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

2005－SLDM－118（10）
2005／1／25

グラフ最小節点被覆問題に対するFPGAを用いたインスタンス依存ハードウェア解法

菊池　健司†　　若林　真一†

† 広島市立大学情報科学部　〒731-3194 広島市安佐南区大塚東 3-4-1
E-mail: †kikuchi@lcl.ce.hiroshima-cu.ac.jp, wakaba@computer.org

あらまし　グラフの最小節点被覆問題を解くインスタンス依存のハードウェア解法を提案し，FPGA 上に実現して，その性能をソフトウェアによる解法と比較した．提案ハードウェア解法は分枝限定法に基づいており，入力グラフのインスタンスに依存したハードウェア構成の回路記述を自動的に生成し，論理合成，配置配線を行った上で回路データを FPGA 上にダウンロードして実行する．ソフトウェアによる解法と比較した結果，回路生成時間を考慮しても提案手法はソフトウェアより短い計算時間で解を得ることが分かった．

キーワード　FPGA, 最小節点被覆，インスタンス，分枝限定，論理合成

An Instance-Specific Hardware Algorithm Using FPGAs for the Mimimum Vertex Cover Problem of a Graph

Kenji KIKUCHI† and Shin'ichi WAKABAYASHI†

† Faculty of Information Sciences, Hiroshima City University
3-4-1, Ozuka-higashi, Asaminami-ku, Hiroshima, 731-3194 Japan
E-mail: †kikuchi@lcl.ce.hiroshima-cu.ac.jp, wakaba@computer.org

Abstract　This report presents a hardware algorithm for finding a minimum vertex cover of a given graph, and shows experimental evaluation of the proposed algorithm on an FPGA. The proposed algorithm is constructed according to a given instance of graph, and can find a minimum vertex cover efficiently based on branch and bound search. The proposed algorithm is supposed to be implemented with hardware, and realizes an efficient branch and bound search with parallel and pipeline processing. The proposed algorithm was implemented on an FPGA, and its running time was measured. Compared with the solving method with software, the proposed algorithm found a minimum vertex cover in a shorter running time.

Key words　FPGA, minimum vertex cover, instance, branch and bound, logic synthesis

1. まえがき

近年の半導体微細加工技術の進展に伴い，FPGA を代表とするリコンフィギャラブルデバイスの進歩は著しく，さまざまな分野で用いられるようになってきている [3]．FPGA の応用分野として近年，注目を集めているもののひとつに，通常のソフトウェアプログラムでは解くことが非常に困難な組合せ問題をリコンフィギャラブルデバイスを用いて高速に解くことがある [4]．この解法では，与えられた組合せ問題を解く専用回路を FPGA 上に構築し，ハードウェアで高速に問題を解く．特に，リコンフィギャラブルデバイスの再構成可能性を利用して，与えられた問題のインスタンスに特化した回路を構成し，問題を高速に解くことが試みられており，これまでに文脈自由言語の受理 [1]，集合被覆問題 [5]，論理式の充足可能性判定 [6] 等に対する FPGA による実現を前提とし，インスタンスに特化したハードウェア解法が提案されている．著者らもグラフの最大クリーク問題に注目し，最大クリークを求めるインスタンスに特化したハードウェア解法を提案している [2], [7].

本稿では最小節点被覆問題に対し，リコンフィギャラブルデバイスの特徴を十分に生かしたハードウェア解法を提案し，ソフトウェア解法より短い計算時間でグラフの最小節点被覆を求めることを目的としている．さらに，提案手法を実際に FPGA 上に実現し，ハードウェアコストとパフォーマンスを評価した結果を報告する．

2. 最小節点被覆問題

$G=(V,E)$ を無向グラフとし，$V=\{v_0,v_1,\ldots,v_{n-1}\}$ を G の節点集合，$E=\{e_0,e_1,\ldots,e_{m-1}\}\subseteqq V\times V$ を G の枝集合とする．G の節点集合 V の任意の空でない部分集合 V' に対し，G の V' に対する誘導部分グラフ $G(V')=(V',E')$ の枝集合 E' を，$E'\subseteqq E$，かつ，$e=(u,v)\in E$ である任意の $u,v\in V'$ に対して $e\in E'$ である，と定義する．

無向グラフ $G=(V,E)$ において，任意の枝 $e=(u,v)$ に対し，枝 e は端点 u，v により被覆されているという．G の節点集合 V の任意の空でない部分集合 V' に対し，G のすべての枝 $e=(u,v)\in E$ が V' 中のいずれかの節点により被覆されているとき，V' を G の節点被覆 (vertex cover) という．V' に真に包含され，かつ G の節点被覆である $V''\subseteqq V$ が存在しないとき，V' を G の極小節点被覆，さらに，G の節点被覆の中で節点数最小の節点被覆を G の最小節点被覆という．最小節点被覆の節点数を $\beta(G)$ で表す．

任意の無向グラフ G の最小節点被覆を一つ見つける問題を最小節点被覆問題という．最小節点被覆問題は NP-困難であることが知られている．

3. 提案手法

3.1 準　　備

提案手法は与えられたグラフ $G=(V,E)$ の最小節点被覆を1つ見つけて出力するアルゴリズムであり，分枝限定法に基づいている．

3.1.1 用　　語

提案手法の概要を説明する前に，提案手法で用いる用語を説明する．節点被覆の要素であるグラフの節点を被覆節点という．被覆節点の集合を被覆集合という．提案手法の実行中，被覆節点となりうる節点を候補節点，その集合を候補節点集合という．被覆節点でも候補節点でもない節点を除去節点という．候補節点集合の要素で，その節点を欠くと節点被覆が得られないとき，その節点を必須節点という．必須節点以外の候補節点を選択節点という．

3.1.2 データ構造

与えられたグラフ $G=(V,E)$ の節点数を n とし $(n=|V|)$，各節点を v_0，v_1，v_2，…，v_{n-1} で表し，節点の添字を節点番号とよぶ．

stack　分枝限定法の解探索の履歴 (探索途中の候補節点集合) を記憶するための2次元配列．スタックポインタ *sp* と共にスタックを実現．

sp　*stack* のスタックポインタ．現在の被覆節点数を示す．

stack_top　現在の候補節点集合を保持する1次元配列．節点 v_i が候補節点であれば $stack\_top[i]=1$，除去節点であれば $stack\_top[i]=0$．

current_VC　現在の被覆集合を保持する1次元配列．候補節点 v_i が被覆節点であれば $current\_VC[i]=1$，除去節点であれば $current\_VC[i]=0$．

essential_V　現在の必須節点を保持する1次元配列．候補節点 v_i が必須節点であれば $essential\_V[i]=1$，選択節点であれば $essential\_V[i]=0$．

essential_num　現在の必須節点数 ($essential\_V[i]=1$ である節点数)．

minimum_VC　現在の最小被覆集合を保持する1次元配列．

mvc　現在の最小被覆節点数．

leftmost　スタックトップの最左端 (すなわち，節点番号が最小) の選択節点の節点番号．

3.1.3 アルゴリズムの概要

与えられたグラフ G の最小節点被覆を分枝限定法に基づいて直接，求めることも可能ではあるが，検討の結果，そのようなアルゴリズムでは効率のよい分枝限定条件の導入が難しいことがわかった．そこで，被覆節点の候補となる候補節点集合を徐々に拡大していくことにより効率のよい分枝限定法を実現した．

アルゴリズムの基本的な考え方を以下に示す．今，$v_0,v_1,\ldots,v_S$ を被覆節点 (ただし，$1\leqq S\leqq n-2$) として与え，$V_{S+1}=\{v_{S+1},v_{S+2},\ldots,v_{n-1}\}$ を候補節点集合としてグラフ G を被覆する最小節点数の被覆集合を求めた場合の，候補節点集合 V_{S+1} から選択された被覆節点集合を $MVC(S+1)\subseteqq V(S+1)$，$MVC(S+1)$ の要素数を $mvc(S+1)=|MVC(S+1)|$ とする．このとき，$v_0,v_1,\ldots,v_{S-1}$ を被覆節点として与え，v_S を除去節点とし，V_{S+1} を候補節点集合としてグラフ G を被覆する最小節点数の被覆節点集合 $MVC(S)\subseteqq V(S)$ を求める．最小被覆節点数 $mvc(S)$ の下界が $mvc(S+1)$ であることは明らかである．探索の結果，もし，$mvc(S+1)$ 個の節点からなる被覆節点集合が求まった場合はその集合が候補節点集合を V_S とした場合の $MVC(S)$ となる．また，$mvc(S+1)$ 個の節点からなる被覆節点集合は存在しないと判明した場合は $MVC(S)=MVC(S+1)\cup\{v_S\}$ とすれば，$MVC(S)$ は G を被覆し，候補節点集合を V_S とした場合の節点数最小の被覆節点集合となる．このとき，被覆節点集合の節点数は $mvc(S)=mvc(S+1)+1$ である．この手順を S を1ずつ減らしながら繰り返していけば，$S=0$ のときの被覆節点集合 $MVC(0)$ が G の最小節点被覆，$mvc(0)$ が最小節点被覆の節点数となる．

アルゴリズムにおいて，V_{S+1} を候補節点集合とし，節点 v_S を除去節点，$v_0,v_1,\ldots,v_{S-1}$ を被覆節点として与えた場合の，$mvc(S+1)$ 個の節点からなる被覆節点集合を求める探索に分枝限定法に基づく探索を用いる．節点被覆の定義と上記の探索手順に基づくいくつかの分枝限定条件を導入し，無駄な探索を省くように工夫している．

3.2 アルゴリズム

Step0　(初期化)

スタックポインタと最小被覆節点数の初期化: $sp=0$，$mvc=1$．

S を自身よりも節点番号の大きい節点と枝を持つ最右端 (すなわち，節点番号が最大) の節点の節点番号とする．節点被覆の定義より部分グラフ $G(V_{S+1})$ の最小被覆節点数は0，$G(V_S)$ の最小被覆節点数は1となる．

必須節点は存在しない: $essential\_V[i] = 0$ $(0 \leq i \leq n-1)$, $essential\_num = 0$.

節点 v_i $(0 \leq i \leq S-1)$ を被覆節点, v_j $(S \leq j \leq n-1)$ を候補節点とする: $stack\_top[i] = 0$ $(0 \leq i \leq S-1)$, $stack\_top[i] = 1$ $(S \leq i \leq n-1)$, $current\_VC[i] = 1$ $(0 \leq i \leq S-1)$, $current\_VC[i] = 0$ $(S \leq i \leq n-1)$.

Step1 (最左端の選択節点を被覆節点集合に加える)

スタックトップの最左端の選択節点 $v_{leftmost}$ を被覆集合に加える: $stack\_top[leftmost] = 0$, $current\_VC[leftmost] = 1$.

Step2 (スタックのプッシュ)

$stack[sp][*]$ に現在の $stack\_top[*]$ の値を記憶させた後, $sp = sp + 1$ とする.

Step3 (節点の除去)

$if(sp + essential\_num > mvc)$ mvc 個の節点で被覆できないことが明らかなので Step6(バックトラック) へ行く.

$else$ 選択節点の中で, 選択節点間を結ぶ枝の次数が 0 である節点 v_i は, 現在 $current\_VC$ に保持されている被覆節点, あるいは $essential\_V$ に保持されている必須節点により v_i に接続するすべての枝が被覆可能であることを意味するので, そのような節点 v_i を候補節点集合から取り除く $(stack\_top[i] = 0$ とする).

$if(sp + essential\_num == mvc)$ 被覆節点集合と必須節点集合の節点で全ての枝が被覆可能かどうかを判定するために Step4(最小被覆の判定) に行く.

$else$ これ以降の探索で mvc 個の節点で被覆可能な集合が見つかるかどうかを判定するために Step5(分枝限定) に行く.

Step4 (最小被覆の判定)

現在の被覆集合と必須節点集合の節点で全ての枝が被覆可能かどうかを判定する. 判定方法は, $leftmost == n$ かどうか, すなわち選択節点がスタックトップに存在するかどうかにより判定する.

$leftmost == n$ ならば最小被覆の発見されたことになり, Step8(次範囲の探索準備) へ行く. $leftmost! = n$ ならば mvc 個の節点で被覆できないので Step6(バックトラック) に行く.

Step5 (分枝限定)

分枝限定条件を調べ, 条件が成立すれば Step6(バックトラック) に行く. そうでなければ, もし分枝限定 4 により必須節点の数が増えれば, もう一度, 最小被覆の判定や分枝限定条件を調べるために Step3 に戻る. 必須節点の数が変わらなければ, Step1 に戻り探索を続ける.

Step6 (バックトラック)

$sp == 0$ ならば部分グラフ $G(V_S)$ に対する探索は終了し (mvc 個の節点では被覆はできない), Step8(次範囲の探索準備) へ行く.

$sp! = 0$ の場合は $current\_VC$ の最右端の被覆節点 $v_{rightmost}$ を被覆集合から除く $(current\_VC[rightmost] = 0$ とする). $sp = sp - 1$ とし, スタックをポップする $(stack\_top[i] = stack[sp][i]$ $(0 \leq i \leq n-1))$.

Step7 (必須節点の計算)

現在の被覆集合 $current\_VC$ とスタックトップ $stack\_top$ を用いて, 除去節点 v_i $(stack\_top[i] = 0$ && $current\_VC[i] = 0)$ に隣接する候補節点を必須節点とする $(essential\_V[j] = 1$ とする). そして必須節点数を計算して $essential\_num$ を更新する(注1). Step3 に戻る.

Step8 (次範囲の探索準備)

Step8 への遷移直前の状態が Step4 ならば mvc の値は変化せず, 直前の状態が Step6 ならば $mvc = mvc + 1$ とする. また, 直前の状態が Step4 ならば $minimum\_VC = current\_VC|essential\_V$ とする. 直前の状態が Step6 の時には $minimum\_VC[S] = 1$ とする.

節点 v_S の最小被覆節点数の下界を $LB[S] = mvc$ とする. 探索範囲 S が 0 であれば全範囲の探索が終了したのでアルゴリズムを終了する. それ以外の時は $S = S - 1$ とし, $stack\_top[i] = 0$ $(0 \leq i \leq S)$, $stack\_top[j] = 1$ $(S < j \leq n-1)$, $current\_VC[i] = 1$ $(0 \leq i < S)$, $current\_VC[j] = 0$ $(S \leq j \leq n-1)$ とする (これにより節点 v_S を除去節点としている). また節点 v_S に隣接する節点を必須節点とする $(essential\_V[i] = 1)$. さらに必須節点数を $essential\_num$ に格納する. Step1 に行く.

3.3 分枝限定

節点被覆を求める提案アルゴリズムは分枝限定法に基づいており, 効率のよい解探索を実現し, 無駄な探索を回避するために 5 つの分枝限定条件に基づく枝刈りを実現している.

3.3.1 分枝限定条件 1: 被覆節点数の下界を用いた分枝限定

$$sp + LB[leftmost] > mvc \tag{1}$$

解探索の途中において, その時点で候補節点集合から選択された被覆節点数を mvc, 候補節点集合 $V_i = \{v_i, \ldots, v_{n-1}\}$ に対する最小被覆節点数を $LB[i]$ とする. この時, $LB[leftmost]$ の値は, これ以降の探索で全節点を被覆するのに最低限必要な節点数であり, この時点までに被覆集合 $current\_VC$ に加えた節点数 (スタックポインタ sp で表される) との和が mvc よりも大きければ, mvc 個の節点での被覆は不可能と判断され, 探索を中断し, バックトラックする.

LB	87	88	89	90	91	92	93	94	95	96	97	98	99
	9	8	7	7	6	5	4	4	3	2	1	0	0

図 1 被覆節点数の下界を用いた分枝限定

(注1): 実験で評価した回路では Step3 以降に並列で事前に必須節点とその総数を調べるようにしている.

次に，この分枝条件が成り立つ場合を例を用いて説明する．節点数 100 ($n = 100$) のグラフにおいて，候補節点集合 $V_{86} = \{v_{86}, \ldots, v_{99}\}$ に対して被覆節点集合を探索し，$mvc = 9$ とする．それぞれの節点に対する最小被覆節点数の下界は図 1 に示すようになっていたとする．ここで上記の分枝限定条件が成り立つためには，最後に被覆集合に加えた節点の節点番号を $rightmost$ とすると，条件 $LB[rightmost] == LB[leftmost]$ が成り立たなければならない．なぜならば，もし，$LB[rightmost]!= LB[leftmost]$ であるならば，$LB[rightmost] - LB[leftmost] \leqq -1$ となる．また sp (被覆節点数) の値は $+1$ されるので，上記の分枝限定条件が満たされることはない．図 1 に示す例において $leftmost = 92$ の時，節点 v_{92} を被覆集合に加えて ($rightmost = 92$)，$leftmost$ が 93 となったとする．この時，sp の値は $+1$ され，また LB の値は -1 されるので mvc の値を超えることはない．

また式 (1) を変形すると，上記の条件 ($LB[rightmost] == LB[leftmost]$) に加えて，被覆集合節点数は ($mvc - LB[leftmost]$) 個存在しなければならないことがわかる．例では，式 (1) が成り立つには $leftmost = 90$ の場合，被覆節点は 3 個必要となる．つまり $current\_VC = \{v_{87}, v_{88}, v_{89}\}$ である時のみこの分枝限定条件は成り立つ．これは $v_{i+1}, \ldots, v_{leftmost-1}$ の全て節点が被覆集合に加えられていることを意味する．$leftmost = 94$ の場合には被覆節点は 6 個必要となる．

なお，この分枝限定の問題点は必要なハードウェア量が多くなる点である．上記の例では，節点 v_0，v_1 などに対しては LB を保持するのに 7 ビット必要となる．そこで実際の回路構成では下記の分枝限定条件を採用する．

(分枝限定条件) $LB[i] == LB[i-1]$ である時，$LB1[i] = 1$，そうでない時，$LB1[i] = 0$ とする．この時，次の式が成り立てばバックトラックする．ただし，S は現在の候補節点集合 V_S の添字の値を示す．

$$(sp + S + 1 == leftmost)\&\&(LB1[leftmost] == 1) \quad (2)$$

($sp + S + 1 == leftmost$) の式は $v_{S+1}, \ldots, v_{leftmost-1}$ の節点全てが被覆集合に加えられていることを意味している．そして ($LB1[leftmost] == 1$) は $LB[leftmost - 1] == LB[leftmost]$ であることを表している．例を図 2 に示す．

LB	87	88	89	90	91	92	93	94	95	96	97	98	99
	9	8	7	7	6	5	4	4	3	2	1	0	0

LB1	87	88	89	90	91	92	93	94	95	96	97	98	99
		0	0	1	0	0	0	1	0	0	0	0	0

図 2　被覆節点数の下界を用いた分枝限定 (2)

この分枝限定条件は $LB[leftmost - 1] == LB[leftmost]$ となる最左端の節点にしか適用できない．$leftmost = 94$ の場合，式 (1) による分枝限定条件は成立するが，式 (2) は成立しない．他にも式 (1) ならば成り立つにも関わらず，式 (2) は成り立たない場合もある．しかしながら，実験の結果，そのような場合は少ないことがわかっている．

3.3.2 分枝限定条件 2: 枝次数を用いた分枝限定

候補節点数を sp，必須節点数を $essential\_num$，現在探索している最小被覆節点数を mvc とすると現在の選択節点の中から m 個 ($m = mvc - (sp + essential\_num)$) の節点を選択して選択節点間に存在する枝を被覆しなければならない．この時，選択節点 v_i に隣接する選択節点数を $deg(v_i)$ とすると，$deg(v_i) > m$ である節点が $(m+1)$ 個以上存在する時，m 個の節点での節点被覆は不可能である．

3.3.3 分枝限定条件 3: 完全グラフの性質を利用した分枝限定

(前処理) グラフ G の節点集合 V を 2 節点から 8 節点までの G の完全部分グラフ β_i に分割する．

$$V = \beta_1 + \beta_2 + \beta_3 + \cdots + \beta_p + V_z \quad (3)$$

$\beta_1, \beta_2, \ldots, \beta_p$ は互いに節点独立 (節点を共有しない) であり，V_z はいずれの β_i にも選ばれなかった残りの節点である．この前処理は最適解を求めるのは困難であるので発見的手法で解を求める．

(分枝限定条件) k 節点の完全グラフの最小被覆節点数は $k-1$ である．よってある完全部分グラフ β_i の節点集合に含まれる選択節点数を $nv(\beta_i)$ とすると，少なくとも $(nv(\beta_i) - 1)$ 個 (ただし $nv(\beta_i) = 0$ の時は 0 個) の節点が被覆集合の要素として必要となる．よって以下の条件が成り立つ時，mvc 個の節点での節点被覆は不可能である．

$$sp + essential\_num + sum(\beta_1) + sum(\beta_2) + \cdots + sum(\beta_p) > mvc \quad (4)$$

ただし $sum(\beta_i)(1 \leq i \leq p)$ は $max\{nv(\beta_i) - 1, 0\}$ である．

3.3.4 分枝限定条件 4: 枝次数を用いた分枝限定 2

選択節点に対する枝次数が $deg(v_i) \geqq 2$ である選択節点 v_i において，この節点が $deg(v_j) == 1$ となる選択節点 v_j と隣接しているのならば，枝 (v_i, v_j) を被覆するためには節点 v_j を選択する必要はなく，常に節点 v_i を選択すればよい．よって節点 v_i を必須節点とする．また $deg(v_i) > m$ である節点については，もしこの節点を除去節点とすると隣接する $(m+1)$ 個以上の節点を被覆集合に加えなければならない．よって $deg(v_i) > m$ である節点を必須節点とする．

3.3.5 分枝限定条件 5: 最大枝次数を用いた分枝限定

選択節点の中で枝次数の最大値 ($dmax$ とする) を求める．この時，以下の式が成り立つ時，バックトラックする．以下の式はどの節点を選択しても最大枝次数分の枝が被覆できると仮定して，現在必要な節点数を被覆集合に加えても，全ての枝を被覆できないことを表している．

$$dmax \times (mvc - (sp + essential\_num)) < e_{total} \quad (5)$$

ここで e_{total} は選択節点間を接続する枝の総数であり，選択節点の枝次数の合計÷2で求めている．ただしこの条件は分枝限定条件としては緩い．そこで上記の条件が成立しなければ，最大枝次数と同じ枝次数を持つ選択節点数 x と次に大きな枝次数 ($dmax2$ とする) を求めて，以下の分枝限定条件を調べる．

$$dmax \times x + dmax2 \times ((mvc - (sp + essential\_num)) - x) < e_{total} \quad (6)$$

ただし $x < (mvc - (sp + essential\_num))$ の時のみこの条件を調べる．

3.4 インスタンス依存の回路生成

提案手法はFPGA上に実現して実行することを前提としている．一般に，組合せ問題の解法をFPGA上に実現して問題を解く方法には2通りの方法がある．ひとつは通常のソフトウェアによる実現と同様，解法をFPGAに実現し，問題のインスタンスは実行時に回路に入力して，問題の解を得る方法である．これに対し，問題の解法とインスタンスが与えられ，FPGA上には問題のインスタンスに特化した解法を実現して，問題の解を得る方法がある．本稿では前者の手法によりFPGA上に実現される回路をインスタンス非依存回路，後者の手法によりFPGA上に実現される回路をインスタンス依存回路とよぶ．

一般にインスタンス非依存回路はインスタンス依存回路と比較して，回路規模が大きくなり，動作周波数は低下する．また，ハードウェアとしての実現が非常に難しい場合もある．例えば，提案手法の完全グラフの性質を利用した分枝限定において，任意のグラフに対してこの分枝限定を適用できるようにするためには非常に複雑で規模の大きい回路を必要とし，実際には実現は困難である．しかしながら，提案手法をインスタンス依存回路で実現すれば，与えられたグラフに特化した回路を実現すればよいので，回路の実現が非常に容易になる．このため，本研究ではインスタンス依存回路による提案手法の実現を採用した．

提案手法をFPGAに実現する場合，Verilog-HDLによるハードウェア記述をFPGA設計ツールで合成して回路のネットリストを生成している．その際，以下の手順でVerilog-HDLによるハードウェア記述を得る．まず，提案手法において，問題のインスタンスに依存しない部分のハードウェア記述 (テンプレートとよぶ) を用意しておく．次に与えられた問題のインスタンスに対応した回路記述をテンプレートを修正することで得る．その後，FPGA設計ツールで回路合成を行い，FPGA上に回路を実現して提案手法を実行し，解を得る．

インスタンス非依存回路による解法の実行時間は回路の実行時間のみであるが，インスタンス依存回路による解法の実行時間は回路の実行時間に加えて，回路のハードウェア記述の生成時間，およびネットリスト生成のための回路合成の実行時間を含めることが必要である．

4. 実験と評価

4.1 実験環境

提案手法の回路合成を行い，ソフトウェア解法と比較する評価を行った．実験環境は以下の通りである．ハードウェア記述にはVerilog-HDLを使用した．インスタンスに対応したハードウェア記述を生成するプログラムはPerl言語を用いて作成し，CPUがUltraSPARC IIIi 1062MHz，主記憶が2GBのワークステーションで実行した．本稿で用いたFPGA設計ツールはAltera社 QuartusII Version 4.1である．このツールはCPUがPentium 4 2.4GHz，主記憶が2GBのPC上で実行した．また，回路合成時に指定したFPGAは，Altera社EP1S60F1020C7 (論理エレメント数:57120) である．提案手法の実行時間は，生成した回路をFPGAボード (三菱電機マイコン機器ソフトウェア株式会社: Power Medusa MU200-SX60) に実現し，計測した．250節点以上のグラフに関しては，FPGA上での実現が困難であったために実験を行っていない．比較対象のソフトウェアとして，提案手法と同等のアルゴリズムを用いたC言語のプログラムを作成した．ソフトウェア解法はv200p005, v200p007, v200p010, v200p012のグラフでは，CPUがPentium 4 4.3GHz，主記憶が3GBのPC上で実行した．残りのグラフでは，CPUがUltraSPARC IIIi 1062MHz，主記憶が2GBのワークステーションで実行した．問題のインスタンスとしては，ランダムグラフを生成するプログラムをC言語により作成し，そのプログラムを実行することにより得たグラフを用いた．

4.2 実験結果

実行結果を表1に示す．表1において n は節点数，ρ は枝密度，β は最小被覆の節点数である．また，software，提案手法はソフトウェア解法と提案手法の実行時間 (単位は秒) である．回路生成時間は回路の生成に要した時間 (単位は秒) であり，HDL生成時間はインスタンスに対応したHDL記述の生成時間である (単位は秒)．LE数は使用した論理エレメント数とその使用率である．

表1より本稿で提案したアルゴリズムは，同じ節点数のグラフで比較すると，枝密度10%付近で実行時間が最大となり，枝密度の増減と共に実行時間が減少していることがわかる．本稿では節点番号の逆順による探索アルゴリズムを採用し，以前に探索した範囲の最小被覆節点数で，現在の探索範囲での被覆が可能かどうかの判定を行っている．なお一般的にはグラフの枝数が増加するにつれて，全枝の被覆のためにより多くの節点を必要とする．よって，枝密度の増加に伴い以前の探索範囲の最小被覆節点数で被覆できる可能性は少なくなる．つまり，探索の早い段階から分枝限定条件が成り立ちやすくなり，分枝数が減少したと考えられる．そして枝密度が減少した場合では，逆に被覆に必要な節点数は少なくなり，節点の組合せの数は限られることが実行時間減少の要因と考えられる．

次にソフトウェア解法と提案手法の比較を行う．解を導き出す実行時間のみで提案手法とソフトウェア解法を比較すると，全てのグラフにおいて提案手法の方が短時間で解を求めてい

表 1　ランダムグラフデータに対する実験結果

グラフ名	n	ρ	β	*software*[秒]	提案手法 [秒]	回路生成時間	HDL 生成時間	LE 数
v100p007	100	0.07	63	1.0	0.005	2201	1.0	12869(22%)
v100p010	100	0.10	71	14.0	0.05	2198	1.0	13928(24%)
v100p012	100	0.12	72	10.0	0.02	2941	1.0	15447(27%)
v100p015	100	0.15	77	18.0	0.06	3346	1.0	16521(28%)
v100p025	100	0.25	83	5.0	0.01	4946	1.0	18560(32%)
v150p005	150	0.05	98	647.0	1.81	6244	1.0	24439(42%)
v150p007	150	0.07	105	2190.0	5.69	6867	1.0	27353(47%)
v150p010	150	0.10	113	2856.0	6.19	7148	1.0	29699(51%)
v150p012	150	0.12	118	3973.0	40	7857	2.0	30839(53%)
v150p015	150	0.15	123	3582.0	6.77	8160	2.0	33165(85%)
v200p005	200	0.05	141	376625.0	2246	13062	2.0	40965(71%)
v200p007	200	0.07	149	465834.0	2360	16525	2.0	46019(80%)
v200p010	200	0.10	159	226244.0	1044	19881	3.0	49592(86%)
v200p012	200	0.12	164	182543.0	826	24519	3.0	51118(89%)
v200p015	200	0.15	169	164876.0	206	27196	4.0	53667(93%)

表 2　各分枝限定条件を取り除いた場合のソフトウェア解法の実行時間 [秒]

グラフ名	オリジナル	条件 1 無し	条件 2 無し	条件 3 無し	条件 4 無し	条件 5 無し
v150p007	2190	2896	2197	2264	45667	4270
v150p010	2856	3177	2867	2898	10335	7110

ることがわかる．しかし提案手法はインスタンス依存ハードウェア解法によるものであるので，回路生成時間も実行時間に含めなければならない．そこで回路生成時間や HDL 生成時間を考慮して比較する．枝密度が 10%のグラフに着目すると，v100p010 のグラフでは，提案手法の実行時間はソフトウェア解法の約 160 倍もの時間が必要となってしまう．しかし v150p010 のグラフでは，実行時間の差は約 2.5 倍まで短縮されている．さらに v200p010 のグラフでは，提案手法の方が短時間で解を求めており，ソフトウェア解法の約 0.09 倍の実行時間となっている．すなわち，グラフの節点数が増大すると回路生成時間を考慮しても提案手法が有利であると言える．また，枝密度が小さい程回路生成時間は短くなっている．節点被覆問題では枝密度が小さい方が解を求めるのに時間がかかる点からも提案手法の有効性が言える．

しかし，グラフが大規模になると回路規模の問題が生じる．v200p015 のグラフでは使用エレメント率が 93%となっている．そして，300 節点以上のグラフは論理エレメント数の制約から今回使用した FPGA 上には実装できなかった．さらに回路規模が増大することにより，回路生成時間も多大となってしまう．よって回路規模の削減が今後の課題として挙げられる．

表 2 は v150p007 および v150p010 のグラフに対し，提案したそれぞれの分枝限定条件を外してソフトウェアで実行した時の実行時間である．表 2 より分枝限定条件 4 が最も効率よく分枝数を減らしていることがわかる．つまり，被覆に必要となる節点は分枝の前にあらかじめ被覆集合に加えておき，不必要な探索や分枝限定条件判定を行わないことが節点被覆問題では重要となる．

5. あとがき

本稿ではグラフの最小節点被覆を求めるハードウェアアルゴリズムを提案し，その有効性を実験的に示した．提案手法では問題のインスタンスに依存して回路が生成され，分枝限定法に基づく解探索により効率よく解を求めることができる．今後の課題としては，提案アルゴリズムをさらに改良し，より高速に最小節点被覆を求めることを可能にすること，および他の組合せ問題に対してもインスタンスに依存したハードウェアアルゴリズムにより解探索を行う手法を開発することなどがある．

謝　辞

本研究の一部は平成 16 年度科学研究費補助金 (基盤研究 (C)) 課題番号 15500040 による．

文　献

[1] J.L.Bordim, Y.Ito, K.Nakano, Instance-specific solutions for the CKY parsing, 第 1 回リコンフィギャラブルシステム研究会論文集, pp.27–33, 2003.

[2] 藤原知幸, 若林真一, 最大クリーク問題を解くインスタンスベースのハードウェア解法と FPGA による実現, 信学技報, VLD2002-138, 2003.

[3] 松本仁, FPGA の進化と今後の FPGA 設計に求められるもの—ロジックデバイスから SoPD へ—, 信学技報, VLD2002-127, 2003.

[4] M.Platzner, Reconfigurable accelerators for combinatorial problems, Computer, 33, 4, pp.58–60, 2000.

[5] C.Plessl, M.Platzner, Instance-specific accelerators for minimum covering, The Journal of Supercomputing, 26, pp.109–129, 2003.

[6] T.Suyama, M.Yokoo, H.Sawada, A.Nagoya, Solving satisfiability problems using reconfigurable computing, IEEE Trans. on VLSI Systems, 9, 1, pp.109–116, 2001.

[7] 若林真一, 菊池健司, 最大クリークを求めるデータ依存ハードウェアアルゴリズムの実装と評価, 情報処理学会システム LSI 設計技術研究会研究報告, 2004-SLDM-113-22, 2004.