システムＬＳＩ設計技術　108－19
(2003. 1. 29)

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

藤原　知幸†　　若林　真一†

† 広島大学大学院工学研究科
〒739–8527 広島県東広島市鏡山一丁目4番1号

E-mail: †{fuji,wakaba}@ecs.hiroshima-u.ac.jp

あらまし　最大クリーク問題はグラフ理論における基本的問題の一つであるが，大規模かつ複雑な問題をソフトウェアで解くと計算時間が膨大になり，実用時間内で解を得ることができない．本研究では，大規模かつ複雑な最大クリーク問題を対象とし，問題のインスタンスに特化したハードウェアをFPGAを用いて実現することにより最大クリーク問題をハードウェアで解くことを目的とする．与えられた問題のインスタンスをハードウェアで解く場合，インスタンスに特化したハードウェアを短時間で構成可能にすると共に，インスタンスの変更に応じて適宜ハードウェアを再構成することが求められる．そこで，再構成可能なLSIであるFPGAを用いることでインスタンスに特化したハードウェアを実現し，最大クリーク問題を短時間で解く手法を提案し，シミュレーション実験により本手法の有効性を示す．

キーワード　最大クリーク，分枝限定法，インスタンスベース，FPGA

Solving the Maximum Clique Problem Using FPGAs with Instance-Specific Information

Tomoyuki FUJIWARA† and Shin'ichi WAKABAYASHI†

† Graduate School of Engineering, Hiroshima University
4-1 Kagamiyama 1 chome, Higashihi-Hiroshima, Hiroshima, 739–8527 Japan

E-mail: †{fuji,wakaba}@ecs.hiroshima-u.ac.jp

Abstract　The problem of finding a maximum clique is known to be one of fundamental problems in graph theory. But solving the problem by software would not be successful within a practical time if the problem is large and complicated. Then we implement instance-specific hardware for a given instance of the maximum clique problem in order to get a solution by hardware. To obtain a solution for a given instance by hardware, we should construct a hardware system dedicated to the specific instance, and make it possible to reconfigure it when another instance is given. For this reason, by using FPGAs that are reconfigurable LSIs, we implement instance-specific hardware and get a solution of the maximum clique problem in a short time. Simulation experiments were performed to show the effectiveness of the proposed method.

Key words　Maximum Clique, Branch and Bound Algorithm, Instance-Based, FPGAs

1. まえがき

無向グラフの極大クリーク，あるいはそれと双対な極大独立節点集合を全て列挙する問題はグラフ理論における重要な基本的問題の一つであり，いくつかの優れたアルゴリズムが考案され，実験的評価がなされてきている[2]．また，極大クリークのうちで節点数の最大なもの，すなわち，最大クリーク(最大完全部分グラフ)あるいはそれと双対な最大独立節点集合を効率よく抽出する問題も興味ある問題であり，種々の実際的応用も知られている．

最大クリーク問題はNP完全であるため，効率の良いアルゴリズムは存在しないと予想されている．このため，これらの問題の解法として分枝限定(branch and bound)法，あるいはSimulated Annealing (SA), Genetic Algorithm(GA)等のメタヒューリスティック手法に基づくアプローチが広く使われているが，SA等のメタヒューリスティック手法は最適解を生成する保証はない．また，これらの手法をソフトウェアで実現した場合，大規模な問題に対しては実用的時間で解を得ることができない．

そこで本研究では，大規模かつ複雑な最大クリーク問題を対象とし，問題の各インスタンスに対応した解法をハードウェアで実現することにより問題の最適解を実用時間内で生成することを目的とする．ある問題のインスタンスをハードウェアで解く場合，そのインスタンスに対応したハードウェア構成を短時間に実現し，かつ，種々のインスタンスに対して適宜再構成可能なハードウェアが必要となる[5]．そこで，本研究では内部論理が再構成可能なハードウェアであるField Programmable Gate Array(FPGA)を用いて，インスタンスに特化したハードウェアを短時間で実現する．また，与えられたインスタンスからハードウェアの回路記述を自動生成し，FPGAにコンフィグレーションしてハードウェアで最大クリーク問題を解くシステムを開発する．

図1に本研究で用いるハードウェア再構成モデルを示す．入力として最大クリーク問題のインスタンスが与えられる．最大クリーク問題を解くアルゴリズムの大部分はインスタンスに独立なので，アルゴリズムにおいてインスタンスに依存しない部分はデザインテンプレートとしてハードウェア記述言語(HDL)による回路記述を用意しておく．次にソースコードの生成では，テンプレートを基にCやPerlなどの言語を用いてインスタンスに特化したハードウェアのHDL記述を生成する．そしてFPGA専用設計ツールを用いてHDL記述を回路データに変換し，最後にインスタンスを解くハードウェアをFPGA上に実現し，問題を解く．

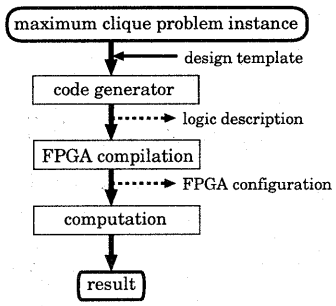

図1　インスタンスに特化した再構成モデル
Fig. 1　Instance-specific reconfiguration

提案アルゴリズムと文献[1]の手法を計算機上に実現し，シミュレーション実験を行った結果，提案手法の有効性が示された．

2. 準　　備

本研究で対象とするのは，自己閉路や並列枝を持たない無向グラフ$G=(V,E)$，但し，Vは節点の集合，$E \subseteq V \times V$は枝の集合，とする．グラフ$G=(V,E)$において，節点vに隣接している節点の集合$\{w \in V | (v,w) \in E\}$，すなわち$v$の近傍(neighborhood)を$\Gamma(v)$で表す．また，節点$v$に隣接している節点の個数$|\Gamma(v)|$を$v$の次数といい，$deg(v)$で表す．ここで一般に，集合$S$に対し，その要素数を$|S|$で表す．なお，節点集合$V$は順序付き集合とし，先頭から$i$番目の要素を$V[i]$で表す．更に，$V$の部分集合を考えるときにも，その要素はもとの集合の順序を引き継いだ順序付き集合とする．

グラフ$G=(V,E)$の部分グラフ$G(Q)=(Q,E(Q))$，但し，$Q \subseteq V$，$E(Q)=\{(v,w) \in E | v,w \in Q\}$において，どの2節点も隣接しているとき，$G(Q)$をクリークと言う．そしてそれが他のクリークの真の部分集合でなければ極大クリーク，また，最も節点数の多い極大クリークであれば最大クリークと呼ぶ．Gの最大クリークのサイズ(節点数)をωで表す．

グラフ$G=(V,E)$の枝密度$\rho(G)$を，$\rho(G)=\frac{2|E|}{|V|(|V|-1)}$で定義する．

3. 提案手法

本研究で対象としているのは，グラフデータが入力されたとき，そのグラフの最大クリークを1つ見つけて出力するアルゴリズムである．以下ではいく

つかの最大クリーク抽出アルゴリズムとその効率化手法について述べる．

3.1 基本アルゴリズム

提案手法の基本アルゴリズムは，ある時点で保持しているクリークに，そのクリーク中の全ての節点と隣接している節点を新たに付け加え，より大きなクリークを得る操作を，深さ優先探索によって進めていくアルゴリズムである．

具体的には，まず初期設定として，探索途中で見出されたクリーク Q を $Q := \emptyset$(空集合) とし，現段階で探索の対象として考えている順序付き節点集合(候補節点集合) を $R := V$ とする．節点順序に関してはグラフが入力された時点で，ある規則に従って番号付けがなされているものとする．番号付けでは種々の方法が考えられるが，以下では 4 章のシミュレーション実験で用いたものを示す．節点 vの順序を $No(v)$ で表す．

《節点彩色による番号付け》

- $(v,w) \in E$であれば，$No(v) \neq No(w)$ である．
- $No(v) = k > 1$ であるとき，$No(w) = 1,2,\cdots,k-1$ なる節点 $w \in \Gamma(v)$ がそれぞれ少なくとも一つ存在する．

この節点彩色は，各節点 vに対して上記の条件を満たす正整数 $No(v)$ を与える操作である．この操作は隣接節点に同じ番号は与えないということであり，ここで番号を色と考えれば，近似節点彩色とみなすことができる．

そして，R中の番号の最も若い節点 p を選び出してこれをクリークに付け加え ($Q := Q \cup \{p\}$)，R中で p に隣接している節点の集合を求めて ($R_p := R \cap \Gamma(p)$)，得られた節点集合 R_p を新たな候補節点集合とみなして同様の操作を適用していく．これにより，各段階の候補節点集合はその時点の Q のいずれの節点にも隣接している節点の集合となっている．

このようにしていって $R_p = \emptyset$ となったとき，そのとき得られている Q は極大クリークである．このとき更にクリーク探索を続けるために，Rから p を取り除いたものを改めて Rと置き直し，Q からも p を削除してから，R中の別の節点を新たに p として選択して再び同様の操作を進めていく．そして $R = \emptyset$ となった場合は，更にもう 1 段階前の候補節点集合へ戻るというバックトラックアルゴリズムで探索していく．そうしていき，最初の候補節点集合 (V) が空になったら探索を終了する．

本研究では最大クリークを 1 つ抽出することを目的としているので，最初に $Q_{max} := \emptyset$ と設定し，より大きな極大クリークが抽出されるたびに Q_{max}を更新していけば，最終的に Q_{max}に記憶されているのが最大クリークである．

しかし上述のままでは節点の全ての組合せについて探索することになり効率が悪い．そこで次に示す方法で効率化を行う．

3.2 分枝限定

クリーク探索の全過程は，全節点集合 Vを根，各時点における候補節点集合 Rを頂点として，Rと R_p の関係にあるものを親子関係の枝で結んで得られる探索木として表現することができる．この探索木における総枝数を分枝数と呼ぶ．アルゴリズムの効率化に当たっては解の探索領域を小さくすることが効果的であり，そのために余分な分枝を制限することが分枝限定法である．

簡単な分枝限定として，新たな Rが生成されたときに $|Q|+|R| \leq |Q_{max}|$ が成立する場合には，そこから先の探索で現在保持している以上のサイズの極大クリークは存在しないことが明らかであるので，それ以上は探索を行わない，とする方法が考えられる．

本研究では基本的にこのような分枝限定法を用いて，効率的なアルゴリズムを開発する．

3.3 提案アルゴリズム mq

提案アルゴリズム mqはハードウェア化を前提として構成するため，単純な操作で分枝限定を実現する必要がある．従って，図2のように記憶領域として候補節点集合を保持する 2 次元配列 $stack$と極大クリークを保持する 1 次元配列 $clique$，最大クリークを保持する 1 次元配列 max_clique 用意し，節点が選択されている状態 (アクティブ)，非選択の状態 (インアクティブ) をそれぞれ 1 と 0 に対応させて実現している．図 2では極大クリークとして $V[1]$ を $clique$ に入れ，$V[1]$ に隣接する節点を $stack$にプッシュした様子を示している ($stack$中の空白は 0 を示す)．

mqは次の手順により実行される．ここで v_active は $stack$の候補節点数，v_clique は $clique$ の節点数，e_active は $clique$ の節点と $stack$の節点間の枝数，e_active_s は $stack$の節点間の枝数 (アクティブ節点間の枝数)，n_clique は v_stackと v_clique の和，$kmax$ は最大クリークの節点数を示す．

《mqの処理フロー》

【Step1】 $stack$の $leftmost$ 節点の計算

【Step2】 $leftmost$ 節点を $clique$ へ

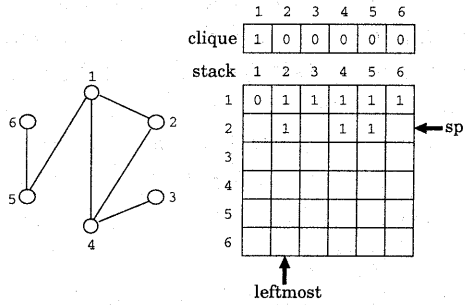

図2 *stack*と*clique*の概念図
Fig. 2 conceptual diagram of *stack* and *clique*

【Step3】 *stack*の中で*leftmost*節点に隣接している節点の集合をプッシュ
【step4】 *e_active*と*e_active_s*の計算
【Step5】 解の判定と分枝限定
〈解の更新〉
if($v\_active == 0$) goto【Step6】
if($n\_clique(n\_clique-1)/2 == e\_active$)
goto【Step6】
〈分枝限定〉
if($n\_clique \leqq kmax$) goto【Step7】
if($e\_active < (kmax+1)kmax/2$)
goto【Step7】
if($e\_active\_s < \{(kmax+1)-v\_clique\}$
$\{kmax-v\_clique\}/2$) goto【Step7】
else goto【step1】
【Step6】 解の更新
if($n\_clique > kmax$)
*max_clique*を更新
【Step7】 分枝限定
*clique*の*rightmost*節点を除去し【step1】へ

まず【Step1】では，*stack*で一番左端の節点(*leftmost*)を計算し，【Step2】でその節点を極大クリーク集合*clique*に加える．【Step3】では，*leftmost*に隣接する節点を*stack*にプッシュする(スタックポインタspの更新)．そして【Step4】では，現段階でのアクティブな枝の数を計算する．【Step5】では，解の更新，または分枝限定を行う．解の更新では，候補節点数*v_active*が0であるため1つの極大クリークを抽出，またはアクティブ節点と枝の関係から極大クリークを抽出，ということを判定している．分枝限定では，解の更新をしない場合の処理をしている．つまり，*clique*に保持されている節点数と*stack*に残っている節点数を足しても，現時点での最大クリークサイズ$kmax$を越えない場合，または枝数と節点数の関係から$kmax$以上のクリークは得られない場合を判定している．【Step6】では，最大クリーク*max_clique*を更新，【Step7】では，一番最後に*clique*に入れた節点(*rightmost*)を除去，つまりバックトラックする．

3.4 逆順探索アルゴリズム *mqr*

逆順探索アルゴリズム*mqr*は，*mq*における節点探索順序を逆にしたアルゴリズムである．この考えは文献[3]に基づくもので，*mqr*では，ハードウェアとして実現できるように，*mq*と同様，*stack*と*clique*を操作することで最大クリークを抽出している．

*mq*では，候補節点pより節点番号の大きい節点を候補節点集合とすることで，同じ探索領域を探索しないようにしている．これは*mqr*で逆順に探索する場合にも同様である．つまり，$S_i=\{v_i, v_{i+1}, \cdots, v_n\}$(インデックス$i$は節点番号)とすると，*mq*では$v_1$を含む$S_1$でクリークを探索し，次に$v_2$を含む$S_2$で，というふうに探索を進める．これに対して*mqr*では，S_nから開始して，S_{n-1}へと進めていく．

*mqr*において*mq*と同様の分枝限定を用いると，探索の初期段階では得られたクリークサイズが小さいため，分枝限定が有効に働かない．そこで，*mqr*に特有の分枝限定として，次のようにする．$c(i)$をS_iの最大クリークサイズとする．明らかに，$1 \leqq i \leqq n-1$に対して，$c(i)=c(i+1)$または$c(i)=c(i+1)+1$である．つまり，$c(i)=c(i+1)+1$であることと，節点v_iを含むサイズ$c(i)=c(i+1)+1$のクリークがS_iに存在することは等価である．従って，$c(n)=1$から探索を開始し，クリークが見つかれば，$c(i)=c(i+1)+1$であり，そうでなければ，$c(i)=c(i+1)$である．最終的に最大クリークサイズは$c(1)$により得られる．この$c(i)$を用いた新たな分枝限定が可能となる．つまり，サイズmaxより大きいクリークを探索しているとして，v_iが$(j+1)$番目の節点であり，$j+c(i) \leqq max$が成り立てば，それ以上の探索を行わない．

3.5 並列処理

*mq*と*mqr*は逐次処理アルゴリズムであるが，これらを組合せることで並列アルゴリズムを構成できる．以下では*mq*のみを並列化した*mq_all*と，*mq*と*mqr*を組合せて並列化した*mq&mqr*を新たに提案する．*mq*は異なる開始節点を指定することで複数の*mq*を同時に実行可能である．一方，*mqr*は*stack*の候補節点pより右に位置する各節点のクリークサイズの上界($c(i)$)が既知でないと分枝限定が行えないため，

複数のmqrを同時に実行することはできない．このため，$mq\&mqr$においてはmqとmqrをそれぞれ実行するプロセッサをk-1個，1個として，並列処理を行っている．また同じ理由から，mqrのみを並列化したモデルは実現できない．

4. シミュレーション

4.1 実験方法

提案アルゴリズムにおいては，ハードウェアでの実行を仮定しているが，現段階ではシミュレーションによる評価までを行っている．

提案アルゴリズムの各シミュレーションモデルはC言語により実現し，仮想的なクロック数により実行時間を換算する．これは，mqとmqrの処理を1クロックで処理可能なステップに区切り，各ステップでクロックをカウントアップしている．以下のシミュレーション結果では，FPGAで実現した場合のクロック周波数80MHzを仮定して実行時間を算出している．

以下にシミュレーションに用いたアルゴリズムを示す．なお，実行時間はグラフデータの読み込みなどを含まないアルゴリズム本体のみの処理時間とした．これらの実行に使用した計算機環境はCPU:Pentium III 1GHzである．

- $dfmax$：DIMACSプロジェクト[4]により公開されている最大クリーク抽出のベンチマークアルゴリズム
- $MCLIQ$：比較手法である文献[1]の分枝限定アルゴリズム
- mq：ハードウェア化を仮定した最大クリーク抽出アルゴリズム
- mqr：mqでの初期節点順序の逆順に探索するアルゴリズム
- mq_all_k：mqのみによる並列処理アルゴリズム(kプロセッサ)
- $mq\&mqr\_k$：mqとmqrによる並列処理アルゴリズム(mq：k-1プロセッサ，mqr：1プロセッサ)

4.2 実験結果

表1にランダムグラフに対するシミュレーション結果を示す．表2は並列度を変化させた場合の実行時間を示している．表3では，並列度を5とした場合の速度向上比を示している．表中の記号の意味は以下の通りである．表中の空白はシミュレーション結果が得られていないことを示す．

- n：節点数
- ρ：枝密度
- ω：最大クリークの節点数

表1の結果をみると，比較アルゴリズムに比べて提案アルゴリズムが短い実行時間で解を出力していることが分かる．

また，表2をみると，並列処理を導入することで，実行時間の短縮が可能であることが示された．並列処理に関しては，並列度を上げると更に実行時間の改善が達成されるが，実際にFPGAで並列処理を実現する場合にはリソース制約により並列度の上限が決定される．

表3の結果では，単純に考えると並列度に応じて速度向上すると予想できる．結果より，mq_allに関しては並列処理を導入しても，並列度と同程度の速度向上しか達成されていないことが分かる．これに対して$mq\&mqr$では入力グラフにより速度向上比が5.0を越えていることが分かる．$mq\&mqr$はmqまたはmqrをそのまま並列化したものではないため，単純な比較はできないが，mqとmqrを組合せることにより互いの分枝限定の効果が高められて，速度向上比が向上したと考えられる．

4.3 ハードウェア実現

本研究では，インスタンスに特化したHDL記述をPerl言語により自動生成し，ハードウェアで解くことを目的としているが，提案アルゴリズムの評価のために，HDL記述を人手で作成し，小規模なランダムグラフに対してシミュレーション実験を行った．インスタンスベースの解法では，節点の接続関係をAND回路で実現し，それによりメモリアクセス回数を削減している．合成結果を表4に，RTLシミュレーション結果を表5に示す．

表5より，比較手法に比べて約3倍の速度向上が達成できた．更に動作周波数を向上できれば，ハードウェアを再構成する時間も含めたシステム全体として，ソフトウェアによる比較手法よりも高速に解を求めることができる．現在のHDL記述はクリティカルパスへの考慮を行っていないなどの問題点があるため，今後，高速ハードウェアの回路構成を検討する予定である．

5. あとがき

本稿では最大クリーク問題を解くインスタンスベースの解法を提案した．今後の課題としては，提案アルゴリズムのインスタンスベースのハードウェア記述を生成するプログラムの開発が挙げられる．また，インスタンスベースのハードウェア記述よりFPGA上に提案アルゴリズムを実現し，ゲートレベルシミュ

表1 ランダムグラフに対するシミュレーション
Table 1 Simulation for random graph

入力グラフ			$dfmax$	$MCLIQ$
n	ρ	ω	$[msec]$	$[msec]$
100	0.25	5	1.81	0.38
	0.50	9	4.47	2.30
	0.75	17	113.4	39.80
200	0.25	7	5.95	3.03
	0.50	10	114.50	89.00
	0.75	21	93260.00	10300.00
300	0.25	7	17.80	10.17
	0.50	12	885.00	591.67
	0.75	23	6459630.00	593000.00
400	0.25	8	42.60	41.00
	0.50	13	5041.00	3200.00
	0.75	26	125854600.00	9391000.00
500	0.25	7	95.00	119.00
	0.50	13	32150.00	15000.00

入力グラフ			mq	mqr	mq_all_5	mq_all_10
n	ρ	ω	$[msec]$	$[msec]$	$[msec]$	$[msec]$
100	0.25	5	0.01	0.04	< 0.01	< 0.01
	0.50	9	0.16	0.18	0.03	0.02
	0.75	17	16.50	5.74	3.26	1.69
200	0.25	7	0.07	0.15	0.01	< 0.01
	0.50	10	5.09	3.16	1.02	0.51
300	0.25	7	0.30	0.39	0.06	0.03
	0.50	12	31.59	25.00	6.32	3.20
400	0.25	8	0.64	0.92	0.13	0.07
	0.50	13			37.55	18.56
500	0.25	7	1.94	1.85	0.39	0.19

入力グラフ			$mq\&mqr\_5$	$mq\&mqr\_10$	$mq\&mqr\_15$
n	ρ	ω	$[msec]$	$[msec]$	$[msec]$
100	0.25	5	< 0.01	< 0.01	< 0.01
	0.50	9	0.04	0.02	0.01
	0.75	17	0.80	0.51	0.48
200	0.25	7	0.02	0.01	< 0.01
	0.50	10	0.60	0.34	0.24
	0.75	21	161.11	82.75	63.71
300	0.25	7	0.07	0.03	0.02
	0.50	12	2.94	1.77	1.32
400	0.25	8	0.14	0.07	0.05
	0.50	13	19.74	10.34	7.30
500	0.25	7	0.31	0.17	0.12
	0.50	13	55.96		

表2 並列度を変化させた場合の実行時間 [msec]
Table 2 Run time vs the number of processing units

手法	$[\#pro]$	$n300\rho025$	$n400\rho025$	$n500\rho025$
mq_all	5	0.06	0.13	0.39
	10	0.03	0.07	0.19
	15	0.02	0.05	0.13
	20	0.02	0.04	0.10
	40	0.01	0.02	0.07
	60	0.02	0.02	0.06
mq_mqr	5	0.07	0.14	0.32
	10	0.03	0.07	0.17
	15	0.02	0.05	0.12
	20	0.02	0.04	0.09
	40	0.01	0.02	0.05
	60	0.01	0.02	0.03

表3 並列処理による速度向上比
Table 3 Speed up by parallel processing

入力グラフ			mq_all_5	$mq\&mqr\_5$	
n	ρ	ω	vs mq	vs mq	vs mqr
100	0.50	10	5.33	4.00	4.50
	0.75	17	5.06	20.63	7.18
200	0.25	7	7.00	3.50	7.50
	0.50	10	4.99	8.48	1.05
300	0.25	7	5.00	4.29	5.27
	0.50	12	5.00	10.74	8.50
400	0.25	8	4.92	4.57	6.57
500	0.25	7	4.97	6.26	5.97

表4 合成結果
Table 4 Synthesis result

動作周波数	20.80MHz
使用セル数	2078

Altera APEX20KE EP20K600EFC672-1

表5 ランダムグラフに対するRTLシミュレーション
Table 5 RTL simulation for random graph

入力グラフ			$dfmax$	$MCLIQ$	mq
n	ρ	ω	$[msec]$	$[msec]$	$[msec]$
20	0.25	4	0.142	0.011	0.003
	0.50	5	0.144	0.025	0.008

レーション，および実際のFPGA上でのパフォーマンス評価を行う予定である．

文 献

[1] 関 友和，富田 悦次，“分岐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，” 電子情報通信学会技術研究報告，COMP2001-50, pp.101–108, 2001.

[2] I.M.Bomze, M.Budinich, P.M.Pardalos, and M.Pelillo, “The maximum clique problem,” Ding-Zhu Du and P.M.Pardalos, ed. , Handbook of Combinatorial Optimization, Supplement Volume A, Kluwer Academic Publishers, pp.1–74, 1999.

[3] Patric R.J.Ostergard, “A fast algorithm for the maximum clique problem,” Discrete Applied Mathematics, vol.120, pp.197–207, 2002.

[4] D.S.Johnson and M.A.Trick, “Cliques, coloring and satisfiability,” DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26, American Mathematical Society, 1996.

[5] P.Zhong, M.Martonosi, P.Ashar and S.Malik, “Solving Boolean satisfiability with dynamic hardware configurations,” Proc. International Workshop on Field Programmable Logic and Applications, pp326–335, 1998.