論　文

複数解の展開と枝刈りを用いる分散制約最適化手法

松井　俊浩[†a)]　　松尾　啓志[†b)]

Distributed Constraint Optimization Method Employing Multiple Solutions and Pruning

Toshihiro MATSUI[†a)] and Hiroshi MATSUO[†b)]

あらまし　分散制約最適化問題 (DCOP) は，マルチエージェントシステムにおける協調問題解決を各エージェントに分散して配置された離散最適化問題として扱う，基礎的な研究分野である．DCOP の厳密解法として，木探索や動的計画法にもとづく分散アルゴリズムが提案されている．木探索に基づく解法は，メッセージ通信を介する反復的な処理を伴うため，通信にオーバヘッドがあれば，その影響を多く受ける．純粋な動的計画法に基づく解法には反復的な探索はないが，必要とする記憶とメッセージのサイズは，制約網に対する疑似木の induced-width に対して指数関数的である．記憶のサイズの制限のもとで，両者を併用する手法は提案されているが，探索点が単一であるか，大域的なコスト値に基づく枝刈りが考慮されていない．メッセージの交換を介する探索処理の反復回数を削減するために，木探索において複数の探索点を同時に展開し，それらに対応する複数の解を同一のメッセージにより送受信することは合理的である．本研究では，記憶とメッセージのサイズの制限のもとで，複数の解を同時に展開する木探索と動的計画法を併用する解法を示し，その効果を実験により評価する．

キーワード　マルチエージェントシステム，分散制約最適化問題，協調問題解決，探索

1. まえがき

分散制約最適化問題 (DCOP) [1]～[4] 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礎的な研究分野である．複数のエージェント間の協調問題は，各エージェントに分散して配置された離散最適化問題として表現される．DCOP に基づく，センサ網，電力スマートグリッド，会議スケジューリングなどの分散資源割り当て問題が提案されている [5]～[7].

DCOP の厳密解法として，木探索や動的計画法にもとづく分散アルゴリズムが提案されている．木探索に基づく解法 [3] は，メッセージ通信を介する反復的な処理を必要とするため，通信にオーバヘッドがあれば，その影響を多く受ける．純粋な動的計画法に基づく解法 [4] には反復的な探索はないが，必要とする記憶とメッセージのサイズ，及びそれらに伴う計算量は，制約網に対する疑似木の induced-width [4] に対して指数関数的である．記憶のサイズの制限のもとで，両者を併用する手法は提案されているが，探索点が単一である [3] か，大域的なコスト値に基づく枝刈りが考慮されていない [8]～[10].

メッセージの交換を介する探索処理の反復回数を削減するために，木探索において複数の探索点を同時に展開し，それらに対応する複数の解を同一のメッセージにより送受信することは合理的である．そこで本研究では，分枝限定法と動的計画法を併用する解法において，記憶とメッセージのサイズの制限のもとで，複数の解を同時に展開する手法を提案する(注1)．このような手法は複数の探索を重複して実行することに相当し，展開中の解の集合を把握することにより複数の探索点の統廃合の余地が得られる．そのため，混合的な探索戦略を用いる解法の基礎となりうると考えられる．

また，従来の解法を一般化するにあたり，特に本質的な計算である，解の展開において副問題の解空間を考慮する部分解の再構成及び，枝刈りのための大域的コスト値の上下界の計算を明示的に用いる．これらは

† 名古屋工業大学，名古屋市
Nagoya Institute of Technology, Nagoya-shi, 466–8555 Japan
a) E-mail: matsui.t@nitech.ac.jp
b) E-mail: matsuo@nitech.ac.jp

(注1)：本論文は [11], [12] を基に，説明と評価を補強した．

 ©一般社団法人電子情報通信学会 2014

一部に差分を用いる表現 [3] よりも平易といえる．

以下では，まず **2.** で研究の背景について述べる．**3.**, **4.**, **5.** で提案手法を示し，その有効性の評価を **6.** に示す．**7.** で結言を述べる．

2. 背　　景

2.1 分散制約最適化問題

本研究では，次のような基礎的な分散制約最適化問題を用いる．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 及び各制約に対応する二項関数の集合 F により定義される．エージェント i は変数 x_i をもつ．x_i は離散値の集合 D_i に含まれる値をとる．変数 x_i の値はエージェント i によってのみ決定される．制約 $c_{i,j}$ は x_i と x_j の関係を表す．各変数値の割り当て $\{(x_i, d_i), (x_j, d_j)\}$ のコストは，二項関数 $f_{i,j}(d_i, d_j) : D_i \times D_j \to \mathbb{N}_0$ により定義される．大域的なコスト値 $\sum_{f_{i,j} \in F,\ \{(x_i,d_i),(x_j,d_j)\} \subseteq \mathcal{A}} f_{i,j}(d_i, d_j)$ を最小化する最適解 $\mathcal{A}$ を求めることが目的である．i は，x_i に関する制約，評価関数を知る．最適解を求める解法は，エージェント間のメッセージ通信を用いる分散アルゴリズムとして構成される．

2.2 疑似木 (pseudo-tree)

疑似木 [13], [14] は制約網に対して生成される，グラフ上の構造である．典型的な疑似木は制約網に対する深さ優先探索木に対応する．例えば図 1 (a) の制約網から図 1 (b) の疑似木が生成される．疑似木では，元の制約網の辺は木辺と後退辺に分類される．木辺は元の制約網の生成木の一つに対応する．後退辺は木辺以外の辺である．疑似木の各頂点を，対応する生成木の頂点とみなし，頂点間の関係として根，葉，親，子などの用語を用いる．また，表現を簡単にするために，疑似木の各頂点及び，各頂点に対応する変数とエージェントを区別せずに扱う．特に次の表記を用いる．

$\boldsymbol{prnt_i}$**:** x_i の親の変数．

$\boldsymbol{Chld_i}$**:** x_i の子の変数の集合．

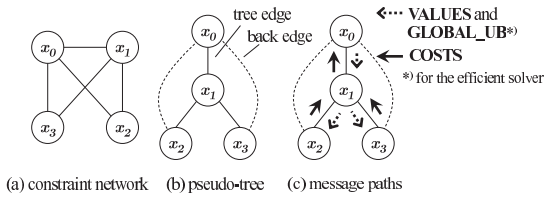

図 1　疑似木とメッセージ
Fig. 1　Pseudo-tree and messages.

$\boldsymbol{Nbr_i^u}$**:** x_i と制約で関係する x_i の祖先の変数の集合．

$\boldsymbol{Ast_i}$**:** x_i の祖先の変数の集合の部分集合．$x_k \in Ast_i$ なる変数 x_k は，x_i を根とする部分木に含まれる変数 x_j の少なくとも一つについて，$x_k \in Nbr_j^u$ である．

疑似木の異なる部分木の間には辺がないため，各部分木における計算を並行できる．

2.3 疑似木におけるコスト値の計算

疑似木を用いる最適コスト値の計算 [3], [4] を示す．ここでは，各エージェントは計算に必要な他のエージェントの変数値やコスト値を，既に受信しているものと仮定する．エージェント i における計算は，Ast_i についての部分解 s_i にもとづく．部分解 s_i と x_i の変数値 d についての，局所コスト $\delta_i(s_i \cup \{(x_i, d)\})$ は次のように定義される．

$$\delta_i(s_i \cup \{(x_i, d)\}) = \sum_{(x_j,d_j) \in s_i,\ j \in Nbr_i^u} f_{i,j}(d, d_j) \quad (1)$$

部分解 s_i，及び x_i を根とする部分木についての最適コスト $g^*(s_i)$ は，次のように再帰的に定義される．

$$g_i^*(s_i) = \min_{d \in D_i} g_i(s_i \cup \{(x_i, d)\}) \quad (2)$$

$$g_i(s_i \cup \{(x_i, d)\}) = \delta_i(s_i \cup \{(x_i, d)\}) + \sum_{j \in Chld_i} g_j^*(s_j) \quad \text{s.t.} \quad s_j \subseteq (s_i \cup \{(x_i, d)\}) \quad (3)$$

実際の探索における計算では，コスト値の一部が未知である場合に，コスト値の上下限が用いられる．本研究では，下限値 0 と上限値 ∞ を用いる．これらの導入により，コスト値は，上下界として表現される．大域的最適コスト $g_r^*(\phi)$ が根の変数 x_r について計算されれば，x_r の最適値が決定される．同様に，残された部分問題についての最適解が，再帰的に計算される．

2.4 従 来 手 法

疑似木に基づく DCOP の厳密解法として，動的計画法と分枝限定法に基づく解法が提案されている．動的計画法 [4] では，葉のエージェントから順に部分木の最適コスト値を計算し，その後，根のエージェントから順に，変数の最適値を決定する．反復的な探索処理はないが，各エージェント i は Ast_i の変数の値の組全てについて一度に $g^*(s_i)$ を計算するため，記憶とメッセージの大きさは，疑似木の induced width [4] に従って指数関数的に増大する．最適コスト値の計算を時分割できる [8], [9] が，探索の枝刈りに課題がある．

分枝限定法と記憶の制限のある動的計画法を併用する解法 [3] では，探索点すなわち各エージェント i における現在の部分解 s_i は一つである．通信を介する反復処理の回数を削減するために，複数の解を同時に計算し，同一のメッセージで交換することは合理的である．しかし，従来研究では，遅延を伴うメッセージの伝搬を短絡するコスト値の計算 [15] や，計算結果のキャッシュによる反復処理回数の削減 [16] は提案されているが，複数の探索点を同時に展開する手法については十分に検討されていない．

3. 複数の解を展開する分散制約最適化手法

本研究では，疑似木を用いる分枝限定法と動的計画法に基づく厳密解法を一般化し，複数の解を同時に展開する解法を示す．まず本章では，基本的な解法を示し，次章で効率化手法を示す．

3.1 基本的な解法

大域的なコスト値による枝刈りを用いない，基本的な解法を示す．この解法は，記憶に制限のある動的計画法 [8] に類似する．この解法では，図 1 (c) に示す 2 種類のメッセージ VALUES 及び COSTS を用いる．図 2 (a) に示すように，疑似木の木辺に沿ってトップダウンに伝搬されるメッセージ VALUES を介して複数の解が同時に展開される．この例では，三つの部分解が同時に展開される．その一方で，木辺に沿うボトムアップなメッセージ COSTS を介して現在の解それぞれのコスト値が集計される．いずれ，根のエージェントでは，大域的なコスト値の上下界が等しくなる．そして，根のエージェントの最適な変数値が選択される．残された部分問題についても同様に，トップダウンな処理により，再帰的に最適解が決定される．以下では，解法を構成する要素についての説明を示し，解法の要点について説明を加える．

DCOP 及び疑似木に関する表現に加えて，エージェント i は次の変数とデータ構造を用いる．

S_i: Ast_i に含まれる変数についての，現在の部分解の集合であり，i の親から受信される．

U_i: S_i についてのコスト値の表をあらわす集合．U_i の要素 u は，$(a(u), lb(u), ub(u))$ の各要素から構成される．$a(u)$ は部分解を表す．$ub(u)$ 及び $lb(u)$ は，それぞれ $a(u)$ についてのコスト値の上下界を表す．U_i は i の親に送信される．

$sAct_i$: S_i の状態を表す．新たな部分解が i の親から受信されたとき，$sAct_i$ の値は真になる．後述の closed(U_i) が真であれば，$sAct_i$ の値は偽となる．

$sOpt_i$: S_i の状態を表す．i の親から受信される．現在の S_i が最適解を含むのであれば，$sOpt_i$ の値は真であり，そうでなければ偽である．

opt_i: i の状態を表す．x_i の最適値が決定されていれば，真であり，そうでなければ偽である．

D_i^*: i が決定した x_i の最適値のみを含む値域．

R_j^i: S_i をもとに，子 j の Ast_j について生成される，部分解の集合．R_j^i の要素は，$\{s' | s' = s \cup \{(x_i, d)\}, s \in S_i, d \in D_i\}\}$ なる集合 R_i の要素から，Ast_j に含まれない変数値の割り当てを除いた，部分解である．R_j^i の要素は少なくとも一つの R_i の要素に対応する．

S_j^i: 子 j に送信する，R_j^i の部分集合．

U_j^i: R_j^i についてのコスト値を表す．U_j^i の要素は子 j から受信する U_j により更新される．

$sFil_j^i$: S_j^i が全ての R_j^i の要素について，子 j に送信されていれば，真となる．そうでなければ偽となる．

また，次の種類のメッセージが用いられる．

VALUES: エージェント i は子 j それぞれに，VALUES により，S_j^i 及び，その状態 $sOpt_i$ を送信する．

COSTS: エージェント i は親 j に，COSTS により，S_i すなわち S_i^j に対応する U_i を送信する．

3.2 複数の解の展開における部分解の再構成

エージェント i は子 j に，現在の解の集合 S_j^i を VALUES メッセージにより通知する．S_j^i は R_j^i の部分集合である．i は S_i の部分問題の部分解を j について繰り返し展開する際に，Ast_j に含まれる変数についての解の集合 R_j^i の要素を網羅しつつ列挙するように，S_j^i を決定する．現在の部分解の集合 S_i に対するコスト値の境界が収束したとき，S_j^i は次の R_j^i 部分集合に更新される．j を根とする部分木のコスト値の計算は，Ast_j にのみ影響されるため，S_i から不要な変数値の割り当てを除くことができる．これにより展開される解の大きさと数が抑制される．また，Ast_i に

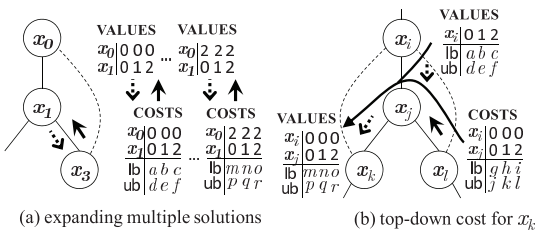

図 2 疑似木に基づくコスト値の計算
Fig. 2 Computation of cost values.

含まれない x_i の祖先の変数値が i に漏えいしない．

opt_i が偽であるとき，すなわち，最適な変数値の割り当てが未定のとき，S_j^i は次のように更新される．R_j^{i+} を $R_j^i \backslash \{a(u)|u \in U_j^i\}$ なる集合とする．このとき，$S_j^i \subseteq R_j^{i+} \wedge |S_j^i| \leq L$ を満足するように S_j^i を更新する．ここで，L は部分解の最大数を制限する値である．そしてコストの初期値 $\{u|a(u) \in S_j^i, lb(u) = 0, ub(u) = \infty\}$ が U_j^i に加えられる．opt_i が真であるとき，R_j^i に唯一含まれる部分解 r を含むように S_j^i が更新される．また，$a(u) = r, r \in R_j^i$ なる要素 u が U_j^i に含まれないが解法の計算において必要であるとき，既定値 $(r, 0, \infty)$ が U_j^i に加えられる．

3.3 コスト値の非同期的な計算

エージェント i は，子 j から部分解の集合 R_j^i についてのコスト値の上下界 U_j を COSTS メッセージにより受信し，U_j^i として保持する．S_i についての，i を根とする部分木のコスト値の情報 U_i は，各 U_j^i と，i の局所コストに基づいて，次のように計算される．

S_i の要素 s 全てについて，$u_s = (s, lb(u_s), ub(u_s))$ なる U_i の要素 u_s を計算する．$lb(u_s)$ と $ub(u_s)$ の計算は，式 (2), (3) に示した g_i^* の計算に従う．まず，D_i の各値 d について，i の局所コスト $\delta_i(s \cup \{(x_i, d)\})$ と，$lb(u')$ s.t. $(a(u') \subseteq s \cup \{(x_i, d)\}, u' \in U_j^i, x_j \in Chld_i)$ なるコスト値の下界 $lb(u')$ 全てを用いて，$s \cup \{(x_i, d)\}$ のコスト値の下界 $lb_{s \cup \{(x_i,d)\}}$ が計算される．次に，$\min_{d \in D_i} lb_{s \cup \{(x_i,d)\}}$ により，$lb(u_s)$ が計算される．$ub(u_s)$ も同様に，コスト値の上界を用いて計算される．ある $(s(u'), lb(u'), ub(u'))$ が U_j^i に含まれないとき，$lb(u')$ と $ub(u')$ の値はそれぞれ 0 と ∞ とする．

i は，U_i の要素 u 全てについて，$lb(u) = ub(u)$ になるまで，COSTS メッセージを親に送信する．コスト値は既知の部分解に対して計算されるため，$lb(u)$ と $ub(u)$ は初期の段階では一致しない場合があることに注意されたい．その一方で，子 j の現在の S_j^i の要素全てについて，U_j^i の各要素のコスト値の上下界が閉じたとき，i は非同期的に S_j^i を次の部分解の集合に変更し VALUES メッセージにより子 j に送信する．以前の部分解の集合に対する COSTS メッセージを子から受信したとき，古いメッセージは無視される．また，上位の部分解 S_i が変更されたとき，$U_j^i \leftarrow \phi$ のように更新する．

3.4 基本的な解法の詳細

基本的な解法の詳細を図 3 に示す．図中で用いられる関数と手続きの詳細を次に示す．

```
1  Main(){
2   Initialize().
3   until(forever){
4    until(receive loop is broken){ receive messages. }
5    if(S_i ≠ φ ∧ sAct_i){ Maintenance(). } } }
6  Initialize(){
7   S_i ← φ. sAct_i ←false. sOpt_i ←false. opt_i ←false. D_i^* ← φ.
8   foreach j s.t. x_j ∈ Chld_i{
9    (S_j^i, U_j^i, R_j^i, sFil_j^i)←(φ, φ, φ, false). }
10  if(x_i is root){ send (VALUES, {φ}, true) to i. } }
11 Receive(VALUES, S, sOpt){
12  S_i ← S. sAct_i ←true. sOpt_i ← sOpt.
13  foreach j s.t. x_j ∈ Chld_i{
14   reset child states (S_j^i, U_j^i, R_j^i, sFil_j^i). } }
15 Receive(COSTS, j, U){
16  if(compatible(U,S_j^i)){ merge U to U_j^i. } }
17 Maintenance(){
18  update U_i and test optimal condition.
19  if(opt_i ∧ (D_i has not been replaced by D_i^*)){
20   replace D_i by D_i^*.
21   foreach j s.t. x_j ∈ Chld_i{
22    reset child states (S_j^i, U_j^i, R_j^i, sFil_j^i). }
23   update U_i and test optimal condition. }
24  MaintainAndSendChildrensContexts().
25  if(x_i is not root){ send (COSTS,i,U_i) to i s.t. x_i = prnt_i. }
26  if(closed(U_i)){ sAct_i ←false. } }
27 MaintainAndSendChildrensContexts(){
28  foreach j s.t. x_j ∈ Chld_i{
29   if(¬(contexts of j are active) ∧ ¬sFil_j^i){
30    set next contexts S_j^i.
31    if(S_j^i is last part of R_j^i){sFil_j^i ←true.}
32    send(VALUES,S_j^i, opt_i) to j. } } }
```

図 3　複数の部分解を展開する探索
Fig. 3　multiple contexts search.

reset child states (S_j^i, U_j^i, R_j^i, $sFil_j^i$): 子 j, について，$(S_j^i, U_j^i, sFil_j^i) \leftarrow (\phi, \phi, \text{false})$ とし，R_j^i を更新する．

compatible(U,S): U の要素 u 全てについて，S が $a(u)$ を含むならば真を返す．そうでなければ偽を返す．

merge U to U': U の要素 u 全てについて，$a(u') = a(u)$ なる U' の要素 u' を次のように更新する．$lb(u') \leftarrow \max(lb(u'), lb(u))$．$ub(u') \leftarrow \min(ub(u'), ub(u))$．

update U_i and test optimal condition: 3.3 に示したように U_i を更新する．U_i の計算の過程で，最適解の条件を判定する．$sOpt_i \wedge (\exists u \in U_i, lb(u) = ub(u)) \wedge \neg opt_i$ であれば，最適解は次のように決定される．$opt_i \leftarrow true$．$D_i^* \leftarrow \{d^*\}$．ただし d^* は，$a(u) \cup \{(x_i, d^*)\}$ についてのコスト値の上界が $lb(u)$ と $ub(u)$ に等しい，D_i の値である．最適解は適切な方法で一つのみを選択する．したがって，$sOpt_i$ が真のとき，$|S_i| = |U_i| = 1$ となる．

closed(U_i): U_i の要素 u 全てについて，$lb(u) = ub(u)$ であれば，真を返す．そうでなければ，偽を返す．

set next contexts S_j^i: 3.2 に示したように，S_j^i を更新する．

contexts of j are active: S_j^i の要素 s いずれかについて，$a(u) = s$ なる U_j^i の要素 u が，$lb(u) \neq ub(u)$ であれば，真を返す．そうでなければ偽を返す．

4. 効率化手法の導入

3. の解法に，効率化手法を導入する．提案手法は主に，大域的コスト値に基づく枝刈りと計算結果の再利用からなる．以下では，先ず手法の概説とその構成要素を示したのち，詳細について述べる．

4.1 大域的コストによる枝刈りとキャッシュの導入

この解法では，大域的なコスト値の上下界にもとづく枝刈りが用いられる．大域的なコスト値の上下界を計算するために，祖先の変数及び，他の部分木に含まれる変数についての，部分解のコスト値のトップダウンな計算が導入される．図 2 (b) の例では，x_j を根とする部分木を除く，他の部分のコスト値が x_i を経由して集計される．

このために VALUES メッセージもコスト値を伝搬するように拡張される．x_j へのトップダウンなコスト値と，x_j を根とする部分木のコスト値を合計することで，大域的なコスト値の下界と上界が得られる．大域的な上界値は，最良のコスト値を改善する．大域的な下界値は，部分解を枝刈りするために評価される．また，大域的なコスト値の最良の上界を伝搬するために，図 1 (c) に示す GLOBAL_UB メッセージが新たに導入される．

GLOBAL_UB メッセージは各ノードが知る大域的コストの上界値を伝達するメッセージであり，VALUES メッセージと同様に，擬似木の木辺に基づいてトップダウンに伝播される．各エージェントは，VALUES 及び COSTS メッセージにより親及び子から受信したコストの上界値と，自身が評価するコスト関数の値を合計して大域的コストの上界値を求める．更に，自身の上界値と GLOBAL_UB メッセージにより親から受信した上界値を比較して，より小さい方を採用する．そして，大域的コストの上界値は GLOBAL_UB メッセージにより子に送信される．これにより，各エージェントは枝刈りの基準となる大域的コストの上界値を更新する．

更に，冗長な探索を削減するために，計算結果が再利用される．以下では，特に解法の変更点及び要点について説明する．エージェント i が用いる変数とデータ構造は，次のように変更される．

S_i: 各部分解に，トップダウンに集計されるコスト値の上下界が付加される．S_i の要素 s は，U_i の要素と同様に，$(a(s),lb(s),ub(s))$ の各要素から構成される．$a(s)$ は部分解を表す．$ub(s)$ と $lb(s)$ は，$a(s)$ についての祖先と他の部分木のコスト値の上下界を表す．

R_j^i: S_i と同様に，R_j^i に含まれる各部分解にコスト値の上下界が付加される．R_j^i の要素 r について，$lb(r)$ は次のコスト値の合計として計算される．

- $(a(r)\backslash\{(x_i,d)\}) \subseteq s$ なる S_i の要素 s の $lb(s)$.
- i の局所コスト $\delta_i(s \cup \{(x_i,d)\})$.
- $a(u') \subseteq s \cup \{(x_i,d)\}, u' \in U_k^i, x_k \in Chld_i \backslash \{x_j\}$ なる全ての $lb(u')$.

$ub(r)$ は同様に，コスト値の上界を用いて計算される．$(a(r)\backslash\{(x_i,d)\}) \subseteq s$ なる S_i の要素 s は複数ありうる．その場合は，各 s に対して計算された $lb(r)$ と $\mathrm{ub}(r)$ のうち，それぞれ最小の $lb(r)$，最大の $ub(r)$ を用いる．

S_j^i: S_i, R_j^i と同様の構成の要素を含む．

U_j^i: R_j^i の各要素についてのコスト値の上下界を表す．また，キャッシュとしても扱う．上位の部分解 S_i が変更されても，U_j^i の要素は保存され，現在の S_i についての R_j^i に対応する，U_j^i の要素があれば再利用される．新たに U_j^i に要素が追加されたとき，$|U_j^i| \leq K$ を満足しなければ，LRU に従って要素が削除される．ただし $K \geq |R_j^i|$ であるものとする．

また，次の要素が新たに用いられる．

$globalUB_i$: コスト値の大域的な上界である．完全解について合計されたコスト値の，現時点での最小値を表す．

$sCls_i$: S_i の状態であり，親 j の cls_j の値を表す．

cls_i: i において，トップダウンに計算されるコスト値の上下界が閉じたことを表す．$sCls_i$ が真であり，かつ，子 j 全て，S_j^i の要素 s 全てについて $lb(s) = ub(s)$ であれば，真となり，そうでなければ偽となる．

$sChg_j^i$: S_i 及び S_i の状態が変化した直後か否かを，明示的に表す，補助的な情報である．

$sFin_i$: S_i の状態を表す．$sFin_i$ の初期値は偽である．根のエージェント r が，x_r の最適値を決定したとき，$sFin_r$ の値は真となり，全てのエージェントに伝搬される．これにより，枝刈りの条件が変更される．

U_i, $sOpt_i$, opt_i, D_i^*, $sAct_i$, $sFil_j^i$ には変更はない．また，メッセージの変更点は次のようである．

VALUES: S_j^i 及び S_j^i の状態の構成が変更される．

また，S_j^i の要素のコスト値の境界が閉じるまで，同一の S_j^i について複数回送信される．

GLOBAL_UB: 各エージェントは，GLOBAL_UB メッセージにより，$globalUB_i$ を子に送信する．

4.2 トップダウンなコスト値の計算

この解法では，最適コストを求めるボトムアップな計算と並行して，各変数の祖先及び他の部分木についての部分解のコストをトップダウンに計算する．祖先と他の部分木のコストと，自身を根とする部分木のコストを結合することで，より精度の高い，大域的なコスト値の上下界を得る．トップダウンなコスト値の計算のために，エージェント i が親から受信する現在の部分解の集合 S_i の各要素 s は，部分解 $a(s)$ 及びコスト値の上下界 $ub(s), lb(s)$ からなる．

エージェント i の子 j についての部分解の集合 R_j^i の各要素のコストは，S_i，i の局所コスト，及び j を除く子 k 全てについての U_k^i のコストを加算することにより得られる (図 2 (b))．**3.2** で述べたように，上位の部分解 S_i から複数の解を展開する際に冗長な変数の割り当てが除去されると，S_i に含まれる複数の要素が，R_j^i の単一の要素に対応しうることに注意されたい．そのような場合は，コスト値の上下界が真のコスト値を超えないように，もっとも広い上下界を用いる．

R_j^i についての現在の部分解 S_j^i は i から j に VALUES メッセージにより送信される．エージェント i に他の子すなわち部分木が存在する場合，S_i が変化した直後は，それらについてのコスト値の境界は閉じていない場合がある．そこで，コスト値の境界が閉じるまで，VALUES メッセージは繰り返し送信される．トップダウンなコスト値の計算は，現在の S_i に依存するが，最適コスト値の計算とは独立している．その収束は $cls_i, sCls_i$ を用いて管理される．

4.3 大域的なコストによる枝刈り

エージェント i は，自身の集計値により，大域的なコスト値の上界 $globalUB_i$ を最小化する．$globalUB_i$ は，GLOBAL_UB メッセージにより他のエージェントにも伝搬される．i は，現在の解のコスト値の下界と $globalUB_i$ の比較により探索を枝刈りする．枝刈りされた部分解は，子のエージェントには送信されない．枝刈りの条件は，最適コスト値の計算の段階すなわち根ノードが最適コストを得ていない段階と，その後の最適解の決定の段階で異なる．最適コスト値の計算の段階では下界が大域的上界以上であれば枝刈りする．最適解の決定の段階では，最適解に対する境界を収束させるために，下界が大域的上界を上回る場合のみ枝刈りする．枝刈りの条件を表す情報として $sFin_i$ をトップダウンに伝搬する．

4.4 計算結果の再利用

エージェント i は子 j から受信したコスト値の情報を U_j^i に保持する．現在の部分解 S_i が変更されても U_j^i の要素 u は，制限値 K のもとで $|U_j^i| \leq K$ を満足するように保持される．このとき，U_j^i の要素は LRU により管理される．ただし，現在の上位の部分解 S_i に対応する R_j^i についての u は，S_i についてのコスト値の計算の期間に，U_j^i から削除されてはならない．そこで，S_i が変化したとき，再利用可能な U_j^i の要素は，紛失を防ぐために LRU キューの最後に移動される．

4.5 効率的解法の詳細

効率化手法を導入した解法を図 4 に示す．図中で用いられる関数と手続きは次のようである．

reset child states (S_j^i, U_j^i, R_j^i, $sFil_j^i$, $sChg_j^i$): 子 j について，それぞれ次のように更新する．$(S_j^i, sFil_j^i, sChg_j^i) \leftarrow (\phi, \text{false}, \text{false})$．$R_j^i$ を現在の S_i と D_j により更新する．また，$\exists r, r \in R_j^i, a(r) = a(u)$ なる U_j^i の要素 u 全てを，キャッシュ管理における LRU キューの最後に移動する．

compatible(U,S): U の全ての要素 u について，S が $a(s) = a(u)$ なる s を含むのであれば，真を返す．そうでなければ偽を返す．

update (U_i, $globalUB_i$) and test optimal condition: U_i の計算に変更点はない．S_i の各要素 s について，大域的なコスト値の上界 gub_s を計算する．まず，s 及び D_i の各値 d についての上界 $gub_{a(s)\cup\{(x_i,d)\}}$ が，次の各コスト値の合計により，得られる．

- $ub(s)$.
- i の局所コスト $\delta_i(a(s) \cup \{(x_i, d)\})$.
- $ub(u')$ s.t. $(a(u') \subseteq a(s) \cup \{(x_i, d)\}, u' \in U_j^i, x_j \in Chld_i)$ なる全ての $ub(u')$.

そして，$gub_s = \min_{d \in D_i} gub_{a(s)\cup\{(x_i,d)\}}$ により gub_s が得られる．gub_s は次の二つの目的に用いられる．

- $globalUB_i \leftarrow \min(gub_s, globalUB_i)$ なる $globalUB_i$ の更新．
- 以下に示す，最適解の決定の条件の一部．

$sOpt_i \wedge ((\exists u \in U_i, lb(u) = ub(u)) \vee ((\exists s \in S_i, gub_s = globalUB_i)) \wedge \neg opt_i$ であれば，最適解を決定する．ここで，$(\exists s \in S_i, gub_s = globalUB_i)$ であれば，

```
1  Main(){
2   Initialize().
3   until(forever){
4    until(receive loop is broken){ receive messages. }
5    if(S_i ≠ φ ∧ (sAct_i ∨ ¬cls_i)){ Maintenance(). } } }
6  Initialize(){
7   S_i ← φ. sAct_i ←false. sOpt_i ←false. sFin_i ←false.
8   sCls_i ←false. opt_i ←false. globalUB_i ← ∞. cls_i ←false.
9   D_i^* ← φ.
10  foreach j s.t. x_j ∈ Chld_i{
11   (S_j^i, U_j^i, R_j^i, sFil_j^i, sChg_j^i)←(φ, φ, φ, false, false). }
12  if(x_i is root){
13   send (VALUES, {(φ, 0, 0)}, true, true, true, false) to i. } }
14 Receive(VALUES, S, sOpt, sChg, sCls, sFin){
15  S_i ← S. sCls_i ← sCls.
16  if(sChg){ sAct_i ←true. sOpt_i ← sOpt. sFin_i ← sFin.
17   cls_i ←false.
18   foreach j s.t. x_j ∈ Chld_i{
19    reset child states (S_j^i, U_j^i, R_j^i, sFil_j^i, sChg_j^i). } } }
20 Receive(COSTS, j, U){
21  if(compatible(U,S_j^i)){ merge U to U_j^i. } }
22 Receive(GLOBAL_UB, globalUB){
23  globalUB_i ← min(globalUB_i, globalUB). }
24 Maintenance(){
25  update (U_i, globalUB_i) and test optimal condition.
26  if(opt_i ∧ (D_i has not been replaced by D_i^*)){
27   replace D_i by D_i^*.
28   if(x_i is root){ sFin_i ←true. }
29   foreach j s.t. x_j ∈ Chld_i{
30    reset child states (S_j^i, U_j^i, R_j^i, sFil_j^i, sChg_j^i). }
31   update (U_i, globalUB_i) and test optimal condition. }
32  MaintainChildrensContexts().
33  cls_i ← sCls_i ∧ (all children's contexts are closed).
34  foreach j s.t. x_j ∈ Chld_i{
35   send(VALUES,S_j^i, opt_i, sChg_j^i,cls_i,sFin_i) to j. }
36  if(sAct_i ∨ ¬sCls_i){
37   if(x_i is not root){ send (COSTS,i,U_i) to j s.t. x_j = prnt_i. }
38   if(closed(U_i)){ sAct_i ←false. } }
39  foreach j s.t. x_j ∈ Chld_i{
40   send (GLOBAL_UB,globalUB_i) to j. } }
41 MaintainChildrensContexts(){
42  foreach j s.t. x_j ∈ Chld_i{
43   update R_j^i. update current contexts S_j^i.
44   if(¬(contexts of j are active) ∧ ¬sFil_j^i){
45    set next contexts S_j^i.
46    if(S_j^i is last part of R_j^i){sFil_j^i ←true.}
47    sChg_j^i ←true. }
48   else{ sChg_j^i ←false. } } }
```

図 4　効率化手法を伴う解法
Fig. 4　Efficient solution method.

$gub_{a(s)\cup\{(x_i,d^*)\}} = gub_s$ なる d^* を，x_i の最適値とする．そうでなければ，従来と同様に最適解を決定する．

closed(U_i): 従来の条件に加えて，S_i の要素 s についての，大域的なコスト値の下界 glb_s を用いる．glb_s の計算は，上界の代わりに下界を用いることを除いて，gub_s の計算と同様である．ここで，$sFin_i$ により場合分けされる条件 $cond\_glb_s$ を次のように定義する．

$$cond\_glb_s \triangleq \begin{cases} glb_s > globalUB_i & sFin_i \\ glb_s \geq globalUB_i & \text{otherwise} \end{cases} \tag{4}$$

U_i の要素 u 全てについて，$lb(u) = ub(u) \vee (cond\_glb_s$ where $a(s) = a(u))$ であれば closed(U_i) は真を返す．そうでなければ偽を返す．

set next contexts S_j^i: opt_i が偽の場合は次のように S_j^i を更新する．${R_j^i}^+$ を，R_j^i から探索済み及び枝刈りされる要素を除いた，集合とする．${R_j^i}^+$ から S_j^i を決定し，S_j^i の要素 s について $a(s) = a(u)$ なる要素 u がなければ，$(a(s), 0, \infty)$ を U_j^i に加える．R_j^i から ${R_j^i}^+$ を得る計算は次のようである．まず，R_j^i の要素 r について，コスト値の下界 $glb_r = lb(r) + lb(u)$ を求める．ここで，u は $a(r) = a(u)$ なる U_j^i の要素である．式 (4) と同様に定義される条件 $cond\_glb_r$ により，$lb(u) = ub(u) \vee cond\_glb_r$ であれば，r は ${R_j^i}^+$ には含まれない．opt_i が真の場合は従来と同様である．

update R_j^i: R_j^i を現在の S_i に基づいて再計算する．

update current contexts S_j^i: 現在の R_j^i に基づいて，S_j^i の要素 s 全ての，$lb(s)$ と $ub(s)$ を再計算する．

all children's contexts are closed: i の子 j 全ての，S_j^i の要素 s 全てについて，$lb(s) = ub(s)$ であれば真を，そうでなければ偽を返す．

contexts of j are active: S_j^i のある要素 s について，$a(u) = a(s)$ なる U_j^i の要素 u が，$lb(u) \neq ub(u) \wedge \neg cond\_glb_s$ のとき真を，そうでなければ偽を返す．

merge U to U'，及び COSTS メッセージに変更はない．

4.6　冗長なメッセージの抑制

図 3，4 に示したアルゴリズムでは，同一の内容のメッセージが連続して送信されうるが，重複するメッセージの送信は抑制してもよい．ただし，状況の変化を確実に伝達するために，VALUES メッセージを受信したとき (図 4 の場合は $sChg_j^i$ が真のときのみ)，及び最適解が決定されたときは，メッセージを送信する．

5.　アルゴリズムの正しさ

提案手法は，疑似木にもとづく従来の解法と本質的な計算は同様である．ボトムアップなコスト値の計算と，トップダウンなコスト値の計算は，現在の部分解のみに従い，部分解が変化しない限り，それぞれのコスト値の境界を単調に狭める．現在の部分解には各エージェントの祖先のエージェントの決定がいずれ反映される．以上により，枝刈りがなければコスト値の境界は必ず最適値に収束する．

枝刈りは，探索を一時的に省略するのみであり，コスト値の境界を破壊しない．また，枝刈りはエージェントが知る現在の状態のみに従う．大域的なコスト値の上界値は単調に減少し，速やかに伝搬する．更に，最適解の決定の際の枝刈りの条件の変更は，$sFin_i$ として速やかに伝搬する．したがって，枝刈りが過剰である状況はいずれ解消される．

非同期処理のためにメッセージの送受信の連鎖が途切れないように，部分解かその状態が変更された際に必ずメッセージを送信すれば，解法は必ず最適解を決定して収束する．

6. 評　　価

実験により提案手法の有効性を評価した．例題として，n 個の 3 値変数と，$l \cdot n$ 個の 2 項制約/関数からなる問題を用いた．問題の種類は次の二つである．

max-csp-gcl: グラフの頂点彩色問題に対する最大制約充足問題を模倣する．評価関数 $f_{i,j}(x_i, x_j)$ の値は，$x_i = x_j$ であれば 1，そうでなければ 0 とする．

cop: 評価関数 $f_{i,j}(x_i, x_j)$ の値は，$[0, 10]$ の整数値のいずれかを一様分布に従ってランダムに選ぶ．

これらと類似する問題は既存研究の評価でも用いられている [1], [3], [4]．制約網は単一連結成分のランダムなグラフとし，各 20 例についての結果を平均した．

評価対象として，b (図 3 の手法)，c (b と U_j^i のキャッシュの併用)，ppgc (図 4 の手法)，ppg (ppgc の U_j^i のキャッシュを除く)，pp (ppg の最適解決定における $globalUB_i$ の条件を除く)，pl (pp の枝刈りにおける下界値から祖先経由のコスト値を除く) の各手法を用いた．各手法ともに，解の展開は，固定の変数順序，変数値順序と深さ優先探索に従った．展開する部分解の数 $|S_j^i|$ の最大値 $L = 1, 3, 9, 27$，キャッシュにおける $|U_j^i|$ の最大値 $K = L \cdot |D_i|$ とした．

提案手法は従来手法の一般化であると考えられる．このうち，$|S_j^i| = 1$ の場合は，従来手法 [3], [9], [10] などに類似すると考えられる．また，$|S_j^i| = 1$ が十分に大きければ動的計画法 [4] に類似すると考えられるが，純粋な動的計画法は，制約密度が比較的小さい場合を除いて，induced width [4] とともにメッセージサイズや表の計算が指数関数的に増加するため，実際的には実行が困難である．ここでは，実装手法や，細かな探索戦略の影響を除外することを意図して，従来手法の実装との比較は行わない．

探索の反復回数はメッセージサイクル数により評価した．各メッセージサイクルでは，各エージェントは受信キューの全てのメッセージを取りだし，処理を行い，必要であればメッセージを送信キューに追加する．その後，メッセージの交換を行い，次のメッセージサイクルに移行する．実験は 10^5 サイクルで中断し，中断した場合は 10^5 サイクル要したものとした．

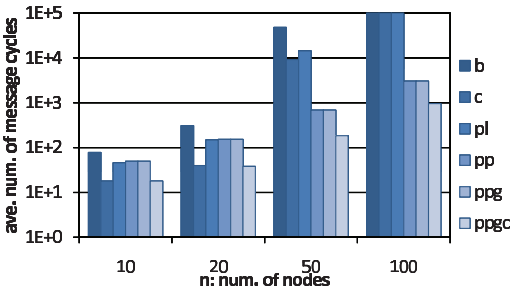

図 5　max-csp-gcl ($l = 1.5, L = 27$)
Fig. 5　max-csp-gcl. ($l = 1.5, L = 27$)

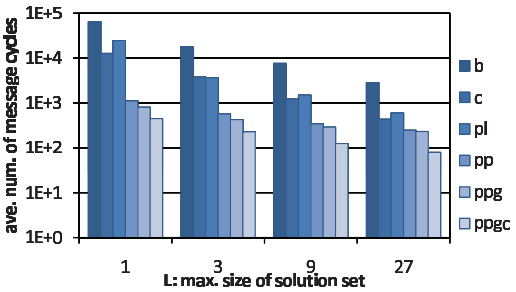

図 6　max-csp-gcl ($n = 20, l = 2$)
Fig. 6　max-csp-gcl. ($n = 20, l = 2$)

実験では提案手法を実装したシミュレーションプログラムを用いた．実行環境のハードウェアとして，Intel Core i7 CPU 920@2.67GHz, 6GB memory または Intel Core i7 CPU 930@2.80GHz, 3GB memory の複数の計算機を用いた．またソフトウェアとして，Fedora 10 Linux x64 及び g++ 3.0 を用いた．なお，シミュレーションにおける各手法の処理時間は，後述するように論理的な指標 (nccops) に基づいて評価されるため，実行環境の差異には影響されない．

max-csp-gcl におけるメッセージサイクル数の比較を示す．図 5 に，$l = 1.5, L = 27$ の場合についての各手法の比較を示す．$n = 50, 100$ の問題では，枝刈りと U_j^i のキャッシュを併用する効果が得られた．n が増加するにつれ，induced-width が増加する傾向があるため，キャッシュのみ用いる c の効果が小さくなると考えられる．図 6 に，$n = 20, l = 2$ の場合を示す．この

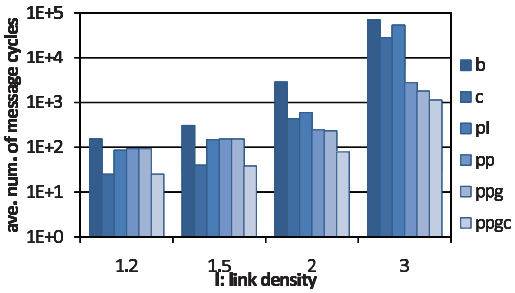

図 7　max-csp-gcl ($n = 20, L = 27$)
Fig. 7　max-csp-gcl. ($n = 20, L = 27$)

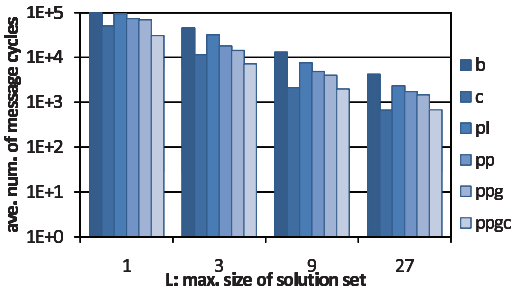

図 8　cop ($n = 20, l = 2$)
Fig. 8　cop. ($n = 20, l = 2$)

結果では，枝刈りを用いる手法においても，L の増加の効果が比較的大きかった．図 7 に，$n = 20, L = 27$ の場合を示す．l の増加に従って，induced-width が増加する傾向があるため，図 5 の n の増加の場合に類似する．枝刈りが有効な場合は pl, pp, ppg の順にメッセージサイクル数が削減される．

cop における $n = 20, l = 2$ の場合のメッセージサイクル数の比較を図 8 に示す．この問題では，キャッシュの効果が枝刈りの効果より顕著である．L を増加する効果は得られるが，$L = 1, 3$ などの枝刈りへの依存が比較的大きい場合でも，c と ppgc の差は比較的小さい．この傾向は他の結果でも同様であった．また，キャッシュと枝刈りを併用することでキャッシュのみよりもサイクル数が増加する場合があった．これは，枝刈りによりキャッシュの内容が乱れる影響のためと考えられる．

疑似木の規模を表 1 に示す．l, n の増加により induced-width が増加し，最大の部分問題の規模 sz が増加する．メッセージサイクルあたりの，メッセージ数及び，1 個以上のメッセージの送受信があった送信元と宛先の組の数を，表 2 に示す．メッセージの受信がないエージェントは静止状態であること，及び

表 1　疑似木のサイズ
Table 1　size of pseudo-trees.

l	1.2			1.5			2			3		
n	td	tb	sz	td	tb	sz	td	tb	sz	td	tb	sz
10	5	1.4	14	6	1.2	32	7	1.2	89	9	1.0	729
20	9	1.5	41	11	1.4	192	13	1.3	2138	15	1.2	84637
50	18	1.6	1334	21	1.5	1.6×10^5	29	1.4	1.5×10^7	34	1.3	1.4×10^{11}
100	30	1.7	32295	42	1.5	5.9×10^9	51	1.4	9.7×10^{14}	64	1.3	2.5×10^{21}

td: depth, tb: branching factor without leaf,
sz: $\max_{x_i} \prod_{k \in Ast_i} |D_k|$

表 2　メッセージサイクルあたりのメッセージ数
Table 2　number of messages per message cycle.
(max-csp-gcl, $n = 50, l = 1.5, L = 9$)

alg.	num. of act. pairs	num. of messages			
		all	VALUES	COSTS	GLOBAL_UB
b	9.3	9.3	4.4	5.0	0
c	3.0	3.0	1.3	1.6	0
pl	4.0	4.1	1.4	2.5	0.1
pp	7.8	8.2	4.4	3.0	0.8
ppg	7.8	8.2	4.4	3.0	0.8
ppgc	6.9	8.1	3.5	2.3	2.3

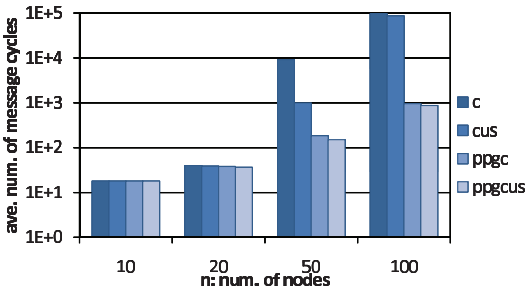

図 9　max-csp-gcl ($l = 1.5, L = 27$)
Fig. 9　max-csp-gcl. ($l = 1.5, L = 27$)

4.6 の冗長なメッセージの抑制により，エージェント数と比較して通信の頻度は小さい．

図 3 と 4 に示した解法では，現在の部分解の集合 S_j^i は全ての要素についてのコストの境界が閉じてから，次の要素に更新された．しかし，S_j^i の要素 s についてのコストの境界が閉じたときは，s を個別に次の要素に更新できる．そこで，このように c と ppgc を拡張した手法を評価し，結果を図 9 に示す．cus 及び ppgcus がそれぞれ拡張された手法である．c のメッセージサイクル数は削減された場合 ($n = 50$) がある一方で，枝刈りを用いる ppgc に対する削減の効果は小さかった．

表 3 に，制約を評価する処理量を考慮した比較を示す．関連研究で用いられる NCCCs (並行せずに実行される制約チェックの回数) [17] と同様の指標として，評価関数の判定回数と部分解の比較回数の総和に基づく量

表 3　制約評価の処理量を考慮した比較
Table 3　comparison based on number of constraint operations.

(max-csp-gcl, $n = 50, l = 1.5$)

L	9				27			
alg	nccops ($\times 10^5$)	iter.	$\alpha^\perp$		nccops ($\times 10^5$)	iter.	$\alpha^\perp$	
			to b	to prev.			to b	to prev.
b	366.9	62522	-	-	1675	47429	-	-
c	503.8	18518	311	311	1948	9157	714	714
cus	52.5	1654	0	0	248	996	0	0
pl	160.3	24846	0	-	601	14265	0	-
pp	8.2	721	0	0	38	681	0	0
ppg	8.2	721	0	-	38	681	0	-
ppgc	8.1	213	0	0	41	184	0	639
ppgcus	11.4	188	0	13479	58	151	0	52861

In 'to prev.', pp is compared to cus.

(nccops) を用いた(注2)．また，nccops，メッセージサイクル数 (iter) 及び，パラメータ α (ただし $\alpha \geq 0$) により，処理時間を nccops+α·iter のように見積もった．基準とする結果を nccops_0 及び iter_0 (ただし $\text{iter}_0 > \text{iter}$) とするとき，処理時間が基準の場合以下となる α の最小値 $\alpha^\perp$ を，$\max(0, (\text{nccops} - \text{nccops}_0)/(\text{iter}_0 - \text{iter}))$ により求めた．比較の基準は b 及び表において 1 行上の結果 (prev.) とした．表の結果は，c を除く各効率化手法が b よりも nccops を削減したことを示している．また，処理時間の比較の大半では，メッセージ交換の時間を無視 ($\alpha = 0$) しても処理時間が削減されることが示されている．ppgcus は部分解の集合の管理により多くの処理を要する一方で，メッセージサイクル数の削減の程度は小さいため，ppgc よりも処理時間を短縮するためには，比較的大きな nccops がメッセージ交換の時間とトレードオフされる必要がある．

提案手法は複数の部分解の情報を単一のメッセージに集約することによる通信オーバヘッドの削減を想定している．メッセージのサイズにおいては，部分解とコストの上下解からなる表が支配的である．この表の各要素では，部分解は (x, d) を表す二つの整数のペアからなる集合として表現され，コストの上下界値がそれぞれ一つの整数として表現される．このとき，変数の数を n とすれば，各要素のサイズは $2n + 1 + 1$ である．これらの整数のうち，各 x, d を 1byte，各上下界値を 4bytes で表現すれば，$n = 100$ における各要素の実際のサイズは 208bytes である．これは，イーサネットのジャンボフレームにおけるデータの一般的なサイズ 8kbytes に 39 個収まる．したがって，上記の評価において同時に送受される部分解の数 L は実際に可能な程度である．

(注2)：提案手法は単純な制約/評価関数のチェックのみではなく，部分解の比較を多用する計算を伴うため，これらを含めた処理の回数を用いた．

7. むすび

本研究では，疑似木に基づく分散制約最適化問題の厳密解法において，枝刈り及び，記憶とメッセージのサイズの活用による通信回数の削減を目的として，複数の部分解を同時に展開し送受信する解法を示した．

また，この解法では，分枝限定法と動的計画法を併用する解法を一般化するにあたり，解の展開において副問題の解空間を考慮する部分解の再構成及び，枝刈りのための大域的コスト値の上下界の計算，を明示的に用いた．このような解法の表現は，今後のアルゴリズムの検討の基礎として一定の意義があると考える．

提案手法を実験により評価し，手法を構成する各要素の探索における有効性について考察した．特に，複数解の同時展開，枝刈り及びキャッシュがそれぞれ効果的な問題では，提案手法が有効であることが示された．

今後の課題として，解集合の展開における複数の探索戦略の導入，及び他の効率化手法の併用による効果の評価が挙げられる．

謝辞　本研究の一部は，科学研究費補助金 (若手研究 (B)22700144，基盤研究 (C)25330257)，柏森情報科学振興財団研究助成，平成 23 年度人工知能研究振興財団研究助成による．

文　献

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

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

[3] P.J. Modi, W. Shen, M. Tambe, and M. Yokoo, "Adopt: Asynchronous distributed constraint optimization with quality guarantees," Artif. Intell., vol.161, no.1-2, pp.149–180, 2005.

[4] A. Petcu and B. Faltings, "A scalable method for multiagent constraint optimization," 19th International Joint Conference on Artificial Intelligence, pp.266–271, 2005.

[5] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg, "Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks,"

Artif. Intell., vol.161, no.1-2, pp.55–87, 2005.
[6] S. Miller, S.D. Ramchurn, and A. Rogers, “Optimal decentralised dispatch of embedded generation in the smart grid,” 11th International Conference on Autonomous Agents and Multiagent Systems, vol.1, pp.281–288, 2012.
[7] R.T. Maheswaran, M. Tambe, E. Bowring, J.P. Pearce, and P. Varakantham, “Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling,” 3rd International Joint Conference on Autonomous Agents and Multiagent Systems, pp.310–317, 2004.
[8] A. Petcu and B. Faltings, “MB-DPOP: A new memory-bounded algorithm for distributed optimization,” 20th International Joint Conference on Artificial Intelligence, pp.1452–1457, 2007.
[9] A. Petcu and B. Faltings, “O-DPOP: An algorithm for open/distributed constraint optimization,” National Conference on Artificial Intelligence, pp.703–708, 2006.
[10] B. Ottens and B. Faltings, “Asynchronous open DPOP,” 10th International Workshop on Distributed Constraint Reasoning, 2008.
[11] 松井俊浩，松尾啓志，“分散制約最適化手法における複数解の展開と枝刈りの併用,” JAWS2010 合同エージェントワークショップ&シンポジウム，2010.
[12] T. Matsui and H. Matsuo, “A distributed cooperative search algorithm using multiple contexts and pruning,” 26th ISCA International Conference on Computers and Their Applications, pp.1–8, 2011.
[13] E.C. Freuder, “A sufficient condition for backtrack-bounded search,” J. Assoc. Comput. Mach., vol.32, no.14, pp.755–761, 1985.
[14] T. Schiex, “A note on csp graph parameters,” Technical report 1999/03, INRA, 1999.
[15] M.C. Silaghi and M. Yokoo, “ADOPTing: unifying asynchronous distributed optimization with asynchronous backtracking,” Journal of Autonomous Agents and Multi-Agent Systems, vol.19, no.2, pp.89–123, Oct. 2009.
[16] W. Yeoh, P. Varakantham, and S. Koenig, “Caching schemes for DCOP search algorithms,” 8th International Conference on Autonomous Agents and Multiagent Systems, pp.609–616, 2009.
[17] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan, “Comparing performance of distributed constraints processing algorithms,” 3rd International Workshop on Distributed Constraint Reasoning, 2002.
（平成 25 年 10 月 27 日受付，26 年 3 月 14 日再受付）

松井　俊浩　（正員）

1995 年名古屋工業大学電気情報工学科卒業．1999 年同大学大学院博士前期課程修了．2006 年同博士後期課程修了．同年名古屋工業大学情報基盤センター助手．2007 年同助教．2011 年同准教授．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)．情報処理学会，人工知能学会，各会員．

松尾　啓志　（正員）

1983 年名古屋工業大学情報工学科卒業．1989 年同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 年同大学院教授．2006 年情報基盤センターセンター長 (併任)．現在に至る．分散システムに関する研究に従事．工学博士．情報処理学会，人工知能学会，IEEE 各会員．