# 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

松井　俊浩[†a)]　　松尾　啓志[†b)]

# A distributed constraint optimization method using multiple contexts and pruning

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

**Abstract.** 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問題解決の基礎的な表現として，研究されている．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必要とするため，通信のレイテンシの影響が大きい．純粋な動的計画法に基づく解法は，反復的な探索を必要としないが，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．特に，通信のオーバヘッドが大きい環境では，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．

**Keywords.** 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

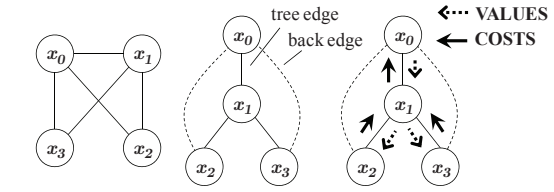
分散制約最適化問題 (DCOP) [1], [3], [4], [5] は，分散資源スケジューリングや分散センサ網などを含む，マルチエージェントシステムにおける問題解決の基礎的な表現として，研究されている．DCOP の厳密解法として，木探索や動的計画法にもとづく分散アルゴリズムが提案されている．木探索に基づく解法 [4] は，メッセージ通信を介する反復的な処理を必要とするため，通信のレイテンシの影響が大きい．純粋な動的計画法に基づく解法 [5] は，反復的な探索を必要としないが，必要とする記憶とメッセージのサイズは，制約網に対する疑似木の induced-width に対して指数関数的である．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 [4] 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 [6], [7]．特に，通信のオーバヘッドが大きい環境では，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，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．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するが，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．混合的な探索戦略を用いる解法の基礎検討としての意義もあると考えられる．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および，枝刈りのための大域的コスト値の上下界の計算，を明示的に用いる方法を示す．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．

† 名古屋工業大学，〒 466-8555 愛知県名古屋市昭和区御器所町
a) E-mail: matsui.t@nitech.ac.jp
b) E-mail: matsuo@nitech.ac.jp

## 2. 背　　景

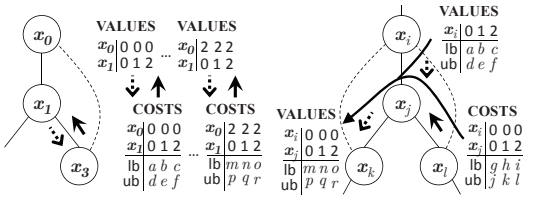
### 2.1 分散制約最適化問題

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 $C$ およ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 $x_i$ を持つ．$x_i$ は離散値の集合 $D_i$ に含まれる値




(a) constraint network (b) pseudo-tree (c) messages of solver

図 1 疑似木とメッセージのパス




(a) expanding multiple solutions (b) top-down cost for $x_k$

図 2 疑似木に基づくコスト値の計算

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 $f_{i,j}(d_i, d_j) : D_i \times D_j \rightarrow \mathbb{N}_0$ 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．最適解を求める探索処理は，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表される.

### 2.2 疑似木 (pseudo-tree)

疑似木 [2], [8] 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．たとえば図 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，疑似木の各頂点および，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.

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

$\boldsymbol{PP_i}$**:** $x_i$ の祖先の変数の集合の部分集合．$x_k \in PP_i$ 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無いため，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 [4], [5] 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，$PP_i$ 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 $s_i$，および $x_i$ 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.

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

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

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 $\infty$ 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 [5] 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無いが，各エージェント $i$ は $PP_i$ 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 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 [6], [7] 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 [4] 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．メッセージ通信のオーバヘッドが大きい環境では，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 [9] や，計算結果のキャッシュによる反復処理回数の削減 [10] は提案されている

```
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^i_j, U^i_j, R^i_j, sFil^i_j)←(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^i_j, U^i_j, R^i_j, sFil^i_j). } }
15 Receive(COSTS, j, U){
16  if(compatible(U,S^i_j)){ merge U to U^i_j. } }
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^i_j, U^i_j, R^i_j, sFil^i_j). }
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^i_j){
30    set next contexts S^i_j.
31    if(S^i_j is last part of R^i_j){sFil^i_j ←true.}
32    send(VALUES,S^i_j, opt_i) to j. } } }
```

図 3　multi-solution search

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．

### 3.1 基本的な解法

まず，大域的なコスト値による枝刈りを用いない，基本的な解法を図 3 に示す．この解法では，図 1(c) に示す 2 種類のメッセージを用いる．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，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り，最適な変数値が選択される．残された部分問題について，同様に，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，次節以降で，要点について説明を加える．

DCOP およ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．

**$S_i$:** $PP_i$ 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．$ub(u)$ および $lb(u)$ 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.

**$R^i_j$:** $S_i$ をもとに，子 $j$ の $PP_j$ について生成される，部分解の集合. $R^i_j$ 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，$PP_j$ に含まれない変数値の割り当てを除いた，部分解である．$R^i_j$ の要素は少なくとも一つの $R_i$ の要素に対応する．

**$S^i_j$:** 子 $j$ に送信する，$R^i_j$ の部分集合.

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

**$sFil^i_j$:** $S^i_j$ が全ての $R^i_j$ 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，$S^i_j$ および，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 $S^j_i$ に対応する $U_i$ を送信する．

図 3 で用いられる関数と手続きの詳細を次に示す．

**reset child states ($S^i_j, U^i_j, R^i_j, sFil^i_j$):** 子 $j$, について，$(S^i_j, U^i_j, sFil^i_j)$←$(\phi, \phi,$ false) とし，$R^i_j$ 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:** $S_i$ の要素 $s$ 全てについて，$(s, lb(u_s), ub(u_s))$ 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，$i$ の局所コストと，$lb(u')$ s.t. $(a(u') \subseteqq s \cup \{(x_i, d)\}, u' \in U_j^i, x_j \in Chld_i)$ 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 $\infty$ とする．$U_i$ 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．$sOpt_i$ が真のとき，$|S_i| = |U_i| = 1$ となる．すなわち最適解は一つのみ選択される．

**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 $S_j^i$:** $opt_i$ が偽の場合は次のように $S_j^i$ を更新する．$R_j^{i+}$ を $R_j^i \backslash \{a(u) | u \in U_j^i\}$ なる集合とする．$S_j^i$ を，$S_j^i \subseteqq R_j^{i+} \wedge |S_j^i| \leqq L$ を満足するように，更新する．ここで $L$ は展開する解の数を制限するパラメータである．各部分解のコストの初期値 $\{u | a(u) \in S_j^i, lb(u) = 0, ub(u) = \infty\}$ を $U_j^i$ に加える．$opt_i$ が真の場合は $R_j^i$ に唯一含まれる部分解 $r$ を $S_j^i$ の要素とする．$a(u) = r$ なる要素 $u$ が $U_j^i$ に含まれなければ，$(a(r), 0, \infty)$ を $U_j^i$ に加える．

**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．

### 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 メッセージで通知する．$i$ は $S_i$ の副問題の部分解を $j$ について展開する際に，$PP_j$ に含まれる変数についての解の集合 $R_j^i$ の要素を網羅するように，$S_j^i$ を決定する．$j$ を根とする部分木のコスト値の計算は，$R_j^i$ 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，$PP_i$ に含まれない $x_i$ の祖先の変数値が $i$ に漏えいしない．

### 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を受信し，$U_j^i$ 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，計算される．$i$ 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．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，非同期的に $S_j^i$ を次の部分解の集合に変更し VALUES メッセージにより送信する．また，部分解 $S_i$ が変更されたとき，$U_j^i \leftarrow \phi$ とする．

```
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. gUB_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(GUB, gUB){
23  gUB_i ← max(gUB_i, gUB). }
24 Maintenance(){
25  update (U_i, gUB_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, gUB_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{ send (GUB,gUB_i) to j. } }
40 MaintainChildrensContexts(){
41  foreach j s.t. x_j ∈ Chld_i{
42   update R_j^i. update current contexts S_j^i.
43   if(¬(contexts of j are active) ∧ ¬sFil_j^i){
44    set next contexts S_j^i.
45    if(S_j^i is last part of R_j^i){sFil_j^i ←true.}
46    sChg_j^i ←true. }
47   else{ sChg_j^i ←false. } } }
```

図 4　multi-solution search with efficient methods

## 4. 効率化手法の導入

本節では，3. 節の解法に，効率化手法を導入する．

### 4.1 大域的コストによる枝刈りとキャッシュの導入

基本的な解法に効率化手法を導入した解法を図 4 に示す．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，図 2(b) に示すように，祖先の変数および，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．このために VALUES メッセージもコスト値を伝搬する．また，大域的なコスト値の最良の上界を伝搬する GUB メッセージが導入される．さらに，計算結果が再利用される．以下では，特に解法の変更点を説明し，次節以降で要点について説明を加える．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．

- $(a(r)\backslash\{(x_i,d)\})\subseteqq s$ なる $S_i$ の要素 $s$ の $lb(s)$.
- $i$ の局所コスト $\delta_i(s\cup\{(x_i,d)\})$.
- $lb(u')$ s.t. $(a(u')\subseteqq s\cup\{(x_i,d)\}, u'\in U_k^i, x_k\in Chld_i\backslash\{x_j\})$ 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．$(a(r)\backslash\{(x_i,d)\})\subseteqq s$ なる $S_i$ の要素 $s$ が複数ある場合，各 $s$ に対して計算された $lb(r)$ と 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が，キャッシュとしても扱う．$S_i$ 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，$|U_j^i|\leqq K$ を満足しなければ，LRU に従って要素が削除される．ただし $K\geqq|R_j^i|$ であるものとする．

また，次の要素が新たに用いられる．

**$gUB_i$:** コスト値の大域的な上界であり，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．

**$sChg_j^i$:** $S_i$ および $S_i$ 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．

メッセージの変更点は次のようである．

**VALUES:** $S_j^i$ および $S_j^i$ 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．

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

図 4 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．($S_j^i$, $sFil_j^i$, $sChg_j^i$)←($\phi$, false, 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 ($U_i$, $gUB_i$) 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，$s$ および $D_i$ 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. $(a(u')\subseteqq a(s)\cup\{(x_i,d)\}, u'\in U_j^i, x_j\in Chld_i)$ 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 2 つの目的に用いられる．

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

$sOpt_i\wedge((\exists u\in U_i, lb(u)=ub(u))\vee((\exists s\in S_i, gub_s=gUB_i))\wedge\neg opt_i$ であれば，最適解を決定する．ここで，$(\exists s\in S_i, gub_s=gUB_i)$ であれば，$gub_{a(s)\cup\{(x_i,d^*)\}}=gub_s$ なる $d^*$ を，$x_i$ の最適値とする．そうでなければ，従来と同様である．

**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．

$$cond\_glb_s \triangleq \begin{cases} glb_s > gUB_i & sFin_i \\ glb_s \geqq gUB_i & \text{otherwise} \end{cases} \quad (4)$$

$U_i$ 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，$R_j^i$ から探索済みおよ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．$opt_i$ が偽の場合は従来と同様である．

**update $R_j^i$:** $R_j^i$ を現在の $S_i$ にもとづいて再計算する．

**update current contexts $S_j^i$:** 現在の $R_j^i$ にもとづいて，$S_j^i$ 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，$lb(u) \neq ub(u) \wedge \neg cond\_lb_s$ のとき真を，そうでなければ偽を返す．

$U_i$, $sOpt_i$, $opt_i$, $D_i^*$, $sAct_i$, $sFil_j^i$, merge $U$ to $U'$, および COSTS メッセージに変更はない．

### 4.2 部分解のコスト値のトップダウンな計算

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，各変数の祖先およ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 $a(s)$ およ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，$S_i$，$j$ を除く子 $k$ 全てについての $U_k^i$ のコスト，および $i$ の局所コストを加算することにより得られる．$S_i$ の複数の要素が $R_j^i$ の一つの要素に対応する場合は，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別に $cls_i, sCls_i$ により収束を管理する．

### 4.3 大域的なコストによる枝刈り

エージェント $i$ は大域的なコスト値の上界 $gUB_i$ を最小化する．$gUB_i$ は，GUB メッセージにより他のエージェントに伝搬される．$i$ は，現在の解のコスト値の下界と $gUB_i$ 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では下界が大域的上界以上であること，最適解の決定の段階では下界が大域的上界を上回ることとする．枝刈りの条件の変更は，$sFin_i$ の伝搬による．

### 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，$|U_j^i|$ の制限内で保持される．$U_j^i$ の要素は，LRU により管理される．ただし，現在の $S_i$ 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．$S_i$ 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 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 $sChg$ が真のときのみ)，およ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

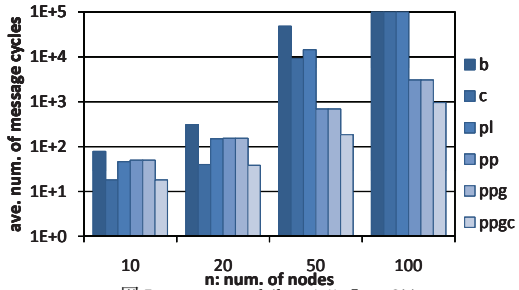


図 5　max-csp-gcl ($l = 1.5, L = 27$)

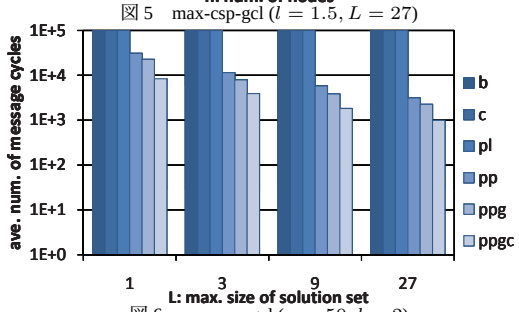


図 6　max-csp-gcl ($n = 50, l = 2$)

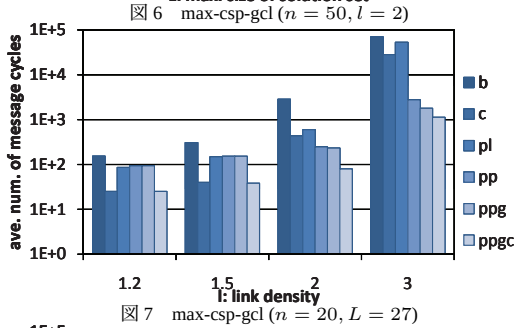


図 7　max-csp-gcl ($n = 20, L = 27$)

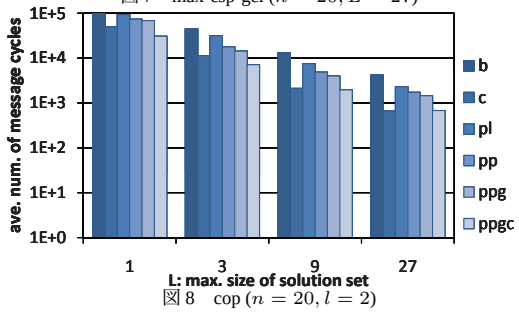


図 8　cop ($n = 20, l = 2$)

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．最適解の決定の際の枝刈りの条件の変更は，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，探索は必ず収束する．

## 6. 評　　価

実験により提案手法の有効性を評価した．$n$ 個の 3 値変数と，$l \cdot n$ 個の 2 項制約/関数からなる次の種類の例題を用いた．ここでは，$l = 1.2, 1.5, 2, 3$ とした．

**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したがってランダムに選ぶ．

ここでは単一連結成分のランダムなグラフ各 20 例についての結果を平均したが，疑似木の induced-width の最大値が結果に大きく影響すると考えられる．

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 $gUB_i$ 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) を比較した．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．

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．

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，$n = 50, l = 2$ の場合を示す．この結果では，枝刈りを用いる手法において，$L$ の増加の効果が比較的顕著であった．図 7 に，$n = 20, L = 27$ の場合を示す．$l$ の増加にしたがって，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

表 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 | $1.6\times10^5$ | 29 | 1.4 | $1.5\times10^7$ | 34 | 1.3 | $1.4\times10^{11}$ |
| 100 | 30 | 1.7 | 32295 | 42 | 1.5 | $5.9\times10^9$ | 51 | 1.4 | $9.7\times10^{14}$ | 64 | 1.3 | $2.5\times10^{21}$ |

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

表 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 | GUB |
| 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 |

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，メッセージ数および，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，および 4.5 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．

## 7. あとがき

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，枝刈りおよび，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および，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，枝刈りおよ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．本論文では，同時に展開する部分解の数をパラメータ $L$ により表した．$L$ の値は $|PP_i|$ と $|D_i|$ を考慮して設定されるべきである．また，各エージェントごとに異なる値に設定できる．解集合の展開においては，初期の検討として，固定された変数順序と変数値順序にもとづいて部分解を列挙した．他の戦略による多点探索および，その戦略に適切な LRU 以外のキャッシュ [10] の検討がさらに必要である．

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，およ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)，柏森情報科学振興財団の助成を受けて遂行された．

## 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. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 639–646, 2008.

[2] Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(14):755–761, 1985.

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

[4] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2005.

[5] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *9th International Joint Conference on Artificial Intelligence*, pages 266–271, 2005.

[6] Adrian Petcu and Boi Faltings. O-dpop: An algorithm for open/distributed constraint optimization. In *National Conference on Artificial Intelligence*, pages 703–708, 2006.

[7] Adrian Petcu and Boi Faltings. Mb-dpop: A new memory-bounded algorithm for distributed optimization. In *20th International Joint Conference on Artificial Intelligence*, pages 1452–1457, 2007.

[8] Thomas Schiex. A note on csp graph parameters. *Technical report 1999/03, INRA*, 1999.

[9] Marius C. Silaghi and Makoto Yokoo. Adopt-ing: unifying asynchronous distributed optimization with asynchronous backtracking. *Journal of Autonomous Agents and Multi-Agent Systems*, 19(2):89–123, 10 2009.

[10] William Yeoh, Pradeep Varakantham, and Sven Koenig. Caching schemes for dcop search algorithms. In *8th International Conference on Autonomous Agents and Multiagent Systems*, pages 609–616, 2009.