原著論文 Original Paper

# 分散制約最適化問題へのソフトアーク整合の適用

# Applying Soft Arc Consistency to Distributed Constraint Optimization Problems

松井 俊浩 名古屋工業大学
Toshihiro Matsui Nagoya Institute of Technology
matsui.t@nitech.ac.jp

Marius C. Silaghi フロリダ工科大学
Marius C. Silaghi Florida Institute of Technology
msilaghi@fit.edu

平山 勝敏 神戸大学大学院海事科学研究科
Katsutoshi Hirayama Graduate School of Maritime Sciences, Kobe University
hirayama@maritime.kobe-u.ac.jp

横尾 真 九州大学大学院システム情報科学研究院
Makoto Yokoo Graduate School of Information Science and Electrical Engineering, Kyushu University
yokoo@is.kyushu-u.ac.jp

松尾 啓志 名古屋工業大学大学院工学研究科
Hiroshi Matsuo Graduate School of Engineering, Nagoya Institute of Technology
matsuo@nitech.ac.jp

**keywords:** distributed constraint optimization multi-agent soft-arc-consistency

**Summary**

The Distributed Constraint Optimization Problem (DCOP) is a fundamental framework of multi-agent systems. With DCOPs a multi-agent system is represented as a set of variables and a set of constraints/cost functions. Distributed task scheduling and distributed resource allocation can be formalized as DCOPs. In this paper, we propose an efficient method that applies directed soft arc consistency to a DCOP. In particular, we focus on DCOP solvers that employ pseudo-trees. A pseudo-tree is a graph structure for a constraint network that represents a partial ordering of variables. Some pseudo-tree-based search algorithms perform optimistic searches using explicit/implicit backtracking in parallel. However, for cost functions taking a wide range of cost values, such exact algorithms require many search iterations. Therefore additional improvements are necessary to reduce the number of search iterations. A previous study used a dynamic programming-based preprocessing technique that estimates the lower bound values of costs. However, there are opportunities for further improvements of efficiency. In addition, modifications of the search algorithm are necessary to use the estimated lower bounds. The proposed method applies soft arc consistency (soft AC) enforcement to DCOP. In the proposed method, directed soft AC is performed based on a pseudo-tree in a bottom up manner. Using the directed soft AC, the global lower bound value of cost functions is passed up to the root node of the pseudo-tree. It also totally reduces values of binary cost functions. As a result, the original problem is converted to an equivalent problem. The equivalent problem is efficiently solved using common search algorithms. Therefore, no major modifications are necessary in search algorithms. The performance of the proposed method is evaluated by experimentation. The results show that it is more efficient than previous methods.

## 1. ま え が き

分散制約最適化問題 (DCOP: Distributed Constraint Optimization Problem) はマルチエージェントシステムにおける協調問題解決の基本的な枠組みとして研究されている [Mailler 04, Modi 05, Petcu 05, Yeoh 08, Zivan 08]. DCOP では，マルチエージェントシステムが，変数の集合，制約の集合，および制約に関連づけられたコスト関数の集合により形式化される．各変数はエージェントの状態を表す．制約と，その制約に関連付けられた関数はエージェントの関係を表す．センサ網，会議スケジューリングなどにおける分散資源割り当て問題が DCOP として形式化されている [Farinelli 08, Junges 08, Maheswaran 04, Zhang 05]．コスト関数の値を大域的に最適化する解を得るために，分散協調探索アルゴリズムが用いられる．これらの研究の基本的な目的は，協調問題解決を最適化問題としてとらえ，その解法を普遍性のある分散アルゴリズムとして構成し，さらにその解法を応用的な問題に適用することにある．マルチエージェントシステムでは，エージェントの自律性や，エージェント固有の情報の漏えいの低減が必要である．そのため，協調問題解決手法の分散処理化は重要な課題である．

本研究では，疑似木 (pseudo-tree) を用いる厳密解法 [Modi 05] に注目する．疑似木 [Freuder 85, Schiex 99] は制約網に含まれる変数に半順序関係を与えるような，疑似的な木構造である．多くの疑似木に基づく厳密解法は，明示的または暗黙的に，最良優先戦略に基づくバックトラック探索を並行に実行する．しかし，コスト関数が広範囲の値を返すとき，これらの厳密解法は多くの反復的な探索を必要とする．この探索処理の増加は，制約の密度が比較的低い問題でも起こりうる．この問題点の主要な原因は，コストの評価値の下界値の冗長な探索にある．そのため，冗長な探索を削減するための追加的な効率化手法が必要となる．

効率化手法は前処理と，元の探索処理への追加的な処理に分類できる．効率化手法のための前処理は，その前処理を疑似木生成のための既存の前処理に容易に統合できるのであれば，合理的であるといえる．元の探索処理への追加的な処理は，探索の過程で得られる情報を活用できるため，より効果的である．しかし，元の分散探索アルゴリズムが比較的複雑である場合，その変更は容易でない．従来手法では，前処理によりコスト値の下界を推定する [Ali 05, Maheswaran 04]．この推定のために動的計画法にもとづく手法が用いられる．しかし，この手法にはさらに効率化の余地がある．また，推定された下界値を利用するために，元の探索処理の変更が必要である．

本研究では，ソフトアーク整合 (soft arc consistency) [Cooper 04, Schiex 00] を DCOP に適用する効率化手法を提案する．ソフトアーク整合はアーク整合を一般化したものであり，DCOP を含む，制約充足問題を拡張したクラスの問題に適用できる．しかし，DCOP においては，ソフトアーク整合の無限ループを回避するための追加的な手法が必要である．提案手法では，ソフトアーク整合は疑似木に従ってボトムアップに実行される．このような手法は方向付きソフトアーク整合 [Cooper 04] の一例であると考えられる．疑似木にもとづく方向付きソフトアーク整合により，大域的な下界値は根の変数についての単項関数に移動され，各コスト関数の値は削減される．その結果として，元の問題は，最適解が元の問題と等しい等価な問題に変換される．この等価な問題は一般的な探索アルゴリズムを用いて効率的に解くことができる．ソフトアーク整合により得られる単項制約の評価以外には，探索アルゴリズムへの変更は必要ない．

方向付きソフトアーク整合は，動的計画法にもとづく従来手法と類似するが，コスト関数値の下界を推定するのみでなく問題自体を変換する．この変換により，元の問題のコスト関数に含まれる冗長な情報が削減される．また方向付きソフトアーク整合は，コスト関数の境界値を用いる近似解法の前処理としても有効である．

マルチエージェントシステムの分散協調処理を構成するためには，協調問題解決の解法だけでなく，疑似木の生成や効率化のための前処理の分散処理化も重要である．

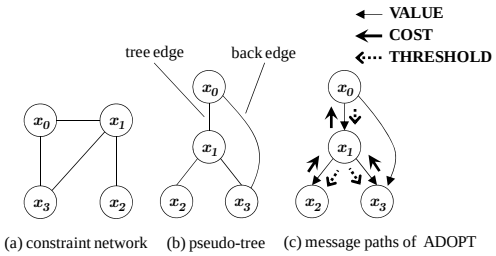



**図 1** 疑似木と，ADOPT のメッセージ配信

方向付きソフトアーク整合は簡単なアルゴリズムであるため，分散処理化や他の分散アルゴリズムとの統合は比較的容易である．

本論文の以降の構成は次のようである．2 章では本研究の背景として，DCOP，疑似木，コスト値の計算，ソフトアーク整合について述べる．次に，疑似木にもとづく方向付きソフトアーク整合について 3 章で述べる．4 章では，提案手法の有効性を実験結果により評価し，従来手法と比較する．5 章では関連研究について考察を示す．6 章で結言を述べる．

## 2. 背　　景

本研究の背景として，分散制約最適化問題，疑似木に基づくコスト値の計算，方向付きソフトアーク整合について述べる．

### 2·1 分散制約最適化問題

分散制約最適化問題は，エージェントの集合 $A$, 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．エージェント $i$ は自身の変数 $x_i$ を持つ．$x_i$ 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．$x_i$ の値はエージェント $i$ のみが決定できる．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，複数のエージェントの協調的な問題解決により得ることが目的である．

本論文ではエージェントと変数が 1 対 1 に対応することから，記述の簡略化のために必要に応じて両者を区別せずに用いる．

### 2·2 疑　似　木

疑似木 (pseudo-tree) [Freuder 85, Schiex 99] は制約網に含まれる変数に半順序関係を与えるような，グラフ上の構造である．あるグラフに対する疑似木は次のような性質をすべて満たすような「疑似的な木」である．

(1) 疑似木は元のグラフと同一の辺と頂点からなる．

(2) 疑似木は元のグラフの生成木のいずれか一つに対応する．元のグラフのすべての辺は，生成木の辺 (木辺) かそれ以外の辺 (後退辺) に分類される．

(3) 生成木の根ノードからいずれか 1 つの葉ノードへのパスに含まれる 2 ノード間にのみ，後退辺がある．異なるパスに含まれる 2 ノード間には，後退辺はない．

典型的な疑似木は，もとのグラフに対する深さ優先探索木を木辺とする疑似木である．疑似木に対応する生成木の各頂点を，疑似木においても同様に木の頂点とみなす．制約網と対応する疑似木の例を図 1(a),(b) に示す．疑似木ではサブツリーの間に後退辺は存在しない．そのため，異なるサブツリーに対する探索に分割統治法を適用できる．この性質を用いることで，探索処理を並列に実行できる．並列度の高い疑似木を生成するために，制約網に対する深さ優先探索において，次数の大きなノードを優先的に子ノードとして選択する方法が用いられる．

### 2·3 疑似木を用いるコスト値の計算

本研究では，どのように探索アルゴリズムが疑似木を用いてコスト値を計算するかが重要である．以下では具体例として，疑似木を用いる DCOP の代表的な解法である，ADOPT [Modi 05] に注目する．ADOPT では, 前処理により生成された疑似木に基づく，変数の半順序を用いる．以下では，各変数の順序関係を，疑似木の木辺に従い，親，子，祖先，子孫などの用語で表現する．ここでは祖先は親を，子孫は子をそれぞれ含む．また，ある変数と制約で直接関係する他の変数を，近傍という用語により表現する．とくに祖先である近傍を上位近傍，子孫である近傍を下位近傍と表現する．

ADOPT の処理は分枝限定法/A*アルゴリズムにもとづく非同期分散処理であり，疑似木の辺に従って次のようなメッセージを交換し，計算を行う．

- VALUE($x$,$d$): 各変数 $x$ の現在の値 $d$ を，下位近傍の変数を持つエージェントに通知する．
- COST($context$,$lb$,$ub$): ある変数を根とするサブツリーにおけるコストの合計の上下界値 $ub$,$lb$ を，その変数の親の変数を持つエージェントに通知する．上下界値の計算に用いた，祖先の変数についての現在の部分解 $context$ も同時に通知される．
- THRESHOLD($context$,$th$): COST メッセージを用いて集計されたコストにもとづき，各エージェントは自身の変数値を決定する．集計されたコストから自身の変数値に対するコストを除いた残りのコストは，子の変数を持つ各エージェントに配分される．このコストの配分 $th$ を THRESHOLD により通知する．コストの配分にもとづき，必要であれば子の変数の値も変更される．$th$ の計算に用いた現在の部分解 $context$ も同時に通知される．
- TERMINATE($context$): アルゴリズムの終了段階では，根から葉に向かって順に，最適解の変数値を決定する．このとき各エージェントは，自身の変数とその祖先の変数の変数値からなる部分解 $context$ を，子の変数を持つエージェントに TERMINATE により通知する．

各メッセージの送受信経路の例を図 1(c) に示す．これらのメッセージを用いる ADOPT の処理は次の 2 つの段階に分けられる．

- **大域的な最適コストの計算:** 各エージェントは大域的なコストの境界を疑似木に従って計算する．この処理では VALUE, COST, THRESHOLD メッセージが用いられる．この計算の結果，疑似木の根の変数を持つエージェントでは，コストの上下界値が最適値に収束する．
- **大域的な最適解の決定:** 最適コストが求められた後，疑似木に従ってトップダウンに，最適解が決定される．この処理では他の 3 つのメッセージに加えて，TERMINATE メッセージが用いられる．

上記の処理の詳細はやや複雑であるため，本論文では，特に提案手法にとって重要な最適コストの計算に注目し，VALUE メッセージ，および COST メッセージにかかわる要素について詳細を示す．また，THRESHOLD メッセージを用いる各変数値の変更の計算は，基本的には，既知のコストの最良の下界値に対応する変数値を選択するものとして，概説にとどめる．なお，ADOPT の詳細は [Modi 05] に示される．

エージェント $i$ は次の情報を用いてコストの情報を計算する．

- $x_i$: エージェント $i$ の変数. 変数 $x_i$ の値 $d_i$ は，制約で直接関連する $x_i$ の下位近傍の変数を持つエージェントへ，VALUE メッセージを用いて送信される．
- $current\_context_i$: $x_i$ の祖先の変数についての現在の部分解. VALUE メッセージに含まれる変数値により，$current\_context_i$ が更新される．これは上位近傍の現在の変数値を保持するために必要である．また，$x_i$ を根とするサブツリー全体のコストの集計のために，$x_i$ の子孫の変数と制約で関係する，$x_i$ の祖先の変数値も保持する必要がある．これらの変数値は，$x_i$ の子の変数を持つエージェントからの COST に含まれる $context$ にもとづいて，更新される．
- $context_i(x,d), lb_i(x,d), ub_i(x,d)$: 変数 $x_i$ の値 $d$ と，$x_i$ の子の変数 $x$ を根とするサブツリーに対する，最適コストの上下界の情報．これらは,$x_i$ の子の変数 $x$ を持つエージェントから，COST メッセージにより受信する．もしも $current\_context_i$ が $context_i(x,d)$ を含むとき，コストの上界は $ub_i(x,d)$，下界は $lb_i(x,d)$ である．これらの情報は $context_i(x,d)$ が $current\_context_i$ に含まれるかぎり保持される．もしも $current\_context_i$ が $context_i(x,d)$ と矛盾するとき，$context_i(x,d)$，$lb_i(x,d)$ および $ub_i(x,d)$ はそれぞれ，$\{\}$，0 および $\infty$ に初期化される．

エージェント $i$ におけるコストの計算は次のように示される．変数 $x_i$ の値 $d$ および $current\_context_i$ に対する，局所コスト $\delta_i(d)$ は次式のように定義される．

$$\delta_i(d) = \sum_{\substack{j \text{ s.t. } (x_j,d_j)\in current\_context_i,\\ x_j \in upper\ neighborhood\ of\ x_i}} f_{i,j}(d,d_j) \quad (1)$$

変数 $x_i$ の値 $d$ および $x_i$ を根とするサブツリーに対する，コストの上界 $UB_i(d)$ と下界 $LB_i(d)$ は次のように定義される．

$$LB_i(d) = \delta_i(d) + \sum_{j \text{ s.t. } x_j \in children\ of\ x_i} lb_i(x_j,d) \quad (2)$$

$$UB_i(d) = \delta_i(d) + \sum_{j \text{ s.t. } x_j \in children\ of\ x_i} ub_i(x_j,d) \quad (3)$$

$x_i$ を根とするサブツリーに対する上界 $UB_i$ と下界 $LB_i$ は次のように定義される．

$$LB_i = \min_{d\in D_i} LB_i(d) \quad (4)$$

$$UB_i = \min_{d\in D_i} UB_i(d) \quad (5)$$

初期の状態において，$LB_i$ は 0 であり，$UB_i$ は $\infty$ である．

各エージェント $i$ は VALUE($x_i, d_i$) メッセージを下位近傍の変数を持つエージェントに，COST( $current\_context_i$, $LB_i, UB_i$) メッセージを親の変数を持つエージェントに，それぞれ送信する．これらのメッセージ交換にもとづき，$current\_context_i$, $context_i(x,d)$, $lb_i(x,d)$, $ub_i(x,d)$ を更新する．

また，各エージェント $i$ は，*backtracking threshold* と呼ばれる値 $thr_i$ を管理する．$thr_i$ は親のエージェントから受信した THRESHOLD メッセージに含まれる $th$ により，$x_i$ を根とするサブツリーに現在割り当てられたコスト値を表す．ただし，非同期性による矛盾を解決するために，THRESHOLD メッセージに含まれる $context$ と $current\_context_i$ の割り当てが整合する場合にのみ，$th$ の値が $thr_i$ に代入される．さらに，$thr_i$ は $LB_i \le thr_i \le UB_i$ を満足するように制限される．現在の変数値 $d_i$ が $thr_i \le LB_i(d_i)$ を満たさないとき，別の変数値が選択され VALUE メッセージにより通知される．これによりバックトラックが行われる．このような変数値の変更は最良優先探索戦略に基づいている．

根の変数を持つエージェント $r$ では，コストの境界が，$LB_r = UB_r$ なる大域的最適コストに収束する．大域的最適解はこの最適コストにしたがって決定される．

図 1(b) の例において，各変数 $x_i$ の値域 $D_i = \{a, b\}$，コスト関数 $f_{0,1}, f_{0,3}, f_{1,2}$ について $f_{i,j}(d_i,d_j) = \{1 \text{ if } d_i = d_j, 0 \text{ otherwise}\}$，また，$f_{1,3}(d_1,d_3) = \{2 \text{ if } d_1 = a, 1 \text{ otherwise}\}$ である場合の ADOPT の実行例を表 1 に示す．初期解は $\{(x_0,a),(x_1,b),(x_2,a),(x_3,b)\}$ である．また，初期状態では，コストの下界値は 0，上界値は $\infty$ とする．$x_0, x_1$ を持つエージェントは，制約で直接関係する子孫のエージェントに VALUE メッセージを送信し，実行が開始される．$step = 0$ では，VALUE メッセージ受信により，各 $current\_context_i$ が更新される．葉ノードの変数 $x_2, x_3$ のエージェントでは，上位の変数の部分解に対するコストの上下界値 $LB_i, UB_j$ が計算されている．$x_3$ では，$LB_3 \le thr_3$ を満たすように，$thr_3 = 1$ となっている．その一方で，$x_0, x_1$ のエージェントでは，子ノードからのコスト値が得られていないため，子ノード以下のサブツリーについては既定の上下界値 $0, \infty$ を用いて，$LB_i, UB_j$ が計算されている．各エージェント $i$ は自身を根とするサブツリーのコスト値 $LB_i, UB_i$ を COST メッセージより，親のエージェントに伝達する．また，コスト値 $thr_i - \delta_i(d_i)$ の値を各子ノードに配分する．コスト値の配分 $th$ は，THRESHOLD メッセージにより各子ノードに伝達される．

**表 1** ADOPT の実行例

| $step$ | $x_i$ | $d_i$ | $current\_context_i$ | $LB_i$ (a) (b) | $UB_i$ (a) (b) | $thr_i$ |
|---|---|---|---|---|---|---|
| init. | $x_0$ | $a$ | { } | 0 0 0 | ∞ ∞ ∞ | 0 |
| | $x_1$ | $b$ | { } | 0 0 0 | ∞ ∞ ∞ | 0 |
| | $x_2$ | $a$ | { } | 0 0 0 | ∞ ∞ ∞ | 0 |
| | $x_3$ | $b$ | { } | 0 0 0 | ∞ ∞ ∞ | 0 |
| VALUE | $x_0 \to x_1,x_3$ | | ($x_0$,$a$) | | | |
| VALUE | $x_1 \to x_2,x_3$ | | ($x_1$,$b$) | | | |
| 0 | $x_0$ | $a$ | { } | 0 0 0 | ∞ ∞ ∞ | 0 |
| | $x_1$ | $b$ | {($x_0$,$a$)} | 0 1 0 | ∞ ∞ ∞ | 0 |
| | $x_2$ | $a$ | {($x_1$,$b$)} | 0 0 1 | 0 0 1 | 0 |
| | $x_3$ | $b$ | {($x_0$,$a$),($x_1$,$b$)} | 1 2 1 | 1 2 1 | 1 |
| COST | $x_1 \to x_0$ | | $context$={($x_0$,$a$)}, $lb$=0, $ub$=∞ | | | |
| THR. | $x_1 \to x_2$ | | $context$={($x_0$,$a$)}, $th$=0 | | | |
| THR. | $x_1 \to x_3$ | | $context$={($x_0$,$a$)}, $th$=0 | | | |
| COST | $x_2 \to x_1$ | | $context$={($x_1$,$b$)}, $lb$=0, $ub$=0 | | | |
| COST | $x_3 \to x_1$ | | $context$={($x_0$,$a$),($x_1$,$b$)}, $lb$=1, $ub$=1 | | | |
| 1 | $x_0$ | $a$ | { } | 0 0 0 | ∞ ∞ ∞ | 0 |
| | $x_1$ | $b$ | {($x_0$,$a$)} | 1 1 1 | 1 ∞ 1 | 1 |
| | $x_2$ | $a$ | {($x_1$,$b$)} | 0 0 1 | 0 0 1 | 0 |
| | $x_3$ | $b$ | {($x_0$,$a$),($x_1$,$b$)} | 1 2 1 | 1 2 1 | 1 |
| THR. | $x_0 \to x_1$ | | $context$={ }, $th$=0 | | | |
| COST | $x_1 \to x_0$ | | $context$={($x_0$,$a$)}, $lb$=1, $ub$=1 | | | |
| THR. | $x_1 \to x_2$ | | $context$={($x_0$,$a$)}, $th$=0 | | | |
| THR. | $x_1 \to x_3$ | | $context$={($x_0$,$a$)}, $th$=1 | | | |
| COST | $x_2 \to x_1$ | | $context$={($x_1$,$b$)}, $lb$=0, $ub$=0 | | | |
| COST | $x_3 \to x_1$ | | $context$={($x_0$,$a$),($x_1$,$b$)}, $lb$=1, $ub$=1 | | | |
| 2 | $x_0$ | $b$ | { } | 0 1 0 | 1 1 ∞ | 0 |
| | $x_1$ | $b$ | {($x_0$,$a$)} | 1 1 1 | 1 ∞ 1 | 1 |
| | $x_2$ | $a$ | {($x_1$,$b$)} | 0 0 1 | 0 0 1 | 0 |
| | $x_3$ | $b$ | {($x_0$,$a$),($x_1$,$b$)} | 1 2 1 | 1 2 1 | 1 |
| VALUE | $x_0 \to x_1,x_3$ | | ($x_0$,$b$) | | | |
| THR. | $x_0 \to x_1$ | | $context$={ }, $th$=0 | | | |
| COST | $x_1 \to x_0$ | | $context$={($x_0$,$a$)}, $lb$=1, $ub$=1 | | | |
| THR. | $x_1 \to x_2$ | | $context$={($x_0$,$a$)}, $th$=0 | | | |
| THR. | $x_1 \to x_3$ | | $context$={($x_0$,$a$)}, $th$=1 | | | |
| COST | $x_2 \to x_1$ | | $context$={($x_1$,$b$)}, $lb$=0, $ub$=0 | | | |
| COST | $x_3 \to x_1$ | | $context$={($x_0$,$a$),($x_1$,$b$)}, $lb$=1, $ub$=1 | | | |
| 3 | $x_0$ | $b$ | { } | 0 1 0 | 1 1 ∞ | 0 |
| | $x_1$ | $a$ | {($x_0$,$b$)} | 0 0 1 | ∞ ∞ ∞ | 0 |
| | $x_2$ | $a$ | {($x_1$,$b$)} | 0 0 1 | 0 0 1 | 0 |
| | $x_3$ | $a$ | {($x_0$,$b$),($x_1$,$b$)} | 1 1 2 | 1 1 2 | 1 |
| | … | | | | | |
| 6 | $x_0$ | $a$ | { } | 1 1 2 | 1 1 2 | 1 |
| | $x_1$ | $a$ | {($x_0$,$b$)} | 2 2 2 | 2 2 2 | 2 |
| | $x_2$ | $b$ | {($x_1$,$a$)} | 0 1 0 | 0 1 0 | 0 |
| | $x_3$ | $a$ | {($x_0$,$b$),($x_1$,$a$)} | 2 2 2 | 2 2 2 | 2 |
| VALUE | $x_0 \to x_1,x_3$ | | ($x_0$,$a$) | | | |
| THR. | $x_0 \to x_1$ | | $context$={ }, $th$=1 | | | |
| TERM. | $x_0 \to x_1$ | | $context$={(0,0)} | | | |
| | … | | | | | |
| | … | | | | | |
| opt. | $x_0$ | $a$ | { } | 1 1 2 | 1 1 2 | 1 |
| | $x_1$ | $b$ | {($x_0$,$a$)} | 1 2 1 | 1 2 1 | 1 |
| | $x_2$ | $a$ | {($x_0$,$a$),($x_1$,$b$)} | 0 0 1 | 0 0 1 | 0 |
| | $x_3$ | $b$ | {($x_0$,$a$),($x_1$,$b$)} | 1 2 1 | 1 2 1 | 1 |

$step = 2$ では，$x_0$ の変数値 $d_0$ が，$thr_0 \leq LB_i(d_0)$ を満たすように，$step = 1$ における $a$ から $b$ に変更される．

$step = 3$ では，$x_1$ のエージェントで $LB_1 = 0, UB_1 = \infty$ となる．これは，$x_0$ において $d_0$ が $b$ に変更され，$current\_context_1 = \{(x_0, b)\}$ となり，子の変数 $x_3$ についての $context_1(x_3, b) = \{(x_0, a), (x_1, b)\}$ と矛盾することから，$x_3$ 以下のサブツリーのコスト値 $lb_1(x_3, b), ub_1(x_3, b)$ が $0, \infty$ にリセットされたことによる．また，親ノードからの COST メッセージにより $thr_1 = 0$ となっている．この影響で，$d_1$ の値が $b$ から $a$ に変更される．

$step = 6$ では $x_0$ のエージェントで，$LB_0 = UB_0 = 1$ となる．これが大域的最適コスト値である．$x_0$ のエージェントは $UB_0(a) = 1$ である変数値 $a$ を最適解として決定し，TERMINATE メッセージにより $x_1$ のエージェントに終了を伝達する．以降では，$x_0 = a$ のもとで $x_1$ 以下のエージェントが同様の計算を行う．結局，大域的最適解として，$\{(x_0, a), (x_1, b), (x_2, a), (x_3, b)\}$ が得られる．

上述のような最良優先探索戦略が有効な問題では，$LB_i$ と $UB_i$ はそれぞれ真の上下界値にすみやかに到達する．その一方で，各コスト関数の評価値が広範囲の値を持つ問題では，完全解に対するコストの下界値が全て集計される前に，部分解の下界値のみに基づいてバックトラックが行われる機会が増加する．そのため，各エージェントは同一の部分解を繰り返し探索しながら，その下界値を徐々に改善してゆく．これは，探索処理の反復回数を増加する．

探索処理の反復回数を削減するために，前処理で得た下界の推定値を用いることができる．これは基本的には次のような方針にもとづく．まず，前処理により，各変数値について最良のコストの下界値を推定しておく．探索処理では，その下界の推定値をコストの下限値として用いる．これにより，元の問題をそのまま解く場合よりも，冗長なバックトラックが削減される．従来手法では，部分的に動的計画法を適用する前処理によって計算された，下界の推定値を用いている [Ali 05, Maheswaran 04]．提案手法では，近年提案されているソフトアーク整合アルゴリズムを前処理として用いる．

### 2·4 ソフトアーク整合

ソフトアーク整合 (soft arc consistency) [Cooper 04, Schiex 00] はアーク整合を一般化した手法である．アーク整合は，探索空間を削減するために元の制約から変数値および変数値の組を禁止する新たな制約を導出する手法であり，制約充足問題において前処理または解法の一部として用いられる．従来のアーク整合手法は，解の評価値が真偽値であることに基づいている．これに対しソフトアーク整合は，DCOP を含む拡張されたクラスの制約充足問題に適用できる．ソフトアーク整合を適用することにより，元の問題は，より効率的に解くことができる等価な問題に変換される．二項関数の集合 $F$ に加えて，

```
1  Projection(f, i, d) begin
2    m ← 0; v ← ∞;
3    foreach t ∈ f s.t. t_↓{i} = (d) do v ← min(v, f( t ));
4    if v affects f_i(d) then begin
5      f_i(d) ← f_i(d) + v; m ← 1;
6      foreach t ∈ f s.t. t_↓{i} = (d) do f(t) ← f(t) − v;
7    end;
8    return m;
9  end.

11 Extension(i, d, f) begin
12   m ← 0; v ← f_i(d);
13   if v is ∞ then begin          // remove for directed soft AC
14     foreach t ∈ f s.t. t_↓{i} = (d) do
15       if v affects f(t) then begin
16         f(t) ← f(t) + v; m ← 1;
17       end;
18     f_i(d) ← f_i(d) − v;
19   end;
20   return m;
21 end.

23 SAC(X, D, F^1 ∪ F) begin
24   m ← 1;
25   while (m) do begin
26     m_p ← m_e ← 0;
27     foreach f ∈ F do
28       foreach i ∈ X_f do
29         foreach d ∈ D_i do m_p ← m_p ∨ Projection(f, i, d);
30     foreach i ∈ X do
31       foreach f ∈ F s.t. i ∈ X_f do
32         foreach d ∈ D_i do m_e ← m_e ∨ Extension(i, d, f);
33     m ← m_p ∨ m_e;
34   end;
35 end.
```

**図 2** ソフトアーク整合アルゴリズム

ソフトアーク整合では単項関数の集合 $F^1$ が用いられる．変数 $x_i$ についての単項関数は $f_i \in F^1$ により表される．また，次の表記が用いられる．$t \in f$ は二項関数 $f$ の変数値の割り当ての組を表す．$t_{\downarrow\{i\}} = (d)$ は組 $t$ が割り当て $(x_i, d)$ を含むことを表す．$X_f \subseteq X$ は関数 $f$ に関係する変数の集合を表す．

ソフトアーク整合の処理を図 2 に示す*1．ソフトアーク整合では，射影 (projection) と拡張 (extension) の 2 つの処理が交互に反復して制約網に適用される．射影は，割り当て $(x_i, d)$ についての二項関数 $f$ のコスト値の下界にもとづいて，コスト値を $f$ から単項関数 $f_i$ に移すような操作である．拡張は，割り当て $(x_i, d)$ についての単項関数 $f_i$ のコスト値の下界にもとづいて，$f_i$ のコスト値を二項関数 $f$ に移すような操作である．このアルゴリズムでは，等価な問題が収束するまで処理が反復される．一般の制約最適化問題の場合は，射影と拡張が逆の処理となるため，ソフトアーク整合は収束しない場合がある．図 2 の 13 行目では，単項関数のコスト値が∞すなわち吸収元であるときに拡張が適用される．このように拡張の適用を制限することで無限に反復処理を行うことが回避さ

*1 元の疑似コードは文献 [Schiex 00] に示される．本論文では便宜のために表記を変更している．

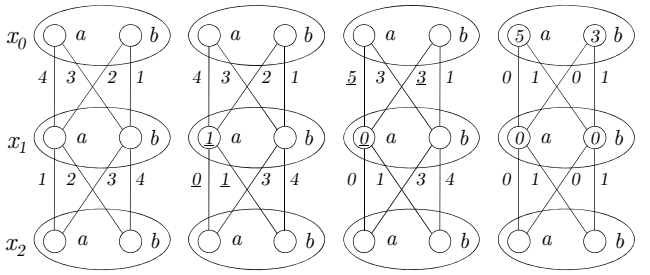



**図 3** 方向付きソフトアーク整合

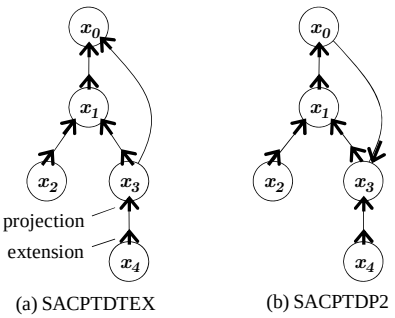



**図 4** 疑似木に基づく方向付きソフトアーク整合

れる．しかし，コスト関数の値が吸収元以外であるとき，拡張は全く適用されない．そのため，拡張は吸収元以外のコスト値についても適用するかわりに，収束のための別の追加的な方法を用いる必要がある．

### 2·5 方向付きソフトアーク整合

方向付きソフトアーク整合 [Cooper 04] は，制約網において定義される，ある方向に基づいて，ソフトアーク整合を順番に適用する．処理が方向性を持つことにより，拡張を吸収元以外のコスト値について適用しても，ソフトアーク整合は収束する．

方向付きソフトアーク整合の例を図 3 に示す. (a) は元の問題を示す．この問題は 3 個の変数と 2 つの二項制約/関数からなる．各変数は値域 $\{a,b\}$ に含まれるいずれかの値をとる．各辺のラベルは変数値の割り当ての組に対応するコスト値を表す．また，各単項関数の初期値はすべて 0 とする．図中では初期値を持つ単項関数は省略する．この例では，方向付きソフトアーク整合は $x_2$ から $x_0$ に向かって適用される．(b) はコスト値を $f_{1,2}$ から $f_1$ に移す射影を表す．(c) はコスト値を $f_1$ から $f_{0,1}$ に移す拡張を表す．図中では変更されたコスト値に下線を引いている．同様に，すべての関数のすべての変数値の割り当ての組について射影と拡張が適用される．(d) は結果として得られる等価な問題を示す．大域的な下界値は上位の変数の単項関数に移されてゆく．結局は大域的な下界値は $f_0$ に集計され，他の単項関数の値は 0 となる．さらに，大域的な下界値 3 を $f_0$ から分離することもできるが，本論文では，(d) に示す等価な問題の表現を用いる．

## 3. 疑似木における方向付きソフトアーク整合

本研究では，疑似木に基づく方向付きソフトアーク整合を DCOP に適用するような，効率化手法を提案する．以下では提案手法について述べる．

### 3·1 方向付きソフトアーク整合の適用

2·3 節に示したように，ADOPT は疑似木に基づいてボトムアップにコスト値を計算する．そのため，方向付きソフトアーク整合を疑似木に基づいて同様の順序で適用することは合理的であると考えられる．図 4 (a) は疑似木に基づく方向付きソフトアーク整合の例を示す．方向付きソフトアーク整合は，葉の変数から根の変数に向かって適用される．射影と拡張の手続きは，吸収元以外のコストについても拡張が適用されることを除いて，図 2 と同様である．各変数を持つエージェントは次の 2 つの段階の処理を行う．

(1) 下位の近傍の変数に対する二項関数それぞれから，自身の単項関数への射影
(2) 自身の単項関数から，親の変数に対する二項関数への拡張

拡張の段階の処理では，自身の単項関数の全体のコストが，上位側の木辺に対応する二項関数に移される．拡張は上位の後退辺に対応する二項関数に対しては行われない．これにより，コストが重複して合計されることが回避される．葉の変数を持つエージェントでは射影は行われない．ただし，葉の変数について初期の単項関数があれば，拡張を適用できる．葉以外の変数を持つエージェントはすべての子からの射影の後で，親への拡張を行う．

後退辺に対応する二項関数からの射影は，上位側の変数の単項関数に適用される．これは，2·3 節で示したコスト値の計算における，後退辺に対応する二項関数の値を合計する際の方向とは，逆である．ADOPT では，後退辺に対応する関数についてのコストの評価は，下位側の変数を持つエージェントが行う．そして，評価されたコスト値を含むコストの合計値は，木辺を経由して祖先の変数を持つエージェントに送信される．たとえば，図 4(a) または (b) の疑似木の場合，ADOPT では $f_{0,3}$ に対するコストは $x_3$ を持つエージェントが評価する．この評価値は $x_1$ と $x_0$ を持つエージェントがコスト値を計算する際にも必要である．それに対し，ソフトアーク整合の方向性は元々は制限されていない．それゆえ，別の方法でソフトアーク整合を適用することもできる．たとえば，次のような方法も可能である．(1) 後退辺の二項関数のコスト値を下位側の変数の単項関数に移す．(2) そして，そのコスト値を下位側の変数の単項関数から上位の木辺の二項関数に移す．これは ADOPT におけるコストの計算

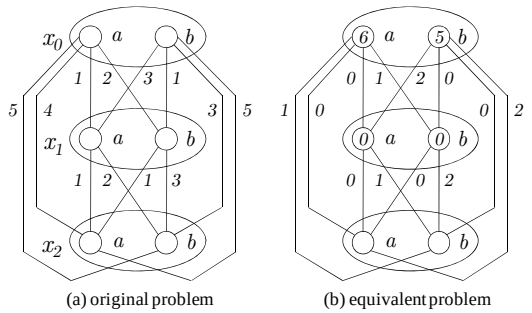



**図 5** 後退辺がある場合の方向付きソフトアーク整合

に類似する．このような例を図 4 (b) に示す．さらに，射影と拡張において，コストの一部のみを移動することもできる．例えば，ある変数についての単項関数のコストを，親以外の祖先も含むすべての上位の変数との二項関数それぞれに均等に分割するように，拡張を適用できる．

疑似木に後退辺が無い場合は，等価な問題は直接的に大域的最適解を与える．根のエージェント $r$ では，最適な割り当ては $(x_r, \mathrm{argmin}_d f_r(d))$ である．大域的な最適コストは $\min_d f_r(d)$ である．根以外のエージェント $i$ では，親のエージェント $j$ の最適な割り当てが $(x_j, d_j)$ であるとき，$i$ の最適な割り当ては $f_{i,j}(d_i, d_j) = 0$ のような $d_i$ である．

疑似木に後退辺がある問題では，ソフトアーク整合の結果のみにもとづく貪欲的な解法では，大域的な最適解が得られないことがある．このような例を図 5 に示す．図 5(a) では，変数の組 $(x_0, x_2)$ に関するコスト関数が後退辺となっている．同図 (b) は (a) にソフトアーク整合を適用して得られる等価な問題である．等価な問題に対して，根ノードではコストが最小値 5 である部分解 $\{(x_0, b)\}$ が選択される．変数値の割り当て $(x_0, b)$ を含む解の各変数値を頂点とする，いずれの部分グラフにもコスト 2 の辺が含まれる．そのため，$(x_0, b)$ を含む解のコストは 7 未満にはならない．その一方で，解 $\{(x_0, a), (x_1, a), (x_2, a)\}$ のコストは 6 であり，これが最適解である．このような問題について，大域的最適解を探索するために DCOP の解法が必要である．

ソフトアーク整合により得られる等価な問題は，ADOPT を含む一般的な DCOP の解法を用いて解くことができる．等価な問題は単項のコスト関数を含むが，多くの DCOP の解法では，単項関数の評価の処理を導入することは容易である．実際には，疑似木にもとづく方向付きソフトアーク整合では，大域的な下界値は疑似木の根に集計される．そのため，吸収元のコストが無ければ，根のエージェントのみが非零の単項関数を持つ．

疑似木に基づく方向付きソフトアーク整合の構成例を図 6 に示す．このアルゴリズムは分散深さ優先探索 [Awerbuch 85] にソフトアーク整合の処理を加えている．基本的には，このアルゴリズムは逐次型の深さ優先探索を模倣する．この処理では, DISCOVER(次のノードの展開), RETURN(バックトラック), VISITED(展開済みノードの通知) および ACK(応答) の 4 つのメッセージが用いられる．結果として，制約網に対する疑似木が生成される．さらに，元のアルゴリズムにソフトアーク整合の処理を埋め込むための変更がなされている．ソフトアーク整合はバックトラックの処理の段階 (17-19, 25, 29-34 行) で行われる．射影と拡張は 2 つのエージェントにまたがって実行されるため，双方のエージェントが情報を共有する．疑似コードの $f^i_{j,k}$ は $i$ の $f_{j,k}$ の複製を表す．また，辺の下位側のエージェントは，ダミーの単項関数を用いて上位側のエージェントの単項関数への射影の一部を模倣している (19 および 34 行)．RETURN メッセージはソフトアーク整合の情報を扱うために拡張されている．

### 3·2 前処理の計算量

ソフトアーク整合では，射影と拡張はそれぞれ，二項関数の変数値の組と，単項関数の変数値について，適用される．これらの計算量は，二項関数の変数値の組の数について線形である．提案手法は射影と拡張を疑似木に基づいて適用する．射影は，各辺に対応する二項関数と，その辺の上位側の変数に対応する単項関数について，適用される．拡張は，各辺に対応する二項関数と，その辺の下位側の変数に対応する単項関数について適用される．したがって，全体としての計算量は，すべての二項関数の変数値の組の数について線形である．ソフトアーク整合で必要となる単項関数についての記憶の大きさは，すべての変数の変数値の数について線形である．これらの処理のオーバヘッドは，ADOPT の反復的な探索処理と比較して十分に少ない．

### 3·3 前処理の正当性

ソフトアーク整合によって得られる等価な問題では，完全解に対する各コスト値は元の問題と等しい．それゆえ，大域的最適解は元の問題と同一である．ソフトアーク整合により生成される単項制約の評価以外には，等価な問題を解くために探索アルゴリズムを変更する必要は無い．そのため ADOPT および他の多くのアルゴリズムの正当性とアルゴリズムの停止には影響がない．

### 3·4 誤差の境界の併用

ソフトアーク整合により元の問題の冗長な探索を削減することができたとしても，多くの変数と密な制約からなる難しい問題では，厳密解法は多くの反復処理を必要とする．そのような場合，誤差の境界を用いる近似解法 [Modi 05] が有効である．誤差の境界により解の品質を保証しつつ探索の反復処理を削減できる．基本的には，近似解法は，根のエージェントにおいて既知のコスト値の上界と下界の差が，パラメータで指定された値に到達したとき

**表 2** DP0, DP1 および DP2 のコスト値の計算

| | |
|---|---|
| DP0 | $h_i(d_i) := \sum_{j \text{ s.t. } x_j \in \text{ children of } x_i} \sum_{k \text{ s.t. } x_k \in \text{ upper neighborhoods of } x_j} \min_{d_j \in D_j} \min_{d_k \in D_k} f_{j,k}(d_j, d_k)$ |
| DP1 | $h_i(d_i) := \sum_{j \text{ s.t. } x_j \in \text{ children of } x_i} \min_{d_j \in D_j} (h_j(d_j) + f_{i,j}(d_i, d_j))$ |
| DP2 | $h_i(d_i) := \sum_{j \text{ s.t. } x_j \in \text{ children of } x_i} \min_{d_j \in D_j} (h_j(d_j) + f_{i,j}(d_i, d_j) + \sum_{k \text{ s.t. } x_k \in \text{ upper neighborhoods of } x_j \setminus \{x_i\}} \min_{d_k \in D_k} f_{j,k}(d_j, d_k))$ |

```
1  // initialize node i
2  let N_i denote neighborhood nodes of i;
3  p_i ← φ;                    // parent node of i
4  L_i ← {};                   // descendant nodes of i
5  U_i ← N_i;                  // unvisited neighborhood nodes of i
6  foreach j ∈ N_i do m_{i,j} ← 0;    // flags
7  if i is the initiator node then send loop-back DISCOVER message to
       i;
8  process messages as follows;

10 // arrive from i's parent node j
11 for DISCOVER message from j do begin
12   p_i ← j;                  // p_i ← i if i is the initiator node
13   foreach k ∈ N_i\{j} do begin   // notify that i has been visited
14     send VISITED message to k; m_{i,k} ← 1;
15   end;
16   if N_i = {j} ∧ j ≠ i then begin   // j is the only one neighbor of i
17     foreach d ∈ D_i do Extension(i,d,f^i_{j,i});
18     send RETURN({i}, f^i_{j,i}) to j;          // backtracking
19     foreach d ∈ D_j do Projection(f^i_{j,i},j,d); // use dummy f_j
20   end;
21 end;

23 // return from i's child node j/resume depth first search
24 for RETURN(L,f) message from j do begin
25   if j ≠ i then begin L_i ← L_i ∪ L; replace f^i_{i,j} by f; end;
26   if there exists k ∈ U_i then begin   // visit next node
27     send DISCOVER to the most prior k; U_i ← U_i\{k};
28   end else begin
29     foreach l ∈ N_i ∩ L_i do foreach d ∈ D_i do Projection(f^i_{i,l},i,d);
30     if p_i ≠ i then begin          // return to parent node
31       foreach d ∈ D_i do Extension(i,d,f^i_{p_i,i});
32       send RETURN(L_i ∪ {i},f^i_{p_i,i}) to p_i;   // backtracking
33       foreach h s.t. h ∈ N_i ∧ h ∉ L_i do
34         foreach d ∈ D_h do Projection(f^i_{h,i},h,d);  // use dummy f_h
35     end else the algorithm has terminated;
36   end;
37 end;

39 // remove i's neighborhood node j from unvisited nodes
40 for VISITED message from j do begin
41   U_i ← U_i\{j}; send ACK message to j;
42 end;

44 // receive acknowledge of VISITED message
45 for ACK message from j do begin
46   m_{i,j} ← 0;
47   if m_{i,k} = 0 for all k ∈ N_i then
48     send loop-back RETURN({},{}) to i;  // resume depth first
           search
49 end;
```

**図 6** 疑似木に基づく方向付きソフトアーク整合アルゴリズム

に，終了する[*2]．疑似木にもとづく方向付きソフトアーク整合は，上記のような近似解法においても有効である．根のエージェントにおける，大域的なコストの下界値は，探索処理においてただちに下界値を制限する．また，各二項関数のコスト値が全体的に削減されることにより，コスト値の上界と下界が許容される誤差の境界値に収束するまでの反復回数が削減される．

## 4. 評　　価

提案手法を実験結果により評価した．提案手法と従来手法の効果の比較，および提案手法の効果についての解析と考察を示す．

### 4·1 実 験 の 設 定

本実験では，DCOP の例題として，$n$ 個の 3 値変数と $ln$ 個の二項制約/関数からなる問題を用いた．これは従来手法の実験で用いられたものと同様である．$l$ は二項制約/関数の密度を決定するパラメータである．二項関数の各変数値の組に対するコスト値として，1 から 100 の整数値を一様分布の乱数により選択した．結果は 50 問の例題について平均した．前処理による効率化手法および ADOPT を各問題に適用した．比較した手法を次に示す．

- ORIG: 前処理を用いない元の問題．
- DP0, DP1 および DP2: 動的計画法に基づく従来手法．
- ソフトアーク整合に基づく手法．
  - SACND: 方向なし，射影のみを適用．
  - SACPTP: 方向付き，射影のみを適用．
  - SACPTDP2: 方向付き，DP2 と同様の方向を使用．
  - SACPTDTEX: 方向付き，提案手法．

DP0，DP1 および DP2 は従来手法 [Ali 05] で提案された動的計画法に基づく手法である．これらは，各変数 $x_i$ の変数値 $d_i$ について，コストの下界の推定値を疑似木にしたがってボトムアップに計算する．それぞれの下界の推定値の計算の方法を表 2 に示す．これらの計算は比較的簡単であるが，下界の推定値を用いるために，ADOPT のアルゴリズムの変更が必要となる．本実験では次のように ADOPT を変更した．

- backtracking threshold: 元の ADOPT では，エージェント $i$ の backtracking threshold $thr_i$ は $LB_i$ と $UB_i$ の間の値をとる．変更された ADOPT では，$thr_i$ の下界値を表 2 に示す $h_i(d_i)$ に制限する．ただし，ここでは $d_i$ は変数 $x_i$ の現在の値を表す．

*2 正確には，誤差の境界を用いる ADOPT は backtracking threshold とコストの下界値の差分をパラメータで指定された値に維持し，backtracking threshold がコストの上界値に到達したときに終了する [Modi 05].

- $lb_i(x_j, d)$: 2·3 節に示したように，元の ADOPT では，$lb_i(x_j, d)$ は 0 に初期化/再初期化される．変更された ADOPT では，$lb_i(x_j, d)$ は $min_{d' \in D_j} h_j(d')$ に初期化/再初期化される．ここでは，$j$ の親のエージェント $i$ は $h_j(d')$ を知るものと仮定する．しかし，ADOPT では，各エージェントは自身の子の現在の変数値を知らない．子の現在の変数値を取得するためには，アルゴリズムを比較的広範囲に変更する必要がある．そこで，ここではそのようなアルゴリズムの変更を避け，推定された下界の最小値を常に用いている．
- $x_i$ の初期値: $x_i$ は $\mathrm{argmin}_d h_i(d)$ に初期化される．

これらの変更については [Ali 05, Modi 05] を参考にされたい．

SACND は，図 2 に示した，方向なしのソフトアーク整合である．本実験の例題のコスト関数値は吸収元を含まないため SACND は拡張の操作を実行しない．拡張は 13 行目の，射影と拡張の無限の反復を避けるための条件により，ブロックされる．SACPTDTEX は 3·1 節に示した疑似木に基づく方向付きソフトアーク整合である．SACPTP は SACPTDTEX の部分的な手法である．SACPTP は射影のみを実行する点で SACND と類似するが，方向付きのソフトアーク整合である．SACPTDP2 は疑似木に基づく方向付きソフトアーク整合の別の方法であり，従来手法の DP2 に類似する．SACPTDP2 の例を図 4 (b) に示す．SACPTDP2 と SACPTDTEX の違いは，後退辺についての射影の方向である．SACPTDTP2 は下位側の変数に向かって射影を適用する．根のエージェントにおける大域的なコストの下界の推定値は，SACPTDP2 と DP2 とでは等しい．しかし，SACPTDP2 は各二項関数のコスト値は全体として削減される．さらに，根のエージェント以外では，吸収元ではない単項関数のコスト値は 0 となる．

疑似木は最も二項制約/評価関数と関係する変数を優先的に展開する順序 (most-constrained order) を用いて，深さ優先探索により生成した．実験結果の評価では，前処理のオーバヘッドは，探索処理の計算よりも十分に小さいため，無視した．等価な問題の解を得るまでの探索処理の反復回数を，各手法の効果の指標とした．実験ではメッセージサイクルを反復するシミュレーションを用いた．各メッセージサイクルでは，先ず各エージェントが受信キューからメッセージを読みだして処理し，必要に応じて送信キューにメッセージを書き込む．各送信キューのメッセージは，各サイクルの終了時に送信先の受信キューに移される．実験時間の制限のため，メッセージサイクル数の上限値を $10^6$ とした．実験はメッセージサイクル数の上限値に達したとき中断するものとし，その場合のメッセージサイクル数は上限値とした．

### 4·2 ADOPT の探索の効率化

ADOPT の探索処理に対する各前処理の効果を評価した．探索を中断した場合を含む終了までのメッセージサイクル数と，メッセージサイクル数の上限までに探索が終了したインスタンスの割合を図 7 に示す．結果は提案手法 SACPTDTEX がもっともメッセージサイクル数の削減に効果があることを示している．$n = 25$ の場合には，SACPTDTEX, SACPTDP2 および DP2 のいくつかのインスタンスのみで探索が終了した．SACPTP は SACND よりもメッセージサイクル数を削減した．これらはともに射影のみを適用するが，SACPTP は，疑似木にもとづく方向を用いるために，SACND よりも効率的であると考えられる．DP2 と SACPTDP2 は根のエージェントにおいては同一の下界の推定値を持つが，結果は DP2 が SACPTDP2 よりも効率的ではないことを示している．

### 4·3 コストの下界値の精度

各前処理により推定されたコストの下界値を評価するために，コストの下界値の精度を次のように定義する．ソフトアーク整合の場合は，変数 $x_i$ の値 $d$ について，$f_i(d)/LB_i(d)$ をコストの下界値の精度とする．また，DP0, 1 および 2 の場合は，変数 $x_i$ の値 $d$ について，$h_i(d)/LB_i(d)$ をコストの下界値の精度とする．

下界値の推定値の精度を図 8 に示す．図 8 (a) は大域的最適解に含まれる各変数値についての下界値の精度である．下界の推定値は，各変数値について計算されるため，本実験の例題では変数ごとに 3 個の下界値が推定される．ここでは特に，ADOPT の結果として得られた最適解に含まれる，各変数 $x_i$ の値についての精度を比較する．DP0 avg.，DP1 avg. および DP2 avg. は葉を除くすべての変数についての精度の平均を示す．それ以外の結果は根の変数を持つエージェントにおける精度を表す[*3]．実験結果は，探索が終了し最適解が得られたインスタンスすべてについて平均した．

DP2, SACPTDP2 および SACPTDTEX は下界の精度が比較的大きい．すなわち，真の下界値に近い，比較的大きな下界値を推定した．疑似木の上位の変数を持つエージェントは，前処理によってより大きな下界値を知る．これにより，上位の部分解の下界値を改善するためのメッセージサイクル数が，削減されると考えられる．上述のように DP2 や SACDP2 では下界値の精度が比較的大きい．また，これらの手法は図 7 に示されるようにメッセージサイクル数の削減の程度も比較的大きい．すなわち，これらの結果は，大きな下界の推定値がメッセージサイクル数の削減に効果があるという考察と整合している．DP2 と SACPTDP2 の精度は根のエージェントでは同一であ

*3 DP0, DP1 および DP2 の従来研究 [Ali 05] では，精度の平均値が評価されている．しかし，本実験では提案手法は根の変数についての単項関数以外はコスト値が 0 となるため，とくに根のエージェントについてのコストの下界値の精度に注目する．

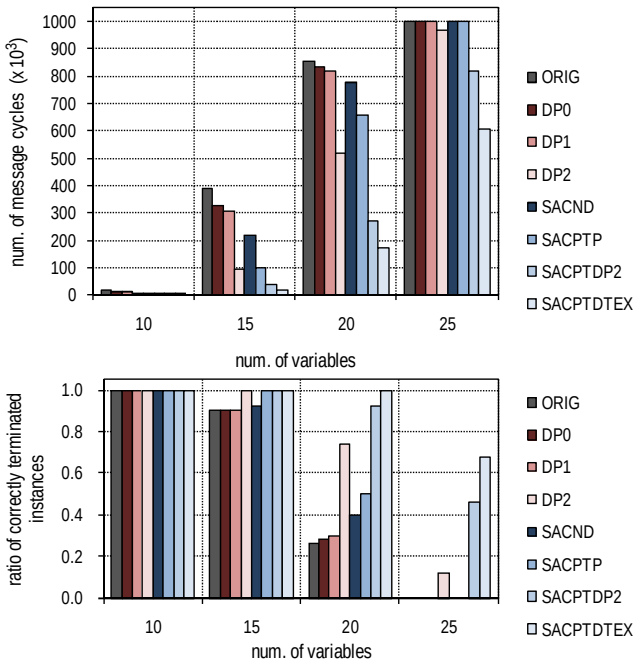
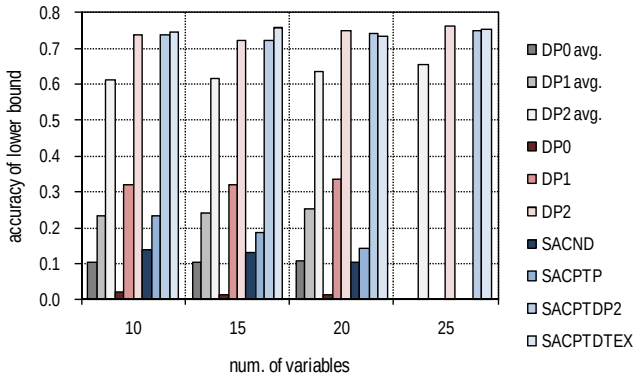
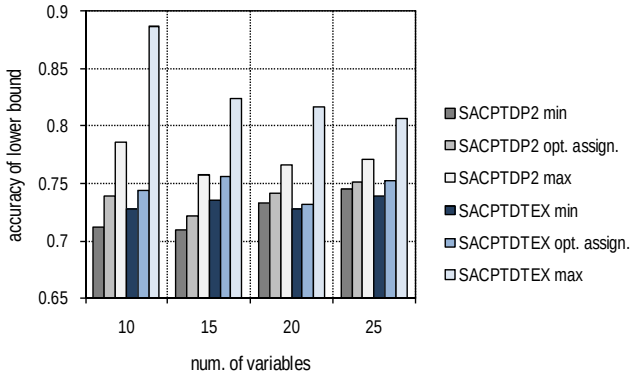



**図 7** ADOPT のメッセージサイクル数への影響 ($l = 2$)




(a) 最適解の変数値についてのコストの下界値の精度




(b) SACPTDP2 と SACPTDTEX のコストの下界値の精度

**図 8** コストの下界値の精度 ($l = 2$)

るが[*4]，図 7 に示されるように，SACPTDP2 の方がメッセージサイクル数をより多く削減している．

*4 図 8 (a) では，DP2 と SACPTDP2 の平均の精度は若干異なる．これはメッセージサイクル数の制限内で終了したインスタンスの数，および異なる複数の最適解の影響である．

**表 3** 二項関数に含まれる変数値の組のコスト ($l = 2$)

| problem | n | weight of tuples | | | num. of tuples [%] | | | |
|---|---|---|---|---|---|---|---|---|
| | | min. | max. | ave. | 0 | 1-50 | 51-100 | 101- |
| ORIG | 10 | 1.3 | 99.8 | 49.9 | 0 | 50.2 | 49.8 | 0 |
| | 15 | 1.1 | 99.9 | 50.0 | 0 | 49.9 | 50.1 | 0 |
| | 20 | 1.0 | 100.0 | 49.9 | 0 | 50.1 | 49.9 | 0 |
| | 25 | 1.0 | 100.0 | 49.9 | 0 | 50.3 | 49.7 | 0 |
| SACPTDP2 | 10 | 0 | 147.3 | 29.3 | 33.6 | 40.9 | 22.0 | 3.5 |
| | 15 | 0 | 163.6 | 29.6 | 33.7 | 40.7 | 21.8 | 3.8 |
| | 20 | 0 | 172.1 | 29.0 | 33.8 | 41.7 | 21.1 | 3.5 |
| | 25 | 0 | 178.2 | 29.0 | 33.7 | 41.4 | 21.5 | 3.4 |
| SACPTDTEX | 10 | 0 | 143.9 | 27.8 | 33.8 | 42.3 | 21.7 | 2.2 |
| | 15 | 0 | 161.6 | 28.3 | 33.8 | 41.9 | 21.3 | 2.9 |
| | 20 | 0 | 170.8 | 28.2 | 33.8 | 42.2 | 21.3 | 2.7 |
| | 25 | 0 | 176.3 | 28.4 | 33.8 | 41.9 | 21.5 | 2.8 |

DP2 により比較的大きい下界の推定値が得られるが，DP2 は元の問題を変更しない．この下界の推定値を用いるために ADOPT のアルゴリズムが変更され，推定された下界値は元の問題の下界値を制限するために用いられる．4·1 節に示したように，各エージェント $i$ は自身の下界値を制限するための推定値 $h_i(d)$ を知る．また，$i$ は子 $j$ の推定値 $h_j(d')$ も知る．$h_j(d')$ も用いることにより，$i$ と $j$ についてのコスト値の下界の精度が改善する．しかし，変更された ADOPT により得られる効果は，SACPTDP2 により問題を変換する場合よりも小さい．ADOPT をさらに改良することは可能かもしれないが，比較的複雑な計算を行うアルゴリズムをさらに変更する必要がある．一方で，疑似木に基づく方向付きソフトアーク整合は元の問題を等価な問題に変換する．大域的な下界値は疑似木の根に集計される．また大半の二項関数のコスト値は削減される．吸収元以外のコスト値の単項関数は，根のエージェント以外では，実質的に削除される．そのため，疑似木のそれぞれの親と子の単項関数が重複して持つ冗長な情報は，可能な限り削減される．従来手法のような，親のエージェントと子のエージェントが持つ下界の推定値の両方に含まれる冗長なコスト値を取り除く計算を，探索アルゴリズムの実行中に行う必要はない．

図 8 (b) は SACPTDP2 と SACPTDTEX の精度の比較を示す．この図では，最適解の変数値についての下界値の精度に加えて，すべての変数値についての下界値の精度の最大値と最小値も示している．SACPTDTEX の下界の最大値は SACPTDP2 の値よりも大きい．この大きな下界値には，根のエージェントの変数値が最良優先探索によって変更される回数を，削減する効果があると考えられる．

二項関数に含まれる変数値の組のコストについての評価を表 3 に示す．元の問題では，コスト値は一様分布に従う．等価な問題では，平均のコスト値は元の問題よりも小さい．特に，各二項関数 $f_{i,j}$ の $|D_i|$ 個の変数値の組に対するコスト値は 0 となる．SACPTDP2 のほぼすべてのインスタンスでは，$3ln$ 回の射影と $3(n-1)$ 回の拡張を実行する．SACPTDTEX は $3ln$ 回の射影と $3(n-1-(\text{葉の数}))$ 回の拡張を実行する．本実験の例題では，単項制約の初期のコスト値はすべて 0 である．しかし，SACPTDP2 は一時的に葉の変数についての単項関数の値を増加する．そのため，葉のエージェントでも拡張が必要となる．

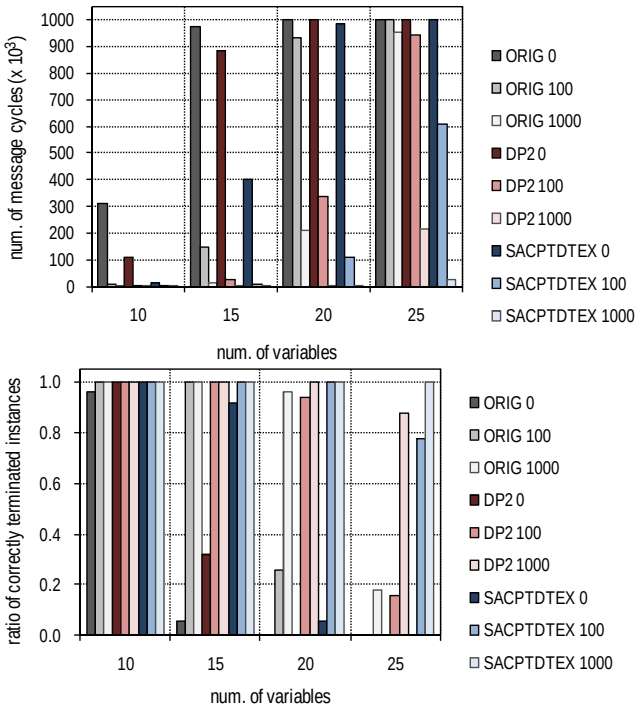
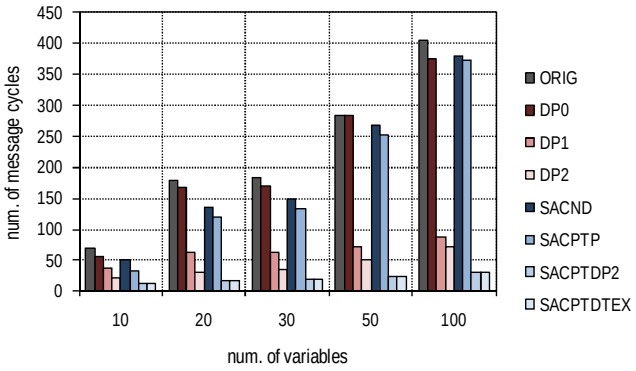



**図 9** 誤差の境界を用いる ADOPT への影響 ($l = 3$)




(全インスタンスでメッセージサイクル数の上限以内に探索が終了した)

**図 10** ADOPT のメッセージサイクル数への影響 ($l = 1$)

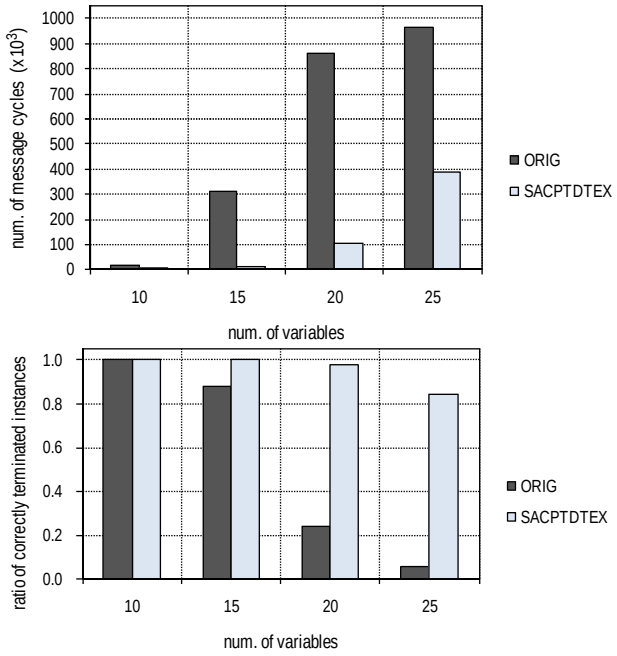



**図 11** ADOPT のメッセージサイクル数への影響 ($l = 2$, $|D_i| \in \{2,3,4\}$)

### 4·4 誤差の境界を用いる ADOPT

誤差の境界にもとづく近似解法として ADOPT を用いる場合について，提案手法の効果を評価した．$l = 3$ の問題における，終了までのメッセージサイクル数と，メッセージサイクル数の制限内で探索が終了したインスタンスの割合を図 9 に示す．図中の凡例では，各手法の名前の右側に誤差の境界値を示している．誤差の境界を用いることにより，ADOPT は制約密度の高い問題でも比較的少ないメッセージサイクル数で探索を終了する．誤差の境界を用いる場合も，提案手法はメッセージサイクル数をより削減する効果がある．誤差の境界を用いる ADOPT は，根のエージェントにおける下界の推定値が，最適コストから誤差の境界のパラメータを引いた差分値よりも，十分に大きいときに，速やかに終了する．提案手法は，この差分値が不十分である場合に DP2 よりも効果的である．

### 4·5 他の問題での効果

図 10 は $l = 1$ であり非常に制約密度が低いが，比較的多くの変数を持つ問題についての結果を示す．また，図 11 は，$l = 2$ であり，一様分布に従い $|D_i| \in \{2,3,4\}$ のような値域をとる問題についての結果を示す．実験結果は，これらの問題においてもソフトアーク整合は ADOPT のメッセージサイクル数を削減することを示している．

## 5. 関 連 研 究

本研究で提案した前処理は，動的計画法に基づく前処理 [Ali 05] および厳密解法 [Petcu 05] と類似する点がある．これらの手法は下界またはその推定値を，疑似木にしたがって一回のボトムアップな計算により求める．疑似木に後退辺が無い場合，いずれの手法も大域的最適コストを得る．後退辺がある場合は，メモリが制限されたアルゴリズムである前処理は，下界の推定値を得ることしかできない．動的計画法に基づく厳密解法は常に大域的最適コストを得るが，必要とする記憶のサイズが疑似木の木幅 (induced width) について指数関数的に増大する．ここで疑似木の木幅とは，疑似木のある変数 $x$ を頂点とするサブツリーに含まれる変数と制約で関係する，$x$ の祖先の変数の個数を表す．そのため，制約密度が比較的高い問題に，厳密解法の動的計画法を適用することは困難である．提案手法は従来手法の前処理 DP2 [Ali 05] と類似するが，3·1 節に述べたように，後退辺についての射影の方向が逆である．

本研究で用いたソフトアーク整合は，元の問題のコスト関数の下界値が 0 の場合には効果が無い．そのような場合は，ある部分解のもとで，他の部分解についてソフトアーク整合を適用する，条件付きソフトアーク整合を用いることが考えられる．元の問題のコスト関数の値が吸収元である場合には，そのコスト値については従来の制約充足問題のための方向なしのアーク整合を用いることができる．

同一の問題であっても疑似木が異なれば，ソフトアーク整合および従来手法ともに，前処理の結果は異なる．ただし，その場合でもソフトアーク整合は各制約のコスト値を削減し，削減された分のコスト値は疑似木の上位におけるコストの下界値を増加する．したがって，ADOPT のメッセージサイクル数の削減に効果はあると考えられる．本研究では，疑似木の生成の際に，最大次数を持つ変数を優先して子ノードとして展開する手法を用いた．これは分枝係数を大きくし，木の深さを小さくすることを目的とした貪欲戦略である．疑似木を別の方法で生成することも可能である．例えばグラフのトポロジがスケールフリーグラフなどの顕著な特徴を持つ場合に，その特徴を考慮して疑似木を生成することが考えられる．異なる疑似木の生成手法に対する，ソフトアーク整合の効果は，疑似木の生成手法自体の有効性とともに検討されるべき課題である．また，本研究では，コスト関数の変数値の組それぞれに対するコスト値は，一様分布に従うものとした．コスト関数の値に偏りがある問題では，ソフトアーク整合および従来手法の影響は，異なる疑似木について大きく異なる場合があると考えられる．このような問題を含む広範囲な評価も今後の課題である．

多数の変数を含む大規模な問題では，疑似木の生成に多くの計算を要する．しかし，疑似木の生成やソフトアーク整合のための，メッセージサイクル数や記憶のサイズは，変数の個数に関して線形な増加に押さえられる．また，本研究では疑似木を生成する方法の簡単な例として，単純に深さ優先探索を模倣する手法を用いた．その一方で，応用システムでは，制約網自体を疑似木を構成するように生成することも現実的な方法と考えられる．このとき，木辺についての分枝係数と深さが全体に均等であるような疑似木を，トップダウンに生成するのであれば，その深さは変数の個数の対数に従うと考えられる．また，各部分木の生成は並列に行える場合がある．このように特定の場合には，比較的多変数の問題において疑似木を生成することは可能であると考えられる．ただし，疑似木生成の方法が異なればソフトアーク整合との処理の統合方法は異なるものになるため，さらに検討が必要である．

## 6. むすび

本論文では，方向付きソフトアーク整合アルゴリズムを分散制約最適化問題に適用する，効率化手法を提案した．提案手法では，方向付きソフトアーク整合を疑似木にしたがってボトムアップに適用する．これにより元の問題は，従来の解法を用いて比較的容易に解くことができるような，等価な問題に変換される．提案手法は，探索処理の反復回数の削減において，コストの下界値を推定する従来手法よりも，効果的であることを実験により示した．また，提案手法は，誤差の境界を用いる近似解法においても有効である．

より効率的なソフトアーク整合の適用方法，より広範囲の種類の問題と解法に対する，提案手法の効果の解析は今後の課題に含まれる．

**謝　辞**

本研究の一部は科学研究費補助金 (基盤研究 B, 課題番号 19300048) による．

## ◇ 参 考 文 献 ◇

[Ali 05] Ali, S. M., Koenig, S., and Tambe, M.: Preprocessing techniques for accelerating the DCOP algorithm ADOPT, in *4th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1041–1048 (2005)

[Awerbuch 85] Awerbuch, B.: A New Distributed Depth-First-Search Algorithm, *Information Processing Letters*, Vol. 20, No. 3, pp. 147–150 (1985)

[Cooper 04] Cooper, M. and Schiex, T.: Arc consistency for soft constraints, *Artificial Intelligence*, Vol. 154, No. 1–2, pp. 199–227 (2004)

[Farinelli 08] Farinelli, A., Rogers, A., Petcu, A., and Jennings, N. R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm, in *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 639–646 (2008)

[Freuder 85] Freuder, E. C.: A sufficient condition for backtrack-bounded search, *Journal of the ACM*, Vol. 32, No. 14, pp. 755–761 (1985)

[Junges 08] Junges, R. and Bazzan, A. L. C.: Evaluating the performance of DCOP algorithms in a real world, dynamic problem, in *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 599–606 (2008)

[Maheswaran 04] Maheswaran, R. T., Tambe, M., Bowring, E., Pearce, J. P., and Varakantham, P.: Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling, in *3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 310–317 (2004)

[Mailler 04] Mailler, R. and Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation, in *3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 438–445 (2004)

[Modi 05] 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)

[Petcu 05] Petcu, A. and Faltings, B.: A Scalable Method for Multiagent Constraint Optimization, in *9th International Joint Conference on Artificial Intelligence*, pp. 266–271 (2005)

[Schiex 99] Schiex, T.: A note on CSP graph parameters, *Technical report 1999/03, INRA* (1999)

[Schiex 00] Schiex, T.: Arc consistency for soft constraints, in *Principles and Practice of Constraint Programming, 6th International Conference*, pp. 411–424 (2000)

[Yeoh 08] Yeoh, W., Felner, A., and Koenig, S.: BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm, in *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 591–598 (2008)

[Zhang 05] Zhang, W., Wang, G., Xing, Z., and Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, com-

parison and applications to constraint optimization problems in sensor networks, *Artificial Intelligence*, Vol. 161, No. 1-2, pp. 55–87 (2005)

[Zivan 08] Zivan, R.: Anytime Local Search for Distributed Constraint Optimization, in *Twenty-Third AAAI Conference on Artificial Intelligence*, pp. 393–398 (2008)

〔担当委員：栗 原 聡〕

2009 年 8 月 7 日　受理

## 著　者　紹　介

**松井　俊浩**（正会員）

1995 年名古屋工業大学電気情報工学科卒業．1999 年同大大学院博士前期課程修了．2006 年同博士後期課程修了．2006 年名古屋工業大学情報基盤センター助手．2007 年同助教．現在に至る．分散協調処理，マルチエージェントシステム，分散制約最適化問題に関する研究に従事．博士 (工学)．電子情報通信学会，情報処理学会，各会員．

**Marius C. Silaghi**

1997 年 Cluj-Napoca 工科大学卒業．1997 年ブラウンシュヴァイク工科大学，Scientific Computing Laboratory, DAAD Researcher．1998–2002 年スイス連邦工科大学 Artificial Intelligence Laboratory, Research Assistant．2002 年スイス連邦工科大学，Ph.D．2002 年フロリダ工科大学，Assistant Professor．Ph.D. (Computer Science)．AAAI (2000–2007), IEEE (2001–2008) 各会員．

**平山　勝敏**（正会員）

1990 年 大阪大学基礎工学部制御工学科卒．1992 年 同大大学院基礎工学研究科博士前期課程修了．1995 年 同大大学院基礎工学研究科博士後期課程修了．博士（工学）．1995 年 神戸商船大学助手．1997 年 同講師．2001 年 同助教授．2003 年 神戸大学海事科学部助教授（神戸大学と神戸商船大学の統合による）．2007 年 神戸大学大学院海事科学研究科准教授．1999 年–2000 年 カーネギーメロン大学ロボティクス研究所客員研究員（文部省在外研究員）．マルチエージェントシステム，制約充足，組合せ最適化に関する研究に従事．電子情報通信学会，情報処理学会，日本オペレーションズリサーチ学会，AAAI 各会員．

**横尾　　真**（正会員）

1984 年東京大学工学部電子工学科卒業. 1986 年同大学院修士課程修了. 同年 NTT に入社. 1990 年–1991 年 ミシガン大学客員研究員. 2004 年より 九州大学大学院 システム情報科学研究院 教授．マルチエージェントシステム, 制約充足問題に関する研究に従事. エージェントの合意形成メカニズム, 制約充足／分散制約充足等に興味を持つ. 博士 (工学). 1992 年, 2002 年 人工知能学会論文賞, 1995 年 情報処理学会坂井記念特別賞, 1999 年, 2005 年, 2008 年 人工知能学会全国大会優秀論文賞, 2004 年 ACM SIGART Autonomous Agent Research Award, 2005 年ソフトウェア科学会論文賞, 2006 年学士院学術奨励賞 受賞. 電子情報通信学会，情報処理学会, 日本ソフトウェア科学会, AAAI 各会員.

**松尾　啓志**（正会員）

1983 年名古屋工業大学情報工学科卒業．1985 年同大学大学院修士課程修了．同年松下電器産業　入社．1989 年名古屋工業大学大学院博士課程修了．同年名古屋工業大学電気情報工学科助手．講師，助教授を経て，2003 年同大学大学院教授．2006 年同大学情報基盤センターセンター長（併任）．現在に至る．分散システム，分散協調処理に関する研究に従事．工学博士．電子情報処理学会, 情報処理学会，IEEE 各会員．