特集論文 「エージェント」

敵対者に対応する協調問題解決： 限量記号付き分散制約充足問題

Cooperative Problem Solving against Adversary: Quantified Distributed Constraint Satisfaction Problem

馬場 里美
Satomi Baba

九州大学大学院システム情報科学府
Graduate School of ISEE, Kyushu University
s-baba@agent.is.kyushu-u.ac.jp

岩崎 敦
Atsushi Iwasaki

(同 上)
iwasaki@is.kyushu-u.ac.jp

横尾 真
Makoto Yokoo

(同 上)
yokoo@is.kyushu-u.ac.jp

Marius C. Silaghi

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

平山 勝敏
Katsutoshi Hirayama

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

松井 俊浩
Toshihiro Matsui

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

keywords: distributed constraint satisfaction problem, quantified constraint satisfaction problem

Summary

In this paper, we extend the traditional formalization of distributed constraint satisfaction problems (DisCSP) to a quantified DisCSP. A quantified DisCSP includes several universally quantified variables, while all of the variables in a traditional DisCSP are existentially quantified. A universally quantified variable represents a choice of nature or an adversary. A quantified DisCSP formalizes a situation where a team of agents is trying to make a robust plan against nature or an adversary. In this paper, we present the formalization of such a quantified DisCSP and develop an algorithm for solving it by generalizing the asynchronous backtracking algorithm used for solving a DisCSP. In this algorithm, agents communicate a value assignment called a *good* in addition to the *nogood* used in asynchronous backtracking. Interestingly, the procedures executed by an adversarial/cooperative agent for *good/nogood* are completely symmetrical. Furthermore, we develop a method that improves this basic algorithm. Experimental evaluation results illustrate that we observe an easy-hard-easy transition by changing the tightness of the constraints, while very loose problem instances are relatively hard. The modification of the basic algorithm is also effective and reduces the number of cycles by approximately 25% for the hardest problem instances.

1. は じ め に

制約充足問題 [Mackworth 92] は全ての制約を満足する、変数への値の割当を求める問題である。各変数は有限で離散的な領域から値を取る。制約充足問題は人工知能の様々な問題を定式化できる。そのため、制約充足問題の研究は非常に重要で、これまでに多くの研究がなされている。

分散制約充足問題 [横尾 92] は制約充足問題の変数と制約を複数エージェントへ分散させた問題である。様々なマルチエージェントシステムの問題 (分散センサネットワーク [Fernández 02] 等) が分散制約充足問題で定式化

できる。また、分散制約充足問題を解く多くのアルゴリズムが提案されている (非同期バックトラッキング [横尾 92], 分散動的バックトラッキング [Bessiere 01], 非同期部分オーバーレイ [Mailler 06] 等)。通常の分散制約充足問題に関する研究では、全てのエージェントは協調的であり、エージェントのチームは全ての事象をコントロールできることを仮定している。外的要因が存在する場合は、その要因はすでに固定されていて、チームがプランの探索/実行を行うときには変化しないことを前提としている。

実際には、この仮定が成立しない状況が多く存在する。例えば、プランニング時に固定されてない、コントロー

ル不可能なランダム要因が存在する可能性がある．また，そのような要因がチームに敵対的な行動をする敵対者のコントロール下にある状況も考えられる．本研究では，自然や敵対者の選択を表現する全称限量化された変数を導入することにより，上記の制約を扱える限量記号付き分散制約充足問題の枠組みを提案する．

開環境での問題解決のモデルとして，制約充足問題の拡張は他にも存在する．例えば，動的制約充足問題では，制約と変数が時間の経過とともに変化する [Verfaillie 94]．open CSP では，変数の領域が予め分かておらず，情報収集によって領域を求める必要がある [Faltings 05]．限量記号付き (分散) 制約充足問題の枠組みは，動的な変化や問題解決の不確実性が敵対者によるという点において上記のアプローチとは異なる．

限量記号付き制約充足問題 [Chen 04] は一部の変数を全称限量記号で限量化することにより制約充足問題を拡張した問題である．限量記号付き制約充足問題では，全称限量化された変数の値に関わらず，存在限量化された変数へ全ての制約を満足する値を割り当てることを目的とする．制約充足問題が NP 完全問題であるのに対し，限量記号付き制約充足問題は PSPACE 完全問題である．限量記号付き制約充足問題では，全称限量化された変数を自然や敵の選択と考えることができるため，不確実性を含むプランニングや敵が存在するゲームなどの問題を定式化できる．

本論文では分散制約充足問題に全称限量化された変数を用いることにより，分散制約充足問題と限量記号付き制約充足問題を組み合わせた限量記号付き分散制約充足問題を提案する．本問題では，協調的なエージェントのチームが，自然や敵に対抗する頑健なプランを構築することを目指す．

さらに，本論文では限量記号付き分散制約充足問題を解くアルゴリズムを提案する．本アルゴリズムは分散制約充足問題を解く非同期バックトラッキングアルゴリズム [横尾 92] を一般化したものである．本アルゴリズムでは，制約を違反する変数への値の割当である *nogood* と，いくつかの制約を満足する変数への値の割当である *good* をエージェントが通信する．そして，本アルゴリズムでは敵対的なエージェントが *good/nogood* に関して協調的なエージェントと対称的な処理を行う．また，本論文では，より小さい *nogood* を生成し，直接の親以外の祖先にも *nogood* を送ることでベーシックアルゴリズムを改善したアルゴリズムを提案する．

評価実験では，通常分散制約充足問題のように，制約の密度によって問題の難しさが easy-hard-easy と転移することが分かった．ただし，エージェントのチームが全称限量化された変数の可能な全ての値の割当を考える必要があるため，制約の密度が低い問題は，制約の密度が高い問題よりも難しい．また，ベーシックアルゴリズムの改善は効果的で，最も難しい問題においてサイクル

数を 25% 削減できた．

2. 関 連 研 究

2.1 限量記号付きブール式

限量記号付きブール式とは，一部の変数を全称限量記号で限量化することにより，充足可能性問題を一般化した問題である．充足可能性問題とは，与えられたブール式を真とする値の割当が存在するかを判定する問題である．一般的に充足可能性問題で扱うブール式は，連言標準形で与えられる．連言標準形とは節の連言であり，節とはリテラル (変数あるいは変数の否定) の選言である．

限量記号付きブール式とは，変数が存在限量記号 ($\exists$) あるいは全称限量記号 ($\forall$) で限量化されているブール式の充足可能性を問う問題である．各限量記号の意味は次の通りである．

- $\exists x F$ が充足可能：式 F を真とするある x の値が存在する．
- $\forall x F$ が充足可能：任意の x の値について式 F は真となる．

限量記号付きブール式は，(1) のように，各変数を限量化する限量記号を定義するシーケンス Q と，連言標準形で与えられるブール式 F で QF と定義される．

$$\exists x_1 \forall x_2 \exists x_3 (x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \quad (1)$$

変数が n 個の限量記号のシーケンスは (2) のように定義される． Q は n 個の限量記号 Q_i と変数 x_i のペアで構成される．

$$Q_1 x_1 \cdots Q_n x_n \quad (2)$$

限量記号のシーケンスは，各変数を限量化する限量記号を定義するとともに，変数の順序も定義している．例えば， $\forall x \exists y \text{loves}(x, y)$ と $\exists y \forall x \text{loves}(x, y)$ の意味は異なる． $\forall x \exists y \text{loves}(x, y)$ は任意の x がいくつかの y を愛していることを意味し，各 x に対して y は異なる可能性がある．一方， $\exists y \forall x \text{loves}(x, y)$ はある特定の y が全ての x から愛されていることを意味する．

限量記号付きブール式の目的は，全称限量化されている変数がとり得る全ての値に対して，ブール式が真となるように存在限量化されている変数へ値を割り当てることである．ある全称限量化された変数 x_i に対して，存在限量化された変数 x_j が x_i より前に現れるなら， x_i がとり得る全ての値に対して式が真とならなければならない．一方， x_j が x_i より後に現れるなら， x_j の値は x_i の値に依存して，異なる値に決定することができる．

限量記号付きブール式を解くアルゴリズムは既に様々なものが開発されており，DPLL アルゴリズム [Davis 62] をベースとした Qube [Giunchiglia 01] やスコレム化を用いた sKizzo [Benedetti 05] などがある．

2.2 限量記号付き制約充足問題

制約充足問題は変数へ制約を満足する値を割り当てる問題である．制約充足問題は， n 個の変数 $x_1, x_2, \dots, x_n$ と l 個の制約 $C_1, C_2, \dots, C_l$ で構成される．各変数 x_i は有限で離散的な領域 D_i から値をとる．

一部の変数を全称限量化することで，制約充足問題を一般化した問題が限量記号付き制約充足問題 [Chen 04] である．また，限量記号付きブール式の一般化でもある．限量記号付き制約充足問題は PSPACE 完全問題である．限量記号付き制約充足問題は，シーケンス Q と制約 C で (3) のように QC と定義する．

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 (x_1 \neq x_3) \wedge (x_1 < x_4) \wedge (x_2 \neq x_3) \quad (3)$$

限量記号付き制約充足問題の意味論は，再帰的に次に示すように定義される．

- C が空ならその問題の制約は満足される． Q が $\exists x_1 Q_2 x_2 \dots Q_n x_n$ の形のとき， $Q_2 x_2 \dots Q_n x_n C[(x_1, a)]$ を満足する値 $a \in D_1$ が存在するならその問題の制約は満足される． Q が $\forall x_1 Q_2 x_2 \dots Q_n x_n$ の形のとき，任意の値 $a \in D_1$ に対して $Q_2 x_2 \dots Q_n x_n C[(x_1, a)]$ を満足する割り当てが存在するならその問題の制約は満足される． (x_1, a) は x_1 に値 a を割り当ててることを意味する．

限量記号付き制約充足問題を解くアルゴリズムの多くは限量記号付きブール式のアルゴリズムをベースに拡張している．それに対し，制約充足問題を解くアルゴリズムを拡張したアルゴリズムには QCSP-Solve [Gent 05] などがある．

2.3 分散制約充足問題

分散制約充足問題 [横尾 92] とは，制約充足問題の変数が複数のエージェントに分散された問題である．

本論文では以下のエージェント間通信のモデルを仮定する．

- エージェント間通信はメッセージ通信によってなされる．
- エージェントは，他のエージェントのアドレスを知っている場合に限りそのエージェントにメッセージを送信できる．
- 任意の二つのエージェントの組合せに関して，送信されたメッセージの順序は保存される．

各エージェントはいくつかの変数を持っておりその値を決定する．しかしながら，エージェント間には制約が存在し，値の割当はその制約を満足しなければならない．非同期バックトラッキングアルゴリズム

非同期バックトラッキングアルゴリズム [横尾 92] の説明を行う際に，簡単化のため，次の仮定を導入する．

- 各エージェントの持つ変数は唯一である．

- 各エージェントは自分に属する変数が関係する制約を全て知っている．
- エージェント間の制約は全て二項関係 (binary) である．

これらの仮定の一般化は容易であり，アルゴリズムの一般性は失われていない．これらの仮定により，分散制約充足問題はネットワークとして表現可能である．このネットワークでは，エージェントをノード，制約をノード間のリンクと考える．制約に対応するリンクには，方向が定められているとする．ある制約に関係する 2 つのエージェントのうち，片方がその制約を評価し，もう片方のエージェントが，自分の変数の値を，制約を評価するエージェントに送信する．すなわち，値を送信するエージェントから制約を評価するエージェントに向かうようにリンクの向きが付けられる．また，各エージェントには，識別子による優先順位が与えられているとする．リンクの向きは，この優先順位に従って設定される．

非同期バックトラッキングアルゴリズムは，分散制約充足問題を解く基本的なアルゴリズムである．非同期バックトラッキングアルゴリズムでは，各エージェントは非同期，並行的に自分の変数の値を決定し，その値を外向きのリンクで結ばれたエージェントに対して送信する．その後，各エージェントはメッセージ待ちの状態となり，以降，到着したメッセージに関する処理を行う．非同期バックトラッキングアルゴリズムでは，次の二種類のメッセージを用いる．

- $(ok?, (x_j, value))$: 変数 x_j の値の割当が $value$ であることを伝えるメッセージ．
- $(nogood, x_j, nogood)$: 新たに発生した $nogood$ を伝えるバックトラックメッセージ． $nogood$ は制約違反を起こす値の組合せである．例えば， $nogood\{(x_i, d_i), (x_j, d_j)\}$ は， (x_i, d_i) と (x_j, d_j) が制約違反を起こすことを表す．

非同期バックトラッキングアルゴリズムでは， $ok?$ メッセージを受信することにより，エージェント x_i は自分より優先順位の高いエージェントとの間にある制約を満足する値を見つける．制約を満足する値がなければ，エージェント x_i は，自分より優先順位の高いエージェントのうち最も優先順位の低いエージェントへ $nogood$ メッセージを送信する．非同期バックトラッキングアルゴリズムでは，アルゴリズムの完全性を保証しつつも，全体の制御なしで各エージェントが非同期，並行的に行動できる．

3. 限量記号付き分散制約充足問題

3.1 問題の定義

限量記号付き分散制約充足問題は，複数のエージェントに変数が分散された限量記号付き制約充足問題である．本問題の目的は，協調的なエージェントのチームが，自然

や敵に対抗する頑健なプランを構築することである．自然や敵の行動は敵対的なエージェントの行動として表す．本問題は以下のように定義される．

- 変数の集合 $X = \{x_1, x_2, \dots, x_n\}$ ，および各変数の領域 $D_1, D_2, \dots, D_n$ ．
- 制約の集合 $C = \{C_1, C_2, \dots, C_l\}$ ．
- 限量記号のシーケンス Q ．
- 協調的なエージェントの集合 $Ag_{\exists}$ および 敵対的なエージェント $a_{\forall}$ ．エージェントの全体集合を $Ag_{\exists} \cup \{a_{\forall}\} = Ag$ と記述する．
- エージェントと変数の対応関係を示す関数 $\phi: Ag \rightarrow 2^X$ ．ただし，変数 $x_j \in \phi(a_{\forall})$ ならば $Q_j = \forall$ であり， $i \in Ag_{\exists}$ に関して，変数 $x_j \in \phi(i)$ ならば $Q_j = \exists$ である．

各エージェント i は，自分が持つ変数 $x_j \in \phi(i)$ について， D_j, Q_j を知っており，また， x_j を含むすべての制約を知っているものとする．さらに，各エージェントは，どの変数が協調的なエージェントに属し，どの変数が敵対的なエージェントに属するかも知っているものとする．

また，限量記号のシーケンスは値の決定順序を定義する．すなわち，シーケンス中で x_i が x_j より前に現れるなら， x_j の値を決定するときには， x_i の値は観測可能である．限量記号付き分散制約充足問題において，協調的なエージェントのチームは，敵対的なエージェントがどのような値を選択しても，全ての制約を満足する値の組合せを求めることを目的とする．

3.2 限量記号付き分散制約充足問題の問題例

限量記号付き分散制約充足問題の例として偶発的な事故を予期したプランニングを考える．本問題の目的は， n 個のノードが互いに（直接的にあるいは間接的に）つながっている通信ネットワークを次の条件の下で構築することである．

- 最終的なネットワークは k 個のノードの故障に対して頑健でなければならない．つまり， k 個のノードが故障しても，最終的なネットワークは互につながっていないなければならない．
- 最終的なネットワークは T ステップ内で構築されなければならない．
- 各ステップで，各ノードは多くとも一つのリンクを張ることができる．
- リンクの総数は l を超えてはならない．
- 敵は各ステップのあとに，任意のノードを一つ故障させることができる（最後の T 番目のステップを除いて）．
- 敵は T ステップのあとに任意の k 個のノードを故障させることができる（つまり，敵は合計で $T + k - 1$ 個のノードを故障させることができる）．

本問題では，各エージェントはノードであると仮定する．本問題は次のようにして限量記号付き分散制約充足

図1 失敗プラン

図2 成功プラン

問題として定式化できる．

- 変数: x_{ij} ($1 \leq i \leq n, 1 \leq j \leq T$) と y_j ($1 \leq j \leq T + k - 1$)． x_{ij} は各ステップにおけるノードを， y_j は敵の行動を表す． i はノードの番号を示し， j ($j \leq T$) はステップを示している． $j \geq T$ における y_j は T ステップ後の敵の行動を表す．
- 領域: x_{ij} は $\{0, \dots, i-1, i+1, \dots, n\}$ から値をとる．各値は x_{ij} が張るリンクの相手ノードを表す． x_{ij} が値 0 をとるとき， x_{ij} はリンクを張らない．
 y_j は $\{0, \dots, n\}$ から値をとる．各値は故障するノードを表す． y_j が値 0 をとるとき，どのノードも故障しない．
- 制約: リンクの総数は l 以下である．全てのノードは y_{T+k-1} の選択のあと，互いにつながっている．
- 限量記号シーケンス: $\exists x_{11} \dots \exists x_{n1} \forall y_1 \dots \exists x_{1T} \dots \exists x_{nT} \forall y_T \dots \forall y_{T+k-1}$

本問題を解いて得られる解は，敵の行動の全ての組合せに対して協調的なエージェントがとるべき行動を示したプランである．したがって，解は木構造となる．実際にネットワークを構築するとき，協調的なエージェントは得られたプランをもとに敵の行動に対する値を決定する．

$n = 6, T = 3, k = 2, l = 9$ のときのプランの実行例を図1と図2に示す．図1では，2ステップ目のあと，敵が左のノードと右のノードを故障させている．すでにリンクを6本張っているため，エージェントはあと3本しかリンクを張れない．したがって，3ステップで2個のノードの故障に頑健なネットワークは構築できない．

一方，図2では，3ステップ内でエージェントは2個のノードに頑健なネットワークを構築できている．目的を達成するためには，エージェントはリンクを張る順番を注意深く決定する必要がある．

4. 限量記号付き分散制約充足問題を解くアルゴリズム

本章では，限量記号付き分散制約充足問題を解くアルゴリズムを提案する．最初のステップとして，まずは非同期バックトラッキングアルゴリズムを拡張したアルゴ

リズムを提案する．非同期バックトラッキングアルゴリズムよりも複雑で効率的な分散制約充足問題を解くアルゴリズムに関する研究は多く存在するが，非同期バックトラッキングアルゴリズムは最も基本的で分かりやすい．このため，本研究では非同期バックトラッキングを新たなアルゴリズムを開発する際の基盤として用いることとする．提案アルゴリズムで用いたテクニックやアイデアは，他の分散制約充足問題のアルゴリズムをベースとしたより効率的な限量記号付き分散制約充足問題のアルゴリズムにも有効であると考えられる．

4.1 基本的なアイデア

以下に，非同期バックトラッキングアルゴリズムからの拡張アイデアを示す．

- エージェント間の優先順位は限量記号のシーケンスに基づいて決定する． x_j が x_i より前に現れるなら， x_j は x_i より優先順位が高い．ただし，シーケンス中で隣り合っている存在限量化された変数の順位は任意に決定される．
- 簡単化のため，エージェントは優先順位によって決定される Depth First Search (DFS) 木を知っているものとする．ここでは，簡単のため，各エージェントは DFS 木の情報を知っていることを仮定するが，エージェントはシーケンス中の変数の順位について知っているため，分散アルゴリズムを用いて DFS 木を構成することも可能である．
- 各全称限量化された変数に対して，敵対的なエージェントを模倣し，探索に協力する仮想的なエージェントを配置する．協調的なエージェントのチームはオフラインでプランを構築すると考えることができる．チームが実際に敵と対戦するとき，オフライン探索で得られたプランを実行する．実際には，チーム中のあるエージェントが仮想的なエージェントとして行動する．チームメンバは誰でも全称限量化された変数 x_i に対して，仮想エージェントとして行動できる．しかしながら，通信コストを削減するために，DFS 木における x_i の親あるいは子にあたるエージェントが仮想エージェントとして行動するべきである．以降，敵対的なエージェントは仮想エージェントを表すこととする．
- エージェントは，ok?メッセージ，nogoodメッセージに加え，goodメッセージを用いて通信を行う．goodはgoodメッセージの送信者とその子孫が持つ制約を満足する値の組合せである．敵対的なエージェントはgood/nogoodに関して，協調的なエージェントと対称的な処理を行う．

4.2 nogood/goodの生成

本節では，nogood/goodに対する存在限量記号/全称限量記号で限量化された変数における処理を比較する．ま

た，その処理が論理的に正しいことを示す．

nogoodは，矛盾を導く変数の値の組合せである．例えば， $\text{nogood}\{(x_1, 1), (x_2, 2)\}$ は， $x_1 = 1 \wedge x_2 = 2 \rightarrow \perp$ であることを表している．すなわち， $x_1 = 1 \wedge x_2 = 2$ という値の組合せのとき，矛盾が生じることを示す．

新たな nogood は，以下のようにして導かれる．

- 領域が $\{1, 2\}$ の x_2 が存在限量化されている変数のとき， $x_2 = 1 \vee x_2 = 2$ ， $\text{nogood}\{(x_1, 1), (x_2, 1)\}$ ， $\text{nogood}\{(x_1, 1), (x_2, 2)\}$ から新たな $\text{nogood}\{(x_1, 1)\}$ を導くことができる．

これは通常の導出処理で， $x_1 = 1 \wedge x_2 = 1 \rightarrow \perp$ ， $x_1 = 1 \wedge x_2 = 2 \rightarrow \perp$ ，そして $x_2 = 1 \vee x_2 = 2$ より $x_1 = 1 \rightarrow \perp$ を導いている．

- x_2 が全称限量化されている変数のとき， $\text{nogood}\{(x_1, 1), (x_2, 1)\}$ より $\text{nogood}\{(x_1, 1)\}$ を導くことができる．

この処理は $x_1 = 1 \wedge x_2 = 1 \rightarrow \perp$ と $x_2 = 1$ から $x_1 = 1 \rightarrow \perp$ を導くものである．

したがって，存在限量化された変数を持つ協調的なエージェントは，自分の持つ変数がとり得る全ての値が（自分の持つ制約と受信した nogood によって）矛盾を引き起こす場合に限り，nogood メッセージを送信する．

一方，全称限量化された変数を持つ仮想的/敵対的なエージェントは，矛盾を引き起こす値を少なくとも一つ発見したなら，すぐに nogood メッセージを送信する．

good はいくつかの制約を満足する値の組合せである．例えば，エージェント x_3 から送信された $\text{good}\{(x_1, 1), (x_2, 2)\}$ は $x_1 = 1 \wedge x_2 = 2$ が x_3 とその子孫の制約を満足することを表している．

good は以下のようにして導かれる．

- x_2 が存在限量化されているとき， $x_1 = 1 \wedge x_2 = 1$ が x_1 と x_2 間の制約を満足するなら，唯一の子 x_3 から送信された $\text{good}\{(x_1, 1), (x_2, 1)\}$ より $\text{good}\{(x_1, 1)\}$ を導くことができる．

この処理は $\{(x_1, 1), (x_2, 1)\}$ が x_1 と x_2 の間の制約を満足するとともに x_3 とその子孫に関する制約も満足することを意味する．したがって， $\{(x_1, 1)\}$ が x_2 とその子孫に関する制約を満足することが導かれる (x_2 が値 1 を選択すると仮定したとき)．

- 領域が $\{1, 2\}$ の x_2 が全称限量化されているとき， $x_1 = 1 \wedge x_2 = 1$ と $x_1 = 1 \wedge x_2 = 2$ が x_1 と x_2 間の制約を満足するなら， $x_2 = 1 \vee x_2 = 2$ と唯一の子 x_3 から送信された $\text{good}\{(x_1, 1), (x_2, 1)\}$ ， $\text{good}\{(x_1, 1), (x_2, 2)\}$ より $\text{good}\{(x_1, 1)\}$ を導くことができる．

この処理は $\{(x_1, 1), (x_2, 1)\}$ と $\{(x_1, 1), (x_2, 2)\}$ が x_1 と x_2 の間の制約とともに x_3 とその子孫に関する制約を満足することを意味する．したがって， $\{(x_1, 1)\}$ が x_2 とその子孫に関する制約を満足することが導かれる (x_2 の選択に関わらず)．

存在限量化された変数を持つ協調的なエージェントは、少なくとも一つの値に対して *good* を受信したなら（かつその値が自身の制約を満足するなら）、すぐに *good* メッセージを送信する。一方、全称限量化された変数を持つ仮想的/敵対的なエージェントは、自分の持つ変数がとり得る全ての値に対して、*good* を受信した（かつ各値が自身の制約を満足する）ときに限り、*good* メッセージを送信する。

4.3 アルゴリズムの詳細

本アルゴリズムでは、非同期バックトラッキングアルゴリズムと同様に、各エージェントは非同期、並行的に値を決定する。簡単化のため、エージェントは *ok?* メッセージを用いて、値を全ての子孫エージェントへ送信するとする。したがって、子孫エージェントは先祖の値変更を直ぐに知ることができる。各エージェントは優先順位によって決定される DFS 木を知っているため、全子孫へメッセージを送ることが可能である。エージェントは値を送信した後、メッセージ待ちの状態になり、以降、到着したメッセージに関する処理を行う。

本アルゴリズムで通信されるメッセージは、*ok?*、*nogood*、*good* の三種類である。図 3 に協調的なエージェントのメッセージ処理、図 4 に敵対的/仮想的なエージェントのメッセージ処理を示す。また、図 5 には、上記の処理で用いられる *backtrack* と *send_good* の処理と各エージェントの初期化処理 *Initialize* を示す。

協調的なエージェントのメッセージ処理は、非同期バックトラッキングアルゴリズムのメッセージ処理とほぼ同じであり、非同期バックトラッキングアルゴリズムとの違いは、*good* メッセージに対する処理である。非同期バックトラッキングアルゴリズムと同様に、協調的なエージェントが *ok?* あるいは *nogood* メッセージを受信したとき、エージェントは *agent_view* と *nogood_list* を更新し、現在の値が *agent_view* と整合がとれているかをチェックする。ただし、葉エージェントが *ok?* メッセージを受信し、制約を満足する値が存在する場合は、親エージェントへ *good* メッセージを送信する。協調的なエージェントが *good* メッセージを受信したとき、エージェントは *good_list* を更新し、全ての子から同じ値の組合せに対して *good* メッセージを受信している場合は、親へ *good* メッセージを送信する。

協調的なエージェントが全ての制約を満足する値を求めるのに対し、敵対的/仮想的なエージェントは制約を違反する値を求めようとする。したがって、敵対的なエージェントは、*good*、*nogood* に関して協調的なエージェントと対称な処理を行う。

敵対的/仮想的なエージェントが *ok?* メッセージを受信したとき、エージェントは制約違反を起こす値が一つでもあるなら *nogood* メッセージを送信する。

協調的なエージェントが *nogood* メッセージを受信し

```
when received (ok?, (xj, value)) do
  add (xj, value) to agent_view;
  check_agent_view;
  when agent is a leaf and,
    agent_view contains all ancestors do
    send_good; end do; end do;
```

```
when received (nogood, xj, nogood) do
  add nogood to nogood_list
  check_agent_view; end do;
```

```
when received (good, xj, good) do
  add good to good_list
  when received consistent good from all children
    and agent_view contains all ancestors do
    send_good; end do; end do;
```

```
procedure check_agent_view;
  when current_value and agent_view are inconsistent do
    change current_value to a new consistent value;
  when cannot find such a value do
    backtrack; end do; end do;
  send (ok?, (xi, current_value)) to descendants;
```

図 3 協調的なエージェントのメッセージに対する処理

たとき、エージェントは制約を満足する他の値を探す。制約を満足する値がなければ、親へ *nogood* メッセージを送信する。一方、敵対的なエージェントが *nogood* メッセージを受信したとき、エージェントは制約を満足する値を探さず、*nogood* メッセージを親へ送信する。なぜなら、エージェントは受信した *nogood* 中の値を選択し、制約を違反できるからである。

協調的なエージェントが同じ値の組合せに対して全ての子から *good* メッセージを受信したとき、エージェントは受信した *good* 中の値を選択し、制約を満足できるため、すぐに親へ *good* メッセージを送信する。一方、敵対的なエージェントが *good* メッセージを受信したとき、エージェントは制約を違反する値を探す。そのような値がなければ、すなわち、そのエージェントの持つ変数がとり得る全ての値に対して *good* メッセージを受信したなら、親へ *good* メッセージを送信する。

簡単化のため、提案アルゴリズムは問題を解決可能か否かのみを求めるものとする。敵に対する行動プランを構築するためには、存在限量化された変数を持つエージェントは子から受信した *good* を記録しなければならない。

4.4 アルゴリズムの実行例

アルゴリズムの実行例を図 6 に示す。この問題は、 $Q = \exists x_1 \forall x_2 \exists x_3 \exists x_4, C = \{nogood\{(x_1, 1), (x_3, 1)\}\}$,

```

when received ( $ok?, (x_j, value)$ ) do
  add  $(x_j, value)$  to  $agent\_view$ ;
  check_agent_view_adversary; end do;

when received ( $nogood, x_j, nogood$ ) do
  add  $nogood$  to  $nogood\_list$ ;
  when  $nogood$  is consistent with  $agent\_view$  and
     $current\_value$ , and  $agent\_view$  contains all ancestors do
    backtrack; end do; end do;

when received ( $good, x_j, good$ ) do
  add  $good$  to  $good\_list$ ;
  check_agent_view_adversary; end do;

procedure check_agent_view_adversary;
  when for some  $v \in D_i$ ,
     $v$  and  $agent\_view$  are inconsistent do
    backtrack; end do;
  when for all  $v \in D_i$ ,
    received  $good$  messages from all children
    (or a leaf agent),
    and  $agent\_view$  and  $v$  are consistent do
    send_good; end do;
  otherwise do choose  $v \in D_i$  so that
    some children have not sent a  $good$  message yet;
    change  $current\_value$  to  $v$ ;
    send ( $ok?, (x_i, current\_value)$ ) to descendants;
  end do;

```

図4 敵対的なエージェントのメッセージに対する処理

$nogood \{(x_2, 1), (x_3, 2)\}, nogood \{(x_2, 2), (x_4, 1)\}\}$,
 $D_1 = D_2 = D_3 = D_4 = \{1, 2\}$ で構成される問題である。

まず, x_1, x_2 から $ok?$ メッセージを受信した後, x_3, x_4 の $agent_view$ は $\{(x_1, 1), (x_2, 1)\}$ となる (図 6 (a))。 x_3 の値はどちらも $agent_view$ と制約を満足しないため, x_3 は親である x_2 へ $nogood$ メッセージを送信する。一方, x_4 は制約を満足できる値を選択できるので, 親である x_2 へ $good$ メッセージを送信する (図 6 (b))。 x_2 はこの $nogood$ メッセージと $good$ メッセージを受信する。 x_2 は, 敵対的なエージェントであるため, x_1 へ $nogood$ メッセージを送信する (図 6 (c))。

x_1 は, x_2 から送信された $nogood$ メッセージより, $x_1 = 1$ が制約違反を引き起こすことを知る。そこで, x_1 は, 値を 2 に変更し, 子孫へ $ok?$ メッセージを送信する (図 6 (d))。

この $ok?$ メッセージを受信した x_2, x_3, x_4 は, その値を $agent_view$ に記録する。 $x_3 = 1$ が制約を満足するため, x_3 は x_2 へ $good$ メッセージを送信する。 x_4 にも制約を満足する値があるため, x_2 へ $good$ メッセージを送信す

```

procedure Initialize
  select  $value$  from  $D_i$ ;
   $current\_value \leftarrow value$ ;
   $agent\_view \leftarrow \{\}$ ;  $nogood\_list \leftarrow \{\}$ ;  $good\_list \leftarrow \{\}$ ;
  send ( $ok?, (x_i, current\_value)$ ) to descendants;

procedure backtrack
   $nogood \leftarrow agent\_view$ 
  when  $nogood = \{\}$  do
    broadcast to other agents that the problem is unsolvable;
    terminate this algorithm; end do;
  send ( $nogood, x_i, nogood$ ) to its parent  $x_j$ ;
  remove  $(x_j, d)$  from  $agent\_view$ ;

procedure send_good
   $good \leftarrow agent\_view$ 
  when  $good = \{\}$  do
    broadcast to other agents that the problem is solvable;
    terminate this algorithm; end do;
  send ( $good, x_i, good$ ) to its parent

```

図5 Initialize, backtrack, send_good

る (図 6 (e))。

そうすると, x_2 は全ての子から値 1 に対して, $good$ を受信したことになる。そこで x_2 は, (制約違反を起こす可能性が残っている) 値 2 をとり, $ok?$ メッセージを送信する (図 6 (f))。

x_3, x_4 は, この $ok?$ メッセージを受信し, $agent_view$ に値を記録する。 x_3, x_4 とともに制約を満足する値をとることができるため, x_2 へ $good$ メッセージを送信する (図 6 (g))。この $good$ メッセージを x_2 が受信すると, x_2 がとり得る全ての値が制約を満足できるため, x_2 は x_1 へ $good$ メッセージを送信する (図 6 (h))。 x_1 は, 協調的なエージェントであるため, 現在の値を選び, 空集合の $good$ を生成する。空集合の $good$ が導かれたため, 全ての制約を満足する解の存在を示した。

4.5 アルゴリズムの健全性と完全性

本アルゴリズムは, 根エージェントで空集合の $nogood$ が得られれば, 解が存在しないことが示され, 停止する。根エージェントで空集合の $good$ が得られれば, 解が存在することが示され, アルゴリズムは停止する。したがって, 本アルゴリズムの健全性と完全性を示すには以下の 2 点を証明すればよい。

- (1) 新たな $nogood/good$ を生成するための処理は論理的に正しい。すなわち, 根エージェントで空集合の $good$ が得られれば, 解が存在し, 空集合の $nogood$ が得られれば, 解は存在しない。
- (2) 根エージェントで $nogood$ あるいは $good$ の空集合

図 6 アルゴリズムの実行例

が導かれる前に停止したり，処理の無限ループに陥ることは無い．

この 2 点を定理 1，定理 2 として証明する．

〔定理 1〕根エージェントで空集合の *good* が生成されるなら解が存在し，根エージェントで空集合の *nogood* が生成されるなら解は存在しない．

《証明》 新たな *nogood*/*good* を生成するための処理が論理的に正しいことを示す．すなわち，空集合の *good* が生成されれば解が存在し，空集合の *nogood* が生成されれば解が存在しないことを示す．

数学的帰納法を用いてこれを示す．まず，葉エージェントの処理が正しいことを示す．葉エージェントが受信するメッセージは *ok?* メッセージのみである．葉エージェントが協調的なエージェントであるとき，制約を満足する値が一つでもあれば，*good* を生成し，親へ送信する．また，制約を満足する値が一つもなければ，*nogood* を生成し，親へ送信する．ここで生成される *good*/*nogood* は明らかに正しい．

葉エージェントが敵対的なエージェントであるとき，制約違反を起こす値が一つもなければ，*good* を生成し，親へ送信する．また，制約違反を起こす値が一つでもあれば，*nogood* を生成し，親へ送信する．ここで生成される *good*/*nogood* は明らかに正しい．

次に，あるエージェント x_i の子孫エージェントで生成される *good*/*nogood* は全て論理的に正しいと仮定する． x_i で生成される *good*/*nogood* が正しいことを示す．

エージェント x_i が協調的なエージェントである場合を考える． x_i がとり得る各値 $v \in D_i$ に対して，(i) $x_i = v$ と *agent_view* が x_i に関する制約を違反するか (ii) $x_i = v$ と *agent_view* に一致する *nogood* が子から送信されているなら， x_i は *agent_view* と等しい *nogood* を生成す

る．したがって，子から送信された *nogood* が正しいとすると，新たに生成された *nogood* も正しい．また，少なくとも一つの値 $v \in D_i$ について，(i) $x_i = v$ が全ての子から *good* メッセージを受信し，(ii) 各 *good* が $x_i = v$ と *agent_view* と一致し，(iii) $x_i = v$ と *agent_view* が制約を満足するなら， x_i は *agent_view* に等しい *good* を生成する．したがって，子から送信された *good* が正しいとすると，新たに生成された *nogood* も正しい．

次に，エージェント x_i が敵対的なエージェントである場合を考える．少なくとも一つの値 $v \in D_i$ について，(i) $x_i = v$ と *agent_view* が x_i に関する制約を違反するか (ii) $x_i = v$ と *agent_view* に一致する *nogood* が子から送信されているなら， x_i は *agent_view* と等しい *nogood* を生成する．したがって，子から送信された *nogood* が正しいとすると，新たに生成された *nogood* も正しい．また， x_i がとり得る各値 $v \in D_i$ に対して， $x_i = v$ と *agent_view* に一致する *good* を全ての子から受信し，かつ， $x_i = v$ と *agent_view* が制約を満足するなら， x_i は *agent_view* に等しい *good* を生成する．したがって，子から送信された *good* が正しいとすると，新たに生成された *good* も正しい．

上記から，各エージェントでの新たな *nogood*/*good* を生成するための処理は論理的に正しい．したがって，根エージェントで空集合の *good* が生成されれば，解が存在し，根エージェントで空集合の *nogood* が生成されれば，解は存在しない．

〔定理 2〕本アルゴリズムを実行した場合，根エージェントで空集合の *good* あるいは *nogood* が導かれる前に停止したり，処理の無限ループに陥ることはない．

《証明》 定理 2 を証明するために，まずは，あるエージェントが *ok?* メッセージを送信したとき，そのエージェ

ントは子から **good** あるいは **nogood** メッセージを受信することを示す。数学的帰納法を用いて、これを示す。まず初めに、葉エージェントが **good** あるいは **nogood** メッセージを必ず親へ送信することは明らかである。

次に、エージェント x_i の子孫エージェントが受信した **ok?** メッセージに対して、必ず **good** あるいは **nogood** メッセージを送信すると仮定する。このとき、エージェント x_i が必ず親へ **good** あるいは **nogood** メッセージを送信することを示す。

x_i が協調的なエージェントである場合を考える。 x_i は **ok?** メッセージを受信したとき、現在の値と *agent_view* に対する **good** メッセージを全ての子から受信しているなら、 x_i は親へ **good** を送信する。もし、**nogood** を受信しているなら、 x_i は値を変更し、**ok?** メッセージを送信する。子孫エージェントは、このメッセージに対して、必ず **good** あるいは **nogood** を送信する。 x_i の持つ変数の領域は有限であるため、永久に値を変えることはできない。いつかは、 x_i は親へ **nogood** メッセージを送信する。

次に、エージェント x_i が敵対的/仮想的なエージェントである場合を考える。 x_i は **ok?** メッセージを受信したとき、現在の値と *agent_view* に対する **nogood** メッセージを受信しているなら、 x_i は親へ **nogood** を送信する。もし、全ての子から **good** を受信しているなら、 x_i は値を変更し、**ok?** メッセージを送信する。子孫エージェントはこのメッセージに対して、必ず **good** あるいは **nogood** を送信する。 x_i の持つ変数の領域は有限であるため、永久に値を変えることはできない。いつかは x_i は親へ **good** メッセージを送信する。

したがって、根エージェントが **ok?** メッセージを送信すると、根エージェントの子は必ず **nogood** あるいは **good** メッセージを送信する。

根エージェントが協調的なエージェントなら、現在の値と *agent_view* に対して **good** メッセージを全ての子から受信したとき、根エージェントは空集合の **good** を生成する。もし、子から **nogood** を受信しているなら、根エージェントは値を変更し、**ok?** メッセージを送信する。子孫エージェントはこのメッセージに対して必ず **nogood** あるいは **good** を送信する。根エージェントの持つ変数の領域は有限であるため、永久に値を変更することはできない。いつかは、根エージェントが空集合の **nogood** を生成し、アルゴリズムは停止する。

また、根エージェントが敵対的/仮想的なエージェントなら、現在の値と *agent_view* に対して **nogood** メッセージを子から受信したとき、根エージェントは空集合の **nogood** を生成する。もし、全ての子から **good** メッセージを受信しているなら、根エージェントは値を変更し、**ok?** メッセージを送信する。子孫エージェントはこのメッセージに対して必ず **nogood** あるいは **good** を送信する。根エージェントの持つ変数の領域は有限であるため、永久に値を変更することはできない。いつかは、根エー

ジェントが空集合の **good** を生成し、アルゴリズムは停止する。

4.6 ベーシックアルゴリズムの改善

ベーシックアルゴリズムの改善方法について示す。主なアイデアは次の通りである。

- [Hirayama 00] で提案された Resolvent-based Learning を用いて、より小さな **nogood** を生成する。
- 新たな **nogood** を送信するとき、全称限量化された変数を飛ばして、最も近い存在限量化された変数に直接 **nogood** を送信する。

改善アルゴリズムは健全かつ完全である。

§1 Resolvent-based Learning

[Hirayama 00] で紹介された Resolvent-based Learning を用いて、より小さな **nogood** を生成する。

エージェント i が領域 D_i の変数 x_i を持ち、現在の *agent_view* の下で D_i 中の全ての値がいくつかの **nogood** に違反しているとする。エージェント i は各値 $d \in D_i$ について次のようにして **nogood** を一つ選択する。

- d に関する **nogood** の中から最も小さいものを選択する。最も小さい **nogood** が複数あった場合、**nogood** 中の最も優先順位の低い変数の優先順位を比較し、その中で優先順位が最も高いものを選択する。

その後、選択した全ての **nogood** の和集合から、 x_i を除去することで新たな **nogood** を生成する。

ベーシックアルゴリズムでは、*agent_view* そのものを新たな **nogood** として生成していた。そのため、違反の原因でない変数が **nogood** に含まれている可能性があり、親が違反の原因でないときでも新たな **nogood** は親に送信される。しかし、Resolvent-based Learning を用いることで、違反の原因でない変数をより含まない小さな **nogood** を生成し、親でない先祖へ直接メッセージを送ることができる。したがって、アルゴリズムの効率は改善される。

§2 全称限量化された変数の省略

新たな **nogood** の受信者が全称限量化された変数であるとき、その全称限量化された変数/仮想的なエージェントを飛ばして **nogood** を送ることができる。**nogood** 中の最も優先順位の低いエージェントが全称限量化されているときは、**nogood** からその変数を除き、新たな **nogood** 中の最も優先順位の低い変数を持つエージェントへ **nogood** メッセージを送ることができる。**nogood** メッセージを受信した敵対的なエージェントは、**nogood** から自身を除いて新たな **nogood** を親へ送るため、この処理は正しい。

また、この処理は、**nogood** 中の最も優先順位の低い変数が全称限量化されている変数である限り、繰り返し行える。その結果、必要なサイクル数を減らすことができる。

5. 実験

本章では、ベーシックアルゴリズムと改善アルゴリズムとの比較実験について示す。

各エージェントが自分自身のクロックを管理する離散イベントシミュレーションを用いる。一つの計算サイクルを実行するごとにエージェントのクロックを一つ増やす。一サイクルは、送信されたいくつかのメッセージを受信し、局所的な計算を行い、他のエージェントに対してメッセージを送信するという処理の単位である。時刻 t で送信されたメッセージは時刻 $t+1$ で受信される。本実験では、問題を解くのに必要なサイクル数を用いて分析する。一サイクルは、各エージェントが外界の状況を認識し、その状況に対応する行動を選択し、それを通信するという処理の単位に対応している。

実験で用いたインスタンスは [Gent 05] で用いられたものを参考に生成した。このモデルでは、変数が三つのブロックに分かれて限量化されている。最初と最後のブロックは存在限量化された変数で構成され、真ん中のブロックは全称限量化された変数で構成される。インスタンスは、7つのパラメータを設定した後、ランダムに生成される。7つの各パラメータ $\langle n, n_{\forall}, n_{pos}, d, p, p_{\exists\exists}, p_{\forall\exists} \rangle$ はそれぞれ次の意味を持つ。 n は変数の数を、 $n_{\forall}$ は全称限量化された変数の数を、 n_{pos} は限量記号のシーケンス Q 中の最初の全称限量化された変数の位置を表す。また、 d はドメインサイズ (全ての変数で同じ) を表わし、 p は可能な二項制約のうち、実際に制約として設ける数の割合を表わす。制約は *nogood* で与えられるとする。 $p_{\exists\exists}$ は、 $\exists x_i \exists x_j c_{ij}$ の形の制約の数をその形の可能な全制約との割合で表わす。 $p_{\forall\exists}$ は、 $p_{\exists\exists}$ と似た意味を持ち、 $\forall x_i \exists x_j, c_{ij}$ の形の制約の数に関するパラメータである。その他二つの形の制約は前処理で除かれるため、生成しない。

$\forall x_i \exists x_j c_{ij}$ の形の制約が多いと、生成されるインスタンスの多くが解決不可能な問題となってしまう。そこで、解決可能な問題を十分に生成するため、次のようにしてこの形の制約を制限する [Gent 05]。

まず、一つのドメインから他のドメインへランダムな全単射を生成する。この全単射以外の組合せを制約から除く。そして、全単射のうち、 $p_{\forall\exists}$ の割合の組合せを $\forall x_i \exists x_j, c_{ij}$ の形の制約とする。

本実験では、 $p_{\exists\exists}$ 以外のパラメータを $n = 15, n_{\forall} = 5, n_{pos} = 6, d = 5, p = 0.35, p_{\forall\exists} = 0.5$ で固定して行った。 $p_{\exists\exists}$ は 0.10 から 0.90 まで 0.05 間隔で設定した。図 7 はベーシックアルゴリズムと改善アルゴリズムの問題解決に必要なサイクル数を示している。各データポイントは 100 のインスタンスの平均を表す。

$p_{\exists\exists}$ の変化によって、問題の難しさが *easy-hard-easy* と遷移することが分かる。ただし、 $p_{\exists\exists}$ が小さい時 (制約が弱い時)、問題は $p_{\exists\exists}$ が大きい場合 (制約が強い時) に

図 7 Required Cycles ($n = 15, n_{\forall} = 5, n_{pos} = 6, d = 5, p = 0.35, p_{\forall\exists} = 0.5$)

比べると難しい。この理由は、制約が弱い場合であっても、エージェントのチームは全称限量化された変数のとり得る全ての値の割当について探索を行う必要があるためである。

また、ベーシックアルゴリズムの改善が効果的であることがわかる。制約が強くなるにつれ生成される *nogood* の数も増えるため、特に $p_{\exists\exists}$ が大きいときに改善の効果がみられる。最も難しい問題 ($p_{\exists\exists}$ が 0.3) においては、改善アルゴリズムは 25% のサイクルを削減している。

6. 結論

本論文では、分散制約充足問題を一般化した限量記号付き分散制約充足問題を提案した。本問題はエージェントのチームが敵に対抗するプランを構築する状況を定式化した問題である。また、本論文では、非同期バックトラッキングアルゴリズムを拡張した限量記号付き分散制約充足問題を解くアルゴリズムを提案し、その改善方法を示した。評価実験では、制約の密度によって、問題の難しさが *easy-hard-easy* と転移することが分かった。ただし、エージェントのチームが全称限量化された変数の可能な全ての値の割当を考える必要があるため、制約の密度が低い問題は、制約の密度が高い問題よりも難しい。

また、ベーシックアルゴリズムの改善は効果的で、最も難しい問題においてサイクル数を 25% 削減できた。

今後の課題としては、限量記号付き分散制約充足問題を解くより効率的なアルゴリズムの開発、完全な解が得られなくとも期限までに行動を決定する実時間アルゴリズムの開発が挙げられる。

謝辞

本研究を進めるにあたり、日本学術振興会科学研究費補助金基盤研究 (A) (課題番号 20240015) の助成を受け

ました．ここに深く感謝いたします．

◇ 参 考 文 献 ◇

- [Benedetti 05] Benedetti, M.: sKizzo: A Suite to Evaluate and Certify QBFs, in *Proceedings of 20th International Conference on Automated Deduction (CADE05)*, pp. 369–376 (2005)
- [Bessiere 01] Bessiere, C., Maestre, A., and Meseguer, P.: Distributed Dynamic Backtracking, in *Proceedings of Constraint Programming (CP01)*, p. 772 (2001)
- [Chen 04] Chen, H. M.: *The computational complexity of quantified constraint satisfaction*, PhD thesis, Cornell University (2004)
- [Davis 62] Davis, M., Logemann, G., and Loveland, D.: A machine program for theorem-proving, *Communications of the ACM*, Vol. 5, No. 7, pp. 394–397 (1962)
- [Faltings 05] Faltings, B. and Macho-Gonzalez, S.: Open Constraint Programming, *Artificial Intelligence*, Vol. 161, No. 1–2, pp. 181–208 (2005)
- [Fernández 02] Fernández, C., Béjar, R., Krishnamachari, B., and Gomes, C. P.: Communication and Computation in Distributed CSP Algorithms, in *Proceedings of Constraint Programming (CP02)*, pp. 664–679 (2002)
- [Gent 05] Gent, I. P., Nightingale, P., and Stergiou, K.: QCSP-Solve: A Solver for Quantified Constraint Satisfaction Problems, in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI05)*, pp. 138–143 (2005)
- [Giunchiglia 01] Giunchiglia, E., Narizzano, M., and Tacchella, A.: QUBE: A System for Deciding Quantified Boolean Formulas Satisfiability, in *Proceedings of International Joint Conference on Automated Reasoning (IJCAR01)*, pp. 364–369 (2001)
- [Hirayama 00] Hirayama, K. and Yokoo, M.: The Effect of Nogood Learning in Distributed Constraint Satisfaction, in *Proceedings of the Twentieth International Conference on Distributed Computing Systems (ICDCS00)*, pp. 169–177 (2000)
- [Mackworth 92] Mackworth, A. K.: Constraint Satisfaction, in Shapiro, S. C. ed., *Encyclopedia of Artificial Intelligence*, pp. 285–293, John Wiley & Sons (1992)
- [Mailler 06] Mailler, R. and Lesser, V.: Asynchronous Partial Overlay: A New Algorithm for Solving Distributed Constraint Satisfaction Problems, *Journal of Artificial Intelligence Research*, Vol. 25, pp. 529–576 (2006)
- [Verfaillie 94] Verfaillie, G. and Schiex, T.: Solution Reuse in Dynamic Constraint Satisfaction Problems, in *Proceedings of the 12th National Conference on Artificial Intelligence*, pp. 307–312 (1994)
- [横尾 92] 横尾 真, エドモンド H. ダーフィ, 石田 亨, 桑原 和宏: 分散制約充足による分散協調問題解決の定式化とその解法, *電子情報通信学会論文誌*, Vol. J-75 D-I, No. 8, pp. 704–713 (1992)

〔担当委員: 山下 倫央〕

2009 年 12 月 18 日 受理

—— 著 者 紹 介 ——

馬場 里美(学生会員)

2009 年 3 月九州大学工学部電気情報工学科卒業．現在，同大学院システム情報科学府修士課程在籍中．制約充足問題 / 分散制約充足問題に関する研究に興味を持つ．

岩崎 敦(正会員)

2002 年神戸大学大学院自然科学研究科博士課程修了．同年より 2004 年まで NTT コミュニケーション科学基礎研究所に勤務．2004 年より九州大学大学院システム情報科学研究院助教．ゲーム理論，学習，オークション，実験経済学に関する研究に従事．博士(学術)．

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，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．情報処理学会，電子情報通信学会，日本ソフトウェア科学会，AAAI 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 年同大学院基礎工学研究科博士前期課程修了．1995 年同大学院基礎工学研究科博士後期課程修了．1995 年神戸商科大学助手．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(神戸大学と神戸商科大学の統合による)．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．博士(工学)．2010 年 IFAAMAS Influential Paper Award 受賞．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．

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．