学位申請日 平成 16 年 1 月 13 日

知識情報 工学専攻	学籍番号	**003317**
申請者氏名	**山本 昌治**	

指導教官氏名	**市川 周一**

論 文 要 旨 (修士)

論文題目	**部分グラフ同型判定のためのデータ依存回路**

部分グラフ同型判定は化学物質の構造活性推測など幅広い応用を持つが, 一般に NP 完全で計算は困難である．部分グラフ同型判定の専用回路には，Ullmann の提案と小西の提案があるが，どちらの提案も回路規模の制限により実装可能な頂点数には限界がある．

本研究では、部分グラフ同型判定のためのデータ依存回路について述べる．評価では，実際に存在する FPGA ボードを利用して実測した値を用いた．データ依存回路とは入力データ毎に最適化した論理回路である．データ依存回路は元になった論理回路よりも回路規模が小さく，信号遅延が小さいため元になった回路よりも高速である．しかし，他の入力データに対して再利用が出来ないため入力データ毎に回路を生成してやる必要がある．また，FPGA のような再構成可能論理素子への実装が前提となる．

本研究では, 部分グラフ同型判定のためのデータ依存回路を設計して，Xilinx XC2V3000 FPGA に実装して評価した．25MHz システムクロックにてデータ依存回路の動作を確認した．ランダムグラフを入力グラフとした場合，最大動作周波数のときデータ依存回路の実行時間はソフトウェアの 7.0%である (頂点数 16)．回路生成時間を含めて考えてもデータ依存回路はソフトウェアの 14.4 倍高速である．頂点数が大きくなれば，データ依存回路の優位性はより高くなる．

本研究では Ullmann のアルゴリズムと小西のアルゴリズムの 2 つのアルゴリズムについてデータに依存した設計手法を評価して，両方のアルゴリズムに対して効果があることを確認した．評価では複数のデータセットを用いて平均値を用いて結果を示した．

本研究で行ったことと得られた知見を以下に示す.

- データに依存した手法は回路生成時間を含んでも，最新プロセッサで動作させたソフトウェアより性能面で優位性がある.
- データ依存回路を Xilinx XC2V FPGA を用いて実際に実装し，FPGA ボード上の 25MHz システムクロックで動作を確認した.
- データに依存した手法の優位性は特定のデータセットやアルゴリズムによるものではない．2 つのアルゴリズムと複数のデータセットによりデータに依存した手法の優位性を確認した.

Name	**Shoji Yamamoto**	Registration Number	**003317**
Department of Knowledge-based Information Engineering			
Graduate Adviser	**Shuichi Ichikawa**		
Keywords	Subgraph Isomorphism Problem, Data-dependent Hardware, FPGA		

Abstract (Master)

Title	**Data-dependent Hardware for Subgraph Isomorphism Problem**

Subgraph isomorphism problems have various important applications, while generally being NP-complete. Though Ullmann and Konishi proposed the custom circuit designs to accelerate subgraph isomorphism problem, they require many hardware resources for large problems.

This study describes the design of data-dependent circuits for subgraph isomorphism problem with evaluation results on an actual FPGA platform. Data-dependent circuits are logic circuits specialized in specific input data. Such circuits are smaller and faster than the original circuit, although it is not reusable and involves circuit generation for each input.

In the present study, the circuits were implemented on Xilinx XC2V3000 FPGA, and they successfully operated at a clock frequency 25MHz. In the case of graphs with 16 vertices, the average execution times at maximum frequency is about 7.0% of the software executed on an up-to-date microprocessor (Athlon XP 2600+ of 2.1 GHz clock). Even if the circuit generation time is included, data-dependent circuits are about 14.4 times faster than the software (for random graphs with 16 vertices). This performance advantage becomes larger for larger graphs.

Two algorithms (Ullmann's and Konishi's) were examined, and the data-dependent approach was found to be equally effective for both algorithms. We also examined several types of input graph sets, and found that the data-dependent approach shows advantage in both cases.

The original contributions of this study are summarized as follows:

- Data-dependent approach was shown to be superior to the software on a state-of-the-art microprocessor, even if the circuit generation time is included.
- Data-dependent circuits were actually implemented on Xilinx XC2V FPGA, and successfully operated on an FPGA board at 25MHz clock.
- The superiority of data-dependent approach is not specific to an algorithm or an data set. Two different algorithm and several distinct data sets were examined, and the superiority of data-dependent approach was sustained in all these cases.

部分グラフ同型判定のためのデータ依存回路

Data-dependent Hardware for Subgraph Isomorphism Problem

2004年3月10日

豊橋技術科学大学 大学院 工学研究科 知識情報工学専攻

学籍番号 003317 氏名 山本昌治

指導教官 市川周一

目 次

図 目 次

表 目 次

1 はじめに

計算量の多い問題には高速化の要求が存在する．高速化には大きくわけて 2 つの方法，スーパーコンピュータなどの高速な汎用プロセッサを投入する方法と，専用回路による方法がある．専用回路は，専用データパスによるデータ転送幅の向上，複数の演算器やパイプライン化による並列処理などによって処理を高速化する．

専用回路は問題に特化しており他の応用に適応できないので，汎用プロセッサに比べてコストが高いという問題がある．しかし，FPGA (Field Programmable Gate Array) などの再構成可能論理素子の出現により，実装コストは低下した．また，近年 CAD (Computer Aided Design) 技術の進歩により設計コストも低下している．これらのコスト面の問題の緩和により，FPGA を用いた専用回路の研究が広く行われるようになった．

部分グラフ同型判定問題を専用回路で高速化する試みは古くから行われてきた．Ullmann[6] は，部分グラフ同型判定を高速化するために refinement procedure を提案し，refinement procedure が論理回路で実装可能であると述べた．市川ら [3] は，Ullmann の提案回路を FPGA で実装した場合の回路規模と性能を評価し，Ullmann の提案回路は回路規模が過大で実装困難であると報告した．小西ら [7][10] は，枝刈りアルゴリズムを簡略化することにより，部分グラフ同型判定専用回路の回路規模を削減することを提案した．専用回路の回路規模を削減すれば，同じ論理規模の LSI で，より大きな問題を解くことができる．小西の回路は Ullmann の回路より動作時間は長いが回路規模が小さく，入力グラフによっては面積時間積 (AT 積) は Ullmann の回路より優れている [3].

しかし，Ullmann の回路も小西の回路も巨大な入力グラフを扱う場合，実装回路規模が増加して実現は難しい．そこで本研究では，同じ回路規模ならより大きな頂点数の入力グラフを扱うことができるとされるデータ依存回路を部分グラフ同型判定問題に応用する．先行研究 [4] では Ullmann のデータ依存回路が Ullmann の回路より小さい回路規模で実現できることを報告した．しかし，先行研究 [4] は部分的な評価までで実装に至っていない．また，評価したのは Ullmann のアルゴリズムのみである．

そこで，以下の 3 点を目的として本研究ではデータ依存回路を実装して評価する．

1. 部分グラフ同型判定のデータ依存回路を FPGA 上に実装し，ボード上で動作を確認するとともに，回路規模・回路生成時間・動作時間を実測する．

2. Ullmann のアルゴリズムだけでなく，小西のアルゴリズムのデータ依存回路を実装し，データ依存回路の優位性を複数の設計で実証する．

3. 入力データ (グラフ) として，ランダムグラフや連結グラフなど特徴の違うものを用い，データ依存回路の優位性を確認する．

データに依存した実装手法は様々な応用で研究されており，グラフアルゴリズムに関する研究も幾つかある (論文 [4] の引用文献を参照のこと)．ただし，部分グラフ同型判定問題についての検討は筆者らの先行研究 [4] と本研究のみである．部分グラフ同型判定問題のデータ依存回路を実際に FPGA に実装して，回路生成時間まで含めた優位性を定量的に検討した研究は，本研究が初めてであると思われる．

本論文では，まず 2 章で部分グラフ同型判定問題を定義し，Ullmann の提案アルゴリズムと小西の提案アルゴリズムについて説明する．小西の提案回路は論文 [3] で既に実装評価されているが，その

実装のままではデータ依存回路として実装する際に高い効果が見込めないことがわかった．そこで本研究では，データ依存回路に適した新たな実装方法を提案した．

3 章ではデータ依存回路について説明し，設計手順と評価項目に関して説明する．4 章では 2 章と 3 章で議論した内容を踏まえて，設計した回路の詳細を説明する．

5 章では，回路規模・回路生成時間・動作時間に関する評価結果を示す．データ依存回路を Xilinx Virtex-II XC2V3000 FPGA に実装した結果，Ullmann の提案回路と小西の提案回路の両者とも回路規模は減少した．入力グラフの頂点数が 16 以上では，データ依存回路は (回路生成時間を含めても) ソフトウェア実装より実行時間が短くなった．ランダムグラフを用いた評価では，頂点数が 16 のとき，小西のデータ依存回路はソフトウェアの 14.4 倍高速で，Ullmann のデータ依存回路は 9.2 倍高速であった．

2 部分グラフ同型判定

この章では，まず最初にグラフ理論に関する用語を説明し，その後に部分グラフ同型判定アルゴリズムについて説明する．

2.1 グラフ理論に関する用語

グラフ G を $G = (V, E)$ と定義する．$V = \{v_1, v_2, \ldots, v_p\}$ は有限個の頂点からなる集合，$p = |V|$ は頂点数とする．$E = \{e_1, e_2, \ldots, e_q\}$ は頂点対の有限集合，頂点対 $e_i = \{v_{i1}, v_{i2}\}$ を辺と呼ぶ．$q = |E|$ は辺数である．

$\deg(v_i)$ とは，頂点 v_i の次数のことで，頂点 v_i から出ている辺の本数，すなわち v_i と隣接している頂点の個数を表す．辺の向き，すなわち頂点対の順番を考慮するものを有向グラフ，考慮しないものを無向グラフと呼ぶ．また，辺が同じ頂点で頂点対をつくっているものが存在しないかつ無向グラフであるものを単純無向グラフと呼ぶ．なお，本研究では単純無向グラフを扱う．

グラフ G の辺密度とは G の辺数 q の，頂点数 p の完全グラフ K_p の辺数 $\frac{p(p-1)}{2}$ に対する比，すなわち $\frac{2q}{p(p-1)}$ である．グラフの表現には隣接行列を用いる．隣接行列は頂点間の隣接関係を表した $p \times p$ の行列である．グラフ G の隣接行列を $A = [a_{ij}]$ とすると，グラフ G の隣接行列の各要素 a_{ij} は

$$a_{ij} = \begin{cases} 1, & \text{if } \{v_i, v_j\} \in E, \\ 0, & \text{if } \{v_i, v_j\} \notin E \end{cases} \quad (1 \leq i, j \leq p) \tag{1}$$

単純無向グラフなら隣接行列は対象行列で，対角要素は 0 になる．v_i 頂点の次数は隣接行列の i 行の全要素の合計値で求めることができる．

2.2 部分グラフ同型判定問題

2 つのグラフ G_α と G_β が与えられたとき，G_α が G_β の部分グラフと同型か否か判定する問題を部分グラフ同型判定問題という．例えば図 1 において，G_β は G_α と同型な部分グラフを持つが，G_γ は持たない．部分グラフ同型判定問題は画像処理やシーン解析など多くの応用問題を持つが，一般に NP 完全で [1]，大規模問題を解くことは困難である．

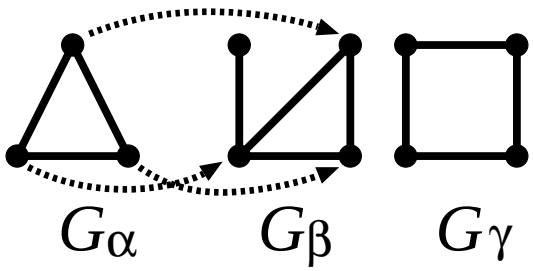


図 1: 部分グラフ同型判定の例

以下，グラフ $G_\alpha = (V_\alpha, E_\alpha)$ の頂点数を $p_\alpha = |V_\alpha|$，辺数を $q_\alpha = |E_\alpha|$ とする．同様に，$p_\beta = |V_\beta|$ と $q_\beta = |E_\beta|$ は，グラフ $G_\beta = (V_\beta, E_\beta)$ の頂点数と辺数であるとする．$p_\alpha > p_\beta$ または $q_\alpha > q_\beta$ の場合は部分グラフ同型でないことは自明なので，以降，$p_\alpha \leq p_\beta$ かつ $q_\alpha \leq q_\beta$ であることを前提とする．

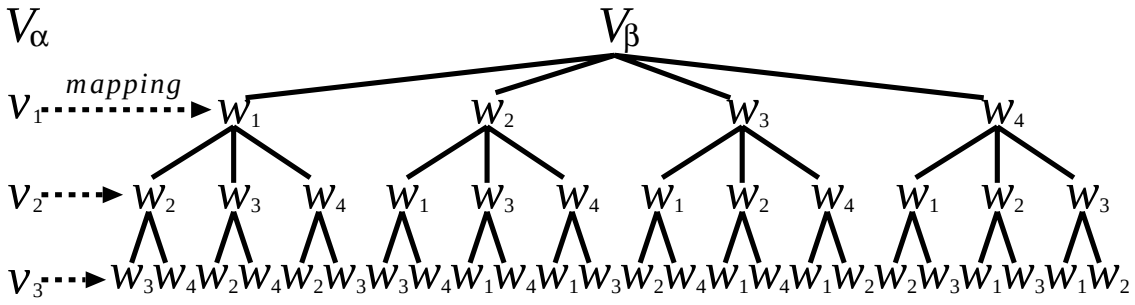


図 2: 探索木の例

部分グラフ同型判定問題はグラフ G_α の全ての頂点と G_β の頂点の対応を列挙し，部分グラフ同型になる組合せを見つけ出す一種の探索問題と言える．この問題を解くアルゴリズムとして，頂点の写像が存在するか否かを決定するために頂点の対応関係をすべて列挙する方法 (列挙アルゴリズム) が考えられる．ここで，頂点集合 $V_\alpha = \{v_1, v_2, v_3\}$, $V_\beta = \{w_1, w_2, w_3, w_4\}$ とすると，頂点の組合せは図 2 のような探索木を用いて列挙して調べあげることで実現可能である．

$$\forall i, j\,(\{v_i, v_j\} \in E_\alpha \Rightarrow \{f(v_i), f(v_j)\} \in E_\beta) \tag{2}$$

これは，頂点の写像をあらわす関数 f が，必要条件式 2 を充たすか否か判定することに等しい．G_β のいずれかの部分グラフに G_α の q_α 本の辺すべてに対応する辺が存在すれば，同型な部分グラフを発見したことになる．探索木の全ての葉を調べても同型な部分グラフがなければ，判定は失敗となる．このアルゴリズムは，上位に探索木巡回部を置き，探索木の葉に来たとき辺存在確認部を呼び出すという形で実装可能である．しかし，この方法は ${}_{p_\beta}\mathrm{P}_{p_\alpha} = p_\beta!/(p_\beta - p_\alpha)!$ の組合せを総当りする必要があるため，p_α と p_β の増加に伴って実行時間は急激に増加し，実用的ではない．判定を高速化するには，探索木の各ノードで，同型になる可能性の無い枝を切り捨てる必要がある (枝刈り)．

Ullmann[6] も小西ら [7] [10] も，枝刈りにより判定を高速化するアルゴリズムを提案した．2 つのアルゴリズムの大きな違いは枝刈りの方法にある．Ullmann のアルゴリズムと小西のアルゴリズムは，両方とも探索木をたどりながら頂点の対応関係を列挙する “探索木巡回部” と探索木の各ノード

で呼び出して不要な部分木を刈り込む “枝刈り判定部” からなる．Ullmann のアルゴリズムについては 2.3 章，小西のアルゴリズムについては 2.4 章で説明する．

2.3 Ullmann のアルゴリズム

Ullmann のアルゴリズムの枝刈り判定部を “refinement procedure” と呼び，探索木巡回アルゴリズムから呼び出される．ここでは，探索木巡回アルゴリズムと refinement procedure について説明する．アルゴリズムの詳細については文献 [6] を参照して頂きたい．

2.3.1 探索木巡回アルゴリズム

探索木巡回アルゴリズムは，深さ優先探索により探索木を巡回するアルゴリズムである．Ullmann のアルゴリズムでは行列 $M = [m_{ij}]$ $(1 \leq i \leq p_\alpha,\ 1 \leq j \leq p_\beta)$ を定義する．M の値は探索木の各ノードに対応し，1 か 0 の値を持つ．$m_{ij} = 1$ は G_α の頂点 v_i から G_β の頂点 w_j への写像において部分グラフ同型になる可能性があることを示す．一方，可能性がない場合 $m_{ij} = 0$ となる．

探索木の深さ d $(1 \leq d \leq p_\alpha)$ での M を M_d とする．すると，終端ノード $(d = p_\alpha)$ での M は M_{p_α} となり，以下のような性質を持つ．

- M_{p_α} の各行は値が’1’ である要素を 1 つ持ち，残りの要素の値は’0’ になる．
- M_{p_α} の各列は最大 1 個だけ’1’ を含む．

以上の性質は，G_α の各頂点が G_β の異なる頂点に一対一に対応して写像されることを意味する．

行列 M の初期値 $M^0 = [{m^0}_{ij}]$ は式 3 のように決定される．

$$ {m^0}_{ij} = \begin{cases} 1, & \deg(v_\alpha) \leq \deg(w_\beta), \\ 0, & otherwise, \end{cases} \qquad (1 \leq i \leq p_\alpha,\ 1 \leq j \leq p_\beta). \tag{3} $$

部分グラフ同型になるためには，G_α と G_β の部分グラフの次数が一致する必要がある．そのため G_α の頂点で G_β の任意の頂点より次数が多いものはこの条件に合わないためあらかじめ初期値 ${m^0}_{ij}$ を 0 にする．

また，探索木の終端ノード M_{p_α} において $m_{ij} = 1$ になるための必要条件は ${m^0}_{ij} = 1$ になることである．よって探索木巡回アルゴリズムは $m_{ij} = 1$ に対応する探索木中のノードを巡回する．終端ノードでない場合 $(d < p_\alpha)$ の M_d には以下の性質がある．

- $1 \leq i \leq d$ のとき，$\left(\sum_{j=1}^{p_\beta} m_{ij}\right) = 1$.
- $d < i \leq p_\alpha$ のとき，$m_{ij} = {m^0}_{ij}$.

探索木巡回アルゴリズムは，$m_{dk} = 1$ となる k を見つけだし，$m_{di} = 0$ $(i \neq k)$ にすることにより深さ d における頂点の写像を決定する．図 3 に探索木巡回アルゴリズムを示す．

アルゴリズム (図 3) の変数について説明する．

M 上述した写像の対応関係を表す行列 M である．

Step 1. $M := M^0$; $d := 1$; $H_1 := 0$;
for all $i := 1, \cdots, p_\beta$ set $F_i := 0$;
refine M; if exit FAIL then terminate algorithm;
Step 2. if there is no value of j such that $m_{dj} = 1$ and $f_j = 0$ then go to step 7;
$M_d := M$;
$k := 0$;
if $d = 1$ then $k := H_1$ else $k := 0$;
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. $M := M_d$;
if there is no $j > k$ such that $M_d(d, j) = 1$ and $f_j = 0$ then go to step 7;
go to step 3;
Step 6. $H_d := k$; $F_k := 1$; $d := d + 1$;
go to step 2;
Step 7. if $d = 1$ then terminate algorithm;
$d := d - 1$; $M := M_d$; $k := H_d$; $F_k := 0$;
go to step 5;

図 3: 探索木巡回アルゴリズム

M^0 上述した M の初期値である.

d 探索木の深さを表す変数である．値は整数で $1 \leq d \leq p_\alpha$ となる.

k まだ割り当ての済んでいない G_β 上の頂点 w_j $(1 \leq j \leq p_\beta)$ を検索するのに用いる変数である．値は整数で $1 \leq k \leq p_\beta$

M_d 深さ d で行列 M を保持するための配列である．各深さにおける M の値は探索木を下るときに M_d に格納され，探索木を遡る時に M_d から読み出される.

H 深さ d で頂点 v_d に対応する w_k の頂点番号 k を保持するための配列である．$H = \{h_1, h_2, \ldots, h_{p_\alpha}\}$. H_i は i 番目の要素を操作することを意味する.

F 幅 p_β の 2 進ベクトルで現時点でどの頂点が使用済みであるかを保持する．$F = \{f_1, f_2, \ldots, f_{p_\beta}\}$ となり，既に w_j 頂点が探索の深さ $d-1$ 以前に使用されているとき $f_i = 1$ となる.

アルゴリズム中の ":=" は代入を表す．なので $d := d + 1$ は d をインクリメントすることを表している．特に $M := M_d$ のように行列について書かれている場合は，行列全体をコピーすることを示す. アルゴリズム中の "refine M" で refinement procedure を実行し探索空間の削減を行う.

2.3.2 Refinement Procedure

Ullmann のアルゴリズムの枝刈り判定を行う部分を “refinement procedure” という．探索木巡回アルゴリズムが探索木の内部節点で refinement procedure を呼び出して “同型可能性判定” を行う．Refinement procedure により部分グラフ同型になる可能性がないと判定された場合，その節点以下の部分木は切り捨てられ探索空間が削減される．Refinement procedure によるオーバーヘッドが発生するが，探索空間削減の効果により実行時間は劇的に減少する．図 4 に refinement procedure を示す．

Step 1. elim := 0; $i := 1$;
Step 2. $k := 1$; sc := $2^{(p_\alpha - 1)}$; $h := 1$;
Step 3. if sc & $A_i = 0$ then go to step 4;
$\mathrm{lst}_k := h$; $k := k + 1$;
Step 4. sc := sc $\times 2^{-1}$;
$h := h + 1$;
if $k \neq \deg_i + 1$ then go to step 3;
Step 5. $j := 1$;
sc := $2^{(p_\beta - 1)}$;
Step 6. if M_i & sc $= 0$ then go to step 9;
$h := 1$;
Step 7. $x := \mathrm{lst}_h$;
if M_x & $B_j = 0$ then go to step 8;
$h := h + 1$;
if $h \neq \deg_i + 1$ then go to step 7 else go to step 9;
Step 8. $M_i := M_i$ & NOT sc;
elim := elim +1;
Step 9. sc := sc $\times 2^{-1}$;
$j := j + 1$;
if $j \neq p_\beta + 1$ then go to step 6;
Step 10. if $M_i = 0$ then go to FAIL exit;
$i := i + 1$;
if $i \neq p_\alpha + 1$ then go to step 2;
if elim $\neq 0$ then go to step 1;
go to SUCCEED exit;

図 4: Refinement procedure

アルゴリズム (図 4) の変数について説明する．h，i，j，x は整数型の変数である．

A G_α の隣接行列である．A_i は隣接行列 A の i 行目を取り出して得られる行ベクトルである．

B G_β の隣接行列である．B_i は隣接行列 B の i 行目を取り出して得られる行ベクトルである．

M 探索木巡回アルゴリズム (図 3) の M と同じである．

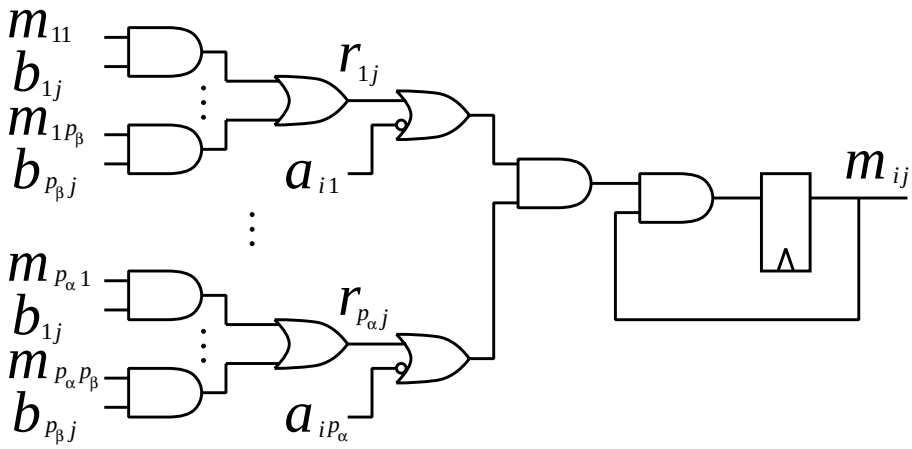


図 5: Refinement procedure の要素回路

elim Refinement procedure により'1' から'0' に変化した M の要素数である．

deg G_β の全頂点の次数を格納する変数である．$\deg_i$ は $\deg(w_i)$ を表す．

lst G_α の i 番目の頂点と隣接している頂点番号のリストである．

sc 1 ビットだけ'1' を持つ 2 進ベクトルで，M の行ベクトル中の'1' が現れる位置の探索に使用する．

アルゴリズム中の&はビットごとの AND 演算を示す (例: 1100 & 1010 = 1000)．NOT は全てのビットの NOT 演算を示す (例: NOT 1100 = 0011)．

次に，2.3.3 章にて refinement procedure による同型可能性判定の判定条件について述べる．

2.3.3 同型可能性判定の必要条件

Refinement procedure は，行列 M を用いて枝刈りを行う．この m_{ij} は部分グラフ同型になる頂点の対応の可能性を表す変数で，0 または 1 の値をとる．$m_{ij} = 1$ なら頂点 v_i と頂点 w_j が対応する可能性があることを表す．Refinement procedure は一種の再帰的な必要条件判定であるといえ，以下の処理 (式 4 と 5) を繰り返し行っていることに等しい [6].

$$r_{xj} = \exists y(m_{xy} \cdot b_{yj}) \tag{4}$$

$$m_{ij} = m_{ij} \cdot (\forall x(\overline{a_{ix}} \vee r_{xj})) \tag{5}$$

ここで r_{xj} は中間変数である．演算子 $\cdot$（ドット）と $\vee$ はそれぞれ論理積と論理和を表す．$\overline{a_{ix}}$ は a_{ix} の否定を表す．Refinement procedure は $m_{ij} = m_{ij}^0$(初期値) で計算を開始して，m_{ij} の値を繰り返し計算する．Ullmann は，この論理をハードウェアで実現することを提案した [6].

Ullmann が提案した回路は上位に探索木巡回部を持ち，探索木の各ノードで refinement procedure を呼び出すという形で実装可能である．Ullmann の提案を回路として実現すると，refinement procedure は図 5 に示した論理回路を $p_\alpha \times p_\beta$ 並べて実現できる．しかし，このとき回路規模は $O(p_\alpha {p_\beta}^2)$ となり，大きな p_α と p_β に関しては実装困難であることが報告されている [3].

2.4 小西のアルゴリズム

小西による提案は，枝刈りの条件を Ullmann のものより簡易なものにし，必要とする回路資源の削減を狙ったものである [7]．回路実装を前提として考えたとき，アルゴリズムは性能だけではなく必要とする回路規模も考慮しなければならないため，市川らは AT 積という指標を用いて Ullmann と小西のアルゴリズムの比較を行った．AT 積というのは回路規模と実行時間の積により表したもので，コストパフォーマンスを表す．市川らの評価では回路規模と性能にはトレードオフの関係があり，入力データによってはアルゴリズムの優劣がわかれることがわかっている [3]．

小西が提案したアルゴリズムは探索木の部分木を刈り込む “辺存在確認アルゴリズム” と，探索木を巡回しながら頂点の組合せを列挙する “探索木巡回アルゴリズム” からなる．探索木巡回アルゴリズムが探索木の各枝で辺存在確認アルゴリズムを呼び出すかたちで実現できる．

2.4.1 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 [7] は，探索空間削減，つまり枝刈りのところを除いては Ullmann の探索木巡回アルゴリズムと同じものである．探索木巡回アルゴリズムを図 6 に示す．

```
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 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

アルゴリズム (図 6) 中の M^0 は 2.3.1 章で説明したものと同じである．d は探索木の現在の深さを

表す．M_d は行列 M の d 行にアクセスすることを表す．$used$ は p_β ビットの記憶用変数で，Ullmann のアルゴリズム (図 3) の F に相当する．頂点 $w_i(1 \leq i \leq p_\beta)$ が既に使用されている場合，$used_i = 1$ で表す (0 なら使用されていない)．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．p は頂点の対応関係を表すための対応表と言える．

2.4.2 辺存在確認アルゴリズム

辺存在確認アルゴリズム [7] 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 v_d と隣接している辺 $\{v_d, v_i\} \in E_\alpha$ が G_β にも対応して存在するか調べる (必要条件の検査)．つまり，以下の式が成り立つか調べる．

$$\underset{1 \leq i \leq p_\alpha}{\forall i} \left(\{v_d, v_i\} \in E_\alpha \Rightarrow \{p(v_d), p(v_i)\} \in E_\beta \right) \tag{6}$$

上式 6 が成り立たないとき，G_β 上には対応する辺が存在しないことを意味する．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．

辺存在確認アルゴリズムを図 7 に示す．

```
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 7: 辺存在確認アルゴリズム

$etsa$ は edge list table starting address であり，G_α の辺を eta と etb を用いて表す．eta と etb は edge list table である．

深さ d のとき，頂点 d と接続される辺リストの先頭番号は $etsa$ で参照される．実際の辺のリストは eta と etb に保持されており，リストの先頭番号から順にならんでいて終端を “0” としている．eta と etb により参照できるのは辺を結ぶ頂点の番号である．B は G_β の隣接行列である．while, if, return 文は C 言語と同じ文法である．深さ d のとき p が必要十分条件 (式 6) を満たすとき OK を返し，そうでなければ NG を返す．

2.4.3 先行研究における実装

市川らの実装 [10] では，小西の提案どおり辺をリスト構造で表現して逐次的に隣接関係を判定している．この実装では，v_i から w_j への対応関係を実現する表が，RAM (Random Access Memory) により実現されている．多くの FPGA アーキテクチャでは FlipFlop より RAM の方が実装密度が高いため，RAM を用いると実装コストが低くなるためである．一方，RAM の使用により辺存在確認部

の処理が逐次化され，動作クロック数は増加している．また，3 章で述べるように，データ依存回路にした場合に論理規模の削減が難しい．

2.4.4 組合せ論理による実装の提案

動作速度とデータ依存回路への適合性を考えた場合，辺存在確認回路は組合せ論理で実装した方が良い．そこで 2 通りの設計案を検討し [9]，本研究では新たな実装回路を採用した．以下に，その実装の概略を示す．

先行研究 [10] では，複数の辺をリスト構造で実現したため，処理が逐次化されていた．これを並列処理する方法を考えねばならない．Ullmann の回路では，行列 M の各要素を並列に更新して，複数の辺に対する並列処理を行っている．そこで小西の必要条件式にも M を導入して，以下の式 (7) のように実装する．

$$\underset{1\leq x\leq p_\alpha}{\forall x}\left((d_{ix}=1)\Rightarrow\underset{1\leq y\leq p_\beta}{\exists y}\ (m_{xy}\cdot b_{yj}=1)\right) \tag{7}$$

ここで d_{ij} は，$j\leq i$ のとき $d_{ij}=a_{ij}$，それ以外は 0 となる (隣接行列 A を元にした下三角行列)．上の式の M は，Ullmann のアルゴリズムの M と異なることがある．レベル i のとき M の $i+1$ 以降の行の要素は全て 0 とする．

一時変数 $X=[x_i]$, $Y=[y_j]$, $W=[w_j]$, $Z=[z_j](1\leq i\leq p_\alpha, 1\leq j\leq p_\beta)$ を用いることで必要条件式 7 を式 8〜12 の論理で表現できる．必要条件が成り立つか否かを変数 OK で表現し，$OK=1$ ならそのレベルの必要条件が満たされていることを表す．なお，$V=[v_i](1\leq i\leq p_\alpha)$ は $v_i=1$ で，それ以外は 0 になる変数である．

$$x_i=(d_{i1}\cdot v_1)\vee\ldots\vee(d_{ip_\alpha}\cdot v_{p_\alpha}) \tag{8}$$
$$y_j=(x_1\cdot m_{1j})\vee\ldots\vee(x_{p_\alpha}\cdot m_{p_\alpha j}) \tag{9}$$
$$w_j=(v_1\cdot m_{1j})\vee\ldots\vee(v_{p_\alpha}\cdot m_{p_\alpha j}) \tag{10}$$
$$z_j=(b_{j1}\cdot w_1)\vee\ldots\vee(b_{jp_\beta}\cdot w_{p_\beta}) \tag{11}$$
$$OK=(\overline{y_1}\vee z_1)\cdot\ldots\cdot(\overline{y_{p_\beta}}\vee z_{p_\beta}) \tag{12}$$

式 8〜12 は図 8 の論理回路で実現可能である．

この実装は，隣接関係の判定を並列化するために，先行研究 [10] の実装より回路規模が大きくなると考えられる[1]．この実装回路を本研究では小西の提案回路と呼ぶことにする．3 章で述べるとおり，データ依存回路では組合せ回路の論理規模を削減する効果があるので，先行研究 [10] の実装より本研究で提案した組合せ回路による実装の方がデータ依存実装に適していると考えられる．

3 データ依存回路

一般に論理回路の入力が定まると，論理の簡単化によりゲートが除去され，回路規模が小さくなる (図 9)．あるゲートの出力が定数に決まれば，それを入力とするゲートも簡単化されるので，ゲート

[1]残念ながら 2 つの実装方法を同じ条件で比較していない．

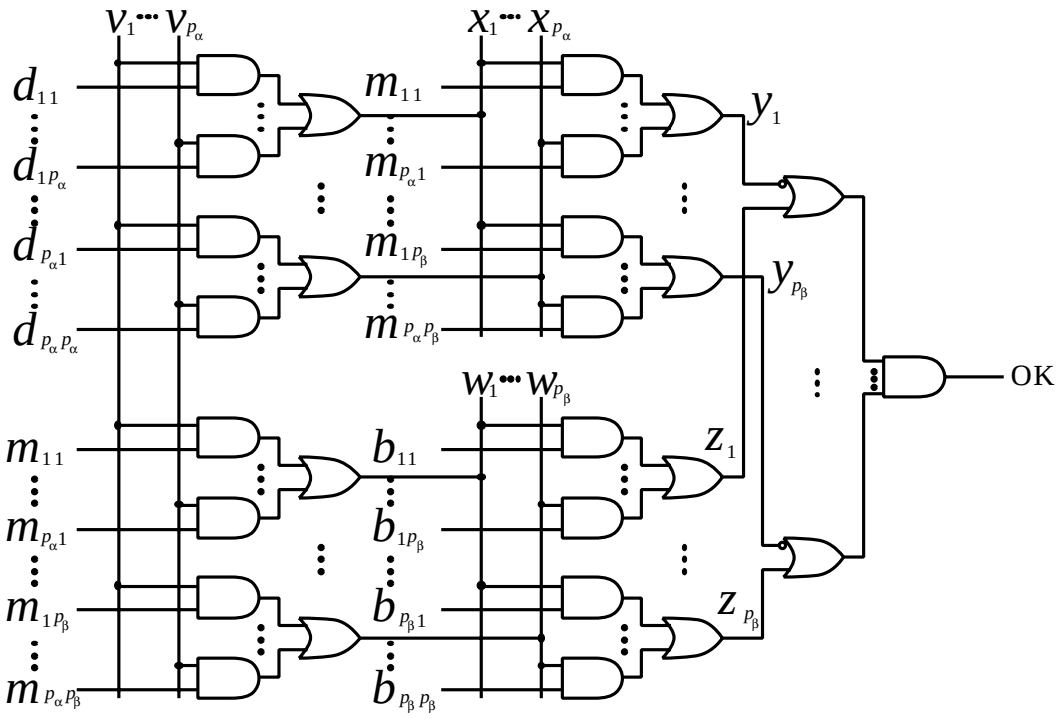


図 8: 辺存在確認部

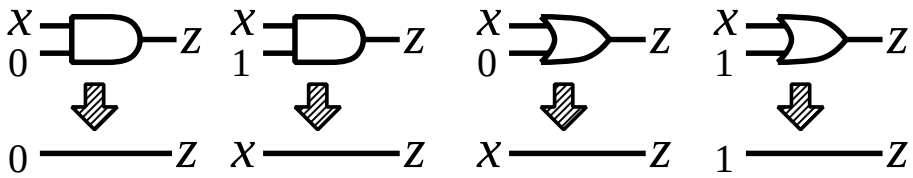


図 9: 論理の簡単化

の削減は再帰的に適用される (定数伝播)．論理が簡単化されると，論理段数が減少して動作周波数の向上も期待できる．記憶素子の入力が定数であるとわかった場合は，その記憶素子も削除できる場合がある．このようにして実現された回路は入力データに依存したものとなるので，以下では “データ依存回路” と呼ぶ．

上述の説明から分かるとおり，ゲートの削除は原則的に組合せ回路に対して行われる．FlipFlop などの記憶素子は，一般には値が実行時まで確定しないため，削除の対象とならない．従って，逐次的な回路よりは，大規模な組合せ論理の方が，データ依存実装による回路規模削減の恩恵を受けやすいと考えられる．

3.1 回路生成

データ依存回路で明らかなのは，入力データ毎に最適化されたため回路が再利用できないことである．そのため，データ依存回路では，入力データが与えられる毎に回路を生成する必要があるので

FPGA のような再構成可能な論理デバイスの利用が前提となる．

以下，入力データが与えられてから FPGA 上にデータ依存回路の構成するまでの処理を回路生成と呼び，処理が終了するまでの時間を “回路生成時間 T_{gen}” と呼ぶことにする．そして，FPGA 上に回路を構成した後に，動作が終了するまでの時間を “実行時間 T_{exec}” と呼ぶことにする．データ依存回路の総実行時間 T は T_{gen} と T_{exec} の合計となる．

$$T = T_{gen} + T_{exec} \tag{13}$$

なお，既存の VHDL 設計フローを用いた場合，回路生成時間には，以下の処理が含まれる．

データに依存した VHDL ソースコード生成 (以降，“VHDL 生成” と略す) 組合せ論理の入力が入力信号を保持する変数であったとき，データ依存回路を実現するには，変数であったところを入力信号に対応した定数に置き換えれば良い．データに依存した VHDL ソースコード生成とは，以上の処理を HDL(Hardware Description Language: 回路記述) の 1 つである VHDL(VHSIC HDL) のソースコードに施すことである (VHSIC は Very High Speed Intergrated Circuit の頭文字を取ったもの)．回路設計が VHDL ソースコードで出力されるので，以降の回路生成処理は既存の回路設計 CAD が利用可能になる．

論理合成 VHDL ソースコードを理解し，回路論理に置き換える．ソースコード内に助長な論理や定数値があった場合，論理合成では，論理の簡単化により回路規模を削減する．

テクノロジ・マッピング 論理合成の結果に対して，デバイスに依存した FPGA 内のプリミティブに置き換える．置き換えられた後は，FPGA の論理単位であるセル (Xilinx のデバイスではスライスと呼ぶ) 毎にまとめられる．

配置・配線 セル毎にまとめられた論理は，FPGA 上の何処に配置するか決めた後，セル同士を配線して，回路の周波数制約を充たすかチェックする．この処理は，周波数制約をパスするまで続けられるため，回路規模が大きい場合，実行時間が長大になる．

BIT 生成 配置・配線の結果から FPGA の構成データであるビット・ストリームに変換する．

ダウンロード 生成されたビット・ストリームを FPGA にダウンロードし，回路を構成する．

回路生成時間 T_{gen} は回路規模と密接な関係があり，回路規模が大きくなると，長大になる傾向がある．一方，実行時間 T_{exec} は実行サイクル数とサイクル時間 (動作周波数の逆数) の積で表すことができる．組合せ論理回路などで並列度を上げて高速化している回路は，実行サイクル数が減少する．しかし，回路規模が大きくなるため回路生成時間が長く，また，クリティカルパスの遅延が増大し，サイクル時間が長くなると考えられる．一方，論理規模が小規模な場合，実行サイクル数が増加するが，回路規模が小さくなるため回路生成時間が短く，また，クリティカルパスの遅延が減少し，サイクル時間が短くなると考えられる．

データ依存回路では組合せ論理回路に対して論理の簡単化が適用できるので，回路のサイクル数はそのままで，回路規模が減少すると期待できる．なので，回路規模と総実行時間の関係については定量的な評価が必要である．

3.2 回路生成の高速化

最も簡単な回路生成は，VHDL 生成にて，入力データの定数情報を VHDL ソースコード内の対応する変数に代入することで実現できる．上で説明したとおり，定数伝播が再帰的に起き，不要なゲートを論理合成が自動的に削減する．しかし，この方法は容易な反面，論理合成でより多くのメモリ空間と実行時間を必要とする．先行研究 [4] にて，VHDL 生成で簡単な前処理を行うことで論理合成時間が激減することを確認した．VHDL 生成での前処理とは，組合せ論理の展開を論理合成前に行ってしまうものである．本研究でも先行研究 [4] と同様に VHDL 生成で前処理を行う方法を採用する．そのため，論理の簡単化は VHDL 生成の工夫について，Ud (4.2 章) を例に説明する．

4 設計

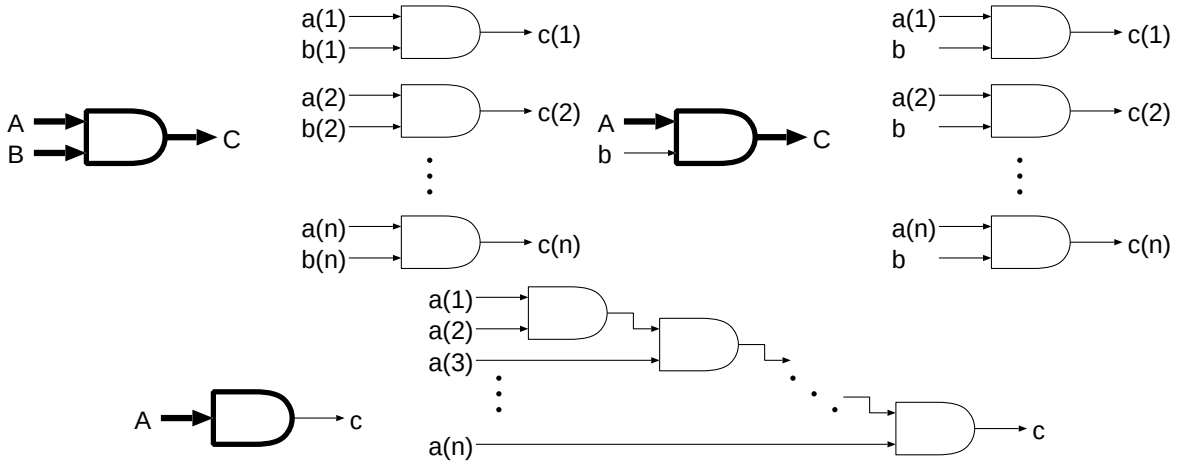


図 10: AND ゲート表記の意味

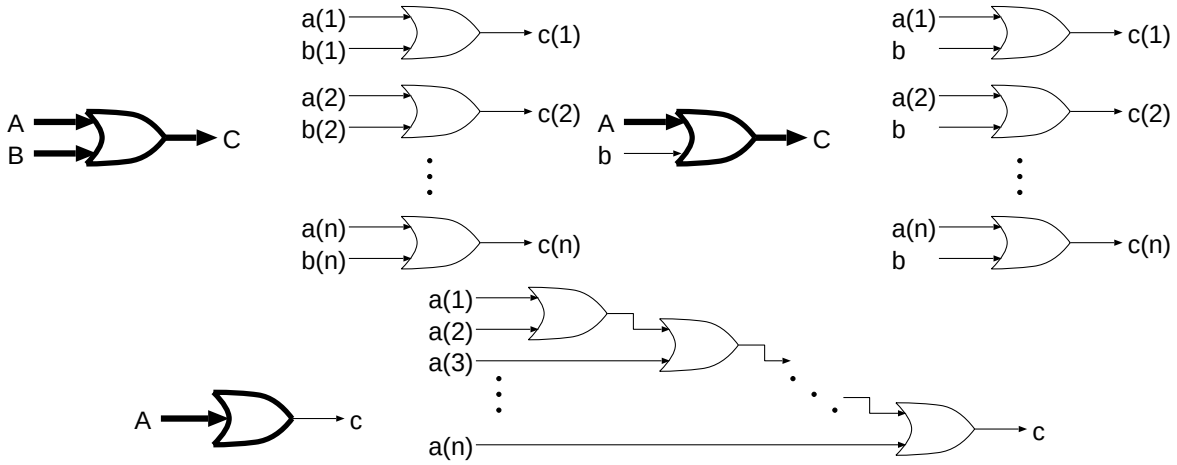


図 11: OR ゲート表記の意味

以降の説明では回路図を取り扱う．本稿で用いる AND ゲートと OR ゲートの独自表記をそれぞれ

図 10 と 11 に示す.

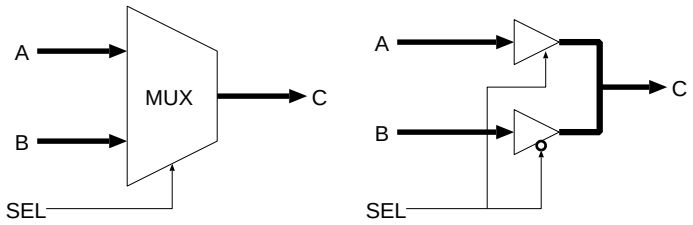


図 12: ワイアード OR による MUX の実装

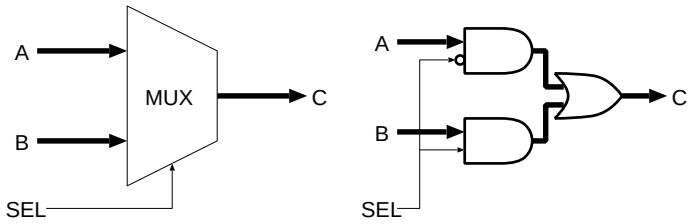


図 13: 組合せ論理による MUX の実装

また，論理回路内の信号経路を制御するのに MUX(マルチプレクサ) を用いる．MUX の実装には図 12 に示した TBUF(3 状態バッファー，出力には'0' と'1' の他にハイ・インピーダンス状態が存在する) の出力をワイアード OR で接続したものと図 13 に示した組合せ論理によるものが考えられる．

本研究のターゲットデバイスは Xilinx Virtex-II[5] シリーズの FPGA である．この Virtex-II シリーズのデバイスは，最小の論理単位であるスライスに対して，利用できる TBUF が少なく設計されているので，先行研究 [4] の設計のままでは，スライスよりさきに TBUF が不足する．そこで，MUX の実装には図 13 に示した組合せ論理によるものを採用した.

4.1 Ullmann の提案回路 (Uo)

Ullmann のアルゴリズム (2.3 章) を実装した回路で，Ullmann の提案どおり refinement procedure を組合せ論理回路で実現する．この章では探索木巡回アルゴリズムを回路実装した “探索木巡回回路” と refinement procedure を回路実装した refinement procedure 回路の 2 つにわけて説明する．なお，Uo は先行研究 [4] の *INDEP* を Xilinx Virtex-II に移植したもので，変更点は以下のとおりである.

プリミティブの変更 *INDEP* では TBUF と SRAM のプリミティブを直接利用していたので，移植するにあたり Xilinx Virtex-II 用に書き換えた.

MUX の変更 *INDEP* ではワイアード OR で接続 (図 12) を採用したが，Uo では組合せ論理 (図 13) を採用した.

4.1.1 探索木巡回回路

探索木巡回回路を実現する回路を図 14 に示す．図 14 中のブロックは以下のとおりである.

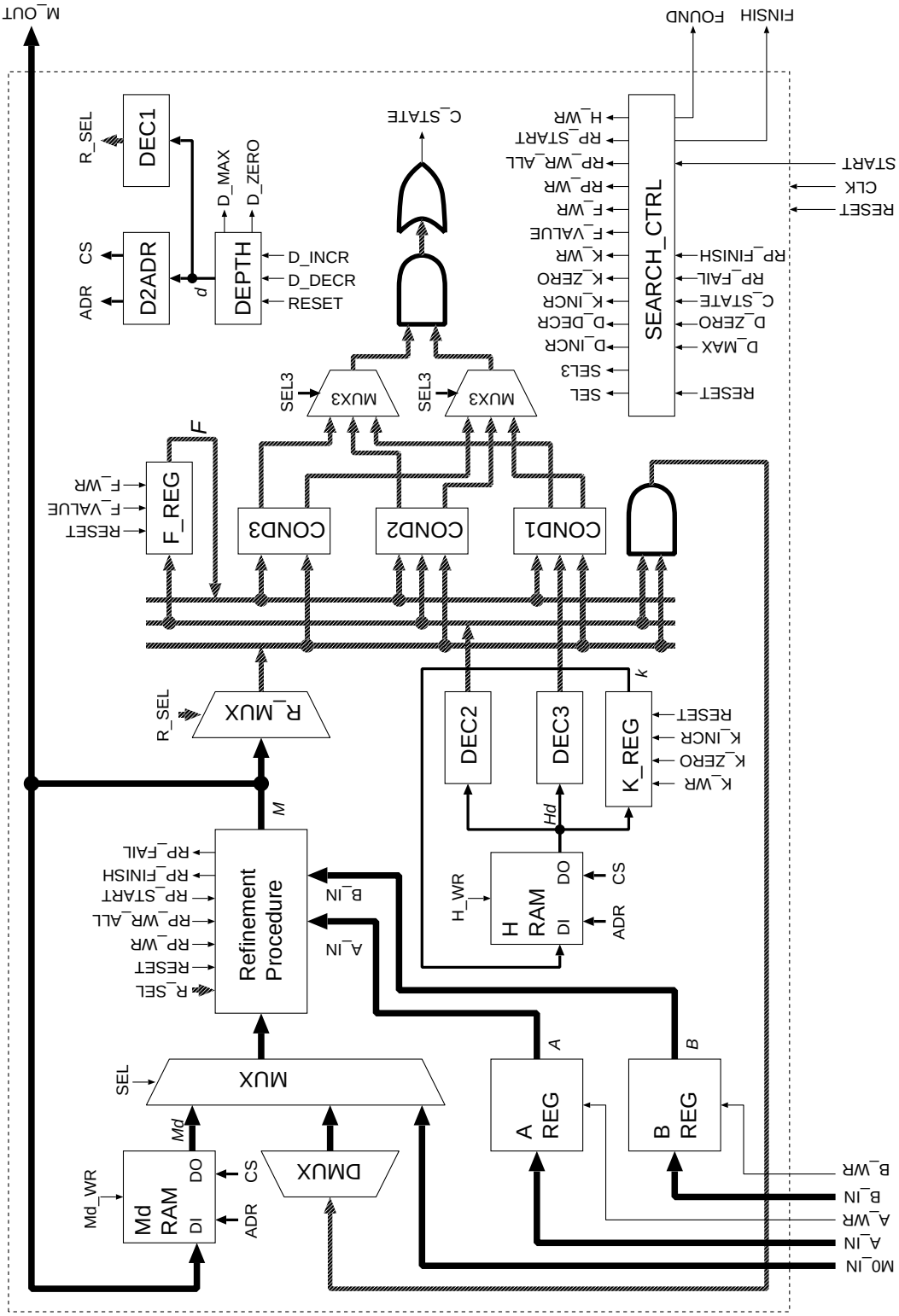

M_OUT
FINISH
FOUND
C_STATE
R_SEL
DEC1
D_MAX
D_ZERO
CS
ADR
D2ADR
DEPTH
D_INCR
D_DECR
RESET
SEARCH_CTRL
H_WR
RP_START
RP_WR_ALL
RP_WR
F_WR
F_VALUE
K_WR
K_ZERO
K_INCR
C_STATE
RP_FAIL
RP_FINISH
D_ZERO
D_DECR
D_INCR
D_MAX
SEL3
SEL
RESET
START
CLK
RESET
MUX3
SEL3
MUX3
F
F_REG
F_WR
F_VALUE
RESET
COND3
COND2
COND1
R_SEL
R_MUX
DEC2
DEC3
K_REG
k
K_WR
K_INCR
K_ZERO
RESET
M
Hd
H_WR
H RAM
DO
DI
CS
ADR
Refinement Procedure
R_SEL
RESET
RP_WR
RP_WR_ALL
RP_START
RP_FINISH
RP_FAIL
B_IN
A_IN
SEL
MUX
A
B
A REG
B REG
Md
Md_WR
Md RAM
DO
DI
CS
ADR
DMUX
B_WR
B_IN
A_WR
A_IN
M0_IN

図 14: 探索木巡回回路

A_REG G_α の隣接行列 A を格納する記憶素子である．FlipFlop で実現する．

B_REG G_β の隣接行列 B を格納する記憶素子である．FlipFlop で実現する．

Md_RAM 探索木巡回アルゴリズムの Md をにあたる．ブロック RAM で実現する．

MUX Refinement procedure 部の入力を切替える．

DMUX 行単位の処理の結果を行列に戻すのに用いる．

Refinement Procedure Refinement procedure 部である．また M を内部に持つ．FlipFlop で実現する．

H_RAM 探索木巡回アルゴリズムの H をにあたる．ブロック RAM で実現する．

R_MUX M の d 行目の行を取り出すのに用いる．

K_REG 探索木巡回アルゴリズムの k をにあたる．FlipFlop で実現する．

COND1 探索木巡回アルゴリズムの条件式の論理にあたる．回路共有のため分割したブロックの 1 つである．

COND2 探索木巡回アルゴリズムの条件式の論理にあたる．回路共有のため分割したブロックの 1 つである．

COND3 探索木巡回アルゴリズムの条件式の論理にあたる．回路共有のため分割したブロックの 1 つである．

F_REG 探索木巡回アルゴリズムの F をにあたる．FlipFlop で実現する．

MUX3 探索木巡回アルゴリズムの条件式の論理にあたる DEC1 と DEC2，DEC3 の出力を選択して実際の if 文が成り立つか判定する組合せ論理回路の入力とする．

DEPTH 探索木巡回アルゴリズムの d にあたり，インクリメントとデクリメント，値の判定の機能がある．FlipFlop で実現する．

D2ADR 探索木巡回アルゴリズムの d の値を実際のブロック RAM へアクセスする際のチップセレクト信号とアドレス信号に変換する．

DEC1 k を入力とし，出力は $f_k = 1$，$f_{i \neq k} = 0$ とする論理回路である．

DEC2 k を入力とし，出力は $f_{i>k} = 1$，$f_{i \leq k} = 0$ とする論理回路である．

DEC3 d を入力とし，出力は p_α のうち d 桁目のビットだけ 1 とし，残りのビットは全て 0 とする論理回路である．

SEARCH_CTRL 探索木巡回アルゴリズムを実現する制御回路 (図 15) である．分岐を表す矢印に重みとして数字がふってある．これは分岐判定の優先順位を表し，小さい値のものを先に評価する．

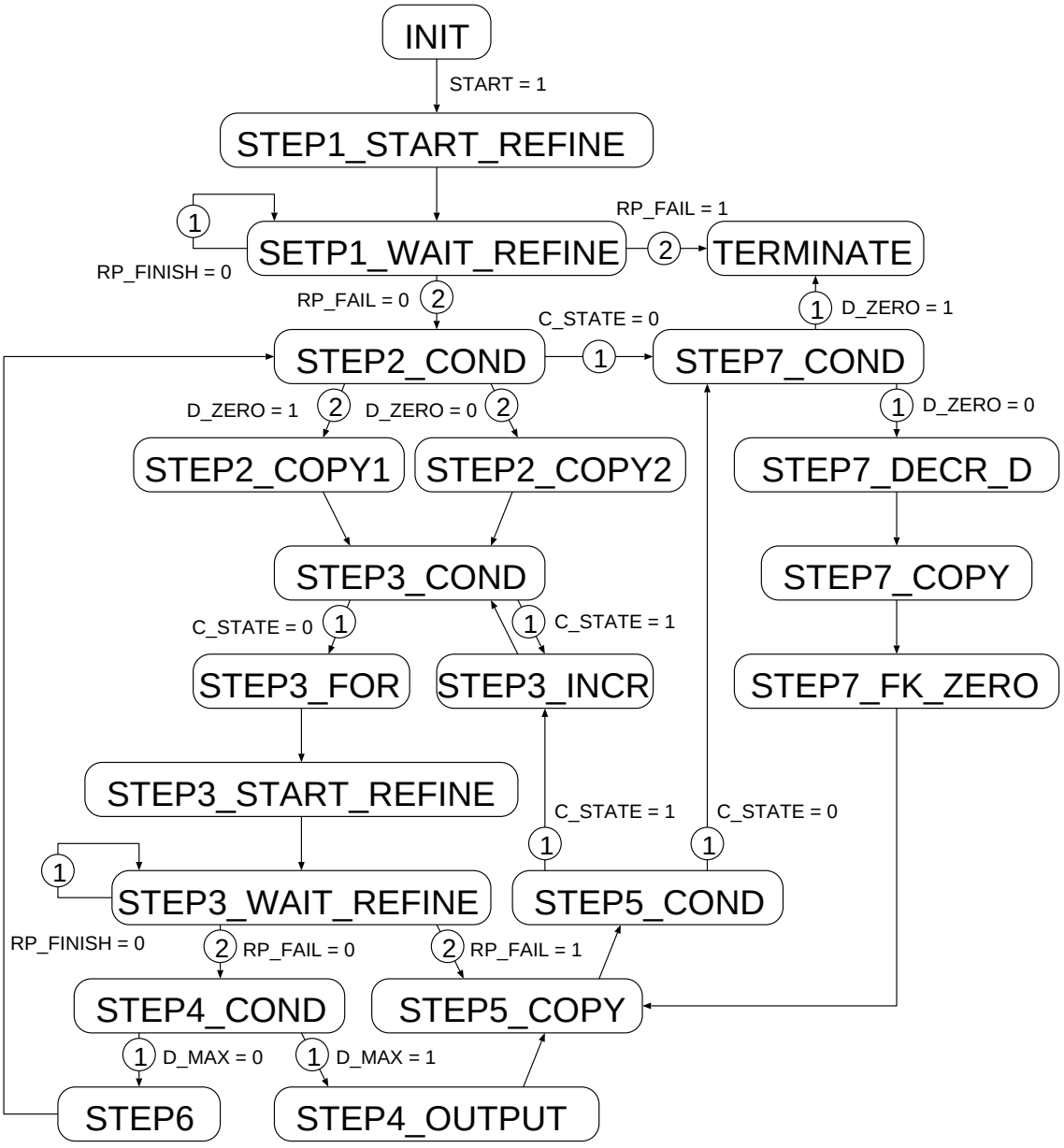


図 15: 探索木巡回回路の制御

4.1.2 Refinement procedure 回路

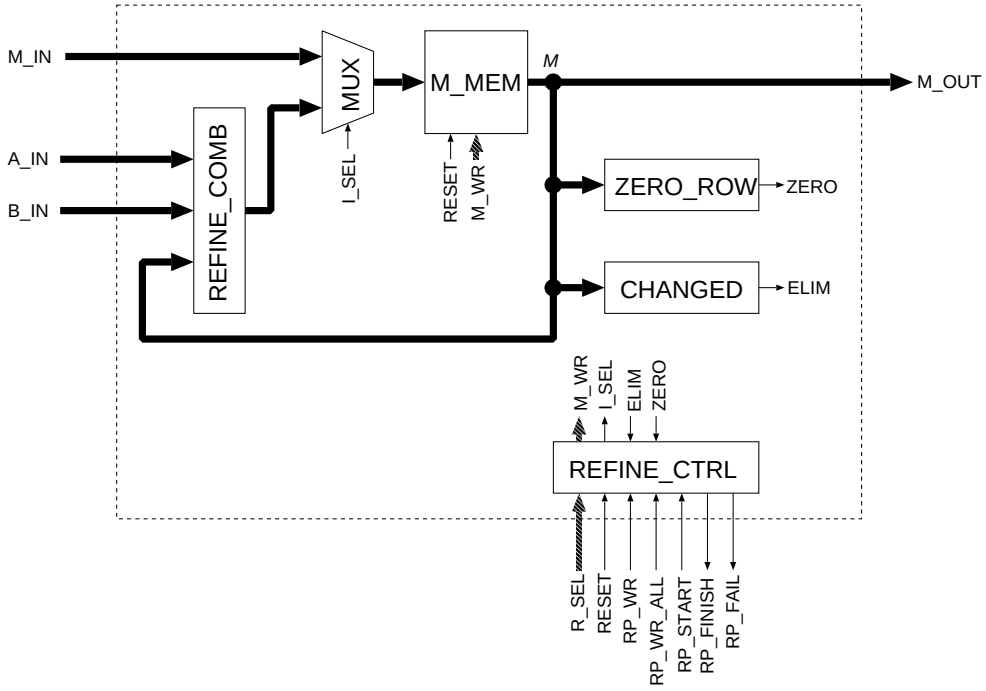


図 16: Refinement procedure 回路

Refinement procedure 回路を実現する回路を図 16 に示す．図 14 中のブロックは以下のとおりである．

REFINE_COMB 式 4 と 5 を実現する論理回路．論理の簡単化による回路規模の削減を考えて，組合せ論理回路で実現する (図 5 を $p_\alpha \times p_\beta$ 要素並べて実現)．

MUX 制御が，探索木巡回部回路に移ったら，MUX は M_IN を M_MEM に接続する．これにより外部回路から行列 M の要素を操作できるようにする．

M_MEM Ullmann アルゴリズム (2.3 章) の行列 M を格納する．$p_\alpha \times p_\beta$ 要素を全て同時に読み書きする必要性があるため FlipFlop で実装した．

ZERO_ROW 行列 M のいずれかの行の要素が全て 0 になったとき ZERO が 1 を出力する (そういった場合以外は 0 を出力する)．これは refinement procedure (図 4) の Step 10 で $M_i = 0$ の判定を行う部分を実現する回路である (図 4 では逐次処理で書かれているが，ここでは組合せ論理で実現されている)．

CHANGED Refinement procedure (図 4) の elim に相当する．ただし，elim は変化した要素数を数えるが，CHANGED は変化したことのみを 0 か 1 で表す．要素が 1 つでも変化したら ELIM は 1，変化しなかったら 0 を出力する．

REFINE_CTRL Refinement procedure を実現する制御回路 (図 17) である．分岐を表す矢印に重みとして数字がふってある．これは分岐判定の優先順位を表し，小さい値のものを先に評価する．

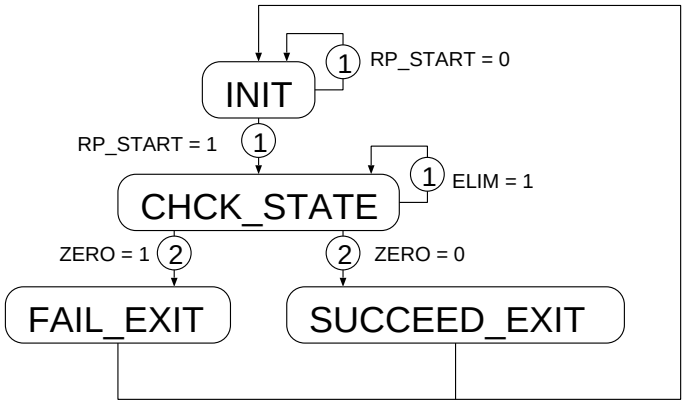


図 17: Refinement procedure 回路の制御

4.2 Ullmann のデータ依存回路 (Ud)

Ullmann のデータ依存回路 Ud は，4.1 章で説明した Uo の，データに依存した実装である．Ud の設計は，論文 [4] の BOTH2 と基本的に同じである．論文 [4] では Lucent OR2C FPGA をターゲットとしていたため，本研究では最低限度の修正を加えて Xilinx Virtex-II FPGA 用に変更したものを用いている．

4.2.1 最も簡単な実現方法

Ullmann のデータ依存回路の最も簡単な実現方法は，入力データ毎に入力信号を隣接行列 A と B に対応した 0 または 1 の定数に置き換えて VHDL ソースコードを作成することである．論文 [4] では，この実現方法を BOTH0 と呼んだ (隣接行列 A と B の両方に依存した論理という意味で “BOTH”)．この方法では論理合成ツールの入力は Ullmann の提案回路とほぼ同等の回路規模の回路が先ず与えらる．論理合成ツールが定数である入力信号を調べて論理の簡単化を行って回路規模を減少させる．そのために，この方法では最初に与える論理回路の回路規模が CAD ツールの処理能力に対して大きいものとなり，場合によってはメモリ不足などにより異常終了を引き起こす．また，論理合成の処理時間が増大するために回路生成時間は実用的ではない．

4.2.2 論理の簡単化を事前に処理する方法

ただ入力信号を定数に置き換えるのではなく，論理の簡単化を論理合成の前段階で行う方法を採用したのが論文 [4] の BOTH1 である．BOTH1 では VHDL を生成するプログラム (VHDL 生成プログラム) が回路論理と入力データの値をみて，論理の簡単化を行う．そして，VHDL 生成プログラムでやり残した論理の簡単化を論理合成ツールが受け持つという流れで回路を生成する．BOTH1 なら論理合成ツールに与えられる VHDL ソースコードに記述された回路論理量が最初から削減されているので，論理合成ツールの処理時間を大幅に短縮可能である [4].

4.2.3 回路構造やアルゴリズムに注目する方法

回路記述を生成するプログラムが回路の内部構造やアルゴリズムに注目して柔軟に回路を生成する方法が考えられ，それが論文 [4] の BOTH2 である．本研究の Ud は入力データ毎に入力を隣接行列 A と B に対応した 0 または 1 の定数に置き換えて回路生成したもので，基本設計は論文 [4] の BOTH2 に相当する．Ud で採用した柔軟に回路を生成する方法以下のとおりである．

- Ullmann のアルゴリズムでは，M の要素 m_{ij} は 1 から 0 にはなるが，一度 0 になると再び 1 にならない．2 つのグラフ G_α と G_β が与えられると M の初期値 M^0 が決まる．M^0 の 0 である要素について論理を削減した．つまり，図 14 の REFINE_COMB を実現する要素回路 5 の m_{ij} は ${m^0}_{ij} = 0$ について削減が可能である．
- 初期値 M^0 による削減は組合せ論理だけではなく M や Md のための記憶素子にも適用可能である．

ただし，回路構造やアルゴリズムに注目する方法を採用した場合，回路規模の減少や回路生成時間の短縮を狙った工夫を盛りこむことが可能になるが，回路記述を生成するプログラムが煩雑になる恐れがある．

4.3 小西の提案回路 (Ko)

3 章で説明したが，ゲートの削除は原則的に組合せ回路に対して行われる．小西のアルゴリズムをデータ依存回路で実装する場合，先行研究 [10] の回路のようなシーケンシャルな回路より 2.4.4 章で説明したものの方がデータ依存回路に適していると考えられる．以降は，2.4.4 章の辺存在確認回路を組合せ論理回路で実現したものを小西の提案回路 (Ko) と呼ぶ．

4.4 アルゴリズム

2.4.4 章で説明方法をより明確に表すアルゴリズムに書き直したものを図 18 に示す．ただし，辺存在確認部は式 8〜12 をそのまま組合せ論理回路で実現するのここでは省略する．なお，$\oplus$ は排他的論理を表している．

M, P それぞれ $[m_{ij}]$, $[p_{ij}]$ $(1 \le i \le p_\alpha, 1 \le j \le p_\beta)$．$P$ は式 6 の写像関数 p に相当する．そのために，P は $\sum_j p_{ij} \le 1$ かつ $\sum_i p_{ij} \le 1$ であるという条件を持つ．つまり G_α 頂点と G_β の頂点は 1 対 1 で対応することを表している．

M^0 2.3.1 章の M^0 と同じものである．式 3 のように決定される．

V $V = [v_i]$ $(1 \le i \le p_\alpha)$．$\sum_i v_i = 1$ が成り立つ．$v_i = 1$ なら割り当てようとしている頂点は i で，探索木の現在の深さも i である．

N 要素数 p_β の 2 進数ベクトルである．$n_j = 1$ なら G_β の頂点 j は写像先として選択されていない．

C 要素数 p_β の 2 進数ベクトルである．$n_j = 1$ なら G_β の頂点 j は写像先として選択されていない．

maxbit(C) C の最上位にある 1 だけ出力する関数である (表 1)．

```
M := M^0;
N := {1, ··· , 1};
V = {v_1, v_2, ··· , v_α} := {1, 0, ··· , 0};
while (1) {
    C := M_pe(V) & N;
    flag:=0;
    while ( ∃j(c_j = 1) ){
        P_pe(V) := maxbit(C);
        C := C ⊕ P_pe(V);
        "辺存在確認" を実行.
        if (OK=1) {
            if (v_α = 1) {
                "写像結果 P" を出力.
            } else {
                N := N ⊕ P_pe(V);
                M_pe(V) := C;
                V := V >> 1;
                flag := 1;
                break;
            }
        }
    }
    if (flag) continue;
    if (v_1 = 1) terminate algorithm;
    P_pe(V) := {0, ··· , 0};
    M_pe(V) := M^0_pe(V);
    V := V << 1;
    N := N ⊕ P_pe(V);
}
```

図 18: 探索木巡回アルゴリズム (辺存在確認の組合せ論理回路実装のために書き直した)

表 1: maxbit(C) の真理値表

C	maxbit(C)
$00\cdots00$	$00\cdots00$
$00\cdots01$	$00\cdots01$
$00\cdots1*$	$00\cdots10$
$\vdots$	$\vdots$
$01\cdots**$	$01\cdots00$
$1*\cdots**$	$10\cdots00$

ただし $*$ **は** *Don't Care*

pe(V) V から探索の深さを返すプライオリティエンコーダである．$v_i = 1$ なら $pe(V) = i$ を返す関数である．よって $M_{\mathrm{pe}(V)}$ は $v_i = 1$ の場合，M_i を意味する．

4.5 実装回路

探索木巡回回路を実現する回路を図 19 に示す．図 19 中のブロックは以下のとおりである．

D_REG アルゴリズム (図 18) の隣接行列 A をもとにした下三角行列 D を格納する．

B_REG アルゴリズム (図 18) の隣接行列 B を格納する．

N アルゴリズム (図 18) の N を格納する記憶素子で FlipFlop で実現する．

AND 図 10 の表記ルールどおりの組合せ論理回路である．

XOR1, XOR2 図 10 や 11 の表記ルールを排他的論理和に適用したものである．

M_RAM アルゴリズム (図 18) の行列 M を格納する．RAM で実装し，探索レベルが d のとき M_d にアクセスできる．

MUX_A, MUX_B, MUX_C データ信号線の制御をする．

C アルゴリズム (図 18) の C を格納する記憶素子で FlipFlop で実現する．

M0_RAM アルゴリズム (図 18) の行列 M^0 を格納する．ROM で実装し，探索レベルが d のとき M_d が参照できる．

P_MEM アルゴリズム (図 18) の行列 P を格納する．アルゴリズムの設計上，全ての要素を同時に読める必要があるため FlipFlop で実装する．

maxbit アルゴリズム (図 18) の maxbit(C) を実現する論理回路．

Edge Check アルゴリズム (図 18) の辺存在確認部で，図 8 の組合せ論理回路で実現する．

SEARCH_CTRL 探索木巡回部を実現する制御回路 (図 20) である．分岐を表す矢印に重みとして数字がふってある．これは分岐判定の優先順位を表し，小さい値のものを先に評価する．

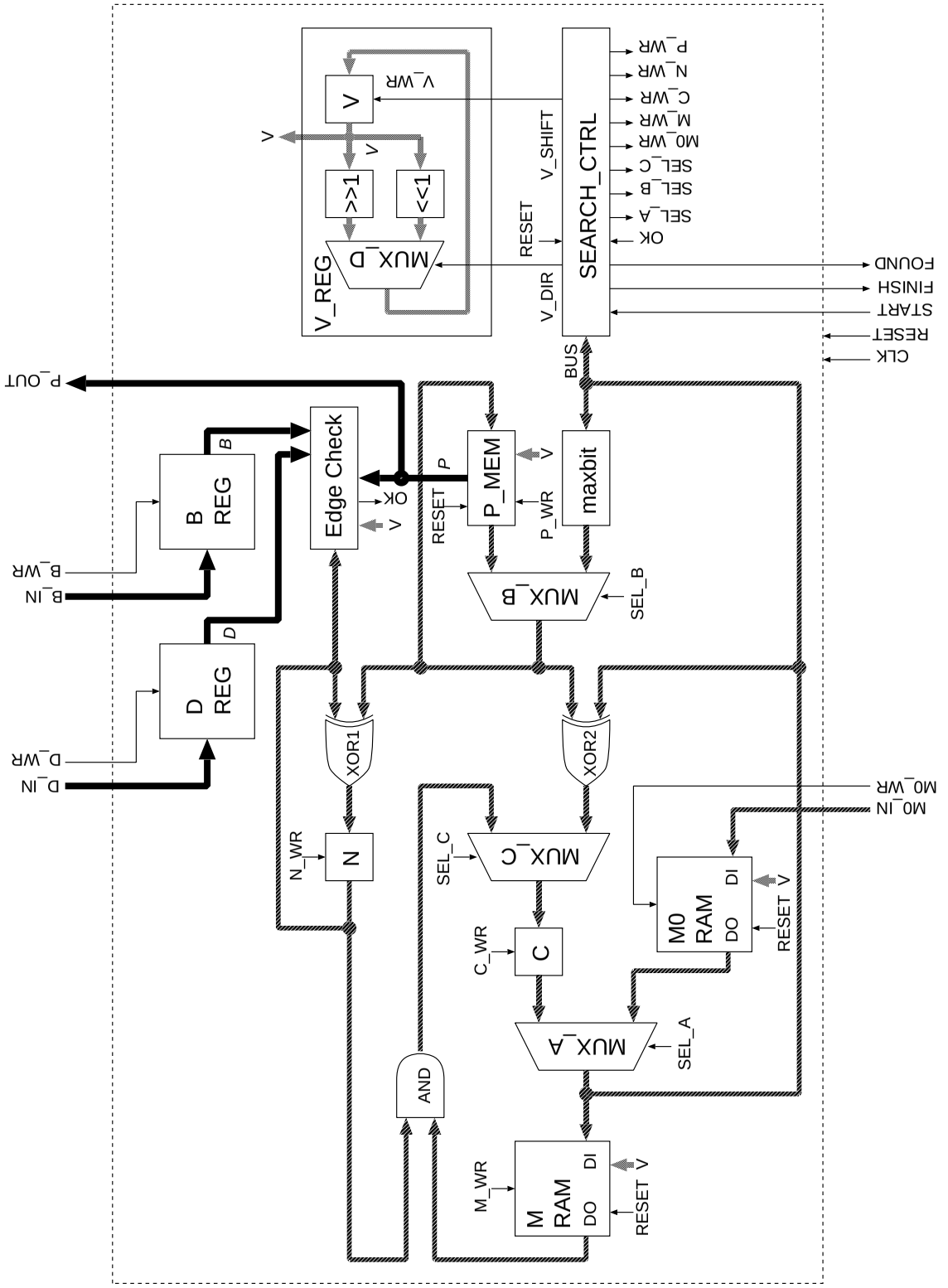


図 19: Ko の探索木巡回部

V_REG アルゴリズム (図 18) の V を格納する記憶素子で FlipFlop で実現する．小西の提案したアルゴリズムはもともと探索レベルを d で表していたが，本提案 (図 18) では V が探索レベルを表している．$d := d + 1$; の処理 (図 6) に相当するは $V := V >> 1$; である．

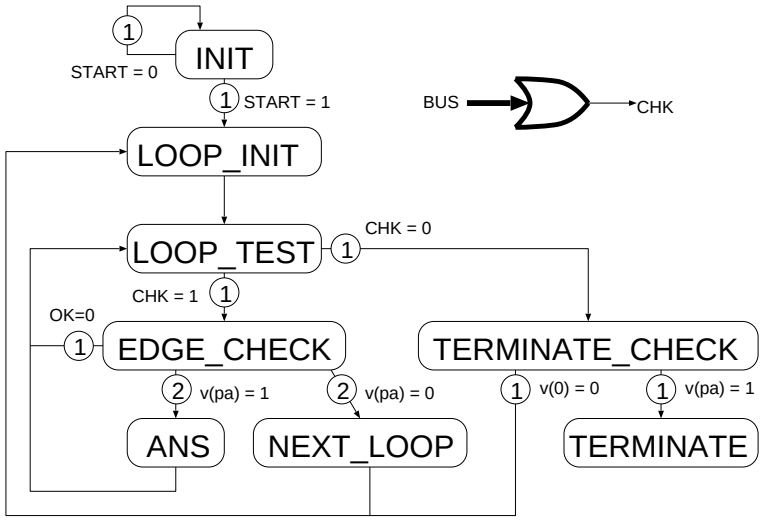


図 20: Ko の探索木巡回部の制御

4.6 小西のデータ依存回路 (Kd)

小西のデータ依存回路 Kd は，4.3 章で説明した Ko のデータ依存実装回路である．辺存在確認部である図 8 の入力信号線は，D (隣接行列 A の下三角行列) と行列 M，隣接行列 B の情報を参照する．この部分を入力グラフの定数情報，つまり隣接行列 A と B，行列 M^0 に対応した 0 または 1 に置き換えて論理回路を作成する．

論理の簡単化により規模が Ko に比べて小さくなると考えられる．Kd は入力データに依存した実装であるため，入力グラフの隣接行列 (A と B) を保持する RAM や FlipFlop も不要になる．4.3 章で説明した通り，本研究の小西の提案回路は，辺存在確認部の組合せ論理回路で実装されるので，データ依存実装により回路規模が削減されやすいと考えられる．

5 評価

5.1 評価環境

Ullmann のデータ依存回路 Ud (4.2 章参照) と小西のデータ依存回路 Kd (4.6 章参照) を，表 2 に示した FPGA ボードに実装し，動作させて評価した (評価に用いた FPGA ボードの外観を図 21 に示す).

データ依存回路はホスト PC 上で作成される．データ依存回路の作成は，入力グラフから “VHDL 生成” によりデータに依存した VHDL ソースコードを作成する．データに依存した VHDL ソースコードに対し，“論理合成 CAD” と “FPGA 実装 CAD” を実行して FPGA の構成データであるビッ

表 2: 評価環境

項目	内容
ホスト PC	Athlon XP 2600+ (2.1GHz)，Memory 1GB，Windows 2000 SP4
FPGA ボード	YDK MIRE-MULTI3000 (Xilinx Virtex-II XC2V3000FF1152-4)
JTAG ケーブル	Xilinx Parallel Cable IV
パラレル I/O	ADTEK aPCI–P31A
時間計測プログラム	Visual Basic 6.0sp5 実装の自作ソフトウェア
回路記述生成	C 実装自作ソフトウェア，コンパイラ Cygwin 上の gcc-3.2
論理合成 CAD	Synopsys FPGA Compiler II (2001.08-FC3.7) （27MHz 動作，エリア優先，エフォート・レベル低で最適化）
FPGA 実装 CAD	Xilinx ISE 5.2i （Virtex-II アーキテクチャ[5] 用）
ソフトウェア実装 及び動作環境	Ullmann のアルゴリズムの C 実装，コンパイラ gcc-3.2.2 Athlon XP 2600+ (2.1GHz)，Memory 512MB，Red Hat Linux 9

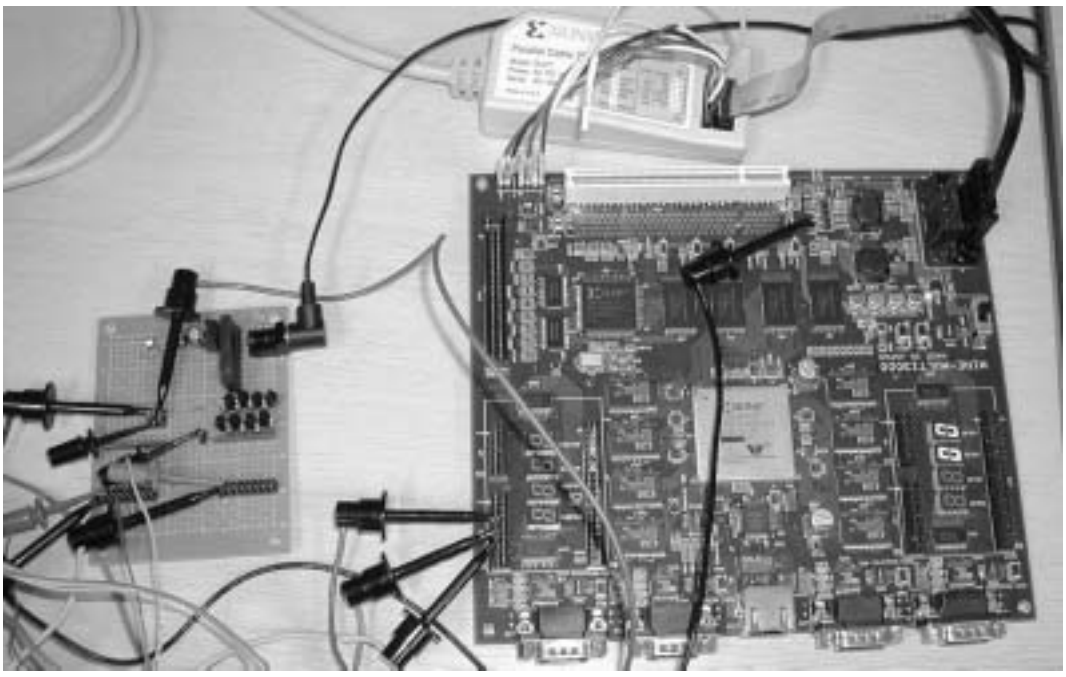


図 21: 評価に用いた FPGA ボード YDK MIRE-MULTI3000

トストリーム（Xilinx Virtex-II シリーズ FPGA [5] の構成データの呼び名）を作成する．ホスト PC と FPGA ボードは JTAG ケーブルで接続されており，作成したデータ依存回路のビットストリームをバウンダリスキャンモードで FPGA にダウンロードし，回路を構成する．入力データが与えられてからビットストリームのダウンロードが終了するまでの時間を回路生成時間 T_{gen} とした．FPGA ボード上の 50MHz オシレータ 2 分周した 25MHz をシステムクロックとして用いるため，論理合成のオプションとして 27MHz 動作を指定している．データ依存回路の最大動作周波数 f_{max} で動作させたときの実行時間を T_{exec} とする．そのために，実測した実行時間 t_{exec} と配置・配線後のトレースから求めた最大の動作周波数 f_{max} を用いて以下の式 14 から T_{exec} を計算した．f_{clk} は実質のシステムクロックの周波数で本評価では $25 \times 10^{+6}$ である．

$$T_{exec} = t_{exec} \times \frac{f_{clk}}{f_{max}} \tag{14}$$

FPGA にデータ依存回路を構成した後，時間計測プログラムがパラレル I/O を操作して，データ依存回路を動作させる．時間計測プログラムは回路を動作させた後，同型判定が終了するまでポーリングする．このポーリングの時間を t_{exec} とした．

小さいグラフに対しては回路遅延が小さい回路が生成されるが，大きいグラフに対しては反対に回路遅延が大きい回路が生成される．回路遅延が大きいとは，動作周波数が低下することで，つまり性能が低下する．これは，頂点数が大きくなれば動作周波数制約を充たすことが難しいことを意味している．このような理由で，f_{max} を動作周波数として用いる評価の方が妥当であると考える．

5.2 入力データ

評価に用いる入力データにはランダムグラフを採用した．ランダムグラフとは，辺があるかないかを表す確率 “辺存在確率” をパラメータに持つグラフである．今回は，評価するパラメータを減らす関係上 $p = p_\alpha = p_\beta$ とし，各頂点数 p に対して 100 組毎のランダムグラフを生成した．

データ依存回路では回路生成時間があるのでソフトウェアの実行時間がある程度長大になる．頂点数 p で実装するのが有効である．しかし，辺存在確率の組合せによってはソフトウェアでも実用的な時間で判定を終了する．そこで 4 つの辺存在確率の組合せを以下のように用意した．

RG0303 G_α と G_β の辺存在確率はそれぞれ 0.3 と 0.3 とする．

RG0306 G_α と G_β の辺存在確率はそれぞれ 0.3 と 0.6 とする．

RG0603 G_α と G_β の辺存在確率はそれぞれ 0.6 と 0.3 とする．

RG0606 G_α と G_β の辺存在確率はそれぞれ 0.6 と 0.6 とする．

上の 4 つのデータセットを表 2 に示したソフトウェア実装環境で実行時間を測定したものを図 22 に示す．4 つのなかでは RG0306 が頂点数 p の増大で実行時間が増加しやすい．つまり，判定の高速化のために回路生成時間を含んでも回路実装する必要性があるので，RG0306 を評価に用いる．

5.3 回路規模

図 23 はテクノロジ・マッピングのレポートから得られた回路規模で，100 組の入力グラフから得た平均値である．回路規模の単位は Xilinx Virtex-II シリーズ FPGA の最小論理単位であるスライ

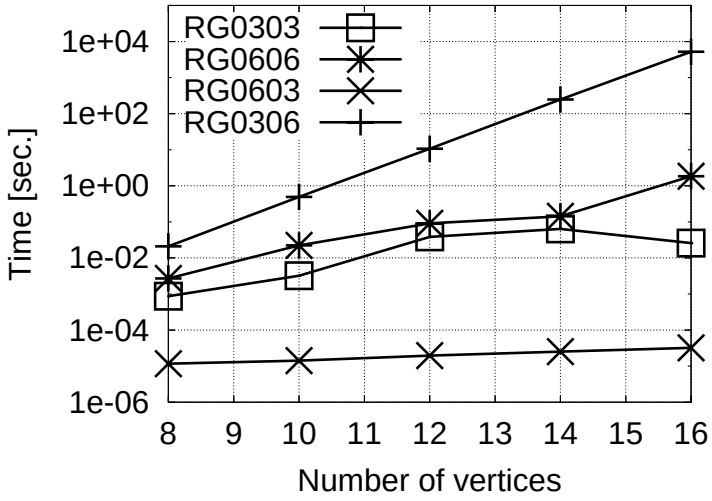


図 22: ソフトウェア実装の実行時間

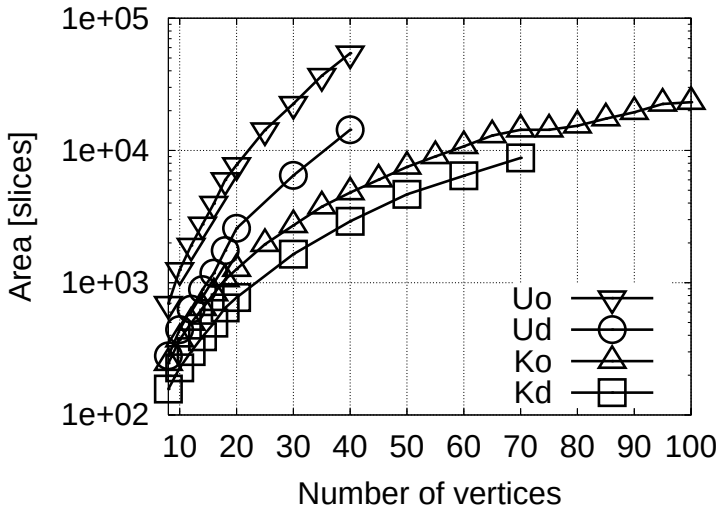


図 23: 回路規模 (RG0306)

ス (slice) で示した．Ud の回路規模は Uo の 40.3〜26.4% ($8 \leq ple40$)，Kd の回路規模は Ko の 63.8〜61.3% ($8 \leq p \leq 70$) になった．Ullmann のデータ依存回路も小西のデータ依存回路もオリジナルの回路規模より減少することが確認できた．

回路規模が小さいということは，より大きな頂点数の入力グラフのための回路が実装できることを意味する．本研究で用いた XC2V3000 を例にすると利用可能な回路資源は 14336 スライスあり，Uo なら $p < 25$，Ud なら $p < 40$，Ko も Kd も $p < 70$ の範囲で実装できる．時間の関係上 Kd の回路規模は $p \leq 70$ の範囲でしか測定していないが，Kd が Ko よりさらに大きな頂点数程度まで実装できると予想される．

Ullmann の提案回路の必要となる回路リソースは $O(p^3)$ となることは過去の研究で明らかになっている [3]．また、小西の提案回路の必要となる回路リソースは $O(p^2)$ である [3]．Ud と Uo の回路規模の増加傾向が似ていることから，データ依存回路で実装してもオリジナルの回路規模のオーダーと同じになると考えられる．同様のことが Ko と Kd でも考えられる．

5.4 回路生成時間

表 3: 回路規模と回路生成時間の内訳 [秒]

設計	p	回路規模 [スライス数]	VHDL 生成	論理合成	テクノロジマッピング	配置配線	ビット生成	ダウンロード	T_{gen}
Kd	8	157.07	0.017	24.74	5.25	29.83	23.45	41.86	125.14
	10	227.27	0.018	28.11	5.77	32.36	23.58	41.81	131.65
	12	301.65	0.019	32.54	6.46	35.52	23.65	41.81	140.00
	14	390.29	0.021	38.93	7.29	38.84	23.84	41.81	150.73
	16	495.12	0.022	48.45	8.03	44.63	24.00	41.82	166.95
Ud	8	279.66	0.021	32.91	6.34	33.17	23.62	41.81	137.87
	10	442.72	0.024	41.07	7.73	41.45	24.03	41.82	156.13
	12	630.60	0.028	51.74	9.44	53.08	24.40	41.81	180.51
	14	886.50	0.033	67.76	11.76	99.34	25.11	41.81	245.82
	16	1180.01	0.040	89.55	14.53	128.53	25.81	41.82	300.28

Kd と Ud の回路生成時間の内訳を表 3 に示す．個々の値は 100 組の入力グラフから回路を作成する処理にかかった時間の平均値である．なお，回路生成時間は回路規模に大きく依存するので回路規模の平均値も載せた (表 3)．頂点数の変化が，特に論理合成と配置・配線時間に影響することが読み取れる．これは頂点の増加で回路規模が増加するからと考えられる．ビットストリームのダウンロード時間は選択した FPGA により決まる．より多くの回路資源をもつ大規模な FPGA はより長いダウンロード時間を要する．

図 24 に回路生成時間の平均を示す．回路生成時間は Kd の 1.10〜1.80 倍になる ($8 \leq p \leq 16$)．同じ回路生成時間なら Kd の方がより大きい頂点数の入力グラフのためのデータ依存回路を作成できる．

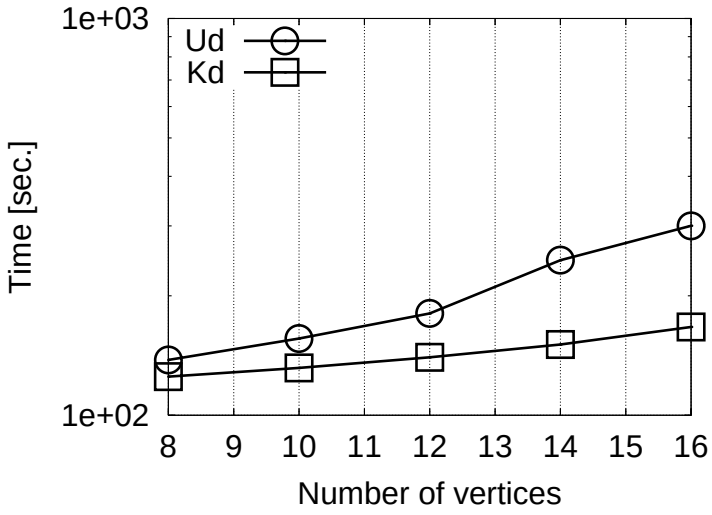


図 24: 回路生成時間 (RG0306)

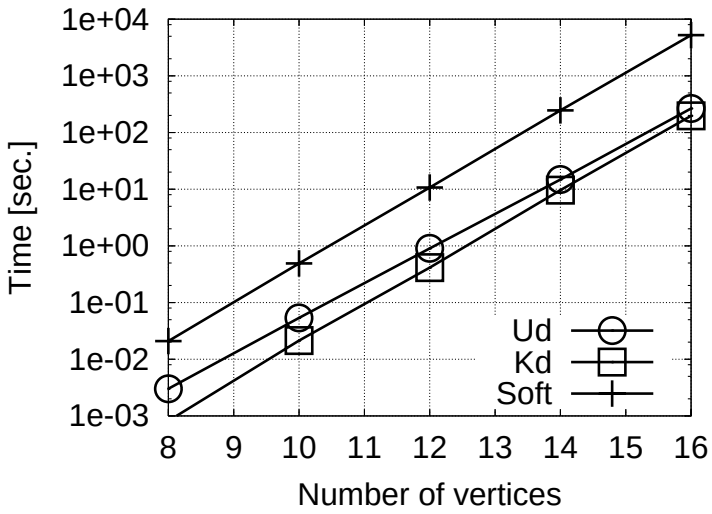


図 25: 実行時間 (RG0306)

5.5 実行時間

図 25 に 100 組の入力グラフから作成した回路の実行時間の平均値とソフトウェアの平均実行時間 (Soft) を示す．ソフトウェア実装の動作環境は表 2 に載せたとおりである．

実行時間で見るとソフトウェアの平均実行時間より Kd と Ud の方が短時間で判定を終了することが分かる．$8 \leq p \leq 16$ では Kd の実行時間が一番短い．$p = 16$ の場合，実行時間を実時間で比較すると，Kd は 5022.61 秒，Ud は 4953.61 秒ソフトウェアより早く判定を終了する．これはつまり，Kd は Soft の 26.5 倍，Ud は Soft の 19.6 倍の演算性能である．しかし，Kd と Ud はデータ依存回路なので回路生成時間を含めて考える必要がある．

5.6 総実行時間

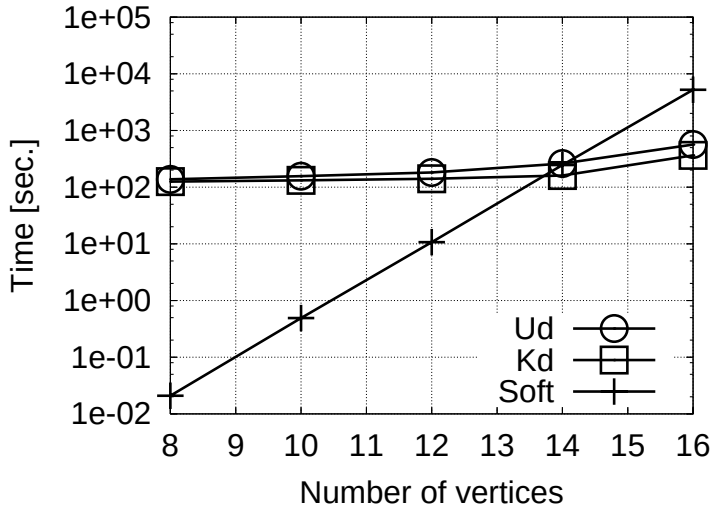


図 26: 総実行時間 (RG0306)

図 26 にデータ依存回路の総実行時間の平均とソフトウェア実装の平均実行時間 (Soft) を示す．データ依存回路では回路生成時間があるので，回路生成時間がソフトウェア実装の実行時間より短い間は実用的ではない．ところが部分グラフ同型判定問題は一般的に NP 完全問題なので，扱う頂点数が増大するにつれ実行時間は急激に増加し，回路生成時間が実行時間に対して無視できる程に小さくなる．RG0306 を用いた結果では，$14 \leq p$ のとき，Kd がソフトウェアより短時間で終了した．また，$16 \leq p$ のとき，Kd も Ud もソフトウェアより短時間で終了した．

$p = 16$ の場合，総実行時間を実時間で比較すると，Kd は 4855.66 秒，Ud は 4653.32 秒ソフトウェアより早く終了する．これはつまり，回路生成時間を含めても Kd は Soft の 14.4 倍，Ud は Soft の 9.2 倍高速である．

5.7 外挿による考察

頂点数の大きな問題の結果は興味深いが，100 組のデータの平均値を測定するのは事実上無理である．既述したとおりで実行時間の測定に時間がかかり過ぎるからである．そこで，近似モデルをたて，

パラメータをフィッティングして考察する．以下のモデル式 15〜18 を用いる．

$$Area_K(x) = k_0 + k_1 x + k_2 x^2 \tag{15}$$
$$Area_U(x) = k_0 + k_1 x + k_2 x^2 + k_3 x^3 \tag{16}$$
$$T_{gen}(x) = k_o \exp(k_1 x) + k_2 \tag{17}$$
$$T_{exec}(x) = k_o \cdot fact(k_1 x) \tag{18}$$
$$fact(x) \approx \sqrt{2\pi x}(x/e)^x \tag{19}$$

式 15 は Ko と Kd の回路規模モデルである．小西の提案回路の回路規模は $O(p^2)$ で増加するので 2 次式にした．また，式 16 は Uo と Ud の回路規模モデルで，Ullmann の提案回路の回路規模は $O(p^3)$ で増加するので 3 次式にした．

式 17 は回路生成時間のモデルである．回路生成時間は合成系に依存するので多項式時間になるのか指数時間になるのか実質わからない．そこで幾つかモデルを考えてフィッティング後とフィッティングに用いたデータの値との誤差が比較的小さくなるモデルとして採用したのが式 17 である．

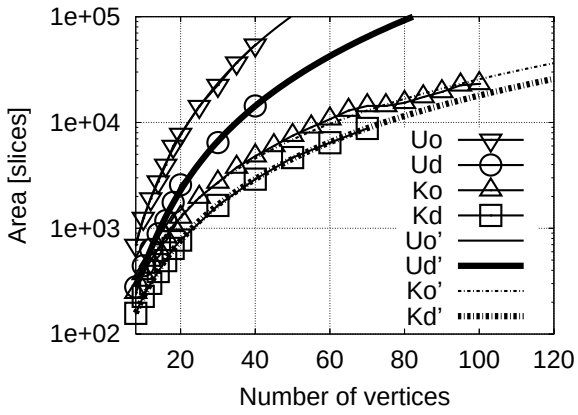
式 19 は $x!$ 計算の近似式である．式 18 は実行時間のモデルである．部分グラフ同型判定は一種の探索問題で，調べなければならない頂点対応の組合せが ${}_{p_\alpha}\mathrm{P}_{p_\beta}$ 存在する．評価条件として $p = p_\alpha = p_\beta$ した．このとき頂点の組合せは $p!$ 存在するので式 18 を実行時間のモデルに採用した．

上で立てたモデル式に対し，パラメータフィッティングを GNUPLOT 3.8j を用いて行う．k_i ($i = 0,\ 1,\ 2,\ 3$) がパラメータで，フィッティングに用いるデータは実測の 100 平均である．フィッティングの範囲は回路規模は Uo と Ud が $8 \leq p \leq 40$，Ko が $8 \leq p \leq 100$，Kd が $8 \leq p \leq 70$ である．回路生成時間と実行時間は $8 \leq p \leq 16$ での結果を用いた．

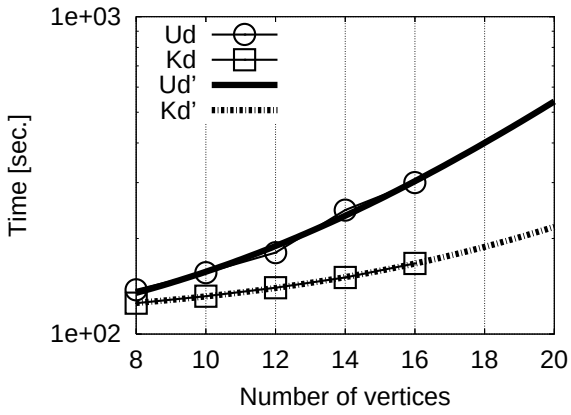
モデルの妥当性についてであるが，式 15 と 16 は，回路規模のオーダーから決めたモデルなので根拠がしっかりしている．しかし，式 17 と 18 は，正しいモデルである言い切れない．そこで実際に，フィッティングに用いたデータとパラメータ抽出後の外挿したグラフを重ねて表示したものを示す．回路規模は図 27(a)，回路生成時間は 27(b)，実行時間は図 27(c)，総実行時間は図 27(d)，AT 積は図 27(e) である．図中の Ko ’は，Ko に対してパラメータフィッティングして得られた外挿結果である．同様に Kd ’，Uo ’，Ud ’，Soft ’はそれぞれ Kd，Uo，Ud，Soft の外挿結果である．

表 4: 得られたフィッティングパラメータ (RG0306)

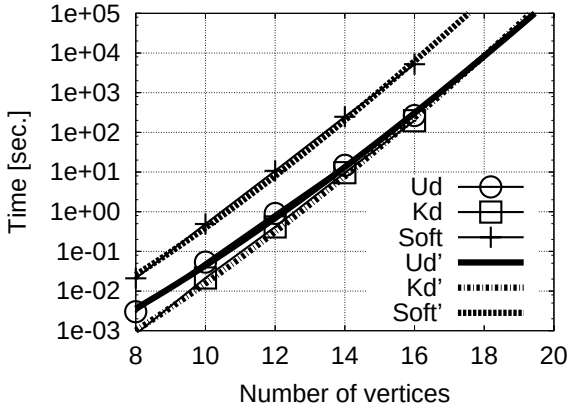
Item	Design	k_0	k_1	k_2	k_3
Area	Ko	-96.233	22.754	2.344	—
	Uo	11.603	12.193	3.228	0.751
	Kd	47.704	-0.573	1.805	—
	Ud	254.621	-37.782	4.040	0.143
T_{gen}	Kd	4.620	0.158	109.009	—
	Ud	11.520	0.184	84.711	—
T_{exec}	Soft	6.468e-5	0.710	—	—
	Kd	2.640e-6	0.708	—	—
	Ud	1.696e-5	0.667	—	—



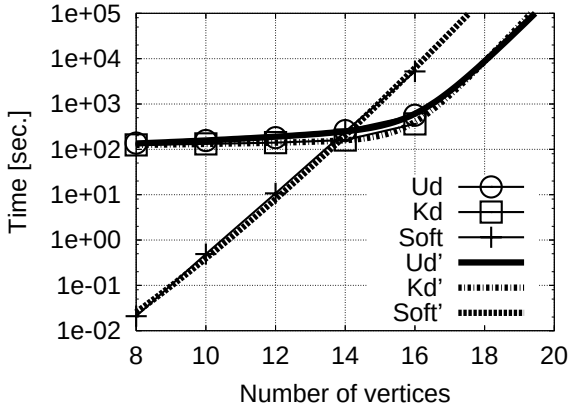
(a) 回路規模



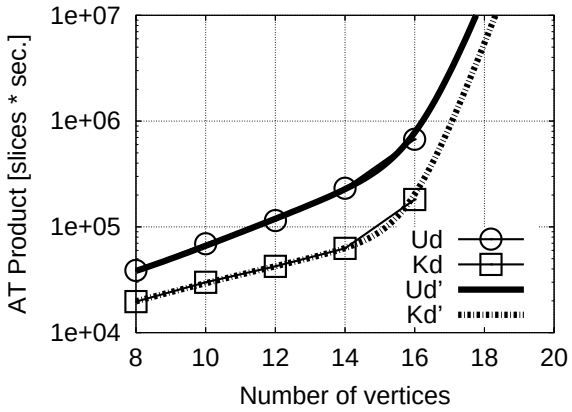
(b) 回路生成時間



(c) 実行時間



(d) 総実行時間



(e) AT 積

図 27: 外挿結果とフィッティングに用いた結果を重ねたグラフ (RG0306)

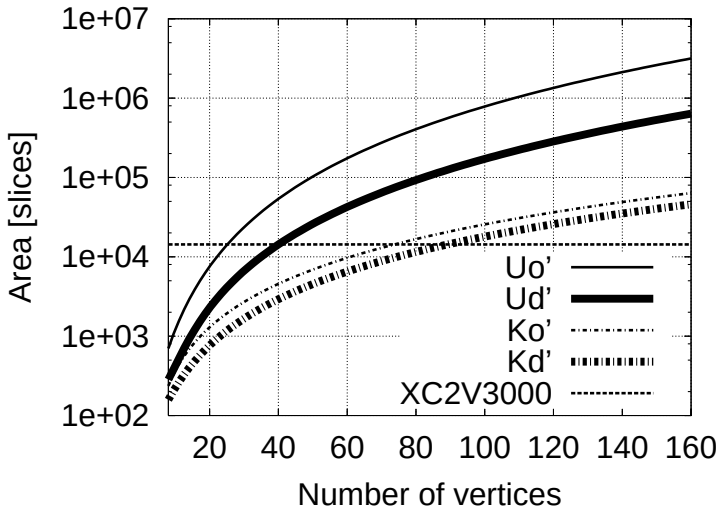


図 28: 回路規模 (RG0306)

図 28 に外挿した回路規模のグラフを示す．図 23 では Kd の実装可能な頂点数が正確に判断できなかった．しかし，外挿結果 (図 28) から Kd が $p < 90$ の範囲で実装できそうであることを示す結果を得た．

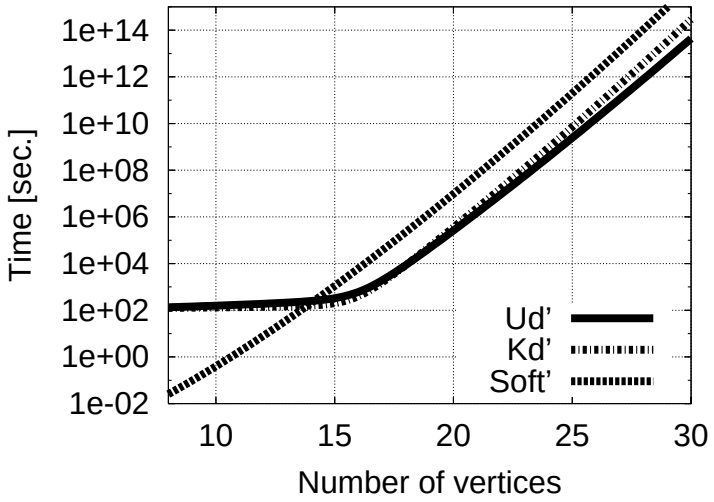


図 29: 総実行時間 (RG0306)

次に外挿した総実行時間を図 29 に示す．頂点数が増加することで Kd と Ud の性能は入れ替わる．これは枝刈りの効率により Kd と Ud の実行時間が入れ替わるためである (図 29)．頂点数が大きい範囲でもデータ依存回路の有用性を示す結果である．

一般に専用回路では高速化のために演算器の並列化を施す．そのために高速化には回路規模の増加をともなうため，回路規模と実行時間はトレードオフの関係にある．データ依存回路では回路規模が小さければ回路生成時間が減少するので AT 積の振る舞いが一般の論理回路とは異なる．

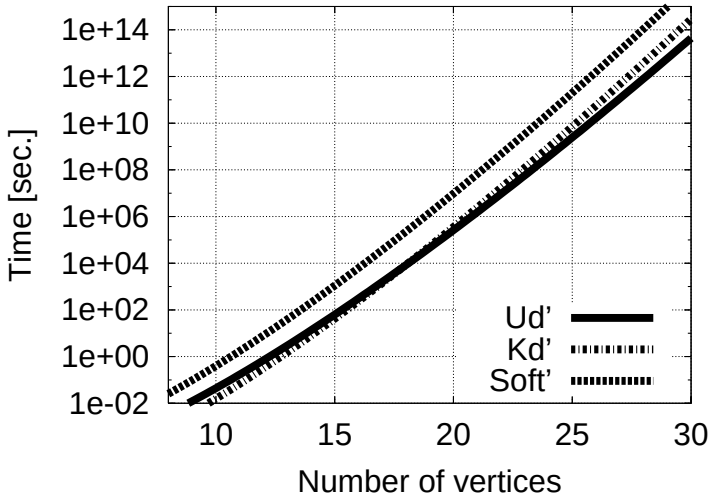


図 30: 実行時間 (RG0306)

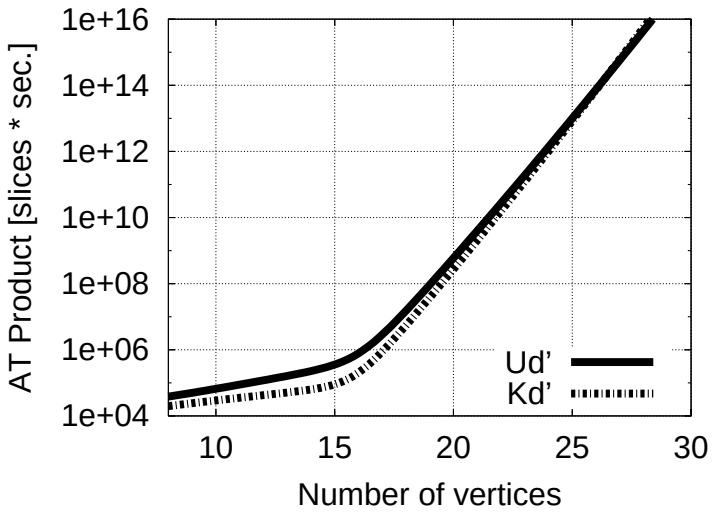


図 31: AT 積 (RG0306)

図 31 に外挿した AT 積を示す．頂点数が大きくなると Ud の方が総実行時間を短くできるのに対し，回路規模では Kd が小さい．RG0306 の場合，Kd の方がコストパフォーマンスが優れた設計であることを示した．

5.8 他のデータセットによる評価

5.8.1 IN0306 による評価

今まではランダムグラフを用いた評価結果を示した．ランダムグラフでは非連結なグラフも含まれるので実用的な応用問題を考えたときに入力データに制限 (意味のある条件) を与える必要がある．ここで用いるデータは連結グラフはウドーンが用いた評価用データセットの “$G_\alpha \subseteq G_\beta$ の場合の入力データ [8]” に相当する．

G_α と G_β には辺密度が与えられており，生成時に全域木を作成した後に指定した辺密度になるように辺の数を増やす．辺密度は頂点数が離散値である関係上指定した値にならない場合がある．そこで平均で指定値に近づくようにした (ただし分散が最小となるようにしている)．その後で G_α を元にして更に指定した辺密度になるように G_β を作成する．このデータセットは部分グラフ同型になる頂点の組合せを 1 つ以上持つ．

ここで用いるデータセットは上で説明した “$G_\alpha \subseteq G_\beta$ の場合” の G_α と G_β の辺密度をそれぞれ 0.3 と 0.6 とし，$p = p_\alpha = p_\beta$ としたものである．

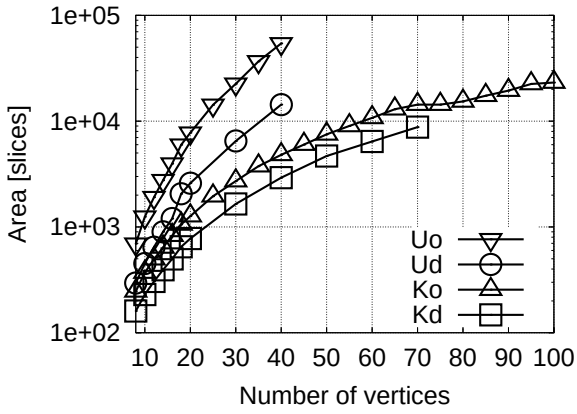
IN0306 を入力グラフとした場合の回路規模と回路生成時間，実行時間，総実行時間をそれぞれ図 32(a) と 32(b)，32(c)，32(d) に示す．

表 5: 得られたフィッティングパラメータ (IN0306)

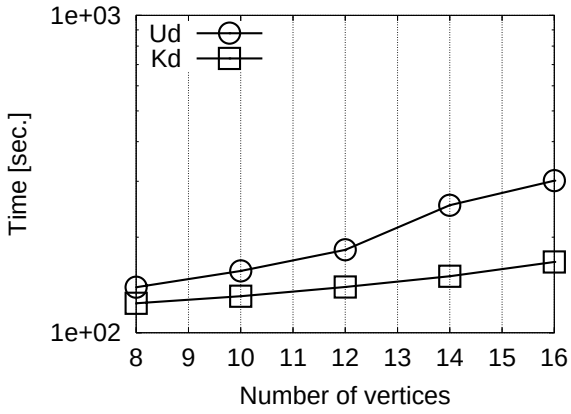
Item	Design	k_0	k_1	k_2	k_3
Area	Ko	-96.233	22.754	2.344	—
	Uo	11.603	12.193	3.228	0.751
	Kd	42.210	0.537	1.790	—
	Ud	571.270	-107.502	8.726	0.061
T_{gen}	Kd	5.358	0.152	105.809	—
	Ud	16.145	0.166	73.852	—
T_{exec}	Soft	9.208e-5	0.657	—	—
	Kd	2.354e-6	0.684	—	—
	Ud	2.411e-5	0.608	—	—

辺密度の観点から観ると IN0306 と RG0306 はほぼ同等である．回路規模についても RG0306 とほぼ同等の結果を得た．しかし，RG0306 に比べ，IN0306 は総実行時間の性能が入れ替わる点が頂点数の大きい方へ移動している．回路生成時間がある Kd と Ud の性能に対し，Soft は単純に実行時間が短くなったため性能が入れ替わる点が移動したと考えられる．また，Ud と Kd の判定性能を比べた場合，RG0306 より IN0306 の方が良くなっている．これは Ullmann の枝刈り方式が，RG0306 より IN0306 の方が効率的に働くためであると考えられる．RG0306 と同様に外挿して考察する．得られたパラメータは表 5 のとおりである．なお，フィッティングの範囲は RG0306 の場合と同じである．

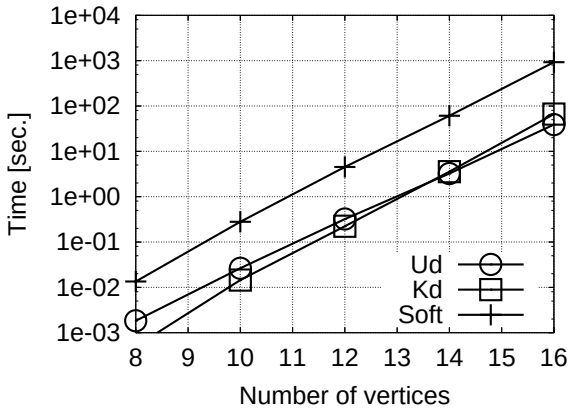
図 33(a) に外挿した回路規模のグラフを示す．IN0306 は RG0306 と同程度の辺密度であるため，



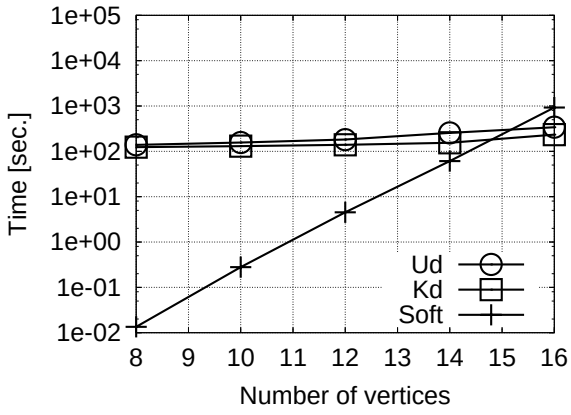
(a) 回路規模



(b) 回路生成時間

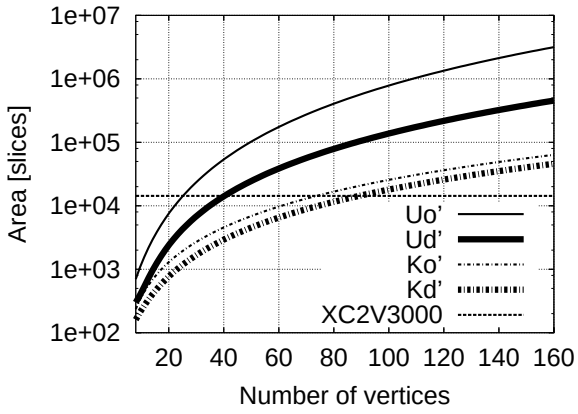


(c) 実行時間

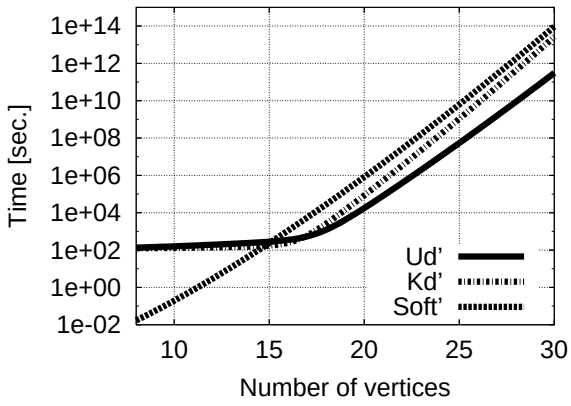


(d) 総実行時間

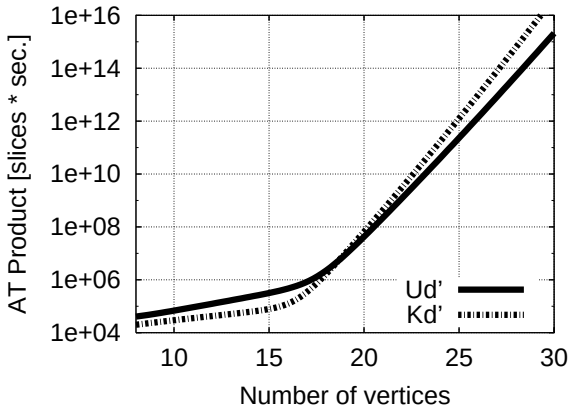
図 32: 実装結果 (IN0306)



(a) 回路規模



(b) 総実行時間



(c) AT 積

図 33: 外挿結果 (IN0306)

回路規模の傾向は同じである．

次に外挿した総実行時間を図 33(b) に示す．頂点数が増加することで Kd と Ud の性能は入れ替わる．これは RRG0306 と同様で，枝刈りの効率により Kd と Ud の実行時間が入れ替わるためだ．Kd と比べた Ud の性能は，RG0306 より IN0306 の場合の方が高い．これは Ullmann の提案枝刈り手法が効率的に働いているためだと考えられる．

外挿した AT 積を図 33(c) に示す．IN0306 の場合，Ullmann の提案手法が効率的に判定を行うので，回路規模が大きい Ud でも判定性能が高いため，AT 積が良くなった (頂点数 18 付近で入れ替わっている)．

5.8.2 Tree による評価

先行研究では Tree を入力グラフとした場合で Ullmann のデータ依存回路を評価した [4]．既に述べたが，先行研究 [4] では実装まで行っていない．そこで，本研究では Tree を入力グラフとした場合の評価の追試を行う．

Tree とは文字どおり「木」である．本研究では，必ず連結グラフになる全域木を扱う．データセット Tree については説明は論文 [4] にゆずる．ただし，G_α と G_β の頂点数 p_α と p_β には，RG0306 と同様に $p = p_\alpha = p_\beta$ という条件を加えた．

回路規模を図 34(a) に示す．Ud の回路規模は Uo の 27.1〜12.9%である ($8 \leq p \leq 40$)．Kd の回路規模は Ko の 53.7〜40.0%である ($8 \leq p \leq 70$)．Tree は連結グラフの中で最も辺密度が薄いデータセットであるので，回路規模の減少率が RG0306 や IN0306 に比べて高くなった．

回路生成時間を図 34(b) に示す．Kd の回路生成時間が Ud より短いのは Kd の方が Ud より回路規模が小さいためである．Ud と Kd の実装した頂点数の範囲が異なるのは，実行時間の関係上である．実行時間は図 34(c) に示したとおりである．Tree を入力グラフとした場合，Ullmann アルゴリズムと小西のアルゴリズムでは枝刈り効率が大きく異なり，小西のアルゴリズムを採用した Kd では，Ullmann のアルゴリズムを実装した Soft より実行時間が長大になる．なお，図 34(c) を見てわかるとおり Ud は Soft の判定性能の数十倍である．

総実行時間を図 34(d) に示す．図 34(d) では明らかでないが，$p = 40$ では，Soft の平均実行時間は 2323.49 秒，Ud の平均総実行時間は 2214.26 秒で，Ud が Soft より実時間で 109.23 秒短い．

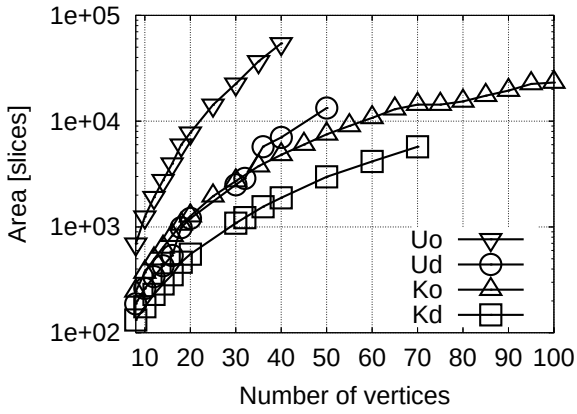
グラフの傾向から頂点数が増大すれば，回路生成時間より実行時間の総実行時間に占める割合が高くなるので，性能比はより高くなると考えられる．つまり，データ依存回路は Tree のようなデータセットでも優位性があると考えられる．

Tree を用いた評価では，実行時間が極端に短くなる入力グラフの組が存在する．それは Tree の場合，Ullmann のアルゴリズムでは一番最初に呼び出す refinement procedure で FAIL exit する組が多く含まれることを意味する．

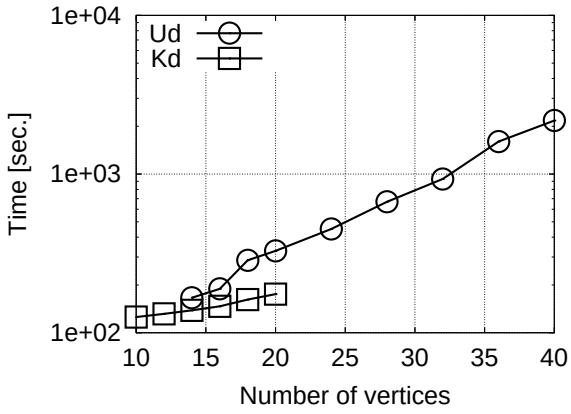
そこで，VHDL 生成前に Ullmann のアルゴリズムの refinement procedure を一度だけ呼び出して，FAIL exit したら終了し，そうでなかったらデータに依存した VHDL のソースコードを生成し，データ依存回路を実装して判定を行うようにした．この判定処理を VHDL 生成判定と呼ぶことにする．

VHDL 生成判定を加えた VHDL 生成プログラムの処理時間に発生するオーバヘッドは数ミリ秒と小さい．VHDL 生成判定によって回路生成時間と総実行時間が改善される．実行時間は測定誤差の範囲でしか変化しない．

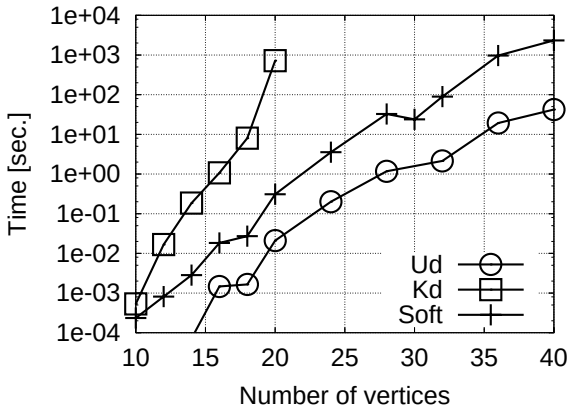
VHDL 生成判定ありの実装結果を示す．回路生成時間を図 35(a)，総実行時間を図 35(b) にそれぞ



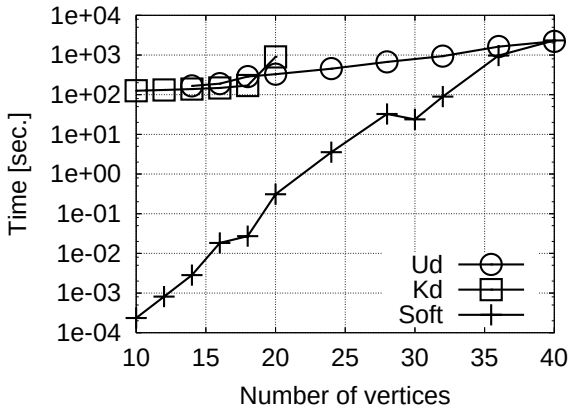
(a) 回路規模



(b) 回路生成時間

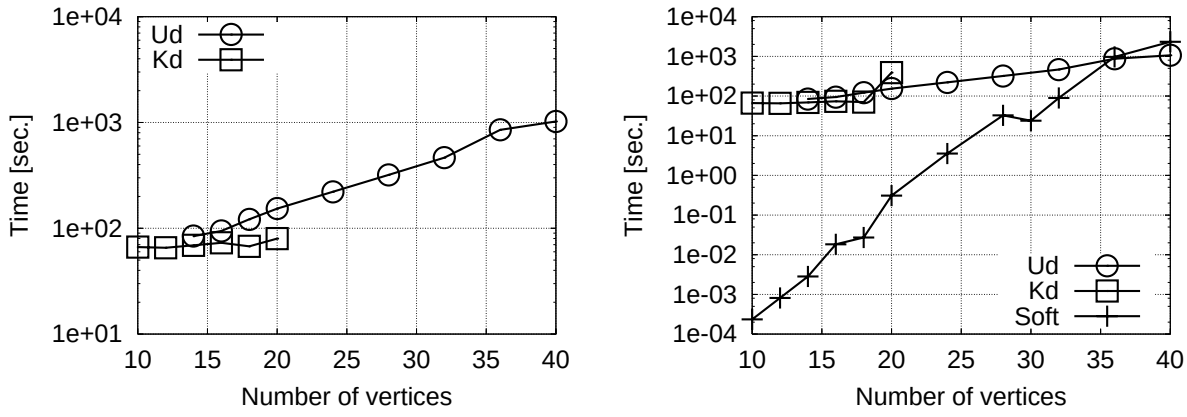


(c) 実行時間



(d) 総実行時間

図 34: 実装結果 (Tree)



(a) 回路生成時間　　(b) 総実行時間

図 35: VHDL 生成判定ありの実装結果 (Tree)

れ示す．Tree の場合，refinement procedure が FAIL exit する組が半分近く存在するので回路生成時間が半分近く短くなった．よって，$p = 40$ では，Ud の総実行時間は 1063.94 秒となり Soft の 2.1 倍高速になった．

5.8.3 RG0303_X100 による評価

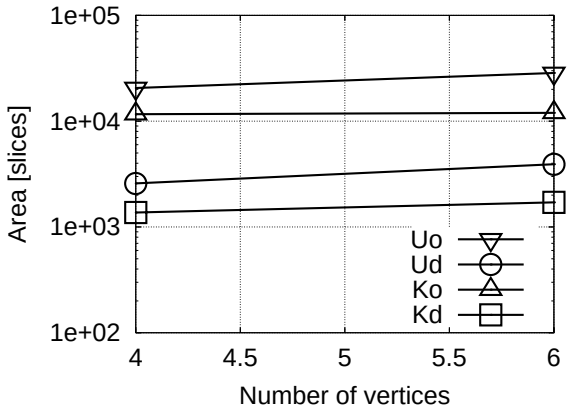
今までの評価では $p_\alpha = p_\beta$ としたが，ここでは $p_\beta = 100$ として p_α を変化させて評価する．これは実用上の応用問題を考えたとき頂点数 100 や 1000 といった入力グラフをデータ依存回路で扱えるかを検証する為である．2 つのグラフの頂点数を共に 100 や 1000 とする評価は現状では困難である．そこで，一方の入力グラフの頂点数は小さく，もう一方は大きいという応用を想定した (実際，このような片側だけ頂点数が小さい応用はデータベースマッチングなどで応用が考えられる)．なお，評価に用いるのはランダムグラフである．実行時間測定の関係上，G_α と G_β の辺存在確率をそれぞれ 0.3 と 0.3 とした (この辺存在確率の組合せに意味があるわけではない．ソフトウェアの実行時間が実用的な時間内で測定できるように選んだだけである)．

回路規模を図 36(a) に示す．グラフの横軸は p_α である．Uo の回路規模は，$p_\alpha = 6$ では 20588 スライス，$p_\alpha = 6$ では 28557 スライスと XC2V3000 の回路リソース (14336 スライス) を大幅に上回った．一方，Ud の回路規模は Uo の 13.7%で，XC2V3000 のリソース量の 27.3%である ($p_\alpha = 6$)．Uo では実装できない回路が Ud では実装できる一例である．なお，Kd の回路規模は Ko の 14.3% で，XC2V3000 のリソース量の 11.9%である ($p_\alpha = 6$)．

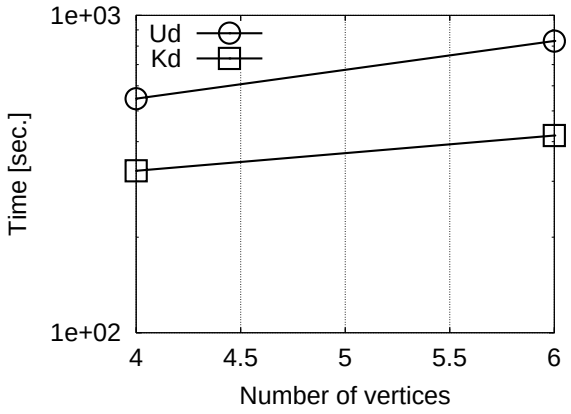
回路生成時間を図 36(b) に示す．回路規模が大きい Ud の方が回路生成時間が長くなるという結果をここでも得た．Ud の回路生成時間は Kd の 2.0 倍である．

実行時間を図 36(c) に示す．RG0303 X100 の場合，判定性能が高いのは Kd であった．$p_\alpha = 6$ にて，Kd は Soft の 16.4 倍，Ud は Soft の 12.5 倍の判定性能である．

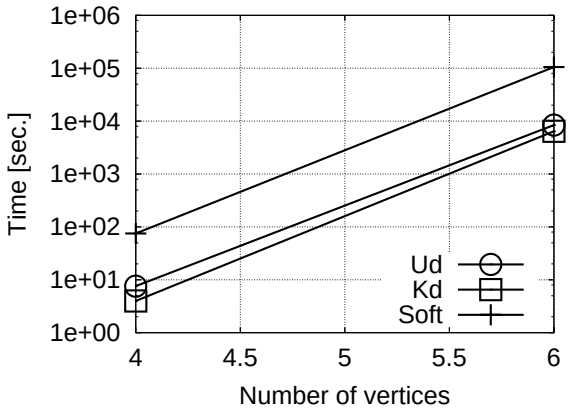
回路生成時間を含めた総実行時間を図 36(d) に示す．$p_\alpha = 6$ では，Ud と Kd は共に Soft より高速な判定を実現した．$p_\alpha = 6$ では，Kd が一番高速であった．このとき Kd は Soft の 15.4 倍，Ud



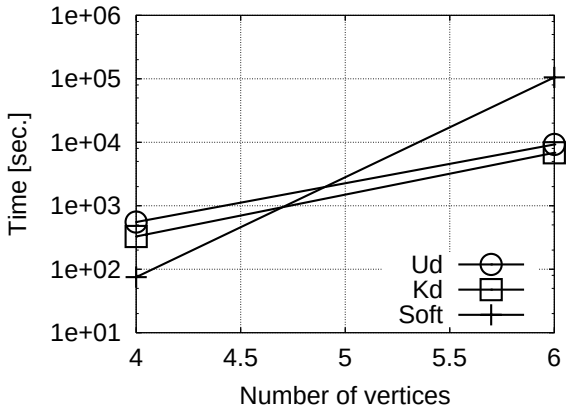
(a) 回路規模



(b) 回路生成時間



(c) 実行時間



(d) 総実行時間

図 36: 実装結果 (RG0303_X100)

は Soft の 11.4 倍高速でとなる．なお，$6 < p_\alpha$ の評価結果がないのは実行時間の測定に時間がかかり過ぎるためである．実装できないことを意味するものでは決してない．

6　おわりに

本研究では複数のデータセットにてデータ依存回路を FPGA 上に実装し，回路規模と回路生成を含めた実行時間を評価した．RG0306 を用いた評価では頂点数 16 で，回路規模は Ud で Uo の 29.7%，Kd で Ko の 59.9%に減少し，ソフトウェアと比較した性能は (回路生成時間を含めても) Kd が 14.4 倍，Ud が 9.2 倍になった．

回路生成時間というオーバヘッドを含めてもデータ依存回路の実行時間がソフトウェアの実行時間より短くなる理由は，頂点数の増加によりソフトウェアの実行時間がデータ依存回路の回路生成時間より長くなるためである．

部分グラフ同型判定問題は NP 完全であるため，頂点数の増加に従って実行時間は指数的に増大する．従って，頂点数の大きな問題であるほど，専用回路による高速化が望まれる．しかし一方で，頂点数が大きいほど専用回路の実装に必要な論理規模も増加し，実装が困難になる．データ依存回路では同じ頂点数でも回路規模を小さくできるため，同じ論理規模の FPGA でも，より大規模な問題に適用することができる．特に，データ依存回路による実装を考えた場合，小西のデータ依存回路は回路規模が小さいため，回路生成が短時間で終了するので Ullmann のデータ依存回路に比べて有用であると期待される．

本研究では，部分グラフ同型判定問題をデータ依存回路で実装することにより，判定時間をソフトウェアより短縮できることを示した．他の応用問題に関してもデータ依存回路がソフトウェアより高速か，という問題は興味深いが，今後の課題としたい．また，本研究では外挿結果より頂点数の大きい場合の考察をしたが，用いたモデルには正当であると言い切れないものが含まれるので，より大きな頂点数での評価を今後の課題としたい．

本研究で用いた “回路記述生成” のためのツールは，アドホックな最適化処理を加えるためにアルゴリズム毎に作成したものである．今後は，アルゴリズムをデータ依存回路として変換するための記述方式の提案などができれば様々な分野でデータ依存回路を容易に実現できるようになると考えている．

謝辞

最後まで御指導してくださった市川先生に，この場をお借りしましてお礼申し上げます．また，3 年間，私と共に楽しいときや苦しいときをとも過ごした岸本君他市川研究室の皆様にも感謝いたします．最後に大学に通わさせてくださった両親に感謝いたします．

参考文献

[1] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, 1979.

[2] J. Gross and J. Yellen. *Graph Theory and Its Applications.* CRC Press, 1998.

[3] S. Ichikawa, H. Saito, L. Udorn, and K. Konishi. Trade-offs in custom circuit designs for subgraph isomorphism problem. *IEICE Transactions on Information and Systems*, E86-D(7):1250–1257, 2003.

[4] S. Ichikawa and S. Yamamoto. Data dependent circuit for subgraph isomorphism problem. *IEICE Transactions on Information and Systems*, E86-D(5):796–802, 2003.

[5] Xilinx Inc. *Virtex^{TM}-II Platform FPGAs: Introduction and Overview*, 2002.

[6] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[7] 小西 幸治. 部分グラフ同型判定アルゴリズムの FPGA を用いた実装手法. 修士論文, 豊橋技術科学大学知識情報工学系, 1999.

[8] ラターナンセンタン・ウドーン. 部分グラフ同型判定アルゴリズムの FPGA を用いた実装. 修士論文, 豊橋技術科学大学知識情報工学系, 2000.

[9] 山本浩司, 市川周一. データ依存回路による隣接判定方式の評価. 2003 年電子情報通信学会総合大会 D-6-4, p. 64, 2003.

[10] 市川周一, L. ウドーン, 小西幸治. 部分グラフ同型判定アルゴリズムの FPGA による実装と評価. 情報処理学会論文誌ハイパフォーマンスコンピューティングシステム, 41(SIG5(HPS1)):39–49, 2000.

A　測定環境について

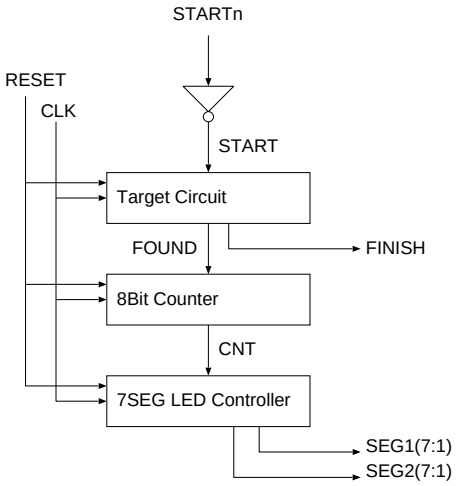


図 37: 実際の実装回路

実装した回路のデバッグを行うために部分グラフ同型になる頂点の組合せを発見する毎に 8 ビットカウンタをカウントアップするようにした．その値を 7 セグメント LED で表示するための回路と一緒に付加されている．5 章で評価した実装回路は図 37 に示したものを評価した．つまり，若干，余分な論理が含まれているが，これは全体に対して無視できる回路規模である．図 37 の Target Circuit の部分は評価対象の回路にあたる．

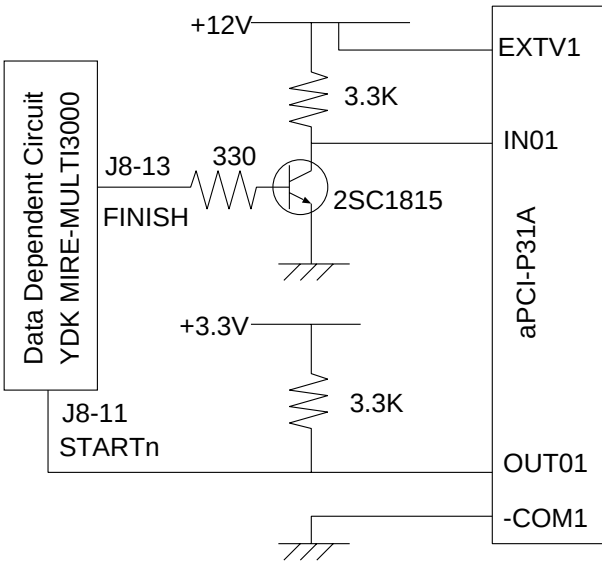


図 38: PC と FPGA ボードを結ぶ電圧レベル変換回路

実行時間測定プログラムを正しく動作させるためには図 38 のようにして FPGA ボードとパラレル I/O ボードを接続する必要がある．

B　ソフトウェア実装について

データ依存回路の性能を評価するための比較対象として，Ullmann のアルゴリズムをソフトウェアで実装したもの (Soft) の実行時間を測った．比較対象として Soft を十分に最適化して性能向上の工夫を尽くすべきである．しかし，実行環境に特化した最適化方法に関しては，再現性や一般性との兼ね合いからいって，採用すべきか判断が難しい面もあると考える．これで充分であるか否かはともかく，本研究での適用手法は以下のとおりである．

ビット演算 計算は多ビットの論理積と論理和，否定からなる．そこで，計算機の 1 ワードに多ビットを分割し，割り当てた．つまり，Athlon XP 2600+なら 64 ビット長のデータは 2 ワードに分割し，32 ビット単位で演算を行う．

データ構造 多ビットをワードに割り当てを行うことで演算に無駄がなくなった．さらにデータ領域が小さくなるように工夫した．

最内周ループ 最内周ループが長くなるようにした．

値の再利用 2 回以上参照される値は 1 回計算した後，メモリに保持するようにした．

メモリアクセス 連続アドレスアクセスになるようにした．

作業メモリ確保 作業メモリ確保を必要最低限回数に抑えた．また，データ構造に工夫してあるので作業メモリのアドレス空間は抑えられている．

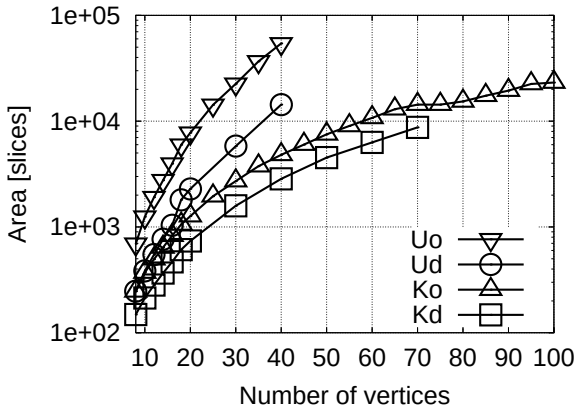
C　本文に載せなかった実験結果

C.1　RG0204 の結果

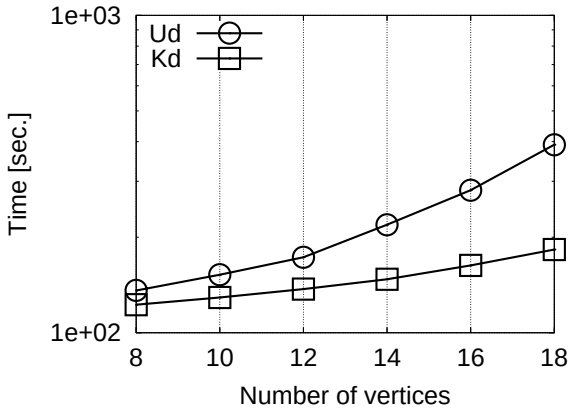
5 章で用いた RG0306 の辺存在確率を変更したデータセットである．G_α と G_β の辺存在確率はそれぞれ 0.2 と 0.4 とした．回路規模を図 39(a)，回路生成時間を図 39(b)，実行時間を図 39(c)，総実行時間を図 39(d) にそれぞれ示す．

表 6: 得られたフィッティングパラメータ (RG0204)

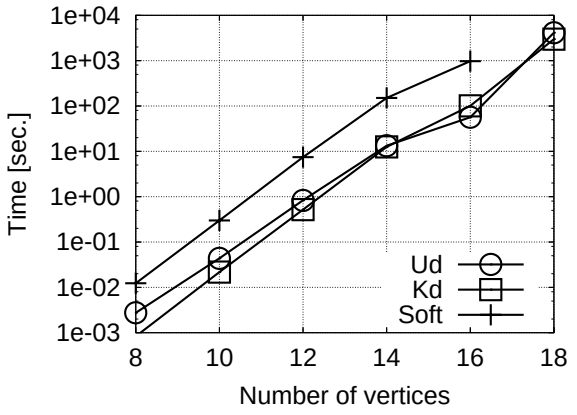
Item	Design	k_0	k_1	k_2	k_3
Area	Ko	-96.233	22.754	2.344	—
	Uo	11.603	12.193	3.228	0.751
	Kd	52.949	-2.380	1.803	—
	Ud	345.899	-56.692	4.538	0.132
T_{gen}	Kd	5.183	0.150	105.625	—
	Ud	3.257	0.247	112.508	—
T_{exec}	Soft	7.854e-05	0.664	—	—
	Kd	4.528e-06	0.676	—	—
	Ud	3.290e-05	0.615	—	—



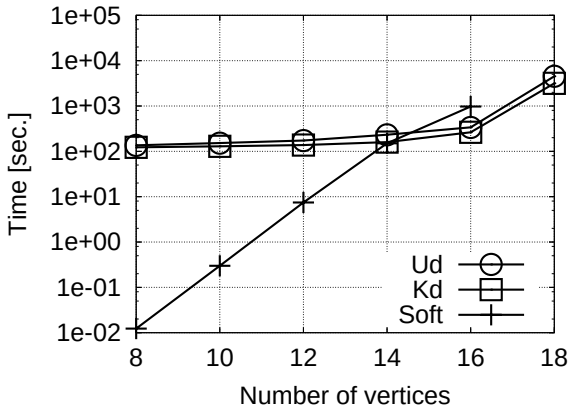
(a) 回路規模



(b) 回路生成時間



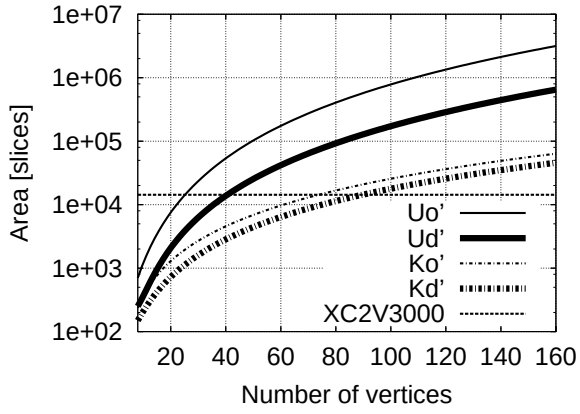
(c) 実行時間



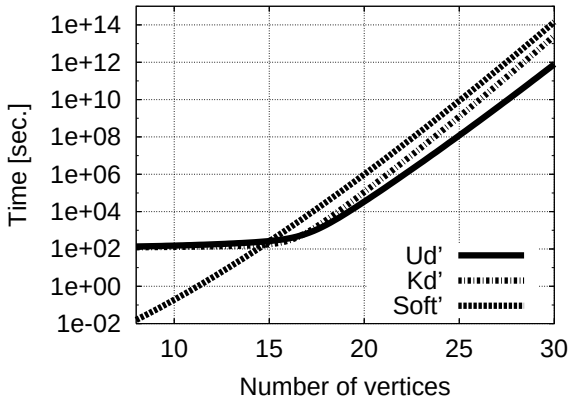
(d) 総実行時間

図 39: 実装結果 (RG0204)

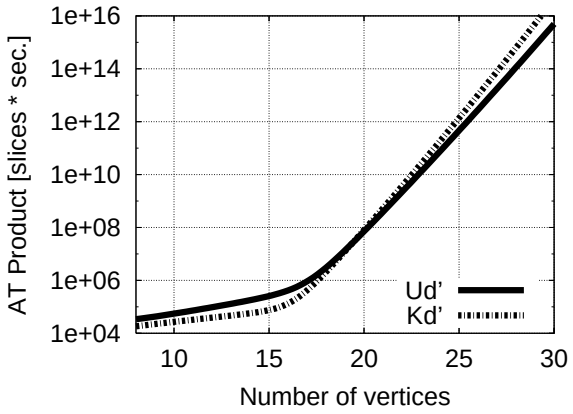
5.7 章と同様に RG0204 の場合の外挿結果を示す．得られたフィッティングパラメータを表 6 に載せる．フィッティングの範囲は回路規模は Uo と Ud が $8 \leq p \leq 40$，Ko が $8 \leq p \leq 100$，Kd が $8 \leq p \leq 70$ である．ソフトウェアの実行時間は $8 \leq p \leq 16$ での結果を用いた．Ud と Kd の回路生成時間と実行時間は $8 \leq p \leq 18$ での結果を用いた．回路規模を図 40(a)，総実行時間を図 40(b)，AT 積を図 40(c) に示す..



(a) 回路規模



(b) 総実行時間



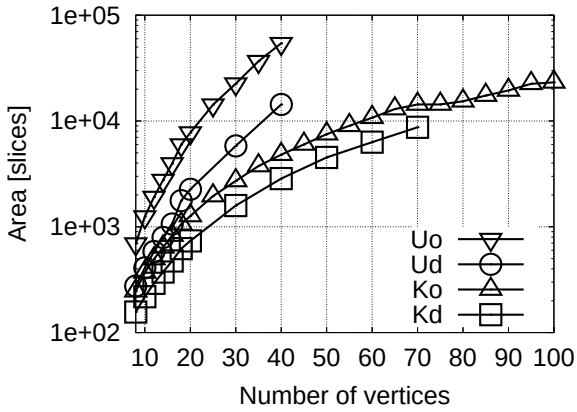
(c) AT 積

図 40: 外挿結果 (RG0204)

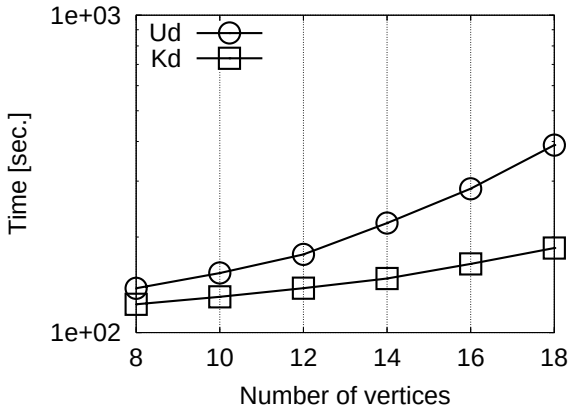
C.2 IN0204 の結果

5.8.1 章で用いた IN0306 の辺密度を変更したデータセットである．G_α と G_β の辺密度はそれぞれ 0.2 と 0.4 とした．回路規模を図 41(a)，回路生成時間を図 41(b)，実行時間を図 41(c)，総実行時間を図 41(d) にそれぞれ示す.

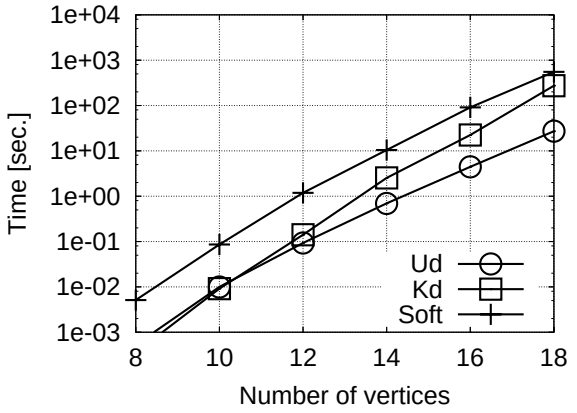
5.7 章と同様に IN0204 の場合の外挿結果を示す．得られたフィッティングパラメータを表 7 に載



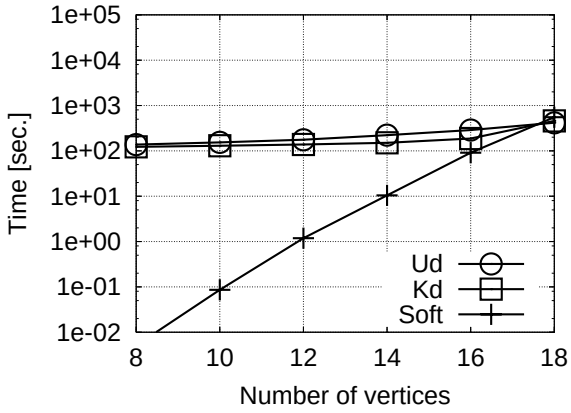
(a) 回路規模



(b) 回路生成時間



(c) 実行時間



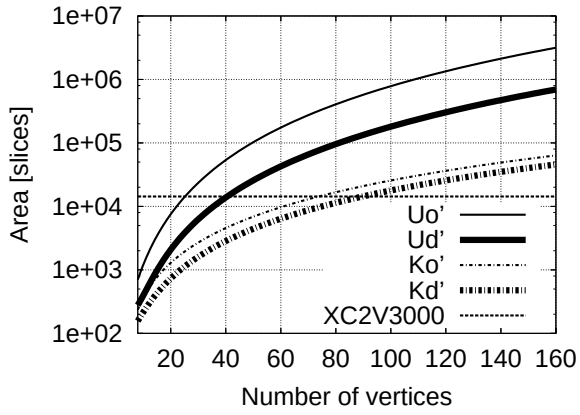
(d) 総実行時間

図 41: 実装結果 (IN0204)

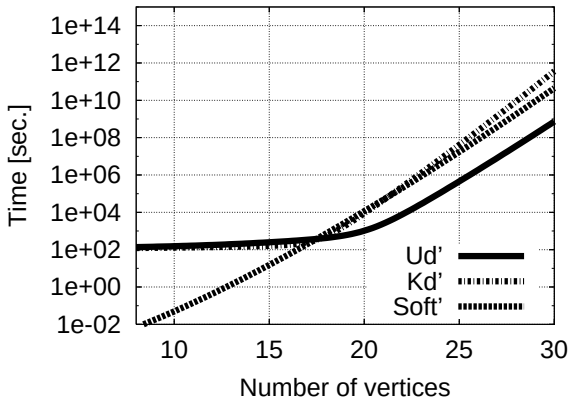
表 7: 得られたフィッティングパラメータ (IN0204)

Item	Design	k_0	k_1	k_2	k_3
Area	Ko	-96.233	22.754	2.344	—
	Uo	11.603	12.193	3.228	0.751
	Kd	60.610	-2.398	1.801	—
	Ud	306.528	-41.476	3.545	0.149
T_{gen}	Kd	5.347	0.150	105.338	—
	Ud	3.864	0.238	111.830	—
T_{exec}	Soft	1.294e-4	0.566	—	—
	Kd	3.832e-6	0.631	—	—
	Ud	1.852e-5	0.541	—	—

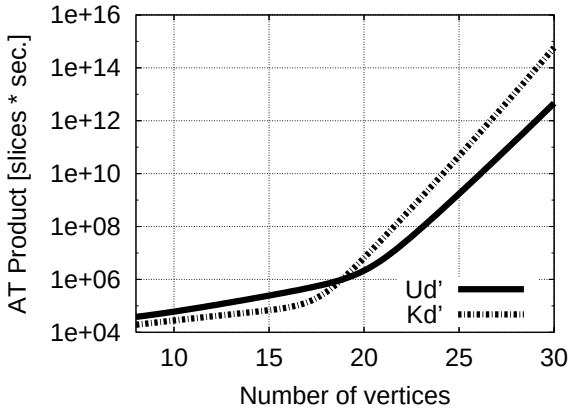
せる．フィッティングの範囲は回路規模は Uo と Ud が $8 \leq p \leq 40$，Ko が $8 \leq p \leq 100$，Kd が $8 \leq p \leq 70$ である．ソフトウェアの実行時間は $8 \leq p \leq 18$ での結果を用いた．Ud と Kd の回路生成時間と実行時間は $8 \leq p \leq 18$ での結果を用いた．回路規模を図 42(a)，総実行時間を図 42(b)，AT 積を図 42(c) に示す..



(a) 回路規模



(b) 総実行時間



(c) AT 積

図 42: 外挿結果 (IN0204)