情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-SLDM-144 No.7
Vol.2010-EMB-16 No.7
Vol.2010-MBL-53 No.7
Vol.2010-UBI-25 No.7
2010/3/26

FPGA を用いた制約最適化問題の解法の検討

松 井 俊 浩[†1]　松 尾 啓 志[†1]

本研究では，組み合わせ最適化問題の一つである，制約最適化問題のハードウェア解法のための基礎検討を行う．FPGA を用いた探索問題のハードウェア解法は，比較的大規模な探索問題の高速解法として研究されている．このような解法は，また，自律・協調的な複数の機器から構成される分散システムにおいて，各機器が有する FPGA を用いて，自律・協調処理のために必要な組み合わせ最適化問題を解くことに適用できる可能性がある．そこで本研究では人工知能分野における基本的な問題の表現である制約最適化問題のためのハードウェア解法を検討する．深さ優先探索に基づく，基本的な解法を FPGA 上に実装し，実験により有効性を評価する．

A Study of hardware solver for Constraint Optimization Problems

TOSHIHIRO MATSUI[†1] and HIROSHI MATSUO[†1]

In this work, a hardware-solver of the constraint optimization problem is studied. Hardware-solvers that are implemented on FPGAs has been studied as effective accelerators of search problems. Such approaches may be applied to optimization problems in autonomous/cooperative system whose sub-systems have FPGAs. Therefore, we study a hardware solver of the constraint optimization problems that are basic representations in artificial intelligence. A simple solver that employs a depth-first search is implemented on FPGA. The solver is evaluated by experiments.

1. はじめに

分散処理の高度化により，自律・協調的に動作する複数の装置から構成されるシステムが提案されている．このようなシステムでは各装置が自律的・協調的な動作のための問題解決を行う必要がある．本研究では特に，そのような各装置の内部処理において，汎用的な最適化問題の表現とその解法が必要な場合を想定する．制約最適化問題[1),6),8)] は組み合わせ最適化問題の一つであり，分散協調問題解決[2)–4),9)] を含む，人工知能分野の基礎的な問題として研究されている．その解法は一般に汎用プロセッサで実行されるソフトウェアとして構成されるが，組み込み機器などに処理系を実装する場合，その計算コストは比較的大きいと考えられる．その一方で，FPGA を用いた探索問題のハードウェア解法の有効性が示されている[5),7),10)–12)]．そこで，組み込み機器が FPGA を有する場合に，その一部を探索問題の解法に利用することが期待される．本研究では，制約最適化問題のハードウェア解法の基礎検討として，基本的な深さ優先探索アルゴリズムを FPGA 上に構成し，実験により有効性を評価する．ハードウェア解法は，インスタンス固有の解法とクラス固有の解法に分類される[11)]．本研究で検討する制約最適化問題は汎用性のある問題の表現であることから，クラス固有の解法を前提とした検討が有用であると考えられる．そこで，問題を構成する，変数，制約/評価関数などを格納するデータ構造と，その情報に基づいて動作する問題解決器からなる解法を構成する．

2. 準　　備

2.1 制約最適化問題

本研究で用いる制約最適化問題を形式化する．制約最適化問題は，変数の集合 X, 二項制約の集合 C，二項関数の集合 F，により定義される．x_i は，離散有限集合である値域 D_i に含まれる変数値をとりうる．制約 $c_{i,j} \in C$ は x_i と x_j の関係を表す．制約で関係する 2 変数についての，ある割り当て $\{(x_i, d_i), (x_j, d_j)\}$ のコストは二項関数 $f_{i,j}(d_i, d_j) : D_i \times D_j \to \mathbb{N}$ により定義される．問題の大域的最適解 $\mathcal{A}$ はコスト関数値の合計 $\sum_{f_{i,j} \in F,\ \{(x_i,d_i),(x_j,d_j)\} \subseteq \mathcal{A}} f_{i,j}(d_i, d_j)$ を最小化する．このような大域的最適解を得ることが目的である．以下ではコスト関数のことを単に関数と呼ぶ．

本研究では，ハードウェア上への 実装を容易にするために，次のような制限を設ける．

(1) 変数と関数を識別するための添え字は 0 から連続する整数値として表現される．
(2) 変数の添え字は，後述する探索処理において探索木を展開するための，全順序関係に従う．
(3) 関数は次のような規則により整列され，添え字付けされる．
　(a) 関数に関係する 2 つの変数のうち，変数値の添え字の大きい方の値を，関数の整列に用いる．

†1 名古屋工業大学
Naogya Institute of Technology

Vol.2010-SLDM-144 No.7
Vol.2010-EMB-16 No.7
Vol.2010-MBL-53 No.7
Vol.2010-UBI-25 No.7
2010/3/26

(b) 上述の変数値の添え字に従って，関数を昇順に整列する．

(c) 整列された関数について，昇順に添え字を付ける．

これにより，後述する探索処理において各変数値を展開するときに評価されるべき関数の集合を，各変数ごとに，関数の添え字の開始値と終了値を用いて表現できる．

(4) 変数値は 0 から連続する整数値として表現される．従って，変数値の最大値をパラメータとして指定すれば，値域を決定できる．

2.2 探索問題のハードウェア解法

探索問題のハードウェア解法はインスタンス固有の解法とクラス固有の解法に分類される[11]．インスタンス固有の解法[7),10),12)] は，問題ごとに論理合成をおこなう必要があるが，各インスタンスに最適化された解法を構成できるため，解法の高速化の点で有利である．その一方でクラス固有の解法は，問題によって回路を再構築する必要がないが，汎用性のためにメモリが必要になるなど回路が複雑になる．

本研究で検討する制約最適化問題は，汎用的な問題の表現であることから，クラス固有の解法を前提とした検討が有用であると考えられる．そこで，変更可能な問題のパラメータを格納するデータ構造を持つ解法を構成することが重要である．ただし，現時点では，問題のパラメータを外部と入出力するインタフェースの実装は省略し，データ構造内の初期値の存在を仮定する．そのため見掛け上はインスタンス固有の解法と区別しにくい．

3. 提 案 手 法

3.1 基本的な方針

本研究では，基礎検討として，深さ優先の木探索に基づく逐次的な解法を用いる．また，2.1 節で述べたように，変数間には全順序関係が定義されているものとし，その全順序に従って探索木を展開する．提案手法で用いる解法の構成を図 1 に示す．以下では，図中に示される要素とその処理について述べる．

3.2 問題のデータ表現

本手法では，クラス固有の解法を前提とするため，問題のパラメータを変更可能なデータとして保持する必要がある．制約最適化問題は，変数と制約/評価関数により記述されるため，それらの情報を FPGA 上に構成される表に格納する．問題解決器からのアクセスの便宜を考慮し，複数の表を用いる．これらの表は図 1 の Data module に含まれる．それぞれについて以下で述べる．

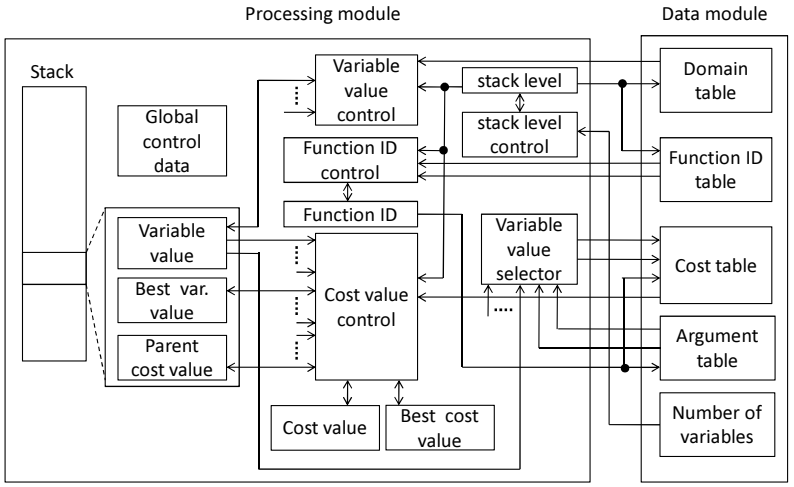

図 1 全体のブロック図

3.2.1 値 域 表

値域表は各変数の値域を格納する要素であり，図 1 では Domain table として表される．2.1 節で述べたように，変数値は 0 から連続する整数値として表現されるため，各変数について，変数値の最大値を格納すれば値域を決定できる．この表は一次元配列であり，変数の添え字を配列の添え字とし，変数値の最大値を配列の値とする．

3.2.2 関数 ID 表

関数 ID 表は，探索処理において各変数値を展開する際に評価されるべき，関数の集合の情報を格納する要素である．図 1 では Function ID table として表される．2.1 節で述べたように，各変数について，関数の添え字の開始値と終了値を用いて，上述の関数の集合を表現できる．この表は 2 つの一次元配列から構成される．それぞれ変数の添え字を配列の添え字とし，それぞれ関数の添え字の開始値，終了値を配列の値とする．

3.2.3 コスト値表

コスト値表は，各関数のコスト値を格納する表であり，図 1 では Cost table として表される．この表は，三次元配列として構成される．配列の添え字の 1 つは，各関数の添え字に対応する．配列の添え字の残り 2 つは，各関数に関係する 2 つの変数それぞれの，値域に対応する．

Vol.2010-SLDM-144 No.7
Vol.2010-EMB-16 No.7
Vol.2010-MBL-53 No.7
Vol.2010-UBI-25 No.7
2010/3/26

3.2.4 関数入力表

関数入力表は，各関数に関係する 2 つの変数の添え字を格納する表である．図 1 では Argument table として表される．この表は 2 つの一次元配列から構成される．それぞれ関数の添え字を配列の添え字とし，それぞれ変数の添え字を配列の値とする．

3.2.5 変　数　値

上記の表以外に，問題の表現に必要なパラメータは，変数の個数である．これは表とは別に与えられるものとする．図 1 では Number of variables として表される．

3.3 問題解決器の要素

本研究では，深さ優先の木探索に基づく逐次的な解法を用いる．これは，スタックを用いた再帰的な処理として構成される．ただし，回路規模の抑制のために，スタックに格納しなくても処理が可能な情報は，大域的な要素を再利用して格納する．各要素について以下で述べる．

3.3.1 スタックに関する要素

問題解決器はスタックと，その最上位を示すポインタを基礎とする．これらは図 1 では，それぞれ stack および stack level として表される．stack level が指すスタックフレームを用いて処理が実行される．各スタックフレームは，現在展開されている部分解の変数値，これまでに発見された最良解の変数値，探索処理において直前に展開された変数までに集計されたコストを格納する．それぞれ図 1 では，Variable value, Best var. value および Parent cost value として表される．

3.3.2 コスト値の集計に関する要素

各変数値を展開する際には，その変数と，展開済みの上位の部分解に含まれる変数，に関する関数のコスト値が評価され集計される．このコスト値は，直前に展開された変数までに集計されたコストと加算される．そして，次に展開される下位の変数の処理の際に用いられる．下位の変数の処理からバックトラックした後には，次の変数値が選択され，同様の処理が行われる．

各変数の値が展開される際に集計される関数は，全くない場合も，複数である場合もある．本研究では，この集計を逐次的に行う．そのために，現在，コストを加算するべき関数の添え字を格納する要素を用いる．これは図 1 では Function ID として表される．また，コストの集計値は同図中で Cost value として表される．

展開された変数値についての関数のコストの集計後は，Function ID の値は不要となる．そのため，大域的な要素を再利用することができる．

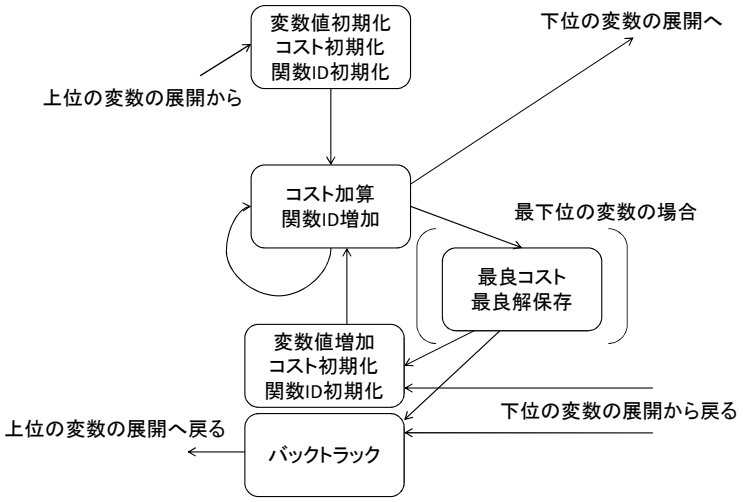

図 2　状態遷移の概要

Cost value の値も，展開された変数については不要である．その一方で，次に展開される下位の変数についてのコストの集計値と加算される必要がある．そのため，大域的な要素を再利用し，下位の変数値の展開処理へコスト値を伝搬する．

下位の変数値の処理からバックトラックした後，次の変数値についてコストを集計するためには，上位で計算されたコスト値を復元する必要がある．そこで，各スタックフレームの処理への入り口で，Cost value の値を，そのスタックフレームの Parent cost value に格納しておき，上述のコスト値の復元に用い る．

3.3.3 最良解/コストの保存に関する要素

最下位の変数値まで展開され完全解のコスト値が集計されたとき，最良のコスト値が得られていれば，その最良コスト値と最良解を保存する．最良コストは図 1 の Best cost value に格納され，最良解は各スタックフレームの Best var. value に格納される．

3.4 処理の概要

問題解決器の状態遷移の概要を図 2 に示す．上位の変数が展開され，各変数値の展開を開始するときには，変数値の初期化，コスト値の初期化，関数 ID の初期化がおこなわれる．変数値の初期化は，現在のスタックフレームの Variable value を 0 にする．コスト値の初期化では，上位で計算された Cost value の値を引き継ぐとともに，現在のスタックフレームの

Vol.2010-SLDM-144 No.7
Vol.2010-EMB-16 No.7
Vol.2010-MBL-53 No.7
Vol.2010-UBI-25 No.7
2010/3/26

Parent cost value に保存し，次の変数値についてのコストの集計の際に復元できるようにする．関数 ID の初期化では，Function ID を，Function ID table が返す，関数の添え字の開始値により，初期化する．

その後，コスト加算と関数 ID 増加が行われる．Function ID の値に対応するコスト値を Cost table から得て Cost value に加算し，Function ID を増加する．これは Function ID の値が終了値に達するまで反復される．

現在の変数値について評価されるべき関数のコスト値が全て加算された後，最下位の変数以外の場合には，次の下位の変数値の展開に遷移する．下位の変数について，同様に処理が行われて，いずれ下位の変数の展開から戻る．最下位の変数の場合は，最良コストの判定および，必要であれば最良コスト/解の保存が行われる．その後の処理は，他の変数で，下位の変数の展開から戻ったときと同様である．

下位の変数の展開から戻ったとき，変数値が最大値でない場合は，変数値の増加，コストの初期化，関数 ID の初期化がおこなわれる．この処理は，次の変数についてのコストの集計を行うための再初期化である．変数値の増加は現在のスタックフレームの Variable value を 1 加算する．コスト値の初期化では，現在のスタックフレームの Parent cost value に保存されている値を Cost value に代入する．関数 ID の初期化は，上位の変数の展開からの遷移した場合の初期化と同様である．

下位の変数の展開から戻ったとき，変数値が最大値の場合は，バックトラックする．すなわち，上位の変数の展開に戻る．

3.5 枝 刈 り

上述の探索処理に，冗長な探索を削減するための基本的な枝刈りを追加する．現在の部分解について集計されたコスト値 Cost value が，既に発見されている最良コスト Best cost value 以上であるならば枝刈りを行う．本研究では，関数値の集計中の遷移に枝刈りの条件を組み込んだ．すなわち，図 3 に示す箇所で枝刈りが行われる．実際には，枝刈りの条件で場合分けを行い，新たな状態を設けた．

3.6 簡単な back-jumping

back-jumping は木探索における枝刈りの効果を改善する手法である．現在の部分解のコストが最良コスト以上であるとき，枝刈りが行われるが，最良コストより小さいコストを得るためには，直接上位の変数よりもさらに上位の変数値の変更が必要な場合がある．このような上位の変数を特定し，その変数まで直接戻ることで冗長な処理を削減する．

ここでは実装が容易であることから，次のような簡単な back-jumping のサブセットを適用する．いま，最下位の変数値が展開されて，完全解のコストが評価され，最良コストが更新されたものとする．このとき，ある上位の変数から最下位の変数までの解の展開の過程で加算されたコストの増加が 0 であれば，その上位の変数までの部分解はもはや最適解にはなりえない．したがって，その上位の変数の一つ上の変数まで一度に戻ることができる．図 4 に概要を示す．この処理を，次のように実装した．

(1) 戻り地点を格納する要素 bj point を設ける．bj point は stack level と同じ型の値を格納する．bj point の初期値は 0 とする．

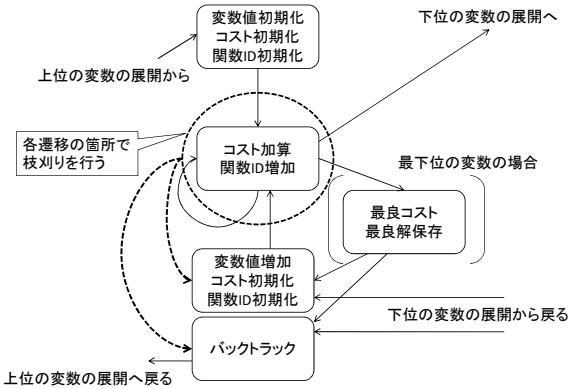

図 3 枝刈りの追加

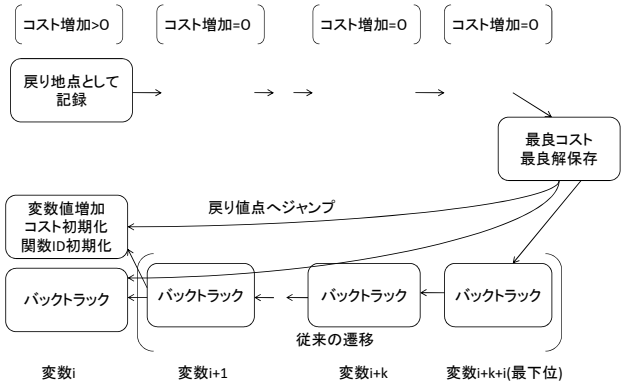

図 4 簡単な backjumping

Vol.2010-SLDM-144 No.7
Vol.2010-EMB-16 No.7
Vol.2010-MBL-53 No.7
Vol.2010-UBI-25 No.7
2010/3/26

表 1 実験環境

FPGA	EP3C120F780C7N (論理要素数 119088)
FPGA ボード	Cyclone III Deveropment Board
論理合成ソフトウェア	Quatus II 9.0 SP1 (WEB edition)
記述言語	Verilog-HDL

(2) 下位の変数の展開に遷移するとき，現在のスタックの Parent cost value と Cost value が等しくなければ bj point に stack level の値を代入する．

(3) 最下位の変数値が展開され，最良コスト/解が得られた時，bj point までスタックを戻す．

(4) 下位の変数の展開から戻った時，bj point が stack level より大きければ，bj point に stack level の値を代入する．

提案手法にこの処理を追加することは容易である．その一方で，コスト値が 0 となるような変数値が連続して展開される機会が多い問題以外では，その効果は小さいと考えられる．

4. 評　　価

FPGA 上に提案手法を実装し，実験により効果を評価した．

4.1 実 験 環 境

実験環境を表 1 に示す．FPGA に構成した回路のクロックの周波数は 50Mhz とした．また，図 1 の Data module 内の各表の要素に，外部からデータを格納するためのインタフェースの実装は現段階では省略し，各問題のパラメータを FPGA の初期値として与えた．また，各表のサイズは各問題の規模に合わせた．

4.2 問 題 設 定

本実験では，例題として，$n = 10, 20$ 個の 3 値変数と $c = l \times n$ 個の二項制約/関数からなる問題を用いた．l は制約/評価関数の密度のパラメータである．ここでは $l = 1, 2, 3$ とした．$l = 3$ の問題は比較的難しい問題である．二項関数の各変数値の組に対するコスト値は，$\frac{1}{3}$ の組のコスト値を 1 とし，その他は 0 とした．この重み付けは最大制約充足問題に類似することを意図している．次に示す手法を評価した．

- noprun: 枝刈りを行わない．
- prun: 各変数値について関数のコストを加算するときに，枝刈りを行う．
- prunbj: prun に加えて簡単な back jumping を行う．

表 2 解を得るまでのクロック数

n	c	noprun	prun	prunbj
10	10	228235	2452	2431
	20	489691	2250	2225
	30	631797	5503	5490
20	20		398164	398068
	40		195776	195687
	60		7200797	7200790

表 3 論理要素数

n	c	noprun			prun			prunbj		
		LE	reg.	mem.. bits	LE	reg.	mem.. bits.	LE	reg.	mem.. bits.
10	10	527	180	320	498	180	256	554	188	256
	20	549	180	320	522	180	256	576	188	256
	30	571	180	320	542	180	256	598	188	256
20	20				657	221	512	720	229	512
	40				705	221	512	761	229	512
	60				740	221	512	796	229	512

(LE:論理エレメント数, reg.:レジスタ数, mem.bits:メモリビット数)

論理合成を含む実験時間の制限のため，結果はそれぞれ 5 問の例題について平均した．また，実行時間が 2^{32} クロックを超えるインスタンスがある場合はそのパラメータでの実験を中止した．

4.3 結　　果

解を得るまでのクロック数を表 2 に示す．本研究では，初期の検討として簡単な枝刈りを用いているが，ある程度の効果を得ている．一方で，簡単な back jumping の効果は小さく，正確な実装の検討が必要である．

提案手法の実装では，各変数値を展開する際に，その変数と上位の変数に関する関数を逐次的に評価する．したがって，変数 x_i の値域 D_i の大きさ $|D_i|$ と，その変数値の展開の際に評価される関数の数 M_i を，全ての変数について，積算した値 $\prod_i |D_i|M_i$ が，枝刈りの無い場合の反復回数に支配的であると考えられる．3 値 10 変数の場合は，全ての解を探索しても，十分に短時間で処理が終了したが，20 変数の場合には，2^{32} クロック以内に処理が終了しなかった．

問題の規模と制約/関数の密度が増加するに従い，枝刈りがある場合の反復回数も指数関数的に増大する．従って，実際面では利用可能な問題の検討が必要である．この実験では，

Vol.2010-SLDM-144 No.7
Vol.2010-EMB-16 No.7
Vol.2010-MBL-53 No.7
Vol.2010-UBI-25 No.7
2010/3/26

20 変数 60 制約の場合の実行時間は，高々144 ms 程度であるが，有効性についての検討は別の議論が必要と考えられる．

なお，表 2 では n=20,c=20 の場合のクロック数が，n=20,c=40 の場合より多いが，これは，問題の難易度が統計的にしか決定できないために生じた，外れ値の影響である．

論理合成ソフトウェアに表示された，論理要素数を表 3 に示す．表 1 に示した FPGA の論理要素数は比較的多いこともあり，合成された回路で用いられた論理要素数はいずれの場合も全要素数の 1%に満たない．

本研究では，初期の検討としてごく基本的な，深さ優先探索の処理系を構成したが，論理要素数の点では，forward checking や back jumping による枝刈り，ヒューリスティクスを用いた探索戦略，パイプライン/並列化などの，より高度な効率化手法を実装する余地は十分にあると考えられる．

5. おわりに

本研究では，制約最適化問題の解法のハードウェア化の基礎検討として，深さ優先探索を用いる解法を FPGA 上に構成し，実験によりその有効性を評価した．より効率的な探索アルゴリズムの導入，FPGA の特徴を活用した並列処理，および，実際的な問題への適用が今後の課題である．

謝辞 謝辞 本研究の一部は財団法人堀情報科学振興財団研究助成による．

参 考 文 献

1) Fruder, E.C. and Wallace, R.J.: Partial constraint satisfaction, *Artificial Intelligence*, Vol.58, No.1, pp.21–70 (1992).
2) Mailler, R. and Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation, *3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pp.438–445 (2004).
3) Modi, P.J., Shen, W., Tambe, M. and Yokoo, M.: Adopt: Asynchronous distributed constraint optimization with quality guarantees, *Artificial Intelligence*, Vol.161, No.1-2, pp.149–180 (2005).
4) Petcu, A. and Faltings, B.: A Scalable Method for Multiagent Constraint Optimization, *9th International Joint Conferece on Artificial Intelligence*, pp.266–271 (2005).
5) Platzner, M. and Micheli, G.D.: Acceleration of Satisfiability Algorithms by Reconfigurable Hardware, *FPL '98: Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm*, London, UK, Springer-Verlag, pp.69–78 (1998).
6) Schiex, T., Fargier, H. and Verfaillie, G.: Valued Constraint Satisfaction Problems: Hard and Easy Problems, *IJCAI (1)*, pp.631–639 (1995).
7) Suyama, T., Yokoo, M., Sawada, H. and Nagoya, A.: Solving satisfiability problems using reconfigurable computing, *IEEE Trans. VLSI Syst.*, Vol.9, No.1, pp.109–116 (2001).
8) Verfaillie, G., Lemaître, M. and Schiex, T.: Russian Doll Search for Solving Constraint Optimization Problems, *AAAI/IAAI, Vol. 1*, pp.181–187 (1996).
9) Yokoo, M., Durfee, E.H., Ishida, T. and Kuwabara, K.: The Distributed Constraint Satisfaction Problem: Formalization and Algorithms, *IEEE Transactions on Knowledge and Data Engineering*, Vol.10, No.5, pp.673–685 (1998).
10) 木村義洋，若林真一，永山　忍：2 次割当問題に対するタブー探索法に基づく FPGA を用いたハードウェア解法 (FPGA とその応用及び一般)，情報処理学会研究報告，2007-SLDM-128, pp.pp.37–42 (2007).
11) 若林真一：FPGA を用いた組合せ最適化問題の高速解法，電子情報通信学会技術研究報告，CAS2006-69 (2007-01), pp.pp.43–48s (2007).
12) 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，情報処理学会研究報告，2004-SLDM-113, pp.pp.125–130 (2004).