論　文

分散制約最適化問題の非同期分散探索における上下界の導出と学習を用いた効率改善手法

松井　俊浩[†]　　松尾　啓志[†]　　岩田　彰[†]

An efficient method for asynchronous distributed constraint optimization using upper and lower bound

Toshihiro MATSUI[†], Hiroshi MATSUO[†], and Akira IWATA[†]

あらまし
分散制約最適化問題 (DCOP) は非同期分散環境における問題解決を扱うものとして，マルチエージェント分野で重要な位置づけにある．本稿では，DCOP の解法として最近提案された分枝限定法に基づく非同期分散探索手法に対し，基本的な効率改善手法を提案する．従来手法は深さ優先探索木に配置されたノード (エージェント) によるコスト計算を基礎として非同期分散探索を実現しているが，バックトラック処理に効率化の余地がある．また非同期性により同一解を再探索する機会が多い．提案手法はバックトラックにおけるオーバヘッドの削減のために，上下界に関連する解の導出および上位ノードへの下界通知のショートカットを導入する．また，再探索の削減のために，探索済みの上下界情報の学習を導入する．計算機実験により提案手法の有効性を評価する．

キーワード
分散制約最適化問題，　分散制約充足問題，　マルチエージェント，　分枝限定法

1. ま え が き

分散制約最適化問題 (DCOP) [5] [7] は，制約最適化問題 (COP) [1] [2] を分散探索問題に適用したものであり，従来の分散制約充足問題 (DCSP) [3] では記述不能な最適化問題を扱うものとして，マルチエージェント分野で重要な位置づけにある．

過制約な制約充足問題に対して，違反コストを最小にする最適解を求める基本的な解法としては，分枝限定法による手法が提案されている [2] が，DCSP を DCOP に拡張する上では近似解法や従来の DCSP のアルゴリズムを拡張する試みが主体であった [4] [5] [6]．これに対し，DCOP の完全解法として分枝限定法に基づく分散アルゴリズムが提案された [8] [9]．この手法は深さ優先探索木に配置されたノード (エージェント) によるコスト計算を基礎として，非同期並列探索を実現している．また，大域的なアルゴリズムの停止までの機構を持つこと，ユーザが指定したコストでの探索の打ち切りが可能であることなど，分散探索において基礎的な概念の幾つかが統合されている．さらに，記憶複雑度が多項式オーダーである利点がある．

しかし，上記手法はバックトラック処理に効率化の余地がある．また，記憶の制限と非同期性により同一解を再探索する機会が多い．本稿ではバックトラックにおけるオーバヘッドの削減のために，上下界に関連する解の導出および上位ノードへの下界情報の通知のショートカットを導入する．また，再探索の削減のために，探索済みの上下界情報の学習を導入する．

以下では先ず 2. で従来手法の概要を示す．次に提案手法として，3. においてバックトラック効率改善のための上下界の導出と下界情報通知のショートカット，さらに 4. において探索済みの上下界情報の学習について述べる．5. で提案手法の最適性について述べ，6. において計算機実験による評価を示す．

[†] 名古屋工業大学，　名古屋市
Nagoya Institute of Technology, Gokiso-cho,Showa-ku,Nagoya-shi, 466-8555,Japan

2. 従来手法 — 分枝限定法に基づく非同期分散探索手法 (ADOPT)

まず，従来手法である分枝限定法に基づく非同期分散探索手法 [8] を示す．対象とする問題は次のとおりである．各ノード (エージェント)i は単一の変数 x_i を持ち，変数 x_i は値域 D_i について $d \in D_i$ の値をとる．各変数は 2 項制約で他の変数 x_j(ノード j の変数) と関係する．各制約に対応するコスト関数 $f_{i,j}(d_i, d_j)$ により，値のペアに対するコストが評価される．各ノードは自分の変数に関連する制約およびコスト関数を知る．また，変数値の選択はその変数を持つノードのみが行う．以下では必要に応じて変数名をノード名と区別せずに表記する。探索するべき最適解はコスト評価の総和 (結合) が最小となる解である．各ノードは任意の他ノードとメッセージ通信を行う．通信においては同一ノード間のメッセージ順序は保存されるが，他ノード間のメッセージ順序は保存されない．

この手法では，2 項制約に対応する制約網について，事前に深さ優先探索木が生成されていることを前提とする．深さ優先探索木では各サブツリー間に制約辺が存在しないため，サブツリー間の処理に並列性が得られる．木を生成する際のノードの展開順序には，最も近傍数が多いものを優先するヒューリスティクスを用いる．深さ優先探索木のための分散アルゴリズムについての文献は [15], [16] [17] などがある．ノード i の親ノードを $parent_i$，子ノードの集合を $children_i$，また，木に含まれない制約辺も含めた上位/下位近傍の集合を $upperNeighbors_i/lowerNeighbors_i$ で表す．

アルゴリズムの基本的な動作は次のとおりである．(1) 各ノードは解のコストと自ノードに分配されたコストに基づいて自変数の値を選択し，制約で関連する下位ノードに同報 (VALUE 送信) する．(2) 各ノードは解のコストを親ノードに通知 (COST 送信) する．(3) 各ノードは解のコストに基づいて，自ノードと子ノードでのコストの分配を決定し，子ノードに通知 (THRESHOLD 送信) する．以上の動作を反復し，最終的に根ノードではコストの上下界が一致する．このとき，根ノードは自変数の値を最適解に固定し子ノードに終了を通知 (TERMINATE 送信) する．子ノードは同様に最適解を発見し再帰的に全体が終了する．

以下にアルゴリズムの構成要素を示す．本稿ではアルゴリズムの本質的な部分を整理するため，文献 [8] のルールおよび表現を修正した．

2.1 変　　数

各ノード i は自分の状態に関わる次の変数を持つ．

- d_i: 変数値
- $threshold_i$: 変数値を変更するコストの閾値

初期値は $d_i \in D_i$, $threshold_i = 0$ である．d_i はコスト評価に応じて変更され，後述の VALUE メッセージにより，関連ノードに送信される．$threshold_i$ は，後述の THRESHOLD メッセージによりノード i 以下のサブツリーに割り当てられたコスト値であり，変数値を変更する基準となる．

また，ノード i は上位ノードの解のキャッシュとして

- $currentContext_i$:上位ノードの解

を持つ．初期値は $currentContext_i = \{\}$ である．後述の VALUE および COST メッセージ受信処理により上位ノードの解が格納される．

下位ノードについては，コスト評価のキャッシュとして，$d \in D_i$,$x \in children_i$ について次の情報を持つ．

- $context_i(d, x)$:子ノードの上位ノードの解
- $lb_i(d, x)$:子ノードのコスト下界値
- $ub_i(d, x)$:子ノードのコスト上界値
- $t_i(d, x)$:子ノードの値変更閾値

それぞれの初期値は $context_i(d, x) = \{\}$, $lb_i(d, x) = 0$, $ub_i(d, x) = \infty$, $t_i(d, x) = 0$ である．後述の COST メッセージ受信処理により子ノードのコスト評価が $context_i(d, x)$, $lb_i(d, x)$, $ub_i(d, x)$ に格納される．また，$t_i(d, x)$ はノード i がノード x に対して設定し，後述の THRESHOLD メッセージにより子ノードに送信される．

2.2 コストの評価

各ノード i が知る現在の (部分) 解に関するコストの評価は以下のように定義される．

[局所コスト $\delta_i(d)$]

ノード i の，$d \in D_i$ についての，上位近傍に対するコスト評価の総和を局所コスト $\delta_i(d)$ とする．各ノードが上位近傍の制約のみを評価することにより，評価の重複が避けられる．

$$\delta_i(d) = \sum_{\substack{(x_j, d_j) \in currentContext_i, \\ x_j \in upperNeighbors_i}} f_{i,j}(d, d_j)$$

[各変数値についての上下界 $LB_i(d), UB_i(d)$]

ノード i を根とするサブツリーの $d \in D_i$ についての上下界 $LB_i(d), UB_i(d)$ は，局所コストと各子ノードの上下界値の総和となる．

$$LB_i(d) = \delta_i(d) + \sum_{x \in children_i} lb(d,x)$$
$$UB_i(d) = \delta_i(d) + \sum_{x \in children_i} ub(d,x)$$

[**上下界** LB_i, UB_i]
ノード i を根とするサブツリーの上下界 LB_i, UB_i は，$d \in D_i$ に対する上下界の最小値になる．

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

2.3 変数に関する条件

各ノード i は変数について以下の条件を維持しなければならない．
[$context_i(d,x)$ **条件**]
子ノードについての $context_i(d,x)$ と $currentContext_i$ が矛盾する場合，$context_i(d,x)$ に付随する $lb_i(d,x)$, $ub_i(d,x)$, $t_i(d,x)$ は無効である．従って $context_i(d,x)$ と $currentContext_i$ は矛盾してはならない．すなわち両者に含まれる同一の変数の値は異なってはならない．

$\forall d \in D, x \in children,$
$context_i(d,x)$ と $currentContext_i$は矛盾しない

もしも矛盾する場合は，$context_i(d,x)$, $lb_i(d,x)$, $ub_i(d,x)$, $t_i(d,x)$ を初期値にリセットする．
[$t_i(d,x)$ **条件 (ChildThresholdInvariant)**]
子ノードの閾値は子ノードの上下界を超えてはならない．

$\forall d \in D, x \in children_i,$
$lb_i(d,x) \leqq t_i(d,x) \leqq ub_i(d,x)$

もしも超える場合には $t_i(d,x)$ の値を上界または下界に制限する．
[$threshold_i$ **条件 (ThresholdInvariant)**]
自ノードの閾値は自ノードの上下界を超えてはならない．

$$LB_i \leqq threshold_i \leqq UB_i$$

もしも超える場合には $threshold_i$ の値を上界または下界に制限する．
[d_i **条件**]
変数値に対する下界の評価値は閾値を上回ってはならない．ただし，閾値と上界が等しい場合は，上界に対応する変数値でなければならない．

$$\begin{cases} UB_i(d_i) = UB_i & if \ \ threshold_i = UB_i \\ LB_i(d_i) \leqq threshold_i & otherwize \end{cases}$$

下界の評価値が閾値より大きい場合，$LB_i(d_i) = LB_i$ となるよう，d_i を変更する．これは最良解優先探索を意図した値変更である．また，下界の評価値が閾値以下であれば値変更しないことにより，バックトラックが抑制される．
[$t_i(d,x)$ **条件 (AllocationInvariant)**]
自ノードの閾値は自ノードを根とするサブツリーに割り当てられたコストである．従って，自ノードの局所コストと子ノードの閾値の総和と等しくなければならない．

$$threshold_i = \delta_i(d_i) + \sum_{x \in children_i} t_i(d_i, x)$$

もしも等しくない場合には任意の $t_i(d,x)$ を増減する．ただし，[$t_i(d,x)$ 条件 (ChildThresholdInvariant)] に矛盾してはならない．

2.4 メッセージ送信

各ノードは他ノードに以下のメッセージを送信する(注1)．
[**VALUE 送信**]
送信内容: (VALUE,(x_i, d_i))
送信する条件: d_i が更新された
送信先: $x \in lowerNeighbors_i$
[**COST 送信**]
送信内容:
(COST,x_i,$currentContext_i$,LB_i,UB_i)
送信する条件:
($\forall x \in upperNeighbors_i$ より VALUE を受信した) $\wedge$ (d_i,$currentContext_i$, $threshold_i$,
子ノードキャッシュのいずれかが更新された)
$\wedge$ (TERMINATE 未受信)
送信先: $parent_i$
[**THRESHOLD 送信**]
送信内容:
(THRESHOLD,$currentContext_i$,$t_i(d_i,x)$)
送信する条件:

(注1)：文献 [8] にはメッセージの厳密な送信条件が明示されていない．本稿では文献とほぼ同程度の平均メッセージ数となるよう，アルゴリズムの動作に支障のない条件を設けた．

d_i,$currentContext_i$, 子ノードキャッシュのいずれかが更新された
送信先: $x \in children_i$

[**TERMINATE 送信**] (注2)
送信内容:
(TERMINATE,
$currentContext_i \cup (x_i, d_i), t_i(d_i, x))$
送信する条件:
終了条件が成立した (後述)
送信先: $x \in children_i$

2.5 メッセージ受信

各ノードはメッセージ受信時に以下の処理を実行する.

[**VALUE 受信**]
(VALUE,(x_k, d_k)) 受信時:
if TERMINATE 未受信 **then**
$currentContext_i$ に (x_k, d_k) を加える,
または置換する.
endif

[**COST 受信**]
(COST,x_k,$context_k$,lb_k,ub_k) 受信時:
$d \leftarrow (x_i, d) \in context_k$ である d
$context_k$ から (x_i, d) を除く.
if TERMINATE 未受信 **then** (2.5-A)
$\forall (x_j, d_j) \in context_k, x_j \notin upperNeighbors_i$ について,$currentContext_i$ に
(x_j, d_j) を加える, または置換する.
endif
[$context_i(d, x)$ 条件] を維持する. (2.5-B)
if $context_k$ と $currentContext_i$ が整合する **then** (注3) (2.5-C)
$\forall (x_j, d_j) \in context_k$ について,
$context_i(d, x_k)$ に (x_j, d_j) を加える,
または置換する.
if $lb_k > lb_i(d, x_k)$ **then**
$lb_i(d, x_k) \leftarrow lb_k$ **endif**
if $ub_k < ub_i(d, x_k)$ **then**
$ub_i(d, x_k) \leftarrow ub_k$ **endif**
endif
(上記 (2.5-B) の [$context_i(d, x)$ 条件] の維持により, $context_i(d, x)$ が $currentContext_i$ と矛盾する場合は, $context_i(d, x)$, $lb_i(d, x)$, $ub_i(d, x)$, $t_i(d, x)$ を初期値にリセットする)

[**THRESHOLD 受信**]
(THRESHOLD,$context_k$,t_k) 受信時:
if $context_k$ と $currentContext_i$ が整合する **then** $threshold_i \leftarrow t_k$ **endif**

[**TERMINATE 受信**]
(TERMINATE,$context_k$,t_k) 受信時:
$currentContext_i \leftarrow context_k$
$threshold_i \leftarrow t_k$
TERMINATE 受信を記録する

2.6 最適解の検出およびアルゴリズムの終了

根ノード i ではいずれ $LB_i = threshold_i = UB_i$ となる. このとき, 根ノードは変数値を最適解に固定し, 終了することができる. ノード j の上位ノードが変数値を固定して終了した後で, $threshold_j = UB_j$ となれば, ノード j は変数値を最適解に固定し, 終了することができる. したがって最適解の検出および終了の判定は次のようになる.

[**終了判定**]
if $threshold_i = UB_i \wedge$
(i が根ノード$\vee$TERMINATE を受信済)
then
([d_i 条件] により $UB_i(d_i) = UB_i$)
終了条件成立を記録する
endif

2.7 全体的な処理の構成

以上の各条件の維持およびメッセージ送受信の処理を, 手続き的に構成する場合の順序は一意では無い. 本稿では各ノードは, メッセージ受信, 各条件の維持/判定, メッセージ送信を反復するサイクル動作とした.

[初期化]
while 終了条件が成立していない **do**
(メッセージ受信)
while 受信メッセージ有り $\wedge$
終了条件が成立していない **do**

(注2): 文献 [8] では, 最後に親ノードで決定された $t_i(d_i, x)$ が子ノードの $currentContext_x$ と矛盾せずに受信されることが, アルゴリズムの最適性の条件の一部となっていると考えられる. 本稿ではこの条件を明確にするため TERMINATE メッセージに $t_i(d_i, x)$ を加えた.
(注3): 文献 [8] では, $context_k$ と $currentContext_i$ が整合すれば $context_i(d, x_k)$, $lb_i(d, x_k)$, $ub_i(d, x_k)$ にそれぞれ $context_k$,lb_k,ub_k を代入する記述となっている. しかし, 複数のノードにおける子ノードキャッシュのリセットとメッセージの遅延により, $currentContext_i$ と整合する $context_k$,$lb_k < lb_i(d, x_k)$,$ub_i(d, x_k) < ub_k$ なる COST メッセージを受信し, コスト評価の単調性が維持されなくなる可能性があると考えられる. これはその後の摂動により解消されると考えられるが, 本稿では, $currentContext_i$ と整合するかぎり, $context_i(d, x_k)$, $lb_i(d, x_k)$, $ub_i(d, x_k)$ の単調性が維持できるよう手続きを修正した. 以上の修正による探索効率への影響は大きくないことを予備実験により確認した.

[VALUE 受信] **or** [COST 受信] **or**
[THRESHOLD 受信] **or**
[TERMINATE 受信]
enddo
(各条件の維持/判定を以下の順で実行)
[$context_i(d,x)$ 条件]
[$t_i(d,x)$ 条件 (ChildThresholdInvariant)]
[$threshold_i$ 条件 (ThresholdInvariant)]
[d_i 条件]
[$t_i(d,x)$ 条件 (AllocationInvariant)]
[終了判定]
(メッセージ送信)
[VALUE 送信]
[COST 送信]
[THRESHOLD 送信]
[TERMINATE 送信]
enddo

3. 提案手法 1 — 上下界の導出と COST メッセージのショートカット

上記のアルゴリズムは深さ優先探索木によるコスト計算を基礎としている．これにより，非同期分散探索の最適解への収束が保証される．しかし，単純なバックトラック動作による処理の冗長性が問題となる．本節では，バックトラックにおけるオーバヘッドの削減を目的とした，上下界の導出と COST メッセージのショートカットによる効率化手法を提案する．提案手法では，(1) 各ノードは根ノードまでの経路に関する知識を持つこと，(2) 制約辺に沿わない任意の通信経路が存在することを前提とする．また，議論を簡単にするため，(3) 終了ノードへのメッセージは無視されるものとする．

3.1 上下界についての部分解の分離

従来手法では，子ノードについての上位ノードの解のキャッシュとして $context_i(d,x)$ を保持している．この情報は $lb_i(d,x)$,$ub_i(d,x)$ の根拠となる部分解である．しかし，下界の証明に必要な部分解は，上位ノード全ての変数割り当てを含まない場合もある．そこで，$context_i(d,x)$ を，$context_i^{lb}(d,x)$ および $context_i^{ub}(d,x)$ に分離し，$lb_i(d,x)$,$ub_i(d,x)$ の証明に必要な最低限の部分解を導出する．これは DCSP における違反解の導出 [10] [11] に類似する操作であると考えられる．もしも，下界に関する部分解に親ノードが含まれない場合，深さ優先探索木をショートカットしたコスト通知を行うことで，バックトラックにおける冗長性を改善できる．これは COP における Backjumping [2] に類似する操作であると考えられる．

3.2 部分解の導出

変数集合 $X'_d \subseteqq upperNeighbors_i$, $X''_d \subseteqq children_i$ について，

$$\forall d \in D_i,$$
$$LB_i \leqq \sum_{\substack{x_j \in X'_d,(x_j,d_j)\in \\ currentContext_i}} f_{i,j}(d,d_j) + \sum_{x\in X''_d} lb_i(d,x)$$

であるとき下界に関わる部分解 $context_i^{lb}$ は，

$$\bigcup_{d\in D_i} (\{(x_j,d_j) \in currentContext_i | x_j \in X'_d\} \cup \bigcup_{x\in X''_d} context_i^{lb}(x,d))$$

である．

このような部分解の導出の基準としては，(1) 小さいサイズの部分解であること，(2) より上位のノードのみを含むことなどが有効であると考えられる．本稿では簡単な導出手法として次の方法を用いた．

各変数値 $d \in D_i$ について，まず局所コストについて評価し，$f_{i,j}(d,d_j) > 0$ なる上位近傍 j の変数 x_j を X'_d に加え，$f_{i,j}(d,d_j)$ をコストとして合計する．このときコストが LB_i に達した時点で導出を終了する．全ての上位近傍についてのコストの合計が LB_i に満たない場合は，各子ノード $x \in children_i$ について $lb_i(d,x) > 0$ なる x を X''_d に加え，$lb_i(d,x)$ をコストとして合計する．このときコストが LB_i に達した時点で導出を終了する．

一方，上界に関わる部分解 $context_i^{ub}$ はノード i 以下のサブツリーに関わる全ての制約に依存するため，

$$\{(x_j,d_j) \in currentContext_i | x_j \in upperNeighbors_i\} \cup \bigcup_{d\in D_i, x\in children_i} context_i^{ub}(x,d)$$

である．従って，メッセージの遅延と，子ノードのキャッシュがリセットされることを無視すれば，ほぼ $currentContext_i$ と等しい．

3.3 COST メッセージの変更

上下解について分離された部分解を送信するため，COST メッセージの形式および送信先を変更する．親ノードへの COST メッセージは上下界に関する部分解を含み，次のようになる．

(COST,x_i,$currentContext_i$,
$context_i^{lb}$,$context_i^{ub}$,LB_i,UB_i)

もしも，$context_i^{lb}$ が親ノードを含まない場合，親ノードへの COST メッセージに加えて，下界に関する COST メッセージをショートカットして送信する．ただし，$LB_i = 0$ の場合は明らかに冗長なので送信しない．送信先 j は $context_i^{lb}$ に含まれる最下位の変数 x_j を持つノードとする．また，メッセージ中の変数 x_k は j の子ノードで i の祖先ノードのものとする．UB_i については上界および原因となる部分解を証明できないため，デフォルトの上界値 ∞ および部分解 {} とする．送信される COST メッセージは次のようになる．

(COST,x_k,$context_i^{lb}$, $context_i^{lb}$,{},LB_i,∞)

COST メッセージ受信処理の上下界の更新 (2.5-C) において，部分解を別に記録するよう変更する．下界の更新については以下のようになる．上界についても同様に $ub_i(x,d)$ が単調に減少するように更新する．

if $context_k$ と $currentContext_i$ が整合する
then
 if $context_k^{lb}$ が (x_i, d') を含む **then**
  $d' \leftarrow (x_i, d') \in context_k^{lb}$ である d'
  $context_k^{lb}$ から (x_i, d') を除く．
 endif
 $context_k^{lb}$ が (x_i, d') を含んでいた場合
  $d = d'$ について，
  そうでなければ $\forall d \in D_i$ について
   if $lb_k > lb_i(d, x_k)$ **or** $(lb_k = lb_i(d, x_k)$
   and $|context_k^{lb}| < |context_i^{lb}(d, x_k)|)$
   then
    $lb_i(d, x_k) \leftarrow lb_k$
    $context_i^{lb}(d, x_k) \leftarrow context_k^{lb}$
   endif
endif

また，[$context_i(d,x)$ 条件] も上下界に分離し，それぞれ $context_i^{lb}(d,x)$ と $lb_i(d,x)$，および $context_i^{ub}(d,x)$ と $ub_i(d,x)$ について別に $currentContext_i$ との整合性を維持する．$t_i(d,x)$ は $lb_i(d,x)$ により押し上げられる形で増加するため，$context_i^{lb}(d,x)$ が不整合の場合は $lb_i(d,x)$ とともにリセットする．

4. 提案手法 2 — 上下界の学習

従来手法の 1 つの利点は記憶複雑度が多項式オーダである点である．各ノード i は，各変数値 d に対する子ノード x の上位ノードの解を $context_i^{lb}(d,x)$ または $context_i^{ub}(d,x)$ (以下 $context_i^{lb/ub}(d,x)$ と表記) により記録し，現在の解と矛盾する場合には初期値にリセットする．しかし，探索の間にリセットが頻発することは，探索回数において大きなオーバヘッドとなる．十分な記憶が利用可能であれば，これまでに得られた部分解のコストを記録することで探索回数を削減することが適当である．そこで，本節ではコスト情報のリセットの回数を削減するための上下界の学習を導入する．このような手法では，CSP/DCSP における違反解の記録 [12] [13] [14] と同様に，理想的には指数オーダの記憶が必要となる点が問題となる．しかし，従来手法の完全性により，許容できるサイズの記憶で妥協し，補助的な効率化手法として導入することができる．

4.1 LRU キャッシュの追加

各ノード i の，$context_i^{lb/ub}(d,x)$ を複数記録することでリセットの回数を削減できる．しかし，これまでに探索した部分解を全て記録することは不可能であるため，LRU により複数の部分解を管理する．$context_i^{lb/ub}(d,x)$ ごとに有限長のバッファを設け，先頭要素を $context_i^{lb/ub}(d,x)$ として用いる．$currentContext_i$ と先頭要素が矛盾する場合には，バッファ内に矛盾しない要素が存在すればそれを先頭に移動し，存在しなければ新規要素を先頭に挿入する．制限長を超えた要素は削除される．

4.2 解の内包関係に基づく統合

LRU による記憶を追加した場合であっても，キャッシュ内の整合性のある上下界情報 $context_i^{lb/ub}(d,x)$, $lb/ub_i(d,x)$ を上書きする形で変更する方法では，これまでに得られた上下界の情報が失われる機会が多い．整合性のある複数の上下界情報により，より広い探索空間の上下界を記録することが望ましいと考えられる．そこで，解と上下界の内包関係にもとづいて解を統合する．すなわち，上下界 $context^{lb/ub}$, lb/ub および $context'^{lb/ub}$, lb'/ub' について，

$$lb \geqq lb' \wedge context^{lb} \subseteqq context'^{lb}$$
$$ub \leqq ub' \wedge context^{ub} \subseteqq context'^{ub}$$

であれば，$context^{lb/ub}, lb/ub$ は $context'^{lb/ub}, lb'/ub'$

を置換する．そうでなければ置換されず，両者とも記録される．

COST メッセージ受信において上記の条件によるキャッシュ要素の置換または追加を行う．また，コスト評価で用いるキャッシュ内の先頭要素は，$currentContext_i$ と整合し上下界を最も狭くするものを選択する．

5. アルゴリズムの最適性について

提案手法では従来手法に対し，上下界に対応した部分解の導出，下界の通知のショートカット，複数の上下界の記録および統合を追加した．これらは従来手法の最適性を損なわない．

部分解の導出は，上界については従来手法と同様である．下界の部分解 $context_i^{lb} \subseteqq currentContext_i$ の導出については，$X'_d \subseteqq upperNeighbors_i, X''_d \subseteqq children_i$ から，

$$\forall d \in D_i,$$
$$\sum_{\substack{x_j \in X'_d, xj \in \\ currentContext_i}} f_{i,j}(d, d_j) + \sum_{x \in X''_d} lb_i(d, x) \leqq$$
$$\sum_{\substack{x_j \in upperNeighbors_i, \\ xj \in currentContext_i}} f_{i,j}(d, d_j) + \sum_{x \in Children} lb_i(d, x) =$$
$$\delta(d) + \sum_{x \in Children} lb_i(d, x) = LB_i(d)$$

であり．下界の定義 $LB_i = \min_{d \in D_i} LB_i(d)$ および導出の条件，

$$\forall d \in D_i, LB_i \leqq \sum_{\substack{x_j \in X'_d, xj \in \\ currentContext_i}} f_{i,j}(d, d_j) + \sum_{x \in X''_d} lb_i(d, x)$$

から，$context_i^{lb}$ のみについての下界のコスト評価は，従来手法による $currentContext_i$ についての下界 LB_i に等しい．

下界のショートカットについては次のとおりである．ノード l から祖先ノード u に，$context_l^{lb}$ がショートカットして通知されたとする．ノード u と l の間のノードの持つ変数の集合を $X_{u,l}$ で表す．各ノードが評価する評価関数の集合 F_* のうち，$X_{u,l}$ に関するものからなる部分集合を $F_{u,l}$，その補集合を $\overline{F}_{u,l}$ で表す．ノード i の変数値 d，評価関数の集合 F，部分解 $context$ によりコストの一部が評価されることを $(d_i, F, context)$ で表す．各評価関数は 1 度のみ評価され加算されるため，ノード i の $LB_i(d)$ の定義は，次のように，$F_{u,l}$ および $\overline{F}_{u,l}$ についての評価に分けることができる．

$$\begin{aligned}\delta_i(d_i) = \ & \delta_i(d_i, F_{u,l}, context) + \\ & \delta_i(d_i, \overline{F}_{u,l}, context)\end{aligned}$$

$$\begin{aligned}lb_i(d_i, x) = \ & lb_i(d_i, x, F_{u,l}, context) + \\ & lb_i(d_i, x, \overline{F}_{u,l}, context)\end{aligned}$$

$$\begin{aligned}LB_i(d_i) = \ & LB_i(d_i, F_{u,l}, context) + \\ & LB_i(d_i, \overline{F}_{u,l}, context)\end{aligned}$$

$$LB_i(d_i, F_{u,l}, context) = \delta_i(d_i, F_{u,l}, context) + \sum_{x \in children_i} lb_i(d_i, x, F_{u,l}, context)$$

$$LB_i(d_i, \overline{F}_{u,l}, context) = \delta_i(d_i, \overline{F}_{u,l}, context) + \sum_{x \in children_i} lb_i(d_i, x, \overline{F}_{u,l}, context)$$

$context_l^{lb}$ は $X_{u,l}$ についての部分解を含まないから，$context_l^{lb} \subseteqq currentContext_l$ についての $LB_l(d_l, F_*, context_l^{lb})$ は，$LB_l(d_l, F_*, context_l^{lb}) = LB_l(d_l, \overline{F}_{u,l}, context_l^{lb}) \leqq LB_l(d_l, \overline{F}_{u,l}, currentContext_l)$ でなければならない．l の祖先で u の子ノードを k とすると，$currentContext_l \subseteqq currentContext_k \cup \bigcup_{x_j \in X_{u,l} - \{x_k\}} \{(x_j, d_j)\}$，$d_j \in D_j$ のとき，$LB_l(d_l, \overline{F}_{u,l}, currentContext_l) \leqq LB_k(d_k, \overline{F}_{u,l}, currentContext_k)$ である．任意の $X_{u,l}$ に関する部分解について $LB_k(d_k, \overline{F}_{u,l}, currentContext_k) \leqq LB_k(d_k, F_*, currentContext_k)$ であるから，ショートカットして通知された下界 $LB_l(d_l, F_*, context_l^{lb})$ は k 以下のサブツリーの下界を超えない．

複数の解の記録および統合では，これまでに得られた最小 (大) の上 (下) 界を保存するのみであるから，従来手法により最終的に得られる上下界を超えず，また，上 (下) 界は単調に減少 (増加) する．

以上より，提案手法により得られる上下界は，従来手法により最終的に得られる上下界を超えないため，従来手法の最適性を失わない．また，上 (下) 界は単調に減少 (増加) するよう統合されるため，提案手法により得られた上下界が最終的な上下界に満たない場合は，従来手法の条件によって得られた上下界により，結局は最終的な上下界に収束する．

6. 実験と評価

提案手法の有効性を計算機実験により評価した．評

価用の問題は従来手法と同様に 3 色彩色問題を用いた．各ノードは 3 値変数 1 個を持ち，他ノードと 2 項制約で関係する．制約数はノード数 n に対して，パラメータ $d = \{2,3\}$ により $n \cdot d$ とした．$d = 2$ の場合は充足可能な問題が比較的多く，$d = 3$ の場合は過制約な問題が多くなる．従来手法 (normal) および，上下界の導出と COST メッセージのショートカット (shortcut)，さらに LRU キャッシュを追加したもの (sc+lru)，さらに内包関係に基づく解統合を追加したもの (sc+lru+integrate) について比較した．各制約の重みを全て 1 とした問題 (最大制約充足問題) および $\{1,\cdots,10\}$ とした問題について実験を行った．以上の各条件について 25 問の問題を生成し結果を平均した．また，LRU キャッシュのサイズは 各 (d,x) について 100 を上限とした．

実験では，まず，シミュレーションにより分散システムを模倣した．システム全体はメッセージ交換と各ノードの処理から成るサイクルで動作し，各ノードは各サイクルごとに，メッセージ受信，条件維持/判定，メッセージ送信の処理を反復するものとした．また，シミュレーションで用いたプログラムを MPI 上に移植し，非同期分散環境下での挙動を確認した．

6.1 最大制約充足問題

各制約の重みを 1 とした場合のサイクル数を図 1,2 に示す．d=2 の場合，メッセージのショートカットの効果と比較して，さらに上下界の学習を追加した場合の効果が小さい．これは，再探索の機会が少ないためであると考えられる．一方，d=3 の場合，上下界の学習による再探索削減の効果が得られた．

ただし，表 3 に示すようにリンク密度によるランダムな問題の生成方法では，問題の最適コストのばらつきのため，サイクル数の平均に対する分散は大きい．また，同様の理由により図 1, および後述の図 4,7(d=2 の場合) のグラフでは，比較的最適コストの大きい問題が多く生成されたノード数 12,14 の問題で，バックトラックと再探索の影響でサイクル数の平均が増加している．改善の程度の詳細として，各問題についての，各提案手法の追加前後でのサイクル数の比較を表 1,2 に示す．各 25 問についてサイクル数が減少した場合 (<)，等しかった場合 (=)，増加した場合 (>) について，各場合の問題数および，各場合についてのサイクル数の最大，最小値を分類した．各提案手法は多くの問題において追加前の手法と比較して，サイクル数が同等か改善した結果が得られた．また，比較的サイクル数を要する問題については，多くの問題でサイクル数の改善が得られた．これらは，後述の他の実験のサイクル数についても同様である．

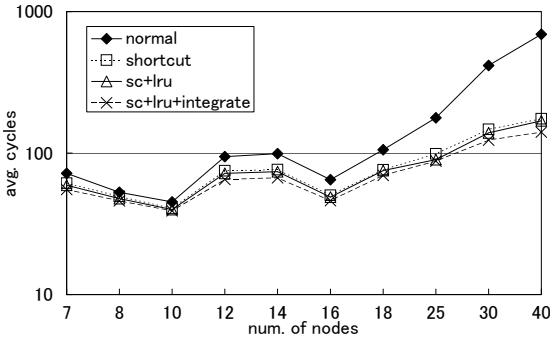

図 1 サイクル数 (制約重み=1,d=2)
Fig. 1 Average number of cycles to find the optimal solution(weight of constraint=1,d=2)

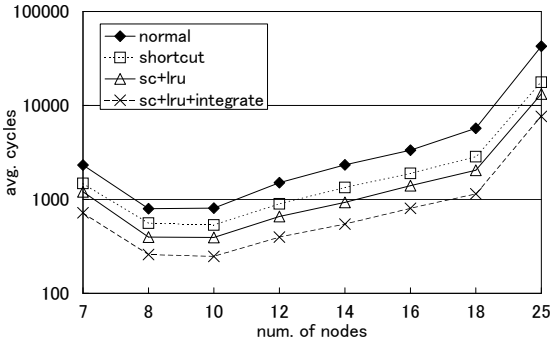

図 2 サイクル数 (制約重み=1,d=3)
Fig. 2 Average number of cycles to find the optimal solution(weight of constraint=1,d=3)

d=3 の場合についてメッセージ数を図 3 に示す．提案手法では COST メッセージのショートカットの影響で，メッセージ数が増加している．しかし全体としての増加は比較的小さい．

6.2 重み付き制約充足問題

各制約の重みを $\{1,\cdots,10\}$ とした場合のサイクル数を図 4,5 に示す．各制約の重み 1 の場合と同様に，d=2 の場合はメッセージのショートカットの効果が得られ，d=3 の場合はメッセージのショートカットおよび上下界の学習の効果が得られた．

6.3 LRU キャッシュ長の影響

各制約の重み 1,d=3, 内包関係に基づく解統合の場合について，キャッシュ長を $\{10,20,40,80,100\}$ とした場合のサイクル数への影響を図 6 に示す．サイズが増加するに従い，サイクル数が減少している．しかし，

表 1　各問題のサイクル数の比較 (制約重み=1,d=2, 括弧内の手法は比較対象)
Table 1 Comparison of cycles for each problem(weight of constraint=1,d=2)

nodes	results			cycles					
	<	=	>	<		=		>	
				max	min	max	min	max	min
normal (. normal)									
7	0	25	0			175	16		
8	0	25	0			146	13		
10	0	25	0			93	14		
12	0	25	0			323	17		
14	0	25	0			275	17		
16	0	25	0			236	19		
18	0	25	0			357	27		
25	0	25	0			1249	28		
30	0	25	0			2120	36		
40	0	25	0			5207	48		
shortcut (. normal)									
7	17	7	1	138	35	66	16	54	54
8	8	17	0	111	42	89	13		
10	9	15	1	75	42	89	14	79	79
12	15	9	1	253	22	80	17	98	98
14	14	11	0	211	24	113	17		
16	11	13	1	169	30	51	19	34	34
18	17	8	0	239	27	33	27		
25	19	6	0	611	35	60	28		
30	21	4	0	839	37	88	36		
40	22	1	2	871	46	48	48	79	63
sc+lru (. shortcut)									
7	11	14	0	130	38	65	16		
8	7	17	1	100	34	74	13	69	69
10	6	19	0	73	41	89	14		
12	8	16	1	227	69	105	17	115	115
14	7	16	2	171	79	143	17	132	127
16	6	19	0	156	30	117	19		
18	8	14	3	213	28	96	27	161	138
25	8	17	0	508	54	66	28		
30	11	11	3	706	47	101	36	274	60
40	4	15	6	818	301	118	48	468	48
sc+lru+integrate (. sc+lru)									
7	15	8	2	112	35	56	16	41	36
8	8	16	1	86	32	74	13	39	39
10	5	18	2	71	42	56	14	92	71
12	13	9	3	174	21	51	17	127	66
14	13	10	2	148	26	82	17	146	82
16	16	9	0	134	23	70	19		
18	12	10	3	168	30	53	27	126	30
25	9	9	7	250	28	63	30	541	30
30	14	6	5	580	46	67	36	220	40
40	16	5	4	660	52	63	48	106	57

表 2　各問題のサイクル数の比較 (制約重み=1,d=3, 括弧内の手法は比較対象)
Table 2 Comparison of cycles for each problem(weight of constraint=1,d=3)

nodes	results			cycles					
	<	=	>	<		=		>	
				max	min	max	min	max	min
normal (. normal)									
7	0	25	0			2316	2316		
8	0	25	0			1332	445		
10	0	25	0			1996	81		
12	0	25	0			3636	210		
14	0	25	0			7143	417		
16	0	25	0			11199	191		
18	0	25	0			19769	272		
25	0	25	0			111535	6130		
shortcut (. normal)									
7	25	0	0	1481	1481				
8	25	0	0	967	298				
10	25	0	0	1132	45				
12	25	0	0	2116	160				
14	25	0	0	3346	286				
16	25	0	0	8128	100				
18	25	0	0	9611	130				
25	25	0	0	62820	2818				
sc+lru (. shortcut)									
7	25	0	0	1207	1207				
8	23	0	2	725	260			351	328
10	23	1	1	835	142	45	45	182	182
12	24	0	1	1418	123			661	661
14	25	0	0	2396	200				
16	25	0	0	5662	96				
18	23	0	2	7991	317			1548	131
25	25	0	0	41010	2013				
sc+lru+integrate (. sc+lru)									
7	25	0	0	719	719				
8	25	0	0	433	197				
10	24	1	0	467	107	45	45		
12	25	0	0	933	111				
14	25	0	0	1196	177				
16	25	0	0	3064	91				
18	24	0	1	4771	121			482	482
25	25	0	0	25319	1162				

解の数は下位ノードでは指数関数的に増加するため，線形にキャッシュのサイズを増加しても得られる効果は次第に小さくなる．

6.4　変数値配信パスの変更による影響

以上の手法では変数値の配信において，VALUE メッセージによる近傍への配信と，COST メッセージによるその他のノードへの配信 (2.5-A) が併用されている．このうち，COST メッセージの *context* の代わりに THRESHOLD メッセージの *context* により値を配信するように変更した場合の結果を図 7 に示す．いずれの場合でもサイクル数には大きな差は無い．これは，探索処理全体の効率において，深さ優先探索木によるコスト計算の方が影響が大きいためであると考えられる．

6.5　MPI 環境における評価

非同期分散環境でのアルゴリズムの挙動を確認するために MPI 環境における実験を行った．ただし，本実験で使用したプログラムは，シミュレーションで使用したプログラムを各ノードについてプロセス化し送受信部に

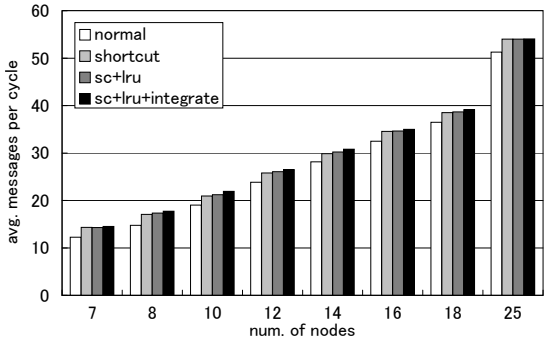

図 3　サイクルあたりのメッセージ数 (制約重み=1,d=3)
Fig. 3　Average number of messages per cycle(weight of constraint=1,d=3)

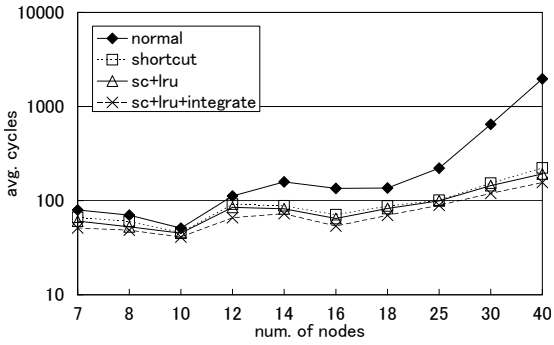

図 4　サイクル数 (制約重み={1, · · · , 10},d=2)
Fig. 4　Average number of cycles to find the optimal solution(weight of constraint={1, · · · , 10},d=2)

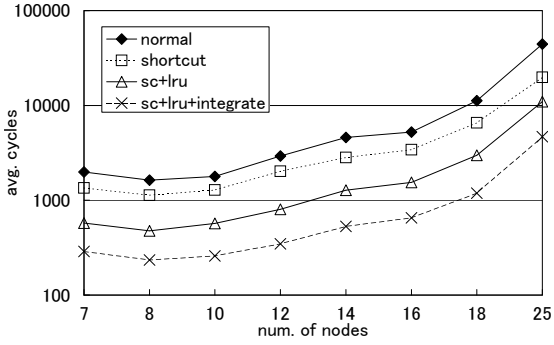

図 5　サイクル数 (制約重み={1, · · · , 10},d=3)
Fig. 5　Average number of cycles to find the optimal solution(weight of constraint={1, · · · , 10},d=3)

表 3　各結果の分散 (ノード数 16)
Table 3　Variances of results (16 nodes)

weight of constraint	1		1...10	
d	2	3	2	3
cycle				
normal	3606	8645435	55839	22302935
shortcut	1372	3453270	7976	10884349
sc+lru	1210	1757614	5001	1856625
sc+lru+int.	970	560947	2114	260599
messages per cycle				
normal		13		
shortcut		17		
sc+lru		20		
sc+lru+int.		25		

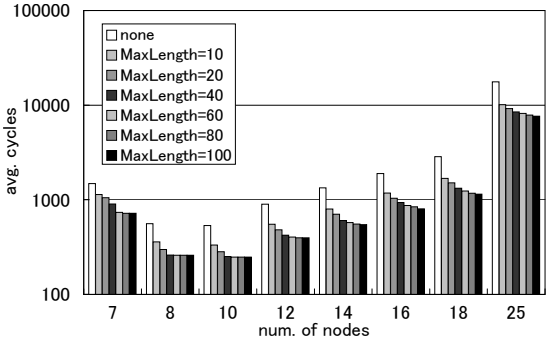

図 6　キャッシュ長のサイクル数への影響 (sc+lru+integrate, 制約重み=1,d=3)
Fig. 6　Effect of cache size on number of cycles to find the optimal solution(sc+lru+integrate,weight of constraint=1,d=3)

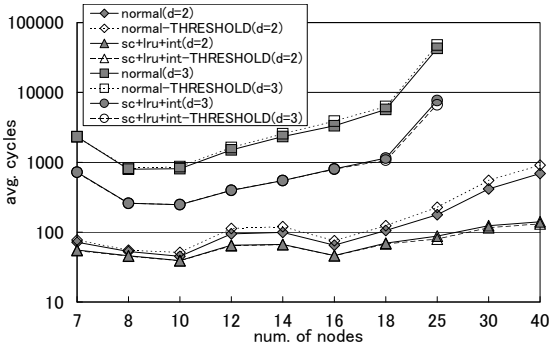

図 7　変数値配信パス変更の影響 (制約重み=1,d=2,3)
Fig. 7　Effect of message path on number of cycles to find the optimal solution(weight of constraint=1,d=2,3)

MPI を適用したものであり，処理の高速化やメッセージ受信のタイミング等に調整の余地がある．制約の重み {1, · · · , 10},d=3 の問題について，各ノード数ごとに任意の 1 つの問題を選択し，各 10 回の試行の結果を平均した．実験で使用した計算機の主な構成は Intel Xeon 1.8GHz(HyperThreading),512MB メモリ,100Mbps Ethernet, Windows2000,MPICH1.2.5 である．1 台の場合の最大サイクル数の平均を図 8 に示す．また，各結果の分散を表 5 に示す．結果には負荷変動，非同

期性等の影響が見られるが，非同期分散環境においてもシミュレーションの結果とほぼ同様の特性が得られた．実行時間を表 4 に示す．実行時間については各提案手法でサイクル数減少の効果による改善が得られた．1 台と 4 台の場合の比較において従来手法とほぼ同等の台数効果が得られ，提案手法による並列性への大きな影響は示されなかった．

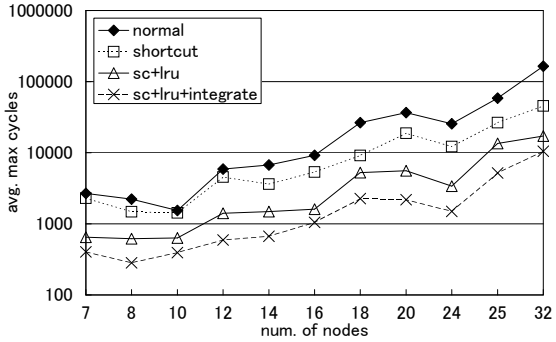

図 8　MPI 環境における最大サイクル数の平均 (制約重み $=\{1,\cdots,10\}$,d=3,1 台)

Fig. 8　Average number of max cycles to find the optimal solution(MPI version,weight of constraint$=\{1,\cdots,10\}$,d=3,1 processor)

7. む す び

深さ優先探索木を基礎とした非同期分枝限定法に基づく，制約最適化の効率化手法を提案した．実験結果により提案手法によるバックトラックおよび再探索におけるオーバヘッド削減の効果が得られた．

本稿では従来手法と同様に，基礎的な検討として，各ノードは単一の変数に対応し制約は 2 項制約となる問題に議論を限定した．これらの一般化は容易であるが多項制約では制約密度は増加する．制約密度が高い問題であるほど，深さ優先探索木の木構造は線形になり並列性は損なわれる．従ってバックトラック効率の改善はより重要である．一方で，大規模で複雑な問題では，ユーザのパラメータにより許容されるコストで探索を打ち切る [8] ことが現実的である．その場合でも本手法のような効率改善手法は有効であると考えられる．

検討の対象としたアルゴリズムは深さ優先探索木の存在を前提としているが，非同期分散探索における単調性の基礎としての木構造は本質的に重要であり，ブロードキャストに基づく対称なノードによる解法を構築する場合でもノードの順序付けの解決等に関連するものと考えられる．より効率的な解空間の学習手法の導入，他の解法との比較，実際的な問題への適用は今後の課題である．

表 4　MPI 環境における実行時間 [ms](制約重み $=\{1,\cdots,10\}$,d=3)

Table 4　Average number of times to find the optimal solution[ms](MPI version,weight of constraint$=\{1,\cdots,10\}$,d=3,1,4 processor(s))

num. of nodes	method	1 processor	4 processors	speed up rate
8	normal	2670	1841	1.5
	shortcut	2934	1769	1.7
	sc+lru	960	625	1.5
	sc+lru+int.	488	278	1.8
12	normal	10366	5245	2.0
	shortcut	11083	5047	2.2
	sc+lru	3380	1684	2.0
	sc+lru+int.	1425	570	2.5
16	normal	22860	10208	2.2
	shortcut	20789	9144	2.3
	sc+lru	6603	3227	2.0
	sc+lru+int.	3174	1450	2.2
20	normal	128135	56606	2.3
	shortcut	81791	30958	2.6
	sc+lru	30674	13208	2.3
	sc+lru+int.	12086	5041	2.4
24	normal	88595	38206	2.3
	shortcut	65573	25033	2.6
	sc+lru	22491	9511	2.4
	sc+lru+int.	11128	4089	2.7
32	normal	895219	343636	2.6
	shortcut	303739	107809	2.8
	sc+lru	144059	53377	2.7
	sc+lru+int.	104459	35062	3.0

表 5　各結果の分散 (MPI 環境, 制約重み$=\{1\cdots10\}$,d=3, ノード数 16)

Table 5　Variances of results (MPI version,weight of constraint$=\{1,\cdots,10\}$,d=3,16 nodes)

method	1 processor	4 processors
cycle		
normal	20074778	
shortcut	2649430	
sc+lru	119562	
sc+lru+int.	208107	
execution time		
normal	8272406	89895
shortcut	2340838	66134
sc+lru	128600	5802
sc+lru+int.	158003	38753

謝辞

本研究の一部は、文部科学省科学研究費補助金（課題番号 15500037) による。

文　献

[1] T. Schiex, H. Fargier and G. Verfaillie, "Valued constraint satisfaction problems: Hard and easy problems," Proc. Int. Joint Conf. on Artificial Intelligence, vol.1, pp.631–639, 1995.

[2] E.C. Freuder and R.J. Wallace, "Partial Constraint Satisfaction," Artif. Intell., vol.58, no.1–3, pp.21–70, 1992.

[3] M. Yokoo, E.H. Durfee, T. Ishida and K. Kuwabara, "The Distributed Constraint Satisfaction Problem: Formalization and Algorithms,"IEEE Trans. Knowledge and Data Engineering, vol.10, no.5, pp.673–685, 1998.

[4] M. Lemaître and G. Verfaillie, "An Incomplete Method for Solving Distributed Valued Constraint Satisfaction Problems," In Proc. AAAI97 Workshop on Constraints and Agents, 1997.

[5] K. Hirayama and M. Yokoo, "Distributed Partial Constraint Satisfaction Problem," Proc. 3rd Int. Conf. on Principles and Practice of Constraint Programming, pp.222–236, 1997.

[6] K. Hirayama and M. Yokoo, "An Approach to Over-constrained Distributed Constraint Satisfaction Problems: Distributed Hierarchical Constraint Satisfaction," Proc. 4th Int. Conf. on Multiagent Systems, pp.135–142, 2000.

[7] J. Liu and K. Sycara, "Exploiting problem structure for distributed constraint optimization," Proc. 1st Int. Conf. on Multiagent Systems, pp.246–253, 1995.

[8] P.J. Modi, W. Shen, M. Tambe and M. Yokoo, "An Asynchronous Complete Method for Distributed Constraint Optimization," Proc. Autonomous Agents and Multi-Agent Systems, pp.161–168, 2003.

[9] P. J. Modi, W. Shen, M. Tambe and M. Yokoo, "Adopt: asynchronous distributed constraint optimization with quality guarantees," Artif. Intell., vol.161, pp.149–180, 2005.

[10] D.L Mammen and V.R. Lesser, "Problem Structure and Subproblem Sharing in Multi-Agent Systems," Proc. 3rd Int. Conf. on Multiagent Systems, pp.174–181, 1998.

[11] K. Hirayama and M. Yokoo, "The Effect of Nogood Learning in Distributed Constraint Satisfaction," Proc. 20th Int. Conf. on Distributed Computing Systems, pp.169–177, 2000.

[12] T. Schiex and G. Verfaillie, "Nogood Recording for Static and Dynamic Constraint Satisfaction Problems," International Journal of Artificial Intelligence Tools, vol.3, no.2, pp.187–207, 1994.

[13] M. Yokoo, "Weak-commitment Search for Solving Constraint Satisfaction Problems," Proc. 12th National Conf. on Artificial Intelligence, pp. 313–318, 1994.

[14] M. Yokoo, "Asynchronous Weak-commitment Search for Solving Distributed Constraint Satisfaction Problems," Proc. 1st Int. Conf. on Principles and Practice of Constraint Programming, pp.88–102, 1995.

[15] B. Awerbuch, "A New Distributed Depth-First-Search Algorithm," Inf. Process. Lett., vol.20, no.3, pp.147–150, 1985.

[16] M.B. Sharma, S.S. Iyengar, "An Efficient Distributed Depth-First-Search Algorithm," Inf. Process. Lett., vol.32, no.4, pp.183-186, 1989.

[17] D. Kumar, S.S. Iyengar, M.B. Sharma, "Corrigenda: Corrections to a Distributed Depth-First Search Algorithm," Inf. Process. Lett., vol.35, no.1, pp.55-56, 1990.

（平成 xx 年 xx 月 xx 日受付）

松井　俊浩　（正員）

平成 11 名工大大学院工学研究科博士前期課程修了．現在，同大学院博士後期課程に在学中．

松尾　啓志　（正員）

1983 名古屋工業大学情報工学科卒．1985 同大学院修士課程了．民間企業を経て 1986 同大学院博士後期課程入学．1989 同課程了．1989 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 同教授，2004 年学科改組により情報工学専攻教授，現在に至る．分散システム，分散学習に関する研究に従事．

岩田　　彰　（正員）

昭 48 名大・工・電気卒．昭 50 同大大学院修士課程了．同年名工大・情報・助手．昭 57 年 4 月より昭 58 年 10 月まで，ドイツ連邦共和国ギーセン大学医学部医用情報研究所客員研究員．昭 59 名工大・情報・助教授．平 5 同大・電気情報・教授．平 14 同大・副学長．平 16 同大・大学院・教授．現在に至る．ニューラルネットワーク，聴覚支援システム，情報セキュリティ、公開鍵基盤（ＰＫＩ）、電子証明書に関する研究に従事．工博．H5 年度本会論文賞受賞，H10 年度情報処理学会 Best Author 賞受賞．情報処理学会，日本ＭＥ学会，日本心電図学会，日本神経回路学会，日本医療情報学会各会員，IEEE Senior Member．

Abstract Distributed constraint optimization problem (DCOP) is an important area of study of multi-agents. Recently, a distributed search algorithm for DCOP is proposed. The algorithm enables asynchronous search based on blanch-and-bound and depth first tree. However, the algorithm has overhead of backtracking and re-exploration. In this paper, some effective methods which reduce overhead of former algorithm are proposed. To reduce overhead of backtracking, partial solution which is related with lower/upper bound is considered. And the information of lower bound is sent to upper level nodes, by a shortcut. To reduce re-exploration, leaning of solution is introduced.

Key words Distributed Constraint Optimization, Distributed CSP, Multi-agent, Blanch-and-Bound