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

2004－ＳＬＤＭ－113　(22)
2004／1／23

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

若林　真一†　　菊池　健司†

† 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: †wakaba@computer.org, ††kikuchi@ce.hiroshima-cu.ac.jp

あらまし　グラフの最大クリークを求めるハードウェアアルゴリズムを提案し，FPGA 上に実装して評価した結果を報告する．提案アルゴリズムは与えられたグラフのインスタンスに基づいて構成され，分枝限定法に基づく解探索により最大クリークを効率よく求めることができる．提案アルゴリズムはハードウェアによる実現を前提としており，並列処理とパイプライン処理により効率のよい分枝限定を実現している．提案手法を FPGA 上に実装して実行時間を実測し，ソフトウェアによる解法と比較して提案アルゴリズムが高速に最大クリークを求めることを確認した．
キーワード　FPGA, 最大クリーク，インスタンス，分枝限定，論理合成

Implementation and Evaluation of an Instance-Specific Hardware Algorithm for Finding a Maximum Clique

Shin'ichi WAKABAYASHI† and Kenji KIKUCHI†

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

Abstract　This report presents a hardware algorithm for finding a maximum clique of a given graph, and shows experimental evaluation of implementation of the proposed algorithm on an FPGA. The proposed algorithm is constructed according to a given instance of graph, and can find a maximum clique efficiently based on branch and bound search. The proposed algorithm is supposed to be implmented 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 produced a maximum clique in a shorter running time.
Key words　FPGA, maximum clique, 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 [5]．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 [7]．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 [2]，グラフの最小被覆問題 [8]，論理式の充足可能性判定 [10] 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．

著者らは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 [4]．最大クリーク問題は多くの応用問題を持つ組合せ問題であるが，NP-困難な問題であるため，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]．このため，分枝限定法に基づく多くの最適解探索手法が提案されている [6], [9]．しかしながら，これらの従来手法はソフトウェアで実現することを前提として開発された手法であるため，ハードウェアで効率よく実現することは困難である．

本稿では最大クリーク問題に対し，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最大クリークを求めることを目的としている．本稿で提案する最大クリーク問題に対するハードウェア解法は文献[4]において著者らが提案したインスタンスに特化したハードウェア解法の分枝限定をさらに強化したものである．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. 最大クリーク問題

2.1 定　　義

$G=(V,E)$を無向グラフとし，$V=\{v_1,v_2,\ldots,v_n\}$をGの節点集合，$E=\{e_1,e_2,\ldots,e_m\}\subseteqq V\times V$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，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$G(V')=(V',E')$が完全グラフであるならば，V'をクリークという．また，V'を真に包含し，かつGのクリークである$V''\subseteqq V$が存在しないとき，V'をGの極大クリークという．さらに，Gの極大クリークの中で節点数の最大のクリークをGの最大クリークといい，最大クリークの節点数を$\omega(G)$で表す．

任意の無向グラフGの最大クリークを一つ見つける問題を最大クリーク問題という．最大クリーク問題はNP-困難であることが知られている[1]．

2.2 従来手法

最大クリーク問題はグラフに関する基本的な組合せ問題であると共にさまざまな応用が知られているため，これまでに多くの解法が提案されている．これらの解法は必ず最大クリークを求めることが保証されている厳密解法と，最大クリークを求める保証はないが計算時間は短い発見的解法に分類できる[1]．前者の大半は分枝限定(Branch and Bound)法を基本にしており，様々な手法が提案されている[6],[9]．しかしながら，著者らの知る限り，本稿で提案するような最大クリーク問題に対するリコンフィギャラブルデバイスを用いたハードウェア解法は，著者らが以前に発表した手法[4]以外は知られていない．

3. 提案手法

3.1 概　　要

提案手法は分枝限定法に基づいている．提案手法の概要を以下に説明する．節点数nの無向グラフ$G=(V,E)$に対し，各節点vにはあらかじめ全順序が定義されているとし，vの順序を$ord(v)$，$1\leqq ord(v)\leqq n$，で表す．また，以下では$ord(p)=i$の節点pを$v_i\in V$とする．探索中のGの最大クリークの候補節点集合を配列$R(*)$に記憶するものとする．節点pに隣接する節点集合を$\Gamma(p)$で表す．グラフGの節点集合$V=\{v_1,v_2,\ldots,v_n\}$の部分集合V_iを$V_i=\{v_i,v_{i+1},v_{i+2},\ldots,v_n\}$と定義する．ただし，$V_1=V$である．

V_iに対するGの誘導部分グラフ$G(V_i)$に対する最大クリーク探索アルゴリズムを以下に示す．

部分グラフ$G(V_i)$に対する最大クリーク探索アルゴリズムmq

ステップ0　(初期化) $R(0)=V_i$，$Q=\emptyset$，$M_i=\emptyset$，$sp=0$.

ステップ1　$R(sp)=\emptyset$であればステップ8にいく．そうでなければ，$R(sp)$から$ord(p)$が最小の節点pを選択する．

ステップ2　$R_p=R(sp)\cap\Gamma(p)$．Qにpを加え，$R(sp)$からpを除去する．

ステップ3　(解の判定) $R_p=\emptyset$であるか，あるいはR_pがクリークであると判定できればステップ6にいく．

ステップ4　(限定) $R_p\cup Q$が$|M_i|$よりサイズの大きいクリークを含まないと判定できればステップ7にいく．[分枝限定条件1,2,3]

ステップ5　(分枝) $R(sp+1)=R_p$，$sp=sp+1$とし，ステップ1に戻る．

ステップ6　(解の更新) $|R_p\cup Q|>|M_i|$ならば$M_i=R_p\cup Q$とする．

ステップ7　Qから$ord(p)$が最大の節点pを除去する．

ステップ8　(終了判定)$sp=0$であれば終了する．そうでなければ，$sp=sp-1$とし，ステップ1に戻る．　□

提案手法においては，QにはグラフGのクリークが常に格納されており，さらに，R_pの各節点とQのすべての節点は隣接していることが保証されている．このため，R_pが空集合になるか，あるいはR_pがクリークになれば，$R_p\cup Q$はGのクリークとなる．提案手法はステップ3においてGの極大クリークが見つかるか，あるいは現在の候補節点集合からは最大クリークが見つからないことが判明するまで探索を進める．配列$R(sp)$はスタックとして実現でき，その場合，spはスタックポインタである．

節点集合に対する全順序はどのようなものであっても提案手法が正常に動作することは保証されているが，どのような全順序を用いるかは手法の効率に影響を与える．本稿では次のように定義される順序付けを用いている．まず，グラフに対する彩色問題を近似的に解く．ここでグラフの彩色問題とは隣接する節点には異なる色を彩色する，という条件のもとで色数を最小とする節点彩色を求める問題である．この問題はNP-困難であるが，効率のよい発見的手法が知られている．発見的手法により節点彩色を求めたら，次に，同じ色ごとに節点をクラス分けし，同じクラスに属する節点については節点次数の大きい順に並べることで全順序を定義する．

与えられたグラフGの最大クリークを求める提案手法mqrの全体を以下に示す．GのV_iに対する誘導部分グラフ$G(V_i)$に対する最大クリークのサイズを$c_{max}(i)$，その上限を$c_{ub}(i)$とする．提案手法mqrでは$G(V_n)$，$G(V_{n-1})$，…，$G(V_2)$，$G(V_1)$の順に，最大クリークを求めている．

Gに対する最大クリーク探索アルゴリズムmqr

ステップ0　(初期化) $i=n$，$c_{ub}(n)=1$，$M=\{v_n\}$.

ステップ1　$i=i-1$．$i=0$であれば終了する．$G(V_i)$の探索を行う必要がない場合は$c_{ub}(i)=c_{ub}(i+1)$とし，ステップ

1 を繰り返す. [分枝限定条件 4]

ステップ 2 アルゴリズム mq により $G(V_i)$ の最大クリーク M_i を求める.

ステップ 3 $c_{max}(i) > c_{ub}(i+1)$ ならば, $M = M_i$.

ステップ 4 $c_{ub}(i) = c_{max}(i)$. ステップ 1 に戻る. □

3.2 解の判定と分枝限定

一般に, 分枝限定法において, 効率よい解探索を実現するためには, 解の判定と分枝限定を効率よく行うことが重要である. 本手法で用いている解の判定と分枝限定の条件を以下に示す.

3.2.1 有効節点次数

節点 v の隣接節点集合 $\Gamma(v)$ の要素数を節点次数とよび, $d(v)$ で表す $(d(v) = |\Gamma(v)|)$. 分枝限定法による解探索の途中において, その時点での最大クリークの探索の対象となっている節点集合を候補節点集合 R とよび, その要素を候補節点という. また, 候補節点 v に隣接する候補節点数を候補節点次数とよび, $d'(v)$ で表す.

候補節点 v に対し, v に隣接する候補節点集合 ($\Gamma'(v)$ で表す) の部分集合 $F \subseteqq \Gamma(v)$ において, F の任意の 2 節点が互いに隣接しないとき, F の要素は高々1 つしか最大クリークの要素になりえない. このことを利用すると, 節点次数を利用した分枝限定を効率よく行うことができる. まず, 節点 v の隣接節点集合 $\Gamma'(v)$ の部分集合として $F_1, F_2, \ldots, F_k$ を求める. ただし, 各 F_i は互いに隣接しない 2 個以上の節点から構成され, かつ, 任意の 2 個の部分集合は互いに節点独立とする. また, $\Gamma'(v)$ の要素で, いずれの F_i にも含まれない節点の集合を F_0 とする.

このとき, 節点 v の有効節点次数 $d_{active}(v)$ を以下のように定義する.

$$\begin{aligned} d_{active}(v) &= d'(v) - \sum_{i=1}^{k} (|F(i) - 1|) \\ &= |F_0| + k \end{aligned}$$

解探索のある時点において, それまでに見つかった最大クリークの節点数を c_{max} で表す. 分枝限定法による解探索の途中において, 有効節点次数 $d_{active}(v)$ が c_{max} より小さい節点は解探索の候補節点集合 R から除去する.

[分枝限定条件 1] 候補節点集合 R において, 有効節点次数が c_{max} 以上の候補節点が $(c_{max}+1)$ 個以上存在しない場合, R に対するそれ以上の探索は行わない. □

また, 候補節点集合 R_p がクリークであるかどうかは R_p のすべての節点 v に対し, $d_{active}(v) = |R_p| - 1$ であることを調べればよい.

なお, 有効節点次数を定義するための節点集合 $F_i, 1 \leqq i \leqq k$ はできるだけ多くの候補節点集合を含み, かつ, 集合の総数 k は少ないほうがよいが, 厳密な解を求めようとすると, この問題自体が NP-困難になる. また, 節点集合 F_i の構成は提案手法の動作の正当性には影響を与えない. このため, 本稿では与えられたグラフ G に対して, 前処理として, グリーディな手法により各節点の F_i を構成している. また, 効率よく F_i を求めるために, F_i の要素数は高々8 としている.

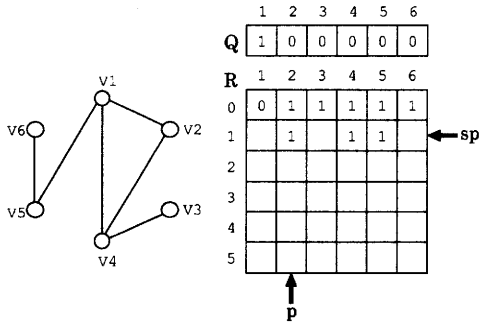

図 1 候補節点集合を記憶するスタック

3.2.2 探索範囲の重複

提案手法のステップ 1 において, $R(sp)$ よりある節点 p を選択することにより次の探索範囲を決めているが, $R(sp)$ の別の節点 q があって $R_q \subseteqq R_p$ が成り立つ場合, R_p を探索すれば, その後, R_q を探索する必要はない.

[分枝限定条件 2] 候補節点集合 R において, 次のステップの探索範囲として $R_p \subseteqq R$ を選択したとき, その後のステップにおいて, $R_q \subseteqq R_p$ である候補節点集合は探索しなくてよい. □

3.2.3 部分グラフに対する探索

[分枝限定条件 3] 候補節点集合 R において, $ord(p)$ が最小の節点を選択して R_p に対して探索を行うとき, $sp + c_{up}(ord(p)) \leqq c_{max}$ ならば, R_p の探索は行う必要がない. □

[分枝限定条件 4] V_i に対する誘導部分グラフ $G(V_i)$ に対する最大クリークのサイズが $c_{max}(i)$ であり, かつ, 最大クリークが v_i を含むとする. このとき, 任意の $v_j, j < i$ に対し, 節点 v_j と $j < k \leqq i$ であるすべての節点 v_k が隣接していなければ, 誘導部分グラフ $G(V_j)$ の最大クリークのサイズは $c_{max}(j) = c_{max}(i)$ となり, $G(V_j)$ に対する探索を行う必要はない. □

4. ハードウェアによる実現

4.1 回路構成

4.1.1 候補節点集合

提案手法は FPGA 上でハードウェアとして実現する. 候補節点集合を格納する配列 $R(*)$ はビット幅が $n = |V|$ のメモリで実現するものとし, メモリアドレスが a 番地の i ビット目を $R[a]_i$ で表す. アルゴリズムにおいて $v_i \in R(a)$ ならば $R[a]_i = 1$, そうでなければ $R[a]_i = 0$ とする. また, 回路において, メモリの各ビットはそれぞれ独立に値を書き込むことができるものとする. 図 1 に 6 節点のグラフに対するアルゴリズムの動作の様子とスタックの内容を示す.

4.1.2 有効節点次数の計算

提案手法においては, 候補節点集合の各節点の有効節点次数を効率よく計算することが重要である. 以下では例を用いて, 有効節点次数の計算法を示す. 今, 節点 v の隣接節点集合を $\Gamma(v) = \{u_1, u_2, u_3, \ldots, u_8\}$ とし, 隣接節点集合の部分集合で, 互いに隣接していない節点集合を $F_1 = \{u_1, u_2, u_3\}$,

$F_2 = \{u_4, u_5\}$ とする．u_6, u_7, u_8 は他の隣接節点のいずれかと隣接しているものとする．各 u_i は候補節点集合に含まれていれば 1，含まれていなければ 0 とする．このとき，v の有効節点次数は

$$d_{active}(v) = OR(u_1, u_2, u_3) + OR(u_4, u_5) + u_6 + u_7 + u_8$$

で計算できる．ただし，OR は u_i の値を論理値と見なしたときの論理和であり，加算は OR の演算結果と各 u_i を数値と見なして加算することを意味する．

ハードウェアで有効節点次数の計算回路を実現する場合，基本的には上記の式を計算する回路を実現すればよいが，実現にあたっては考慮すべき点が 3 点ある．まず，加算回路の構成であるが，ある節点の有効節点次数を求める場合，加算の対象となる項数は最大で $(n-1)$ となる (n は節点数)．このため，加算器を 4 分木構成で組合せて加算回路を構成する．この場合，木の高さは $O(\log_4 n)$ となる．

次に，FPGA 上に実現する回路の動作速度は回路のクリティカルパスによって決まり，加算器を 4 分木構成して有効節点次数の計算回路を組合せ回路として実現すると，クリティカルパスが長大となり，回路の動作周波数が著しく低下することが予想される．これを避けるために，4 分木の各レベルごとにクリティカルパスを考慮し，必要に応じてレジスタを挿入する．

さらに，提案手法では，各節点の有効節点次数を並列に計算することにしているが，節点数が大きい場合，回路規模が膨大となり，FPGA 上に実現できなくなる恐れがある．このことを避けるため，有効節点次数の計算式の項を複数に分割し，セレクタで選択しながら，パイプライン演算により有効節点次数を計算する．

有効節点次数の計算式に含まれる OR 演算と節点の総数が 64 項であるときの回路構成を図 2 に示す．入力はまず 4-1 セレクタで選択され，4 個ずつ加算される．図中の R はパイプラインレジスタであり，クロックごとに異なる入力が選択されて加算回路に入力される．パイプラインの最終段はアキュムレータとなっており，それまでの入力の総和が計算される．この回路では，6 クロックで有効節点次数が計算できる．

ある節点の有効節点次数が計算されるとその結果がその時点までに見つかった最良の極大クリークのサイズと比較され，その結果，有効節点次数が最良の極大クリークのサイズ未満であれば，その節点は候補節点集合から除去される．そうすると他の節点が候補節点でなくなる可能性が出てくるので，候補節点集合が更新されなくなるまで，有効節点次数の計算を繰り返す．一般には数回の繰り返しで計算が収束する．

4.1.3 探索開始節点の探索

提案手法では，候補節点集合の中で最も $ord(p)$ の小さな節点，あるいは大きな節点を選択する必要がある．この回路も節点単位で比較することで探索することにすると，回路のクリティカルパスの長さが $O(n)$ となり，動作周波数が著しく低下する恐れがある．このため，加算回路の桁上げ先見回路と同様の仕組みを導入することで，クリティカルパスの長さを $O(\log n)$ としている．

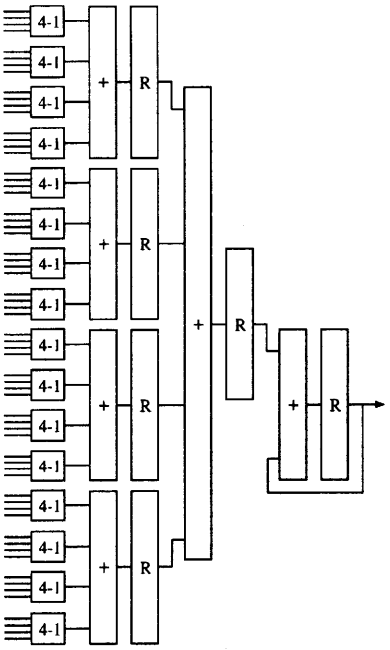

図 2　有効節点次数の計算回路

4.2 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．

インスタンス非依存回路による解生成では，回路の設計と FPGA 上への実現は 1 度行えばよく，インスタンスは実行時に回路の入力として与えられる．一方，インスタンス依存回路による解生成では，回路の設計と FPGA 上への実現はインスタンスが変更されるごとに行う必要があり，回路中にインスタンスの情報は組み込まれているので，実行時にはインスタンスを回路に入力する必要はない．

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，提案手法の有効節点次数の計算において，任意のグラフの有効節点次数を計算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．

5. 評　価

5.1 実験環境

提案手法をFPGA評価ボード上に実現し，評価を行った．実験環境を以下に示す．提案手法のハードウェア記述にはVerilog-HDLハードウェア記述言語を用いた．問題のインスタンスに対応したハードウェア記述をテンプレートファイルから生成するための変換プログラムはPerl言語で記述した．生成されたハードウェア記述をAltera社製のFPGA設計ツールQuartus Version 3.0で論理合成し，配置配線を行った．FPGA評価ボードとしてはボード上にFPGAチップとしてAltera社EP20K1500E (最大システムゲート数239万，公称ゲート数150万，論理エレメント数51840)が搭載されている三菱電機マイコン機器ソフトウェア株式会社製のFPGA評価ボードPowerMedusa MU200AP1500を用いた．実験時にはFPGAのクロック周波数を20MHzに設定した．FPGA設計ツールはCPUがPentium 4 2.4GHzのPC(OSはWindows2000，主記憶2GB)上で実行した．また，提案手法の分枝数(ステップ5の実行回数)を評価するために，提案手法の動作をシミュレートするプログラムを作成し，分枝数を求めるのに用いた．

比較対象のソフトウェアとして，最大クリーク問題の手法の評価のために米国離散数学理論計算機科学センター(DIMACS)より公開されている最大クリーク問題のためのベンチマークソフトウェアdfmaxを用いた[3]．dfmaxはC言語で記述されており，CPUがPentium M 1.4GHzでOSがLinuxのノートパソコン上でコンパイルの上，実行した．

問題のインスタンスとしては，DIMACSより公開されている最大クリーク問題のベンチマークデータを用いた[3]．実験で使用したFPGAチップのゲート数の制約(公称ゲート数150万ゲート)から，200節点より大きなグラフに対してはFPGA上での実現が困難だったので，実験においては節点数が200節点のデータからdfmaxでは効率よく解を求めることのできないベンチマークデータを選んで，dfmaxと提案手法で実験し，実行時間を比較した．また，参考のため，節点数400のベンチマークデータに対する提案手法のソフトウエアシミュレーションの結果も示す．

提案手法の実行時間は回路中にカウンタをハードウェアで組み込み，実行に要したクロック数をカウントして，算出した．ソフトウェアの実行時間はLinuxのtimeコマンドにより測定した．また，設計ツールの実行時間はツールが表示する回路生成時間を用いた．

また，一部のデータについては4.1.2に示した有効節点次数の計算回路ではゲート数の制約のために最適な回路がFPGA上に実現できなかったため，次数を逐次的に計算する回路となっている．このため，節点次数に比例するクロック数が必要となり，大幅な実行時間の増大を招いている．また，今回の回路生成においては，回路サイズの制約から提案手法の分枝条件3による分枝限定も実現していない．この条件による分枝限定の実現も今後の課題である．

5.2 実験結果

実験結果を表1に示す．表においてnはグラフの節点数，mは枝数，ρはグラフの枝密度($\rho = \frac{2\times m}{n\times(n-1)}$)，$\omega$は最大クリークの節点数，dfmaxは比較対象のベンチマークプログラムdfmaxの計算時間(単位は秒)，mqrは提案手法mqrの実行時間(単位は秒)，LE数はFPGA上で回路を実現するのに使用した論理エレメント数(カッコ内は論理エレメントの使用率)，節点次数は各節点の有効節点次数を計算する回路の1回の計算あたりのクロック数，生成時間は回路の生成に要した時間(単位は分)，分枝数は提案手法をソフトウェアでシミュレーションすることにより求めた提案手法の分枝数(ステップ5の実行回数)である．節点次数において*のついているデータは逐次的に節点次数の計算を行う回路が使われたことを示す．dfmaxの計算時間において，">24h"となっているのは24時間以上実行しても解が求まらなかったことを示す．節点数400のデータ(san400_0.7_1, san400_0.7_2)の結果はdfmaxと提案手法のソフトウェアシミュレーションによる結果である．

実験の結果，brock200_1を除いて，提案手法のほうが比較手法dfmaxより短い実行時間で最大クリークを求めている．また，回路の生成に必要な時間を実行時間に加えてもsan200_0.7_2とsan200_0.9_3に対しては提案手法のほうが比較手法より実行時間が短い．提案手法の実行時間とソフトウェアシミュレーションによる分枝数が比例しないのは有効節点次数の計算における繰り返し回数の影響とソフトウェアシミュレーションでは分枝条件3による分枝限定を行っていることが主な要因と考えられる．400節点のデータについても，より大きなFPGAが利用可能になればdfmaxより非常に高速に最大クリークを求めることが可能になることが予測される．

回路の生成時間は論理エレメント(LE)の使用率が80%を越えると急に増大する．これは配置配線が困難になるためだと考えられる．より大規模なFPGAが利用可能になれば配置配線の困難さが少なくなり，回路生成に必要な計算時間を短縮できると考えられる．

5.3 考察と課題

提案手法に関する考察と課題を以下に示す．まず，提案手法の解探索能力について考察する．実験結果の分枝数からわか

表 1　ベンチマークデータに対する実験結果

データ	n	$m(\rho)$	ω	dfmax[秒]	mqr[秒]	LE 数	節点次数	生成時間 [分]	分枝数
san200_0.7_1	200	13930 (0.7)	30	7532.89	1.60	41663 (80.4%)	5	204	27944
san200_0.7_2	200	13930 (0.7)	18	46937.71	0.031	29056 (56.0%)	5	128	49990
brock200_1	200	14834 (0.75)	21	39.51	51.9	42448 (81.9%)	40*	225	5839721
san200_0.9_3	200	17910 (0.9)	44	>24h	335.0	44502 (85.8%)	16*	714	30098685
san400_0.7_1	400	55860 (0.7)	40	>24h					2366386
san400_0.7_2	400	55860 (0.7)	30	>24h					4443166

るように，同じ節点数のグラフでも枝密度等によって分枝数が大きく異なっている．特に brock200_1 に対しては，比較対象としたベンチマークプログラム dfmax と提案手法 mqr の実行時間は同等である．これは，他のデータとは異なり，dfmax は brock200_1 に対しては効率よい分枝限定を行ったのに対し，mqr の分枝限定は効率が悪かったことを示している．ソフトウェアで実現することを前提とした最大クリーク問題に対する解探索手法としては多くのものが提案されており，これらはいずれも分枝限定の効率化を考慮している．その結果，dfmax と比較してデータによっては 1000 分の 1 以下の計算時間で解を求めることが可能な手法も提案されている [9]．これらの手法はソフトウェアによる実現を前提としているため，必ずしもハードウェア化が容易であるとは限らないが，これらの先進的なソフトウェア解法も参考にしながら提案手法の分枝限定を改良することが今後の課題の一つである．

また，著者らが [4] で示しているように，複数の節点からの探索を同時に実行する並列処理も解探索の効率化に効果がある．このような解探索においては探索途中で得られる情報 (部分グラフに対する最大クリークのサイズなど) を共有することで複数の節点からの解探索を独立に行う場合より解探索が効率化する．このような並列分枝限定手法の研究も今後の課題である．

今後，FPGA の回路規模はますます増大し，数年以内には今回の実験に用いた FPGA の 10 倍以上の規模の FPGA も利用可能になると予想される．また，動作可能周波数も 10 倍以上になると予想される．そのような FPGA の大規模化，高速化を前提とした場合の実行時間について考察する．提案手法の分枝限定は節点数を n としたとき，$O(\log n)$ のクロック数で実行可能である．一方，ソフトウェア解法による分枝限定は工夫してプログラムを作成したとしても原理的には 1 回の分枝限定に少なくとも $O(n)$ の計算時間を必要とする．たとえば提案手法の分枝限定をソフトウェアで実現した場合の 1 回の分枝限定あたりの計算時間は $O(n^2)$ である．このことは FPGA の規模が大規模になり大規模なグラフを扱えるようになるほど，提案手法はソフトウェア解法より実行時間において有利になることを意味している．本稿の実験では，FPGA のクロック周波数が 20MHz に対し，比較手法 dfmax を実行した CPU のクロック周波数が 1.4GHz と 70 倍の差があるのにも関わらず，提案手法は比較手法と同等かそれ以上の効率のよい解探索を実現した．今後の FPGA の大規模化に伴い，回路の生成時間を考慮しても提案手法の有利性が大きくなることが予想される．

最後に，提案手法はインスタンスに依存したハードウェア回路の生成を行うが，この生成時間を短くする，あるいはなくする工夫が望まれる．生成時間を短くするためには配置配線を含めたテンプレートのような構造をあらかじめ用意しておき，与えられたインスタンスに応じて必要な配線を加える，などの処理が考えられる．あるいは，提案手法を完全にデータ非依存のハードウェアとして実現することも考えられる．FPGA が大規模化，高速化すればこのような実現も可能になると予想される．

6. あとがき

本稿ではグラフの最大クリーク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，より高速に最大クリーク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 15 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] I.M.Bomze, M.Budinich, P.M.Pardalos, M.Pelillo, The maximum clique problem, in Handbook of Combinatorial Optimization, Supplement Volume A, pp.1–74, Kluwer Academic Publishers, 1999.

[2] 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.

[3] DIMACS Implementation Challenges, http://dimacs.rutgers.edu/Challenges/

[4] 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.

[5] 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.

[6] P.R.J.Östergård, A fast algorithm for the maximum clique problem, Discrete Applied Mathematics, 120, pp.197–207, 2002.

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

[8] C.Plessl, M.Platzner, Instance-specific accelerators for minimum covering, Proc. 1st Int'l Conf. on Engineering of Reconfigurable Systems and Algorithms, pp.85–91, 2001.

[9] 関友和, 富田悦次, 分枝限定法を用いた最大クリーク抽出アルゴリズムの効率化, 信学技報, COMP2001-50, 2001.

[10] 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.