論　文

Technical Papers

分散不完全制約充足問題

Distributed Partial Constraint Satisfaction Problem

平山　勝敏*1　　横尾　真*2
Katsutoshi Hirayama　Makoto Yokoo

*1 神戸商船大学
Kobe University of Mercantile Marine, Kobe 658–0022, Japan.
*2 NTTコミュニケーション科学基礎研究所
NTT Communication Science Laboratories, Kyoto 619–0237, Japan.

1998年4月9日　受理

Keywords: constraint satisfaction problem, distributed CSP, partial CSP, multiagent system.

Summary

Many problems in multi-agent systems can be described as *distributed Constraint Satisfaction Problems* (distributed CSPs), where the goal is to find a set of assignments to variables that satisfies all constraints among agents. However, when real-life problems are formalized as distributed CSPs, they are often over-constrained and have no solution that satisfies all constraints. This paper provides the *Distributed Partial Constraint Satisfaction Problem* (DPCSP) as a new framework for dealing with over-constrained situations. We also present new algorithms for solving *Distributed Maximal Constraint Satisfaction Problems* (DMCSPs), which belong to an important class of DPCSP. These algorithms are called the *Synchronous Branch and Bound* (SBB) and the *Iterative Distributed Breakout* (IDB). Both algorithms were tested on hard classes of over-constrained random binary distributed CSPs. The results can be summarized as SBB is preferable when we are mainly concerned with the optimality of a solution, while IDB is preferable when we want to get a nearly optimal solution quickly.

1. はじめに

AIにおける多くの問題は制約充足問題（Constraint Satisfaction Problem: CSP）として定式化できるため，これまでに多くの研究者が，その性質やアルゴリズムに関する理論的，実験的な研究を行なってきた．しかしながら，より現実的な問題を扱う場合には，必ずしもCSPの記述力は十分とは言えず，近年，その枠組を拡張する試みがなされている [横尾 97]．

[横尾 92, Yokoo 98a] では，複数のエージェントが関わる問題を扱う一般的な枠組として分散制約充足問題（分散CSP）が提案されている．分散CSPでは，CSPにおける変数および制約が複数エージェントに分散されており，エージェントが互いに情報交換しながら，すべての制約を満たすことが要求される．マルチエージェントシステムにおける多くの問題，例えば，分散解釈問題 [Lesser 83]，分散資源配分問題 [Conry 91]，分散スケジューリング問題 [Sycara 91]，マルチエージェントTMS[Huhns 91] などは分散CSPとして定式化される．

一方，設計者が現実の問題をCSPとして定式化する際，制約が強過ぎて解がないという場合がよくあるが，このような過制約なCSPに対し，従来の制約充足アルゴリズムは，ただ解がないことを発見するのみである．そのため，変数への何らかの値の割り当てを求めるためには，設計者自らが不要な制約を削るなどして，CSPが過制約にならないよう注意深く記述する必要がある．このような過制約なCSPを扱う一般的な枠組の一つとして，[Freuder 89, Freuder 92] では，不完全制約充足問題（不完全CSP）が提案されている．不完全CSPでは，もとの過制約なCSPの制約を許容できる範囲内で緩和し，その緩和して得られたCSPの解を求めることが要求される．

CSPの場合と同様に，マルチエージェントシステムにおける問題を分散CSPとして定式化した場合にも

過制約になることは十分に考えられる．

例えば分散解釈問題 [Lesser 83] では，個々のエージェントがセンサデータの一部を担当して可能な解釈を生成し，それらを交換しながら全体として無矛盾な解釈を求める．しかし，センサデータに含まれるノイズ等の原因によりエージェントが誤った解釈を生成すると，全体として無矛盾な解が存在しない場合があり得る．

また，分散資源配分問題を解く手法の 1 つであるマルチステージネゴシエーション [Conry 91] では，エージェントがゴール（変数）とゴールを達成するための複数の代替プラン（値域）をもち，かつ，プラン実行時に必要な資源の利用に関してエージェント間のプランが競合する（制約）という設定のもとで，全エージェントのゴールを達成するプランを求める．この場合，エージェントのゴールの設定や資源の状態によっては過制約な問題になる可能性がある．

分散 CSP を解く従来のアルゴリズム [Hirayama 95, 横尾 92, 横尾 96, Yokoo 98a, 横尾 98b] はすべての制約を満たす変数への値の割り当てを解として求める．しかし，過制約でそのような値がない場合には，[Hirayama 95, 横尾 92, 横尾 96, Yokoo 98a] のアルゴリズムは単にその事実を発見するのみであり，[横尾 98b] のアルゴリズムはそれさえも発見できずに停止しない．このような過制約な場合でも問題の解決につながる何らかの手掛かりを得たいというのは自然な要求だと思われるが，このような要求に対して従来の分散 CSP の枠組は十分に対応できていない．

過制約な分散 CSP に対する初期の研究として [横尾 95] では，制約の重要度を反映した階層 [Borning 92] を導入し，充足可能な最適な階層を求めるという問題が提案されている．それに対して本論文では，より一般的な問題として分散不完全制約充足問題（分散不完全 CSP）を提案する．また，分散不完全 CSP の特殊な場合として，分散最大制約充足問題（分散最大 CSP）を取り上げ，それを解くアルゴリズムである同期型分枝限定法，反復分散ブレイクアウト法を提案する．

次章以降の本論文の構成は次の通りである．**2 章**と**3 章**で分散 CSP と不完全 CSP の概要を説明する．また，**4 章**で分散不完全 CSP を定義し，次にその構成要素を特定して分散最大 CSP という部分クラスを導入する．**5 章**では分散最大 CSP を解く二つのアルゴリズムを与え，**6 章**でそれらのアルゴリズムを評価する．最後に **7 章**にまとめと今後の課題を示す．

2. 分散制約充足問題

CSP は，変数，変数の値域，変数間の制約からなる．変数の値域とは，変数が取り得る値の集合であり，一般に有限かつ離散的だと仮定する．また，変数間の制約とは，複数の変数に対して許される値の集合 —— 対応する値域の直積上の部分集合 —— である．CSP を解くとは，すべての制約を満たすような変数への値の割り当てを 1 組または全組求めることである．

一方，分散 CSP は，CSP における変数や制約が複数エージェントに分散された問題と見なすことができる．形式的には次のように記述される [横尾 92, Yokoo 98a]．

- エージェントの集合 $1, 2, \ldots, m$ が存在する．
- 各変数 x_j に対して，その属するエージェント i が定義される．なお，変数 x_j がエージェント i に属することを $\mathrm{belongs}(x_j, i)$ と表す．
- エージェントは全体の制約のうち一部のみを知っている．エージェント i が制約 C_l を知っていることを $\mathrm{known}(C_l, i)$ と表す．

一般には，エージェントは自分に属する変数に関する制約のみを知る．また，エージェントが知っている制約の中には，他のエージェントに属する変数を含むものがある．これをエージェント間制約とよぶ．逆に，自分に属している変数だけからなる制約をエージェント内制約とよぶ．

次の場合に分散 CSP が解けたという．すべてのエージェント i において，

- $\forall x_j \mathrm{belongs}(x_j, i)$ について，x_j の値が d_j に決定される．
- $\forall C_l \mathrm{known}(C_l, i)$ なる制約が，$x_1 = d_1, x_2 = d_2, \ldots, x_n = d_n$ のもとで真となる．

CSP の例としてよく用いられる n クイーン問題において，各クイーンに対応する独立なエージェントが存在し，それらのエージェントが自分のクイーンの位置を決定しようとしていると考えれば，この問題は分散 CSP として定式化される．

3. 不完全制約充足問題

不完全 CSP は，過制約な CSP を扱う一般的な枠組の一つであり，形式的には次のように記述される [Freuder 89, Freuder 92]．

$$\langle (P, U), (PS, \leq), (M, (N, S)) \rangle$$

ここで，Pは，過制約で解がないCSPである．また，各変数の潜在的な値の集合をユニバース（universe）というが，Uは，Pの全変数のユニバースの集合である．$(PS, \leq)$は問題空間とよばれ，PSは，Pを何らかの方法で緩和したCSPの集合（ただし，Pを含む）で，その要素間には半順序$\leq$が導入されている．また，問題空間上では，PSの要素であるCSPとPとの間の距離を測る距離関数Mが定義され，さらにその距離に関して必要値Nと充分値Sが定義される．

不完全CSPの解とは，PSの要素であるCSPのうち，Pとの距離が必要値N未満の過制約でない充足可能なCSPとその解のことである．また，Pとの距離がS以下の不完全CSPの解が見つかると、充分な解が得られたと見なして探索を終了することができる．なお，Pとの距離が最小となる解を不完全CSPの最適解といい，そのときの距離を最短距離という．

不完全CSPの重要な部分クラスの一つに最大制約充足問題（最大CSP）がある．最大CSPでは，Pからいくつかの制約を取り除いたCSPの集合が問題空間となり，取り除いた制約の数がPとの距離になる．最大CSPの最適解を求めることは，取り除く制約の数が最小である充足可能なCSPとその解を求めることだが，これは，Pの変数への値のうち，違反する制約の数が最小のものを求めることに相当する．

4. 分散不完全制約充足問題

4・1 問題の定義

分散不完全CSPの構成要素は次のとおりである．

- エージェントの集合：$1, 2, \ldots, m$
- 任意のエージェントiに対する不完全CSP:
  $$\langle (P_i, U_i), (PS_i, \leq), (M_i, (N_i, S_i)) \rangle$$
- 個々のエージェントの距離から，エージェント全体の距離（大域距離）を求める大域距離関数：
  $$G : \Re^m \rightarrow \Re$$

ここでエージェントiに対する不完全CSPのうち，(P_i, U_i)はそれぞれ，エージェントiに分散されているCSPとその全変数のユニバースの集合である．また，$(PS_i, \leq)$は問題空間であり，$(M_i, (N_i, S_i))$はそれぞれ，距離関数，P_iとの距離の必要値と充分値である．これらの詳細については**3章**で説明したとおりである．

エージェントiは，PS_iに含まれるCSPのうち，P_iとの距離がN_i未満のものを充足させようとする．すべてのエージェントがこれを達成したとき分散不完全CSPが解けたといい，そのときの全エージェントのCSPとそれらの解を分散不完全CSPの解という．また，すべてのエージェントが距離S_i以下のCSPとそれらの解を見つけると，分散不完全CSPの解としては充分であり探索を終了することができる．また，分散不完全CSPの解に対して，大域距離関数Gを用いて大域距離が計算できるが，その値が最小となる解を分散不完全CSPの最適解とよび，そのときの大域距離を最短大域距離とよぶ．

4・2 分散最大制約充足問題

本論文では，分散不完全CSPの構成要素の一部を次のように特定した分散最大CSPを導入する．

任意のエージェントiについて，

- P_iからいくつかの制約を取り除いたCSPがPS_iの要素となる．可能な取り除き方すべてについてPS_iの要素が対応する．
- PS_i内の問題P_i'とP_iの距離は，取り除いた制約の数である．
- 距離をd_iとおくと，大域距離関数Gは$\max_i d_i$である．

分散最大CSPで任意のエージェントiは，P_iからN_i個未満の制約を取り除いた充足可能なCSPとその解を求める．言い換えると，P_iの変数に対して，違反する制約の数がN_i個未満となるような値を求める．全エージェントがそのような値を求めたとき，分散最大CSPが解けたといい，そのときの全エージェントのCSPとそれらの解を分散最大CSPの解という．また，すべてのエージェントが制約違反数がS_i以下の値を見つけると，分散最大CSPの解としては充分であり探索を終了することができる．さらに，分散最大CSPの解のうち，$\max_i d_i$ —— 個々のエージェントの制約違反数のうち，全エージェント内での最大値 —— を最小にする解を最適解とよび，そのときの違反数の最大値を最短大域距離という．

図1の分散色塗り問題を用いて，分散最大CSPを具体的に説明する．この図でグラフの頂点は変数（エージェント），辺が変数間の制約を表す．目的は，互いに辺でつながっている頂点どうしが異なる色になるように，全体を黒と白の2色で塗り分けることである．エージェントは1つの頂点を担当するが，その頂点につながっている辺で表される制約のみを知っているとする．例えばエージェント1は制約$\{a, c, d\}$のみを知っている．この場合，エージェント1の問題P_1は，値域が{黒，白}である変数1と制約$\{a, c, d\}$である．変数への値の割り当てが図1の通りであるとき，エージェント1の制約違反数は1なので距離は1になる．他の

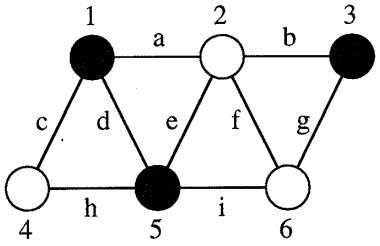


図1 分散色塗り問題

エージェントについては，エージェント2，エージェント5，エージェント6の距離が1，エージェント3とエージェント4の距離は0である．今，任意のエージェントiの距離の必要値N_iが2だとすると，図1の値の割り当てで分散最大CSPの解が得られている．しかし，1だとすると，エージェント3とエージェント4以外のエージェントの距離が1未満でないため，この値の割り当てでは解は得られていない．なお，この場合の大域距離は，個々のエージェントの距離の最大値ということで1になる．この例では，エージェントがどのように値を割り当てても大域距離を1未満にすることはできないので，この値の割り当てに対応する解が最適解であり，最短大域距離は1である．

次に，分散最大CSPを過制約な分散資源配分問題に適用する場合の概略を述べる．過制約な分散資源配分問題では，エージェントのプラン間の資源競合が避けられない．競合を伴うプランを実行するには競合解消などの特別な対処が必要になるため，個々のエージェントの目標はできるだけ競合の少ないプランを求めることであろう．では，複数のエージェントがいて，それぞれがこれを達成しようとする場合にどうするかが問題となる．この問題を分散最大CSPとして定式化した場合，全エージェント内での競合の数の最大値をできるだけ小さくするように各エージェントが自分のプランを選択することになる．すなわち，各エージェントはたくさんの競合を伴うプランをもつエージェントをできるだけなくそうとする．

5. 分散最大制約充足アルゴリズム

分散最大CSPの最適解を求めるアルゴリズムである同期型分枝限定法と反復分散ブレイクアウト法を提案する．本論文では，アルゴリズムの設計にあたって以下のことを仮定する．

まず，これらのアルゴリズムでは最適解を求めることを目指すため，任意のエージェントiについて充分値S_iが零であると仮定する．

次に，エージェント間の通信モデルとして以下のようなものを想定する．

- エージェントは，アドレスを知っているエージェントに対して，メッセージを直接送ることができる．
- メッセージには有限の通信遅れがあるが，その上限はわからない．
- 任意の2エージェント間では，メッセージは発信順に受信される．

これは，非同期通信システムのモデルとしてよく利用されており，十分妥当なものだと考える．

さらに，対象とする問題のクラスを次のように限定する．

- 1エージェントは唯一つの変数をもっている．
- すべての制約は2変数間で定義されている．このような制約を特にバイナリ（binary）制約という．

これは，アルゴリズム設計の単純化のために設けたものであり，このクラスの問題に対して設計したアルゴリズムをより広いクラスの問題に対応できるよう拡張することは可能である．具体的な拡張方法については[Yokoo 98c]を参照されたい．なお，このようなクラスの問題は変数（エージェント）を頂点，制約を辺とする制約グラフで表現できる．

5・1 同期型分枝限定法

同期型分枝限定法は，通常の分枝限定法を複数エージェントでシミュレートするアルゴリズムである．このアルゴリズムでは，前もって変数順序（エージェント順序）と値順序が固定され，パス —— 変数，変数の値，その変数をもつエージェントの制約違反数からなる三つ組のリスト —— をエージェント順序に従って順番に送り，徐々にパスを伸ばしていく．動作の概略は次の通りである．

- エージェント順序で1番目にあたるエージェントがアルゴリズムを起動し，自分がもつ変数の1番目の値とその制約違反数（この場合は0）をパスとして，次のエージェントに送る．
- エージェント順序が1つ前のエージェントからパスを受け取ると，それを保存し，そのパスと自分の変数の1番目の値を合わせて評価して，大域距離の下界値を見積もる．具体的には，パス内のそれぞれエージェントの制約違反数が自分の値によりいくらになるか計算するとともに，その値による自分の制約違反数を求め，それらの最大値を下界値とする．下界値が現時点での大域距離の上界値未満であれば，パス内の各エージェントの制約違反数を更新し，自分の変数の値と制約違反数を

```
procedure initiate /* done only by the first agent for starting the algorithm */
    d_i ← the first value in domain;
    n_i ← known upper bound; previous_path ← nil;
    send (token, [[x_i, d_i, 0]], n_i) to the next agent;

when i received (token, current_path, ub) from the previous agent do
    previous_path ← current_path; n_i ← ub;
    next ← get_next(domain);
    send_token; end do;

when i received (token, current_path, ub) from the next agent do
    [x_i, d_i, nv_i] ← the element related to x_i in current_path; n_i ← ub;
    next ← get_next(domain minus all elements up to d_i);
    send_token; end do;

procedur send_token
    if next ≠ nil then
        if i = the last agent then
            next_to_next ← next;
            while next_to_next ≠ nil do
                when max nv_j in new_path < n_i do
                    n_i ← max nv_j in new_path;
                    best_path ← new_path; end do;
                when n_i = 0 do
                    terminate the algorithm; end do;
                next_to_next ← get_next(domain minus all elements up to next_to_next);
            end do;
            send (token, previous_path, n_i) to the previous agent;
        else
            send (token, new_path, n_i) to the next agent; end if;
    else
        if i = the first agent then
            terminate the algorithm;
        else
            send (token, previous_path, n_i) to the previous agent; end if; end if;

procedure get_next(value_list)
    if value_list = nil then
        return nil;
    else
        d_i ← the first value in value_list; new_path ← nil; counter ← 0;
        if check(previous_path) then
            return d_i;
        else
            return get_next(value_list minus d_i); end if; end if;

procedure check(path)
    if path = nil then
        add [x_i, d_i, counter] to new_path;
        return true;
    else
        [x_j, d_j, nv_j] ← the first element in path;
        if [x_i, d_i] and [x_j, d_j] are not consistent then
            counter ← counter + 1;
            if counter ≥ n_i or nv_j + 1 ≥ n_i then
                return false;
            else
                add [x_j, d_j, nv_j + 1] to new_path;
                return check(path minus the first element); end if;
        else
            add [x_j, d_j, nv_j] to new_path;
            return check(path minus the first element); end if; end if;
```

図2　同期型分枝限定法

加えて新たなパスとし，次のエージェントに送る．下界値が上界値以上であれば次の値で同様の評価を行うが，これを繰り返して値が尽きれば前のエージェントにパスを送り返す．なお，大域距離の上界値とは過去に得られた大域距離のうちの最小値であり，この値もパスとともに送られる．

- エージェント順序が1つ後のエージェントからパスを受け取ると，パス中の自分の変数の値を次の値に変更して再評価する．下界値が現時点の上界値未満なら新しいパスを次のエージェントに送り，上界値以上ならさらに次の値で同様に評価を行うが，これを繰り返して値が尽きれば，過去に受け取って保存していたパスを前のエージェントに送る．

アルゴリズムの詳細を図2に示す．

同期型分枝限定法は，エージェント間で逐次的に分枝限定法の動作をシミュレートし，複数エージェントに分散している探索空間をしらみ潰し的に探索する．そのためアルゴリズムの完全性は保証される．しかし現状では，一つのパスを伸ばしていくだけであり，パスを受け取ったエージェント以外は何の動作も行わない

ため，複数エージェントによる並列実行の利点は生かされていない．また，各エージェントがエージェント順序という一種の大域的な情報をもつことが前提であるため，適用できる問題の範囲も限定される．

5・2 反復分散ブレイクアウト法

〔1〕 分散ブレイクアウト法

分散ブレイクアウト法 [横尾 98b] は分散 CSP を解くアルゴリズムで，その特徴は，複数のエージェントが互いの動作を相互排除しながら並列に山登り法 [Hirayama 95] を実行すること，エージェントが疑似局所最適解に陥ったときの手続きとしてブレイクアウト法 [Morris 93] を採用していることの 2 点である．

分散ブレイクアウト法では，エージェントは最初に自分の変数に任意の値を割り当て，その値を近傍 —— エージェント間制約により直接関連のあるエージェントの集合 —— に *ok*?メッセージを使って送る．以降，エージェントは次の手続きを繰り返す．

- 近傍の全エージェントから *ok*?メッセージを受け取ると，まず，自分がもつ変数の現在の値のコストを計算する．コストとは，その値のもとで違反している制約の重み和である．なお，制約の重みとは，制約ごとに定義された正整数で，その初期値は 1 である．次に，別の値に割り当てを変更した場合のコストの改善量の最大値（*improve* とよぶ）を計算し，その値を *improve* メッセージを使って近傍に送る．
- 近傍のエージェントから *improve* メッセージを受け取ると，その *improve* を自分のものと比較し，もしも相手の方が大きければ，自分の値は変更しない．また，*improve* が同じ場合は，両エージェントの識別子から優先順位を求め，自分の優先順位が低い場合にも値を変更しない．一方，近傍の全エージェントとの比較が終わった段階で上の条件にあてはまらない場合には値を変更する．最後に，値変更の有無にかかわらず，変数の現在の値を近傍に *ok*?メッセージを使って送る．

これを繰り返すだけで解が求められることもあるが，多くの場合，あるエージェントが疑似局所最適解に陥る．疑似局所最適解とは，変数の現在の値のコストが 0 より大きく，しかも，値をどう変更してもコストが下がらない状態をいう．分散ブレイクアウト法では，疑似局所最適解にいるエージェントがその時点で違反している制約の重みを上げ，コスト計算のための関数を更新する．

分散ブレイクアウト法では，個々のエージェントの違反制約のチェックは並列に行われる．また，あるエージェントが変数の値を変更する際，近傍の値変更は抑制されるが，それ以外のエージェントについては抑制されないので，全体としては並列に変数の値が変更される．そのため，解が存在する非常に困難な問題に対して，特に効率がよいことが示されている [横尾 98b]．また，解を発見したことを検出するアルゴリズムが埋め込まれていることも分散ブレイクアウト法の利点である．従来の分散制約充足アルゴリズムでは，その検出に分散スナップショット [Chandy 85] などの別の手続きを呼び出すことが前提にされていた．一方，分散ブレイクアウト法の欠点は完全性が保証されないことである．すなわち，解が存在したとしてもそれを見つけることができるとは限らないし，また，解がないことを発見できない．

〔2〕 反復分散ブレイクアウト法

反復分散ブレイクアウト法では，分散最大 CSP に対し，分散ブレイクアウト法を繰り返し適用する．基本的な動作は次の通りである．まず，任意のエージェント i の不完全 CSP の必要値 N_i に共通の値 ub を代入し，分散ブレイクアウト法を実行する．この分散ブレイクアウト法は，それぞれのエージェントの距離が N_i 未満になると終了し，そのことをあるエージェントが検出する．検出したエージェントは次の N_i の値を現在の大域距離に設定するように他のエージェントにメッセージを送信する．これを受け取ったエージェントは再び分散ブレイクアウト法を実行し，以下同様に繰り返す．この繰り返しは，大域距離が零となる変数への値の割り当てが見つかると終了する．

反復分散ブレイクアウト法では，分散ブレイクアウト法と同じ *ok*?メッセージと *improve* メッセージが用いられ，また，それぞれを受け取ったときに実行される手続きも分散ブレイクアウト法の手続きの一部を変更したものに等しい．そのため，以下では主な変更部分のみを述べる．

- 実行前に，各エージェントの必要値に代入する共通の値 ub の初期値を決める．ub は，*improve* メッセージを使って送信される．
- 近傍の全エージェントから *ok*?メッセージを受け取ったとき，エージェント i は，違反している制約の数が必要値 N_i 未満ならば，現在の値のコストを零にする．N_i 以上ならば分散ブレイクアウト法と同様に違反制約の重み和としてコストを計算する．*improve* を求める場合の他の値のコスト計算も同様に行う．
- 分散ブレイクアウト法では，あるエージェントの

```
procedure initiate /* done by every agent for starting the algorithm */
    current_value ← the value randomly chosen from domain;
    n_i ← known upper bound; my_termination_counter ← 0; counter ← 0;
    send (ok?, x_i, current_value) to neighbors;
    goto wait_ok? mode;

wait_ok? mode
when i received (ok?, x_j, d_j) do
    counter ← counter + 1; add (x_j, d_j) to agent_view;
    if counter = # of neighbors then
        send_improve; counter ← 0;
        goto wait_improve mode;
    else
        goto wait_ok? mode; end if; end do;

procedure send_improve
    if # of currently violated constraints < n_i then
        current_eval ← 0;
    else
        current_eval ← weighted sum of violated constraints; end if;
    my_improve ← possible maximal improvement;
    new_value ← the value which gives the maximal improvement;
    if current_eval = 0 then
        consistent ← true;
    else
        consistent ← false; my_termination_counter ← 0; end if;
    if my_improve > 0 then
        can_move ← true; quasi_local_minimum ← false;
    else
        can_move ← false; quasi_local_minimum ← true; end if;
    send (improve, x_i, my_improve, current_eval, my_termination_counter, n_i) to neighbors;

wait_improve mode
when i received (improve, x_j, improve, eval, termination_counter, ub) do
    counter ← counter + 1;
    my_termination_counter ← min(termination_counter, my_termination_counter);
    when n_i ≠ ub do
        n_i ← min(n_i, ub); consistent ← false; end do;
    when improve > my_improve do
        can_move ← false; quasi_local_minimum ← false; end do;
    when improve = my_improve and x_j precedes x_i do
        can_move ← false; end do;
    when eval > 0 do
        consistent ← false; end do;
    if counter = # of neighbors then
        send_ok; counter ← 0; clear agent_view;
        goto wait_ok? mode;
    else
        goto wait_improve mode; end if; end do;

procedure send_ok
    when consistent = true do
        my_termination_counter ← my_termination_counter + 1;
        when my_termination_counter = diameter of the constraint graph do
            n_i ← current global distance;
            if n_i = 0 then
                notify neighbors that a solution has been found; terminate the algorithm;
            else
                my_termination_counter ← 0; end if; end do; end do;
    when quasi_local_minimum = true do
        increase the weights of violated constraints; end do;
    when can_move = true do
        current_value ← new_value; end do;
    send (ok?, x_i, current_value) to neighbors;
```

図3　反復分散ブレイクアウト法

$termination_counter$という内部変数の値が，問題に対応する制約グラフの直径以上になるとき，分散CSPの解が得られていることが保証されている．反復分散ブレイクアウト法では，同じ条件のとき，すべてのエージェントの不完全CSPの解が$\forall i(N_i = ub)$のもとで得られていることが保証される（理由は後述）．このことを検出したエージェントiは，N_iに現在の大域距離を代入し，その値を新たなubとして$improve$メッセージを使って他のエージェントに送る．

- 分散ブレイクアウト法では，あるエージェントは，自分の制約違反数が零で，近傍の全エージェントの制約違反数も零のとき，かつ，そのときに限り$consistent$という内部変数を真にする．一方，反復分散ブレイクアウト法では，自分の制約違反数がub未満で，近傍の全エージェントの制約違反数も同じub未満のとき，かつ，そのときに限り$consistent$を真にする．

表1 最適解を求めるまでのサイクル数の中央値（25例／クラス）

problem class	median cycle	mean minimal global distance
⟨10, 10, 18/45, 0.8⟩	3500	1.0
⟨10, 10, 18/45, 0.9⟩	18262	2.0
⟨10, 10, 27/45, 0.8⟩	46247	2.4
⟨10, 10, 27/45, 0.9⟩	499841	3.4
⟨10, 10, 36/45, 0.8⟩	336416	3.6
⟨10, 10, 36/45, 0.9⟩	1985700	5.0
⟨10, 10, 45/45, 0.8⟩	3435984	4.9
⟨10, 10, 45/45, 0.9⟩	21834077	6.0

アルゴリズムの詳細を図3に示す．

ある ub について $\forall i(N_i = ub)$ で，*termination_counter*がグラフの直径以上になるとき，すべてのエージェントの不完全CSPの解が $\forall i(N_i = ub)$ のもとで得られていることは次の事実から帰納的に証明できる．すなわち，必要値が $N_i = ub$ であるエージェント i が，*termination_counter*の値を d から $d+1$ に上げるのは，次のことが同時に成り立つとき，かつそのときのみである．エージェント i の近傍のすべてのエージェントの 1) 必要値が ub である；2) 制約違反数が ub 未満である；3) *termination_counter*の値が d 以上である．

同期型分枝限定法が，変数への値の割り当てや制約違反のチェックを逐次的に行うのに対し，反復分散ブレイクアウト法は，それらを並列に実行するという利点をもつ．しかし，反復分散ブレイクアウト法の欠点の一つは，アルゴリズムの完全性が保証されないことである．また，もう一つの欠点は，分散最大CSPの最適解を得たとしても，それを最適解だと判定できないことである．なぜなら，ある ub が実際に最短大域距離だった場合，反復分散ブレイクアウト法は，$\forall i(N_i = ub+1)$ の解を発見することはできても，$\forall i(N_i = ub)$ で解がないことを発見できないからである．

6. 評　　価

同期型分枝限定法と反復分散ブレイクアウト法の性能を実験により評価する．

この実験で対象とする問題クラスは，$\langle n, m, p_1, p_2\rangle$ という4つのパラメータで表されるランダムバイナリ分散CSPである．これは，4つのパラメータの値に応じて生成されるランダムバイナリCSPの変数と制約を複数エージェントに分散し，各エージェントは1つの変数をもち，自分の変数に関する制約のみをもつように設定した分散CSPである．ここで，n は変数の数，m は各変数の値域のサイズである．また，p_1 は，n 個の変数に対して定義可能なバイナリ制約 $n(n-1)/2$ 個に対する実際の制約の数の割合である．そして，p_2 は，2変数間の可能な値のペア m^2 個に対して各制約が禁止する値のペアの割合を表す．特定の $\langle n, m, p_1, p_2\rangle$ に対して実際の問題例をつくる際，$n(n-1)p_1/2$ 個の変数のペアをランダムに選択し，各変数ペアに対して，ランダムに選んだ m^2p_2 個の値のペアが禁止されるように制約を定義する．

評価に使用したのは，ランダムバイナリ分散CSPのうち $n = 10$，$m = 10$，$p_1 \in \{18/45, 27/45, 36/45, 45/45\}$，$p_2 \in \{0.8, 0.9\}$ というクラスの問題である．これらのクラスのランダムバイナリCSPは最大CSPの問題として特に困難な部類に入る [Larrosa 96]．なお，最大CSPと分散最大CSPでは問題自体が異なるため，分散最大CSP用のアルゴリズムを評価するのにこれらのクラスの問題が適当かどうかは不明である．しかし，初期の評価用の問題としては妥当であると考える．

実験は，複数エージェントの並行動作をシミュレートする離散事象シミュレータを作成し，その上で行う．このシミュレータでは，マネージャという仮想的なエージェントを想定する．マネージャは離散的な時を刻む時計をもち，エージェント間のメッセージの配送を行う．全体の動作は，次の一連の処理を1サイクルとした繰り返しである．1) エージェントから他のエージェントに向けて送信されたメッセージをマネージャがすべて集める；2) マネージャが時計の時刻を1単位すすめる；3) マネージャが集めたメッセージを宛先となるエージェントに届ける；4) 個々のエージェントが，受け取ったメッセージに対して処理をおこない，他のエージェントにメッセージを送信する．このシミュレータ上で先の2つのアルゴリズムを実現し，サイクル数（マネージャの時計が刻んだ時間）をコストとして評価する．

6・1 最適解発見のコスト

まず，最適解を得るコストを評価する．

同期型分枝限定法では最適解を得ることが保証されているため，それを得るまでのサイクル数を求めること

ができる．実験では，ランダムバイナリ分散CSPの各問題クラスに対して25例をランダムに生成し，それぞれに同期型分枝限定法を適用した．なお，同期型分枝限定法のエージェント順序は，conjunctive widthヒューリスティックスのwidth/domain-sizeの場合[Wallace 93]に従って決定し，値順序は辞書式順序とした．また，解く例に対応する制約グラフの最大次数を大域距離の上界値の初期値とした．なぜなら，大域距離は制約グラフの最大次数より大きくなることがないからである．また，充分値S_iは前述の通り零とした．表1は，各問題クラスごとの，最適解を得るまでにかかったサイクル数の中央値，および，最短大域距離の平均値を示している．同期型分枝限定法で最適解を得るには全般にコストがかかり，特に最短大域距離が大きい問題に対しては，莫大なコストがかかっている様子が窺える．

一方，反復分散ブレイクアウト法は完全ではないため，最適解に達しない場合があり，同期型分枝限定法と同様には評価できない．しかし，どれくらい最適解への到達に成功するかを調べることは可能である．今回の実験では，$p_1 = 27/45$，$p_2 = 0.8$の25例のそれぞれにつき，初期値を変えて10回ずつ合計250回にわたって反復分散ブレイクアウト法を実行してみた．なお，ubの初期値は制約グラフの最大次数とした．また，ubの値は，初期値から始めて各値での解が得られるごとに1ずつ減少するように設定した．結果は，同じ例に対して同期型分枝限定法が最適解を発見したサイクル数までに反復分散ブレイクアウト法が最適解に達することができたのは250回中30回のみであった．

6・2 Anytime Curves

次に，同期型分枝限定法と反復分散ブレイクアウト法の最適解に至るまでの動作を評価するためにAnytime Curvesを求める．Anytime Curvesとは，アルゴリズムをスタートさせて以後，時間経過と共に中間結果がどのように改善されるかをグラフ化したものである．ここでは，x軸にアルゴリズムをスタートさせてからのサイクル数，y軸に大域距離をとり，各アルゴリズムごとに，あるサイクル数までに得られた解のうち，もっとも良い解の大域距離をプロットする．図4の実線は，$n = m = 10$，$p_1 = 27/45$，$p_2 = 0.9$の1つの例に対する同期型分枝限定法のAnytime Curveを示している．このクラスでは，この例に対して，同期型分枝限定法はもっとも短いサイクル数で最適解を見つけている．また，破線は，同じ例に対する反復分散ブレイクアウト法のAnytime Curveである．反復分散ブレイクアウト法では，初期値を変えて10回試行し，その平均値をプロットしている．

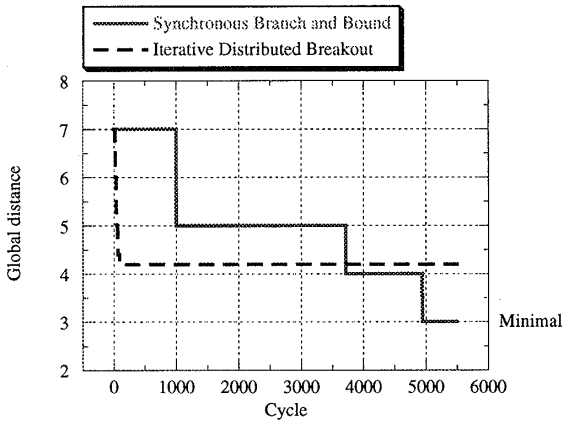


図4 $\langle 10, 10, 27/45, 0.9\rangle$のある問題例に対するAnytime curve

図4から明らかなように，同期型分枝限定法のAnytime Curveは，最終的には最短大域距離まで減少していくものの，減少の仕方は緩やかである．一方，反復分散ブレイクアウト法の場合は，非常に早い段階で急減し，以降，最短大域距離の近く（準最短大域距離）でほぼ一定となり，それ以下には下がらない．これと同様のことは，この例のみならず，他のクラスの他の例についても観察された．また，各クラスの問題について，先ほどと同様に例を1つ決めて，その例に対して同期型分枝限定法と反復分散ブレイクアウト法のそれぞれが準最短大域距離に達するまでのサイクル数を比べてみると，すべての場合において，反復分散ブレイクアウト法の方が速くそれに達することが確認された．

この結果は次のように説明できる．今回使用した反復分散ブレイクアウト法では，全エージェント共通の必要値としてubを決め，各エージェントが自分の距離がub未満になるように分散ブレイクアウト法を実行し，全エージェントがub未満になると，次は$ub - 1$にして...と繰り返すわけだが，ubの値が大きいときには，分散ブレイクアウト法がほとんど疑似局所最適解に出会うことなく終了し，すぐに次の回をスタートできる．そのため，ubの値が大きい反復分散ブレイクアウト法の初期段階には大域距離が急速に減少すると考えられる．一方，同期型分枝限定法では，初期段階では良い上界値が得られていないため有効な枝刈りができず，広い範囲で探索を行うことになる．そのため，なかなか大域距離が減少しないと考えられる．

以上の結果をまとめると，高いコストを払っても最適解が必要な場合は，同期型分枝限定法が適しているが，準最適解でもいいからコストを抑えたい場合には，反復分散ブレイクアウト法が適しているといえる．

7. まとめと今後の課題

本論文では，過制約な分散 CSP を扱う一般的な枠組である分散不完全 CSP を提案した．また，過制約な分散 CSP への具体的な対処法として，分散不完全 CSP の部分クラスにあたる分散最大 CSP を取り上げ，それを解くアルゴリズムである同期型分枝限定法と反復分散ブレイクアウト法を提案した．今後の課題は次の通りである．

実際の問題においては，反復分散ブレイクアウト法で十分な精度の準最適解が得られる保証はない．このような反復分散ブレイクアウト法で得られる解の精度に関する議論，および精度の改善方法については今後の検討課題である．

分散不完全 CSP には，今回提案した分散最大 CSP 以外にもいくつかの部分クラスが存在する．その中でも，分散最大 CSP の大域距離関数 G を $\sum_i d_i$ に置き換えた問題，すなわち全エージェントの制約違反数の総和を最小化する問題には多くの応用が期待される．例えば，**1** 章で述べた過制約な分散解釈問題では，全体として矛盾の少ない解釈を求めることが妥当であると考えられるが，この問題はそのような応用の一つといえる．このように，分散不完全 CSP の他の部分クラスについても応用問題や複雑度等を検討し，アルゴリズムを設計することが今後の課題である．

◇ 参 考 文 献 ◇

[Borning 92] Borning, A., Freeman-Benson, B. and Wilson, M.: Constraint Hierarchies, *Lisp and Symbolic Computation*, Vol.5, pp. 223–270 (1992).

[Chandy 85] Chandy, K. and Lamport, L.: Distributed Snapshots: Determining Global States of Distributed Systems, *ACM Trans. on Computer Systems*, Vol.3, No.1, pp. 63–75 (1985).

[Conry 91] Conry, S. E., Kuwabara, K., Lesser, V. R. and Meyer, R. A.: Multistage Negotiation for Distributed Constraint Satisfaction, *IEEE Trans. on Systems, Man and Cybernetics*, Vol.21, No.6, pp. 1462–1477 (1991).

[Freuder 89] Freuder, E. C.: Partial Constraint Satisfaction, *IJCAI-89*, pp. 278–283 (1989).

[Freuder 92] Freuder, E. C. and Wallace, R. J.: Partial Constraint Satisfaction, *Artificial Intelligence*, Vol.58, No.1–3, pp. 21–70 (1992).

[Hirayama 95] Hirayama, K. and Toyoda, J.: Forming Coalitions for Breaking Deadlocks, *ICMAS-95*, pp. 155–162 (1995).

[Huhns 91] Huhns, M. N. and Bridgeland, D. M.: Multiagent Truth Maintenance, *IEEE Trans. on Systems, Man and Cybernetics*, Vol.21, No.6, pp. 1437–1445 (1991).

[Larrosa 96] Larrosa, J. and Meseguer, P.: Phase Transition in MAX-CSP, *ECAI-96*, pp. 190–194 (1996).

[Lesser 83] Lesser, V. R. and Corkill, D. D.: The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks, *AI Magazine*, Vol.4, No.3, pp. 15–33 (1983).

[Morris 93] Morris, P.: The Breakout Method for Escaping from Local Minima, *AAAI-93*, pp. 40–45 (1993).

[Sycara 91] Sycara, K. P., Roth, S., Sadeh, N. and Fox, M.: Distributed Constrained Heuristic Search, *IEEE Trans. on Systems, Man and Cybernetics*, Vol.21, No.6, pp. 1446–1461 (1991).

[Wallace 93] Wallace, R. J. and Freuder, E. C.: Conjunctive Width Heuristics for Maximal Constraint Satisfaction, *AAAI-93*, pp. 762–768 (1993).

[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，信学論，Vol.J-75 D-I, No.8, pp. 704–713 (1992).

[横尾 95] 横尾真：分散制約充足問題における制約緩和，情処学論，Vol.36, No.2, pp. 275–282 (1995).

[横尾 96] 横尾真：柔軟で動的なエージェントの組織構造を用いた分散制約充足アルゴリズム，人工知能学会誌，Vol.11, No.6, pp. 933–940 (1996).

[横尾 97] 横尾真，平山勝敏：CSP の新しい展開：分散／動的／不完全 CSP，人工知能学会誌，Vol.12, No.3, pp. 381–389 (1997).

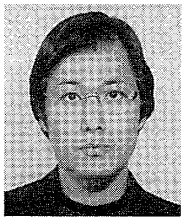
[Yokoo 98a] Yokoo, M., Durfee, E. H., Ishida, T. and Kuwabara, K.: The Distributed Constraint Satisfaction Problem: Formalization and Algorithms, *IEEE Trans. on Knowledge and Data Engineering*, Vol.10, No.5, pp. 673–685 (1998).

[横尾 98b] 横尾真，平山勝敏：分散 breakout：反復改善型分散制約充足アルゴリズム，情処学論，Vol.31, No.1, pp. 106–114 (1998).

[Yokoo 98c] Yokoo, M. and Hirayama, K.: Distributed Constraint Satisfaction Algorithm for Complex Local Problems, *ICMAS-98*, pp. 372–379 (1998).

〔担当委員：國藤　進〕

著 者 紹 介



平山　勝敏(正会員)

1967 年生まれ．1990 年大阪大学基礎工学部制御工学科卒業．1992 年同大学院基礎工学研究科博士前期課程修了．1995 年同博士後期課程修了．神戸商船大学商船学部助手を経て，1997 年神戸商船大学商船学部講師．制約充足，マルチエージェントシステムに関する研究に従事．博士 (工学)．情報処理学会，日本ソフトウェア科学会，AAAI 各会員．〈hirayama@ti.kshosen.ac.jp〉

横尾　真(正会員)

1962 年生まれ．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．現在 NTT コミュニケーション科学基礎研究所に勤務．制約充足問題，マルチエージェントシステム，分散人工知能に関する研究に従事．制約充足／分散制約充足，エージェントの合意形成メカニズム等に興味を持つ．博士 (工学)．1992 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受賞．情報処理学会，日本ソフトウェア科学会，AAAI 各会員．

〈yokoo@cslab.kecl.ntt.co.jp〉