 1 \to 0$にできる条件を示す．$v_{\alpha i}$と隣接する頂点の集合が$\{v_{\alpha 1}, ..., v_{\alpha x}, ..., v_{\alpha\gamma}\}$であるとすれば，部分グラフ同型の定義から$v_{\alpha x}(x = 1, ..., \gamma)$に対応する$v_{\beta y}$が$V_\beta$中に存在し，かつ$v_{\beta y}$は$v_{\beta j}$と隣接していないといけないことは自明である．もちろん$v_{\beta y}$はまだ写像されない頂点でなければならない．これは(2)式で表現できる．

$$\underset{1 \le x \le p_\alpha}{\forall x}\ (\ (a_{ix} = 1)\ \Rightarrow\ \underset{1 \le y \le p_\beta}{(\exists y)}\ (m_{xy} \cdot b_{yj} = 1)\ ) \tag{2}$$

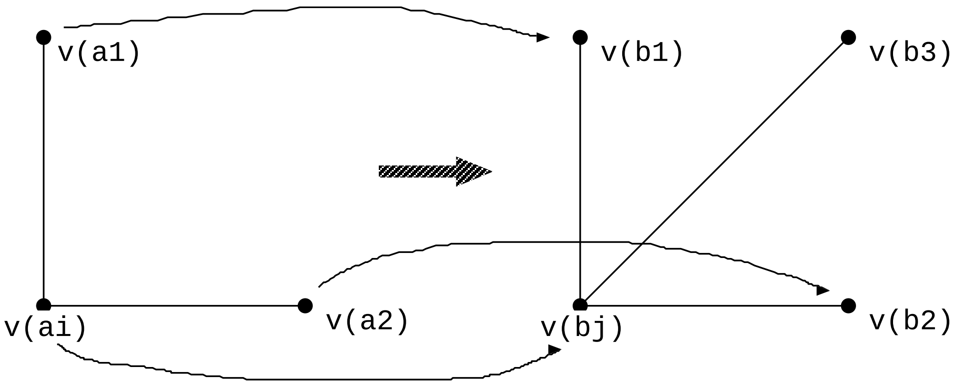


図 5: refinement procedureの処理イメージ

(2)式を満たす例を図5に示す．$v(ai)$が$v_{\alpha i}$に，$v(bj)$が$v_{\beta j}$に相当する頂点である．$v(a1),\ v(a2)$が$v_{\alpha 1},\ v_{\alpha 2}$に，$v(b1),\ v(b2),\ v(b3)$が$v_{\beta 1},\ v_{\beta 2},\ v_{\beta 3}$に相当する．ただし$m_{v(ax)v(by)} = 1(1 \le x \le 2,\ 1 \le y \le 3)$とする．$v(ai)$と隣接した頂点$v(a1),\ v(a2)$に対応する$v(bj)$と隣接した頂点が存在すれば(2)式を満たす．図5には$v(bj)$と隣接する頂点が3個存在し，$v(a1),\ v(a2)$に対応する頂点が存在することがわかる．よってこの図の例では(2)式を満たしている．

Refinement procedureは単に(2)式を満たしているかどうかを調べているにすぎない．(2)式を満たさないm_{ij}は$m_{ij} = 1$から$m_{ij} = 0$に変更される．ただし(2)式では配列Mも参照しているので，$m_{ij} = 1$から$m_{ij} = 0$へ変更されたことにより，再度refinement procedureを実行した時にMが変化するかもしれない．しかし$m_{ij} = 0$となる写像を用いても，早かれ遅かれ部分グラフ同型とはならないことが判明するので，refinement procedureは配列Mが変化しなくなるまで繰り返し実行する．

refinement procedureを実行することにより，Mのある行(例えばi行目とする)の要素全てが0になることがある．これは$v_{\alpha i}$の写像としてV_βのどの頂点を選択しても，部分グラフ同型にはならないことを示している．

```
Step 1.  $M := M^0$;  $d := 1$;  $H_1 := 0$;
         for all $i := 1, ..., p_\alpha$ set $F_i := 0$;
         make $lst_i$ and $deg_i$ for all $i := 1, ..., p_\alpha$;
         refine $M$; if exit FAIL then go to step 7;
Step 2.  If there is no value of $j$ such that $m_{dj} = 1$ and $f_j = 0$ then go to step 7;
         $M_d := M$;  $k := 0$;
Step 3.  $k := k + 1$;
         if $m_{dk} = 0$ or $f_k = 1$ then go to step 3;
         for all $j \neq k$ set $m_{dj} := 0$;
         refine $M$; if exit FAIL then go to step 5;
Step 4.  if $d < p_\alpha$ then go to step 6 else give output to indicate that an isomorphism has been found;
Step 5.  If there is no $j > k$ such that $M_d(d, j) = 1$ and $f_j = 0$ then go to step 7;
         $M := M_d$; go to step 3;
Step 6.  $H_d := k$;  $F_k := 1$;  $d := d + 1$;
         go to step 2;
Step 7.  If $d = 1$ then terminate algorithm;
         $d := d - 1$;  $M := M_d$;  $k := H_d$;
         $F_j := 0$, if $j \neq H_i$ for all $i < d$;
```

図 6: 探索木巡回アルゴリズム [1]

Refinement procedure を実行中にこのような状態になると，FAIL exit にジャンプして refinement procedure の実行を中止する．

部分グラフ同型という判定は，深さ $d = p_\alpha$で refinement procedure を実行した時に Mの内容が変化しない場合に下される．つまり (2) 式を満たしているということであり，V_αから V_βへの 1 対 1 対応と，E_αから E_βへの 1 対 1 対応がある場合にしか起こりえない．これは部分グラフ同型の定義そのものである．

4.2 探索木巡回アルゴリズム

探索木巡回アルゴリズムを図 6に示す．基本的には (根付木の) 深さ優先探索と同じアルゴリズムである．4.1.1 節で説明されていない記号についてこの節で説明する．

$A = [a_{ij}]$ は G_αの隣接行列であり，A_iは配列 A の i 行目を取り出したものを表す．

$M^0 = [m^0_{ij}]$ は配列 Mの初期値であり，以下のように決定される．

$$m^0_{ij} \begin{cases} = 1 & deg(v_{\beta j}) \geq deg(v_{\alpha i}) \\ = 0 & \text{それ以外} \end{cases}$$

$m^0_{ij} = 0$ となるような写像をしても，$v_{\alpha i}$と隣接した頂点に対応する $v_{\beta j}$と隣接する頂点数が足りなくなるので，部分グラフ同型になりえない．

$\{F_1, ..., F_i, ..., F_{p_\beta}\}$ は p_βビットの 2 進数ベクトルであり，$F_i = 1$ は Mの i 列目，すなわち写像先として $v_{\beta i}$が既に使用されていることを表す．$\{H_1, ..., H_d, ..., H_{p_\alpha}\}$ は p_α個の要素を持つベクトルであり，$H_d = k$は深さ d で Mの k列目，すなわち $v_{\alpha d}$の写像として $v_{\beta k}$が設定されていることを表す．

なお $M := M^0$のように行列の代入が記述された場合には，全ての要素を代入することを示す．M_dは配列 Mと同じ大きさの行列である．m_{ij} は配列 Mの i 行 j列の要素を示し，$M_d(i, j)$ は配列 M_dの i 行 j列の要素を示す．“make lst_i and deg_i” は G_αの頂点に関して，隣接頂点集合と次数を作成するという意味である (4.1.1節参照)．“refine M” は refinement procedure を実行する．

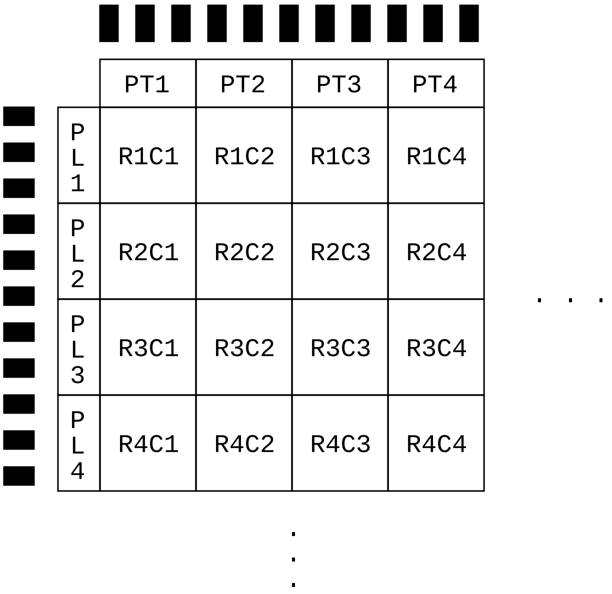


図 7: OR2C15A アレイ [4]

表 1: ORCA OR2CxxA シリーズ FPGA[4]

デバイス	ラッチ/FF	ユーザー I/O 数	アレイサイズ
OR2C04A	400	160	10 × 10
OR2C06A	576	192	12 × 12
OR2C08A	784	224	14 × 14
OR2C10A	1024	256	16 × 16
OR2C12A	1296	288	18 × 18
OR2C15A	1600	320	20 × 20
OR2C26A	2304	384	24 × 24
OR2C40A	3600	480	30 × 30

5 ORCA OR2CxxA FPGA と OPERL ボード

この章では 6章で提案する部分グラフ同型判定アルゴリズムを実装する Lucent Technologies 社の ORCA OR2CxxA シリーズ FPGA[4] と，同 FPGA を搭載し PCI バスに付加して使用する OPERL ボード [5] について説明する．

5.1 ORCA OR2CxxA FPGA

図 7に ORCA OR2C15A FPGA のアレイの一部を示す．OR2CxxA シリーズ FPGA には表 1に示すように 8 種類存在するが，基本的な構造は全て同じである．ORCA OR2CxxA シリーズ FPGA は，PLC(Programmable Logic Cell) と PIC(Programmable Input/Output Cell) の基本セル要素から構成されている．PLC は 2 次元配列状に配置されており，配置場所により RxCy(x 行 y 列目) というラベルが付けられている．OR2C15A の場合は 20 行 20 列の PLC 配列が用意されている．PLC の周囲には PIC が配置されている．PIC は上下左右に配置されており，配置場所によりラベル付けされている．上部は PTx(最上列の左から x 番目)，下部は PBx(最上列の左から y 番目)，左部は PLx(最左列の上から x 番目)，右部は PRx(最右列の上から x 番目) で表される．

PLC はプログラム可能な機能ユニット (PFU: Programmable Function Unit) と，配線リソースから構成されている (図 8)．PFU は図 9のようになっており，組合せ論理はルックアップテーブル (LUT: Look Up Table) で処

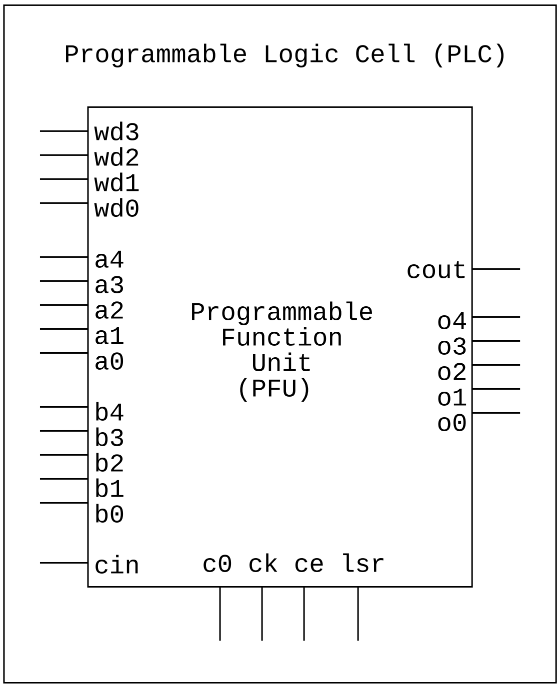


図 8: PFU ポート [4]

理される．4 入力 1 出力つまり 16 エントリの LUT(QLUT とよぶ) が４個あり，1PFU あたり 64 ビット LUT が利用可能である．LUT は各種論理に対応した３モードが用意されており，組合せ論理，リップル，メモリのうち１つで動作する．リップルモードでは高速キャリー論理を利用して加減算機能，乗数機能，拡張データパス機能が実現可能となる．メモリモードを利用すると，16 × 4 ビット RAM(ROM) や 16 × 2 ビットデュアルポートメモリとして使用可能となる．

順序回路として PFU を利用することを考えよう．図 9をみると，PFU 内には 16 × 4 ビット RAM と 4 個の FF(Flip Flop) の２つが存在する．ただし RAM と FF を同時に利用することはできない．このことより，1PFU に記憶できるビット数という点からみると、RAM は FF の 16 倍も多いことがわかる．つまり OR2CxxA FPGA のリソースを有効利用するには，RAM を活用すればよいことになる．実際に 6章で提案する部分グラフ判定アルゴリズムでも，積極的にメモリモードを使用してリソースの節約を図っている．当然だがメモリモードを使用すると，全てのデータに対して同時にアクセスすることが不可能になる．このような目的には FF を使用する他ない．

また ORCA OR2CxxA FPGA が持つ重要な特徴として動的再構成可能というものがある．これは FPGA に電源が供給されている時にも，論理の書き換えが可能という特徴である．FPGA に対して複数の論理データを用意しておけば，短時間に複数の論理データを切替えることにより高速な動作が期待できる．このように実行すると 2 ~ 8 ミリ秒というオーバーヘッドが生じるが，実際に FPGA に実現可能な回路規模以上の論理が実現できることになる．8章では動的再構成を利用した提案アルゴリズムの拡張について触れる．

5.2 OPERL ボード

図 10に OPERL(ORCA PCI Easily Reconfigurable Logic または ORCA PErsonal Reconfigurable Logic) ボード [5] のブロック図を示す．OPERL ボードは PCI FPGA と USER FPGA の２個の FPGA から構成されている．PCI FPGA は PCI バスへのインターフェース機能を持ち，論理は電源投入時に SROM(Serial ROM) で設定する．動的再構成の対象となるのは USER FPGA のみで，論理のプログラムは PCI バス (PCI FPGA) 経由で行う．FPGA には 5.1節で説明した ORCA OR2CxxA シリーズを採用している．ボード上への実装はソケットを用いているため，FPGA の容量は任意に変更可能である．

OPERL ボードは PCI のバスクロックに同期して動作する．最大 33MHz が供給されるので，ボード上の回路

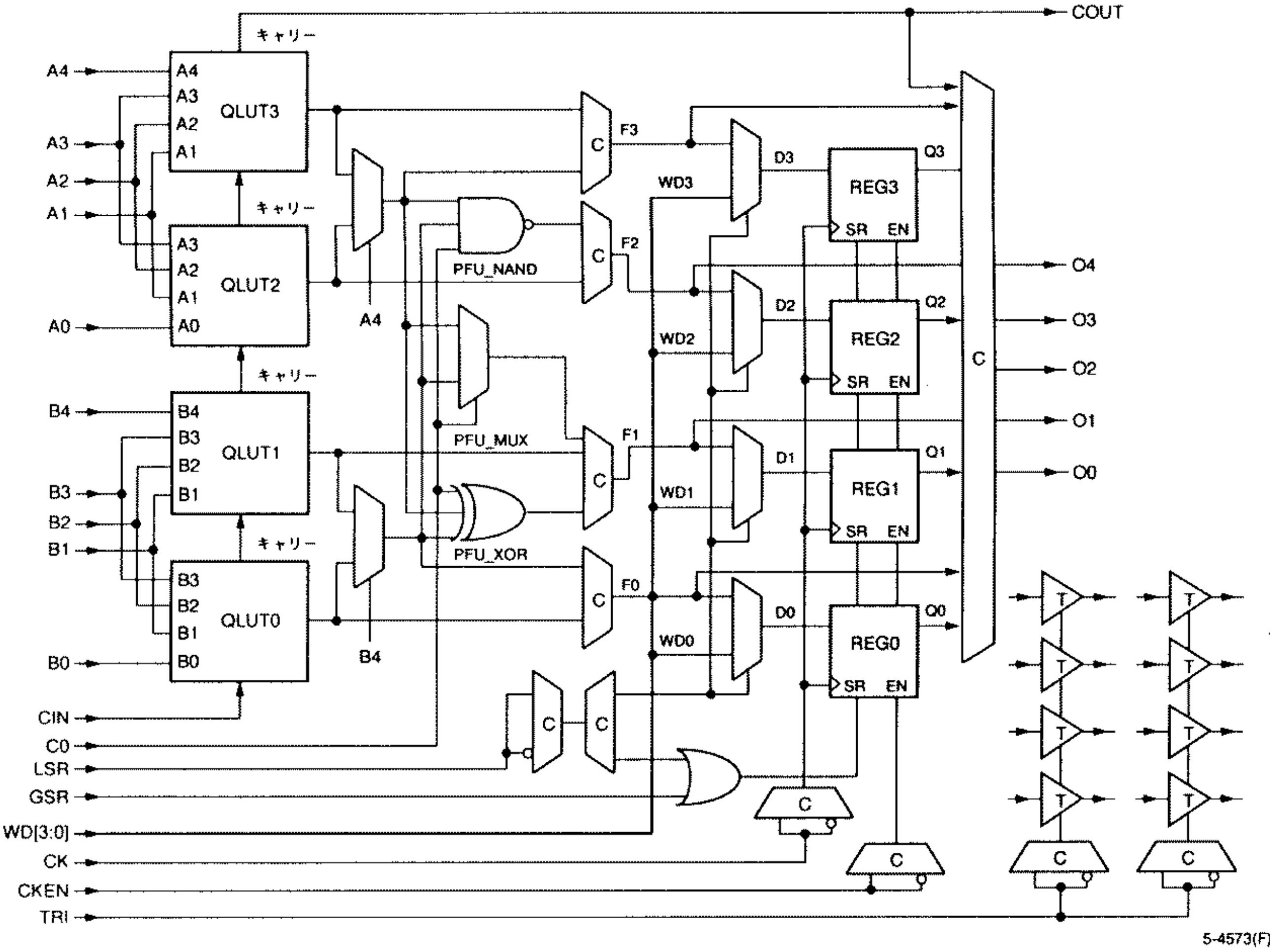


図 9: PFU のブロック図 [4]

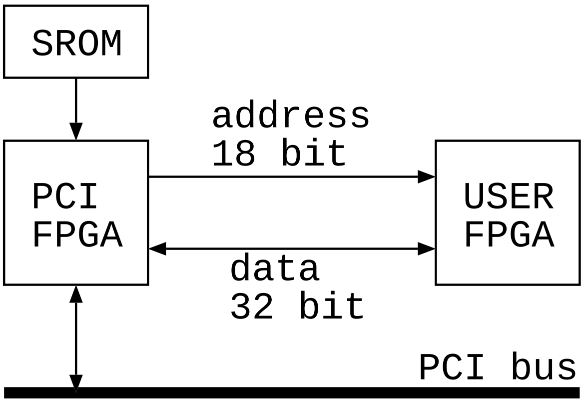


図 10: OPERL ボードのブロック図 [5]

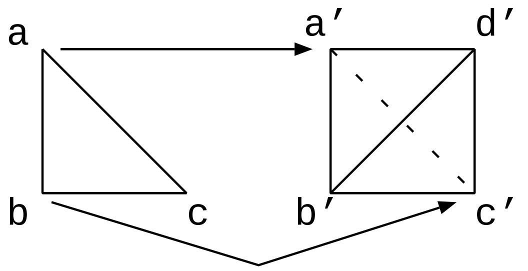


図 11: 部分グラフ同型となり得ない写像の例 (左 G_α，右 G_β)

も最大 33MHz で動作する．

6 OR2CxxA シリーズFPGA に実装する部分グラフ同型判定アルゴリズム

この章ではOPERL ボード [5] 上に搭載されたORCA OR2CxxA シリーズFPGA[4] に実装する，部分グラフ同型検出アルゴリズムを説明する．まず本論文で提案するアルゴリズムで行う探索空間削減法について説明する．次の 2 節で提案アルゴリズムを説明する．6.2節では，G_αの辺に対応するG_βの辺が存在するかどうかを調べる辺存在確認アルゴリズムについて述べる．6.3節では，この部分グラフ同型検出アルゴリズム用に作成された探索木巡回アルゴリズムを説明する．6.4節ではFPGA リソース消費量を定式化する．最後にrefinement procedure をハードウェアに実装する際の問題点をあげ，提案アルゴリズムを採用した理由を述べる．以下，refinement procedure はUllmann の部分グラフ同型判定アルゴリズム [1] を指す．

6.1 提案アルゴリズムの探索空間削減法

3章で写像 fは V_αの 1 要素毎，つまり深さ 1 毎に決定していくことを説明した．Refinement procedure で行っていたように，提案アルゴリズムでも写像が決定される毎に探索空間が削減できるかどうかを調べている．

提案アルゴリズムの探索空間削減は以下のようにして行う．深さ d までの写像が決定した時 $v_{\alpha d}$と $v_{\alpha i}(1 \leq i < d)$ で構成される辺 e_αを写像する．$f(e_\alpha)$ が G_βに含まれていれば部分グラフ同型となり得ると判断し，そのまま探索木の巡回を続ける．言い換えると G_αの部分グラフ G'_αを写像したものが G_βに含まれるかどうかを調べている．G'_αが G_βの部分グラフとなることは，G_αが G_βの部分グラフとなるための必要条件である．Ullmann のアルゴリズムでは部分グラフ同型における必要十分条件を調べているのに対して，提案アルゴリズムでは部分グラフ同型における必要条件を調べているので探索空間削減部分が単純になっている．

例えば図 11で考える．深さ 2 までの写像を a → a’，b → c’ とした時点で，G_βには {a’,c’} が存在しないことがわかる．この時点で辺の 1 対 1 対応が成立しないことになり，c を残りの b’ や d’ に写像しても (1) 式を満たすことは絶対にない．辺が存在するかどうかを調べる作業については，6.2節の辺導出部で説明する．Refinement procedure で行っていることより簡単なものになっているが，これによりハードウェア量が格段に減少する．

図 11を例にとって，全ての写像可能性 (以下 “探索空間”) と削除できる候補を図 12に示す．× で示した候補が，上の方法によって削除される候補である．

6.2 辺存在確認アルゴリズム

辺存在確認アルゴリズムは図 13のようなアルゴリズムである．$etsa$ は 6.2.1節で説明するedge list table starting address であり，eta はedge list table である．Bは G_βの隣接行列である．これを OR2CxxA シリーズ FPGA に実装するための手法を以下に説明する．

辺存在確認アルゴリズムを 2 つの部分に分解する． 1 つは写像 fから $e_{\alpha i}$に対応した $e_{\beta j}$を導出する部分 (以下 “辺導出部”) である．もう 1 つは導出された辺 $e_{\beta j}$が G_βの頂点集合 E_βに含まれているかどうかを調べる部分 (以下 “存在確認部”) である．

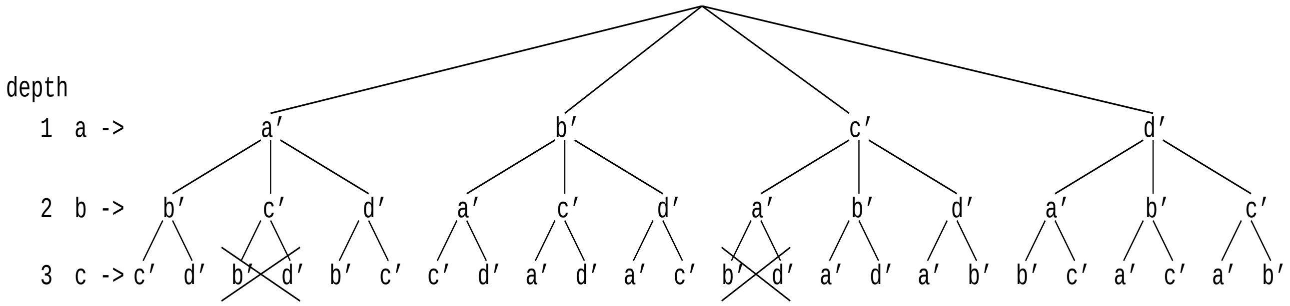


図 12: 探索空間の削減

```
$n := etsa(d);\ i := 0;\ a := eta(n+i);\ b := etb(n+i);$
while $(a \neq 0$ or $b \neq 0)$ {
    if $(B(p(a)\ ,p(b)) = 0)$ return(NG);
    $i := i+1;\ a := eta(n+i);\ b := etb(n+i);$
}
return (OK);
```

図 13: 辺存在確認アルゴリズム

6.2.1 辺導出部

辺導出部の概念図を図 14に示す．破線で囲まれた部分が辺存在確認アルゴリズムである．探索木巡回アルゴリズム (図中では “STV algorithm”) からの入力は $depth$ と $vb(i)(1 \le i \le p_\alpha)$ であり，存在確認部 (図中では “edge existence check module”) からの入力は $exist$ である．また探索木巡回アルゴリズムへの出力は $result$，存在確認部への出力は a と b である．$depth$ は探索木中の深さ $d(1 \le d \le p_\alpha)$ である．$vb(i)$ は G_αの頂点 $v_{\alpha i}(1 \le i \le p_\alpha)$ が写像された G_βの頂点 $v_{\beta j}$の頂点番号，つまり jにあたるものである．

まず，ある深さ d において調べる必要のある辺について述べる．それは $v_{\alpha d}$と $v_{\alpha i}(1 \le i < d)$ で構成される辺，つまり接続行列 $a_{di} = 1$ のものである．これ以外の辺については，既に調べ終っているか，この時点では写像が決定されておらず調べても無駄である．図 2を使って具体的に説明する．$d = 1$ では $1 \le i < 1$ をみたす i が存在しないので該当する辺はなく (1 個の頂点のみでは辺が存在しえないので自明である)，$d = 2$ では $\{v_{\alpha 2}$，$v_{\alpha 1}\}(1 \le i < d = 2$ より) である．同様にして，$d = 3$ では $\{v_{\alpha 3}\ ,v_{\alpha 1}\}$ と $\{v_{\alpha 3}\ ,v_{\alpha 2}\}$ が導かれる．

Edge list table には深さ 1 から p_αまで，上のようにして導かれた “調べる必要のある” 辺を構成する 2 頂点の頂点番号が順番に納められている．ただし深さが変わる時には識別するために (0,0) を挿入する．先程の例では，$d = 1$ の時には該当する辺が存在しないので edge list table には (0,0) のデータしか納めない．$d = 2$ の時には $\{v_{\alpha 1}\ ,v_{\alpha 2}\}$ の頂点番号 (1,2) を納めて，次のアドレスには (0, 0) を入れる．同じように，$d = 3$ では (1,3)，(2,3)，(0,0) を入れればよい．

Edge list table stating address は，深さ d での edge list table の開始アドレスを保持している．図 14では深さ 1 の時には edge list table のデータが 1 番地から開始され，深さ 2 の時には 2 番地から開始されることを示している．

Edge list table から 2 個のデータ $data1$ と $data2$ が出力されると，$vb(1), ..., vb(p_\alpha)$ の中から $vb(data1)$ と $vb(data2)$ を選択し，存在確認部へ出力する．

以上で辺導出部の基本要素の説明は終了したが，実際には control unit が辺導出部全体を管理している．Control unit のアルゴリズムを図15に示す．$addr$と i は control unit の内部変数である．存在確認部からの出力 $exist$ =TRUE は辺が存在したことを表し，$exist$ =FALSE は存在しなかったことを表している．r =OK は，深さ d で調べるべき辺 e_αと辺 e_βの 1 対 1 対応が全て確認できたことを示し，r =NG は確認できなかったことを示す．

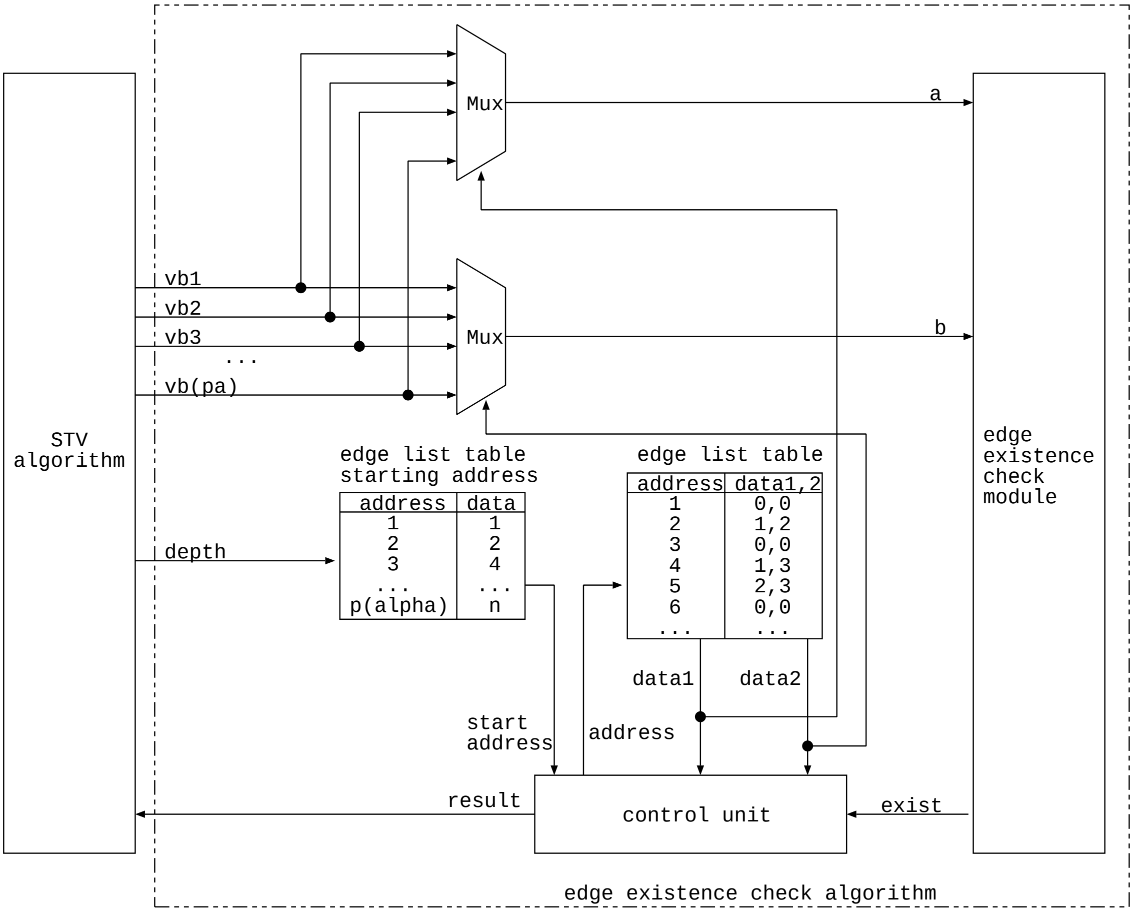


図 14: 辺導出部の概念

```
$i := 0;$
$addr := start\ address;$
while $(data1 \neq 0\ or\ data2 \neq 0)$
{
    if $(exist$ =FALSE)
    {
        $result$ =NG; finish this algorithm;
    }
    $i := i + 1;$
    $addr := start\ address + i;$
}
$result$ =OK; finish this algorithm;
```

図 15: Control Unit のアルゴリズム

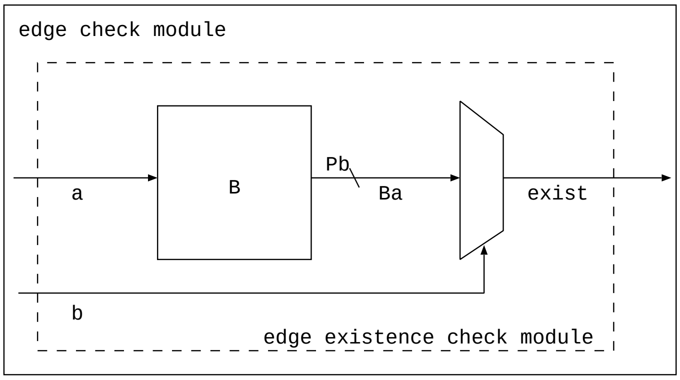


図 16: 存在確認部の概念

```
$M := M^0;\ IM := M^0;\ d := 1;\ used := 0;$
while (1) {
    $current := M_d\ \&\ used;$
    while $(current \neq 0)$ {
        current$(i) = 1$ となる $i$ を設定;
        $p(d) := i;\ current(i) := 0;$
        “辺存在確認アルゴリズム” を実行;
        if (返り値 = OK) {
            if $(d = p_\alpha)$ 写像を記憶;
            else $\{used(i) := 0;\ M_d := current;\ d := d + 1;\}$
        }
    }
    if $(d = 1)$ terminate algorithm;
    $M_d := IM_d;\ d := d - 1;\ used(p(d)) := 1;$
}
```

図 17: 探索木巡回アルゴリズム

6.2.2 存在確認部

存在確認部の概念図を図 16に示す．ここでは辺導出部から入力された $\{v_{\beta a}\ ,v_{\beta b}\}$ という辺が実際に存在するかどうかを調べる．図中の B は G_βの隣接行列である．B は SRAM で構成されており，$p_\beta \times p_\beta$の大きさを持つ．

まず a を B のアドレスとして与えると，配列 B の a 行目の内容 Ba が出力される．Ba に対してマルチプレクサで b 番目の内容を選択すれば $B(a,b)$ が得られる．これが存在確認部の出力となる．

6.3 探索木巡回アルゴリズム

図 17に探索木巡回アルゴリズムを示す．探索空間削減等の判定部分以外は refinement procedure の探索木巡回アルゴリズムと同じものであり，深さ優先探索を行っている．if 節と while 節は C 言語と同じ使用方法である．なお:=は代入を表し，&はビット毎の論理積を表す．ある行列 Xに関して X_iと書いた場合は，i 行目を抽出した列ベクトルを示す．

行列 Mは p_α行 p_β列で構成され，1 または 0 の要素をもつ．$M^0 = [m^0_{ij}]$ は配列 Mの初期値であり，refinement procedure の時と同じように決定される．

$$m^0_{ij} \begin{cases} = 1 & deg(v_{\beta j}) \geq deg(v_{\alpha i}) \\ = 0 & \text{それ以外} \end{cases}$$

d は深さである．p_β ビット幅の $used$ は写像の記憶用変数であり，$used(j) = 1$ ならば $v_{\beta j}$への写像が使用済であ

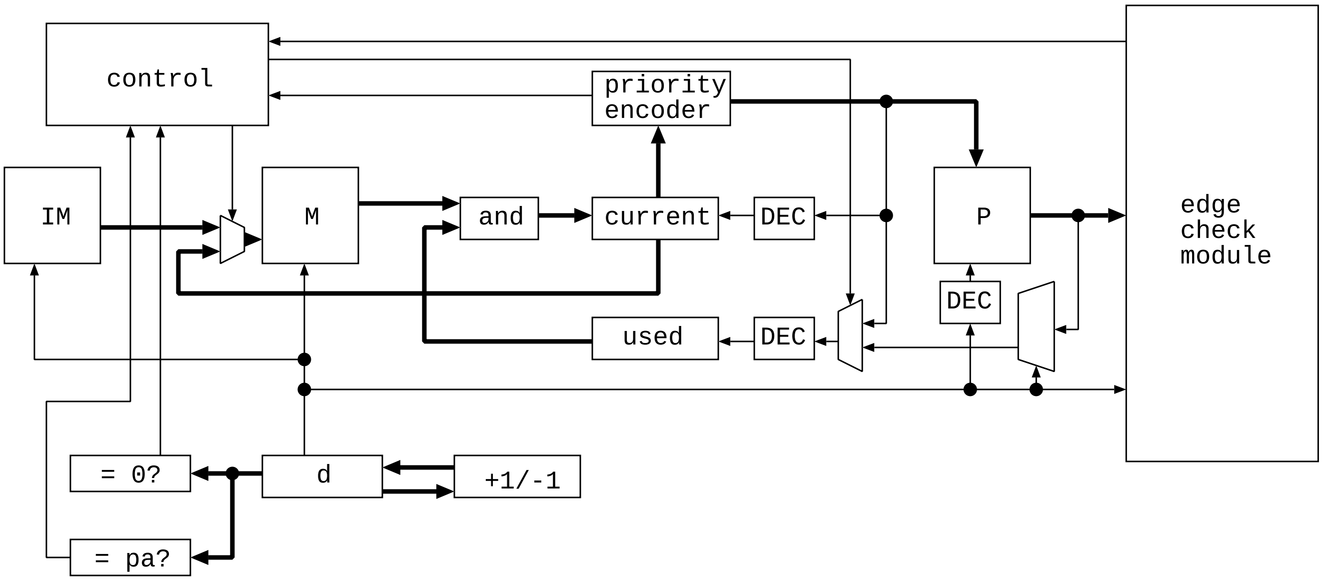


図 18: 探索空間巡回アルゴリズムの概念図

る．p は p_α要素からなり，$p(i)$ には $v_{\alpha i}$の写像先頂点番号が入る．

図 18に探索木巡回アルゴリズムの概念図を示す．この図はあくまで実装する際の概念図であり，このまま実装できるわけではないことに注意してほしい．図中の太い矢印はデータ，細い矢印はアドレスや制御信号を表す．

and は M_dと $used$ のビット論理積をとる論理ゲートである．and ゲートを p_β個並べれば実現可能である．priority encoder は 1 となっているビット位置を 2 進数で出力する回路である．ただし複数ビットが 1 になっている場合は，上位ビットを優先する．この priority encoder には全てが 0 の場合も区別できるようにする．+1/-1 は d を 1 刻みで加減算する回路である．=0?と=pa?は入力データが 0 と p_αであるかどうかを出力する回路である．DEC はデコーダである．デコーダはフリップフロップ $(current, used, P)$ の書き込み制御信号用に使われる．

control は探索木巡回アルゴリズムを管理する部分である．この部分は実際に実装してみないことには図示できないので，現在の時点では今後の課題とする．

6.4 リソース消費量の定式化

前節まで説明してきたアルゴリズム中から，最低限使用される回路を基にリソース消費量を定式化する．考慮した回路は以下の 7 種類である．

- 16 word × 4 bit の SRAM
- 1 bit のフリップフロップ (FF)
- 加算器・減算器・比較器 (ASC)
- マルチプレクサ (MUX)
- 論理積ゲート (AND)
- プライオリティエンコーダ (PRIEN)
- デコーダ (DEC)

それぞれ必要となる回路規模を PFU 使用量で表した．その結果は以下の式のようになる．ただしプライオリティエンコーダについては数式化することができないので，入力ビット幅 (p_βに相当する) を変化させて実際に論

表 2: プライオリティエンコーダの必要 PFU 数

入力ビット幅	必要 PFU 数
16	9
32	21
48	30
64	45
80	57
96	63
112	78
128	94

理合成・テクノロジマッピングを行った．このプライオリティエンコーダは入力されたビットが全て 0 の場合と最下位ビットが 1 の時を区別するようになっている．その結果を表 2に示す．

$$SRAM = 2 \cdot \lceil \frac{\log p_\alpha}{4} \rceil \cdot \lceil \frac{p_\alpha + q_\alpha}{16} \rceil + \lceil \frac{\log(p_\alpha + q_\alpha)}{4} \rceil \cdot \lceil \frac{p_\alpha}{16} \rceil + \lceil \frac{p_\beta}{4} \rceil \cdot \lceil \frac{p_\beta}{16} \rceil + 2 \cdot \lceil \frac{p_\beta}{4} \rceil \cdot \lceil \frac{p_\alpha}{16} \rceil \tag{3}$$

$$FF = \lceil \frac{\lceil \log p_\alpha \rceil + p_\alpha \cdot \lceil \log p_\beta \rceil + \lceil \log p_\beta \rceil + 2 \cdot p_\beta}{4} \rceil \tag{4}$$

$$ASC = 5 \cdot \lceil \frac{\lceil \log p_\alpha \rceil}{4} \rceil + \lceil \frac{\lceil \log(p_\alpha + q_\alpha) \rceil}{4} \rceil \tag{5}$$

$$MUX = \lceil \frac{p_\beta}{8} \rceil + 3 \cdot \lceil \log p_\beta \rceil \cdot \lceil \frac{p_\alpha}{8} \rceil + \lceil \frac{\lceil \log p_\beta \rceil}{4} \rceil + \lceil \frac{p_\beta}{4} \rceil \tag{6}$$

$$AND = \lceil \frac{p_\beta}{4} \rceil \tag{7}$$

$$DEC = 2 \cdot 2 \cdot \lceil \frac{\lceil \log p_\beta \rceil}{3} \rceil + 2 \cdot \lceil \frac{\lceil \log p_\alpha \rceil}{3} \rceil \tag{8}$$

パラメータは (3)～(8) 式と表 2をみればわかるように，p_α，p_β，q_αの 3 つであり，これらの数値を合計したものが与えられたパラメータにおける必要 PFU 数の見積りとなる．以下に (3)～(8) 式の導出方法を説明する．

- SRAM の 1 項目
  辺導出部 (図 14) の edge list table の data 1,2 用である．data 1,2 は組になっているので，以下では data 1 についてのみ記述する．data 1 は値域 $[1, p_\alpha]$ なので $\lceil \log p_\alpha \rceil$ ビット必要となる．SRAM 1 個あたり 4 ビット格納可能なので，data 1 を 1 個格納するには $\lceil \frac{\lceil \log p_\alpha \rceil}{4} \rceil$ 個の SRAM が同時に必要となる．また data 1 は 0 が p_α個 (区切用)，それ以外のデータが q_α個格納される．使用する SRAM は 16 エントリなので，$p_\alpha + q_\alpha$個のデータを格納するには $\lceil \frac{p_\alpha + q_\alpha}{16} \rceil$ 個必要である．結局 data 1 に関しては，$\lceil \frac{\lceil \log p_\alpha \rceil}{4} \rceil \cdot \lceil \frac{p_\alpha + q_\alpha}{16} \rceil$ 個の SRAM が必要となる．data 2 の必要リソースも data 1 と同量なので，SRAM の 1 項目が導出された．

- SRAM の 2 項目
  辺導出部の edge list table starting address に格納される data である．値域は $[1, p_\alpha + q_\alpha]$ なので，data が 1 個あたり $\lceil \log(p_\alpha + q_\alpha) \rceil$ ビット必要になる．SRAM 1 個に 4 ビット格納可能なので，1 個の data に対しては $\lceil \frac{\lceil \log(p_\alpha + q_\alpha) \rceil}{4} \rceil$ 個の SRAM が必要となる．data は p_α個存在するので $\lceil \frac{\log(\lceil p_\alpha + q_\alpha) \rceil}{4} \rceil \cdot \lceil \frac{p_\alpha}{16} \rceil$ だけ SRAM が必要となる．

- SRAM の 3 項目
  存在確認部 (図 16) の隣接行列 Bである．値域 $[0, 1]$ のデータ (1 ビットのみ必要) が $p_\beta \times p_\beta$個必要となるので，$\lceil \frac{p_\beta}{4} \rceil \cdot \lceil \frac{p_\beta}{16} \rceil$ 個の SRAM が必要である．

- SRAM の 4 項目
  探索空間巡回アルゴリズム (図 18) の行列 M, IM である．M と IM は同じ大きさなので M のみを説明する．値域 $[0,1]$ のデータが $p_\alpha \times p_\beta$ 個必要となるので，$\lceil \frac{p_\alpha}{4} \rceil \cdot \lceil \frac{p_\beta}{16} \rceil$ 個の SRAM が必要である．

- FF
  以下 “FF の ~ 項目” と書いた場合は，分子の項を指すものとする．FF の 1 項目は探索空間巡回アルゴリズムの d を格納するものである．値域は $[1, p_\alpha]$ なので $\lceil \log p_\alpha \rceil$ 個のフリップフロップが必要となる．FF の 2 項目は探索空間巡回アルゴリズムの p を格納するものである．p の値域は $[1, p_\beta]$ なので，1 個あたり $\lceil \log p_\alpha \rceil$ 個のフリップフロップが必要となる．p は p_α 個あるので $p_\alpha \cdot \lceil \log p_\alpha \rceil$ 個のフリップフロップが必要である．FF の 3 項目は探索空間巡回アルゴリズムの i(アルゴリズム中の一時変数) を格納するものである．値域は $[1, p_\beta]$ なので $\lceil \log p_\beta \rceil$ 個のフリップフロップが必要となる．FF の 4 項目は探索空間巡回アルゴリズムの $current, used$ を格納するものである．$current$ と $used$ の必要リソース量は同じである．値域は $[0,1]$ で p_β 個必要なので p_β 個のフリップフロップが必要となる．PFU 1 個あたり 4 個のフリップフロップが格納可能である．よって FF で必要な PFU 数は (4) 式のようになる．

- ASC の 1 項目
  探索空間巡回アルゴリズムの+1/-1 部分 (加減算器)，=0?部分 (比較器)，=pa?部分 (比較器) と，Control Unit(図 15) の $data1 \neq 0$(比較器)，$data2 \neq 0$(比較器) である．n ビット加減算器は $\lceil \frac{\lceil \log n \rceil}{4} \rceil$PFU で実装可能である．また比較器は加減算器と同じ論理で実現可能である．これらの加減算器および比較器は，すべて p_α ビット幅であることから 1 項目が導かれる．

- ASC の 2 項目
  Control Unit の $start\ address + i$ を実現する加算器である．値域は $[1, p_\alpha + q_\alpha]$ なので，$\lceil \frac{\lceil \log(p_\alpha + q_\alpha) \rceil}{4} \rceil$ 個の PFU が必要である．

- MUX の 1 項目
  存在確認部の $p_\beta : 1$ マルチプレクサである．OR2CxxA では 8:1 マルチプレクサが PFU 1 個で実現可能である．8:1 マルチプレクサをツリー上に接続すれば，任意のビット幅に対するマルチプレクサが作成できる．n:1 マルチプレクサの 1 段目は $\lceil \frac{n}{8} \rceil$ 個，2 段目は $\lceil \frac{p_\alpha}{2^3} \cdot \frac{1}{8} \rceil$ 個，3 段目は $\lceil \frac{p_\alpha}{2^6} \cdot \frac{1}{8} \rceil$ 個... となる．$2^7 = 128$ 程度の入力ビット幅を想定すると，2 段目はたかだか 2 個になる．そこで $p_\beta : 1$ マルチプレクサを実現するのに $\lceil \frac{p_\beta}{8} \rceil$ 個の PFU が必要だと近似した．

- MUX の 2 項目
  辺導出部のマルチプレクサ 2 個と，探索空間巡回アルゴリズムの $p \rightarrow used$ にあるマルチプレクサである．これらのマルチプレクサは，$p_\alpha : 1$ マルチプレクサを $\lceil \log p_\beta \rceil$ 個並べたものとなっている．MUX の 1 項目で行った近似により，$p_\alpha : 1$ マルチプレクサ 1 個は $\lceil \frac{p_\alpha}{8} \rceil$ 個の PFU が必要となる．あとは単に $\lceil \log p_\beta \rceil$ 倍すればよい．

- MUX の 3 項目
  探索空間巡回アルゴリズムの $used$ に入る直前にあるマルチプレクサである．これは $2:1$ マルチプレクサを $\lceil \log p_\beta \rceil$ 個並べたものとなっている．1 個の PFU に $2:1$ マルチプレクサは 4 個入るので，結果的に 3 項目のようになる．

- MUX の 4 項目
  探索空間巡回アルゴリズムの $IM \rightarrow M$ にあるマルチプレクサである．これは $2:1$ マルチプレクサを p_β 個並べたものとなっている．

- AND
  探索空間巡回アルゴリズムの and 部分である．2 入力の AND ゲートを p_β 個並べればよい．1 個の PFU に 4 個の AND が格納できるので，$\lceil \frac{p_\beta}{4} \rceil$ 個の PFU で実現可能である．

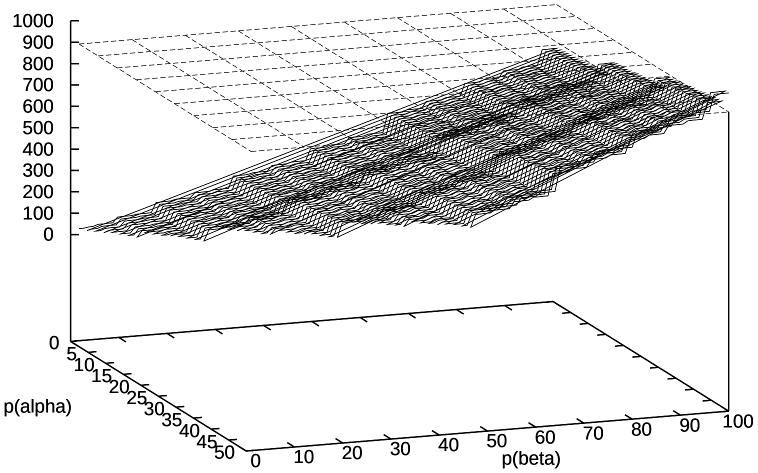


図 19: 提案手法の使用 PFU 数 $(ed_\alpha = 0.10)$

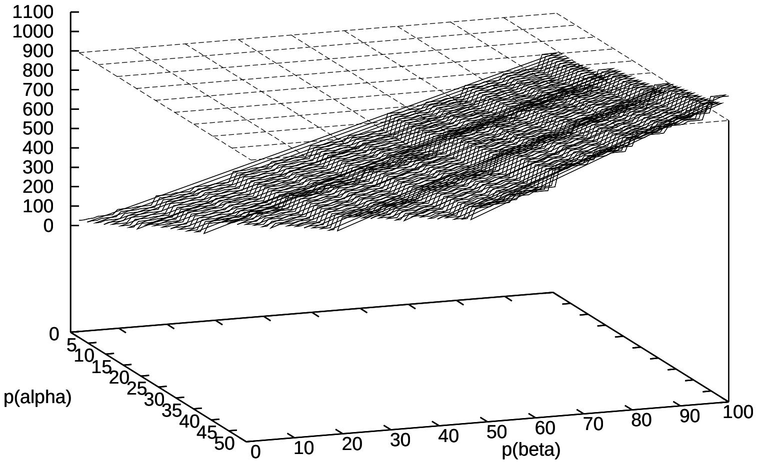


図 20: 提案手法の使用 PFU 数 $(ed_\alpha = 0.20)$

- DEC の 1 項目
  探索空間巡回アルゴリズムの $current$ と $used$ にあるデコーダ部分である．OR2CxxA で $3:8$ デコーダは PFU が 2 個必要となる．$8:1$ マルチプレクサのように考えると，$\log n : n$ デコーダは $\lceil \frac{n}{8} \rceil$ 個の $3:8$ デコーダで実現できるものとして近似した．ここでは $\log p_\beta : p_\beta$ デコーダが必要なので，DEC の 1 項目が導かれる．

- DEC の 2 項目
  探索空間巡回アルゴリズムの p にあるデコーダ部分である．これは $\log p_\alpha : p_\alpha$ デコーダである．DEC の 1 項目と全く同じように求められる．

$1 \leq p_\alpha \leq 50$，$1 \leq p_\beta \leq 100$ まで変化させて見積りを行った．PFU 数は G_α の辺数 q_α によって変化するため，辺数を辺密度 (edge density)ed_α を使って表す $(q_\alpha = ed_\alpha \cdot \frac{p_\alpha(p_\alpha - 1)}{2})$．図 19は ed_α =0.10，図 20は ed_α =0.20，図 21は ed_α =0.50 の時に見積った PFU 数である．図中の p(alpha) は p_α，p(beta) は p_β を表している．ただし $p_\alpha > p_\beta$ の場合は，部分グラフ同型判定問題自体が意味をなさないため記していない．

(3)～(8) 式および表 2をみると，オーダーは $O(p_\alpha^2 \cdot \log p_\alpha)$，$O(p_\beta^2)$ で大きいものとなる．ちなみにプライオリティエンコーダのオーダーは表の値より $p_\beta \log p_\beta$ としている．PFU 数のオーダーが曖昧なのは p_α と p_β は $p_\alpha \leq p_\beta$ という関係しかないため，どちらのパラメータが支配的なのかがわからないためである．図によれば辺密度が 0.50 以下の時は，1 個の OR2C40A FPGA に $p_\alpha + p_\beta \leq 110$ 程度の回路が実装可能である．

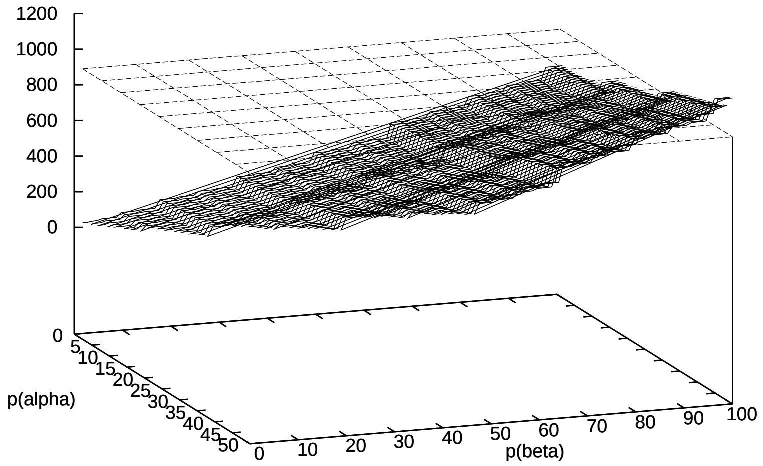


図 21: 提案手法の使用 PFU 数 ($ed_\alpha = 0.50$)

6.5 提案アルゴリズム採用の理由

Refinement procedure で行っている探索空間削減部分 (図 4) をハードウェアに実装しようとすると，以下のような論理が必要となる [1]．ただし $R = [r_{xj}]$ は便宜上定義した $p_\alpha \times p_\beta$ の大きさを持つ 2 進数の配列である．

$$\begin{aligned} r_{xj} &= (\exists y)(m_{xy} \cdot b_{yj}) \\ m_{ij} &= m_{ij} \cdot \overline{(\exists x)(a_{ix} \cdot \overline{r_{xj}})} \end{aligned}$$

$r_{xy} = (\exists y)(m_{xy} \cdot b_{yj})$ を 1 個構成するのに p_β 個の AND ゲートと 1 個の OR ゲートが必要となる．$\overline{r_{xy}}$ は r_{xy} に 1 個の NOT ゲートをつなげば実現できる．$\overline{(\exists x)(a_{ix} \cdot \overline{r_{xj}})}$ は p_α 個の AND ゲートと 1 個の OR ゲート，さらに 1 個の NOT ゲートが必要となる．ここまでで $(p_\beta + 2)(p_\alpha + 2)$ 個のゲートが必要である．さらに $\overline{(\exists x)(a_{ix} \cdot \overline{r_{xj}})}$ は m_{ij} との論理積をとる必要がある．m_{ij} は p_α 行 p_β 列で構成される配列 M の要素なので，$\overline{(\exists x)(a_{ix} \cdot \overline{r_{xj}})}$ と m_{ij} との論理積をとる AND ゲートが $p_\alpha \cdot p_\beta$ 個必要となる．以上のことから refinement procedure の探索空間削減部分をハードウェアで実装するには $p_\alpha p_\beta (p_\beta + 2)(p_\alpha + 2)$ 個のゲートが必要である．つまり $O({p_\alpha}^2 {p_\beta}^2)$ となる．

OR2CxxA では 1 個の PFU に AND, OR, NOT ゲートを合計 4 個まで格納できる．このことから OR2C40A では $900 \times 4 = 3,600$ ゲート使用することが可能である．しかし $p_\alpha = 10$,$p_\beta = 10$ 程度 (この場合 19,600 ゲート) でも OR2CxxA シリーズ FPGA には収まらない．対して提案アルゴリズムでは探索空間削減法を refinement procedure より単純にして，ハードウェアの使用量を抑えている．その結果 6.4節で示したような大きさのグラフに対しても適用可能となった．

7 両方式の性能比較

まず提案アルゴリズムを実装するハードウェアと，refinement procedure を実装した計算機の環境について説明する．次に性能比較の方法について説明し，結果を検討する．

7.1 実行環境

本論文で提案した部分グラフ同型判定アルゴリズムは，5章で説明した OPERL ボード [5] に実装することを仮定している．

Refinement procedure は FreeBSD-2.2.6-RELEASE 上に C 言語で実装した．性能測定を行った環境を図 22 に示す．この計算機は 1998 年後半から 1999 年前半にかけて一般的な仕様である．

7.2 性能測定方法

本論文でいう性能とは実行時間のことであり，実行時間が小さいほど高性能である．まず提案アルゴリズム，refinement procedure 両方法に共通した性能測定の方法について説明する．続いて提案アルゴリズムでの実行時間

プロセッサ	Intel Pentium-II 333MHz
マザーボード	Intel Pentium-II LX System Board
２次キャッシュ	512KB(Pentium-II に内蔵)
主記憶	SDRAM 128MB(60ns)
基本ソフトウェア	FreeBSD-2.2.6-RELEASE

図 22: Refinement procedure を実装した計算機仕様

```
$M := M^0$; $IM := M^0$; $d := 1$; $used := 0$;[ここまでは初期設定]
while (1) {
    $current := M_d$ & $used$;[1]
    while ($current \neq 0$) {
        current($i$) = 1 となる $i$ を設定;[前行と重畳で priority encoder]
        $p(d) := i$; $current(i) := 0$;[両方で 1]
        “辺存在確認アルゴリズム” を実行;
        if (返り値 = OK[辺存在アルゴリズムと重畳]) {
            if ($d = p_\alpha$[1]) 写像を記憶;[1]
            else {$used(i) := 0$; $M_d := current$; $d := d + 1$;[全部で 1]}
        }
    }
    if ($d = 1$[1]) terminate algorithm;
    $M_d := IM_d$; $d := d - 1$; $used(p(d)) := 1$;[全部で 1]
}
```

図 23: 探索木巡回アルゴリズムのクロック数設定

測定方法，refinement procedure での実行時間測定方法を順番に説明する．

7.2.1　性能測定の基本

以下ではed_αと$ed_\beta (0 \leq ed_\alpha, ed_\beta \leq 1)$をそれぞれ$G_\alpha$，$G_\beta$の辺密度であるとする．$ed_\alpha = 0.10, 0.20, 0.30, 0.40, 0.50$，$ed_\beta = 0.05, 0.10, 0.15, 0.20, 0.25$ のように変化させ，ed_αと ed_βの全てつまり 25 通りの組合せについて基本測定を行った．基本測定ではある ed_α, ed_βの組合せに対して，p_α，p_βを $1 \leq p_\alpha \leq 10$，$1 \leq p_\beta \leq 15$ の範囲で１ずつ変化させ，それぞれについて 50 回ずつ G_αと G_βをランダムに与えた時の合計実行時間を測定している．ただし $p_\alpha > p_\beta$ の場合は，部分グラフ同型問題自体が意味をなさないため測定を行っていない．

7.2.2　提案アルゴリズムの実行時間測定方法

実際に OR2CxxA 上に実装したわけではないので，提案アルゴリズムの実行時間測定はあくまでシミュレーションである．最終的な実行時間は OPERL ボード上にアルゴリズムを実装し，実機で測定しなければならない．実行時間のシミュレーションは，提案アルゴリズムを C 言語のプログラムで実装し，最低限必要となるであろうクロック数を合計することによって行った．クロック数が求まれば周波数の逆数 ($\frac{1}{33 \times 10^6}$) をかけてやることにより結果が求まる．実際には 1 クロックですむと考えている場所が 2・3 クロック必要であったり，制御部分の論理が複雑でさらにクロック数が増えるということも考えられる．さらに配線遅延が予想より増加するということも多いに考えられる．

提案アルゴリズムのクロック数計算は図 23と図 24の設定で行った．設定したクロック数については図中の [ ] 内に記してある．なお priority encoder については，個別に必要クロック数を設定した．Priority encoder のクロック数については表 3に示す．

```
$n := etsa(d);\ i := 0;$ [以上2命令で1] $a := eta(n+i);\ b := etb(n+i);$ [以上2命令で1]
while $(a \neq 0$ or $b \neq 0)$ {
    if $(B(p(a)\ ,p(b)) = 0[1])$ return(NG);
    $i := i+1;$ [1] $a := eta(n+i);\ b := etb(n+i);$ [1]
}
return (OK);
```

図 24: 辺存在確認アルゴリズムのクロック数設定

表 3: プライオリティエンコーダの設定クロック数

入力ビット幅	設定クロック数
16	2
32	3
48	3
64	5
80	4
96	6
112	8
128	9

また提案アルゴリズムでは性能を向上させるために，以下のことを仮定している．6章でリソース使用量の見積りを行ったが，もしもp_α, p_βが小さくてPFU数に余裕がある場合は，残りの部分は完全に遊んでいることになってしまう．そこでリソースに余裕がある場合は，同じp_α, p_βの提案アルゴリズムを複数個並列に動作させ，実行時間を短縮することを考えた．複数個並列に実行する提案アルゴリズムのことをモジュールと呼ぶことにし，$module_i$と記すことにする．6章での見積り結果をモジュール1個あたりの必要リソース量として，1個のFPGAにモジュールがいくつ実装できるかを計算し，これを並列に実行可能なモジュール数とした．

複数のモジュールを並列に実行するためには，$N(\geq 1)$個の$module_i(1 \leq i \leq N)$に探索空間を割り当てる必要がある．これ以降は具体的な例をあげて説明する．探索木は図3とし($p_\alpha = 3, p_\beta = 4$)，モジュール数$N$を2とする．探索空間を単純に等分するならば，図25のように分割すればよい．提案アルゴリズムで探索空間分割を反映させるためには，Mの初期値を変更する．もしも分割前のMが(9)式ならば，分割後の$M_{module(1)}, M_{module(2)}$は(10)式になる．これにより$module_1$はa → a',b' しか調べず，$module_2$はa → c',d' しか調べないので，所望の動作が行えることになる．

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{9}$$

$$M_{module(1)} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \quad M_{module(2)} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{10}$$

Mが(9)式のような場合はいいが，例えば(11)式のようになると問題が生じる．この場合の探索空間を図示してみると，図26のようになる．a → b' という写像はありえないために，その部分の探索空間が消えている．図をみればわかるように$module_1$の探索空間は$module_2$の半分である．しかし実行時間は最も処理に時間がかかったモジュールの実行時間によって決まるために，図25の場合と変わらなくなってしまう．

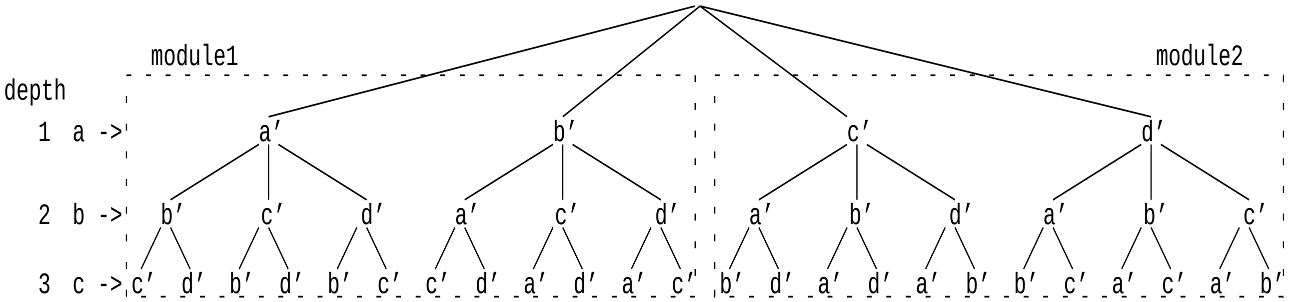


図 25: 単純な探索空間分割

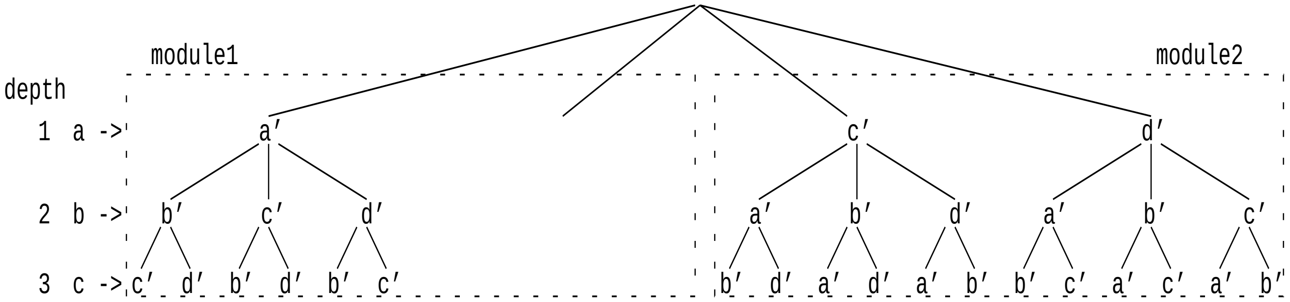


図 26: 問題の生じる探索空間分割

$$M = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{11}$$

これを避けるために，本論文ではMの 1 行目にある 1 の数を数え，それを等分した節を各モジュールへ割り当てる方式をとっている．この方法では図 26の場合，問題が残ったままとなってしまうが，もしも更に適切に割り当てたい場合には深さ 2 以上の節も加味すればよい．これについては時間の都合上行っていない．

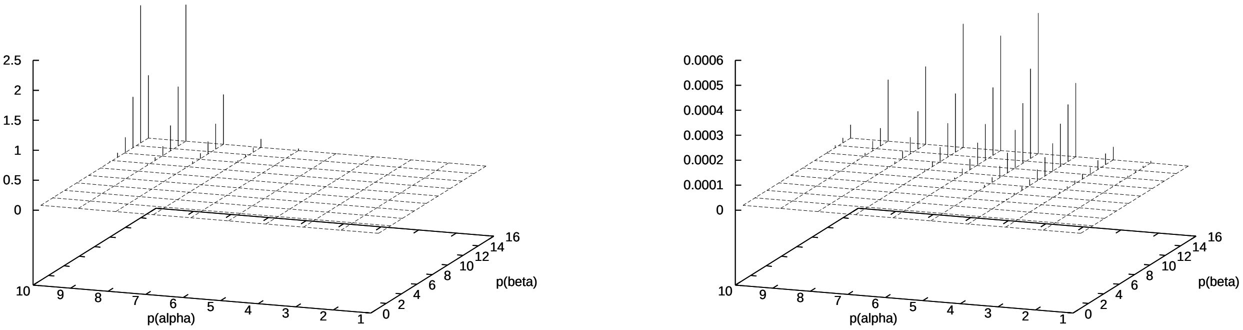
7.2.3 Refinement procedure の実行時間測定方法

Refinement procedure の実行時間測定については，C 言語のプログラムの該当箇所に **getrusage()** 関数を挿入して行った．この関数では 1 マイクロ (10^{-6}) 秒単位まで測定可能である．測定する範囲は図 6において，step 1 の refine Mから step 7 までである．

7.3 性能比較の結果

$ed_\alpha = 0.10, 0.50$，$ed_\beta = 0.25$ の実行時間を示す．図 27は提案アルゴリズムの実行時間，図 28は refinement procedure の実行時間である．両方法ともにp_αを固定してみるとp_βの 2 乗に比例している．全体の傾向は変わらないが，提案アルゴリズムの実行時間が refinement procedure と比較して，$ed_\alpha = 0.10$ の時で 2 倍程度，$ed_\alpha = 0.50$ の時で 40 倍程度小さい値となっている．

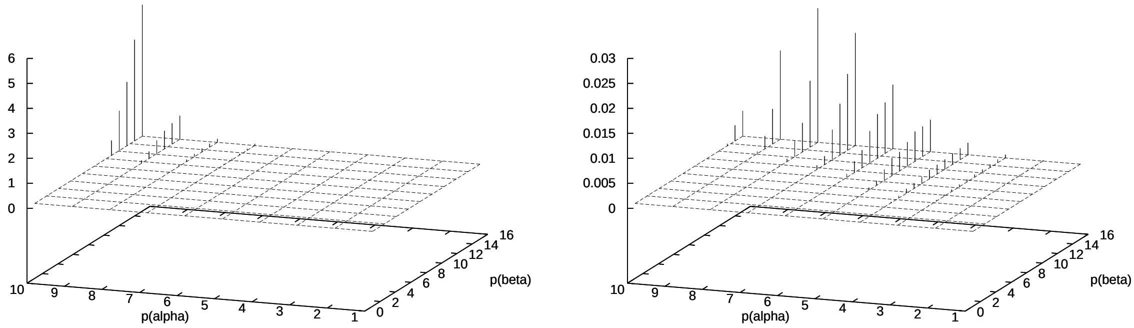
図 29は提案アルゴリズムと refinement procedure の実行時間比を示したものである．提案アルゴリズムの実行時間が refinement procedure より小さければ正の値で，逆ならば負の値で示してある．図をみると提案アルゴリズムでは明らかに $ed_\alpha = 0.10$ の時が不得意であることがわかる．なぜこのような現象が起こるのかについては，現在調査中である．一方 $ed_\alpha = 0.50$ の時は全体的に提案アルゴリズムが高性能となっている．結果は示していないが $ed_\alpha = 0.10$ 以外は，$ed_\alpha = 0.50$ と同様に全ての p_α, p_βの組合せに対して，提案アルゴリズムの方が高性能であった．



(a) $ed_\alpha = 0.10$

(b) $ed_\alpha = 0.50$

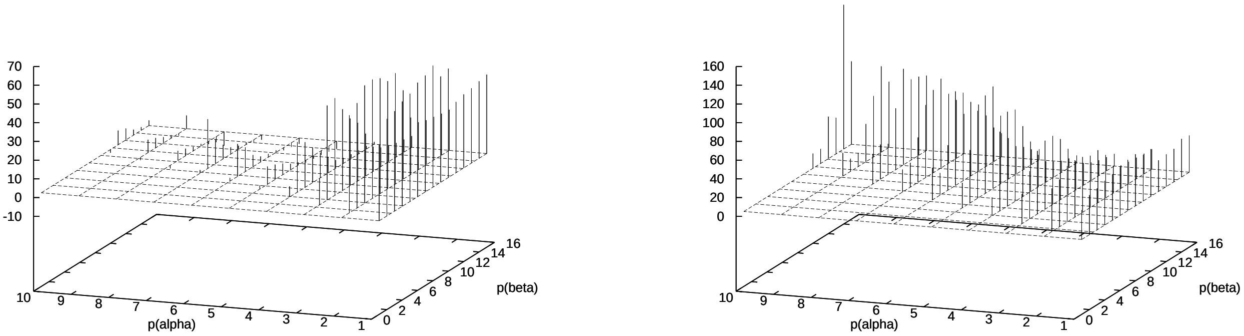
図 27: 提案アルゴリズムの実行時間 ($ed_\beta = 0.25$)



(a) $ed_\alpha = 0.10$

(b) $ed_\alpha = 0.50$

図 28: Refinement procedure の実行時間 ($ed_\beta = 0.25$)



(a) $ed_\alpha = 0.10$

(b) $ed_\alpha = 0.50$

図 29: 提案アルゴリズムと Refinement procedure の実行時間比 ($ed_\beta = 0.25$)

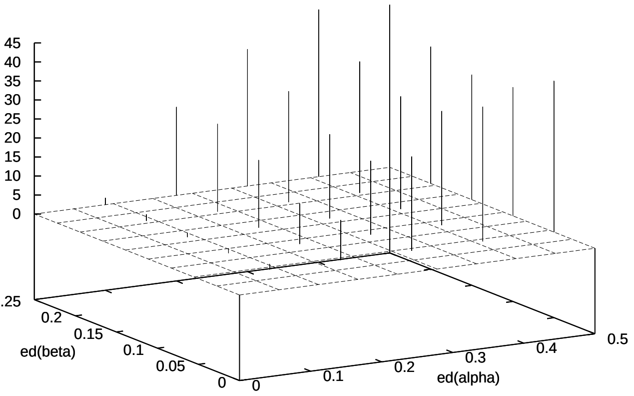


図 30: 両方式の平均性能比

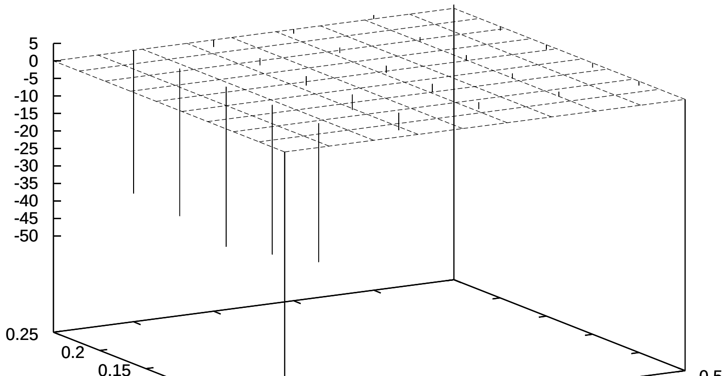


図 31: 提案アルゴリズム (ソフトウェア) と refinement procedure の性能比

図30に両方式の平均性能比を示す．平均性能比は以下のようにして算出した．各方法で$1 \le p_\alpha \le 10$，$1 \le p_\beta \le 15$の範囲で変化させた実行時間の合計を取る．両方法の合計実行時間の比を，ed_α, ed_βを軸に取って表示したものである．図をみると，先ほど指摘した$ed_\alpha = 0.10$の性能が悪くなっていることが顕著に現れている．

7.4 提案アルゴリズム(ソフトウェア)とrefinement procedure(ソフトウェア)の性能比較

図31は提案アルゴリズムをソフトウェアで実装したものと，refinement procedureをソフトウェアで実装したものの性能比である．マイナスの値は提案アルゴリズムの実行時間がrefinement procedureの実行時間より大きいことを示す．

$ed_\alpha = 0.10$のときは，提案アルゴリズムに比べて約40倍も実行時間が大きくなる．$ed_\alpha = 0.20$でも2ないし5倍程度遅い．$ed_\alpha > 0.20$では1.2ないし1.5倍程度の性能低下にとどまるが，それでも性能が悪いことにはかわりがない．グラフでは2ヶ所だけrefinement procedureより高性能な場合があるが，1.0x倍という微々たる差でしかない．グラフの傾向はハードウェア実装のシミュレーション時と同じである．

OR2C FPGA実装時にはrefinement procedureより高性能となり，ソフトウェア実装時にrefinement procedureより性能が悪くなるということは，専用アーキテクチャを利用可能なメリットが存在することを意味している．

8 動的再構成を利用した拡張

本研究で実装対象としたOR2C FPGAには動的再構成という機能が存在する．動的再構成はFPGA動作中(電源オン時)の構成データ書換えである．動的再構成を利用して単体のOR2C FPGAで扱えるG_βより大きなグラフを処理できるように，提案アルゴリズムの拡張を行う．

簡単に拡張方法を説明すると，部分グラフ同型であるグラフが全て検出できるようにG_βを分割するということである．G_βが連結グラフでない場合は，G_βを構成する連結成分ごとに拡張手法を適用すればよい．

8.1 拡張方法適用の準備

以下では具体的なG_α, G_βを用いて概念の説明を行う．

G_αは図32，G_βは図33である時を考える．また$p_{\alpha c} = 4, p_{\beta c} = 8$までのグラフしかハードウェアには格納できないものとする．部分グラフ同型を判定したいG_α, G_βでは$p_\alpha = p_{\alpha c}$, $p_\beta > p_{\beta c}$であるので，このままでは実装不可能である．そこでG_βを適当に分割することによって，限られたハードウェア資源でより大きなグラフに対しての処理を可能とする．なお，以下で提案する手法では$p_\alpha > p_{\alpha c}$の場合は扱えない．

まずグラフの直径(diameter)を定義する．G_αの直径とかいた場合には，G_α中の任意の頂点$v_{\alpha i}(1 \le i \le p_\alpha)$から他の頂点$v_{\alpha x}(1 \le x \le p_\alpha$かつ$x \ne i)$へ到達する最短経路(minimum path)中で，最大の長さ(つまり辺数)を指すものとする．図32では，1 → 2の最短経路は辺12，1 → 3は辺13，1 → 4は辺13と辺34，2 → 3は辺23，2 → 4は辺23と辺34，3 → 4は辺34である．つまり任意の頂点間は2個の辺で到達可能なので，G_αの直径は2ということになる．

拡張方法を適用する準備として，G_βの全頂点$v_{\beta i}(1 \le i \le p_\beta)$に対し$v_{\beta i}$から直径$r$($r$は$G_\alpha$の直径を表す．例では$r = 2$)以内にある頂点の集合を求める．それぞれのグラフを$G_{\beta i}(1 \le i \le p_\beta)$とし図34に示す．黒い丸は基準となる頂点，斜線の丸は基準点から直径1の点，網掛された丸は基準点から直径2の点を表している．これらのグラフを“サブグラフ”と呼ぶことにする．図中に$p_{\beta i}$を示してあるが，$p_{\beta i} > p_{\beta c}$となるサブグラフ$(G_{\beta 6}, G_{\beta 7}, G_{\beta 9}, G_{\beta 10}, G_{\beta 11})$はハードウェアに収まらず，$p_{\beta i} < p_{\alpha c}$となるサブグラフ$(G_{\beta 1})$は部分グラフ同型となり得ないのでこの時点で除去する．もしも全てのサブグラフについて$p_{\beta i} > p_{\beta c}$となった場合は，この手法は適用不可能である．

8.2 拡張方法の適用例

拡張方法適用の準備が終った時点で残っているサブグラフから，構成される頂点数$p_{\beta i}$が最大なものを選ぶ．例では$G_{\beta 3}$と$G_{\beta 14}$が$p_{\beta 3} = p_{\beta 14} = 8$で最大となるが，ここでは$G_{\beta 3}$を選択する．ここまでで出来上がったグラフ

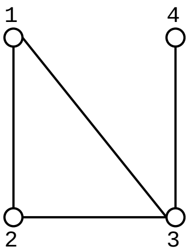


図 32: G_α

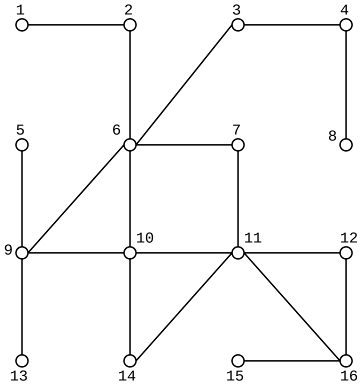


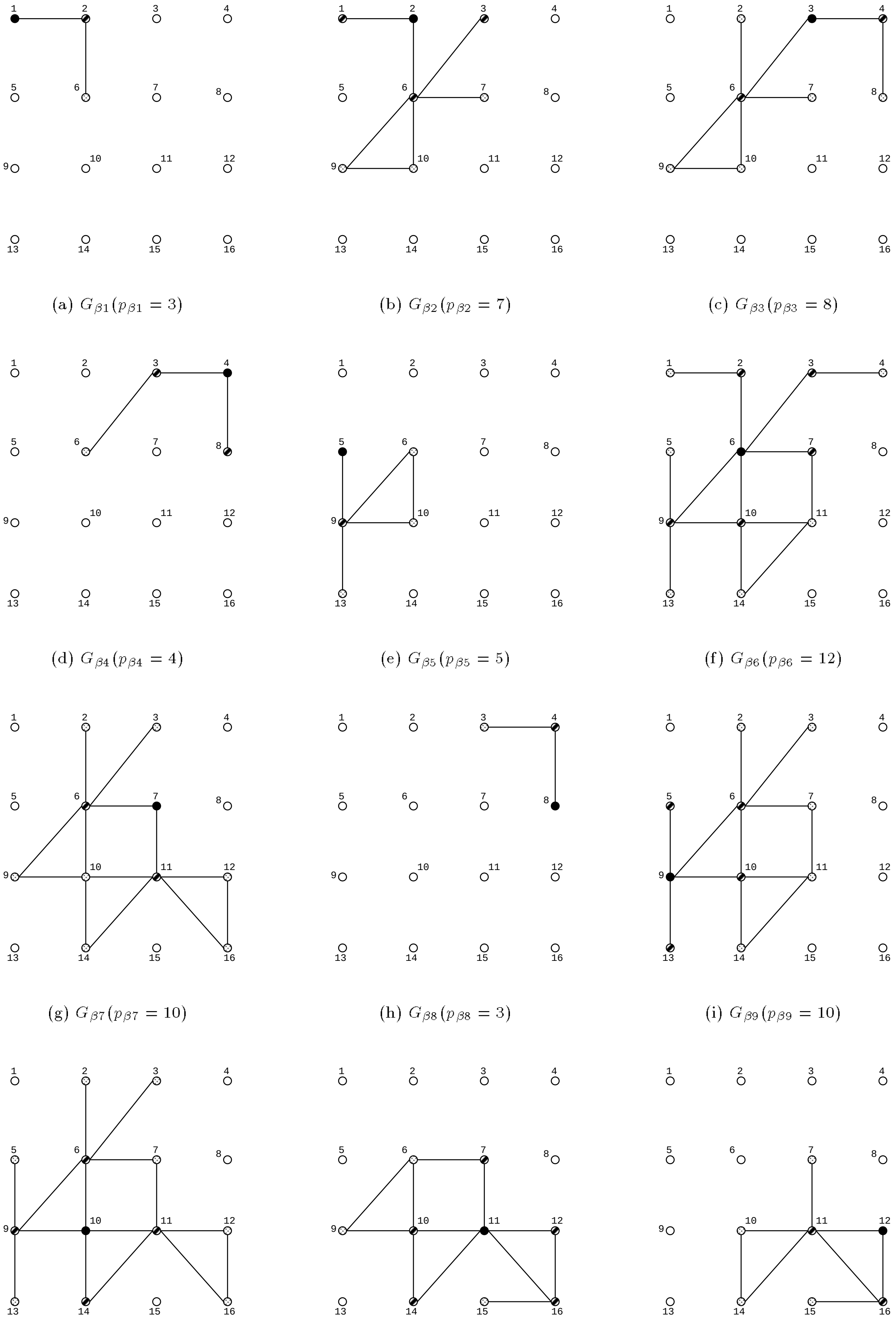
図 33: G_β

G_{tmp1}を図 35に示す．破線は G_βと比較して G_{tmpi}では存在しない辺を表す．黒丸は破線で表された辺と接続している頂点を表している．この頂点を他のサブグラフと接続する点という意味をこめて “接続頂点” と呼ぶことにする．接続頂点については，他のサブグラフと組み合わせないと全ての部分グラフ同型が検出できないことになる．“他のサブグラフと組み合わせる” と書いたが，ただ単に接続頂点を含んでいるサブグラフと組み合わせると問題が生じる．例えば G_βの頂点 $\{7,10,11,12,14,16\}$ を含むサブグラフ $G_{\beta16}$と組み合わせた時を考えよう．この時の様子を図 36に示す．長い線の破線は他方のサブグラフに含まれる辺を表す．確かに接続頂点 7,10 はカバーされるが，$G_{\beta3}$と $G_{\beta16}$について別々に部分グラフ同型判定を行うと，頂点 7,10 をまたぐ部分グラフについては検出されない．例えば頂点 $\{6,9,10,11\}$ については部分グラフ同型となるのだが，別々のサブグラフに含まれているために検出できないのである．

この問題を回避するには以下のようにすればよい．G_αの直径は $r = 2$ であることから，黒丸の頂点から直径 $r - 1 = 1$ の頂点を含むようなサブグラフと組み合わせれば全ての部分グラフ同型な部分グラフが検出できる (条件 **8-1**)．頂点 7 では頂点 6，頂点 10 では頂点 6 と 9 がこのような頂点である．頂点 $\{6,9,10,11\}$ を部分グラフ同型なものとして検出するためには，(例えば)$G_{\beta14}$を組み合わせればよい (図 37)．この場合は頂点 6,9 両方ともに $G_{\beta4}$と $G_{\beta14}$に含まれているので，接続頂点 7,10 を含む部分グラフについては部分グラフ同型となる部分グラフが検出できることになる．

ここまで述べてきたことをまとめると，条件 8-1 を満たすようにサブグラフを組み合わせて G_βと同一なグラフが構成できればよいことになる．もちろん組み合わせるサブグラフ $G_{\beta i}$の順番は構わないが，プログラムとして実装する際には意識する必要がある．順番に関しては現在のところ確立されていないので，修士論文に書く段階ではきちんと詰める．

話は前後するが，G_{tmp1}で接続頂点 2 を解決することから始めよう．頂点 2 については頂点 6 も含んだサブグラフを組み合わせる必要がある．頂点 2 に隣接している頂点で G_βに含まれていない頂点は 1 であるので，結局頂点 1,2,6 を含んだサブグラフを組み合わせればよいことになる．該当するサブグラフは $G_{\beta2}$しか存在しない ($G_{\beta1}$は $p_{\beta1} = 3 < p_{\alpha c}$より除外されている)．$G_{\beta2}$と $G_{\beta3}$を組み合わせたグラフ G_{tmp2}は図 38のようになる．接続頂点 2 は



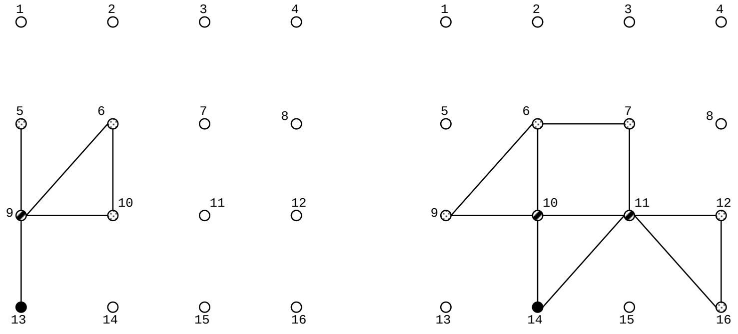
(a) $G_{\beta 1}(p_{\beta 1} = 3)$ (b) $G_{\beta 2}(p_{\beta 2} = 7)$ (c) $G_{\beta 3}(p_{\beta 3} = 8)$

(d) $G_{\beta 4}(p_{\beta 4} = 4)$ (e) $G_{\beta 5}(p_{\beta 5} = 5)$ (f) $G_{\beta 6}(p_{\beta 6} = 12)$

(g) $G_{\beta 7}(p_{\beta 7} = 10)$ (h) $G_{\beta 8}(p_{\beta 8} = 3)$ (i) $G_{\beta 9}(p_{\beta 9} = 10)$

(j) $G_{\beta 10}(p_{\beta 10} = 12)$ (k) $G_{\beta 11}(p_{\beta 11} = 9)$ (l) $G_{\beta 12}(p_{\beta 12} = 7)$

図 34: $G_{\beta 1} \sim G_{\beta 12}$


(a) $G_{\beta 13}(p_{\beta 13} = 5)$ (b) $G_{\beta 14}(p_{\beta 14} = 8)$

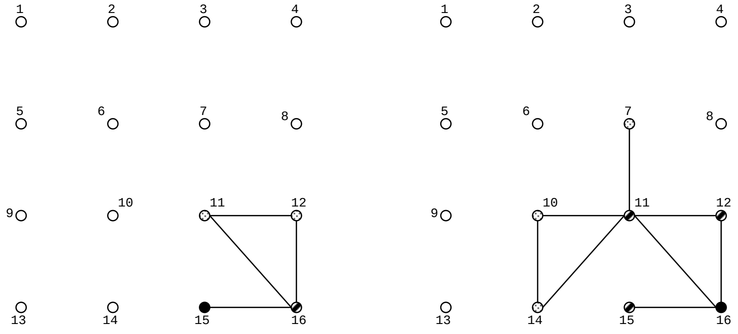

(c) $G_{\beta 15}(p_{\beta 15} = 4)$ (d) $G_{\beta 16}(p_{\beta 16} = 7)$

図 34: $G_{\beta 13} \sim G_{\beta 16}$

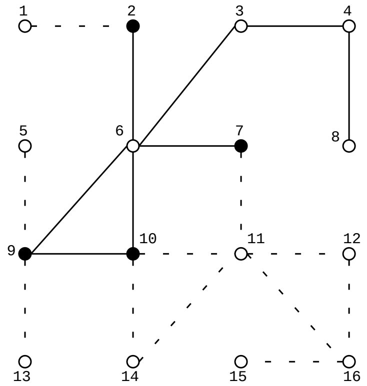


図 35: G_{tmp1}

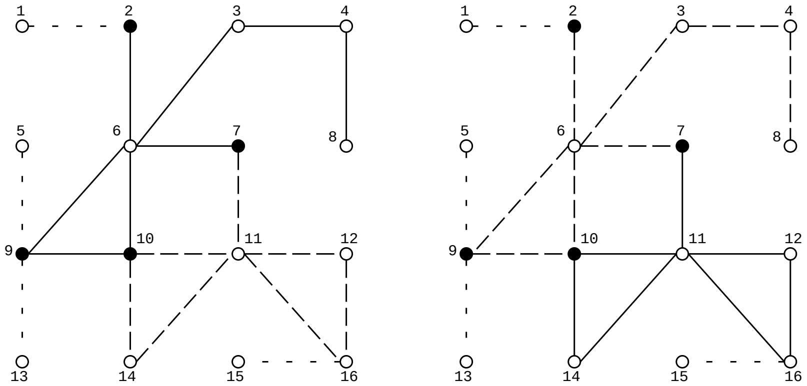


図 36: $G_{\beta 3}$(左) と $G_{\beta 16}$を組み合わせたグラフ

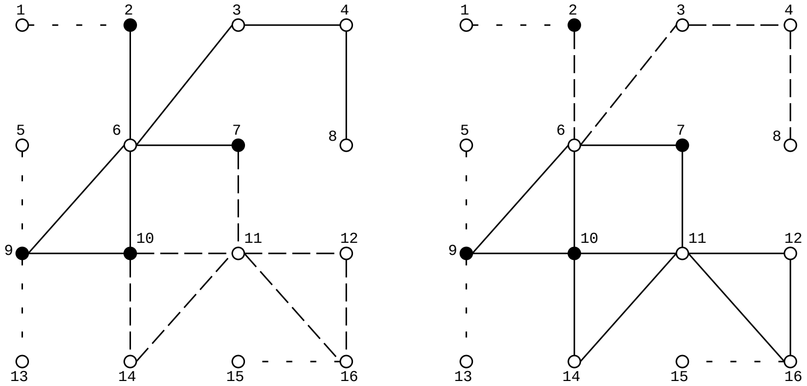


図 37: $G_{\beta 3}$(左) と $G_{\beta 14}$を組み合わせたグラフ

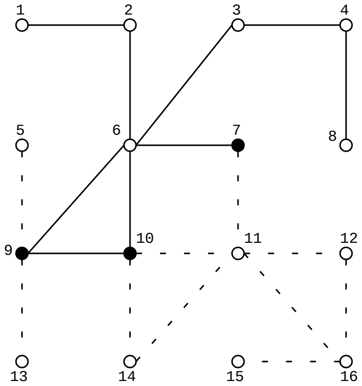


図 38: G_{tmp2}

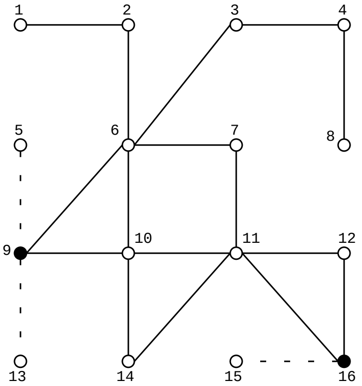


図 39: G_{tmp3}

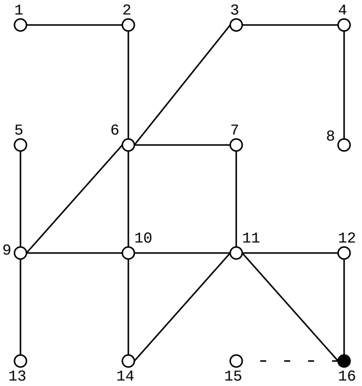


図 40: G_{tmp4}

解決され，頂点 1 も G_βとの差がないために接続頂点とはならない．結局 7,9,10 が接続頂点として残されている．

次に接続頂点 7 を解決しよう．他のサブグラフと組み合わせられる条件は頂点 6 も含まれていることである．頂点 6,7 を含んでいて，未解決の頂点をもっとも多く含んでいるサブグラフは $G_{\beta14}$である．G_{tmp2}と $G_{\beta14}$を組み合わせたグラフ G_{tmp3}は図 39のようになる．$G_{\beta14}$は G_{tmp2}で接続頂点であった頂点 10 に対しても条件 8-1 を満たしているため，頂点 7,10 が解決できた．接続頂点として頂点 9,16 が残っている．

続いて接続頂点 9 を解決する．条件 8-1 を満たすために頂点 6 と 10 を含む必要がある．条件に合致するサブグラフは $G_{\beta5}$と $G_{\beta13}$が存在するが，ここでは $G_{\beta5}$を採用する．結果 G_{tmp4}は図 40のようになる．残る接続頂点は 16 だけとなった．

最後に接続頂点 16 を解決する．条件 8-1 を満たすために頂点 11 と 12 を含む必要がある．条件に合致するサブグラフは $G_{\beta12}$，$G_{\beta15}$と $G_{\beta16}$が存在するが，もっとも頂点数の少ない $G_{\beta15}$を採用する．なぜならば残る頂点は 15 だけなので，頂点 15 を含むサブグラフの中で最小のものを選んだ方が，計算量が少なくてすむからである．G_{tmp4}と $G_{\beta15}$を組み合わせると G_βと同一なグラフとなる．

以上のことから，$G_{\beta2}$，$G_{\beta3}$，$G_{\beta5}$，$G_{\beta14}$と $G_{\beta15}$それぞれに対して部分グラフ同型問題を適用したものと，G_βに部分グラフ同型問題を適用したものの結果は同一となる．ただし分割した場合には，複数のサブグラフで同じ部分グラフが検出される場合がある．

9 おわりに

本論文では OR2C FPGA に適した部分グラフ同型判定アルゴリズムを提案した．OR2C40A FPGA には $p_\alpha + p_\beta \leq 110$ という実用的な大きさのグラフが実装可能であり，refinement procedure と比較して最大 40 倍程度の性能改

善が可能であるという見積り結果も得られた．また同 FPGA が持つ動的再構成という特徴を利用して，より大きなグラフが扱えるような拡張方法についても提案を行った．

しかし得られた結果はあくまでも“見積り”の結果である．したがって実際に OPERL ボード上の OR2C FPGA へ実装を行い，評価を行う必要がある．

参考文献

[1] J. R. Ullmann: An Algorithm for Subgraph Isomorphism, Journal of the Association for Computing Machinery, Vol. 23, No. 1, pp. 31-42, 1976.

[2] Michael R. Garey, David S. Johnson: Computers and intractability - a guide to the theory of NP-completeness, W. H. Freeman, San Francisco, 1979.

[3] Duncan A. Buell et al.: Splash 2: FPGAs in a Custom Computing Machine, IEEE Computer Society Press, Los Alamitos, 1996.

[4] ルーセントテクノロジ半導体販売株式会社: ORCA^{TM} OR2CxxA と OR2TxxA シリーズ・フィールド・プログラマブル・ゲートアレイ・データ・シート 1997年10月, ルーセントテクノロジ半導体販売株式会社, 東京都, 1997.

[5] 市川周一，島田俊夫: PCI バスに付加する再構成可能ボードの試作評価，信学技報 CPSY96-97，pp. 159-166，1996.

[6] Lucent Technologies Inc.: ORCA^{TM} Foundry Macro Reference Library Manual Version 4.2, Lucent Technologies Inc., U.S.A., 1996.

謝辞

最後まで御指導頂いた市川先生に，この場を借りて御礼申し上げます．
また 2 年という短い時間でしたが，私とともに楽しい時や苦しい時を共に過ごした赤羽君他研究室の皆さんにも感謝いたします．最後に 6 年という長期にわたり大学に通わせて頂いた両親に感謝いたします．