LC-002

最小支配集合問題を解くインスタンス依存ハードウェア解法

An Instance-Specific Hardware Algorithm for Solving the Minimum Dominating Set Problem

有路 忠臣* 菊池 健司† 若林 真一*

Tadaomi Ariji Kenji Kikuchi Shin'ichi Wakabayashi

1 はじめに

FPGA を代表とするリコンフィギャラブルデバイスはさまざまな分野で用いられるようになってきている [6]. FPGA の興味深い応用分野の 1 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 [3]. この方法では FPGA の再構成可能性を利用して，与えられた組合せ問題のインスタンスに特化した専用回路を FPGA 上に構築し，ハードウェアで高速に最適解を得る．これまでに充足可能性判定問題 [4] などの問題に対するハードウェア解法が提案されており，著者らもグラフの最大クリーク問題 [5]，最小節点被覆問題 [2] に対してハードウェア解法を発表している．

本稿ではグラフの最小支配集合問題に対し，インスタンスに特化したハードウェア解法を提案する．最小支配集合問題はグラフに関する基本的な組合せ問題の 1 つである．この問題は多くの応用が知られているが，NP 困難であり，グラフが複雑かつ大規模になると最適解を効率よく求めることが難しいことが知られている [1]. 提案解法はリコンフィギャラブルデバイスとしての FPGA の特徴を利用して，ソフトウェア解法より短い計算時間でグラフの最小支配集合を求める．さらに，提案解法を FPGA ボード上に実現し，ソフトウェア解法との性能の比較評価を行った．

以下では，まず 2. において最小支配集合を定義する．次に 3. において，最小支配集合問題に対し，分枝限定法に基づくインスタンスに特化したハードウェア解法を提案する．4. において提案手法に対する計算機実験による評価を行い，最後に 5. で本稿をまとめる．

2 最小支配集合問題

節点集合 V, 枝集合 E からなるグラフ $G=(V,E)$ における部分節点集合 D に対し，すべての節点 $u \in (V-D)$ において，他方の端点が $v \in D$ である枝 $(u,v) \in E$ が存在するとき，その D を支配集合という．与えられた任意の無向グラフの最小節点数の支配集合を 1 つ見つける問題を最小支配集合問題という．グラフ G の最小支配集合の節点数を G の支配数という．最小支配集合問題は NP 困難であり，最小支配集合を求める多項式時間アルゴリズムは存在しないと予想されている [1].

* 広島市立大学情報科学部

† 広島市立大学大学院情報科学研究科

3 提案手法

3.1 提案手法の概要

提案手法は分枝限定法 [7] に基づいている．以下に提案手法の概要を示す．グラフの節点集合を V，節点数を n $(n=|V|)$ とする．

まず，提案手法の主なデータ構造について説明する．$stack[sp,i]$ は 2 次元配列であり，スタックポインタ sp と一緒にスタックを実現し，支配節点の候補となる節点（候補節点という）を記憶するために使用される．すなわち，節点 v_i が候補節点ならば $stack[sp,i]=1$，候補節点でなければ $stack[sp,i]=0$ とする．候補節点でない節点を除去節点という．初期値ではすべての節点を候補節点とし，$sp=0$，$stack[0,i]=1$ $(1 \le i \le n)$ である．1 次元配列 $ds[i]$ $(1 \le i \le n)$ は支配節点として選択された節点を記憶し，節点 v_i が支配節点に選択されていれば $ds[i]=1$，そうでなければ $ds[i]=0$ とする．また，分枝限定法の各ステップで選択された支配節点を記憶する 1 次元配列 $ds\_stack[i]$ を用意する．ds_stack は $stack$ と共通のスタックポインタ sp と共にスタックを実現する．1 次元配列 $min\_ds[i]$ $(1 \le i \le n)$ は最小支配集合を記憶し，節点 v_i が最小支配集合の要素であれば $min\_ds[i]=1$，そうでなければ $min\_ds[i]=0$ とする．最小支配集合のサイズ（すなわち支配数）を変数 num_ds で記憶する．提案手法においては，高速な発見的手法により節点数のできるだけ小さい支配集合を求めて，そのサイズを num_ds の初期値としている．

分枝限定法の実行途中において，まだ支配されていない節点集合を F とする．また，支配節点の候補となる節点集合 U の要素 v に対し，v の有効節点次数を $d_e(v)=|\{u|(v,u) \in E, u \in F\}|$ とする．すなわち，v に隣接する F の節点数を v の有効節点次数とする．さらに，候補節点 v の有効支配数 $nd(v)$ を $nd(v)=d_e(v)+dd(v)$ と定義する．ただし，$dd(v)$ は節点 v がまだ支配されていない場合は 1，すでに他の節点に支配されている場合は 0 とする．

分枝限定法の実行途中において，支配節点として選択する必要のなくなった候補節点を除去可能節点という．例えば，有効支配数 $nd(v)$ が 0 の候補節点 v は除去可能節点である．このような候補節点は候補節点集合から除去する．さらに，分枝限定法の実行途中において，必ず支配集合に含める必要がある候補節点 v を必須節点という．必須節点が見つかった場合は配列 $es[i]$ に記憶し，その節点については支配節点の選択対象から除外する．例えば，有効節点次数が 1 の除去節点に隣接している候補

節点は必須節点となる．除去可能節点，および必須節点の条件は上記以外にもいくつかあるが，紙面の都合上，ここでは省略する．

以下に提案手法の概要を示す．

Step0（初期設定）：
スタックと配列の初期化．$sp \leftarrow 0$.
Step1（必須節点，除去可能節点）：
スタックトップの候補節点集合から必須節点を選択し，配列 es に記憶する．さらに除去可能節点を選択し，スタックトップから除去する．支配節点とスタックトップの候補節点集合が許容解を含まない場合は Step7 へ．
Step2（支配節点の選択）：
スタックトップの候補節点集合の中から，有効支配数が最大で，かつ必須節点ではない候補節点（v_j とする）を選択し，候補節点集合から除去する（$stack[sp, j] \leftarrow 0$）．
Step3（探索）：
現在の支配集合に Step2 で選択した候補節点 v_j を加える（$ds[j] \leftarrow 1$）．さらに，v_j を ds_stack のスタックトップに記憶する．
Step4（解の判定）：
すべての節点が支配節点か必須節点，あるいは被支配節点となれば Step8 へ．
Step5（分枝限定）：
分枝限定条件 1 ～ 3（後述）が成立するかどうかを調べる．成立すれば Step2 で選択された支配節点 v_j を支配集合から除去して Step1 へ．そうでなければ Step6 へ．
Step6（分枝）：
スタック $stack$ と ds_stack を push し，$stack$ のスタックトップを現在の候補節点集合に更新する．Step1 へ．
Step7（バックトラック）：
$sp > 0$ であればスタック $stack$ と ds_stack を pop し，ds_stack のスタックトップの支配節点を支配集合から除去し，Step1 へ．そうでなければ探索を終了する．
Step8（解の更新）：
ds と es に記憶されている必須節点を含む支配節点数が num_ds より小さい場合は ds と es の支配節点を min_ds にコピーし，num_ds を更新する．Step2 で選択された支配節点 v_j を支配集合から除去し，Step7 へ．

3.2 分枝限定条件

提案手法の Step5 の分枝限定条件 1 ～ 3 を以下に示す．

3.2.1 分枝限定条件 1

グラフ $G = (V, E)$ において，2 つの節点 u，v に対し，u，v 間に枝がなく，かつ u，v の双方に枝で接続している節点 w がない場合，u と v は 2 節点独立という．節点部分集合 $I \subseteq V$ について，I のすべての節点対が 2 節点独立であるとき，I を 2 節点独立集合という．G の 2 節点独立集合で，極大となるものを極大 2 節点独立集合という．v を含む G の極大 2 節点独立集合を $I(v)$ で表す．このとき，$I(v)$ の節点数 $|I(v)|$ は，G の最小支配集合の節点数の下界となる．すべての $v \in V$ に対して $I(v)$ を求め．その中で，同一集合となるものは 1 つだけにした結果を $I_0, I_1, I_2, ..., I_m$ とする．これらは事前に計算しておく．

最小支配集合を求める分枝限定法の実行途中で，すでに選ばれた支配集合の節点数を k，まだ支配されていない節点集合を F とするとき，$\max_j\{k + |F \cap I_j|\}$ を計算する．この値が現在の支配集合の節点数と等しいか大きい場合は，これ以上，探索しても最小支配集合は更新されないので，ここで枝刈りを行う．

3.2.2 分枝限定条件 2

提案手法の Step5 において，ある非必須節点 u を支配集合 ds の要素に選択した場合に減少する未支配節点数は，定義より u の有効支配数 $nd(u)$ である．このことから次の分枝限定条件を導くことができる．

未支配節点の集合を F とする．候補節点集合 C の非必須節点（m 個あるとする）を有効支配数の降順に並べた節点のリストを $u_1, u_2, \ldots, u_m$ とする．節点のリストの最初の $min(m, num\_ds - nds - nes - 1)$ 個の節点について以下の式を計算する．ただし，nds，nes はそれぞれその時点での支配節点数，および必須節点数である．

$$ub(C) = \sum_{i \leftarrow 1}^{min(m, num\_ds - nds - nes - 1)} nd(u_i)$$

このとき，$|F| > ub(C)$ であれば枝刈りする．

3.2.3 分枝限定条件 3

分枝限定法の実行途中において，支配集合の要素でないと決定した節点（すなわち除去節点）の集合を Q とする．ある節点 v とその隣接節点集合 $A(v)$ に対し，$(\{v\} \cup A(v)) \subseteq Q$ であれば，その Q に対しては支配集合は存在しないので枝刈りを行う．

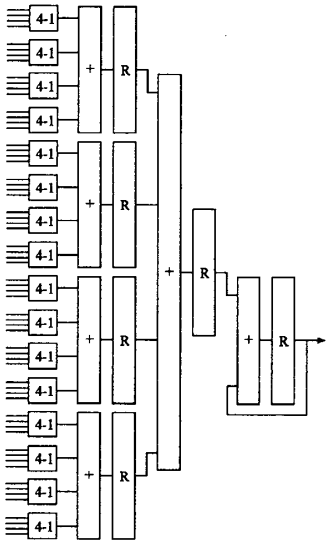

図 1　4 分木状構成のパイプラインビット加算器

3.3 ハードウェアによる実現

本稿では提案手法を FPGA を用いてハードウェアとして実現する．$stack$ などの配列は FPGA の内部メモリで実現し，変数はレジスタで実現する．スタックトップの候補節点に対する演算は並列に実行する．提案手法では配列中の 1 の総数を数える操作が頻繁に使われるため，効率の向上のためにはこの操作を高速化することが

有効である．本稿では，著者等の以前の研究 [5, 2] と同じく，図 1 に示す 4 分木状に構成したパイプラインビット加算器を導入して，高速化を実現した．図中において “4-1” は 4-1 マルチプレクサ，“+” は加算器，R はパイプラインレジスタを表す．

3.4 インスタンス依存ハードウェア解法の実現手順

本稿で提案する最小支配集合を求めるハードウェア解法の実行手順について説明する．まず入力データとして最小支配集合問題のインスタンス (グラフデータ) が与えられる．最小支配集合問題を解くアルゴリズムの大部分はデータに対して独立であるので，あらかじめ Verilog-HDL で記述されたデザインテンプレートとして用意しておく．次に与えられたインスタンスを読み込み，デザインテンプレートを修正することによって Verilog-HDL の回路記述を完成させる．デザインテンプレート修正プログラムは Perl 言語を用いて作成した．図 2 に Verilog-HDL 回路記述の生成手順の例を示す．そして完成した Verilog-HDL 記述を論理合成ツールを用いて FPGA の回路データに変換する．最後に FPGA 上に回路データをダウンロードして提案手法を実行することによって解を得る．

4 実験的評価

4.1 実験環境

提案手法を FPGA ボードを用いて実際に構成し，ソフトウェア解法との比較を行った．提案手法の実験環境を以下に示す．ハードウェア記述言語は Verilog-HDL を用いた．アルゴリズムの大部分はグラフデータに対して独立であるので，あらかじめ Verilog-HDL で記述したデザインテンプレートを用意し，そのデザインテンプレートの修正プログラムは Perl 言語を用いた．生成された Verilog-HDL 回路記述を FPGA 上に実装するために Altera 社の FPGA 設計ツール Quartus II Version 5.0 により論理合成，配置配線を行った．実験で用いた FPGA ボードには三菱電機マイコン機器ソフトウェア株式会社の FPGA 評価ボード Power Medusa MU200-SX60 を用いた．本ボード上の FPGA は Altera 社の FPGA EP1S60F1020C7(論理エレメント数 57120) が搭載されている．なお Quartus II Version 5.0 は CPU が Pentium4 3.40GHz，主記憶は 3GB の PC 上で実行した．また提案手法は FPGA 評価ボードのクロック周波数を 80MHz に設定して実行した．提案手法の実行時間は回路に組み込んだハードウェアカウンタにより計測した．

比較対象のソフトウェア解法としては，最小支配集合に対する厳密解法はほとんど発表されていないため，ここではアルゴリズムとしては提案ハードウェア解法と基本的には同一の分枝限定法に基づくアルゴリズムを採用し，C 言語を用いてプログラムとして実現した．ただし，元のハードウェア解法では並列処理で実現している部分はすべて逐次実行で実現している．このプログラムを CPU が UltraSPARC IIIi 1.062GHz，主記憶は 2GB のワークステーション上で実行した．実験で用いた最小支配集合問題のインスタンスのグラフデータは，節点数と枝数を入力するとランダムなグラフが生成されるプログラムを C 言語で作成し，生成した．ただし，生成されたグラフはすべて連結グラフであることが保証されている．

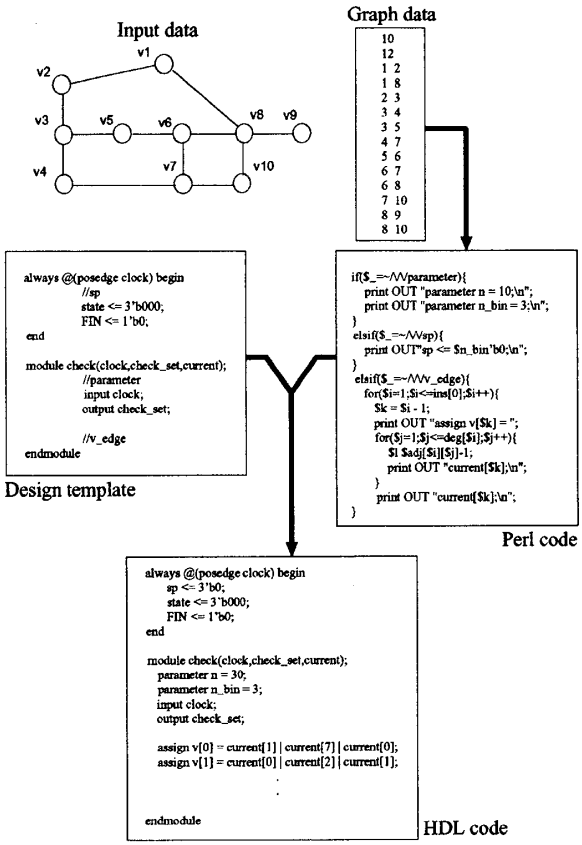

図 2 $Verilog - HDL$ 回路記述の自動生成

4.2 実験結果

各グラフデータに対して比較手法のソフトウェア解法，提案手法のハードウェア解法をそれぞれ実行した結果を表 1 に示す．表中の枝密度とは，同じ節点数の完全グラフに対する枝数の比率を示す．すなわち，節点数 n，枝数 e のグラフ G の枝密度 $\rho(G)$ は $\rho(G) = \frac{2e}{n(n-1)} \times 100(\%)$ である．支配数は最小支配集合の節点数である．また，条件 1 は分枝限定条件 1 を実現するために算出され，FPGA に組み込まれた極大 2 節点独立集合の総数である．LE は回路の実現に要した論理エレメントの利用率である．表中の “$> 24h$” は 24 時間以内では結果が求まらなかったことを示す．

まず，表 1 で比較手法 (ソフトウェア) と提案手法の実行時間に着目してみると提案手法のクロック周波数は比較手法のクロック周波数の $\frac{3}{40}$ であるにもかかわらず圧倒的に高速に求めていることがわかる．しかし，本研究の提案手法はインスタンスに依存しており毎回インスタンスに応じて回路を再構成する必要があるため回路の合成時間も考慮に入れなくてはならない．提案手法は合成時間も含めて比較手法と比べるとグラフデータが小規模な場合は比較手法より劣ってしまっていることがわかる．提案手法が有利になるのはグラフデータが大規模であるときである．

表 1　実験結果: ソフトウェア解法との比較

節点数	枝数	枝密度 (%)	支配数	ソフトウェア (秒)	提案手法 (秒)	合成時間 (秒)	LE(%)	条件 1
100	248	5	20	293	0.23	784	26	200
100	495	10	12	51	0.09	818	26	200
100	743	15	9	11	0.02	968	27	100
100	990	20	7	2	0.002	1110	31	54
100	1237	25	6	1	0.001	1225	35	8
120	357	5	21	9151	7.42	930	34	240
120	714	10	13	716	1.12	1099	33	240
150	559	5	23	$> 24h$	1546.87	1755	43	300
150	1117	10	14	23637	29.27	2277	45	300
200	1990	10	15	$> 24h$	1832.23	11850	69	300

次に，グラフの枝密度と実行時間にどのような関係があるかを考察する．実験結果の表 1 を見ると（特に節点数 100 のグラフ），枝密度が小さいほど比較手法，提案手法共に実行時間が増加することがわかる．また，枝密度が小さくなるほど提案手法が有利になることがわかる．一般に，枝密度の小さいグラフほど最小支配集合を効率よく見つけることは困難になるので，提案手法はソフトウェア解法では実用的な計算時間で解くことが困難な枝密度の小さいインスタンスに対しては，非常に有効な解法であることがわかる．

4.3 考察

前節の実験結果から提案手法は実行時間に関して優れていることがわかる．この理由として考えられるのは，まず，データに依存した解法であることがあげられる．ソフトウェアによる解法はどのようなデータにも対応できる汎用性はあるが，グラフの情報はアルゴリズム自体には含まれていない．そのため例えばある候補節点の有効支配数を調べたい場合，まずはその節点に隣接する節点を調べる．次にその隣接節点が被支配節点であるかどうかを調べ，被支配節点でなければ有効支配数を増やすという手順になる．一方，提案手法では節点間の隣接関係も回路化してあるので実行に必要な動作クロック数を減らすことができる．

次の理由として挙げられるのがハードウェア化により並列処理が可能になった点である．例えばある節点が支配節点あるいは被支配節点であるかどうかを調べる場合，ソフトウェア解法では各節点に対して，その節点自身が支配節点かどうか，あるいはその隣接節点に支配節点があるかどうかを逐次的に調べることになる．しかしハードウェア解法の場合，節点の隣接関係はすべて回路に埋め込まれており，すべての節点に対して，その節点自身が支配節点であるかということと，その節点の隣接節点に支配節点があるかという判定を同時に行うことができるので実行における動作クロック数を減らすことができる．

今後の課題としては，まず提案手法の分枝限定法の改良によってより効率的な探索を実現することで実行時間を減らすことが挙げられる．また回路規模の増加が回路の合成時間に直結するので回路設計の工夫によって回路規模を削減することも課題の一つである．

5 おわりに

本稿では最小支配集合問題に対するデータ依存ハードウェア解法のアルゴリズムを提案し，FPGA に実装し，比較手法のソフトウェア解法と実行時間を比較することにより提案手法の有効性を実証した．今後の課題としてはより効率的な分枝限定法の開発と回路規模の削減があげられる．

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)，および財団法人サタケ技術振興財団サタケ研究助成金の支援により行った．

参考文献

[1] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H.Freeman and Company, 1979.

[2] 菊池健司, 若林真一: “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”, 情報処理学会システム LSI 設計技術研究会研究報告, 2005-SLDM-118, pp.51–56, 2005.

[3] M. Platzner, “Reconfigurable accelerators for combinatorial problems,” in *IEEE Computer, 33(4)*, pp. 58–60, 2000.

[4] T. Suyama, M. Yokoo *and* H. Sawada, “Solving satisfiability problems on FPGAs,” in *Proc. 6th International Workshop on Field-Programmable Logic and Applications (FPL'96)*, pp. 136–145, 1996.

[5] 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”, 信学技報, VLD2003-135, 2004.

[6] Wayne Wolf, *FPGA-Based System Design*, Prentice Hall, 2004.

[7] 柳浦睦憲, 茨木俊秀: 組合せ最適化，朝倉書店, 2001.