部分グラフ同型判定アルゴリズムの FPGAを用いた実装

2000年2月23日

知識情報工学専攻 967751
ラターナセンタン・ウドーン
Udorn Lerdtanaseangtham

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，化学物質の中に特定の構造が含まれるかどうか判定する問題（構造活性の推測）や，計算機によるシーン解析すなわちパターン認識など，様々の分野に応用可能である [1]．しかし部分グラフ同型判定問題は NP 完全であるため，効率的に解くアルゴリズムは存在しない [2]．

部分グラフ同型判定は，2 つのグラフの頂点間に存在する対応（写像）全てについて同型という条件を満たすかどうか調べる一種の探索問題である．この探索を総当りで行うと効率が悪いので，無駄な探索を省いて探索空間を削減する方法が重要となる．一般に最も標準的に用いられるアルゴリズムは，Ullmann[1] による refinement procedure である (以後，本論文では Ullmann のアルゴリズムと呼ぶ)．

小西 [6] は，部分グラフ同型判定問題をプログラム可能な論理デバイスの一種である FPGA (Field Programmable Gate Array) に実装し，実行を高速化する手法について検討した．Ullmann のアルゴリズムは効率が良いかわり複雑なので，ハードウェアに実装することはゲート数の点から困難である．そこで小西は，Ullmann のアルゴリズムより探索空間削減の効率は劣るが，より単純でハードウェアによる実装に適したアルゴリズムを提案し，その実装規模と性能について検討した．以後，本論文では小西 [6] の提案したアルゴリズムを小西のアルゴリズムと呼ぶ．

小西 [6] の報告によれば，小西のアルゴリズムを FPGA 上に実現して 33 MHz で動作させた場合，Ullmann のアルゴリズム (ソフトウェア) を 333 MHz の PentiumII で実行する場合に比べて 10〜50 倍性能が高くなるという．ただし小西はハードウェアの実装を行っていないので，実際に回路が 33 MHz で動くかどうかは明らかでない．また同時に，同型判定の対象となるグラフの辺密度が小さいときは小西のアルゴリズムの優位性が失われることも報告されたが，その原因は明らかでなかった [6]．

そこで本研究では，(1) 入力としたグラフの辺密度が小さい時に小西のアルゴリズムの優位性が失われた原因を解明し，(2) 小西のアルゴリズムを実際に OPERL ボード [4] 上へ実装して性能を評価することを目的とする．

以下，2 章では部分グラフ同型判定問題について説明する．3 章では小西のアルゴリズムの性能評価について述べる．そして 4 章では小西のアルゴリズムの FPGA による実装について説明する．最後に，5 章では評価結果の検討を行う．

2 部分グラフ同型判定問題

この章では，まずこの論文で使用するグラフの用語と表記法についての定義を説明する．次に Ullmann のアルゴリズムについて述べ，最後に小西のアルゴリズムについて説明する．

2.1 用語と表記法について

本論文では，論文 [6] で使用されている用語や記号をそのまま使用する．

グラフ G は，有限な p 個の頂点からなる頂点集合 V と，q 個の辺からなる辺集合 E で構成される．一般に $G = (V,\ E)$ と記述する．ここで V の要素を $v_i(1 \leq i \leq p)$ とし，E の要素 $e_k(1 \leq k \leq q)$ を $\{v_i, v_j\}$ と記述する $(i \neq j)$. この論文では無向グラフを対象とし，同じ 2 頂点で構成される辺が複数存在しないとする．さらに自分自身の頂点と結合している辺 $\{v_i, v_i\}$ はないとする．なお，辺 $\{v_i, v_j\}$ は辺 $\{v_j, v_i\}$ と同一とする．

辺集合 E を表現する方法として隣接行列がある．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．

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

ある 2 つのグラフ $G_\alpha = (V_\alpha, E_\alpha)$ と $G_\beta = (V_\beta, E_\beta)$ が与えられたとする $(p_\alpha \leq p_\beta)$.

G_β の部分グラフ G_α とは $V_\alpha \subset V_\beta$, $E_\alpha \subset E_\beta$ となっているものをいう．2 つのグラフ G_α と G_β の頂点の間に一対一対応があり，しかも G_α の任意の 2 点を結ぶ各辺が G_β の対応する 2 点を結ぶ各辺と等しい時に，G_α と G_β は同型 (isomorphic) であるという．さらに $V \subseteq V_\beta, E \subseteq E_\beta$ とした時，以下のような写像 f が存在すれば，「G_α

と G_β は部分グラフ同型」であるという．

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

上記の条件を満たす写像 f が存在するかどうかを調べる問題を「部分グラフ同型判定問題」と呼ぶ．以下では，部分グラフ同型問題を解くアルゴリズムを部分グラフ同型判定アルゴリズムとする．

この問題は一種の探索空間問題なので，ここでは深さ優先探索を行う．従って，f は V_α の全要素に対して一度に決定されず，$v_{\alpha 1}, v_{\alpha 2}, \cdots, v_{\alpha p_\alpha}$ の順番に決定される．ここで，$v_{\alpha 1}, v_{\alpha 2}, \cdots, v_{\alpha p_\alpha}$ の写像先をそれぞれ深さ $1, 2, \cdots, p_\alpha$ の写像と記述し，全体の写像に対して深さ p_α の探索木を作る．以下では，深さ d の写像と書かれた場合は，$v_{\alpha d}$ に対しての写像 f を指すものとする $(1 \leq d \leq p_\alpha)$.

次に辺密度の定義について説明する．ここで頂点数 p の完全グラフを K_p とする．グラフ G の辺密度 ed とは，完全グラフ K_p の辺数に対して実際に存在する辺数 q の比である．

$$ed \ = \ \frac{q}{\frac{p(p-1)}{2}} \tag{2}$$

本研究の部分グラフ同型判定問題では，扱うグラフが連結であることが前提とされている．グラフ G が連結であるために必要な最小の辺数は $(p-1)$ なので，$q \geq p-1$ が必要である．従って，G が連結であるための必要条件は (3) 式で与えられる．

$$ed \ \geq \ \frac{2}{p} \tag{3}$$

辺数 $(p-1)$ でグラフ全体が連結である場合，そのグラフは木 (tree) である．任意の頂点 p 個を $(p-1)$ 本の辺で結んで木にすることは常に可能なので (生成木)，(3) 式を満たしていれば必ず対応する連結グラフを生成することができる．

2.2 Ullmann のアルゴリズム

Ullmann の部分グラフ同型判定アルゴリズム [1] は，2 つのアルゴリズムから構成されており，一つは探索空間削減の refinement procedure であり，もう一つは探索木巡回アルゴリズムである．以下，refinement procedure と探索巡回アルゴリズムについて説明する．

2.2.1 Refinement procedure

2.1 節で述べたように，部分グラフ同型を検出するためには，(1) 式が成立するような写像 f をみつける必要がある．しかし，この写像 f の選び方は ${}_{p_\beta}P_{p_\alpha}$ 通りあるので，p_α と p_β の値が大きくなると実用時間内では全数探索が不可能になる．そこで計算量を減らすために refinement procedure で探索空間の削減を行う．

Refinement procedure を説明する前にこれから使用する記号について定義を行う．G_α,G_β の隣接行列をそれぞれ $A = [a_{ij}], B = [b_{ij}]$ とする．配列 $M = [m_{ij}]$ 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 $v_{\beta j}$ へ写像した時に，部分グラフ同型の条件を満足する可能性があることを示す．M の初期値は以下の条件によって決定される．$deg(v)$ は頂点 v の次数である．

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

Refinement procedure は全ての i, j に対して (4) 式を満たすか否かを調べる．もし否であれば，$m_{ij} = 0$ とする．この作業は配列 M が変化しなくなるまで繰り返す．

$$\underset{1 \leq x \leq p_\alpha}{\forall x} \ (\ (a_{ix} = 1) \ \Rightarrow \ \underset{1 \leq y \leq p_\beta}{(\exists y)} \ (m_{xy} \cdot b_{yj} = 1)\ ) \tag{4}$$

Refinement procedure アルゴリズムを図 1 に示す．図中で B_j は配列 B の j 列目を取り出した列ベクトルを表す．M_i は M の i 行目を取り出したものを示す．$elim$ は refinement procedure を実行後，削除された候補数であ

る．deg_i は $v_{\alpha i}$ の次数である．sc は p_β ビットの幅を持ち，1 ビットのみが 1 となるデータである．:= は代入を表す．& はビット毎の論理積で，NOT はビット毎の反転である．do-while 文,while 文,if 文,break 文,return 文は C 言語と同じ使用法である．

Refinement procedure を実行中に，ある行の配列 $M(M_i$ とする) の全要素が 0 になるときがある．これは，頂点 $v_{\alpha i}$ に対してどの頂点 $v_{\beta j}$ を選択しても部分グラフ同型にならないことを示している．このような状態が起こると，アルゴリズムの実行を中止し，返却値を FAIL として返す．

```
do
{    elim := 0;
   i := 1;
   while(i ≤ pα) {
     j := 1;
     sc := 2^(pβ−1);
     while(j ≤ pβ) {
       if((Mi&sc ≠ 0) {
         h := 1;
         while(h ≤ degi) {
           x := lsth;
           if((Mi&Bj ≠ 0) {
             Mi := Mi&NOTsc;
             elim := elim + 1;
             h := h + 1;
             break;
           }
           h := h + 1;
         }
       }
       sc := sc/2;
       j := j + 1;
     }
     if(Mi = 0) return(FAIL);
     i := i + 1;
   }
} while(elim ≠ 0);
return(SUCCEED);
```

図 1: Refinement procedure [1]

2.2.2 探索木巡回アルゴリズム

探索木巡回アルゴリズムは，深さ優先探索 (depth-first search) と同じアルゴリズムである．図 2 は探索木巡回アルゴリズムである．図中で M^0 は配列 M の初期値である．$M := M^0$ は配列 M^0 の全要素を配列 M に代入する．なお，$M_d := M$ の場合は配列 M と同じ大きさ M_d に配列 M の全要素を代入する．$H_i = k$ は深さ i で $v_{\alpha i}$ の写像として $v_{\beta k}$ が設定されていることを表す $(1 \leq i \leq p_\alpha)$.$F_i = 1$ は写像先として $v_{\beta i}$ が既に使用されていることを表す $(1 \leq i \leq p_\alpha)$．make lst_i and deg_i は，G_α の頂点集合と次数を作成する．refine M は，refinement procedure アルゴリズムを呼び出す．

2.3 小西のアルゴリズム

2.2.1 節で述べたように，部分グラフ同型判定問題を実用時間内で計算するために，Ullmann のアルゴリズムでは refinement procedure を用いる．refinement procedure はハードウェア化することが可能だが，探索空間削減の部分のみで，論理素子が $O(p_\alpha p_\beta^2)$ ゲート必要となる [7]．これでは，$p_\alpha = 15, p_\beta = 15$ 程度でも OR2CxxA シリーズ FPGA には収まらない．これに対して，小西が提案したアルゴリズム [6] では探索空間削減法を refinement procedure より単純にすることによって，ハードウェアの使用量を押さえている．以下，小西 [6] が提案した探索空間削減を行う辺存在確認アルゴリズムと，探索木巡回アルゴリズムについて説明する．

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; $F_j := 0$, if $j \neq H_i$ for all $i < d;$ $d := d - 1;\ M := M_d;\ k := H_d;$ go to step 5;

図 2: 探索木巡回アルゴリズム [1]

2.3.1 辺存在確認アルゴリズム

辺存在確認アルゴリズム [6] では以下のようにして探索空間の削減を行なう．深さ d までの写像が決定された時，G_α の分部グラフに含まれる辺の写像が G_β に含まれるかどうかを調べる (必要条件の検査)．もし該当辺がなければ，部分グラフ同型にならない (必要条件を満たさない) ので次の写像を調べる．

図 3 は辺存在確認アルゴリズムである．$etsa$ は edge list table starting address であり，eta, etb は edge list table である．B は G_β の隣接行列である．while 文,if 文,return 文は C 言語と同じ使用法である．

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

図 3: 辺存在確認アルゴリズム [6]

この辺存在確認アルゴリズムは，2 つのサブユニットからなる．一つは写像 f から辺 $e_{\alpha i}$ に対応した辺 $e_{\beta j}$ を導出する部分である (これ以後,“ 辺導出部 ”と呼ぶ)．もう一つは導出された辺 $e_{\beta j}$ 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である (これ以後,“ 存在確認部 ”と呼ぶ)．

- 辺導出部

  図 4 は辺導出部の概念図である．破線で囲まれた部分が辺存在確認アルゴリズムである．このアルゴリズムへの入力データは，探索木巡回アルゴリズム (図中では“ STV algorithm ”) から出力された $depth$ と $vb(i)(1 \leq i \leq p_\alpha)$ である．そしてもう一つは，存在確認部 (図中では“ edge existence check module ”) からの入力は $exist$ である．このアルゴリズムからの出力は，探索木巡回アルゴリズムへの出力信号 $result$，存在確認部への出力データ a と b である．$depth$ は探索木中の深さ $d(1 \leq d \leq p_\alpha)$ である．$vb(i)$ は G_α の頂点 $v_{\alpha i}(1 \leq i \leq p_\alpha)$ 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．

  Edge list table は，$p_\alpha + q_\alpha$ の要素からなり，深さ 1 から p_α まで調べる必要がある辺を構成する 2 頂点の点番号が順番に格納されている．但し深さが変わる時には識別するために (0,0) が挿入されている．図中で eta, etb はそれぞれ Edge list table に格納されているデータの第 1 項目と第 2 項目である．$etsa$(Edge list table starting address) は p_α の要素を持ち, 深さ d での edge list table の開始アドレスを保持している．

- 存在確認部

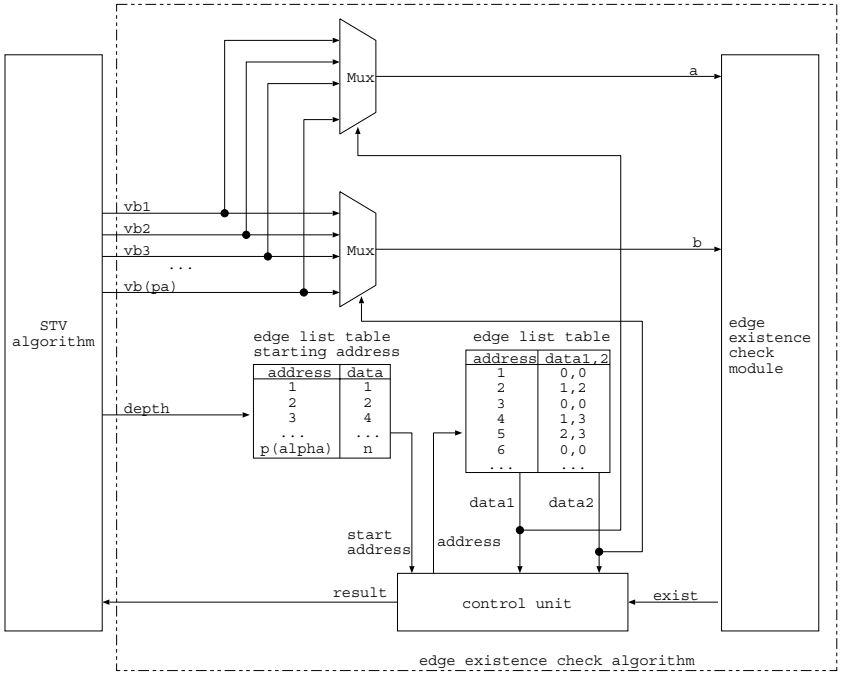


図 4: 辺導出部の概念 [6]

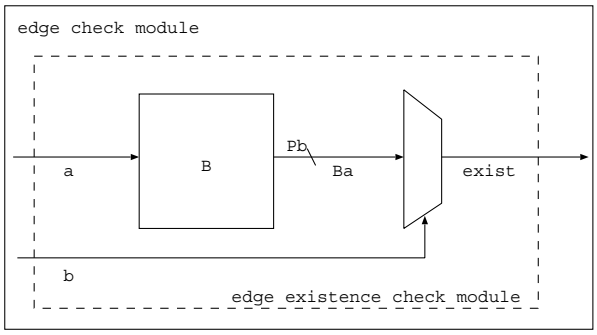


図 5: 存在確認部の概念 [6]

存在確認部の概念図を図 5 に示す．この回路の入力としては，辺導出部から出力された $\{v_{\beta a}, v_{\beta b}\}$ 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．

2.3.2 探索木巡回アルゴリズム

小西の探索木巡回アルゴリズム [6] は，探索空間削減のところを除いて Ullmann の探索木巡回アルゴリズムと同じものである．図 6 は探索木巡回アルゴリズムである．図中で M^0 は 2.2.1 節で使用された初期値配列と同じものである．d は深さである．M_d は配列 M の d 行目である．p_β ビットの $used$ は写像の記憶用変数である．$used(i) := 0$ は $v_{\beta i}$ への写像が使用済みである．p は p_α の要素を持ち，$p(i)$ は $v_{\alpha i}$ の写像先頂点番号が格納されている．

図 7 は探索空間巡回アルゴリズムの概念図である．図中に 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．ただし複数のビットが 1 になった場合，上位のビットの優先度の方が高い．DEC はデコーダである．control は回路全体制御する部分である．これらの回路の実装について論文 [6] では今後の課題として残した．

```
M := M^0; IM := M^0; d := 1; used := 1;
while (1) {
    current := M_d & used;
    while (current ≠ 0) {
      current(i) = 1 となる最初の i を設定;
      p(d) := i; current(i) := 0; flag := 0;
      “ 辺存在確認アルゴリズム ”を実行;
      if (返り値 = OK) {
        if (d = p_α) 写像を記憶;
        else {used(i) := 0; M_d := current; d := d + 1;
              flag := 1; break;}
      }
    }
    if (flag) continue;
    if (d = 1) terminate algorithm;
    M_d := IM_d; d := d − 1; used(p(d)) := 1;
}
```

図 6: 探索木巡回アルゴリズム [6]

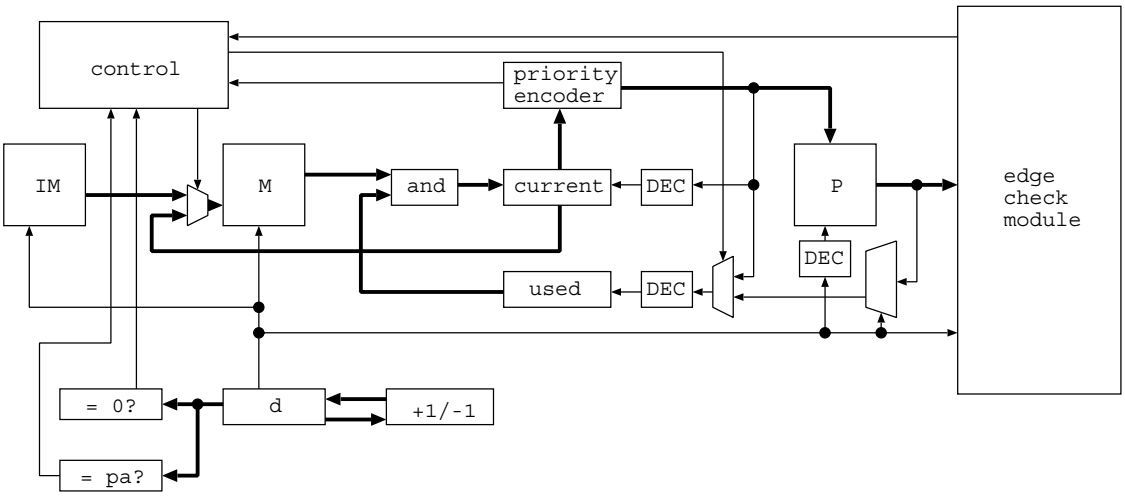


図 7: 探索空間巡回アルゴリズムの概念図 [6]

3 小西のアルゴリズムの性能評価

この章では Ullmann のアルゴリズムと小西のアルゴリズムの性能評価を行うためのシミュレーション方法について説明する (以下，Ullmann のアルゴリズムと小西のアルゴリズムをまとめて呼ぶ時に，両方式のアルゴリズムと言う)．次に両方式のアルゴリズムへの入力データの生成方法について説明する．その後シミュレーションの実行環境と測定方法について述べる．最後に，両方式のアルゴリズムの性能評価を行う．

3.1 シミュレーション方法

この節では小西のシミュレーション [6] の問題点を明らかにし，その後本研究での改善点について説明する．

3.1.1 小西のシミュレーションの問題点

論文 [6] では，入力としたグラフの辺密度が小さいときに小西のアルゴリズムの優位性が失われるとされている．原因を調査した結果，小西のシミュレーションでは両方式のアルゴリズムへ与えた入力グラフの辺密度が小さすぎる場合があることがわかった．つまり，(3) 式を満たさない辺密度の入力データを入力したため，非連結なグラフが両方式のアルゴリズムへの入力となっている場合があった．

非連結グラフを入力として与えると，refinement procedure では配列 M のある行の要素全てが 0 になる．このような状態が生じると refinement procedure は FAIL exit にジャンプして実行を中止し，呼び出し側の探索木巡回アルゴリズムに戻ってアルゴリズムの実行を中断する．

一方，小西のアルゴリズムの場合は，配列 M のある行の要素全てが 0 であることを確認せず，グラフ G_α の全ての深さに対して，部分グラフとなり得るか否かを調べている．このため入力グラフが非連結グラフであっても，小西のアルゴリズムはグラフ G_α の全ての深さに対して調べてから，アルゴリズムの実行を終了する．

つまり，refinement procedure による探索木巡回アルゴリズムは，$deg(v_{\alpha i}) > deg(v_{\beta j})$ の場合 (部分グラフ同型になり得ない条件の一つ) に満たすグラフを始めから計算しないが，小西のアルゴリズムは部分グラフ同型になり得るかなり得ないかにもかかわらずアルゴリズムの最後まで実行するため性能が悪いように見える．しかし，そもそもこれは部分グラフ同型判定の用途を逸脱した使い方であるため，本質的に無意味である．そこで本論文では入力データを再検討し，より適切で詳細な性能評価を行なう．

3.1.2 本研究での改善点

本研究で行うシミュレーションの改善点は 3 つである．一つ目はグラフの連結性の考慮，二つ目は入力グラフの頂点数の範囲の吟味，三つ目は入力データの性質を考慮した性能評価であり，それぞれについて以下に説明する．

- グラフの連結性

  本研究では全ての入力データに対して連結性なグラフを与えるために，生成木からグラフを生成することにする．

- 入力グラフの頂点数の範囲の吟味

  連結グラフを生成するために，(3) 式を満たす辺密度と頂点数を選ばなければならない．小西のシミュレーション [6] では，グラフの頂点数が 1〜10 までの範囲と，辺密度 $ed_\alpha = 0.2$ の時に入力グラフの生成を行った．しかし，この頂点数の範囲の中で連結グラフを作成できるのは，頂点数が 10 の時のみである．よって，広範囲の性能評価を行うために，入力グラフの頂点数を吟味する必要である．

- 入力データの性質に関する考慮

  ここで言う入力データの性質による性能評価とは，(A) 部分グラフ同型の判定に成功する入力データと，(B) 失敗する入力データについての性能評価のことである．(A),(B) の場合の入力データで性能が異なる可能性があるので，本研究ではそれぞれついて評価を行うことにする．

  ここで (A),(B) の場合について定義する．(A) は，G_β に G_α と同型な部分グラフが 1 つ以上含まれる場合であり，以下 $G_\alpha \subset G_\beta$ と表す．また，(B) は，G_β に G_α と同型な部分グラフが含まれない場合であり，以下 $G_\alpha \not\subset G_\beta$ と表す．

3.2 入力データの生成方法

この節では両方式のアルゴリズムへの入力データを生成するために必要な制約条件について述べる．次に入力グラフの生成アルゴリズムを説明する．その後 $G_\alpha \subset G_\beta$ の場合と，$G_\alpha \not\subset G_\beta$ の場合の入力データを生成する方法について説明する．

3.2.1 入力データの制約条件

まずグラフ G_α と G_β が共に連結であることが必要である．これ以後，G_α, G_β の辺密度をそれぞれ ed_α, ed_β とし，G_α, G_β の頂点数をそれぞれ p_α, p_β とする．また G_α, G_β の辺数をそれぞれ q_α, q_β とする．

以下，$p_\alpha, p_\beta, q_\alpha, q_\beta$ を求める式について説明する．また，p_α と p_β，q_α と q_β の関係を示す．

部分グラフ同型判定問題では，扱っている p_α が p_β 以下でなければ解を持たない．従って本論文でも以下の条件を満たす場合についてシミュレーションを行う．

$$p_\beta \geq p_\alpha \tag{5}$$

同様に，G_α が G_β の部分グラフであるための必要条件として，q_β が q_α 以上であることが必要である．

$$q_\beta \geq q_\alpha \tag{6}$$

(3) 式より，p_α,p_β は以下の式を満たす必要がある．

$$p_\alpha \geq \frac{2}{ed_\alpha} \tag{7}$$

$$p_\beta \geq \frac{2}{ed_\beta} \tag{8}$$

一方 (2) 式を変形すると以下の式が得られる．

$$q_\alpha = \frac{p_\alpha(p_\alpha - 1)}{2} \cdot ed_\alpha \tag{9}$$

$$q_\beta = \frac{p_\beta(p_\beta - 1)}{2} \cdot ed_\beta \tag{10}$$

ここで，辺数 q_α, q_β は整数でなければならないが，(9),(10) 式では q_α, q_β が整数になるとは限らない．このため切り上げ，切り下げ，四捨五入などの手法で整数化する必要がある．切り上げと切り下げの場合は，自分が定めた期待値に対して，得られる値がそれ以上かそれ以下のどちらかに偏ってしまう性質を持っている．一方，四捨五入の場合はどちらにも偏らないことが期待できる (表 6 を参照のこと)．従って，本研究では q_α, q_β を求める時，四捨五入を行うことにする．結局 q_α, q_β は以下の式で求めることにする．ただし，$round()$ は四捨五入を行う関数である．

$$q_\alpha = round\left(\frac{p_\alpha(p_\alpha - 1)}{2} \cdot ed_\alpha\right) \tag{11}$$

$$q_\beta = round\left(\frac{p_\beta(p_\beta - 1)}{2} \cdot ed_\beta\right) \tag{12}$$

3.2.2 入力グラフ生成のアルゴリズム

図 8 は入力グラフ生成のアルゴリズムである．このアルゴリズムは，a 与えられた頂点数 p, 辺数 q を持つ生成木を持つグラフ G をランダムに作成する．その手順は以下の通りである．(i) まず与えられた頂点数 p の生成木を作成し，(ii) その生成木に $q-(p-1)$ 本の辺を追加して，与えられた辺数 q を持つグラフ G を作る．ここで，p と q は (5),(6),(7)〜(12) 式を満たすものとする．

図中の C は G の隣接行列 $[c_{ij}]$ である．if 文,while 文,for 文,break 文,return 文は C 言語と同じ使用方法である．

3.2.3 $G_\alpha \subset G_\beta$ の場合の入力データ

$G_\alpha \subset G_\beta$ の場合の入力データを生成するために，本研究では図 9 に示す方法で行うことにする．図中の A は G_α の隣接行列 $[a_{ij}](1 \leq i, j \leq p_\alpha)$，$B$ は G_β の隣接行列 $[b_{kl}]$ である $(1 \leq k, l \leq p_\beta)$.

$G_\alpha \subset G_\beta$ となるように，G_α の隣接行列 $[a_{ij}]$ を G_β の隣接行列 $[b_{ij}]$ に埋め込む．これだけでは $G_\beta = G_\alpha$ となるので，更に頂点 $(p_\beta - p_\alpha)$ 個と辺 $(q_\beta - q_\alpha)$ 個を追加することにより，所定の頂点数と辺数を持つ G_β を生成す

```
gen_graph(C, p, q)
{
  while(1){
    for all i, j := 1, ..., p, set c_ij := 0;
    for(k := 1; k ≤ (p − 1); k++) {
      乱数によって i, j を決定 (i ≠ j, 1 ≤ i, j ≤ p);
      C[i][j] := 1;
    }
    if(C = 木) break;
  }
  for(k := 1; k ≤ q − (p − 1); k++) {
    while(1) {
      乱数によって m, n を決定 (m ≠ n, 1 ≤ m, n ≤ p);
      if(c_mn ≠ 1) { c_mn := 1; break; }
    }
  }
  return C;
}
```

図 8: 入力グラフ生成アルゴリズム gen_graph

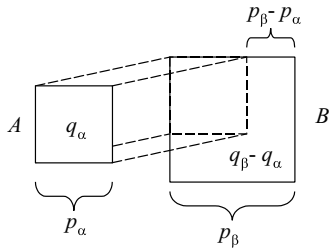


図 9: G_αを G_β に埋め込む場合

ることにする．頂点 $(p_\beta - p_\alpha)$ 個を追加してグラフ G_β を連結に保つには明らかに $(p_\beta - p_\alpha)$ 以上の辺を追加する必要がある．このことより (13) 式が必要条件として得られる．

$$q_\beta - q_\alpha \quad \geq \quad p_\beta - p_\alpha \tag{13}$$

上記の方法を基に $G_\alpha \subset G_\beta$ の場合の入力データを生成するアルゴリズムを図 10 に示す．図中に使用する p_α,p_β,q_α,q_β は (5),(6),(7)〜(13) 式を満たすものとする．if 文,while 文,do-while 文,for 文,break 文は C 言語と同じ使用方法である．return X, Y は隣接行列 X と Y を返す．

このアルゴリズムの入力は，G_α の頂点数 p_α, 辺数 q_α と G_β の頂点数 p_β, 辺数 q_β である．最初のアルゴリズムでは，G_α の生成木を隣接行列 A に代入する．そして，G_β の隣接行列 $B = [b_{ij}]$ を初期化し，上記の方法で G_α を G_β の中に埋め込む．さらに $(q_\beta - q_\alpha)$ 本の辺を G_β に追加する．G_β が連結グラフになるまでこの作業を繰り返す．最後に，$G_\alpha \subset G_\beta$ という関係を満たす G_α と G_β を返す．

3.2.4 $G_\alpha \not\subset G_\beta$ の場合の入力データ

本研究では，素朴な方法で $G_\alpha \not\subset G_\beta$ なる入力データを作成することにした．それは，G_α と G_β に対してそれぞれランダムに連結グラフを作成し，$G_\alpha \not\subset G_\beta$ のような関係を満たす 2 つのグラフを選んで入力データとして扱う方法である．そのアルゴリズムを図 11 に示す．図中で使用されている記号は 3.2.3 節で説明した Ga_into_Gb()

```
Ga_into_Gb(p_α, q_α, p_β, q_β)
{
  gen_graph(A, p_α, q_α);
  while(1) {
  for all i, j := 1, ..., p_β, set b_ij := 0;
  A を B に埋め込む；
    for(k := q_α; k ≤ q_β; k++) {
      while(1) {
        乱数によって i, j を決定 (i ≠ j, 1 ≤ i, j ≤ p_β);
        if(b_ij ≠ 1) { b_ij := 1; break; }
      }
    }
    if ( B = 連結グラフ )   break;   }
  return A, B;
}
```

図 10: $G_\alpha \subset G_\beta$ の場合の入力データ生成のアルゴリズム Ga_into_Gb

関数と同じ意味を持つものである．

このアルゴリズムの入力は，G_α の頂点数 p_α, 辺数 q_α と G_β の頂点数 p_β, 辺数 q_β である．最初のアルゴリズムでは，G_α と G_β の生成木をそれぞれ隣接行列 A, B に代入する．そして，G_β の中に G_α と部分グラフ同型が存在するか否かを調べる．もし存在していれば，G_α と G_β の生成木を作り直す．もし否であれば，$G_\alpha \not\subset G_\beta$ のような関係を持つ G_α と G_β を返す．

```
Ga_notin_Gb(p_α, q_α, p_β, q_β)
{
  while(1) {
    gen_graph(A, p_α, q_α);
    gen_graph(B, p_β, q_β);
    if( A ⊄ B ) break;
  }
  return A, B;
}
```

図 11: $G_\alpha \not\subset G_\beta$ の場合の入力データ生成のアルゴリズム Ga_notin_Gb

3.3 実行環境

小西の研究 [6] では，33MHz で動作している OPERL ボード上に小西のアルゴリズムを実装すると仮定していた．本研究でも同じ仮定を置いて性能評価を行う．

Ullmann のアルゴリズムは FreeBSD-3.1 上に C 言語で実装した．実行環境を表 1 にまとめる．この計算機は 1999 年時点で，標準的な仕様を設定している．

プロセッサ	Intel Pentium-II 400MHz
チップセット	Intel 440BX AGPset
２次キャッシュ	512KB(Pentium-II に内蔵)
主記憶	SDRAM 256MB(60ns)
基本ソフトウェア	FreeBSD-3.1
C 言語コンパイラ	gcc ver.2.7.2.1

表 1: Ullmann のアルゴリズムを実装したシステム仕様

3.4 性能測定方法

ここで行なう性能とはアルゴリズムの実行時間のことであり，実行時間が短いほど高性能であると言える．以下，両方式のアルゴリズムに与える入力パラメータの設定について述べる．次に Ullmann のアルゴリズムの性能測定方法について説明する．最後に，小西のアルゴリズムの性能測定方法を説明する．

3.4.1 入力パラメータの設定

両方式のアルゴリズムに与える入力パラメータは 4 つである．

- グラフ G_α,G_β の辺密度 ed_α, ed_β
- グラフ G_α,G_β の頂点数 p_α, p_β

これらの組み合わせは無数にある．しかし $G_\alpha \subset G_\beta$ の場合または $G_\alpha \not\subset G_\beta$ の場合の入力データを生成する際，不可能な組み合わせが存在する．以下，ed_α と p_α，ed_β と p_β をどのように組み合わせると妥当なパラメータになるか否か説明する．

(I) 入力データが $G_\alpha \subset G_\beta$ である場合

p_αと ed_α，p_βと ed_β は (5),(6),(7)〜(13) 式を満たす必要がある．ed_α, ed_β を決めたとき，この条件を満たす p_α, p_β は以下のように求められる．これから，(5),(6),(7)〜(13) 式を使って入力パラメータを求める例を示す．

例 1: $(ed_\alpha, ed_\beta) = (0.2, 0.2)$ の場合

(5),(6) 式より，$p_\alpha \geq \frac{2}{0.2} = 10$, $p_\beta \geq \frac{2}{0.2} = 10$

- $p_\alpha = 10$ とすると，
  - (11) 式より，$p_\beta \geq 10$
  - (7) 式より，$q_\alpha = round\left(\frac{10 \cdot 9}{2} \cdot 0.2\right) = 9$
  - 同様に (8) 式より，$q_\beta \geq 9$　(これも (12) 式を満たす)
  - 以上より，$p_\alpha = 10$ の時に (13) 式を満たす p_β は $p_\beta = 10, 11, 12, \ldots$
- $p_\alpha = 11$ とすると，
  - (11) 式より，$p_\beta \geq 11$
  - (7) 式より，$q_\alpha = round\left(\frac{11 \cdot 10}{2} \cdot 0.2\right) = 11$
  - 同様に (8) 式より，$q_\beta \geq 11$　(これも (12) 式を満たす)
  - 以上より，$p_\alpha = 11$ の時に (13) 式を満たす p_β は $p_\beta = 11, 12, 13, \ldots$ □

例 2: $(ed_\alpha, ed_\beta) = (0.4, 0.2)$ の場合

(5),(6) 式より，$p_\alpha \geq \frac{2}{0.40} = 5$, $p_\beta \geq \frac{2}{0.2} = 10$

- $p_\alpha = 9$ とすると，

– (11) 式より，$p_\beta \geq 10$

– (7) 式より，$q_\alpha = round\left(\frac{10\cdot 9}{2}\cdot 0.4\right) = 14$

– 同様に (8) 式より，$q_\beta \geq 9$

– しかし，(12) 式を満たすために，$q_\beta \geq 14$

– p_α, q_α の値を (13) 式に代入すると，

$p_\beta - 9 \leq q_\beta - 14$

$q_\beta \geq p_\beta + 5$

$p_\beta \geq 10$，$q_\beta \geq 15$，上の式を満たすものは，

$p_\beta = 10 \Rightarrow q_\beta = round\left(\frac{10\cdot 9}{2}\cdot 0.2\right) = 9$ ×

$p_\beta = 11 \Rightarrow q_\beta = round\left(\frac{11\cdot 10}{2}\cdot 0.2\right) = 11$ ×

$p_\beta = 12 \Rightarrow q_\beta = round\left(\frac{12\cdot 11}{2}\cdot 0.2\right) = 14$ ×

$p_\beta = 13 \Rightarrow q_\beta = round\left(\frac{13\cdot 12}{2}\cdot 0.2\right) = 16$ ×

$p_\beta = 14 \Rightarrow q_\beta = round\left(\frac{14\cdot 13}{2}\cdot 0.2\right) = 18$ ×

$p_\beta = 15 \Rightarrow q_\beta = round\left(\frac{15\cdot 14}{2}\cdot 0.2\right) = 21$ ○

$p_\beta = 16 \Rightarrow q_\beta = round\left(\frac{16\cdot 15}{2}\cdot 0.2\right) = 24$ ○

$\vdots$

– 以上より，$p_\alpha = 9$ の時に (13) 式を満たす p_β は

$p_\beta = 15, 16, 17, \ldots$ □

(5),(6),(7)〜(13) 式を満たす p_αと ed_α，p_βと ed_β の組み合わせは無数に存在するが，本研究で実装に使用するハードウェアの制約に合わせて $p_\alpha, p_\beta \leq 15$ の範囲で検討を行うこととする．

これまでは ed_α, ed_β が決まっているものとして p_α, p_β を考えてきた．実際に ed_α, ed_β を与える際には，どのような値が適切か考える必要がある．ed は 0〜1 の値を持つが，実応用上では余り大きい値を持つ (dense な) グラフを扱うことは少ない．そこで，ed_α と ed_β の大小関係も考えて以下の 4 通りを検討することにする．

(a) ed_α が薄い,ed_β が薄い場合: $(ed_\alpha, ed_\beta) = (0.2, 0.2)$

(b) ed_α が薄い,ed_β が濃い場合: $(ed_\alpha, ed_\beta) = (0.2, 0.4)$

(c) ed_α が濃い,ed_β が薄い場合: $(ed_\alpha, ed_\beta) = (0.4, 0.2)$

(d) ed_α が濃い,ed_β が濃い場合: $(ed_\alpha, ed_\beta) = (0.4, 0.4)$

上記の 4 つの場合に対してそれぞれ上に述べた方法で p_α と p_β の範囲を決める．求めた結果をそれぞれ表 2,3,4,5 にまとめて示す．表中で○は入力パラメータとして使用可能であり，×は入力パラメータとして使用不可能であることを示す．

表 2: $G_\alpha \subset G_\beta$，$(ed_\alpha, ed_\beta) = (0.2, 0.2)$ の場合，p_α と p_β の関係

		p_β					
		10	11	12	13	14	15
p_α	10	○	○	○	○	○	○
	11	×	○	○	○	○	○
	12	×	×	○	○	○	○
	13	×	×	×	○	○	○
	14	×	×	×	×	○	○
	15	×	×	×	×	×	○

次に，ed_α と ed_β の平均値と標準偏差について説明する．本研究では，q_α と q_β を計算する時に四捨五入を行うため，上の (a)〜(d) で定めた ed_α, ed_β を用いてグラフを生成しても，得られたグラフの ed_α, ed_β は一般に狙った値と等しくはならない．そこで ed_α と ed_β の平均値と散布度を検討した．表 2,3,4,5 で使用可能な入力パラメータについて，既に述べたアルゴリズムで生成したグラフの ed_α と ed_β について，平均値と標準

表 3: $G_\alpha \subset G_\beta$，$(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合，p_α と p_β の関係

p_α \ p_β	10	11	12	13	14	15
10	○	○	○	○	○	○
11	×	○	○	○	○	○
12	×	×	○	○	○	○
13	×	×	×	○	○	○
14	×	×	×	×	○	○
15	×	×	×	×	×	○

表 4: $G_\alpha \subset G_\beta$，$(ed_\alpha, ed_\beta) = (0.4, 0.2)$ の場合，p_α と p_β の関係

p_α \ p_β	10	11	12	13	14	15
5	○	○	○	○	○	○
6	×	○	○	○	○	○
7	×	×	○	○	○	○
8	×	×	×	×	○	○
9	×	×	×	×	×	○

偏差求めた結果を表 6 に示す．表 6 からわかるように，得られたグラフの ed_α, ed_β は狙った値 ed_α, ed_β とほとんど変わらなかった．標準偏差も小さく収まっている．

最後に，両方式の性能測定について説明する．本研究では，入力グラフ G_α,G_β をランダムに 100 回与えた時の合計実行時間を平均したものを求める．当然ながら与える回数が多いほど性能測定結果の精度が良くなるが，膨大な計算時間も必要となる．実際に試みた結果，100 回の平均と 500 回の平均で結果に有意な差はなかった．

(II) 入力データが $G_\alpha \not\subset G_\beta$ である場合

$G_\alpha \subset G_\beta$ の場合の性能評価と同様に検討する．p_α と p_β の範囲を決める方法は (I) の例 1 に示した方法と同様であるが，$G_\alpha \not\subset G_\beta$ の場合 (13) 式を満たす必要はない．

$G_\alpha \not\subset G_\beta$ であるような入力データを生成するためには，(5),(6),(7)〜(12) 式を満たすだけでは不十分であり，生成したグラフのパターン数も考慮する必要がある．なぜなら，p_α,q_α が小さい時には生成可能なグラフ G_α のパターン数が限られているため，大きなグラフ G_β の中に G_α と同型な部分グラフが存在しないように G_α と G_β を生成することが困難になるからである．

表 7,8,9,10 に p_α, p_β の範囲を求めた結果を示す．表中で○は入力パラメータとして使用可能であり，×は入力パラメータとして使用不可能であることを示す．△は入力パラメータとして使用可能であるが $G_\alpha \not\subset G_\beta$ のグラフを作成することが困難であることを示す．

表 7,8,9,10 で使用可能な入力パラメータについて，生成されたグラフの ed_α と ed_β の平均値と標準偏差を表 11 に示す．表 11 からわかるように，四捨五入で q_α と q_β を求めても誤差は許容範囲内である．

最後に，両方式の性能測定について説明する．ここで $G_\alpha \not\subset G_\beta$ の場合の両方式の性能測定は，$G_\alpha \subset G_\beta$ と同様に行う．

表 5: $G_\alpha \subset G_\beta$，$(ed_\alpha, ed_\beta) = (0.4, 0.4)$ の場合，p_α と p_β の関係

p_α \ p_β	5	6	7	8	9	10	11	12	13	14	15
5	○	○	○	○	○	○	○	○	○	○	○
6	×	○	○	○	○	○	○	○	○	○	○
7	×	×	○	○	○	○	○	○	○	○	○
8	×	×	×	○	○	○	○	○	○	○	○
9	×	×	×	×	○	○	○	○	○	○	○
10	×	×	×	×	×	○	○	○	○	○	○
11	×	×	×	×	×	×	○	○	○	○	○
12	×	×	×	×	×	×	×	○	○	○	○
13	×	×	×	×	×	×	×	×	○	○	○
14	×	×	×	×	×	×	×	×	×	○	○
15	×	×	×	×	×	×	×	×	×	×	○

表 6: $G_\alpha \subset G_\beta$ の時，ed_α, ed_β の平均値と標準偏差

Given		Average		s.d.	
ed_α	ed_β	$ed_\alpha^{av.}$	$ed_\beta^{av.}$	$ed_\alpha^{s.d.}$	$ed_\beta^{s.d.}$
0.20	0.20	0.19811	0.19758	0.00267	0.00280
0.20	0.40	0.19811	0.39760	0.00267	0.00231
0.40	0.20	0.39436	0.19772	0.00783	0.00270
0.40	0.40	0.39474	0.39611	0.00665	0.00470

3.4.2 Ullmann のアルゴリズムの性能測定方法

Ullmann のアルゴリズムの性能測定には，FreeBSD のシステムコール getrusage() を用いる．探索木巡回アルゴリズム [1] の step1 の始めと step7 の終わりに getrusage() を置いて時間の差を求める．これは，一組のグラフに対する実行時間である．本研究では，ランダムに生成した 100 パターンの入力グラフ G_α,G_β に関する実行時間を測定し，平均値を求めたものを Ullmann のアルゴリズムの実行時間とする．

3.4.3 小西のアルゴリズムの性能測定方法

小西のアルゴリズムの性能測定は，辺存在確認アルゴリズムと探索木巡回アルゴリズムを C 言語で実装し，OPERL ボード上で最小限必要となるであろうクロック数を見積もることによって行なう．処理した合計のクロック数が求まれば周波数の逆数をかけることにより，小西のアルゴリズムの実行時間が求まる．ここでクロックの周波数は 16.5MHz であることと，OPERL ボードに実装した回路の個数は 1 であることを仮定する．クロック数の見積もりは図 12,13 と同様に設定している．図中で [ ] は見積もったクロック数を示している．本研究では，ランダムに生成した 100 パターンの入力グラフ G_α,G_β に関する実行時間を測定し，平均値を求めたものを小西のアルゴリズムの実行時間とする．

表 7: $G_\alpha \not\subset G_\beta$，$(ed_\alpha, ed_\beta) = (0.2, 0.2)$ の場合，p_α と p_β の関係

		p_β					
		10	11	12	13	14	15
p_α	10	○	○	○	○	○	○
	11	×	○	○	○	○	○
	12	×	×	○	○	○	○
	13	×	×	×	○	○	○
	14	×	×	×	×	○	○
	15	×	×	×	×	×	○

表 8: $G_\alpha \not\subset G_\beta$，$(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合，p_α と p_β の関係

		p_β					
		10	11	12	13	14	15
p_α	10	○	△	△	△	△	△
	11	×	○	△	△	△	△
	12	×	×	○	△	△	△
	13	×	×	×	△	△	△
	14	×	×	×	×	△	△
	15	×	×	×	×	×	△

表 9: $G_\alpha \not\subset G_\beta$，$(ed_\alpha, ed_\beta) = (0.4, 0.2)$ の場合，p_α と p_β の関係

		p_β					
		10	11	12	13	14	15
p_α	5	○	○	△	△	△	△
	6	○	○	○	○	○	○
	7	○	○	○	○	○	○
	8	×	○	○	○	○	○
	9	×	×	×	○	○	○
	10	×	×	×	×	○	○

表 10: $G_\alpha \not\subset G_\beta$，$(ed_\alpha, ed_\beta) = (0.4, 0.4)$ の場合，p_α と p_β の関係

p_α \ p_β	5	6	7	8	9	10	11	12	13	14	15
5	○	○	△	△	△	△	△	△	△	△	△
6	×	○	○	○	○	○	△	△	△	△	△
7	×	×	○	○	○	○	○	△	△	△	△
8	×	×	×	○	○	○	○	○	○	○	○
9	×	×	×	×	○	○	○	○	○	○	○
10	×	×	×	×	×	○	○	○	○	○	○
11	×	×	×	×	×	×	○	○	○	○	○
12	×	×	×	×	×	×	×	○	○	○	○
13	×	×	×	×	×	×	×	×	○	○	○
14	×	×	×	×	×	×	×	×	×	○	○
15	×	×	×	×	×	×	×	×	×	×	○

表 11: $G_\alpha \not\subset G_\beta$ の時，ed_α, ed_β の平均値と標準偏差

Given		Average		s.d.	
ed_α	ed_β	$ed_\alpha^{av.}$	$ed_\beta^{av.}$	$ed_\alpha^{s.d.}$	$ed_\beta^{s.d.}$
0.20	0.20	0.19750	0.19819	0.00281	0.00264
0.20	0.40	0.19862	0.40633	0.00242	0.00361
0.40	0.20	0.39438	0.19817	0.00635	0.00265
0.40	0.40	0.39808	0.39820	0.00243	0.00226

3.5 性能評価の結果

3.5.1 入力データが $G_\alpha \subset G_\beta$ である場合

$(ed_\alpha, ed_\beta) = (0.2, 0.2), (0.2, 0.4), (0.4, 0.2), (0.4, 0.4)$ の実行時間をそれぞれ図 14,15,16,17 に示す．それぞれの図中で (a) は小西のアルゴリズムの実行時間，(b) は Ullmann のアルゴリズムの実行時間，(c) は小西のアルゴリズムと Ullmann のアルゴリズムの実行時間比である．グラフ (a),(b),(c) は p_α を x 軸に取って表示している．グラフ (a),(b) の y 軸は実行時間を対数目盛で表している (単位:秒)．グラフ (c) の y 軸は実行時間比を対数目盛で表している (単位:倍)．グラフ (c) では，小西のアルゴリズムの実行時間が Ullmann のアルゴリズムの実行時間より小さければ 1 より大きな値で，大きければ 1 より小さな値で示す．

$(ed_\alpha, ed_\beta) = (0.2, 0.4)$ である時，小西のアルゴリズムの性能が Ullmann のアルゴリズムより 10 倍程度優れている．また (ed_α, ed_β)=(0.4,0.4) である時は，多くの点が Ullmann のアルゴリズムより性能が向上していることを示しているが，一部の点で Ullmann のアルゴリズムより性能が低下している．特に $(ed_\alpha, ed_\beta) = (0.2, 0.2)$ の時では 10 倍以上の性能が低下している場合もある．これは，小西のアルゴリズムの探索空間削減の効率が Ullmann のアルゴリズムより劣るためである．

3.5.2 入力データが $G_\alpha \not\subset G_\beta$ である場合

$(ed_\alpha, ed_\beta) = (0.2, 0.2), (0.2, 0.4), (0.4, 0.2), (0.4, 0.4)$ の実行時間をそれぞれ図 18,19,20,21 に示す．$(ed_\alpha, ed_\beta) = (0.2, 0.2), (0.4, 0.2), (0.4, 0.4)$ の場合では，多くの点で 1 以下の性能比になっている．これは，Ullmann のアルゴリズムでは，1 回目の refinement procedure を呼び出して探索空間削減を行うときに，対応可能な写像が存在しないので，次の探索木を巡回することを早い段階 (Step 1) でやめてしまう．これのため，Ullmann のアルゴリズム

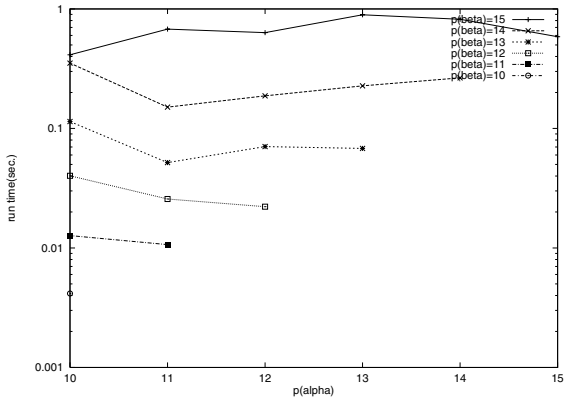
```
M := M^0; IM := M^0; d := 1; used := 1;
 [ここまでは初期設定]
while (1) {
   current := M_d & used; [1]
   while (current ≠ 0) {
     current(i) = 1 となる最初の i を設定; [全行と重畳で 1]
     p(d) := i; current(i) := 0; flag := 0; [全部で 1]
     “ 辺存在確認アルゴリズム ”を実行;
     if (返り値 = OK  [辺存在確認アルゴリズムと重畳]) {
       if (d = p_α [1]) 写像を記憶; [1]
       else {used(i) := 0; M_d := current; d := d + 1;
             flag := 1; break; [全部で 1] }
     }
   }
   if (flag) continue;
   if (d = 1 [1]) terminate algorithm; [1]
   M_d := IM_d; d := d − 1; [1]
   used(p(d)) := 1; [1]
}
```

図 12: 探索木巡回アルゴリズム

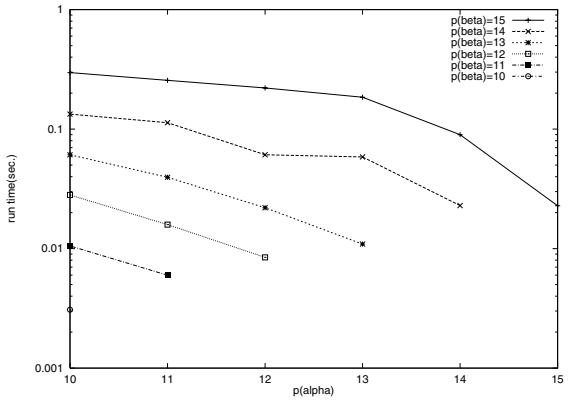
```
n := etsa(d); i := 0; a := eta(n); b := etb(n);
while (a ≠ 0 or b ≠ 0) { [1]
   if (B(p(a), p(b)) = 0[1]) return(NG); [1]
   i := i + 1; a := eta(n + i); b := etb(n + i); [全部で 1]
}
return (OK); [1]
```

図 13: 辺存在確認アルゴリズム

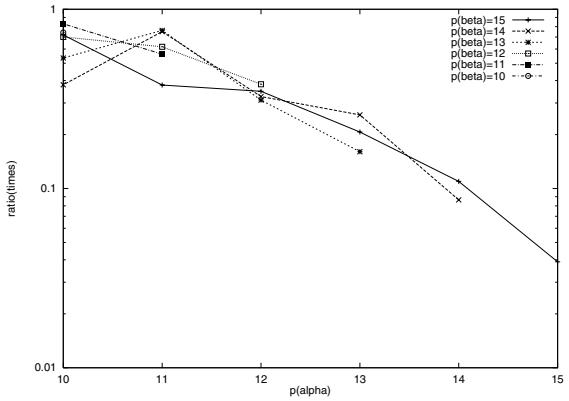
の実行時間が短くなっている．$(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合では，全ての点で Ullmann のアルゴリズムより性能が向上している．



(a) 小西のアルゴ リズムの実行時間



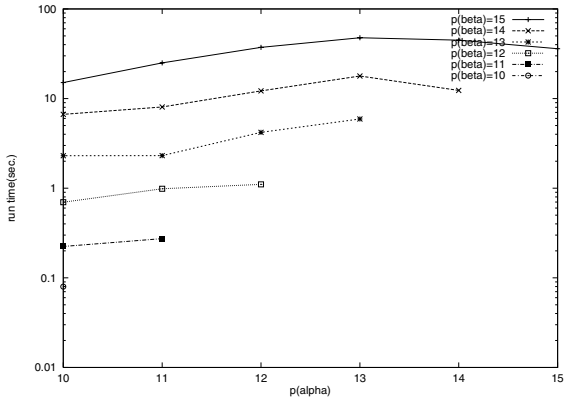
(b) Ullmann のアルゴ リズムの実行時間



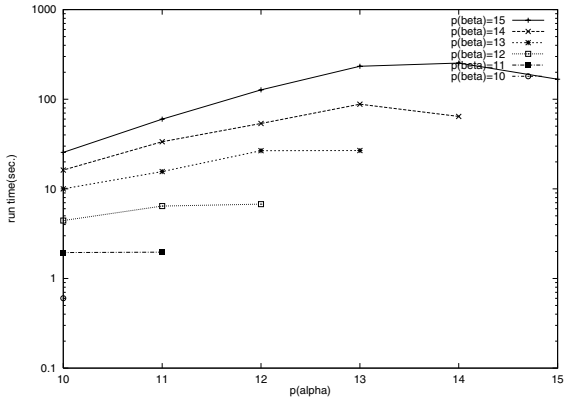
(c) 小西のアルゴ リズムと Ullmann のアルゴ リズムの実行時間比

図 14: $(ed_\alpha, ed_\beta) = (0.19811, 0.19758)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00280)$ の場合

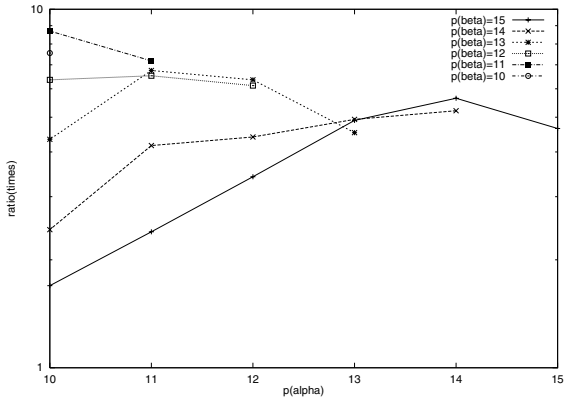
($G_\alpha \subset G_\beta$ の入力データ)



(a) 小西のアルゴ リズムの実行時間



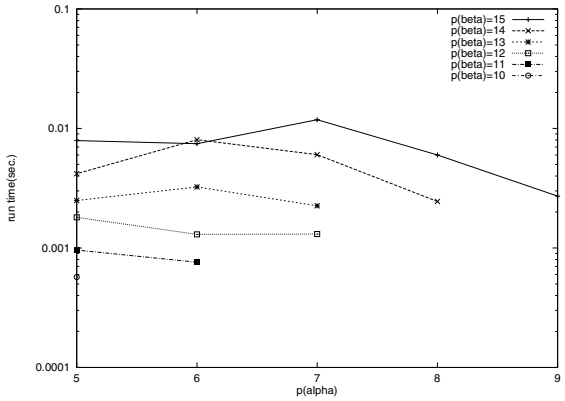
(b) Ullmann のアルゴ リズムの実行時間



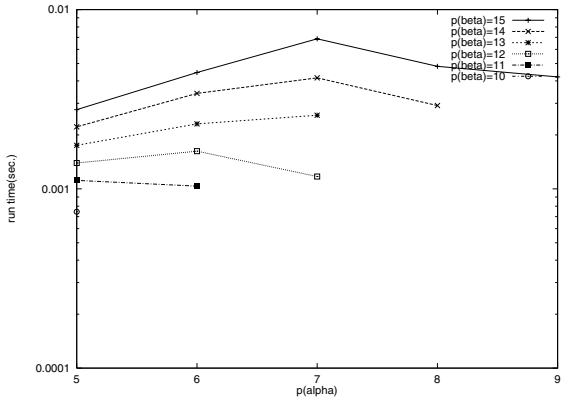
(c) 小西のアルゴ リズムと Ullmann のアルゴ リズムの実行時間比

図 15: $(ed_\alpha, ed_\beta) = (0.19811, 0.39760)$,$(s.d._\alpha, s.d._\beta) = (0.00267, 0.00231)$ の場合

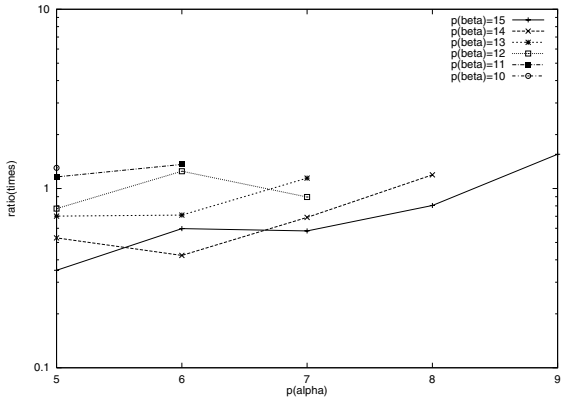
($G_\alpha \subset G_\beta$ の入力データ)



(a) 小西のアルゴ リズムの実行時間



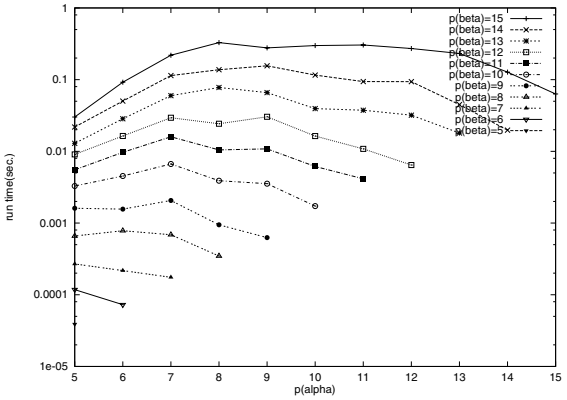
(b) Ullmann のアルゴ リズムの実行時間



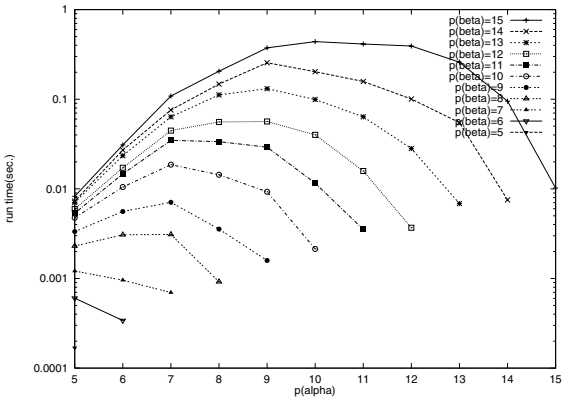
(c) 小西のアルゴ リズムと Ullmann のアルゴ リズムの実行時間比

図 16: $(ed_\alpha, ed_\beta) = (0.39436, 0.19772)$,$(s.d._\alpha, s.d._\beta) = (0.00783, 0.00270)$ の場合

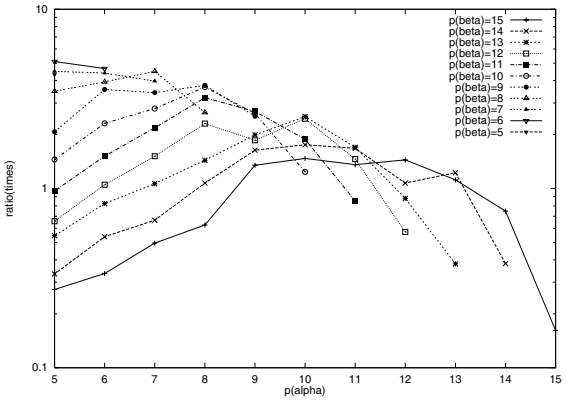
($G_\alpha \subset G_\beta$ の入力データ)



(a) 小西のアルゴ リズムの実行時間



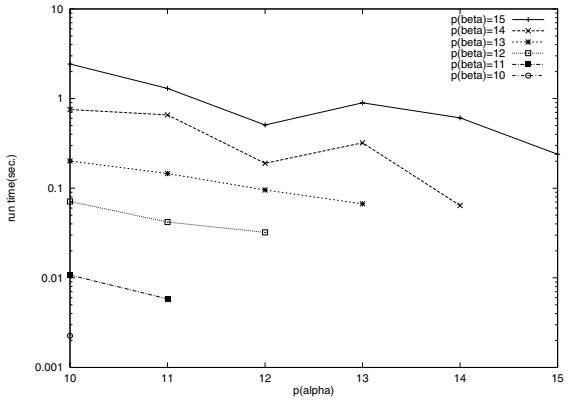
(b) Ullmann のアルゴ リズムの実行時間



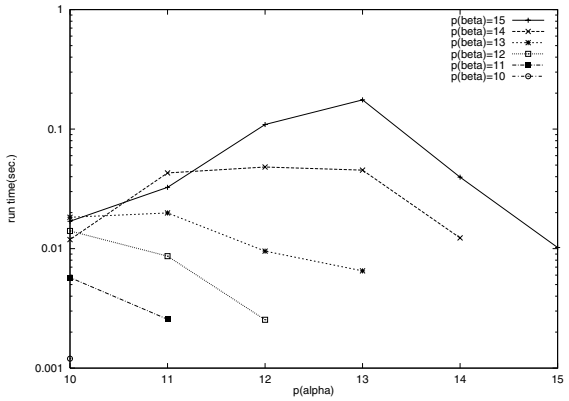
(c) 小西のアルゴ リズムと Ullmann のアルゴ リズムの実行時間比

図 17: $(ed_\alpha, ed_\beta) = (0.39466, 0.39611)$,$(s.d._\alpha, s.d._\beta) = (0.00665, 0.00470)$ の場合

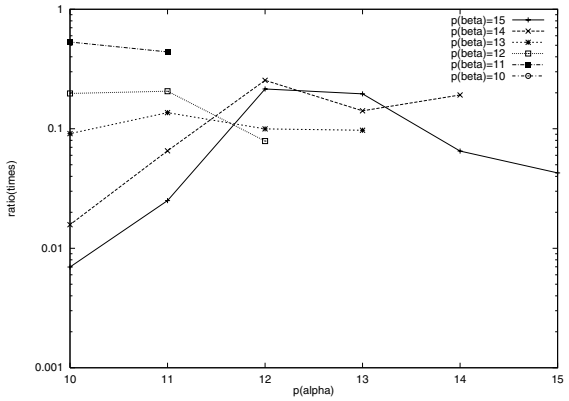
($G_\alpha \subset G_\beta$ の入力データ)



(a) 小西のアルゴ リズムの実行時間



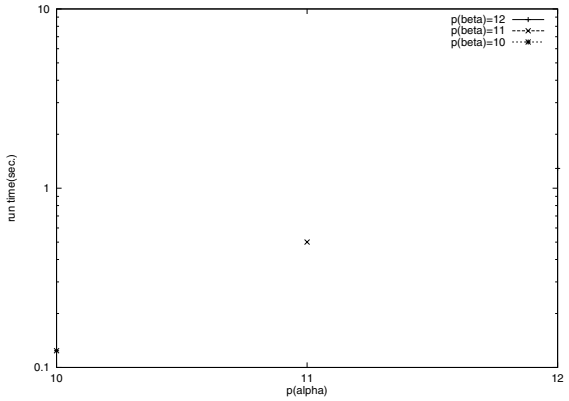
(b) Ullmann のアルゴ リズムの実行時間



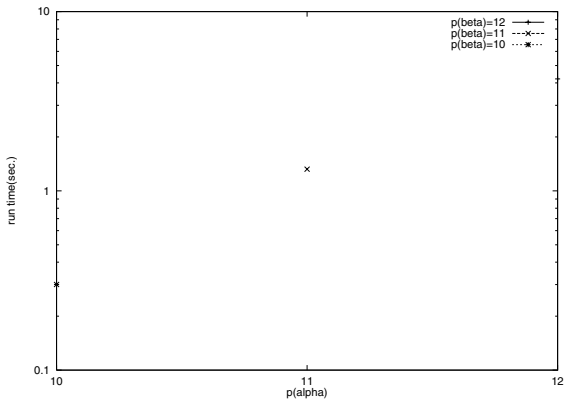
(c) 小西のアルゴ リズムと Ullmann のアルゴ リズムの実行時間比

図 18: $(ed_\alpha, ed_\beta) = (0.19811, 0.19758)$,$(s.d._\alpha, s.d._\beta) = (0.00267, 0.00280)$ の場合

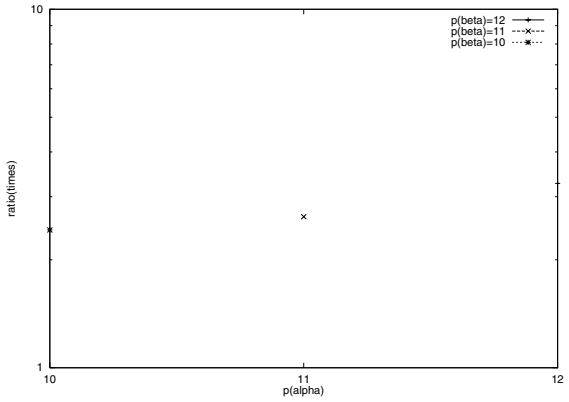
($G_\alpha \not\subset G_\beta$ の入力データ)



(a) 小西のアルゴ リズムの実行時間



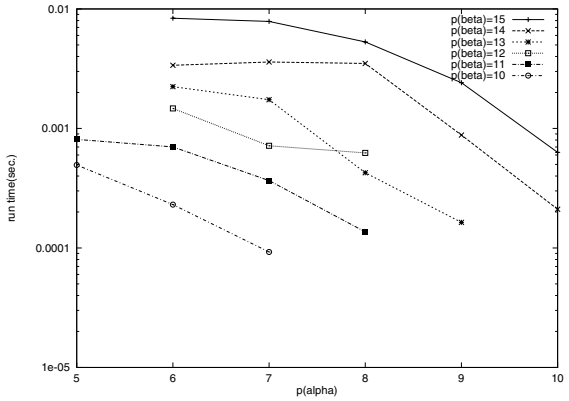
(b) Ullmann のアルゴ リズムの実行時間



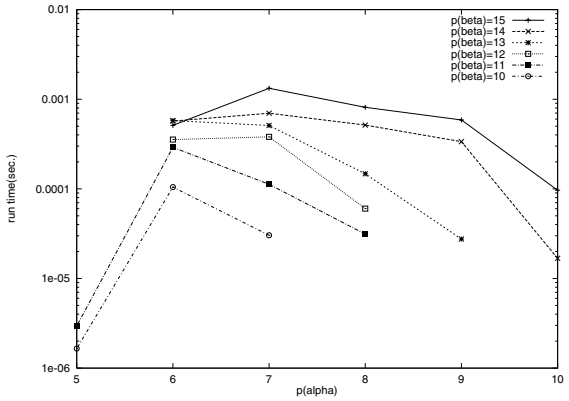
(c) 小西のアルゴ リズムと Ullmann のアルゴ リズムの実行時間比

図 19: $(ed_\alpha, ed_\beta) = (0.19899, 0.39798)$,$(s.d._\alpha, s.d._\beta) = (0.00143, 0.00286)$ の場合

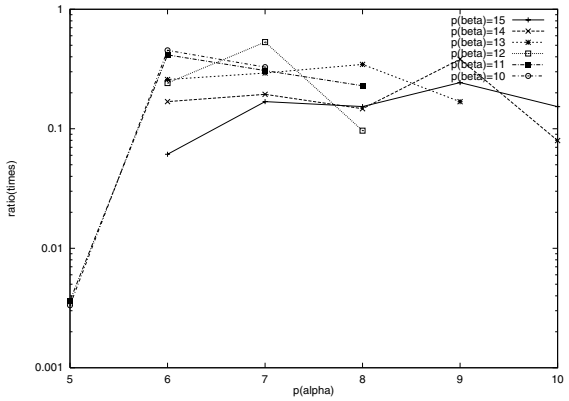
($G_\alpha \not\subset G_\beta$ の入力データ)



(a) 小西のアルゴ リズムの実行時間



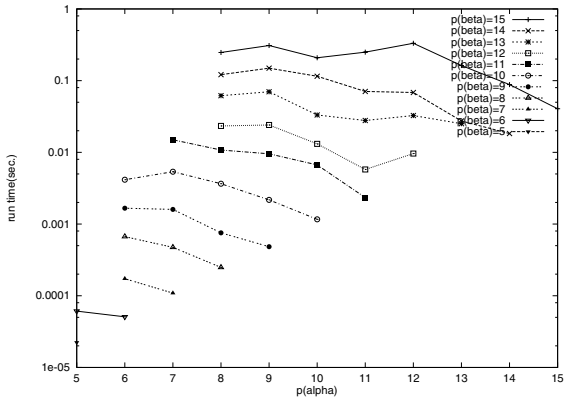
(b) Ullmann のアルゴ リズムの実行時間



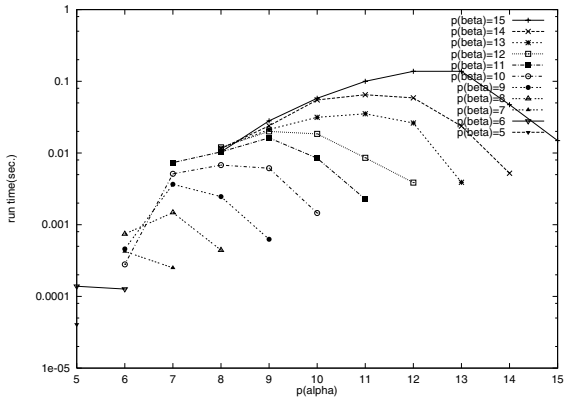
(c) 小西のアルゴ リズムと Ullmann のアルゴ リズムの実行時間比

図 20: $(ed_\alpha, ed_\beta) = (0.39236, 0.19788)$,$(s.d._\alpha, s.d._\beta) = (0.00764, 0.00274)$ の場合

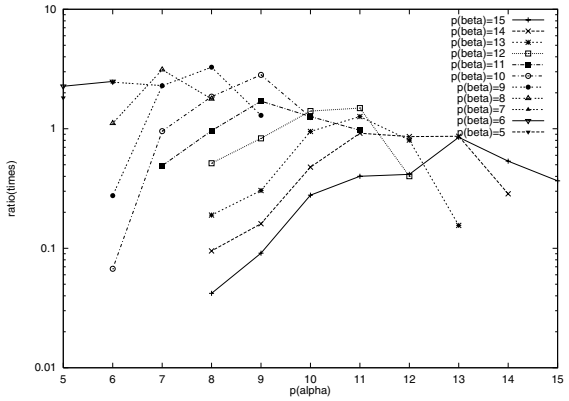
($G_\alpha \not\subset G_\beta$ の入力データ)



(a) 小西のアルゴ リズムの実行時間



(b) Ullmann のアルゴ リズムの実行時間



(c) 小西のアルゴ リズムと Ullmann のアルゴ リズムの実行時間比

図 21: $(ed_\alpha, ed_\beta) = (0.39436, 0.39624)$,$(s.d._\alpha, s.d._\beta) = (0.00606, 0.00468)$ の場合

($G_\alpha \not\subset G_\beta$ の入力データ)

4　小西のアルゴリズムの FPGA による実装

4.1　設計

この節では，小西のアルゴリズムをハードウェア化するための設計について説明する．

4.1.1　実装環境

小西のアルゴリズムを実装する環境は，Lucent 社の ORCxxA シリーズ FPGA[5] を搭載する OPERL ボード [4] である．OPERL ボードは 2 つの FPGA から構成されており，一つは PCI FPGA, もう一つは USER FPGA である (図 22)．PCI FPGA は，PCI バスと USER FPGA との間でデータを受け渡す中間回路であり，SROM により初期化設定が行われる．USER FPGA は，動的再構成できる部分である．この部分には，ユーザが設計した回路の構成データ (bit stream) をダウンロードできる．また，USER FPGA はソケットで装着されているため，FPGA の容量を任意に変更する可能である．

本研究では，Lucent 社の OR2C15A FPGA を USER FPGA として使用する．OR2C15A の使用可能なゲート数は 19,200〜44,200 である [5]．OR2C シリーズ FPGA のリソース単位は PFU(Programable Function Unit) であり，OR2C15A では最大 400PFU を利用することができる．

OR2C15A を搭載する OPERL ボードは，ホストコンピュータの PCI スロットに指している状態になっている．ホストコンピュータの仕様を表 12 に示す．ホストコンピュータから PCI バス経由で OPERL ボードとデータを送受する．今回実装する回路ではデータ転送量が非常に少ないため (付録 A を参照のこと)，直接入出力ポートに読み書きすることにする．

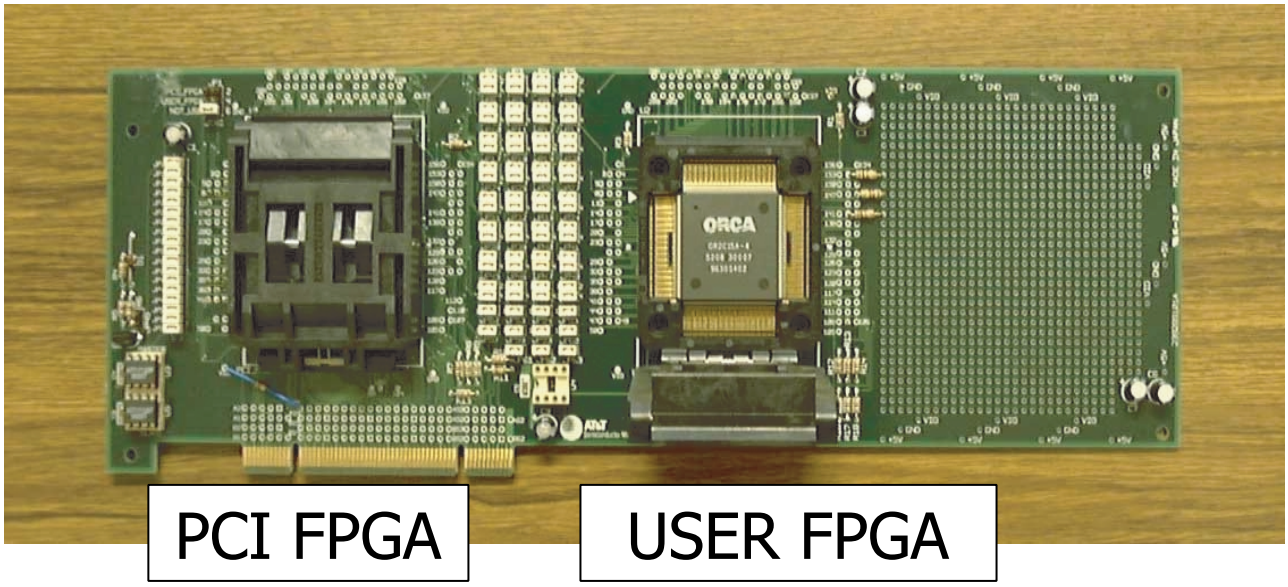


図 22: OPERL ボード [4]

表 12: ホストコンピュータ

プロセッサ	AMD K6-III 400 MHz
チップセット	Intel 430HX
２次キャッシュ	256KB(K6-III に内蔵)
主記憶	DRAM 64MB
基本ソフトウェア	FreeBSD-2.2.1
C 言語コンパイラ	gcc ver.2.7.2

4.1.2 回路の構成

図 23 は，USER FPGA のブロック図である．図中でそれぞれのブロックを以下に説明する．

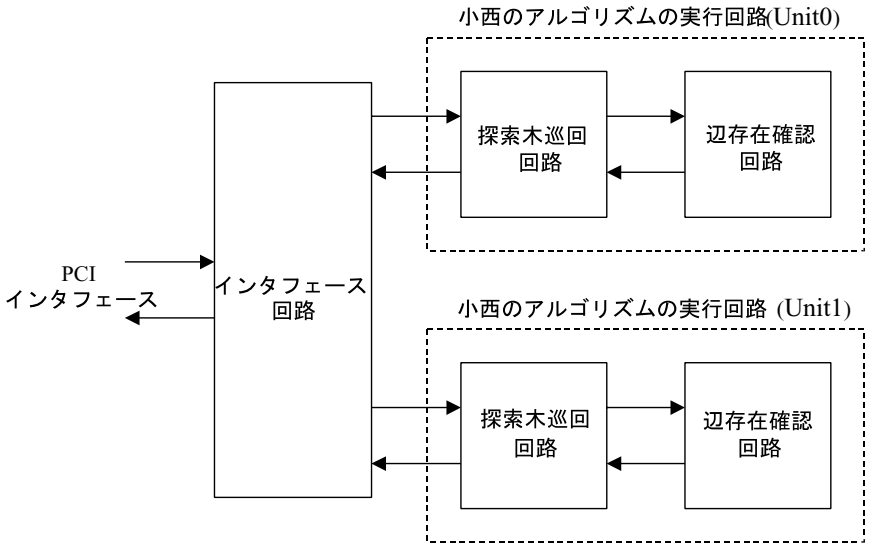


図 23: 小西のアルゴリズムの実行回路のブロック図

- 小西のアルゴリズムの実行回路

  本研究では，小西のアルゴリズムをハードウェア化し，その回路の名称を「小西のアルゴリズムの実行回路」とする．小西のアルゴリズムの実行回路は，2 つの回路から構成されている．一つは「探索木巡回回路」であり，もう一つは「辺存在確認回路」である．

  図中で Unit0, Unit1 はそれぞれ小西のアルゴリズムの実行回路であり，独立に部分グラフ同型判定を実行可能になっている．2 ユニットを並行に動作させると，2 倍のスループットが期待できる．

- 探索木巡回回路

  探索木巡回アルゴリズムをハードウェア化し，その回路の名称を「探索木巡回回路」とする．探索木巡回回路は，グラフ G_α から別のグラフ G_β への写像可能な頂点の組合せを列挙する．

- 辺存在確認回路

  辺存在確認アルゴリズムをハードウェア化し，その回路の名称を「辺存在確認回路」とする．辺存在確認回路は，G_α(検査対象となる) 辺の両端点を G_β に写像し，G_β の 2 頂点間に辺があるかどうか確認する．

- インタフェース回路

  ホスト側の PCI インタフェースと各ユニットとの間で入出力のデータを受け渡すものである．このインタフェース回路を経由で，アドレスをデコードし，該当なユニットに対して制御，初期化，状態監視を行うことが可能である．インタフェース回路についての設計と実装方法についての詳細は付録 A を参照．

4.1.3 回路への入出力データ

- 入力データ

  各ユニットへの入力データは，以下のものである．

  – グラフ G_α の隣接行列 A

  – グラフ G_α の頂点数 p_α

- グラフ G_β の隣接行列 B
- 配列 M,IM
- Edge list table (eta,etb)
- Edge list table starting address (etsa)

- 出力データ

  各ユニットからの出力データは，以下のものである．

  - 回路の実行終了
  - 同型である部分グラフを発見したこと
  - 同型である部分グラフを発見した数
  - 同型である部分グラフを発見した数のキャリーアウト

4.1.4 回路の実行可能な範囲

今回実装する回路は，G_α, G_β ともに頂点数 15 までのグラフを対象とする．つまり $(p_\alpha, p_\beta) = (15, 15)$ までを扱える回路である．その理由は以下の通りである．

小西のアルゴリズムでは，探索削減を行うために，深さごとに“ 調べる必要のある ”辺の 2 頂点番号を予めテーブルに格納する必要がある．さらに，深さが変わるときには識別するために (0,0) を挿入することになっている．このため，初期化を行う際に，メモリ IM,M,B の 0 番地が使用できなくなる．つまり，4 ビットのアドレスを持つメモリの場合では，0 番地を予約として，1 番地から 15 番地までの 15 個の頂点データを格納するしかできない．

また，本研究では，VHDL を記述する際に，Lucent 社の OR2CxxA FPGA 用ライブラリを使用する．このライブラリで使用できる RAM マクロなどは基本的に 4 ビットの単位である．従って，FPGA のリソースを考慮する実装を行うために 4 ビット単位の RAM を構成していくことが最も効果的で，4 ビット以上の構成になると実装効率の低下が見込まれる．

4.2 実装方法

この節では，辺存在確認回路，探索木巡回回路の概要図を示し，アルゴリズムとの対応，回路の動作について説明する．

4.2.1 辺存在確認回路

辺存在確認回路のデータパスを図 24 に示す．図中のやや太い矢印はアドレスやデータパス，細い線は制御線を表す．

以下に辺存在確認アルゴリズムと設計した回路との対応を説明する．また，これからについての説明では，p_α はグラフ G_α の頂点数，q_α はグラフ G_α の辺数，p_β はグラフ G_β の頂点数を表す．

- メモリ ETS(A),ETS(B)

  図 4 の Edge list Table のデータの第 1 項目 (data1) と第 2 項目 (data2) に相当する．ETS(A),ETS(B) のアドレスのビット幅は $log(p_\alpha + q_\alpha)$ であり，入出力のデータのビット数は $\log(p_\beta)$ である．これを実装するには，トライ・ステートの出力を持つ 16x4 ビット RAM を用いる．このトライ・ステートの出力を持つ RAM を用いることによって，ETS(A),ETS(B) を構成するために使われる MUX の数が削減される．その上，論理段数を減るため，配置配線を行う際によりよい結果が得られることを期待できる．ETS(A) の回路構成を図 25 に示す．ETS(B) も ETS(A) と全く同じ構造を持っている．

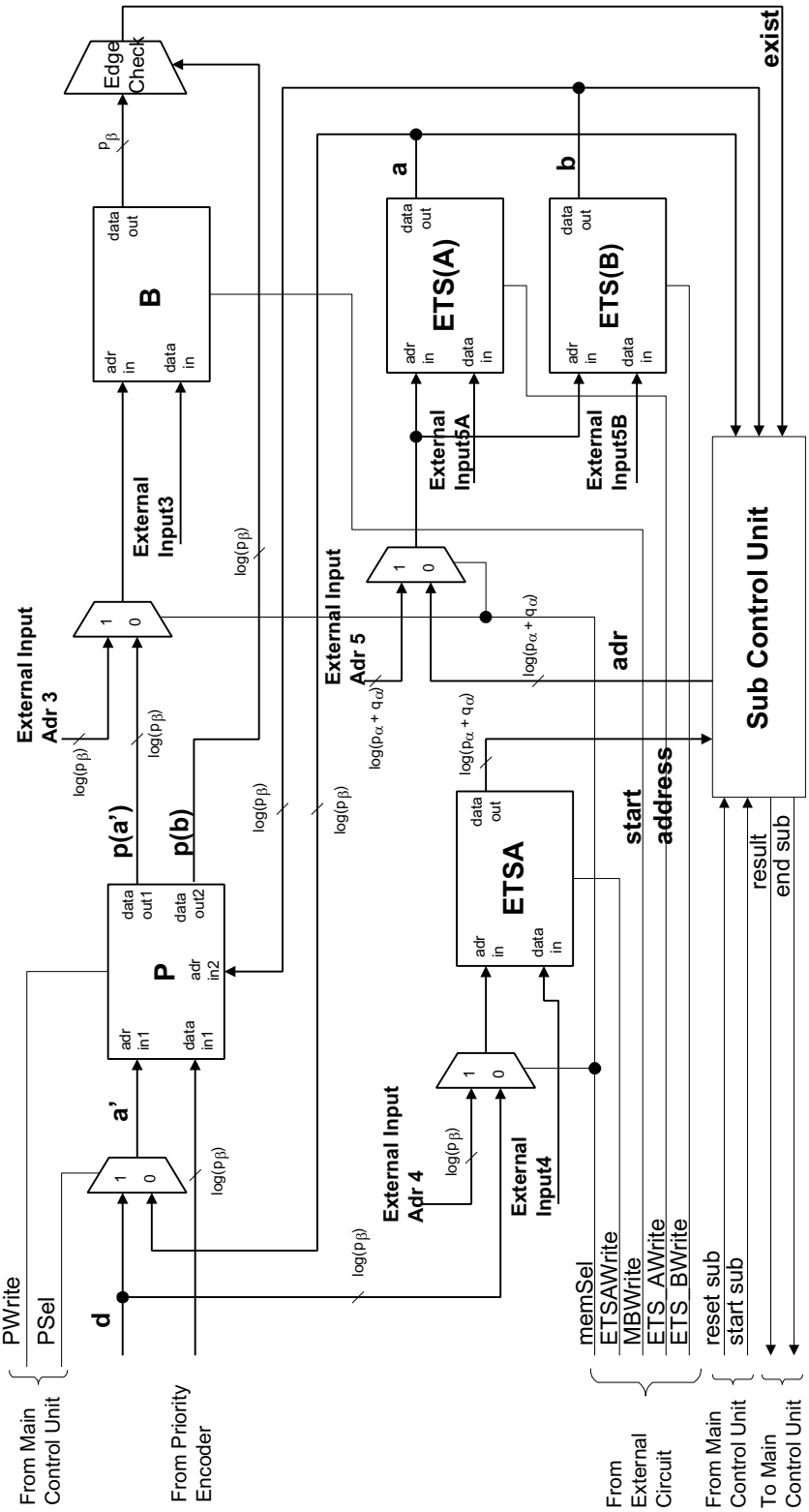


図 24: 辺存在確認回路のデータパス

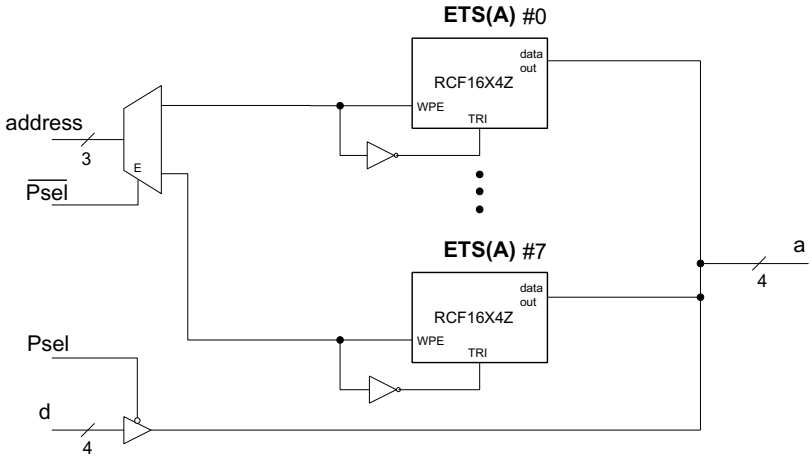


図 25: メモリ ETS(A) の回路

- メモリ ETSA

  図 4 の Edge list table starting address に相当する．ETSA のアドレスのビット幅は，$\log(p_\beta)$ であり．入出力のデータのビット数は，$\log(p_\alpha + q_\alpha)$ である．これを実装するには，16x4 ビット RAM を用いる．

- メモリ P

  探索木巡回回路の P(図 7)，図 4 に描かれている $vb(i)(1 \leq i \leq p_\alpha)$ の信号線とマルチプレクサ (MUX) に相当する．P のアドレスのビット幅は，$\log(p_\beta)$ であり．入出力のデータのビット数は，$\log(p_\beta)$ である．これを実装するには，デュアルポート・メモリを用いる．デュアルポート・メモリは，データを格納する時に一般のメモリの動作と同じである．しかし，データを呼び出す時には，2 つのアドレスを同期または非同期に指定してそれぞれのアドレスに対応するデータを取ることが可能である．

  本研究では，このデュアルポート・メモリの特長を生かして，リソース消費量を節約することができる．論文 [6] の方法で実現するには，4 ビットのレジスタが $p_\alpha \cdot \lceil \frac{\log p_\alpha}{4} \rceil$ 個，4 入力マルチプレクサが $2 \cdot \log(p_\alpha) \cdot \sum_{i=1}^{\log_4 p_\alpha} \lfloor \frac{p_\alpha}{4^i} \rfloor$ 個，2 入力マルチプレクサが $2 \cdot \log(p_\alpha) \cdot (\lceil \frac{\log p_\alpha}{2} \rceil - \lfloor \frac{\log p_\alpha}{2} \rfloor)$ 個が必要となっている．例えば，これについて Lucent 社の OR2CxxA のマクロライブラリ [8] を用いて実装した場合，4 ビットのレジスタ (RD4P3D) の面積が 0.5，4 入力マルチプレクサ (MUX41) の面積が 0.25，2 入力マルチプレクサ (MUX21) の面積が 0.125 であるので，$p_\alpha = 15$ のときのリソース消費量は 18 個の PFU が必要となる．一方，本研究で使用する 16x2 ビットデュアルポート・メモリ (DCE16X2) の面積が 1 なので，$p_\alpha = 15$ のときのリソース消費量は 2 個の PFU が必要となっている．

- メモリ B と Edge Check

  図 5 の辺導出部に相当する．メモリ B のアドレス幅は，$\log(p_\beta)$，入力データのビット数は $\log(p_\beta)$，出力データのビット数は p_β である．Edge Check の入力データビット数は p_β, コントロールのデータビット数は $\log(p_\beta)$，出力データのビット数は 1 である．メモリ B の部分は 16x4 ビット RAM で実装できる．Edge Check の部分は G_β の 2 頂点の間に辺があるか確認する (動作記述で実装した)．

  本研究では，必要なハードウェア資源量，論理段数の削減を考慮した実装を行うために，メモリ B と Edge Check を統合することにした．統合した回路を図 26 に示す．メモリ B の初期値を入力する際に次のように行う．入力データ 1 ビット目をメモリ B のデータイン 15 ビット目に，入力データ 2 ビット目をメモリ B のデータイン 14 ビット目に，$\cdots$, 入力データ 15 ビット目をメモリ B のデータイン 1 ビット目につなぐ．但し，入力データ 0 ビット目はメモリ B のデータイン 0 ビット目につながるように実装する．

- Sub Control Unit

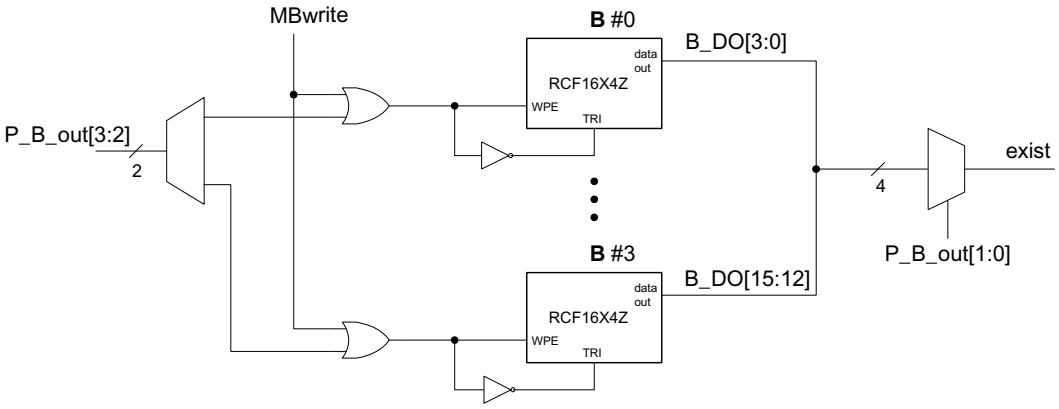


図 26: メモリ B と Edge Check を統合した回路

辺存在確認回路のコントロールユニットである．このコントロールユニットは，ムーア型ステート・マシンを用いて実現したものである．回路の状態遷移図を図 27 に示す．この回路への入出力と各状態における動作は以下の通りである．

– 入力信号

  * reset sub

    sub control unit の状態をリセットする信号線である．この信号線は探索木巡回回路 (4.2.2 節を参照のこと) のメインコントロールユニットから出た制御信号線の一つである．この reset sub が有効 (つまり 1) であれば，sub control unit がどの状態を実行中でも，必ず初期状態に戻る．

  * start sub

    辺存在確認回路の実行をスタートさせる信号線である．この信号線は探索木巡回回路 (4.2.2 節を参照のこと) のメインコントロールユニットから出た制御信号線の一つである．初期状態になっている時，この start sub が有効であれば，辺存在確認回路の実行が始まる．

  * start address

    Edge list table の開始アドレスである．この信号は，メモリ ETSA の出力された $\log(p_\alpha + q_\alpha)$ ビットのデータである．

  * exist

    辺導出部から出力された $\{v_{\beta a}, v_{\beta b}\}$ の辺が存在しているかどうかの調べた結果の信号線である．この信号線は，Edge Check からの出力されたものである．

– 出力信号

  * adr

    Edge list table のアドレスである．この信号は，start address を 1 にインクリメントしたものかまたはインクリメントしないものである．adr のビット数は，$\log(p_\alpha + q_\alpha)$ である．

  * result

    辺存在確認回路の実行結果である．result のビット数は 1 である．result が OK(つまり 1) である時に，深さ d で $v_{\alpha d}$ と隣接している辺 e_α の写像は，G_β の中に対応している辺 e_β が存在しているという意味である．一方，result が NG(つまり 0) である時に，G_β の中に対応している辺 e_β が存在していないという意味である．

  * end sub

    辺存在確認回路の実行が終了したことを示す信号である．end sub のビット数が 1 である．end sub が有効であれば，辺存在確認回路の実行が終了したという意味である．

– 各状態における動作

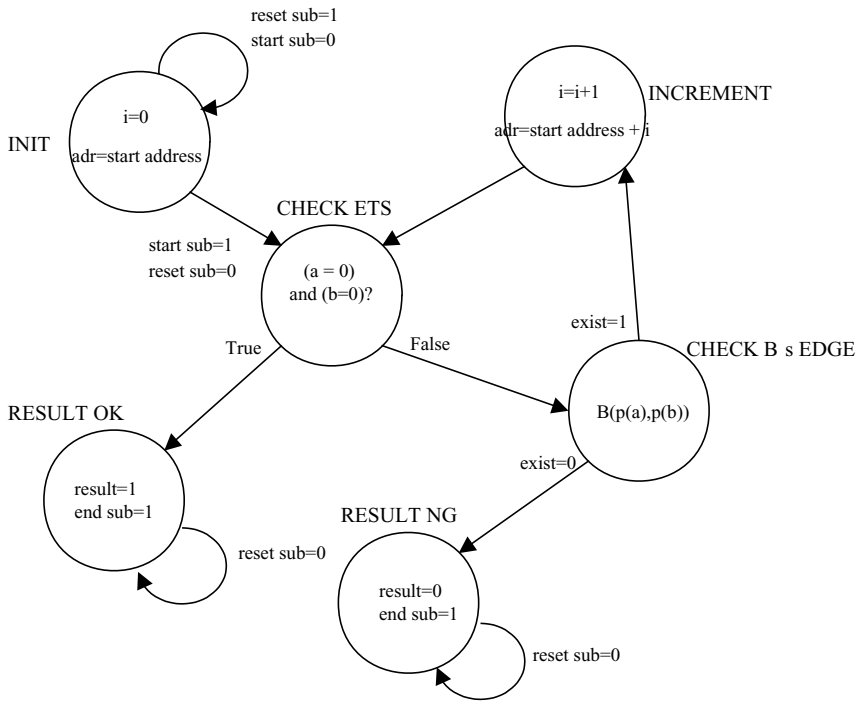


図 27: 辺存在確認回路の状態遷移図

* INIT
  辺存在確認回路の初期状態である．この初期状態ではインクリメント用の変数 i を 0 にセットすることと，adr の値を start address に設定することである．この状態は，start sub 信号が有効である時意外，この状態をそのままに保つことになっている．
* CHECK ETS
  Edge list table に格納されているデータの第 1 項目 (data1) と第 2 項目 (data2) が 0 であるかどうかを調べるための状態である．調べた結果が真であれば，RESULT OK に移る．偽であれば，CHECK B's EDGE に移る．
* CHECK B's EDGE
  CHECK EDGE から出力信号 exist が有効であるかどうかを調べる状態である．exist が有効であれば，INCREMENT に移る．有効でなければ，RESULT NG に移る．
* INCREMENT
  次の data1 と data2 の値を用意する状態である．そのために，i 変数を 1 にインクリメントして，次のアドレスとなる adr を start address + i に設定する．この状態が終了すると，CHECK ETS に移る．
* RESULT OK
  result 信号と end sub 信号を有効にする状態である．この状態では，reset sub 信号が有効である時意外，この状態をそのままに保つことになっている．
* RESULT NG
  result 信号を無効に，end sub 信号を有効にする状態である．この状態では，reset sub 信号が有効である時意外，この状態をそのままに保つことになっている．

4.2.2 探索木巡回回路

探索木巡回回路を図 28 に示す．図中のやや太い矢印はアドレスやデータパス，細い線は制御線を表す．以下に探索木巡回アルゴリズムと設計した回路の対応を説明する．

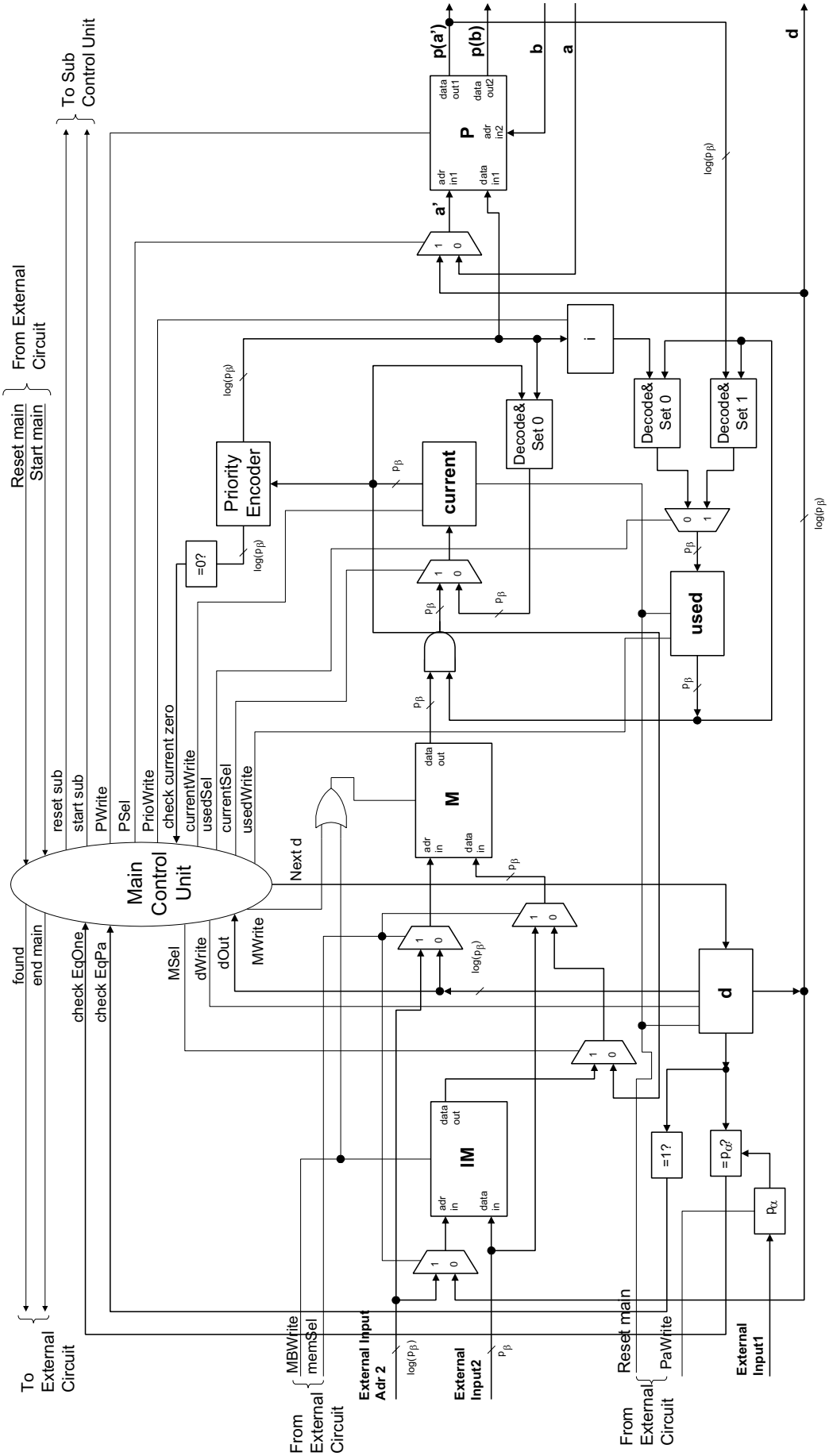


図 28: 探索木巡回回路のデータパス

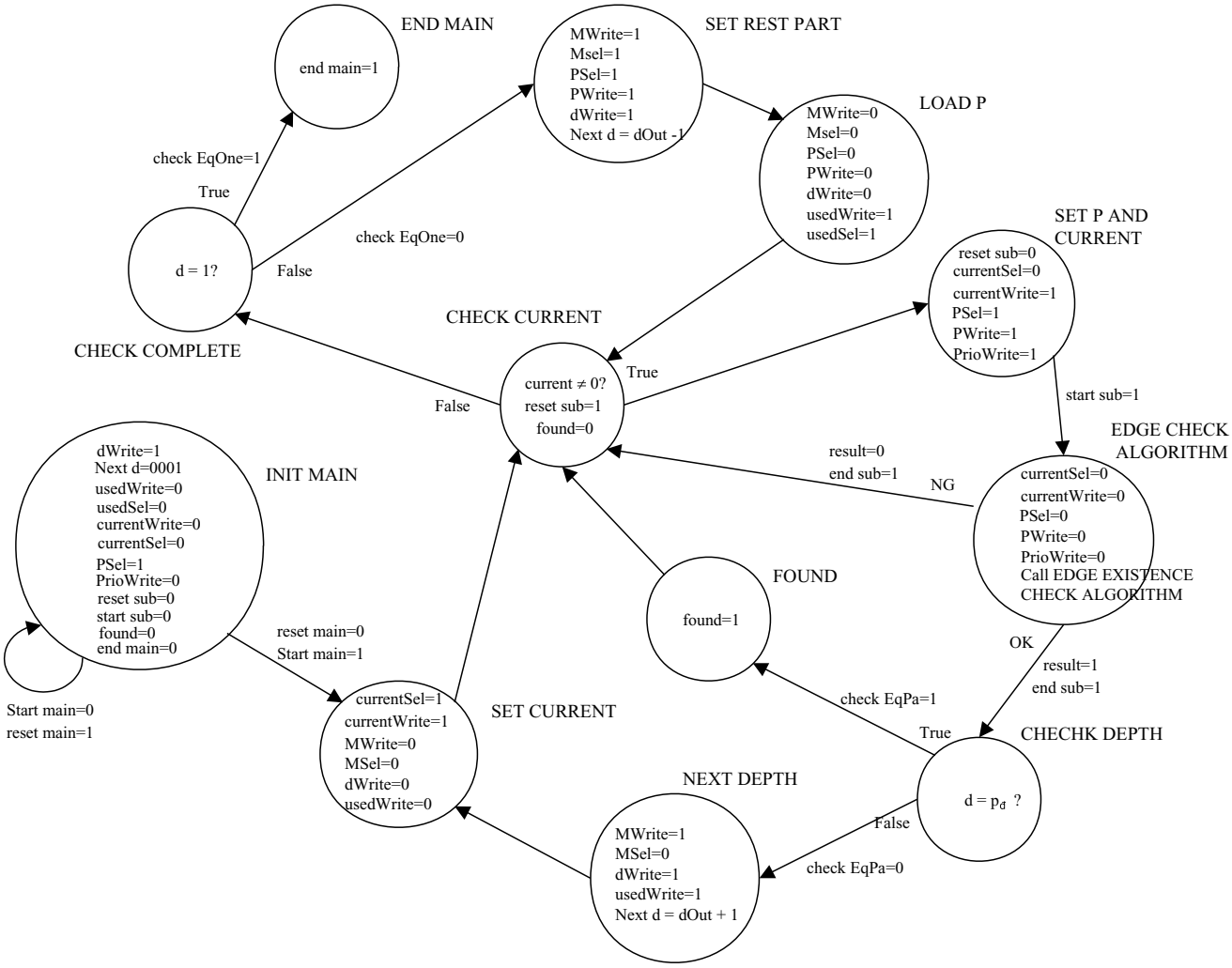


図 29: 探索木巡回回路のデータパスを制御する有限状態機械の全体図

- メモリ M と IM

  図 7 の M と IM に相当する．メモリ M と IM のアドレスのビット数は $\log(p_\beta)$ であり，入出力のデータのビット数は $\log(p_\beta)$ である．これを実装するには，16x4 ビット RAM を用いる．

- レジスタ current

  図 7 の current に相当する．レジスタ current の入出力のビット数は p_β である．レジスタ current の初期値は，すべてのビットが 0 である．これを実装するには，クリア付き 4 ビットレジスタを用いる．

- レジスタ used

  図 7 の used に相当する．レジスタ used の入出力のビット数は p_β である．レジスタ used の初期値は，すべてのビットが 1 である．これを実装するには，プリセット付き 4 ビットレジスタを用いる．

- レジスタ d

  図 7 の d に相当する．レジスタ d の入出力のビット数は $\log(p_\alpha)$ である．レジスタ d の初期値は，すべてのビットが 0 である．これを実装するには，クリア付き 4 ビットレジスタを用いる．

- レジスタ p_α

  p_α の値を格納するためのレジスタである．レジスタ p_α の入出力のビット数は $\log(p_\alpha)$ である．レジスタ p_α の初期値は，すべてのビットが 0 である．これを実装するには，クリア付き 4 ビットレジスタを用いる．

- Priority Encoder

  レジスタ current の出力を，G_α から G_β へ写像可能な頂点番号にデコードする．また図 7 の Priority encoder に相当する．Priority encoder の入力ビット数は p_β であり，出力のビット数は $\log(p_\beta)$ である．この分部は動作記述で実装された．

- レジスタ i

  Priority Encoder の値を格納するためのレジスタである．Priority Encoder から出力されたデータは，レジスタ current の出力が変わるたびに更新されるので，現在の調べている頂点を記憶する必要がある．レジスタ i の入出力のビット数は $\log(p_\beta)$ である．レジスタ i の初期値は，すべてのビットが 0 である．これを実装するには，クリア付き 4 ビットレジスタを用いる．

- Decode & Set 0

  p_β ビットの入力に n ビット目を 0 にするためのデコーダである．このデコーダは 2 つの入力データと 1 つの出力データからなるものである．2 つの入力データのうちの一つ目は input1，もう一つは input2 とする．input1 のビット数は p_β，input2 のビット数は $\log(p_\beta)$ である．このデコーダの動作では，input1 が p_β ビットの入力とし，input2 が n ビット目とする．出力データのビット数は p_β である．例えば，input1 は $(01010\cdots)_2$, input2 は $(0010)_2$ とする．このデコーダの出力結果は，$(00010\cdots)_2$ になる．このよう回路は動作記述で実装するのが効果的である．

- Decode & Set 1

  p_β ビットの入力に n ビット目を 1 にするためのデコーダである．このデコーダは 2 つの入力データと 1 つの出力データからなるものである．2 つの入力データのうちの一つ目は input1，もう一つは input2 とする．input1 のビット数は p_β，input2 のビット数は $\log(p_\beta)$ である．このデコーダの動作では，input1 が p_β ビットの入力とし，input2 が n ビット目とする．出力データのビット数は p_β である．例えば，input1 は $(01010\cdots)_2$, input2 は $(0001)_2$ とする．このデコーダの出力結果は，$(11010\cdots)_2$ になる．このよう回路は動作記述で実装するのが効果的である．

- =1?

  レジスタ d(深さ d) が 1 になったかどうかを調べるための比較器である．この比較器の入力のビット数は $\log(p_\alpha)$，出力のビット数は 1 である．もしレジスタ d が 1 であれば，1 を出力する．そうでなければ，0 を出力する．これを実装するには，4 ビットの比較器を用いる．

- $=p_\alpha$?

  レジスタ d(深さ d) が p_α になったかどうかを調べるための比較器である．この比較器の入力のビット数は $\log(p_\alpha)$，出力のビット数は 1 である．もしレジスタ d が p_α であれば，1 を出力する．そうでなければ，0 を出力する．これを実装するには，4 ビットの比較器を用いる．

- Main Control Unit

  探索木巡回回路のコントロールユニットである．このメインコントロールユニットは，ムーア型ステート・マシンを用いて実現したものである．回路の状態遷移図を図 29 に示す．この回路へ入出力と各状態における動作は以下の通りである．

  - 入力信号

    * Reset main

      探索木巡回回路をリセットする信号線である．この信号線はインタフェース回路 (付録 A を参照のこと) から出力された制御信号線の一つである．Reset main のビット数は 1 である．この Reset main が有効であれば，Main Control Unit がどの状態を実行中でも，必ず初期状態に戻る．その同時に，全てのレジスタ (d, current, used,i) にもリセットされる．

* Start main
  探索木巡回回路の実行をスタートさせる信号線である．この信号線はインタフェース回路 (付録 A を参照のこと) から出力された制御信号線の一つである．Reset main のビット数は 1 である．初期状態になっている時，この Start main が有効であれば，探索木巡回回路の実行が始まる．
* check EqOne
  現在実行している深さ d が 1 になっているかどうかを示す信号線である．この信号線は，=1?の比較器からの出力信号である．
* check EqPa
  現在実行している深さ d が p_α になっているかどうかを示す信号線である．この信号線は，$=p_\alpha$?の比較器からの出力信号である．
* check current zero
  深さ d で $v_{\beta j}$ へ写像できる頂点が存在しているかどうかを判定した結果の信号線である．この信号線は，=0?の比較器からの出力信号である．
* dOut
  現在実行中している深さ d がどこまでであるかを示す信号線である．dOut のビット数は $\log(p_\alpha)$ である．

– 出力信号

* reset sub
  辺存在確認回路をリセットする信号である．この信号は，辺存在確認回路の実行をスタートさせる前に，必ず有効になる．
* start sub
  辺存在確認回路の実行をスタートさせる信号である．
* PSel
  メモリ P への入力アドレスを選択するための信号である．PSel のビット数は 1 である．この信号が 1 であれば，レジスタ d の出力データを入力アドレスとして選択する．0 であれは，Edge list table のデータ第 1 項目を入力アドレスとして選択する．
* PWrite
  メモリ P の書き込みイネーブルである．PWrite が 1 である時に，メモリ P に入力データが書き込まれる．
* PrioWrite
  レジスタ i の書き込みイネーブルである．PrioWrite が 1 である時に，レジスタ i に Priority Encoder の値が書き込まれる．
* currentSel
  レジスタ current への入力データを選択する信号である．入力選択信号 currentSel が 1 ならば論理 AND の出力結果 (M_d & $used$ の結果) を，また 0 ならば Decode & Set0 の出力結果 ($current(i)$:=0 の結果) を選択する (図 6 を参照).
* currentWrite
  レジスタ current の書き込みイネーブルである．currentWrite が 1 である時に，マルチプレクサからの出力データが書き込まれる．
* usedSel
  レジスタ used への入力データを選択する信号である．入力選択信号 usedSel が 1 ならば Decode & Set1 の出力結果 ($used(p(d))$:=1 の結果) を，また 0 ならば Decode & Set0 の出力結果 ($used(i)$:=0 の結果) を選択する (図 6 を参照).
* usedWrite
  レジスタ used の書き込みイネーブルである．usedWrite が 1 である時に，マルチプレクサからの出力結果が書き込まれる．

* Msel

  メモリ M への入力データを選択する信号である．入力選択信号 Msel が 1 ならば IM の出力結果 (M_d:=IM_d の結果) を，また 0 ならば current の出力結果 (M_d:=$current$ の結果) を選択する (図 6 を参照)．

* MWrite

  メモリ M への入力データを書き込めるイネーブル信号である．MWrite が 1 である時に，マルチプレクサからの出力結果が書き込まれる．

* Next d

  次に調べる深さ d である．Next d は，dOut の値を 1 にインクリメントする，または 1 にディクリメントすることによって生成された値である．

* dWrite

  レジスタ d の書き込みイネーブルである．dWrite が 1 である時に，Next d のデータが書き込まれる．

* found

  部分グラフ同型が発見されたことを表す信号である．found のビット数は 1 である．

* end main

  探索木巡回回路の実行が終了したことを表す信号である．end main のビット数は 1 である．

– 各状態における動作

* INIT MAIN

  探索木巡回回路の初期状態である．Start main=0 であれば，この状態をそのままに保持する．Start main=1 かつ Reset main=0 であれば，SET CURRENT に移る．この初期状態では以下のことを行っている．

  · レジスタ d に深さ 1 を格納する (dWrite=1, Next d= 0x0001).

  · Decode & Set0 の出力結果 ($used(i)$:=0 の結果) をレジスタ used への入力データとして選択しておく (usedSel=0)．また，レジスタ used に書き込めない状態にする (usedWrite=0).

  · Decode & Set0 の出力結果 ($current(i)$:=0 の結果) をレジスタ current への入力データとして選択しておく (currentSel=0)．また，レジスタ current に書き込めない状態にする (currentWrite=0).

  · レジスタ d からの出力結果をメモリ P への入力アドレスとして選択する (PSel=1).

  · レジスタ i に書き込めない状態にする (PrioWrite=0).

  · 辺存在確認回路をリセットする (reset sub=0)．また，この回路の計算を始めないようにする (start sub=0).

  · 部分グラフ同型が発見されたことを表す信号を初期化する (found=0).

  · 探索木巡回回路の実行が終了したことを表す信号を初期化する (end main=0).

* SET CURRENT

  レジスタ current に論理 AND の出力結果 (M_d & $used$ の結果) を格納する状態である．この時の動作としては，論理 AND の出力結果を受けるために currentSel を 1，書き込みイネーブル currentWrite を 1 にセットする．なお，メモリ M のへ入力データの選択 MSel を 0，メモリ M の書き込みイネーブルを 0，レジスタ d の書き込みイネーブル dWrite を 0，レジスタ used の書き込みイネーブルを 0 にセットする．そして，次のステップの SET CURRENT に移る．

* CHECK CURRENT

  レジスタ current に格納されているデータが 0 であるかどうかを判定する状態である．この状態では，辺存在確認回路をリセットし (reset sub=1)，発見数を示す信号 found をクリアする (found = 0). そして，=0?の比較器から出力された check current zero が 0 であれば，CHECK COMPPLETE に移る．1 であれば，SET P AND CURRENT に移る．

* SET P AND CURRENT

深さ d で $v_{\alpha d}$ の写像先頂点番号 i をメモリ P に格納することと，レジスタ current の i 番目のビットを 0 にする状態である．頂点番号 i をメモリ P に格納するために，入力選択 Psel を 1，書き込みイネーブル Pwrite を 1 にセットする．レジスタ current の i 番目のビットを 0 にするために，Decode & Set 0 の出力結果をレジスタ current の入力として選択する．また，頂点番号 i をレジスタ i に格納する (PrioWrite=1)．そして，EDGE CHECK ALGORITHM に移る．

* EDGE CHECK ALGORITHM

辺存在確認回路の実行をスタートさせる状態である．辺存在確認回路を実行する前に，メモリ P の入力アドレスがメモリ ETS(A) の出力から受け取るように，入力アドレスの選択 Psel を 0 にする．Start sub を 1 にすると，辺存在確認回路の実行が開始する．また，この状態では，辺存在確認回路から出力されるプロセスの終了信号 end sub と辺存在確認の結果 result を待つ．end sub が 0 であれば，待ち続ける．result=1 かつ end sub=1 であれば，CHECK DEPTH に移る． result=0 かつ end sub=1 であれば，CHECK CURRENT へ戻る．

* CHECK DEPTH

現在深さ d が p_α まで調べているかどうかを判定する状態である．比較器=p_α?からの出力信号 check EqPa が 1 であれば，FOUND に移る．0 であれば，NEXT DEPT に移る．

* FOUND

部分同型グラフが発見されたことを示す状態である．動作としては，found 信号を 1 にする．そして，CHECK CURRENT に移る．

* NEXT DEPTH

次の深さ d+1 へ調べる状態である．動作としては，$v_{\beta i}$ への写像が使用済みであること (=0) を used の i 番目のビットに記憶して，current の値を M_d にコピーする．そして，深さ d を 1 インクリメントする．used の i 番目のビットに 0 を格納するためには，usedSel を 0，usedWrite を 1 にセットする．レジスタ current の値をメモリ M の d 番地に格納するために，メモリ M への入力データの選択 MSel を 0，書き込みイネーブル Mwrite を 1 にする．深さ d を 1 インクリメントするためには，メインコントロールユニットから出力された next d をレジスタ d に格納する．この時，書き込みイネーブル dWrite を 1 にする．最後に，SET CURRENT へ戻る．

* CHECK COMPLETE

全ての深さ d を調べたからどうかを判定する状態である．比較器=1?から出力した信号 check EqOne が 1 であれば，END MAIN に移る．0 であれば，SET REST PART に移る．

* SET REST PART

現在の深さ d から 1 つ戻る状態である．前の深さ d-1 に戻るために, メモリ IM の d 番地をメモリ M の d 番地にコピーして，深さ d を 1 ディクリメントする．メモリ IM の d 番地をコピーするために，メモリ M への入力データの選択 MSel を 1，書き込みイネーブル Mwrite を 1 にする．深さ d を 1 ディクリメントするには，メインコントロールから出力された Next d を d に格納する．この時，書き込みイネーブル dWrite を 1 にする．そして，LOAD P に移る．

* LOAD P

$v_{\alpha d}$ から $v_{\beta i}$ への写像が使用済みになっているものを復帰する状態である．動作としては，メモリ P の d 番地目のデータをレジスタ used の i ビット目として使用して，そのビットを 1 にセットする．メモリ P から d 番地目のデータを取るために，Psel を 1，PWrite を 0 にセットする．レジスタ used の i ビット目を 1 にセットするには，decode & set1 を使用する．decode & set1 からの出力をレジスタ used に格納するには，usedSel を 1 にセットする．そして，CHECK CURRENT へ戻る．

* END MAIN

探索木巡回回路の終了を示す状態である．end main 信号を 1 にする．そして，reset main が 0 であれば，この状態をそのまま保持する．1 であれば，INIT MAIN に移る．

4.3 実装

この節では，OPERL ボードに小西のアルゴリズムの実行回路を実装するために行った方法，実装結果について説明する．

4.3.1 実装方法

小西のアルゴリズムの実行回路を実装するために以下の手順で行った．

- 論理回路の記述

  今回実装する回路の論理は全て VHDL で記述する．記述する際には，実際に実装を考慮して Lucent 社の OR2CxxA シリーズ用のライブラリを直接指定することがほとんどである (例えば，RAM のマクロ，レジスタ，マルチプレクサなど)．しかし，各回路のコントロールユニットは動作が複雑であるため，VHDL で動作記述し，回路の構成を論理合成の段階に任せる．

- 論理合成

  記述した回路の論理 (VHDL ファイル) は Synopsys 社の Design Compiler で論理合成を行う．この論理合成を行う際に，テクノロジライブラリを指定することにより VHDL の機能記述を OR2C FPGA の適切なコンポーネントに変換することができる．論理合成を行った環境を表 13 に示す．今回実装する回路の論理合成は 10 分程度かかる．論理合成の出力ファイルは，EDIF ネットリスト形式である．

表 13: 論理合成を行う環境

プロセッサ	Ultra SPARC 167MHz
主記憶	128MB
基本ソフトウェア	Solaris 2.5.1

- マッピングと配置配線

  ORCA FPGA のセル単位に写像 (分割) させるために，得られた EDIF ネットリスト形式ファイルをマッピングする．マッピングの出力は ncd ファイルである．マッピングで生成されたファイルで配置配線を行う．配置配線は，ORCA FPGA 内にどのセルに配置するかまたは，配置したセルと他のセルとの間でどのように配線するべきかを行う．これらのツールは，Lucent 社の ORCA Foundry 9.35 を用いた．また，行った環境を表 14 に示す．今回実装する回路のマッピングと配置配線を行うには，2 時間程度かかる．この作業で出力された構成データ (bit stream) をホストコンピュータ経由で USER FPGA にダウンロードする．

表 14: マッピングと配置配線を行った環境

プロセッサ	Intel Pentium II 450 MHz
チップセット	Intel 440BX
２次キャッシュ	512KB(CPU に内蔵)
主記憶	SDRAM 256MB
基本ソフトウェア	MS-Windows NT Workstation 4.0

4.3.2 実装結果

OR2C FPGA に実装した回路規模と動作周波数をまとめて表 15 に示す．1 つのユニットの PFU 数は 160 程度であるので，最大 400FPU を利用できる OR2C15A には，2 つのユニットを実装することが可能である．インタ

フェース回路は PCI FPGA に入出力データを受け渡す回路なので，PCI バスと同じクロックで動作している．それぞれのユニットは，クロックの倍周期で動作している．それぞれのユニットをパイプライン化することにより，さらに性能向上が期待できると思われるが，これについては時間の都合上行っていない．

表 15: 回路規模と動作周波数

回路	PFU 数	動作周波数 (MHz)
インタフェース	23	33
Unit0	160	16.5
Unit1	160	16.5
合計	343	–

各ユニットの内部構成の PFU 数を表 16 に示す．辺存在確認回路の PFU 数の約半分は，RAM マクロとして利用されている．OR2C FPGA は SRAM ベース FPGA であるので，今回の実装は FPGA の特徴の一つを有効に活用していることがわかる．表 15 に示している 1 つのユニット PFU 数は，表 16 の PFU の合計数と若干異なっているが，これはテクノロジマッピングによるものであると考えられる．マッピングは，リソースを節約するために PFU 内に空いているところがあれば，無関係なセルを詰め込もうとしている．このため，回路の内訳にはマッピング度に PFU 数が若干異なることになる．

実装した各ユニットの中に，16 ビットカウンタの回路が含まれている．これは，各ユニットが正しい動作しているかを確認するために，同型名部分グラフの発見数をカウントする．この回路は，表 16 のその他項の PFU 数に含まれている．

表 16: ユニットの内部構成の PFU 数

回路	PFU 数	RAM(個)
探索木巡回	79	8
辺存在確認	56	24
その他	24	–
合計	159	32

本論文では，今回実装したインタフェース回路を拡張し，小西のアルゴリズムの実行回路を 1 から 4 つまで並列に利用できるようにインタフェース回路を設計してみた (拡張方法については付録 A を参照)．このインタフェースを実装し，4 つまで並行に利用できる回路の PFU 使用率を調べた．その結果を表 17 に示す．表中で n ユニットの意味は次の通りである (n は $1 \leq n \leq 4$)．1 ユニットはインタフェース回路とユニット 0，2 ユニットはインタフェース回路, ユニット 0,1 より構成されている．同様に 3 ユニットはインタフェース回路, ユニット 0,1,2, 4 ユニットはインタフェース回路, ユニット 0,1,2,3 より構成されている．3 ユニット以上を実装した場合の PFU 数は，2 ユニットの PFU 数に 160 の倍数を足しても，結果が 160 の倍数と若干異なっている．これは，テクノロジマッピングによるものと，インタフェース回路の拡張によるものであると考えられる．表より，OR2C FPGA の容量が大きいほど，より多くのユニットを実装することができる．これによってさらに性能向上も期待できる．

5 性能評価

この章は，OR2C15A に実装した 1 ユニットと 2 ユニット小西のアルゴリズムの実行回路の性能を測定する方法と，Ullmann のアルゴリズムをソフトウェアで実装した場合の性能と比較した結果について説明する．また，性能改善について説明する．以後，1 ユニット小西のアルゴリズムの実行回路を「1 ユニット」, 2 ユニット小西のアルゴリズムの実行回路を「2 ユニット」と呼ぶ．

表 17: 小西のアルゴリズムの実行回路の PFU 使用率

	2C15A	2C26A	2C40A
最大使用可能の PFU 数	400	576	900
1 ユニットの場合の PFU 使用率 (%) (PFU 数:183)	45	31	20
2 ユニットの場合の PFU 使用率 (%) (PFU 数:343)	85	59	38
3 ユニットの場合の PFU 使用率 (%) (PFU 数:499)	124	86	55
4 ユニットの場合の PFU 使用率 (%) (PFU 数:664)	166	115	73

5.1 1 ユニットの場合

5.1.1 測定方法

入力データは，Ullmann のアルゴリズムと小西のアルゴリズム (ソフトウェア) に与えたグラフ (100 セット) と同じものを用いる．ハードウェアの実行時間の測定方法を図 30 に示す．図中で「実行時間 $_j$」は，j ループ目の T2 と T1 の間の時間差である．この測定方法では，回路の実行されている分部のみを測っている．

5.1.2 性能評価の結果

- $G_\alpha \subset G_\beta$ の場合

  $(ed_\alpha, ed_\beta) = (0.2, 0.2), (0.2, 0.4), (0.4, 0.2), (0.4, 0.4)$ の時の 1 ユニットと Ullmann のアルゴリズムの性能比をそれぞれ図 31, 32, 33, 34 に示す．得られた 1 ユニットの実行時間が小西のアルゴリズム (ソフトウェア) と比較して，5%以内の誤差で実行時間の傾向が変わらない．

- $G_\alpha \not\subset G_\beta$ の場合

  $(ed_\alpha, ed_\beta) = (0.2, 0.2), (0.2, 0.4), (0.4, 0.2), (0.4, 0.4)$ の時の 1 ユニットと Ullmann のアルゴリズムの性能比をそれぞれ図 35, 36, 37, 38 に示す．得られた 1 ユニットの実行時間が小西のアルゴリズム (ソフトウェア) と比較して，5%以内の誤差で実行時間の傾向が変わらない．

5.2 2 ユニットの場合

この節では，ランダムに生成した G_α, G_β を入力データとし，2 ユニットと Ullmann のアルゴリズムのソフトウェアで処理し，それらの性能について評価を行う．

5.2.1 測定方法

部分グラフ同型判定の実行時間は，入力データに依存している．このため，本研究ではランダムに生成した 100 パターンのグラフに関して実行時間を測定し，合計の実行時間の平均値を求めて評価する．入力グラフ G_α, G_β を生成するには，3.2.2 節で説明したアルゴリズムを利用する．入力グラフのパラメータとして $p_\alpha, p_\beta, ed_\alpha, ed_\beta$ を用いて，$2 \leq p_\alpha, p_\beta \leq 15, (ed_\alpha, ed_\beta) = (0.2, 0.2), (0.2, 0.4), (0.4, 0.2), (0.4, 0.4)$ についての傾向を観察する．

```
#define LOOP 100

for (pα と pβ の組合せ) {
データ生成；
   for(j=0;j<2;j++) {
      T1:   getrusage()
        for(回数= LOOP) {
          if(j) {
            ハードウェアの初期化;
            ハードウェアの実行開始;
            ハードウェアの終了信号を待つ；
          }
        }
      T2:   getrusage()
   temptime[j]=実行時間 j;
   }
   used_time = (temptime[1]-temptime[0])/LOOP;
   used_time を記録;
}
```

図 30: ハードウェアの実行時間の測定方法

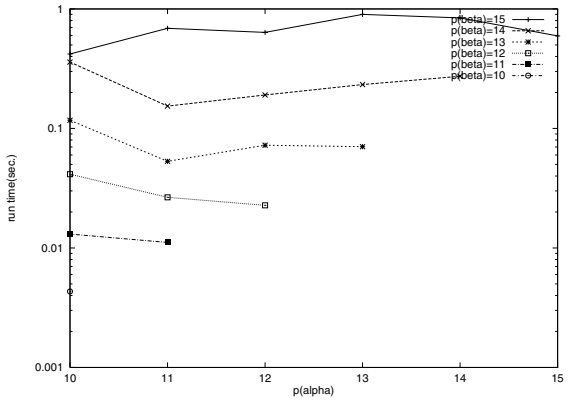
5.2.2 ソフトウェアの性能

性能の比較対象としては，表 1 の環境上で Ullmann のアルゴリズムをソフトウェアで処理した場合の実行時間を用いる．以下，このシステムを比較対象システムと呼ぶ．比較対象システムでソフトウェアで処理した実行時間を測定し，その合計の実行時間の平均値を求めることを行った．図 39 には $(ed_\alpha, ed_\beta) = (0.2, 0.2)$ の場合，図 40 には $(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合，図 41 には $(ed_\alpha, ed_\beta) = (0.4, 0.2)$ の場合，図 42 には $(ed_\alpha, ed_\beta) = (0.4, 0.4)$ の場合の結果を示す．各図 (図 39〜42) の (b) のグラフは，それぞれの場合に対する部分グラフ同型検出パターン数であり，(c) のグラフはそれぞれの場合に対する部分グラフ同型非検出パターン数である．以下，性能測定では，この比較対象システムで実行時間を基準として，性能比のみを示すこととする．

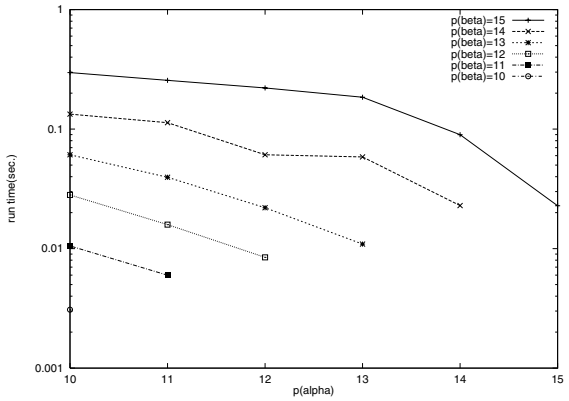
5.2.3 専用回路の性能

比較対象システムと同じ 100 パターンのグラフを専用回路で処理し，実行時間を測定した．専用回路の性能を測定するために次のような方法を利用した．ホスト側のソフトウェアでは，PCI バス経由で OPERL ボード上に Unit0 と Unit1 にグラフを 1 組ずつ割り当てる．その後，inl() 命令 [4] で各ユニットの状態を監視しながらループして待ち続ける状態になる．処理が終えたユニットに新たなグラフを割り当てる．

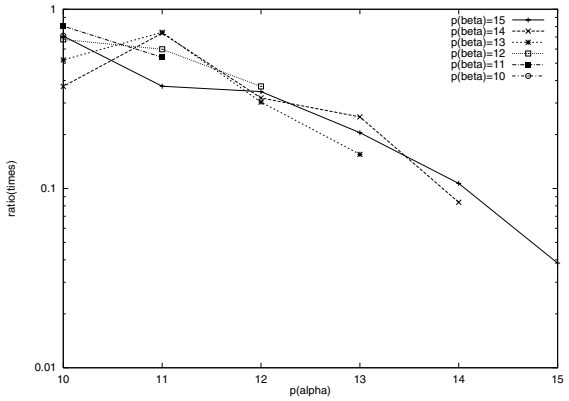
測定した結果を比較対象システムとの性能比を求めた．図 43 には $(ed_\alpha, ed_\beta) = (0.2, 0.2)$ の場合，図 44 には $(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合，図 45 には $(ed_\alpha, ed_\beta) = (0.4, 0.2)$ の場合，図 46 には $(ed_\alpha, ed_\beta) = (0.4, 0.4)$ の場合の結果を示す．2 ユニットの性能は，$(ed_\alpha, ed_\beta) = (0.2, 0.4), (0.4, 0.4)$ のとき，多くの点で比較対象システムより 10〜40 倍程度優れていることがわかった．しかし，$(ed_\alpha, ed_\beta) = (0.2, 0.2), (0.4, 0.2)$ のときは，比較対象システムより 10 倍程度の性能が低下していることがわかった．これは前にも述べたように，小西のアルゴリズムの探索空間の削減の効率が Ullmann のアルゴリズムより劣ることが主な原因である．



(a)1 ユニットの実行時間



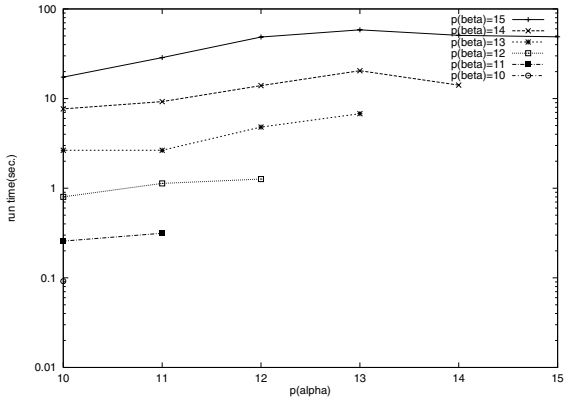
(b) Ullmann のアルゴ リズムの実行時間



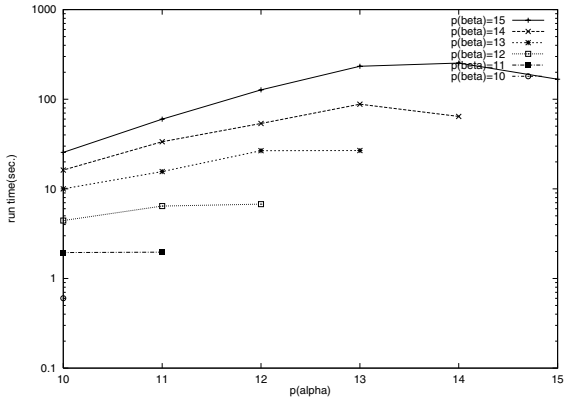
(c) 1 ユニットと Ullmann のアルゴ リズムの実行時間比

図 31: $(ed_\alpha, ed_\beta) = (0.19811, 0.19758)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00280)$ の場合

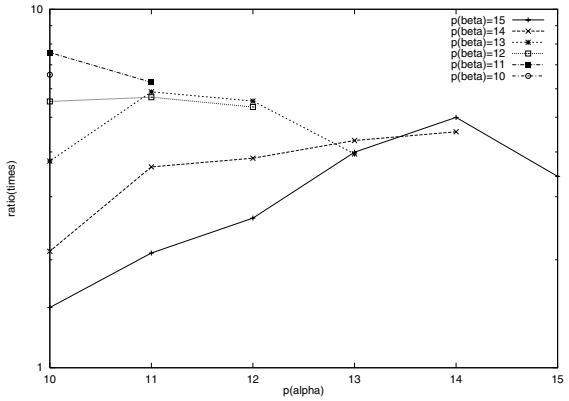
($G_\alpha \subset G_\beta$ の入力データ)



(a)1 ユニットの実行時間



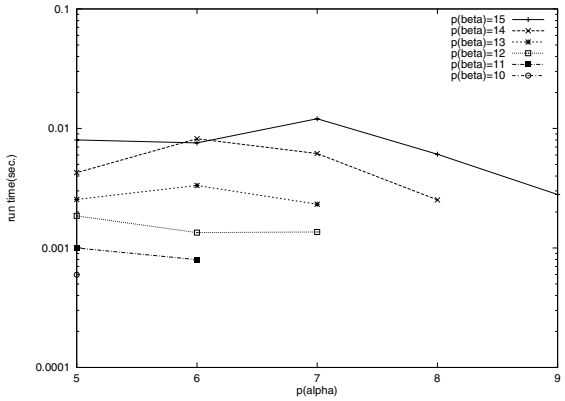
(b) Ullmann のアルゴ リズムの実行時間



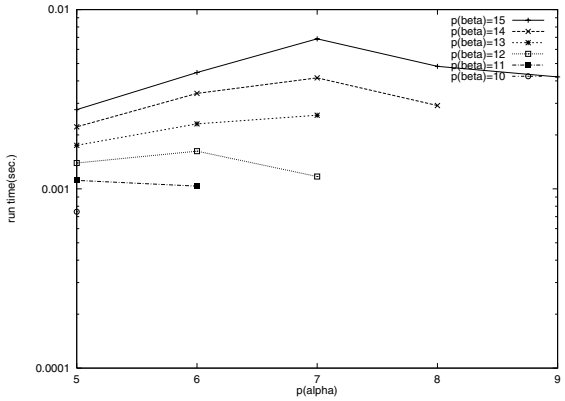
(c) 1 ユニットと Ullmann のアルゴ リズムの実行時間比

図 32: $(ed_\alpha, ed_\beta) = (0.19811, 0.39760)$,$(s.d._\alpha, s.d._\beta) = (0.00267, 0.00231)$ の場合

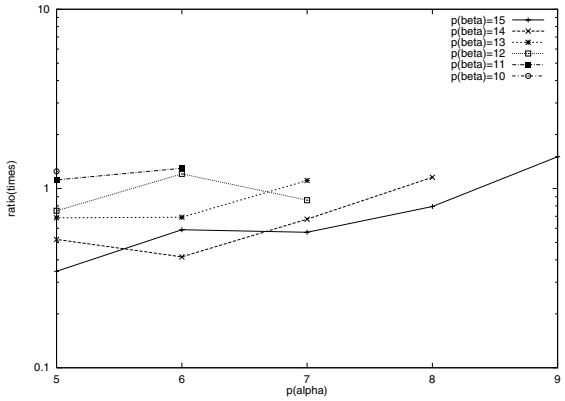
($G_\alpha \subset G_\beta$ の入力データ)



(a)1 ユニットの実行時間



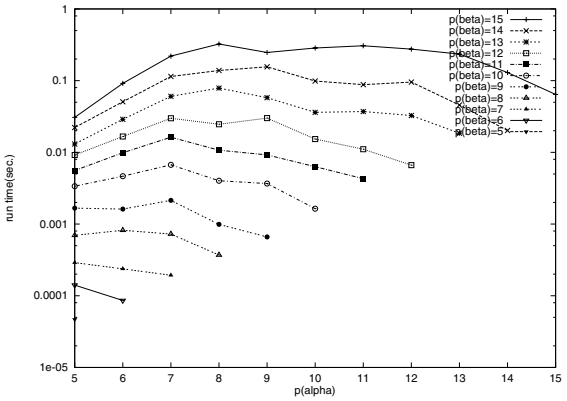
(b) Ullmann のアルゴ リズムの実行時間



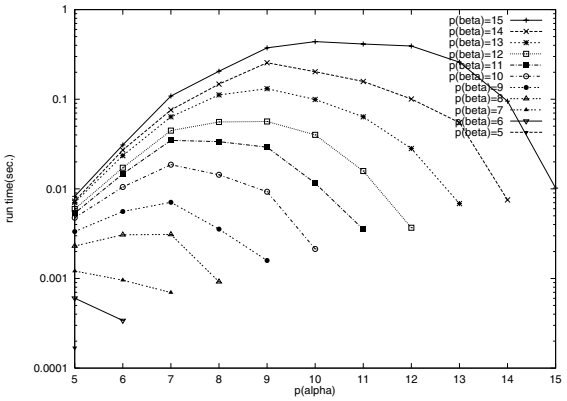
(c) 1 ユニットと Ullmann のアルゴ リズムの実行時間比

図 33: $(ed_\alpha, ed_\beta) = (0.39436, 0.19772)$,$(s.d._\alpha, s.d._\beta) = (0.00783, 0.00270)$ の場合

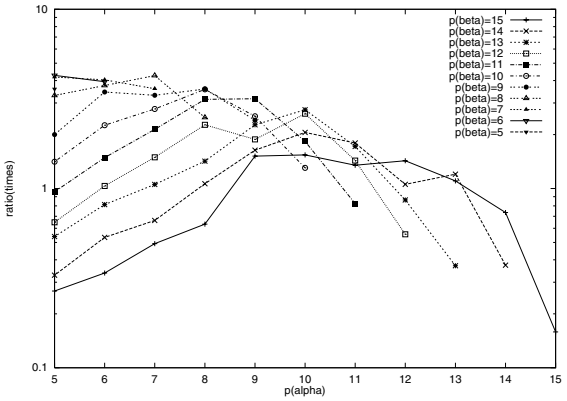
($G_\alpha \subset G_\beta$ の入力データ)



(a)1 ユニットの実行時間



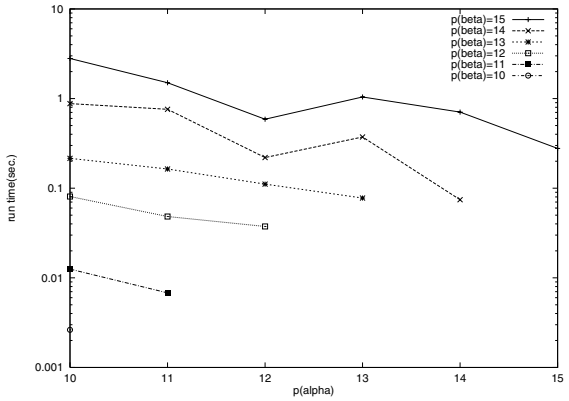
(b) Ullmann のアルゴ リズムの実行時間



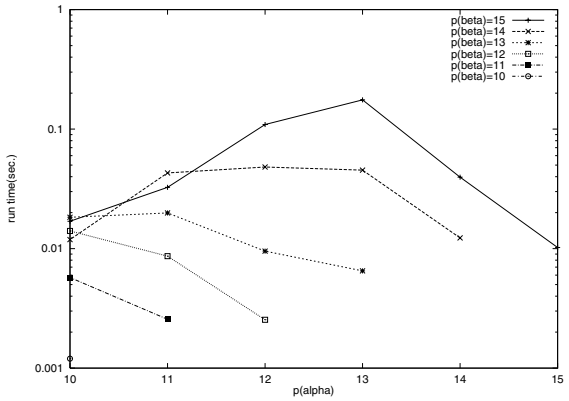
(c) 1 ユニットと Ullmann のアルゴ リズムの実行時間比

図 34: $(ed_\alpha, ed_\beta) = (0.39466, 0.39611)$,$(s.d._\alpha, s.d._\beta) = (0.00665, 0.00470)$ の場合

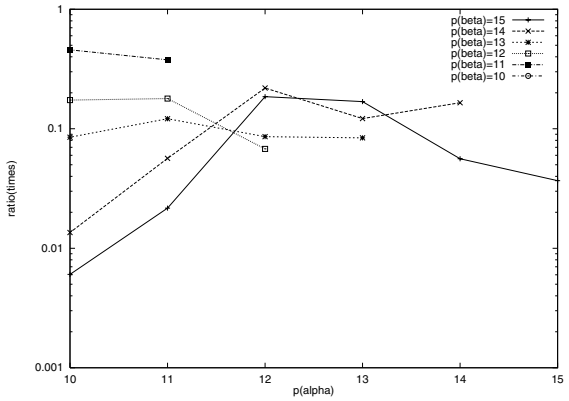
($G_\alpha \subset G_\beta$ の入力データ)



(a)1 ユニットの実行時間



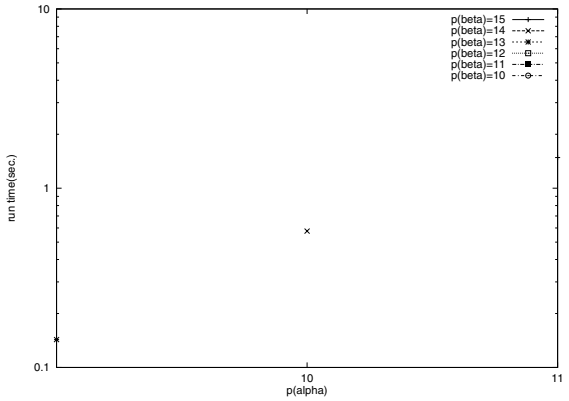
(b) Ullmann のアルゴ リズムの実行時間



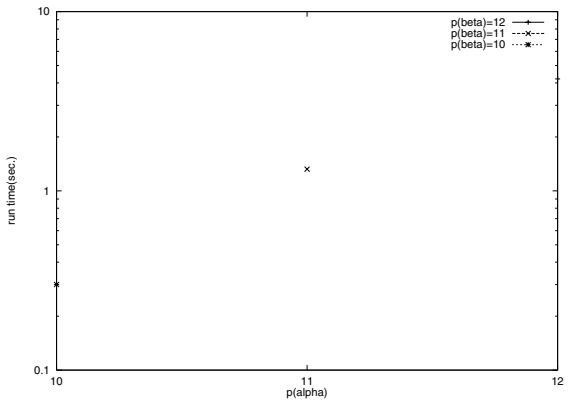
(c) 1 ユニットと Ullmann のアルゴ リズムの実行時間比

図 35: $(ed_\alpha, ed_\beta) = (0.19811, 0.19758)$,$(s.d._\alpha, s.d._\beta) = (0.00267, 0.00280)$ の場合

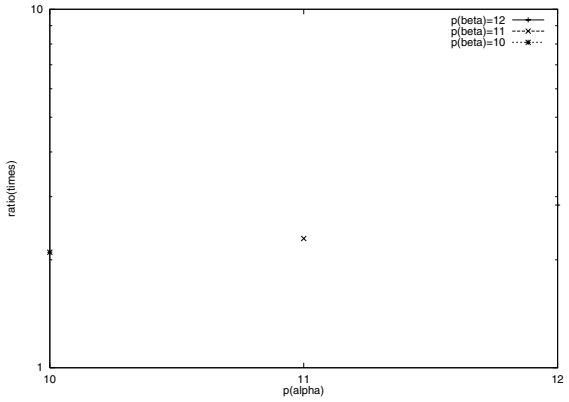
($G_\alpha \not\subset G_\beta$ の入力データ)



(a)1 ユニットの実行時間



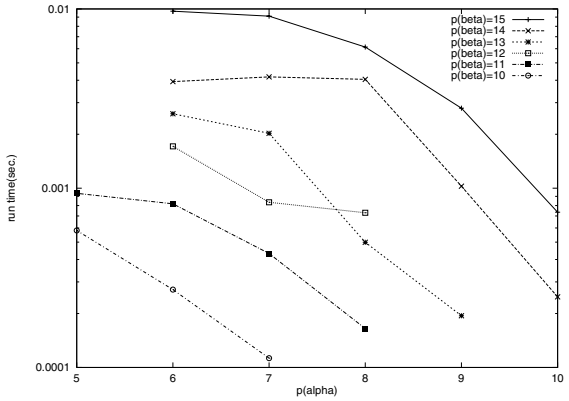
(b) Ullmann のアルゴ リズムの実行時間



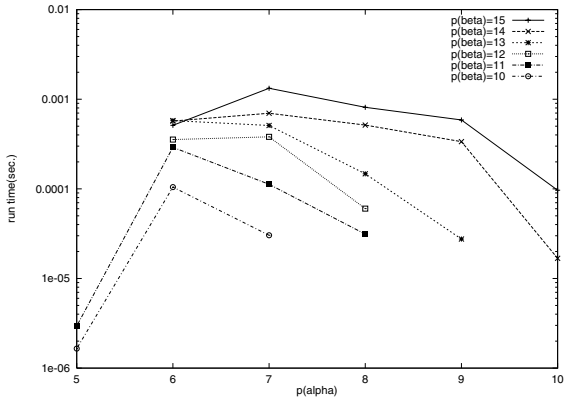
(c) 1 ユニットと Ullmann のアルゴ リズムの実行時間比

図 36: $(ed_\alpha, ed_\beta) = (0.19811, 0.39760)$,$(s.d._\alpha, s.d._\beta) = (0.00267, 0.00231)$ の場合

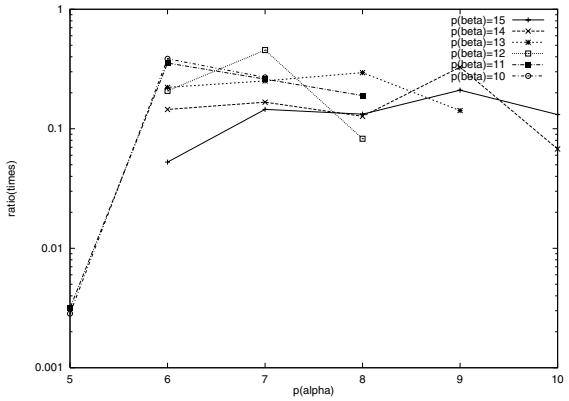
($G_\alpha \not\subset G_\beta$ の入力データ)



(a)1 ユニットの実行時間



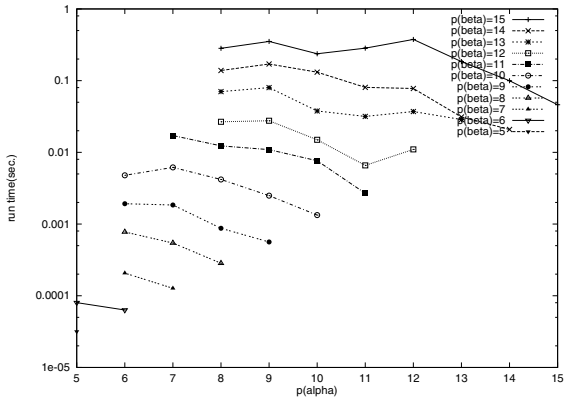
(b) Ullmann のアルゴ リズムの実行時間



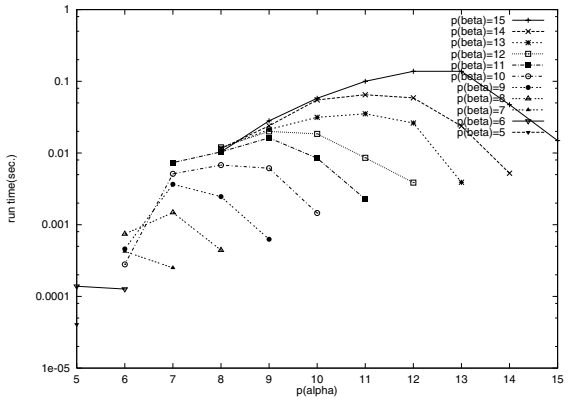
(c) 1 ユニットと Ullmann のアルゴ リズムの実行時間比

図 37: $(ed_\alpha, ed_\beta) = (0.39436, 0.19772)$,$(s.d._\alpha, s.d._\beta) = (0.00783, 0.00270)$ の場合

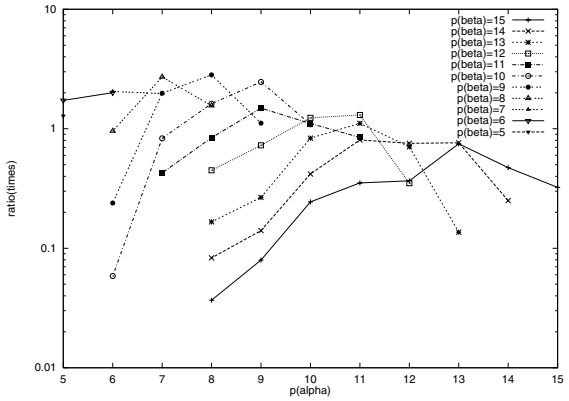
($G_\alpha \not\subset G_\beta$ の入力データ)



(a)1 ユニットの実行時間



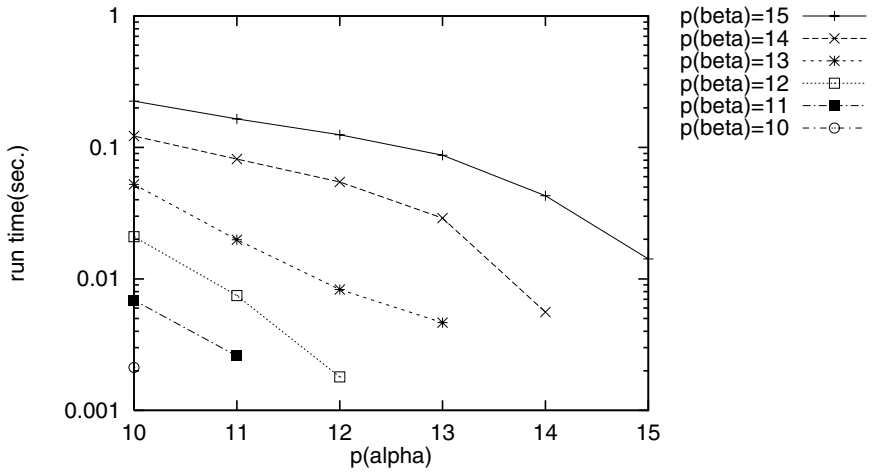
(b) Ullmann のアルゴ リズムの実行時間



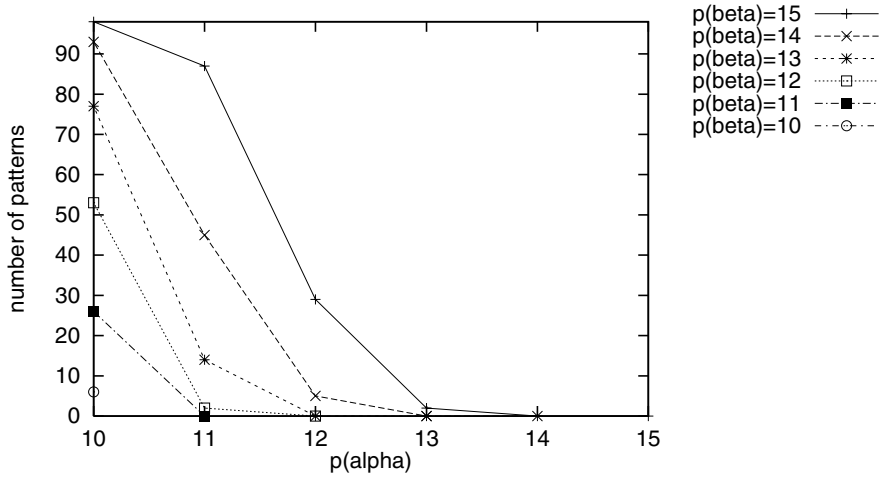
(c) 1 ユニットと Ullmann のアルゴ リズムの実行時間比

図 38: $(ed_\alpha, ed_\beta) = (0.39466, 0.39611)$,$(s.d._\alpha, s.d._\beta) = (0.00665, 0.00470)$ の場合

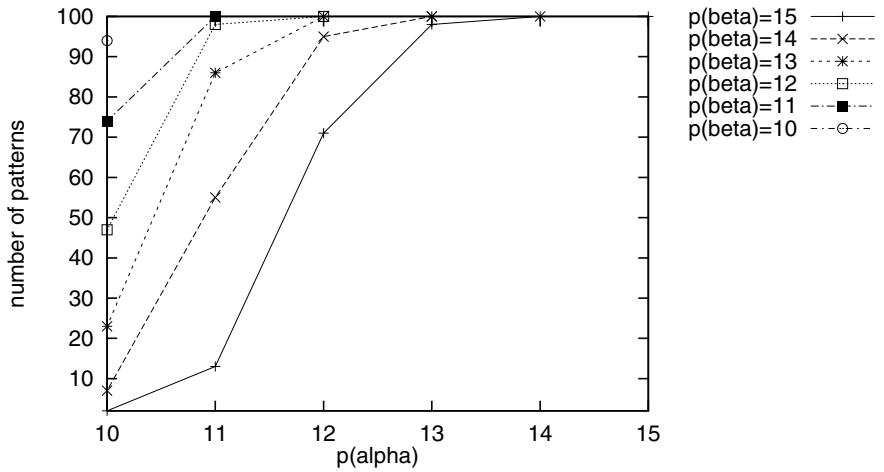
($G_\alpha \not\subset G_\beta$ の入力データ)



(a) Ullmann アルゴリズムの実行時間

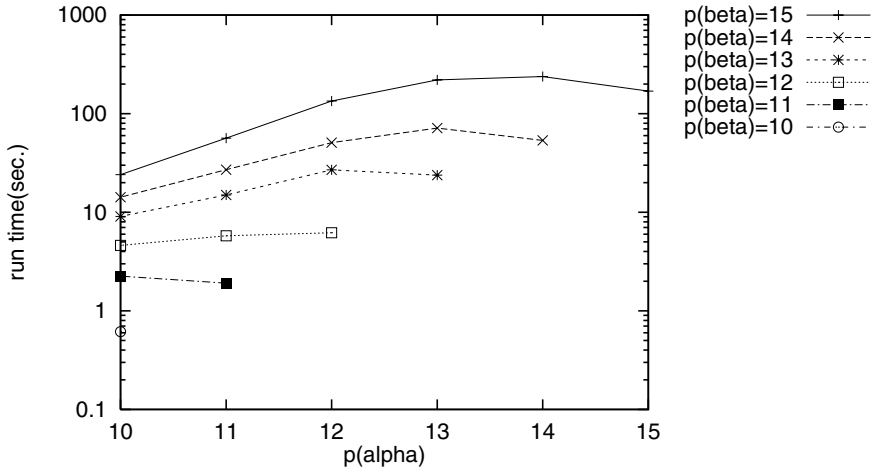


(b) 部分グラフ同型検出パターン数

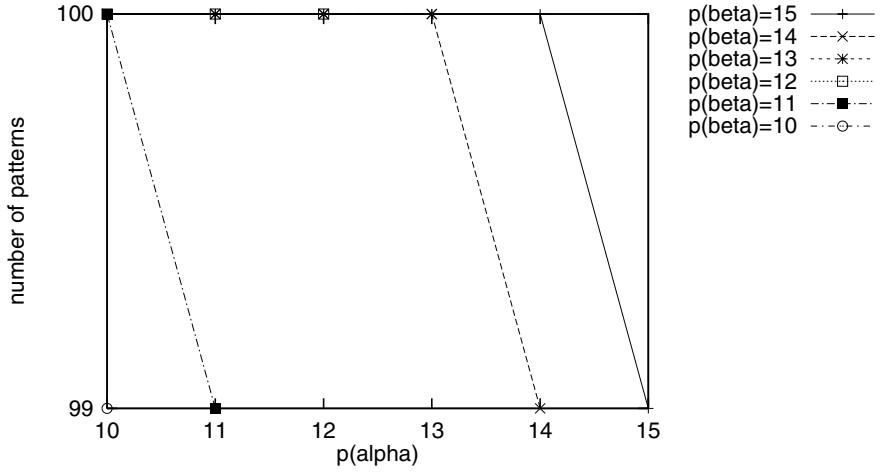


(c) 部分グラフ同型非検出パターン数

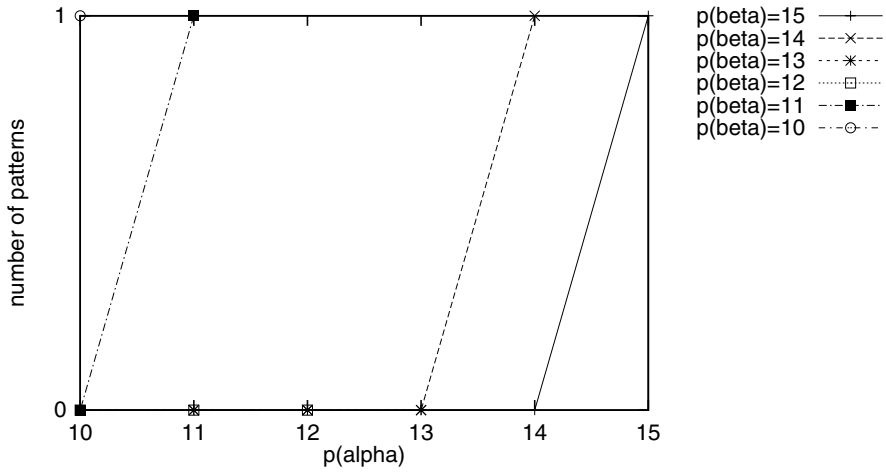
図 39: $(ed_\alpha, ed_\beta) = (0.19811, 0.19758)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00280)$ の場合 (G_α, G_β をランダムに生成した)



(a) Ullmann のアルゴ リズムの実行時間

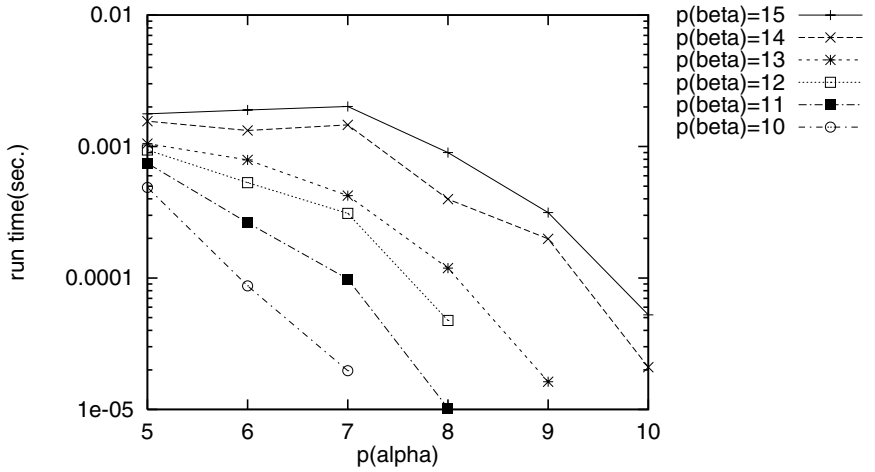


(b) 部分グラフ同型検出パターン数

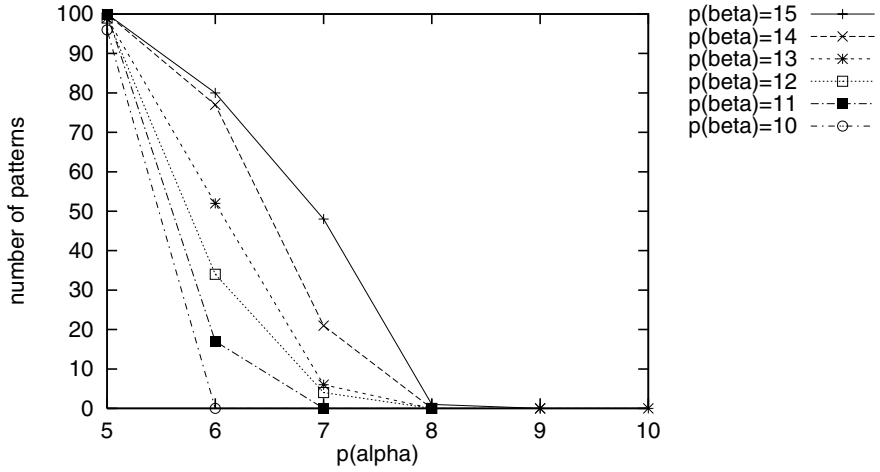


(c) 部分グラフ同型非検出パターン数

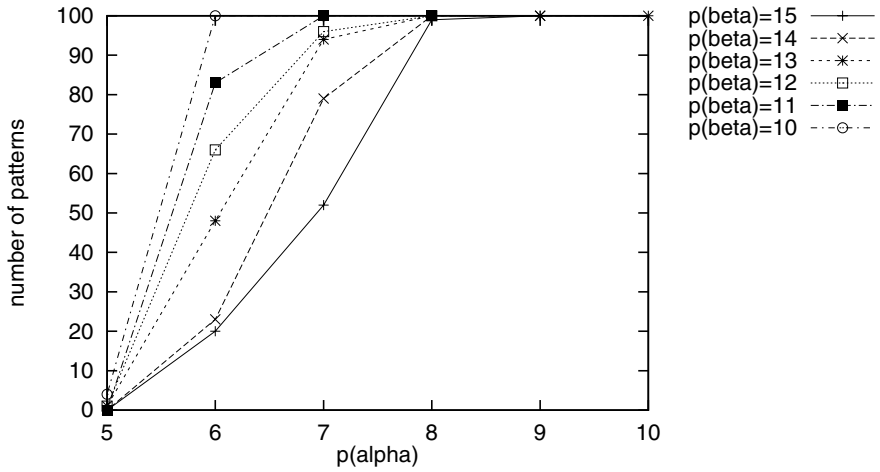
図 40: $(ed_\alpha, ed_\beta) = (0.19811, 0.39760)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00231)$ の場合 (G_α, G_β をランダムに生成した)



(a) Ullmann のアルゴ リズムの実行時間

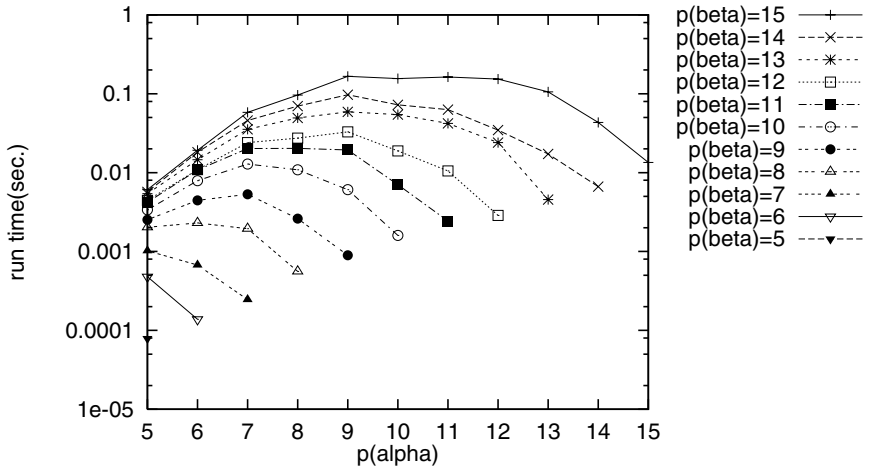


(b) 部分グラフ同型検出パターン数

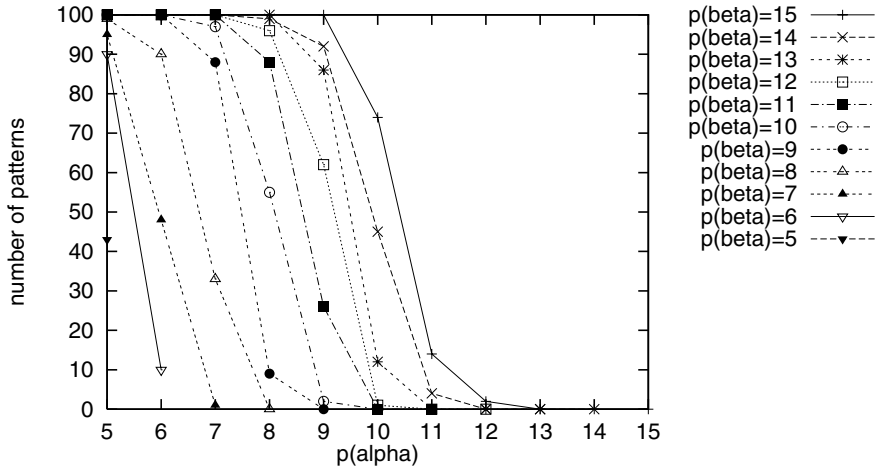


(c) 部分グラフ同型非検出パターン数

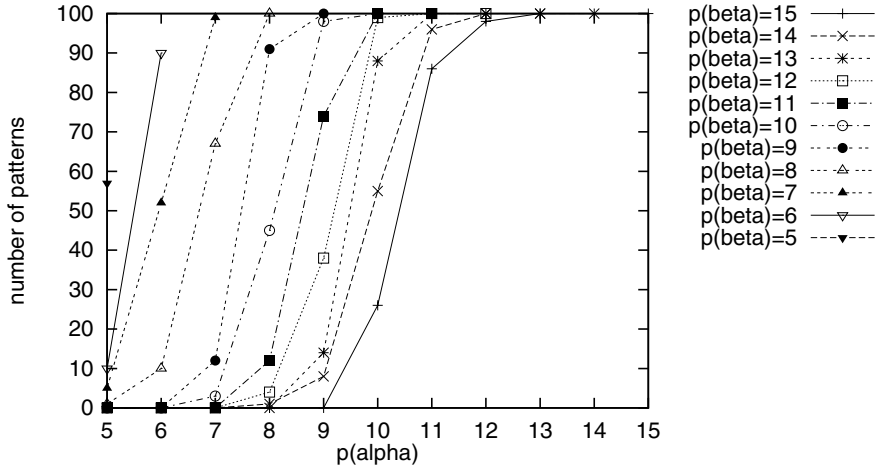
図 41: $(ed_\alpha, ed_\beta) = (0.39345, 0.19772)$, $(s.d._\alpha, s.d._\beta) = (0.00756, 0.00278)$ の場合 (G_α, G_β をランダムに生成した)



(a) Ullmann のアルゴ リズムの実行時間

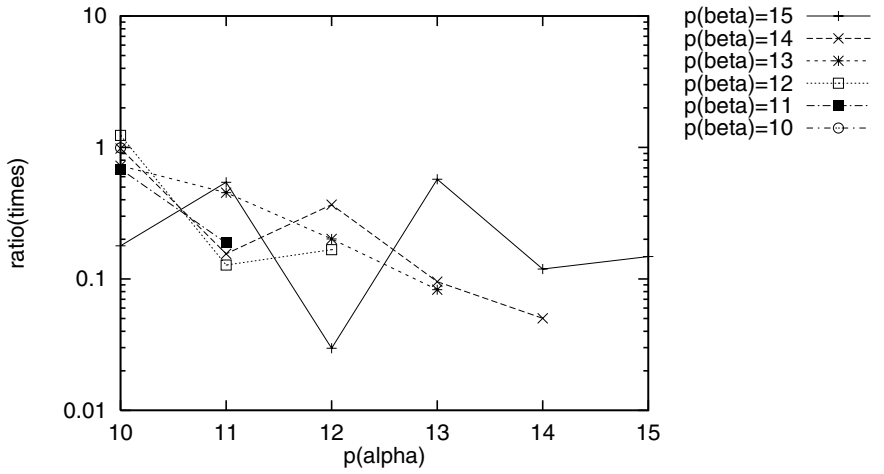


(b) 部分グラフ同型検出パターン数

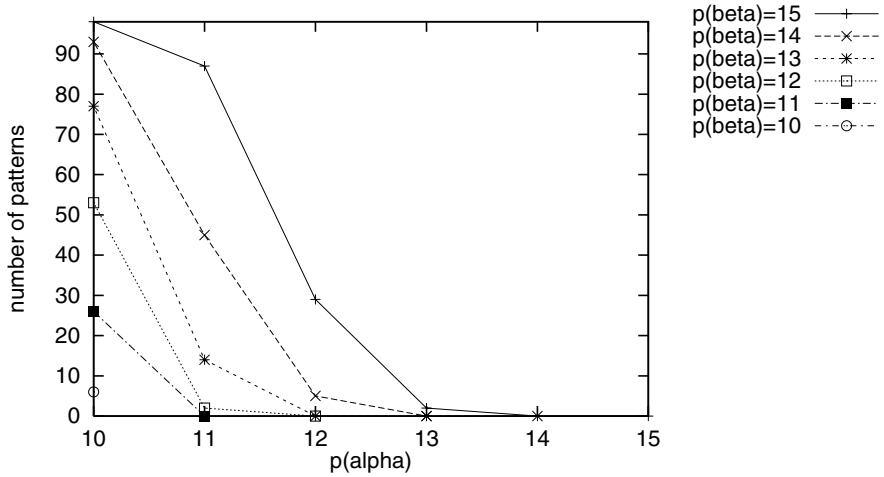


(c) 部分グラフ同型非検出パターン数

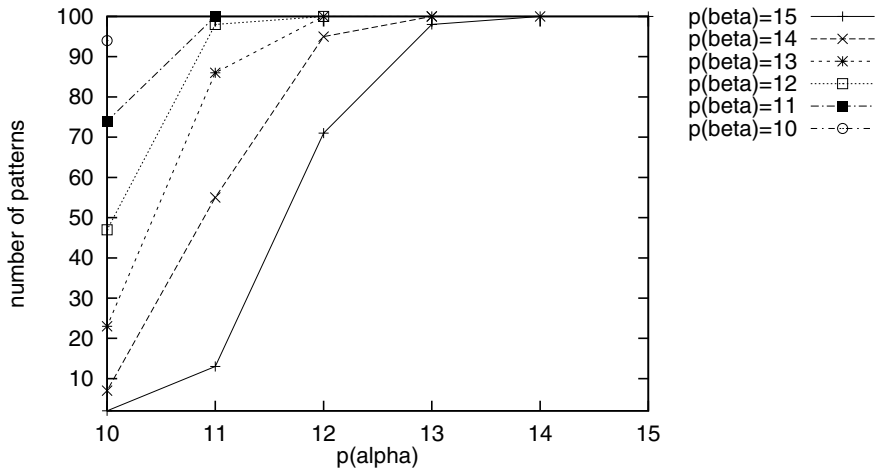
図 42: $(ed_\alpha, ed_\beta) = (0.39474, 0.39611)$, $(s.d._\alpha, s.d._\beta) = (0.00665, 0.00470)$ の場合
(G_α, G_β をランダムに生成した)



(a) 2 ユニットと Ullmann アルゴリズムの実行時間比

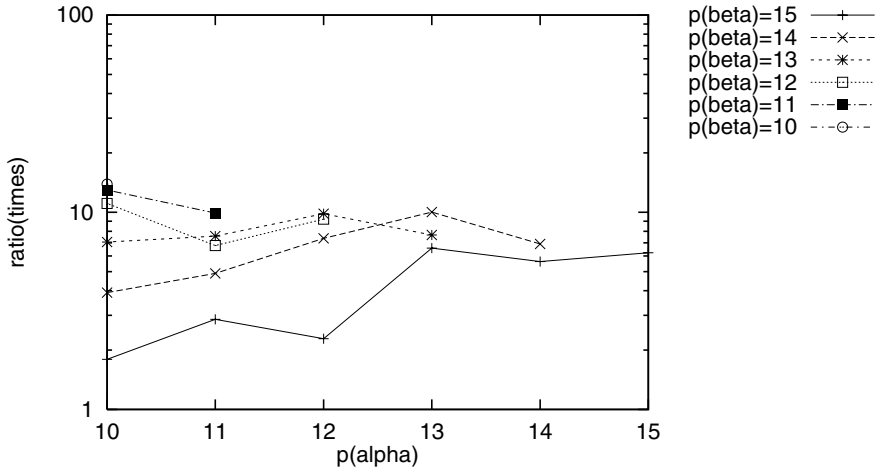


(b) 部分グラフ同型検出パターン数

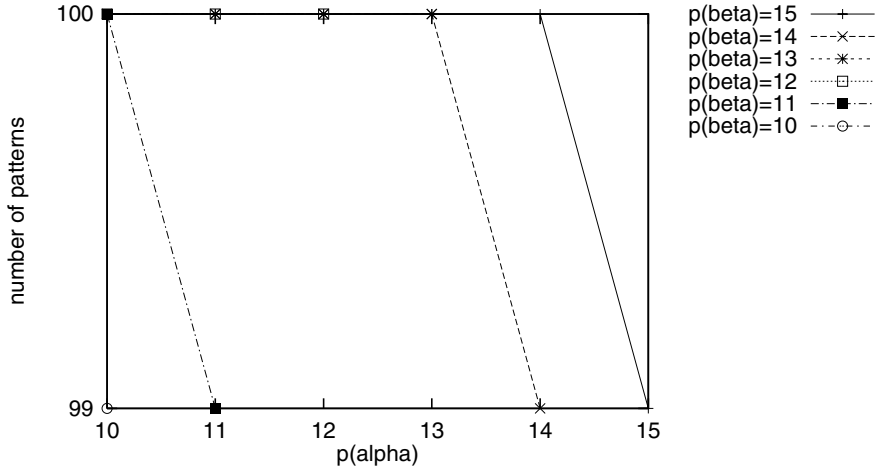


(c) 部分グラフ同型非検出パターン数

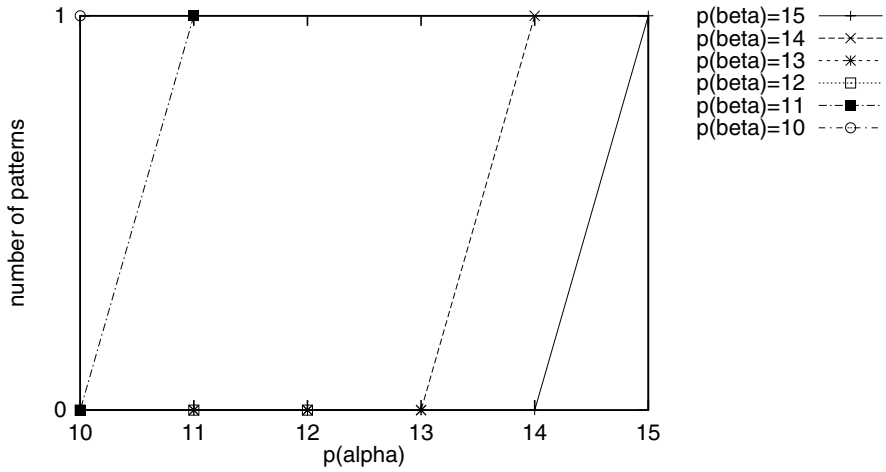
図 43: $(ed_\alpha, ed_\beta) = (0.19811, 0.19758)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00280)$ の場合 (G_α, G_β をランダムに生成した)



(a) 2 ユニットと Ullmann アルゴリズムの実行時間比

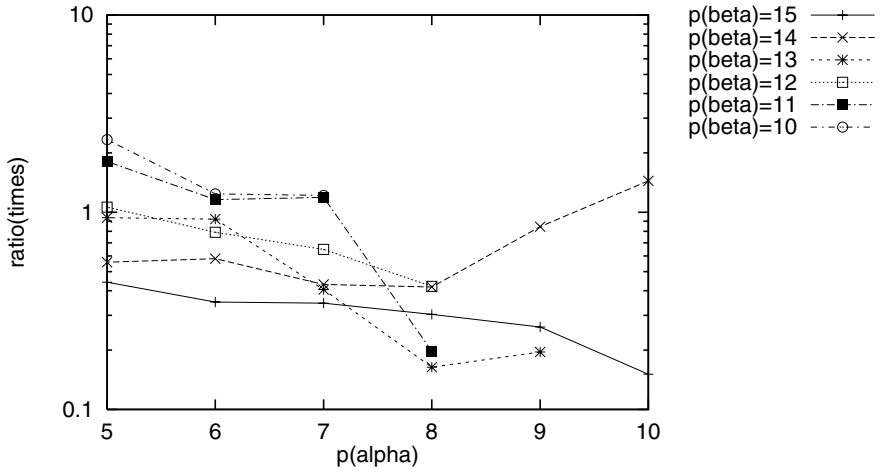


(b) 部分グラフ同型検出パターン数

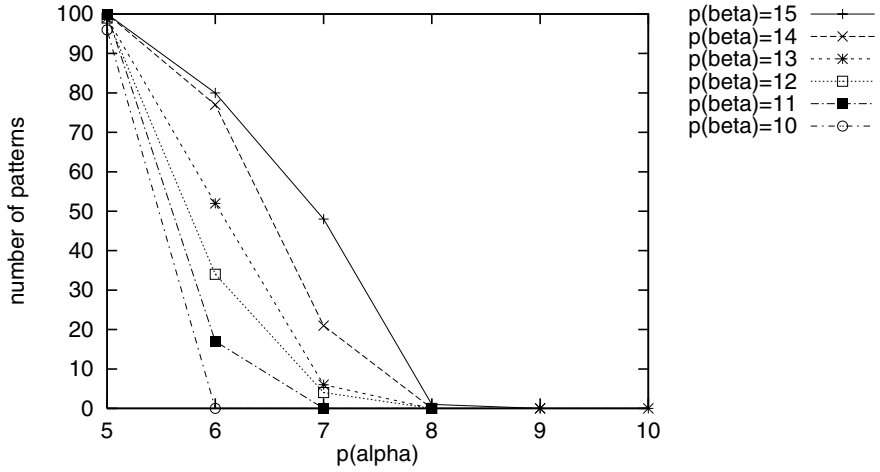


(c) 部分グラフ同型非検出パターン数

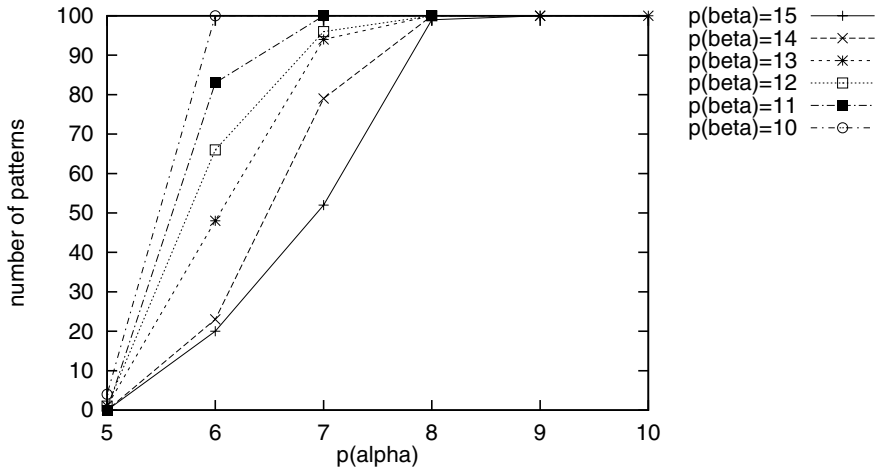
図 44: $(ed_\alpha, ed_\beta) = (0.19811, 0.39760)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00231)$ の場合
(G_α, G_β をランダムに生成した)



(a) 2 ユニットと Ullmann アルゴリズムの実行時間比

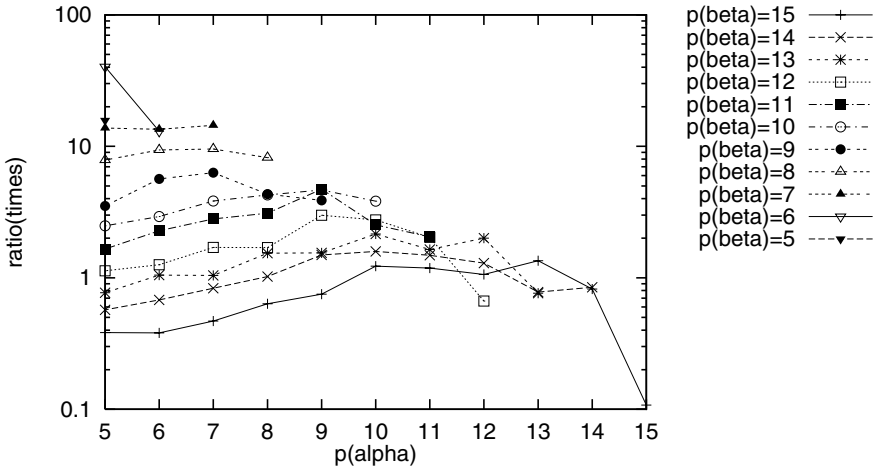


(b) 部分グラフ同型検出パターン数

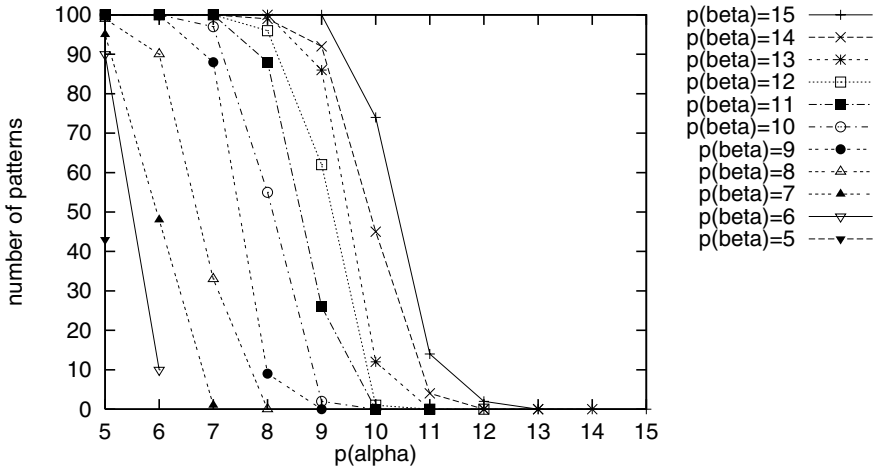


(c) 部分グラフ同型非検出パターン数

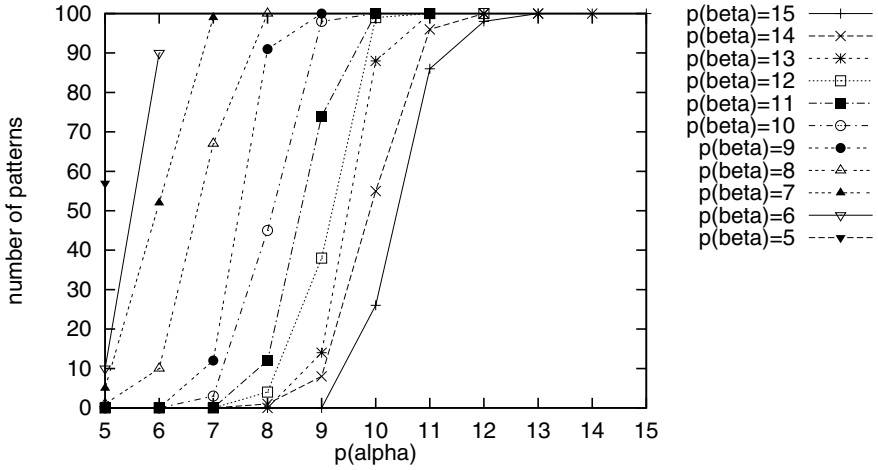
図 45: $(ed_\alpha, ed_\beta) = (0.39345, 0.19772)$, $(s.d._\alpha, s.d._\beta) = (0.00756, 0.00278)$ の場合 (G_α, G_β をランダムに生成した)



(a) 2 ユニットと Ullmann アルゴリズムの実行時間比



(b) 部分グラフ同型検出パターン数



(c) 部分グラフ同型非検出パターン数

図 46: $(ed_\alpha, ed_\beta) = (0.39474, 0.39611)$, $(s.d._\alpha, s.d._\beta) = (0.00665, 0.00470)$ の場合
(G_α, G_β をランダムに生成した)

5.2.4 性能改善

この節では，専用回路の性能が低下しているところを改善する方法について説明する．改善方法としては，選択法，分担法，協調法の 3 つの方法が考えられる．以下それぞれの方法の詳細について説明する．

- 選択法

  専用回路の性能が低下しているところでは，専用回路の代わりにホスト側のソフトウェアを用いて処理する方法．しかし，問題になるのは，部分グラフ同型判定を行う前に，専用回路またはホスト側のソフトウェアのどちらを使うべきかを予測することである．本研究での予測方法としては，グラフの属性である $p_\alpha, p_\beta, ed_\alpha, ed_\beta$ を用いて，それぞれのパラメータの組合せに対して事前に専用回路と比較対象システムとの性能比の統計を取り，データベースを構築する．実際に部分グラフ同型判定を行う際に，与えられた $p_\alpha, p_\beta, ed_\alpha, ed_\beta$ に対して，構築したデータベースのもとで専用回路の性能を予測する．予測した結果，専用回路の性能が上がりそうなところは，専用回路を利用する．専用回路の性能が下がりそうなところは，ホスト側のソフトウェアを利用する．

  当然，データベースを構築するために用いる 100 パターンのグラフと，実際に計算するときのグラフは別のものである．また，本研究で使用しているホストの環境と，比較対象システムの環境は異なっている．ホスト上で Ullmann のアルゴリズムのソフトウェアを用いて部分グラフ判定を行う場合と，比較対象システム上で同アルゴリズムを実行した場合の性能比が約 1.31 であるため，予測する時にデータベースのデータ (専用回路と比較対象システムの性能比) に 1.31 をかけて行う．

  選択法を用いて性能を測定した．図 47 には $(ed_\alpha, ed_\beta) = (0.2, 0.2)$ の場合，図 48 には $(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合，図 49 には $(ed_\alpha, ed_\beta) = (0.4, 0.2)$ の場合，図 50 には $(ed_\alpha, ed_\beta) = (0.4, 0.4)$ の場合の結果を示す．図 47〜50 からわかるように，ほとんどの場合に対して予測した結果がよく的中し，性能低下を回避できる．しかし，一部で外れることもあるが，1 倍程度に留まっている．また，これらのグラフは 1 ではなく，0.76 に近く集中しているのは，ホストの環境と比較対象システムの性能の差である．

- 分担法

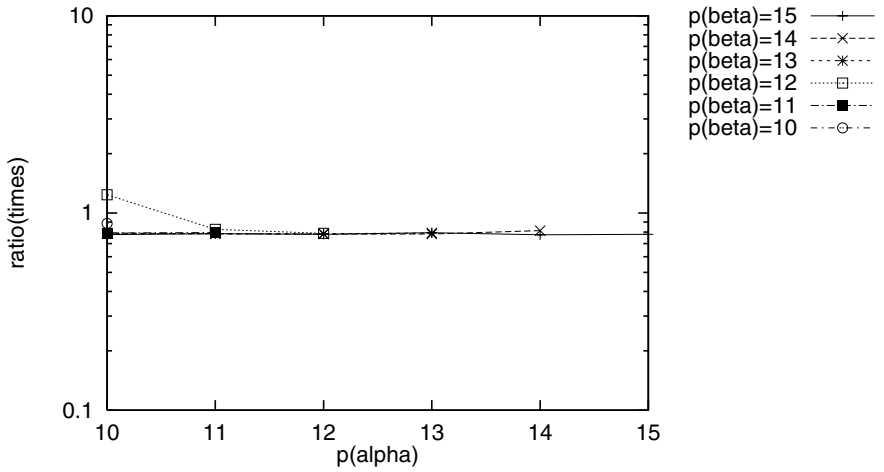
  ホストは，通常専用回路の終了を持っている状態になっている．これは無駄なので，待っている間にホスト側でも Ullmann のアルゴリズムのソフトウェアを用いて部分グラフ同型判定を分担することを考える．方法としては，ホスト側は 1 セットのグラフを判定してから，各ユニットの状態をチェックする．空いているユニットに対してデータ (グラフ) を与える．そしてホストは次のグラフの判定を行う．全部の処理数の合計は 100 になるまで，この処理を続けて行う．

  分担法を用いて性能測定を行った．図 51 には $(ed_\alpha, ed_\beta) = (0.2, 0.2)$ の場合，図 52 には $(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合，図 53 には $(ed_\alpha, ed_\beta) = (0.4, 0.2)$ の場合，図 54 には $(ed_\alpha, ed_\beta) = (0.4, 0.4)$ の場合の結果を示す．2 ユニットの性能と比べて，多くの性能低下が改善されているがわかる．しかし，ハードウェアがソフトウェアより速い場合もホストを使うため，ピーク性能が低下する (図 52,54 を参照)．

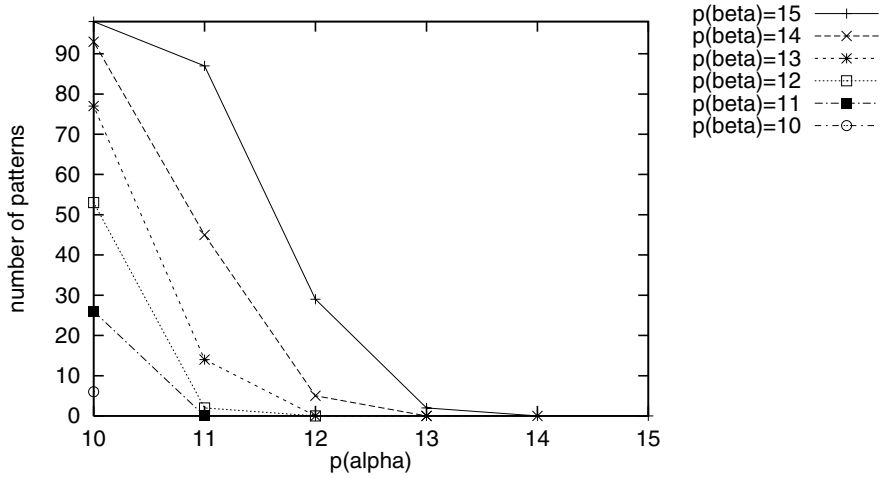
- 協調法

  協調法は，選択法と分担法を併用したものである．具体的には，選択法と同じデータベースを基に，与えられた $p_\alpha, p_\beta, ed_\alpha, ed_\beta$ から専用回路の性能を予測する．もし，専用回路の性能がホスト側のソフトウェアより K 倍以上速いと予測される場合は，専用回路を利用する．ホスト側のソフトウェアの性能が専用回路より K 倍以上速いと予測される場合は，ホスト側のソフトウェアを利用する．どちらでもないと予測される場合は，分担法を利用する．ここでパラメータ K を 1 にすると，選択法と同じ動作になる．また，K の値が大きくなると，選択法が多く使われることになる．

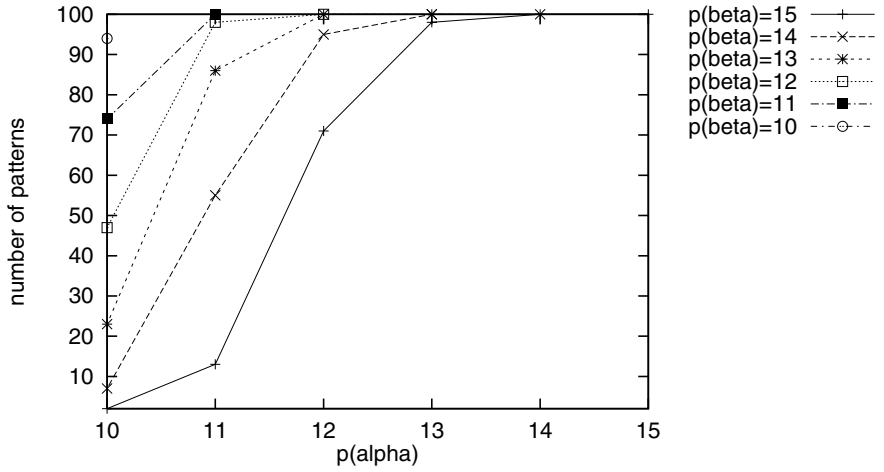
  K=2 として，協調法を用いて性能測定を行った．図 55 には $(ed_\alpha, ed_\beta) = (0.2, 0.2)$ の場合，図 56 には $(ed_\alpha, ed_\beta) = (0.2, 0.4)$ の場合，図 57 には $(ed_\alpha, ed_\beta) = (0.4, 0.2)$ の場合，図 58 には $(ed_\alpha, ed_\beta) = (0.4, 0.4)$ の場合の結果を示す．図 55,57,58 は，選択法と比べて，1 に近い部分は分担法によって性能が向上されることがわかる．図 56 は，専用回路のみが使用されている．



(a) 選択法と Ullmann アルゴリズムの実行時間比

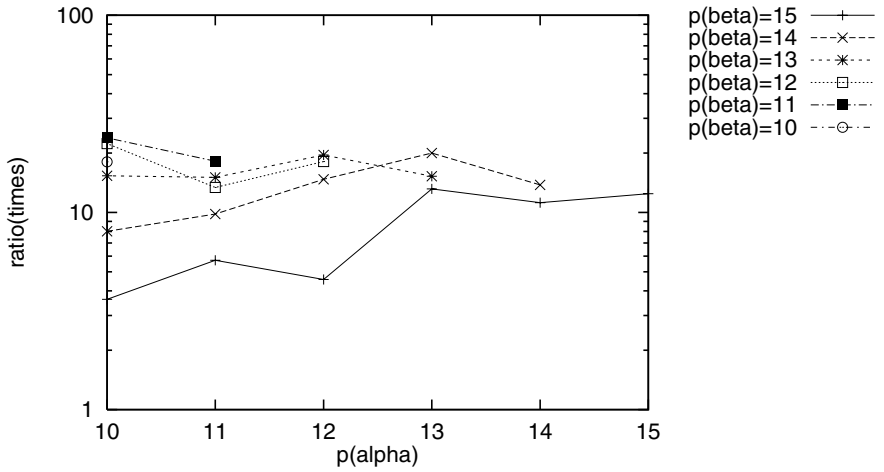


(b) 部分グラフ同型検出パターン数

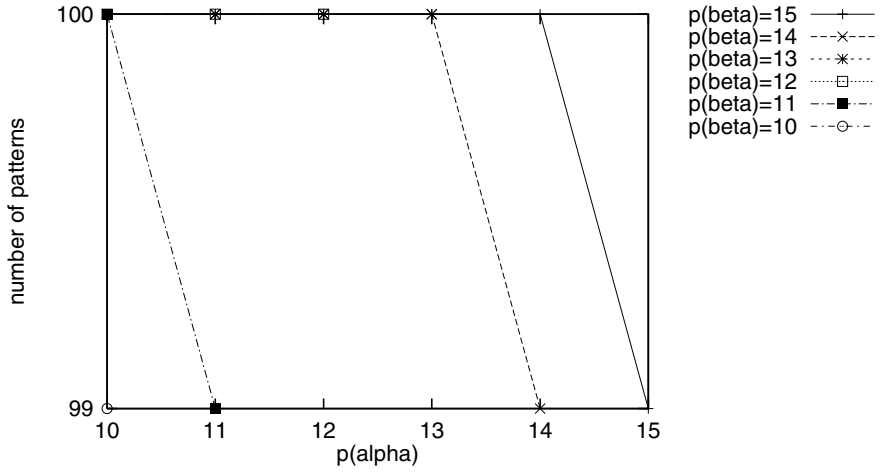


(c) 部分グラフ同型非検出パターン数

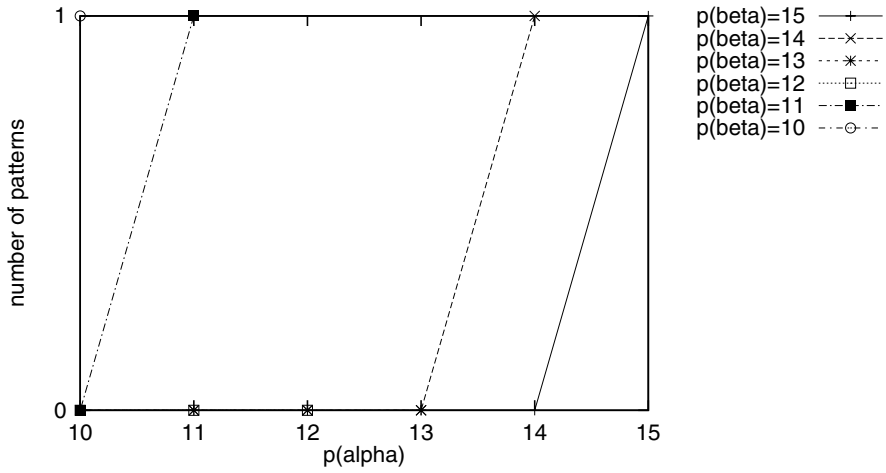
図 47: $(ed_\alpha, ed_\beta) = (0.19811, 0.19758)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00280)$ の場合



(a) 選択法と Ullmann のアルゴ リズムの実行時間比

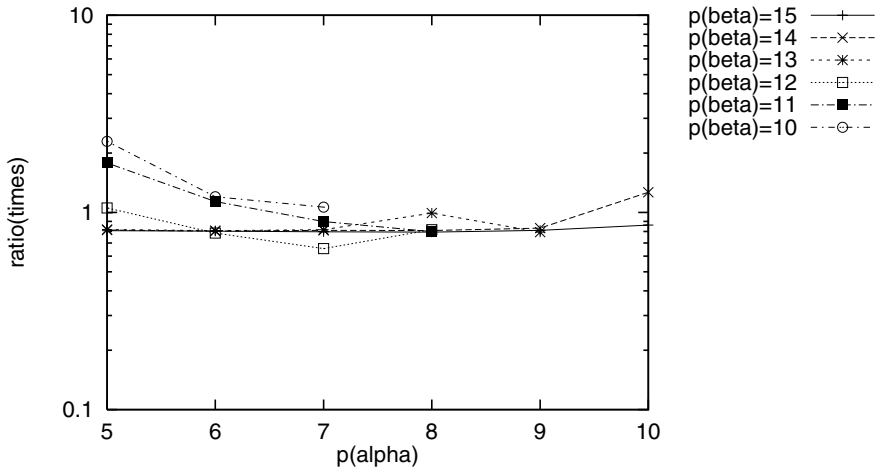


(b) 部分グラフ同型検出パターン数

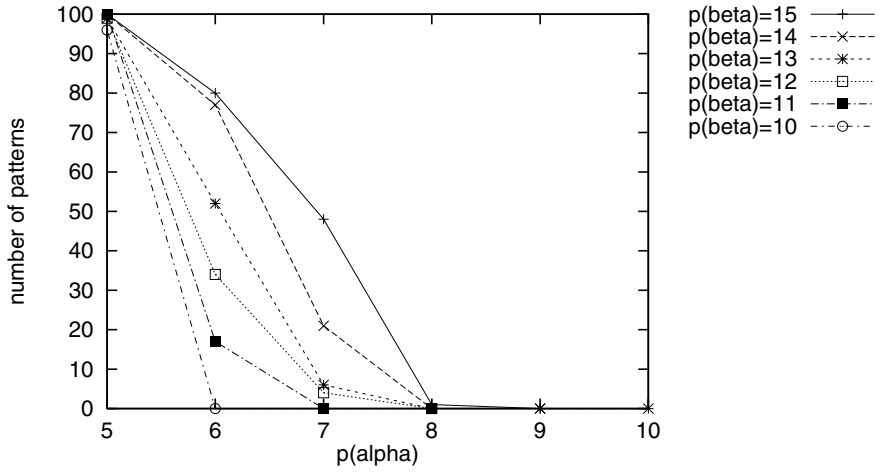


(c) 部分グラフ同型非検出パターン数

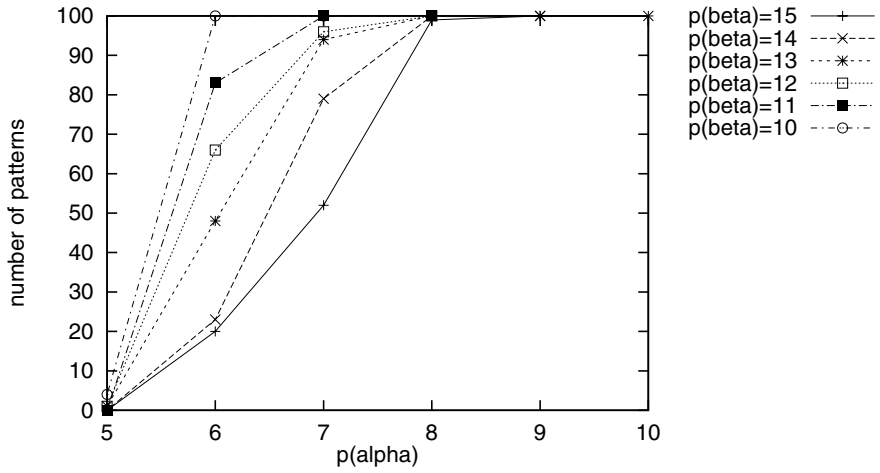
図 48: $(ed_\alpha, ed_\beta) = (0.19811, 0.39760)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00231)$ の場合



(a) 選択法と Ullmann のアルゴ リズムの実行時間比

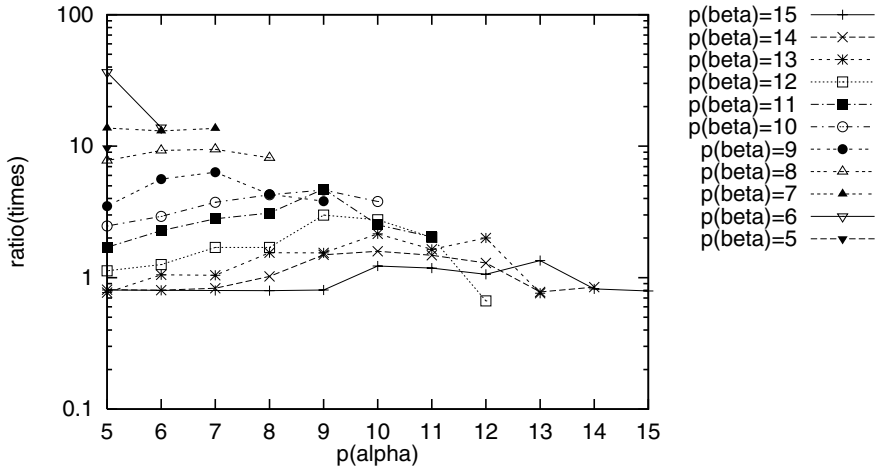


(b) 部分グラフ同型検出パターン数

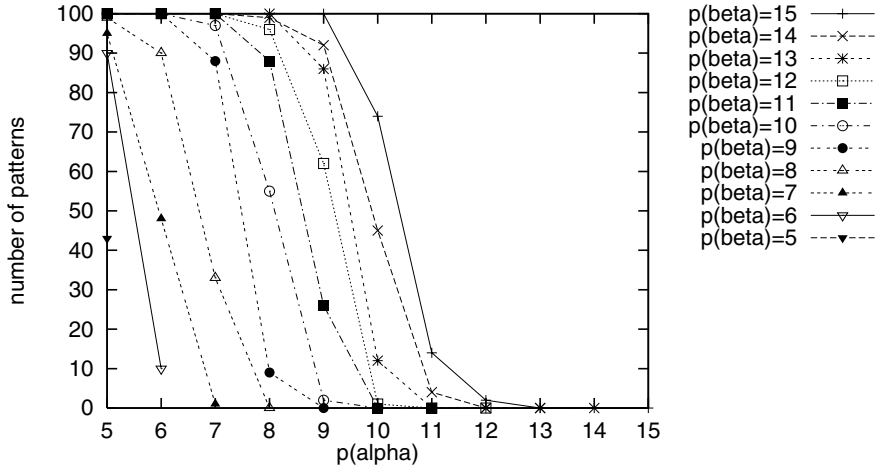


(c) 部分グラフ同型非検出パターン数

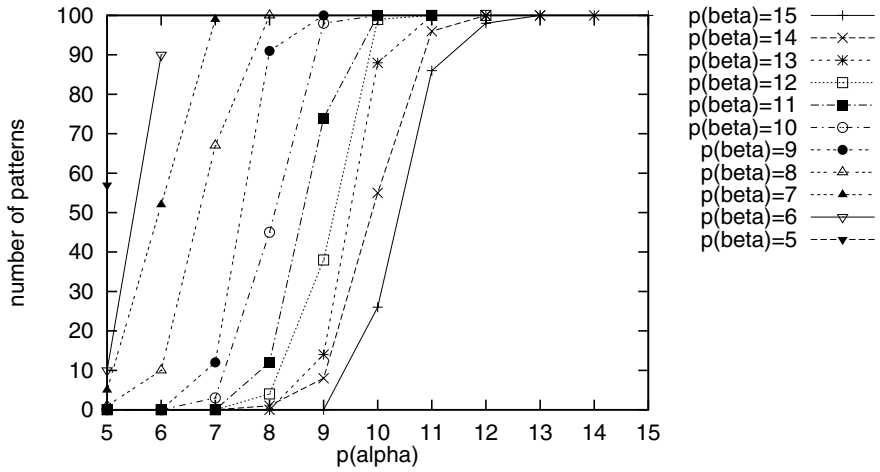
図 49: $(ed_\alpha, ed_\beta) = (0.39345, 0.19772)$, $(s.d._\alpha, s.d._\beta) = (0.00756, 0.00278)$ の場合



(a) 選択法と Ullmann のアルゴリズムの実行時間比

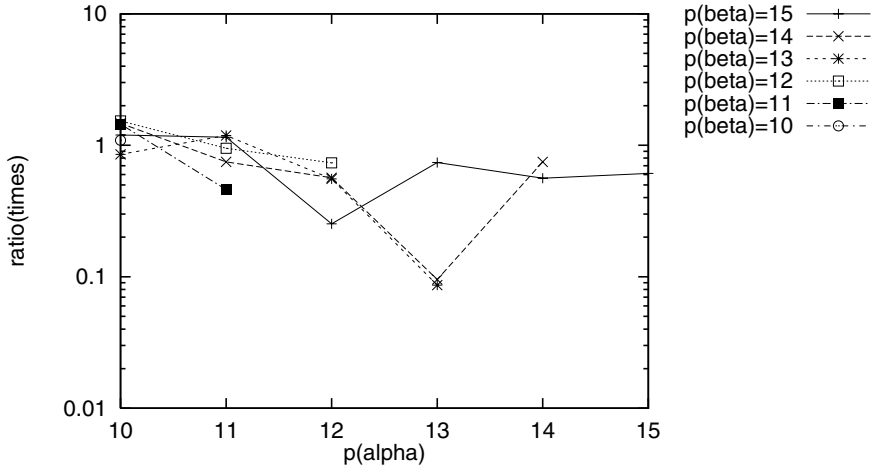


(b) 部分グラフ同型検出パターン数

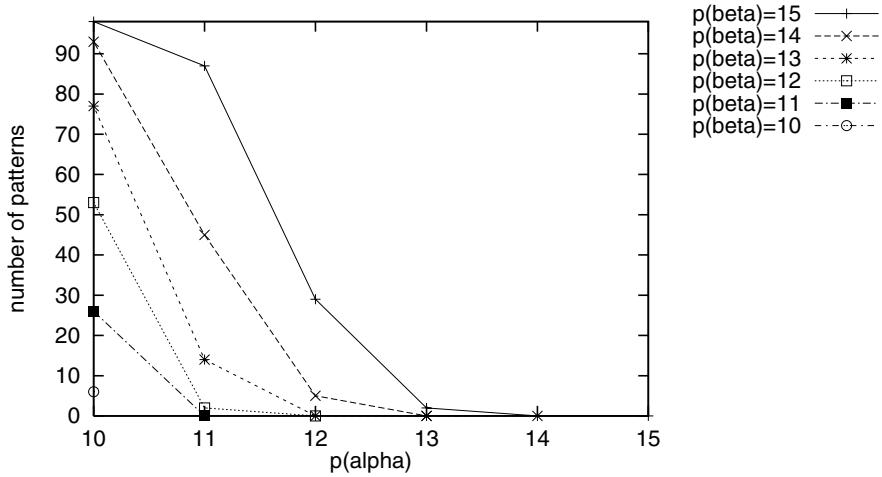


(c) 部分グラフ同型非検出パターン数

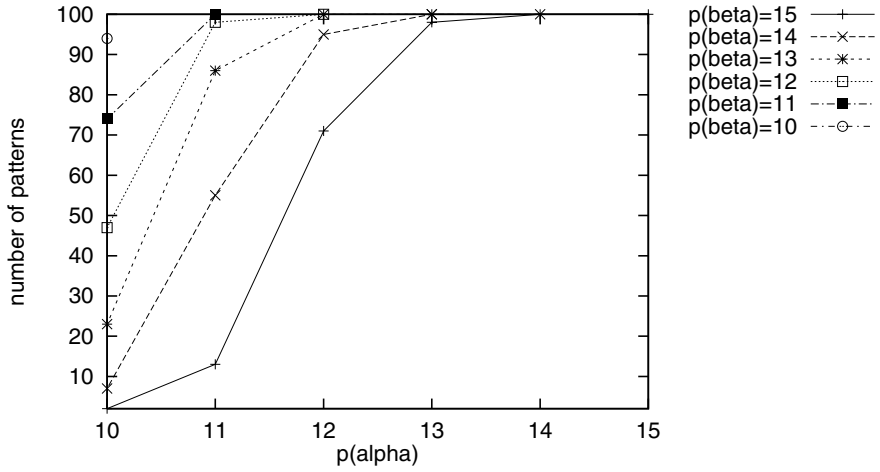
図 50: $(ed_\alpha, ed_\beta) = (0.39474, 0.39611)$, $(s.d._\alpha, s.d._\beta) = (0.00665, 0.00470)$ の場合



(a) 分担法と Ullmann アルゴリズムの実行時間比

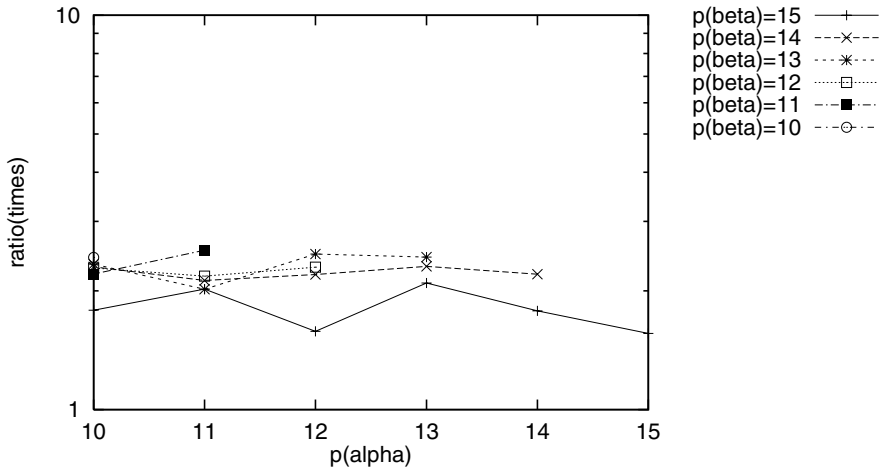


(b) 部分グラフ同型検出パターン数

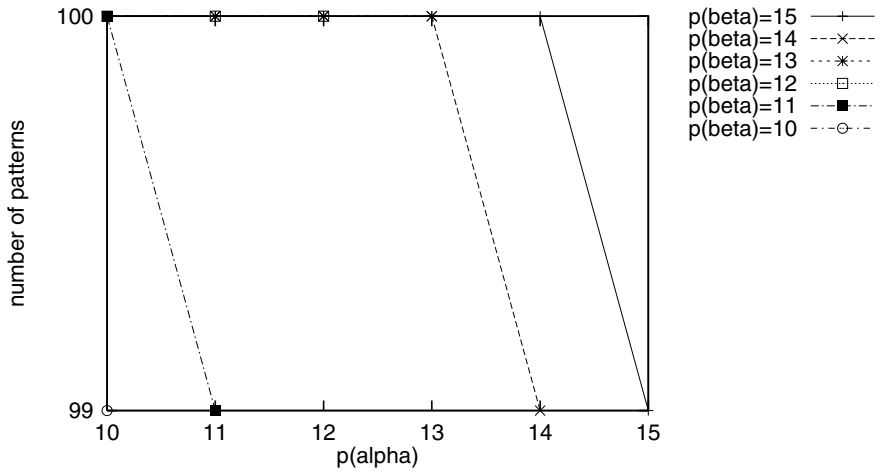


(c) 部分グラフ同型非検出パターン数

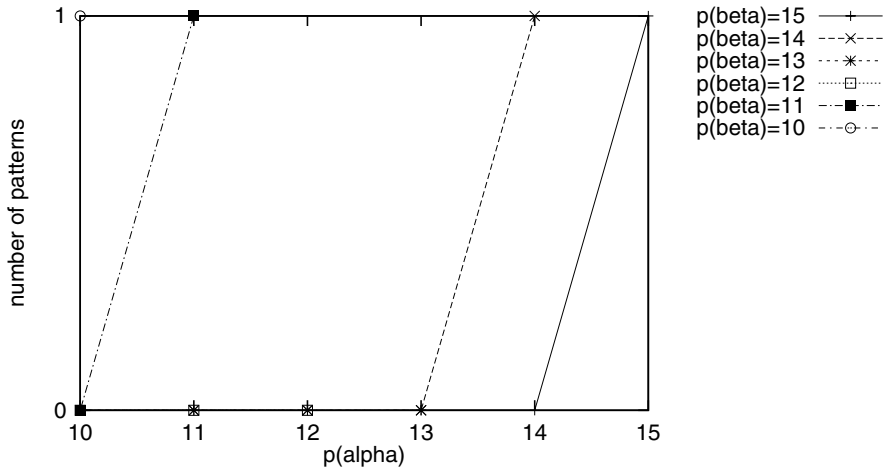
図 51: $(ed_\alpha, ed_\beta) = (0.19811, 0.19758)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00280)$ の場合



(a) 分担法と Ullmann のアルゴ リズムの実行時間比

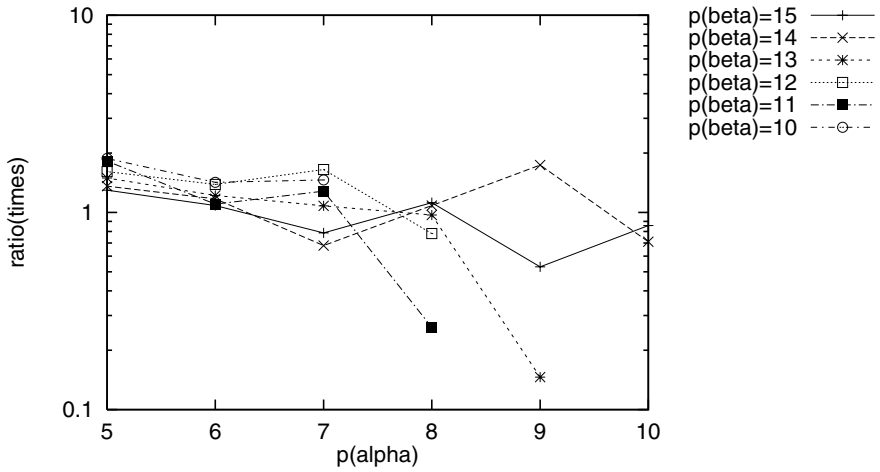


(b) 部分グラフ同型検出パターン数

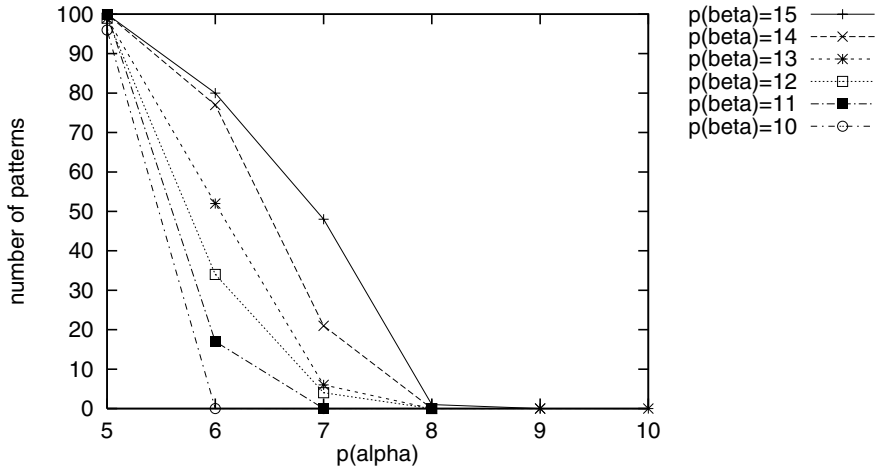


(c) 部分グラフ同型非検出パターン数

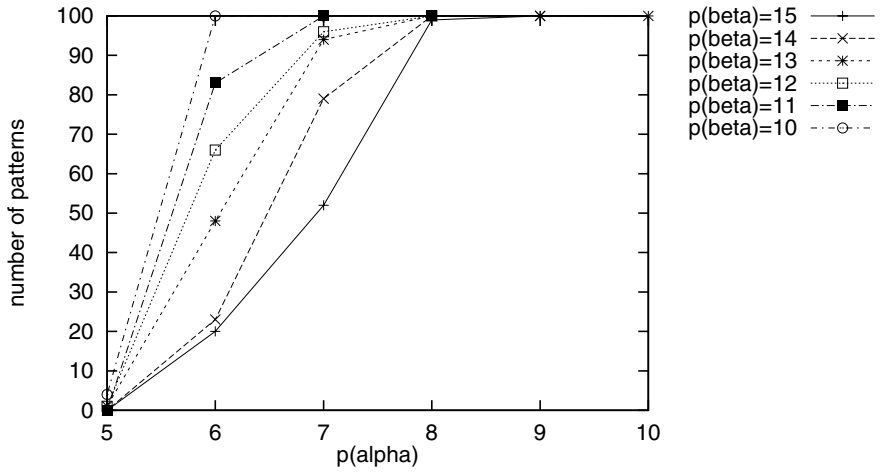
図 52: $(ed_\alpha, ed_\beta) = (0.19811, 0.39760)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00231)$ の場合



(a) 分担法と Ullmann のアルゴ リズムの実行時間比

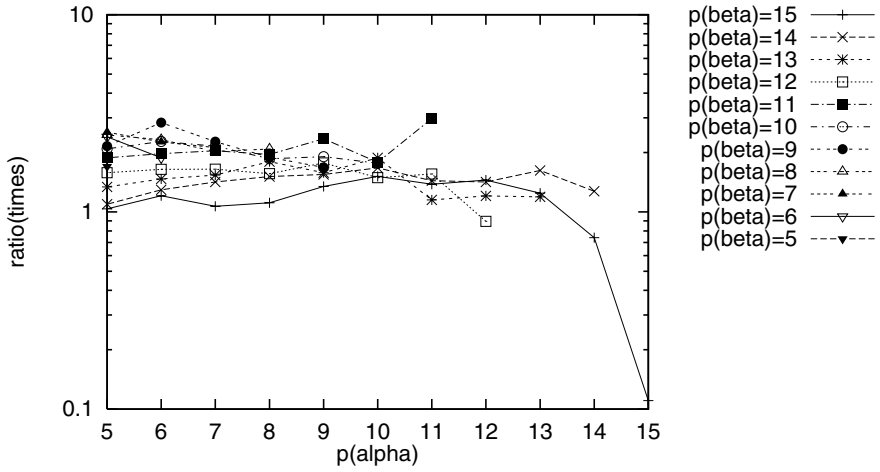


(b) 部分グラフ同型検出パターン数

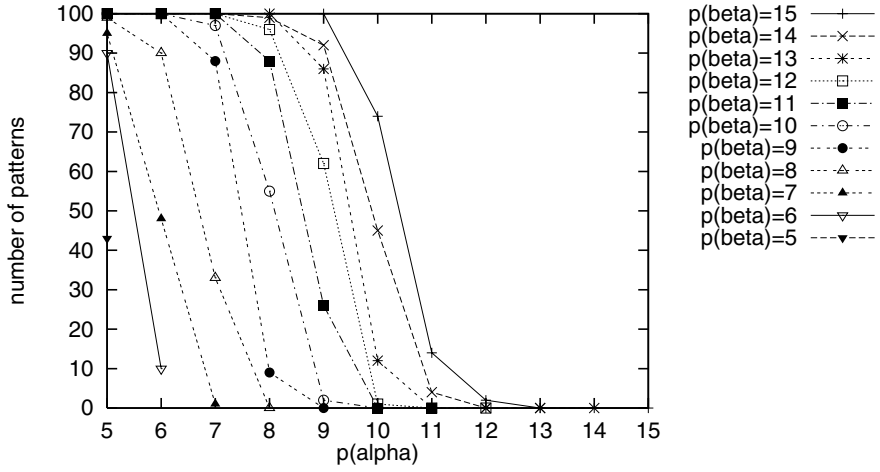


(c) 部分グラフ同型非検出パターン数

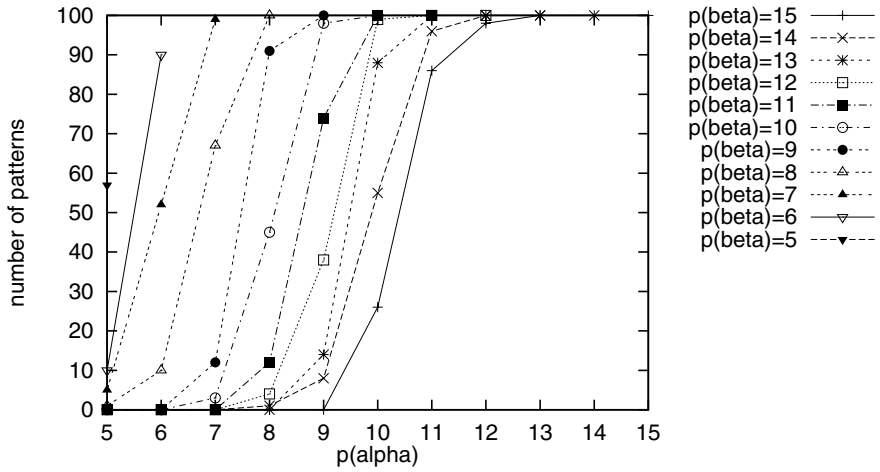
図 53: $(ed_\alpha, ed_\beta) = (0.39345, 0.19772)$, $(s.d._\alpha, s.d._\beta) = (0.00756, 0.00278)$ の場合



(a) 分担法と Ullmann のアルゴリズムの実行時間比

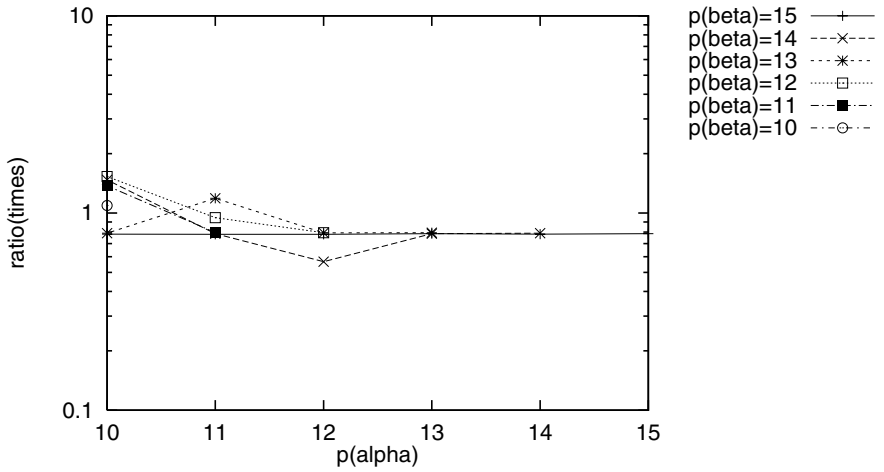


(b) 部分グラフ同型検出パターン数

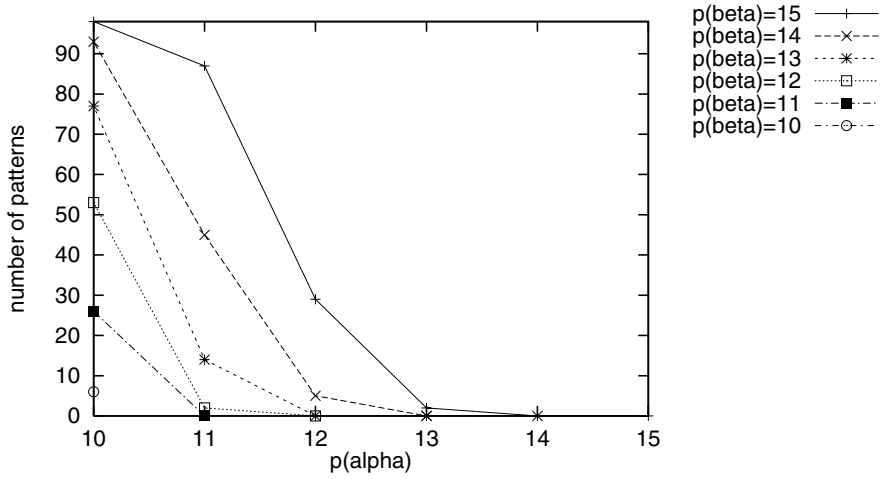


(c) 部分グラフ同型非検出パターン数

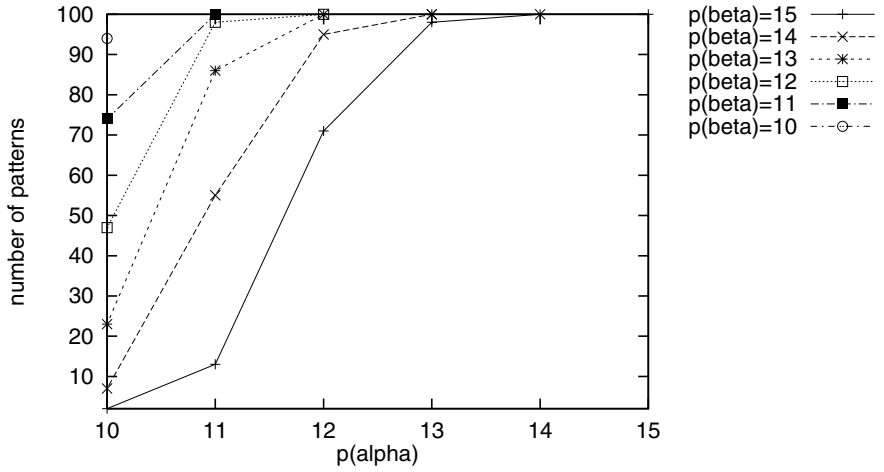
図 54: $(ed_\alpha, ed_\beta) = (0.39474, 0.39611)$, $(s.d._\alpha, s.d._\beta) = (0.00665, 0.00470)$ の場合



(a) 協調法と Ullmann アルゴリズムの実行時間比

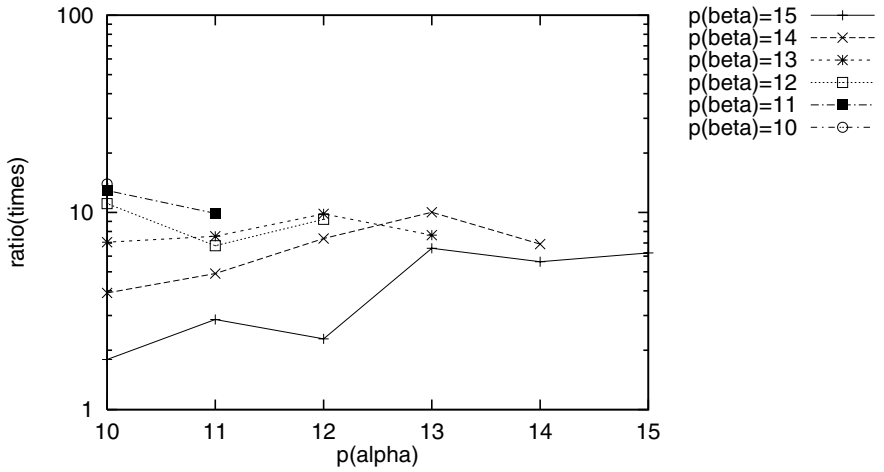


(b) 部分グラフ同型検出パターン数

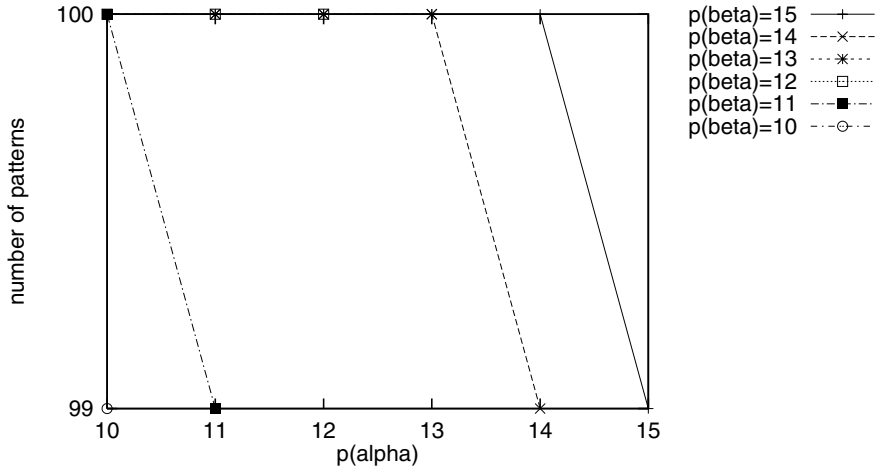


(c) 部分グラフ同型非検出パターン数

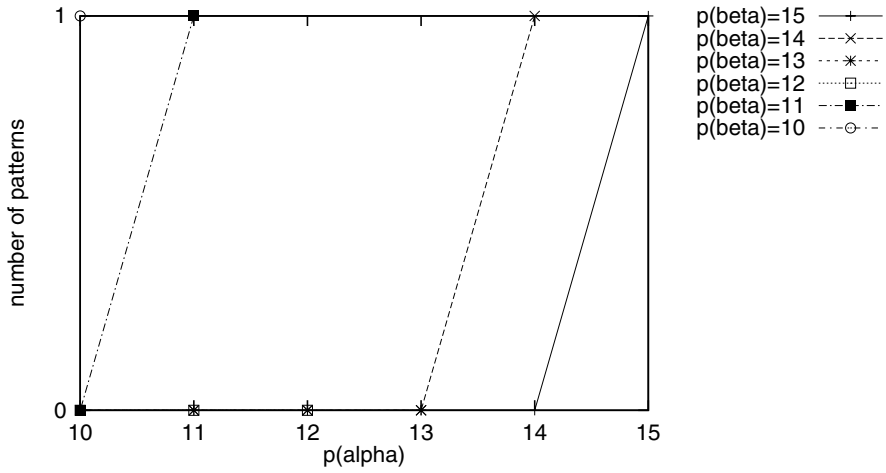
図 55: $(ed_\alpha, ed_\beta) = (0.19811, 0.19758)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00280)$ の場合



(a) 協調法と Ullmann のアルゴ リズムの実行時間比

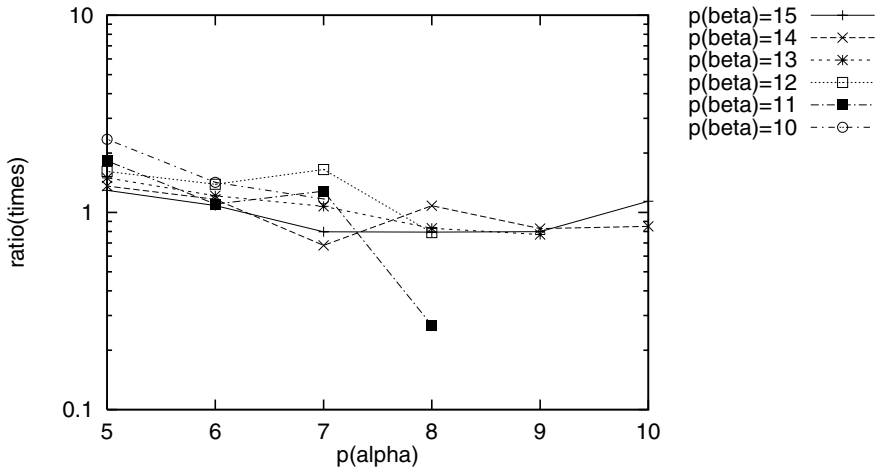


(b) 部分グラフ同型検出パターン数

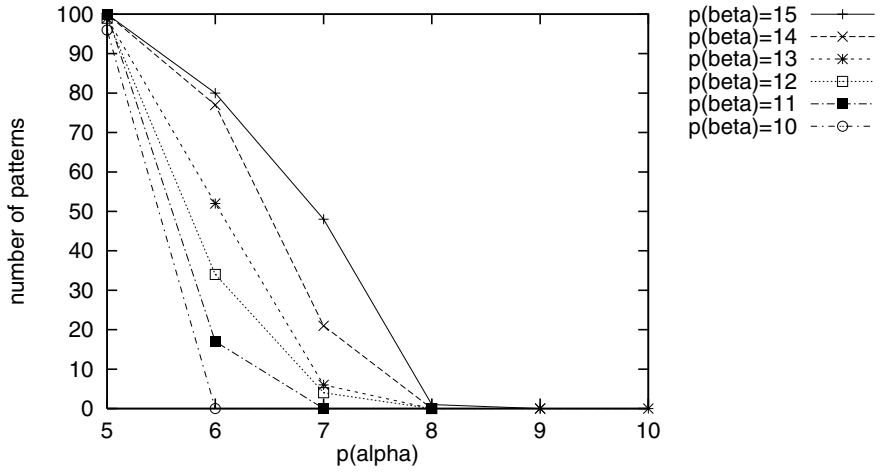


(c) 部分グラフ同型非検出パターン数

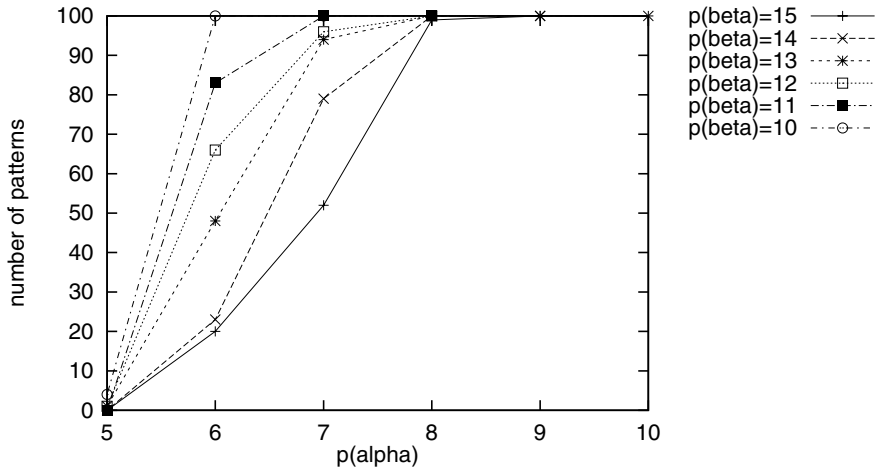
図 56: $(ed_\alpha, ed_\beta) = (0.19811, 0.39760)$, $(s.d._\alpha, s.d._\beta) = (0.00267, 0.00231)$ の場合



(a) 協調法と Ullmann のアルゴ リズムの実行時間比

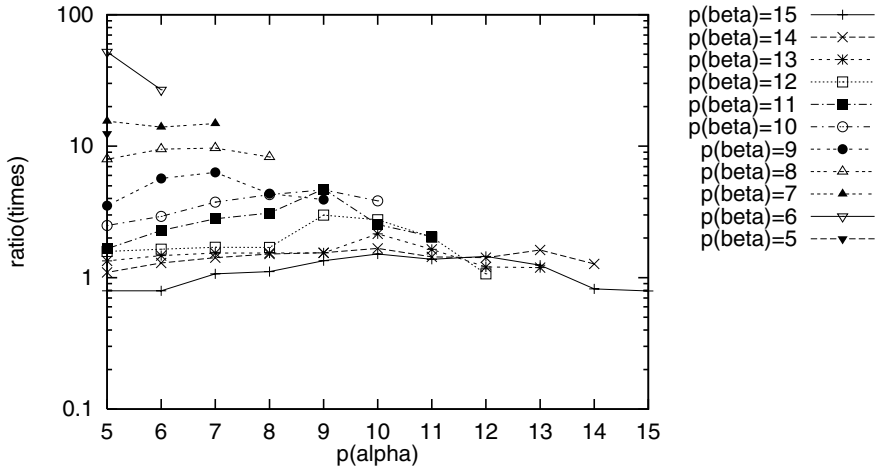


(b) 部分グラフ同型検出パターン数

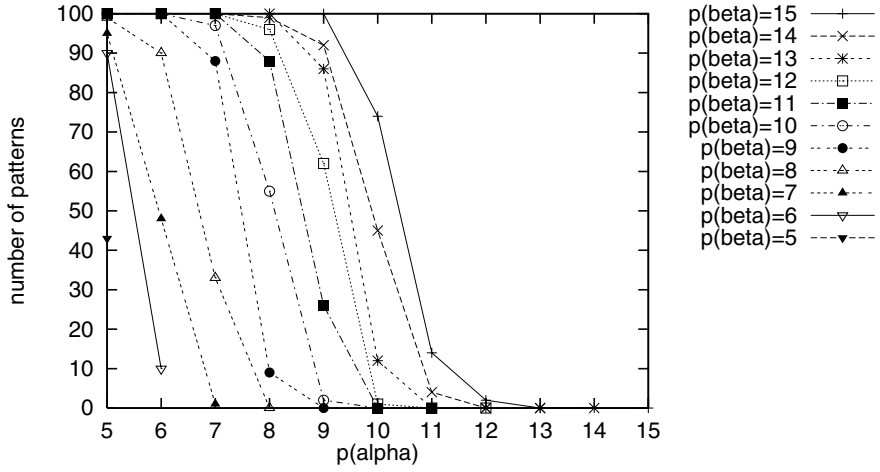


(c) 部分グラフ同型非検出パターン数

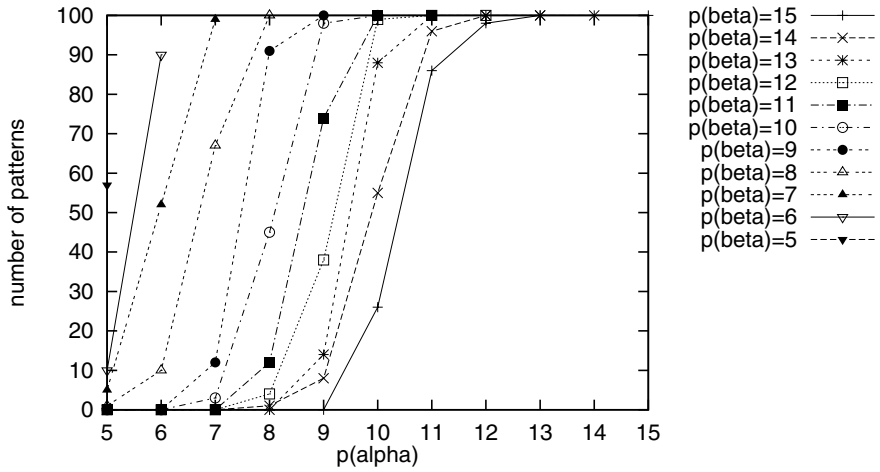
図 57: $(ed_\alpha, ed_\beta) = (0.39345, 0.19772)$, $(s.d._\alpha, s.d._\beta) = (0.00756, 0.00278)$ の場合



(a) 協調法と Ullmann のアルゴリズムの実行時間比



(b) 部分グラフ同型検出パターン数



(c) 部分グラフ同型非検出パターン数

図 58: $(ed_\alpha, ed_\beta) = (0.39474, 0.39611)$, $(s.d._\alpha, s.d._\beta) = (0.00665, 0.00470)$ の場合

6 おわりに

本論文では，OPERLボード上の OR2C15 FPGA に小西のアルゴリズムを実装した．今回の実装では，$(p_\alpha, p_\beta) = (15, 15)$ まで扱える回路であり，1 つの OR2C15A に 2 ユニットの小西のアルゴリズムの実行回路を搭載できることを示した．16.5MHz で動作している 2 ユニットを並列に実行し，Ullmann のアルゴリズムのソフトウェア (Pentium II 400MHz で実行した場合) より，10〜40 倍の性能向上が得られた．

また，専用回路の性能が低下する場合もあるが，本研究ではハードウェアとソフトウェアの協調という方法を用いて，専用回路の性能低下を避けながら，より一層の性能向上が可能であることを示した．

今回実装した回路は単純なマルチサイクル方式であるが，さらに性能を向上するために回路をパイプライン化する方法がまだ残されている．また，より大規模な FPGA に複数のユニットを搭載することや，複数の OPERL ボードを使用することなどによって，さらに性能向上を期待できる．

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] 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.

[5] 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.

[6] 小西幸治, 市川周一: FPGA に対する部分グラフ同型判定アルゴリズムの実装手法の提案, 1999 年電子情報通信学会総合大会 (一般講演 A-3-2), 1999 年 3 月.

[7] 齋藤秀充，ラターナセンタン ウドーン，市川周一: Ullmann のアルゴリズムのハードウェアによる実装に関する研究，2000 年電子情報通信学会総合大会, 2000.

[8] Lucent Technologies Inc.: Lucent Techologies ORCA^{TM} Foundry Library Manual Version 9.35, Lucent Technologies Inc., U.S.A., 1999.

[9] Lucent Technologies Inc.: ORCA^{TM} シリーズ FPGA 製品の PCI ターゲットバス・アプリケーション, Lucent Technologies Inc., U.S.A., 1996.

[10] 河西朝雄: C プログラミングの定石，技術評論社，1993.

謝辞

最後まで御指導頂いた市川周一先生に，この場を借りて御礼申し上げます．また 2 年間私の親代わりにいろいろの面を世話して下さったロータリー米山奨学会をはじめ，蒲郡ロータリーグラブの皆様，カウンセラーの竹内様，清水様に，この場を借りて御礼申し上げます．そして 2 年という短い時間でしたが，楽しい時や苦しい時を共に過ごした齋藤秀充君をはじめ，市川研究室の皆様に感謝いたします．最後に 7 年半にわたり日本での留学時に心の支えとなった家族に感謝いたします．

付録

A　インタフェース回路の設計と実装

この節では小西のアルゴリズムの実行回路を OPERL ボード [4] に実装する場合に，OPERL ボードに入出力データを受け渡すインタフェース回路の設計と実装について説明する．

A.1　入出力データの見積り

まず，結論から述べると，入出力データの見積りにより必要のバイト数の合計は 256 バイト以内に納まることがわかった．従って，小西のアルゴリズムの実行回路と OPERL ボードとの間にデータを入出力するインタフェースは I/O ウィンドウを用いて実現することにする．以下は，その理由と入出力データの見積りについての詳細を説明する．

小西のアルゴリズムの実行回路では，メモリ IM,M,B,ETSA,ETS(A),ETS(B) に初期値のデータを与えてから，回路の実行が始まる．これらのメモリを初期化するために，ユーザ側が OPERL ボードに初期値の入力データを予め渡す必要がある．また，回路から出力された結果 (OPERL ボードからの出力) も受け取る必要がある．これらの入出力データを受け渡すために，OPERL ボードでは 2 つのタイプのウィンドウ空間から選んで受け渡すことができる．一つは 256 バイトのサイズを持つ I/O ウィンドウであり，もう一つは 1M バイトのサイズを持つメモリ・ウィンドウである．論文 [4] ではこれらのウィンドウを用いてデータ転送性能測定を行っていた．この論文 [4] からわかるように，入出力データが小さい時，I/O ウィンドウはレジスタの読み書きなどに適している．

- 入力データ

  メモリ IM,M,B,ETSA,ETS(A),ETS(B) に対して必要な初期値の入力データのバイト数を計算してみた．その結果を表 18 にまとめて示す．

表 18: $2 \leq p_\alpha, p_\beta \leq 15$ の場合，それぞれのメモリの必要なバイト数

メモリ名	メモリサイズ (アドレス幅 × データ幅)	実現するのに必要とする メモリ 16×4 ビットの個数 (個)	必要なバイト数 (バイト)
M/IM	4×16	4	32
B	4×16	4	32
ETSA	4×8	2	16
ETS(A)	7×4	8	64
ETS(B)	7×4	8	64

表 18 より，$2 \leq p_\alpha, p_\beta \leq 15$ の場合に対してそれぞれのメモリを初期化するために必要なバイト数の合計は 208 バイトである．また，回路全体を制御する信号 Reset main, Start main と深さ p_α の必要なバイト数の合計は 1 バイト以内である．

- 出力データ

  小西のアルゴリズムの実行回路から出力は，8 つの信号線からなる．そのうちの 4 つはユニット 0 の出力信号であり，残りの 4 つはユニット 1 の出力信号である．ユニット 0 とユニット 1 の出力信号の構成は同じであるので，ユニット 0 のみの出力について説明する．一つ目は，部分グラフ同型が発見された数を表す 13 ビットの出力信号 (count unit0) である．二つ目は，部分グラフ同型が発見された数のキャリーアウトを表す 1 ビットの出力信号 (carry0) である．三つ目は，部分グラフ同型が発見されたことを表す 1 ビットの出力信号 (found0) である．最後の四つ目は，回路の実行が終了したことを表す 1 ビットの出力信号 (end unit0) である．ユニット 0 の出力ビット数の合計は 16 ビットである．従って，2 つのユニットの必要のビットの合計は 32 ビットである．

A.2 入出力データと I/O ウィンドウとの関係

A.2.1 入力データの場合

256 バイトの I/O ウィンドウと入力データとの関係設定を図 59 に示す．図中で縦軸はアドレス，横軸はデータのビット幅を示している．同図より，I/O ウィンドウを縦に 4 つの大きなブロックに分ける．各ブロックのサイズは 32 ビット幅 ×4 ビットアドレスとする．ここで，各ブロックについて上から順に「ブロック 1」,「ブロック 2」,「ブロック 3」,「ブロック 4」と呼ぶ．以下，それぞれのブロックについて説明する．

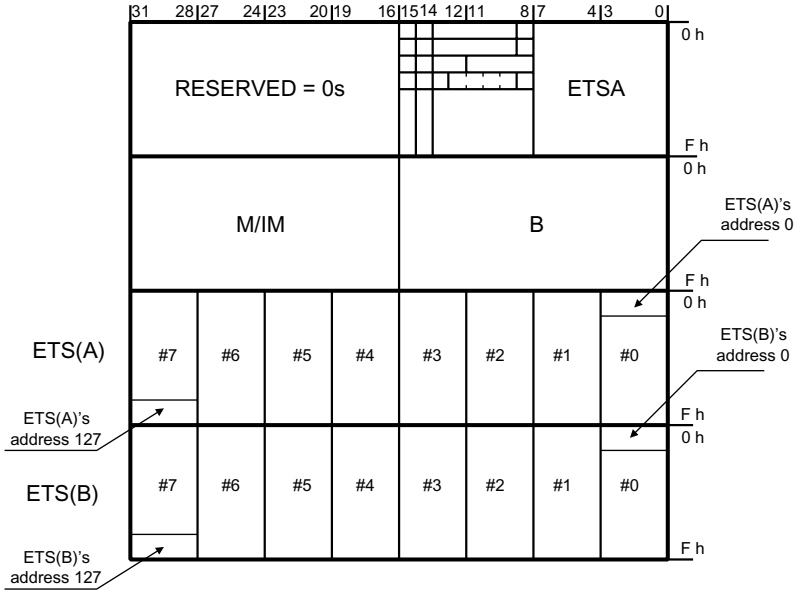


図 59: I/O ウィンドウと入力データの関係図

- 「ブロック 1」(図 60 を参照)

  ブロック 1 には ETSA のデータ，制御信号 (Reset_unit, Start_unit)，深さ p_α，ユニットを選択する制御信号 (unit_en)，全てのユニットをリセットさせる信号 (reset_system) が格納される．それぞれの使用範囲は以下の通りである．

  - アドレス 0〜Fh の 0〜7 ビット目に ETSA の初期値の入力データが格納される．アドレス 0h は小西のアルゴリズムの都合により必ず 0 が格納されるようにする．アドレス 1h では ETSA の初期値の入力データの 1 番目に格納される．アドレス Fh は ETSA の初期値の入力データの 15 番目に格納されている．
  - アドレス 0h の 8 ビット目に Reset_unit 信号のデータが格納される．
  - アドレス 1h の 8 ビット目に Start_unit 信号のデータが格納される．
  - アドレス 2h の 8〜11 ビット目に p_α のデータが格納される．
  - アドレス 3h の 9〜12 ビット目に unit_en 信号のデータが格納される．unit_en が $(0001)_2$ であれば，ユニット 0 が選択されるという意味である．$(0010)_2$ であれば，ユニット 1 が選択されるという意味である．
  - アドレス 3h の 8 ビット目に reset_system 信号のデータが格納される．reset_system が 1 であれば，全てのユニットがリセットされる．
  - アドレス 0〜Fh の 15 ビット目に 0 か 1 が格納される．このビットはこのブロック 1 の中にどの部分を使用するかの制御信号である．1 であれば，0〜7 ビット目までのデータを使用する．0 であれば，0〜7 ビット目までのデータを使用しない．

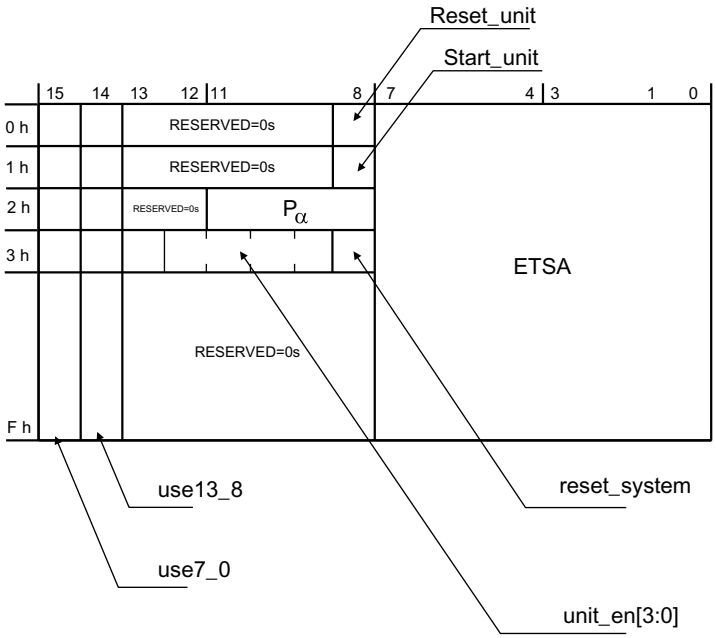


図 60: ブロック 1 の詳細

- アドレス 0〜Fh の 14 ビット目に 0 か 1 が格納される．このビットはこのブロック 1 の中にどの部分を使用するかの制御信号である．1 であれば，8〜13 ビット目までのデータを使用する．0 であれば，8〜13 ビット目までのデータを使用しない．
- 上記に述べたところ以外は，予約として 0 が格納される．

- 「ブロック 2」

  ブロック 2 には M/IM と B のデータが格納される．

  - 0〜15 ビット目に B の初期値の入力データが格納される．アドレス 0h では，小西のアルゴリズムの都合により必ず 0 が格納されるようにする．アドレス 1〜Fh の 0〜15 ビット目には 15×15 の隣接行列 B に相当する．15×15 より小さな隣接行列を入力する場合は，使われていないところに必ず 0 が格納されるようにする．
  - 16〜31 ビット目に M と IM の初期値の入力データが格納される．アドレス 0h では，小西のアルゴリズムの都合により必ず 0 が格納されるようにする．アドレス 1〜Fh の 16〜31 のビット目には 15×15 の隣接行列 M/IM に相当する．15×15 より小さな隣接行列を入力する場合は，使われていないところに必ず 0 が格納されるようにする．

- 「ブロック 3」

  ブロック 3 には ETS(A) のデータが格納される．ETS(A) は格納できるデータの最大数が 128 個であり，8 個の 16x4 ビットメモリとマルチプレクサなどからなるものである．ETS(A) の初期化時間を短縮するために，8 個の 16x4 ビットメモリを同時に初期化することによって実現できる．このために，0〜31 ビットまでのデータ幅を 4 ビット単位で分けると，8 個の 16x4 ビットのデータ列を作成できる．それぞれの使用範囲は以下の通りである．

  - 0〜3 ビット目はブロック#0 という．このブロック#0 は，ETS(A) の 0〜15 番目の初期値入力データが格納される．つまり，ブロック#0 は 1 個目の 16x4 ビットメモリに相当する．
  - 4〜7 ビット目はブロック#1 という．このブロック#1 は，ETS(A) の 16〜31 番目の初期値入力データが格納される．つまり，ブロック#1 は 2 個目の 16x4 ビットメモリに相当する．

- 8〜11 ビット目はブロック#2 という．このブロック#2 は，ETS(A) の 32〜47 番目の初期値入力データが格納される．つまり，ブロック#2 は 3 個目の 16x4 ビットメモリに相当する．
- 12〜15 ビット目はブロック#3 という．このブロック#3 は，ETS(A) の 48〜63 番目の初期値入力データが格納される．つまり，ブロック#3 は 4 個目の 16x4 ビットメモリに相当する．
- 16〜19 ビット目はブロック#4 という．このブロック#4 は，ETS(A) の 64〜79 番目の初期値入力データが格納される．つまり，ブロック#4 は 5 個目の 16x4 ビットメモリに相当する．
- 20〜23 ビット目はブロック#5 という．このブロック#5 は，ETS(A) の 80〜95 番目の初期値入力データが格納される．つまり，ブロック#5 は 6 個目の 16x4 ビットメモリに相当する．
- 24〜27 ビット目はブロック#6 という．このブロック#6 は，ETS(A) の 96〜111 番目の初期値入力データが格納される．つまり，ブロック#6 は 7 個目の 16x4 ビットメモリに相当する．
- 28〜31 ビット目はブロック#7 という．このブロック#7 は，ETS(A) の 112〜127 番目の初期値入力データが格納される．つまり，ブロック#7 は 8 個目の 16x4 ビットメモリに相当する．

- 「ブロック 4」

  ブロック 4 には ETS(B) のデータが格納される．ETS(B) の構造は ETS(A) と全く同じであるが，初期値の入力データが異なっている点だけである．従って，ETS(B) の初期化は ETS(A) と同様に行う．

A.2.2 デコード方法

OPERL ボードの I/O ウィンドウの使用可能なアドレス範囲は 00〜FFh である．前節で説明したブロック 1〜ブロック 4 を選択できるために，この I/O ウィンドウのアドレスを用いてデコードする必要がある．I/O ウィンドウのデコードの設定を図 61 に示す．図 61 より，0〜1 ビット目は 0 である．これは，I/O 命令を用いた場合，一度に転送できるのは 4 バイトだけであるためである．2〜5 ビット目は，各ブロックに対応しているアドレス (addr) である．addr の値は 0〜Fh までである．6〜7 ビット目は，ブロックを選択するための制御信号 (grp) である．grp が 00 であるとき，ブロック 1 が選択される．grp が 01 であるとき，ブロック 2 が選択される．grp が 10 であるとき，ブロック 3 が選択される．grp が 11 であるとき，ブロック 4 が選択される．8〜15 ビット目は E0h であるが，これはマザーボードの BIOS が割合っているものであり，デバイスドライバ経由で ioctl で知ることが可能である．

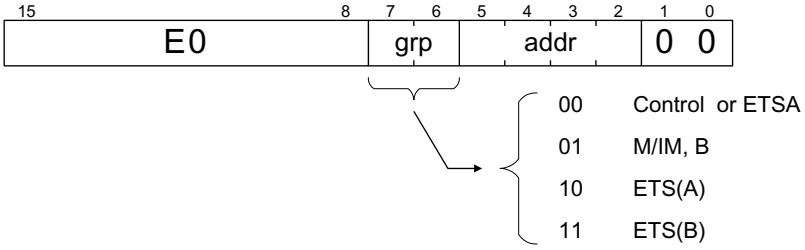


図 61: I/O ウィンドウのアドレスのデコード設定

A.2.3 出力データの場合

256 バイトの I/O ウィンドウと出力データとの関係設定を図 62 に示す．図中で縦軸はアドレス，横軸はデータのビット幅を示している．それぞれの出力信号と配置は以下の通りである．

- 1 または 2 ユニットの場合
  - アドレス 0h の 0 ビット目にユニット 0 の end unit0 信号のデータが格納される．
  - アドレス 0h の 1 ビット目にユニット 0 の found0 信号のデータが格納される．

– アドレス 0h の 2 ビット目にユニット 0 の carry0 信号のデータが格納される．
– アドレス 0h の 3 ビット目にユニット 1 の end unit1 信号のデータが格納される．
– アドレス 0h の 4 ビット目にユニット 1 の found1 信号のデータが格納される．
– アドレス 0h の 5 ビット目にユニット 1 の carry1 信号のデータが格納される．
– アドレス 0h の 6〜18 ビット目にユニット 0 の count unit0 信号のデータが格納される．
– アドレス 0h の 19〜31 ビット目にユニット 1 の count unit1 信号のデータが格納される．
– 上記に述べたところ以外は要約として，扱わないとする．

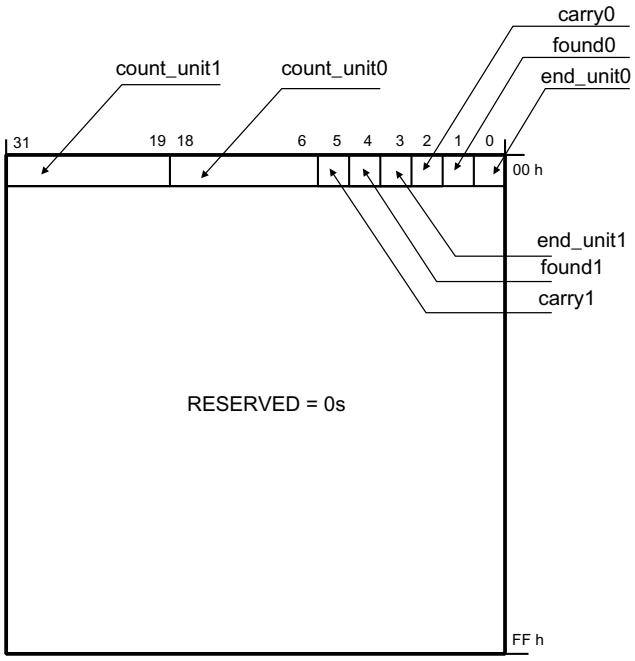


図 62: I/O ウィンドウと 2 ユニットの出力データの関係図

• 3 または 4 ユニットの場合 (図 63 を参照)

– ユニット 0 の出力
  ∗ アドレス 0h の 0 ビット目に end main 信号のデータが格納される．
  ∗ アドレス 0h の 1 ビット目に found 信号のデータが格納される．
  ∗ アドレス 0h の 2 ビット目に over flow 信号のデータが格納される．
  ∗ アドレス 0h の 16〜31 ビット目に count found 信号のデータが格納される．
– ユニット 1 の出力は，基本的にユニット 0 と同じ配置であるが，使用されているアドレスは 1h である．
– ユニット 2 の出力は，基本的にユニット 0 と同じ配置であるが，使用されているアドレスは 2h である．
– ユニット 3 の出力は，基本的にユニット 0 と同じ配置であるが，使用されているアドレスは 3h である．
– 上記に述べたところ以外は要約として，扱わないとする．

A.2.4 インタフェース回路の実装

実装したインタフェース回路を図 64 に示す．このインタフェース回路は，1 から 4 ユニットまでの小西のアルゴリズムの実行回路に入出力データを受け渡すことができる．1 つのユニットとインタフェース回路の詳細関係を図

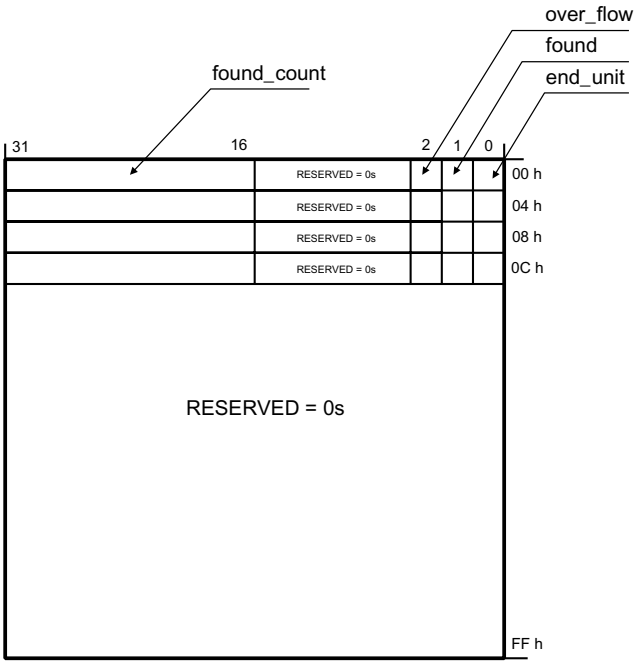


図 63: I/O ウィンドウと 4 ユニットの出力データの関係図

65 に示す．図中で黒太い縦線の左側にある全ての論理は，雛方ファイルに記述している基本の論理である．但し，datvldRR のフリップフロップ，datvldRR とつながっている和論理，clk_d2 を出力フリップフロップ，sel_unit_en とつながっているフリップフロップが書き加えられたものである．この分部の論理回路 (左側の論理回路) は，PCI バスと同じクロックで動作している (33MHz)．一方，黒太い縦線の右側にある全ての論理はバスの倍周期で動作している (16.5MHz). 以下，黒太い縦線の右側にある論理回路について説明する．

- XXX_c2 のフリップフロップ (XXX は文字列である)

  PCI バスクロックで動作している回路 (黒太い縦線の左側) から出力されたデータをラッチする．以下それぞれの信号線についての意味を説明する．

  - ldiRR_c2

    PCI FPGA から出力された 32 ビットの入力データ・バスである．

  - rdwrRR_c2

    メモリの書き込みまたは呼び出し信号である．信号レベル“ H ”の時に書き込み，レベル“ L ”の時に呼び出しである．

  - ldaRR_c2

    PCI FPGA から出力された 32 ビットのアドレス・バスである．本研究では，0〜7 ビット目までをデコードすることで，ブロック 1〜4 を選択する．

  - datvldRRR_c2

    ライト・サイクルでは，Pldio 上の有効データを示す信号線である．この信号レベル“ H ”の時に Pldio 上の有効データが現れていることを示す．

  - memioRR_c2

    I/O ウィンドウとメモリ・ウィンドウのどちらのモードが選択されるかを示す信号線である．信号レベル“ H ”の時に I/O ウィンドウを使用する．

  - unit_enRR_c2

    選択されるユニットにアクティブする．信号レベル“ H ”の時に，ユニットにアクティブする．

- 2 to 4 DeMUX with Enable(破線枠の外)

  得られたアドレスをデコードし，どのメモリを初期化するかを振り分ける．このデマルチプレクサからの出力信号の意味を以下に説明する．

  - ctrle

    メモリ ETSA，P_α, 回路のリセットのうち初期化を行うイネーブル信号である．

  - MBwe

    メモリ M,B を初期化するイネーブル信号である．

  - ETS_awe

    メモリ ETS(A) を初期化するイネーブル信号である．

  - ETS_bwe

    メモリ ETS(B) を初期化するイネーブル信号である．

- 2 to 4 DeMUX with Enable(破線枠の中)

  得られた入力データをデコードし，回路のリセット，回路の実行開始，レジスタ P_α の書き込みイネーブルのうち選択する．このデマルチプレクサからの出力信号の意味を以下に説明する．

  - Resete

    回路のリセット・イネーブル信号である．

  - Starte

    回路の実行開始のイネーブル信号である．

  - Pawe

    レジスタ P_α の書き込みイネーブル信号である．

- memSel

  初期化を行うメモリを選択する．

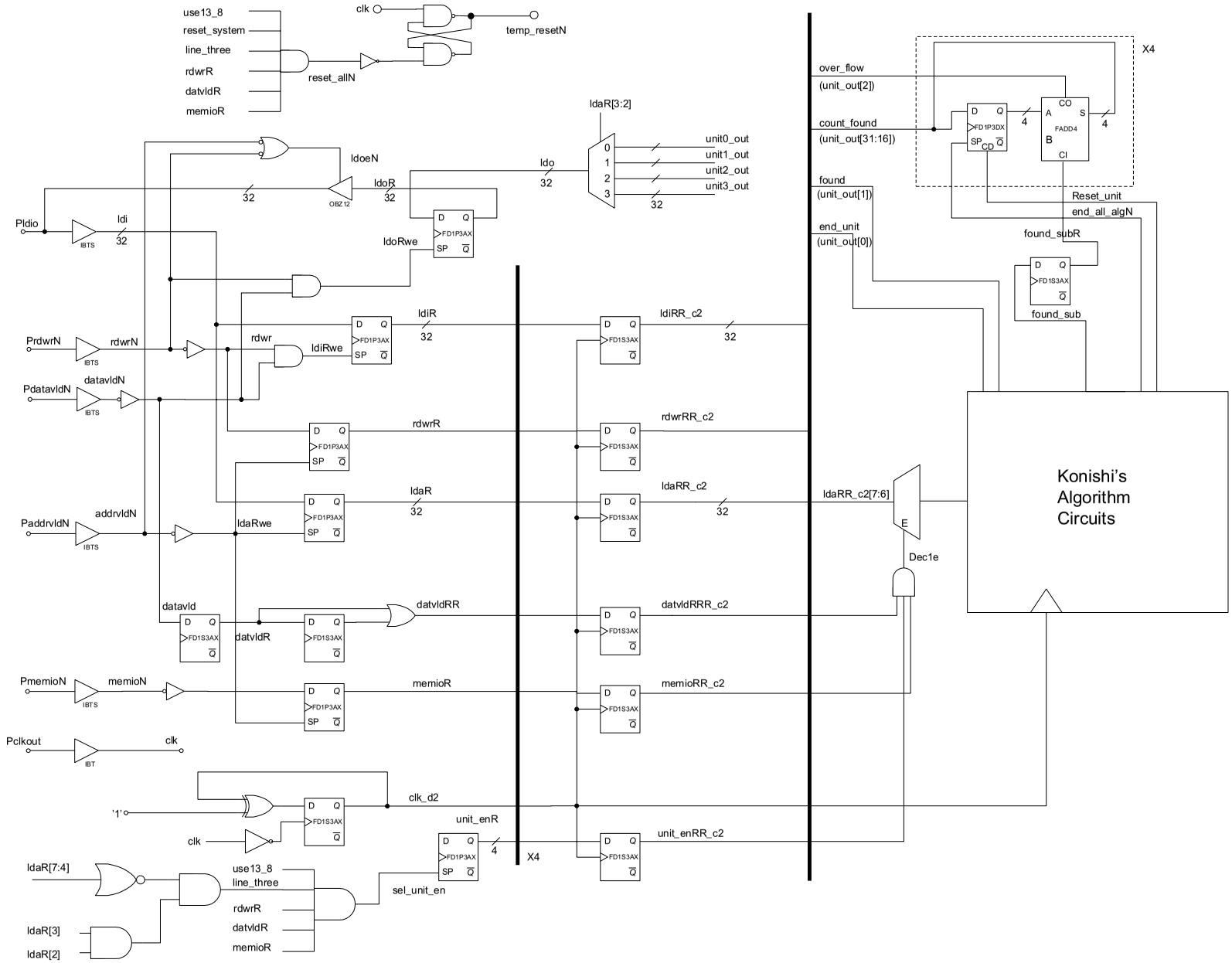


図 64: I/O ウィンドウを用いたインタフェース回路

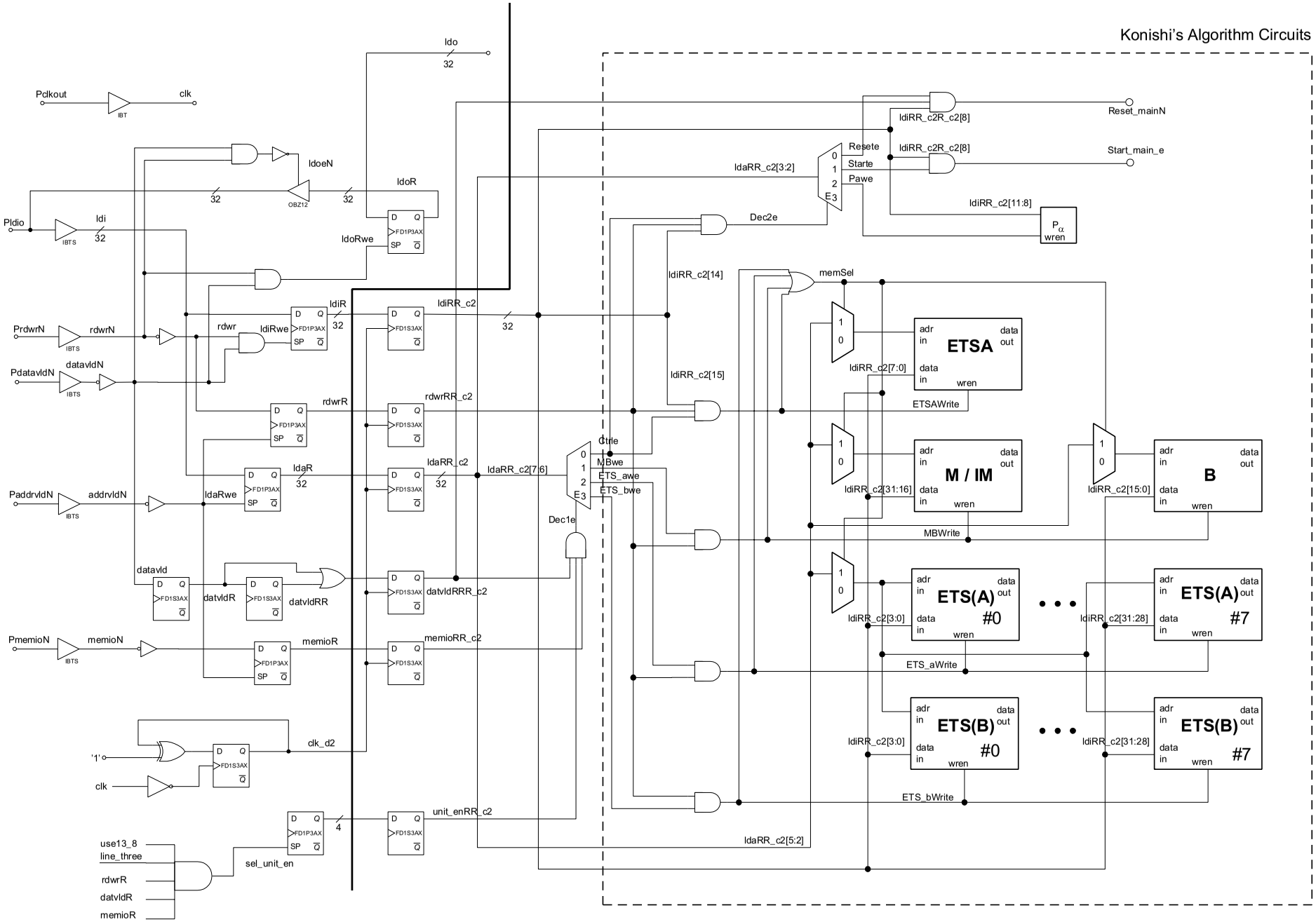


図 65: 1 つのユニットとインタフェース回路の詳細関係