社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－SLDM－127 (34)
2006／11／30

最小p-準クリーク被覆問題に対するハードウェアアルゴリズム

渡辺 秀一† 北道 淳司† 奥山 祐市† 黒田 研一†

† 会津大学大学院コンピュータ理工学研究科 〒965-8580 福島県会津若松市一箕町鶴賀
E-mail: †{m5091212,kitamiti,okuyama,kuroken}@u-aizu.ac.jp

あらまし 本論文では最小p-準クリーク被覆問題に対するハードウェアアルゴリズムを提案し，それをFPGA上へ実装する．本問題はNP完全として知られ，応用分野の一つに遺伝子発現プロファイル解析がある．本問題の計算時間の削減は遺伝子解析の高速化にとって重要である．本アルゴリズムは計算時間の削減のためニューラルネットワークを採用する．リングネットワークを用いた本アーキテクチャは全てのモジュールが互いに独立して並列動作が可能であるので，本アルゴリズムを高速に実行できる．本手法は既存手法より解の探索能力が高く，計算時間が小さいことを示す．
キーワード 最小p-準クリーク被覆問題，遺伝子発現プロファイル解析，ニューラルネットワーク

A Hardware Algorithm for the Minimum p-quasi Clique Cover Problem

Shuichi WATANABE†, Junji KITAMICHI†, Yuichi OKUYAMA†, and Kenichi KURODA†

† The Graduate School of Computer Science and Engineering, The University of Aizu
Tsuruga, Ikki-machi, Aizu-Wakamatsu City, Fukushima 965-8580 JAPAN
E-mail: †{m5091212,kitamiti,okuyama,kuroken}@u-aizu.ac.jp

Abstract This paper proposes a hardware algorithm for the minimum p-quasi clique cover problem, and it is implemented on FPGA. This problem is known as NP-complete and appears in the gene expression profile analysis. The reduction of the calculation time for this problem is important to accelerate the gene analysis. Our algorithm adopts a neural network for the reduction of the calculation time. Our architecture using a ring network can accelerate the execution of our algorithm because all modules can execute in parallel independently. We show our method is better than the existing methods in the searching ability of a solution and the calculation time.
Key words The Minimum p-quasi Clique Cover Problem, Gene Expression Profile Analysis, Neural Network

1. はじめに

最小p-準クリーク被覆問題は，連結グラフ$G=(V,E)$が与えられたとき，p-準クリークの数が最小になるように，Gの各頂点を少なくとも1個のp-準クリークで被覆する組み合わせを求める．ただし，Vは頂点集合であり，Eは頂点間を結ぶ枝集合である．N頂点p-準クリークとはN頂点クリークから数本の枝を除去して得られる連結グラフである．p-準クリークの各頂点がもつ枝の数は，完全グラフとの近さを表すパラメータpに依存する．最小p-準クリーク被覆問題はNP完全である[2]．

最小p-準クリーク被覆問題の応用分野の一つに遺伝子発現プロファイル解析が知られている[3]～[5]．遺伝子発現プロファイルは遺伝子情報の一つであり，様々な条件下において遺伝子から発現量を測定した結果により得られる集積データである．ここで，発現量とは遺伝子から生成されるタンパク質の量である．遺伝子発現プロファイルは特定の条件下で類似した発現量をもつ遺伝子集合を求めて解析されることが多い．その解析手法の一つとして遺伝子発現プロファイル解析を最小p-準クリーク被覆問題に帰着させる手法が知られている[3]～[5]．この手法は，まず，遺伝子をグラフの頂点に対応させ，遺伝子間の類似性が閾値以上であれば遺伝子に対応する頂点間を枝で結合することによりグラフを構成する．次に，そのグラフおよびパラメータpを最小p-準クリーク被覆問題の入力とする．そして，問題の出力として得られたp-準クリークを求めるべき遺伝子集合とする．松田らは最小p-準クリーク被覆問題に対する貪欲法を用いた発見的アルゴリズムを提案し，遺伝子発現プロファイルの解析能力を向上させている[3]，[4]．筆者らは松田らとは異なる制約条件下での遺伝子集合を求めるため，最小p-準クリーク被覆問題を応用した問題に対してハードウェアアルゴリズムおよびそれを専用ハードウェア上へ実装する手法を提案した[1]．

最小p-準クリーク被覆問題を用いた手法は遺伝子発現プロファイルの解析能力を向上させているが，最小p-準クリーク被覆問題はNP完全であり，入力グラフの頂点数が増加するにつれて計算が困難になる．松田らのアルゴリズムの計算量は入力

グラフの頂点数 N に対して $O(N^4)$ であるが, 今まで誰も注目しなかった生物学的に有用な可能性がある遺伝子集合を求めるため, 同一の遺伝子発現プロファイルに対して数百回から数千回程度の解析を必要とするので, 計算時間は増大する. 繰り返しの解析を必要とする理由は問題の最適解が必ずしも生物学的に有用であるとは限らず, 局所最適解の中に有用な解が含まれることが多いためである. また, 今後, 遺伝子発現プロファイルの数は急速に増加することが予想されており, 計算時間はさらに増大する. そこで, 膨大な計算量を必要とする遺伝子発現プロファイル解析に対して, より計算量の小さいアルゴリズムまたは専用ハードウェアを用いて計算時間を削減することが有効であると考えられる.

本研究では最小 p-準クリーク被覆問題に対する発見的アルゴリズムを提案する. そして, 本アルゴリズムを FPGA 上へ実装し, 本問題を数百回から数千回程度繰り返し解くことにより遺伝子発現プロファイル解析の高速化を図る. 本アルゴリズムは文献 [7] の最大クリーク問題に対する Hopfield ニューラルネットワーク (HNN) アルゴリズムに基づく. HNN は発見的かつ並列処理可能なアルゴリズムの実現が容易であり, ハードウェア上で効率的に並列実行できる. リングネットワークを用いた本アーキテクチャはモジュール間の接続およびモジュールの配置が単純なので回路の拡張性が高いなどの特徴をもつ.

2. 最小 p-準クリーク被覆問題

[定義 1] (p-準完全グラフ)

p-準完全グラフ $G = (V, E)$ は各頂点の次数が $\lceil p \cdot (|V| - 1) \rceil$ 以上となる連結グラフである [3]〜[5]. ただし, V は頂点集合であり, E は頂点間を結ぶ枝集合である. また, $|V|$ は G の頂点数である.

パラメータ p ($0 < p \leq 1$) は完全グラフとの近さを表す. p=1 のとき, 1-準完全グラフは完全グラフである. 図 1 に p-準完全グラフの例を示す. 図 1 (a), (b) および (c) の p の値はそれぞれ 1, 0.5, 0.2 であるので, 各頂点の次数はそれぞれ $\lceil 1 \cdot (8-1) \rceil$=7 以上, $\lceil 0.5 \cdot (8-1) \rceil$=4 以上, $\lceil 0.2 \cdot (8-1) \rceil$=2 以上である. 図 1 (a) は完全グラフである.

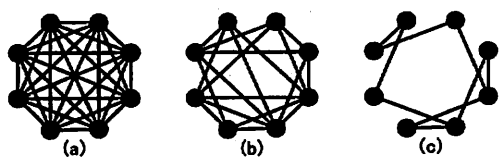

図 1 p-準完全グラフの例, (a) p=1, (b) p=0.5, (c) p=0.2

[定義 2] (p-準クリーク)

p-準クリーク $PQC = (v, e)$ はグラフ $G = (V, E)$ において, p-準完全グラフ構造をもつ連結部分グラフである. ただし, $v \in V$ であり, $e \in E$ である. p=1 のとき, 1-準クリークはクリークである.

[定義 3] (最小 p-準クリーク被覆問題)

最小 p-準クリーク被覆問題は以下に述べる制約条件下で目的関数を最小にする.

入力: 連結グラフ $G = (V, E)$, パラメータ p.
出力: p-準クリーク $PQC_1, ..., PQC_k$.
制約条件: G の各頂点は少なくとも 1 個の PQC で被覆される.
目的関数: G の各頂点の被覆に必要な PQC の数.

図 2 に最小 p-準クリーク被覆の例を示す. 図 2 (a) の G は p=0.5 のとき, 最小で 2 個の PQC により各頂点の被覆が可能である. 図 2 (b) はその 2 個の PQC である. p=1 のとき, 最小 1-準クリーク被覆問題は最小クリーク被覆問題 [2] へ帰着できる. 最小クリーク被覆問題は NP 完全であるので, それを含む最小 p-準クリーク問題は NP 完全より簡単ではない.

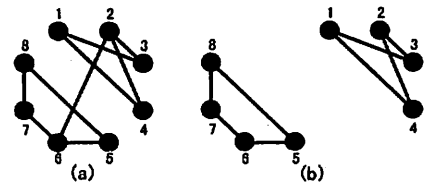

図 2 最小 p-準クリーク被覆の例 (p=0.5), (a) G, (b) 最適解

3. ハードウェアアルゴリズム

本アルゴリズムは, G の各頂点が少なくとも 1 個の PQC に被覆されるまで, 以下に述べる処理を繰り返し適用する.

(1) G の中から 1 個の PQC を探索する.

(2) G から (1) で探索した PQC の頂点を除く.

本アルゴリズムは貪欲的に頂点数の大きい 1 個の PQC を繰り返し探索し, 本問題の目的関数である被覆に必要な PQC の数の最小化を目指す. 本研究は計算時間の削減のため, 1 個の PQC の探索に対して HNN アルゴリズムを採用する [1], [6]〜[8]. 本アルゴリズムは最大クリーク問題に対する HNN アルゴリズムに基づく [7]. HNN アルゴリズムは発見的アルゴリズムの一つであり, 様々な組み合わせ最適化問題へ応用され, その有効性が示されている. また, 各ニューロンが互いに独立して並列処理が可能であるので, 文献 [1], [8] では, HNN アルゴリズムを専用ハードウェア上で高速に実行している.

以下に任意の問題に対する HNN アルゴリズムを示す. ここでは, 各ニューロンに任意の初期値を与えて解を探索する 1 回の試行を示す.

(1) $t \leftarrow 0$, 初期値を設定する.

(2) ニューロン関数を用いて出力値を更新する.

(3) 動作方程式を用いて入力値を更新する.

(4) 終了条件を満たすならば, 終了する. そうでないならば, $t \leftarrow t + 1$ とし, (2) へ戻る.

一般的に, 終了条件は最適解または局所最適解が得られたときや更新回数 t が上限回に達したときに終了する.

3.1 1 個の PQC の探索に対する HNN の定義

ニューロン関数を定義する. ニューロン関数とはニューロンの入力 U と出力 V の関係を表す関数である. 本ニューロン関

数はバイナリニューロン関数を採用する．式 (1) にニューロン i に対する本ニューロン関数を定義する．

$$V_i = \begin{cases} 1, & \text{if } U_i \geqq 0 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

V を問題空間へ対応づける．本 HNN では，N 頂点グラフに対して N 個のニューロンを用い，ニューロン i を頂点 i に対応させる．V_i は頂点 i が PQC に属するか否かを表す．もし V_i が 1 ならば，頂点 i は PQC に属し，V_i が 0 ならば属さない．

式 (2) にニューロン i に対する動作方程式を定義する．

$$\frac{dU_i}{dt} = -Af\Big(p, |v|, sum_i\Big) + Bh\Big(f\Big(p, |v|, sum_i\Big) + V_i\Big) \quad (2)$$

$$sum_i = \sum_{j=1}^{N} (1 - d_{ij}) \cdot V_j \quad (3)$$

$$d_{ij} = \begin{cases} 1, & \text{if the edge is existed} \\ & \text{between the vertex } i \text{ and the vertex } j \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

where $d_{ij} = d_{ji}$, $d_{ii} = 1$

$$f\big(p, |v|, sum_i\big) = \begin{cases} 0, \text{ if } \lceil p \cdot (|v|-1) \rceil \leq |v| - (sum_i + 1) \\ sum_i, \text{ ohterwise} \end{cases} \quad (5)$$

$$h(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

式 (2) の第一項は PQC に属さない頂点を排除するために，V が 0 となるように作用する．式 (2) の第二項は PQC の頂点数を大きくするために，V が 1 となるように作用する．関数 f は頂点 i が PQC に属するならば 0 を返し，そうでなければ sum_i を返す．関数 f の引数 $|v|$ は PQC の頂点数である．関数 h は頂点 i が PQC に属し，かつ，V_i が 0 であるならば 1 を返し，そうでないならば 0 を返す．A, B は係数である．

3.2 疑似言語を用いた本アルゴリズムの説明

本節は 3.1 の定義に基づき，最小 p-準クリーク被覆問題に対するアルゴリズムについて述べる．図 3 に本アルゴリズムのフローチャートを示す．**Step 1.** では，入出力および変数を初期化する．各パラメータはソフトウェアシミュレーションにより決定した．**Step 2.** では，G の中から頂点数が大きい 1 個の PQC を探索する．もし Covd[i] = true ならば，頂点 i は既に 1 個の PQC で被覆されているので，探索の対象から除く．**Step 3.** では，**Step 2.** で探索した 1 個の PQC を出力する．**Step 4.** では，**Step 2.** で探索した 1 個の PQC の頂点に対して Covd[i] を true とする．Covd[i] は仮想的に $G \leftarrow G - PQC$ を実現する．次の **Step 2.** では，仮想的に実現された $G - PQC$ の中から 1 個の PQC を探索する．**Step 5.** では，各頂点が少なくとも 1 個の PQC で被覆されたならば終了し，そうでないならば **Step 2.** へ戻る．

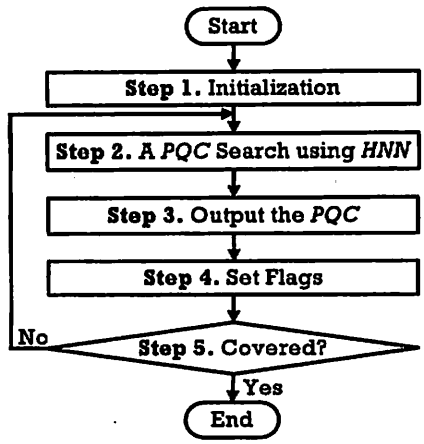

図 3 本アルゴリズムのフローチャート

入力

d_{ij} $(1 \leqq i, j \leqq \mathrm{N})$: 式 (4) に基づく $G = (V, E)$ の接続情報．ただし，$\mathrm{N} = |V|$ である．

p : 完全グラフとの近さを表すパラメータ $(0 < p \leqq 1)$.

出力

PQC_1, ..., PQC_k $(k \leqq \mathrm{N})$: p-準クリーク．

変数

A, B : 式 (2) の係数．

U_{min}, U_{max} : U_i の最小値，U_i の最大値．

t, T_{max} : U_i の更新回数，U_i の更新回数の上限．

$Covered$: G の各頂点が少なくとも 1 個の PQC で被覆されたか否かを示す Boolean 変数．

V[i] $(1 \leqq i \leqq \mathrm{N})$: V_i に対応する配列．

U[i] $(1 \leqq i \leqq \mathrm{N})$: U_i に対応する配列．

DU[i] $(1 \leqq i \leqq \mathrm{N})$: ΔU_i に対応する配列 $\left(\Delta U_i = \frac{dU_i}{dt}\right)$.

Sum[i] $(1 \leqq i \leqq \mathrm{N})$: 式 (3) の sum_i に対応する配列．

Covd[i] $(1 \leqq i \leqq \mathrm{N})$: 頂点 i が既に少なくとも 1 個の PQC で被覆されたか否かを示すフラグ．

関数

Random(x, y) : xy 間の整数乱数を返す関数．

CalcV(U[i]) : 式 (1) に基づき，V[i] を返す関数．

CalcDU(d_{ij}, p, A, B, Sum[i]) : 式 (2) に基づき，DU[i] を返す関数．

Limit(U[i]) : もし U[i] が U_{max} より大きいならば U_{max} を返し，もし U[i] が U_{min} より小さいならば U_{min} を返し，そうでないならば U[i] を返す関数．

Step 1. (初期化)

$A \leftarrow 1$, $B \leftarrow 1$, $U_{min} \leftarrow -\frac{\mathrm{N}}{2}$, $U_{max} \leftarrow \frac{\mathrm{N}}{2} - 1$, $T_{max} \leftarrow 2\mathrm{N}$, $Covered \leftarrow$ false

for $i = 1$ **to** N **do** Covd[i] $\leftarrow$ false

Step 2. (1 個の PQC の探索)

if Covd[i] = true **then the neuron** i **does not execute from Step 2.1 to Step 2.5.**

Step 2.1 (ニューロンの初期化)

$t \leftarrow 0$

for $i = 1$ **to** N **do** U[i] $\leftarrow$ Random(U_{min}, U_{max})

Step 2.2 ($V_i(t+1)$ の計算)
  for i = 1 to N do V[i] ← CalcV(U[i])
Step 2.3 (ΔU_i の計算)
  for i = 1 to N
    do for j = 1 to N
      do Sum[i] ← Sum[i] + $(1 - d_{ij})$·V[j]
  for i = 1 to N do DU[i] ← CalcDU(d_{ij}, p, A, B, Sum[i])
Step 2.4 ($U_i(t+1)$ の計算)
  for i = 1 to N do U[i] ← U[i] + DU[i]
Step 2.5 (U_i の上下限値の制限)
  for i = 1 to N do U[i] ← Limit(U[i])
Step 2.6 (更新終了判定)
  if $t < T_{max}$ then $t \leftarrow t+1$, goto Step 2.2
Step 3. (出力)
  output V[i] for all i ($1 \leqq i \leqq$ N)
Step 4. (フラグ設定)
  for i = 1 to N do if Covd[i] = false ∧ V[i] = 1
    then Covd[i] ← true, V[i] ← 0
Step 5. (終了判定)
  if Covd[i] = true for all i ($1 \leqq i \leqq$ N)
    then *Covered* ← true
  if *Covered* = false then goto Step 2.
  else then terminate this procedure

4. FPGA への実装

本研究ではインスタンス依存回路を採用し，本アルゴリズムと本問題を繰り返し解くための処理を FPGA 上に実装する．インスタンス依存回路は問題の任意のインスタンスに特化した回路であり，インスタンスに対応するレジスタの初期値をパラメータとして与える [9], [10]．一般的に，インスタンス依存回路はインスタンス非依存回路より回路面積が小さく，動作周波数が高いが，インスタンス毎に回路を合成する必要があり，回路の合成時間が回路の実行時間より大きくなる場合が多い．しかし，本研究では，同一のインスタンスに対して本問題を繰り返し解くことにより大量の局所最適解を求める必要があるので，インスタンス依存回路が有効であると考えた．

本アーキテクチャはリングネットワークアーキテクチャを採用する．リングネットワークアーキテクチャは全てのモジュールが互いに独立して並列に動作が可能であり，モジュールの結合が単純なのでモジュール間の配線およびモジュールの配置が容易である．また，複数のデバイスを用いても一つのリングネットワークを容易に構成できるので，回路の拡張性が高い．

4.1 リングネットワークアーキテクチャ

図 4 に本アーキテクチャを示す．回路の入力はなく，出力は N ビット列 $\{V_1, ..., V_N\}_k$ (k = 1, 2, ...) であり，1 個の N ビット列は 1 個の PQC に対応する．ニューロンモジュール i (NM_i) はニューロン i に対応する．NM_i (i = 1, ..., N-1) は NM_{i+1} への単方向バスで接続され，NM_N は NM_1 への単方向バスで接続される．データはリングネットワーク上の全ての NM_i を

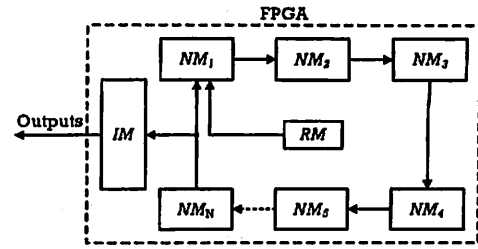

図 4 リングネットワークアーキテクチャ

中継し，リレー式に送信される．インターフェースモジュール (IM) は N ビット列を FPGA 外部へ出力する．IM は N ビット列を保持するための N ビットのバッファをもち，NM_N と接続される．乱数生成モジュール (RM) は乱数を生成し，NM_1 と接続される．

4.1.1 ΔU_i の計算

全ての NM_i は互いに独立して並列に ΔU_i を計算する．まず，NM_i は sum_i (式 (3)) を計算する．全ての NM_i は ID 番号と V_i の値 (IDV_i) をリングネットワーク上へ送信する．全ての IDV_i はリングネットワーク上の全ての NM_i を中継し，リレー式に送信される．表 1 に IDV_i のリレー式通信の過程を示す．全ての NM_i は通信処理と並列に sum_i を計算する．全ての IDV_i がリングネットワーク上を一周したならば，全ての sum_i の計算が終了したとみなし，通信を終了する．

表 1 IDV_i のリレー式通信の過程

steps	NM_1	NM_2	NM_3	...	NM_N
0	IDV_1	IDV_2	IDV_3	...	IDV_N
1	IDV_N	IDV_1	IDV_2	...	$IDV_{(N-1)}$
2	$IDV_{(N-1)}$	IDV_N	IDV_1	...	$IDV_{(N-2)}$
⋮			⋮		
N	IDV_1	IDV_2	IDV_3	...	IDV_N

4.1.2 終了判定

本回路は *Covered* を求める．*Covered* は 1 ならば，本回路は本問題を 1 回解き，そうでないならば解いていないことを表す．*Covered* は N 段の AND 回路により実現される (*Covered* ← $Covd_1$_Reg ∧ $Covd_2$_Reg ∧ ... ∧ $Covd_N$_Reg)．図 5 に N 段の AND 回路を示す．NM_i は $Covd_i$_Reg と $And_{(i-1)}$_Reg の値を AND 演算し，その結果を And_i_Reg へ格納する．ただし，And_0_Reg の値は 1 であり，And_N_Reg の値は *Covered* である．また，$Covd_i$_Reg は頂点 i が既に 1 個の PQC で被覆されたか否かを示し，And_i_Reg は G の各頂点が 1 個の PQC で被覆されたか否かを判定する．

4.1.3 再初期化

再初期化は本問題を繰り返し解くための処理である．*Covered* は NM_N の And_{out} ポートから出力され，NM_1 の Rst_{in} ポートへ入力される．図 6 に NM_1 と NM_N の接続を示す．NM_i は *Covered* を NM_1 から NM_N へリレー式に送信する．*Covered* は Rst_i_Reg へ格納され，全ての NM_i が *Covered* を受信した

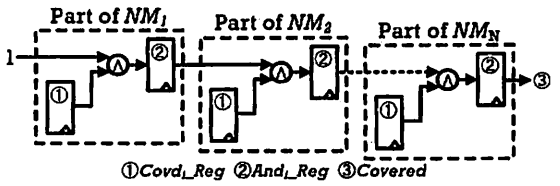

図 5 *Covered* のための N 段の AND 回路

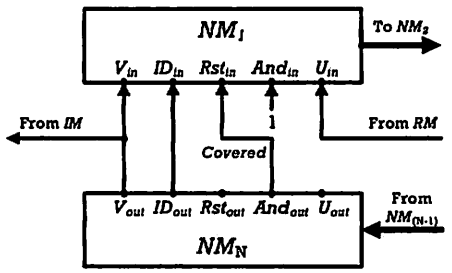

図 6 NM_1 と NM_N の接続

ならば, 通信を終了する. Rst_i_Reg は本問題を繰り返し解くために NM_i の全てのレジスタを再初期化するか否かを判定する. Rst_i_Reg の値が 1 ならば, NM_i は本問題を繰り返し解くために全てのレジスタをリセットし, そうでないならば何もしない.

4.2 ニューロンモジュール

NM_i はインスタンス N, p, A, B, U_{min}, U_{max}, T_{max}, ID 番号, 頂点 i と他の全ての頂点との接続情報 d'_i が与えられる. ただし, $d_{ij} = d'_1 \vee d'_2 \vee ... \vee d'_N$ である. 図 7 に, NM_i のデータパスを示す. V_i_Reg はニューロンの出力値 V_i に対応する. U_i_Reg はニューロンの入力値 U_i に対応する. ID_i_Reg は ID 番号を格納する. $Covd_i$_Reg は頂点 i が既に 1 個の PQC に属しているか否かを示す. And_i_Reg は G の各頂点が 1 個の PQC で被覆されたか否かを判定する. Rst_i_Reg は本問題を繰り返し解くために全てのレジスタを再初期化するか否かを判定する. また, NM_i は $\lceil \log N \rceil$ ビットカウンタおよび $\lceil \log (2N) \rceil$ ビットカウンタをもつ. $\lceil \log N \rceil$ ビットカウンタはニューロン間の通信を制御する. $\lceil \log (2N) \rceil$ ビットカウンタは U_i の更新を制御する.

5. 評　　価

5.1 得られた p-準クリークの数

ソフトウェアシミュレーションにより解の探索能力を評価する. 入力は, N={256, 512, 1024}, G の枝密度 ED={0.25, 0.5, 0.75}, p=0.5 の組み合わせである. ここで, N 頂点グラフの枝密度 ED とは N 頂点グラフの枝の数 $\#edges$ と N 頂点完全グラフの枝の数 $\#edges'$ との比である (ED=$\#edges/\#edges'$). また, p の値は文献 [3]〜[5] を参考にし, 中間的な値を設定した. 入力グラフは同一の {N, ED} の組み合わせに対して乱数を用いて 5 個の連結グラフを発生させた. 表 2 に得られた p-準クリークの数 k を示す. 各結果は各グラフに対する実行結果の平均である. 本手法により得られた k は松田らの手法により得られた k と同じかそれより小さい. 両手法において, p の値が小さくなるほど, 本問題の制約条件が緩和されるため, k は小さくなった. また, $p < ED$ のとき, 本問題の制約条件はさらに緩和されるため, G の全ての頂点は 1 個の PQC で被覆された.

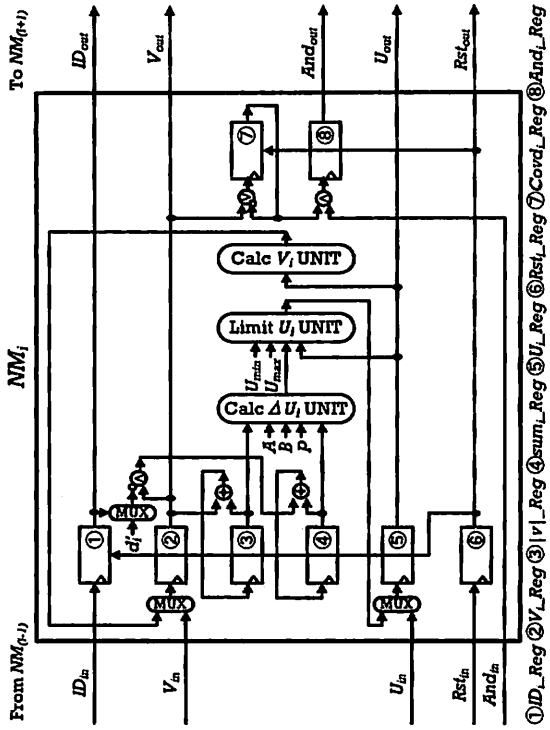

図 7 NM_i のデータパス

表 2 得られた p-準クリークの数 k (p=0.5)

N	ED	Matsuda	Ours
256	0.25	44.0	29.2
	0.5	9.4	7.0
	0.75	1.0	1.0
512	0.25	73.6	45.6
	0.5	14.0	7.8
	0.75	1.0	1.0
1024	0.25	124.8	71.2
	0.5	18.6	9.0
	0.75	1.0	1.0

5.2 実行時間

5.2.1 環　　境

本手法は 4. で述べたアーキテクチャを Verilog-HDL を用いて記述し, ALTERA 社の論理合成ツール QuartusII Version 6.0 を用いて論理合成, 配置および配線を行った. FPGA デバイスは同社の StratixII EP2S180F1508C4 を想定する. 本デバイスは総 ALUT (Adaptive Lookup Table) 数が 143500 である. 合成時間の削減のため, 最適化オプションは面積優先を, Fitter Effort は Fast Fit を選択した. 合成ツールは Xeon 3.2GHz, Memory 4GB RAM, Linux の汎用計算機上で実行した. また,

松田らの手法の実行時間は松田らのアルゴリズムをC言語を用いて実装し，上述の汎用計算機上で実行した結果である．

5.2.2 本回路の実行時間の算出方法

本回路の実行時間はシミュレーションを用いて算出する．インスタンス依存回路の実行時間は回路の生成時間と回路の実行時間の合計である．本研究では，回路の生成時間 T^{gen} は論理合成，配置および配線に要する各時間の合計とする．回路の実行時間 T_x^{exec} は本問題を繰り返し x 回解くための時間であり，実行クロック数 *clocks* とクロック動作周波数 *freq* から算出する．式(7)に本回路が本問題を1回解くための *clocks* を示す．ただし，k は被覆に必要な p-準クリークの数である．

$$clocks = k \cdot (2N^2 + 5N + 2) \tag{7}$$

式(8)に本問題を繰り返し x 回解くための本回路の実行時間 T_x を示す．ただし，*cycle* はクロックサイクルであり，*freq* の逆数である．また，解の出力のためのオーバヘッド時間は非常に小さいので無視する．本回路は同一のインスタンスに対して本問題を繰り返し解けるので，回路の合成は1回だけである．

$$T_x = T^{gen} + T_x^{exec} = T^{gen} + x \cdot (clocks \cdot cycle) \tag{8}$$

5.2.3 結　　果

表3に本回路の合成結果を示す．レジスタはALUT内部のフリップフロップにより構成した．Nが1024のとき，本回路は本デバイスのハードウェア資源の制約により本デバイス上へ実装できなかった．

表3　合成結果 (p=0.5)

N	ED	#ALUTs (%used)	*freq* [MHz]	T^{gen} [min]
256	0.25	34447 (24)	125	23
	0.5	34448 (24)	125	23
	0.75	34447 (24)	125	23
512	0.25	78964 (55)	117	60
	0.5	78549 (55)	113	62
	0.75	78712 (55)	119	63

表4に本問題を繰り返し1000回解くための実行時間を示す．本手法の実行時間 T_{1000} は式(8)，表2，表3の *freq* および T^{gen} を用いて算出した．松田らの手法の実行時間は本手法と同一の入力パターンに対して1000回実行した結果である．ED が0.5および0.75のとき，本手法は松田らの手法より2倍から16倍高速である．

遺伝子発現プロファイル解析では，遺伝子数が数千個以上の遺伝子発現プロファイルを解析する場合が多く，文献[3],[4]では，遺伝子数がそれぞれ1246個，4586個の遺伝子発現プロファイルを解析している．現段階では，本手法は実用化が困難であるが，本アーキテクチャの高い拡張性を用いて，大規模な回路を複数個のデバイス上へ実装することが有効であると考えられる．

6. ま と め

本論文では最小 p-準クリーク被覆問題に対するハードウェアアルゴリズムおよびFPGA上への実装について述べた．本アルゴリズムは最大クリーク問題に対する *HNN* アルゴリズムに基づき，松田らのアルゴリズムより解の探索能力が高いことが分かった．本アーキテクチャはリングネットワークアーキテクチャを採用した．本回路はインスタンス依存回路を採用し，遺伝子発現プロファイル解析に適用するため，同一のインスタンスに対して本問題を繰り返し解くための処理と共にFPGA上へ実装した．本手法は入力グラフの ED が大きいとき，松田らの手法より2倍から16倍高速であり，本問題を繰り返し解くことにより大量の局所最適解を求める必要がある遺伝子発現プロファイル解析に対して有効であることが分かった．今後の課題として，複数個のデバイスを用いた大規模な入力への対応などがある．

表4　本問題を繰り返し1000回解くための実行時間 (p=0.5)

N	ED	Matsuda [min]	Ours T_{1000} [min]	Ours T_{1000}^{exec} [sec]
256	0.25	9.1	24	31
	0.5	50	23	7.4
	0.75	117	23	1.1
512	0.25	50	64	210
	0.5	417	62	3.6
	0.75	1035	63	4.4

文　　献

[1] 渡辺秀一，北道淳司，黒田研一，竹中要一，"遺伝子発現プロファイル解析のためのクラスタリングアルゴリズムの提案とFPGAへの実装，" 第18回回路とシステム軽井沢ワークショップ論文集，pp. 187–192, 2005.

[2] M. R. Garey and D. S. Johnson: Computers and Intractability: A Guide to the Theory of NP-Completeness, Freemam, 1979.

[3] H. Matsuda, T. Ishihara, and A. Hashimoto, "A Clustering Method for Molecular Sequences based on Pairwise Similarity," Genome Informatics, No. 7, Universal Academy Press, pp. 23–32, 1996.

[4] H. Matsuda, T. Ishihara, and A. Hashimoto, "Classifying molecular sequences using a linkage graph with their pairwise similarities," Theoretical Computer Science, Vol. 210, No. 2, pp. 305–325, 1999.

[5] S. Seno, R. Teramoto, Y. Takenaka and H. Matsuda, "A Method for Clustering Gene Expression Data Based on Graph Structure," Genome Informatics, Vol.15, No.2, pp. 151–160, 2004.

[6] Y. Takefuji: Neural Network Parallel Computing, Kluwer Academic Publishers, 1992.

[7] N. Funabiki, Y. Takefuji, and K. C. Lee, "A Neural Network Model for Finding a Near-Maximum Clique," Journal of Parallel and Distributed Computing, Vol. 14, No. 3, pp. 340–344, 1992.

[8] 山下博司，黒川恭一，古賀義章，"相互結合型バイナリーニューラルネットワークのハードウェア化，" IEICE Trans. on Inf. & Syst., Vol. J77-D-II, No. 10, pp. 2130–2137, 1994.

[9] T. Suyama, M. Yokoo, H. Sawada, and A. Nagoya, "Solving Satisfiability Problems using Reconfigurable Computing," IEEE Trans. on VLSI Systems, Vol. 9, No. 1, pp. 109–116, 2001.

[10] S. Wakabayashi and K. Kikuchi, "An Instance-Specific Hardware Algorithm for Finding a Maximum Clique," Proc. of Int'l Conf. on Field-Programmable Logic (FPL2004), pp. 516–525, 2004.