分散 breakout：反復改善型分散制約充足アルゴリズム

横尾 真[†] 平山勝敏[††]

本論文では分散制約充足問題を解くための新しいアルゴリズムである分散 breakout アルゴリズムを提案する．本アルゴリズムは集中型の制約充足問題を解くための反復改善型のアルゴリズムである breakout アルゴリズムに触発されたものであり，各エージェントは近傍のエージェントと現在の変数の値の割当，可能な改善方法を交換し，エージェント全体として制約条件違反の個数が減少するように値の変更を行う．また，エージェント全体として局所最適に陥ったことを検出するのではなく，近傍との通信のみで検出できる，より弱い条件である準局所最適を検出し，制約の重みを変更することにより準局所最適から脱出する．実験的な評価により，本アルゴリズムはグラフの色塗り問題の非常に難しい問題のインスタンスに関して既存のアルゴリズムより効率的であることを示す．

Distributed Breakout: Iterative Improvement Algorithm for Solving Distributed Constraint Satisfaction Problem

MAKOTO YOKOO[†] and KATSUTOSHI HIRAYAMA[††]

This paper presents a new algorithm for solving distributed constraint satisfaction problems (distributed CSPs) called the *distributed breakout* algorithm, which is inspired by the breakout algorithm for solving centralized CSPs. In this algorithm, each agent tries to optimize its evaluation value (the number of constraint violations) by exchanging its current value and the possible amount of its improvement among neighboring agents. Instead of detecting the fact that agents as a whole are trapped in a local-minimum, each agent detects whether it is in a quasi-local-minimum, which is a weaker condition than a local-minimum, and changes the weights of constraint violations to escape from the quasi-local-minimum. Experimental evaluations show this algorithm to be much more efficient than existing algorithms for critically difficult (phase transition) problem instances of distributed graph-coloring problems.

1. まえがき

分散人工知能とは，複数の自律的に動作するエージェントの相互作用，特にこれらのエージェント間の協調に関する人工知能の研究の1分野である．近年の計算機の小型化，低価格化，および計算機ネットワーク技術の進歩により，分散計算機環境が急速に普及しつつあり，このような自律的なエージェントに関する研究の必要性は非常に大きくなっている．このため，分散人工知能は人工知能の中でも非常に活発な分野となっている．

一方，制約充足問題[8]とは，有限で離散的な領域から値をとる複数の変数に対して，制約を満足する値の割当を発見する問題であり，人工知能の様々な問題がこの問題により定式化できることが知られている．著者らは文献14)，19)において，制約充足問題の変数，制約が複数のエージェントに分散された問題として分散制約充足問題を定義し，分散人工知能の様々な応用問題（分散資源割当問題[3]，分散スケジューリング問題[13]，マルチエージェントによるデータの真偽値管理[7]等）が分散制約充足問題として定式化可能であることを示した．このように，分散人工知能，マルチエージェントシステムで生じる様々な問題が分散制約充足問題として定式化可能であるため，分散制約充足問題を解くための分散アルゴリズムは，エージェント間の協調を実現するための重要なインフラストラクチャであると考えることができる．

著者らは分散制約充足問題を解く基本的なアルゴリズムとして，各エージェントが局所的な知識に基づいて，非同期，並行に解を探索する非同期バックトラッキングアルゴリズム[14),19)]を，さらに，非同期バックトラッキングアルゴリズムを拡張し高速化した非同期弱コミットメント探索アルゴリズムを提案している[17),18)]．また，著者らはエージェントが提携を結ぶ

† NTT コミュニケーション科学研究所
NTT Communication Science Laboratories
†† 神戸商船大学
Kobe University of Mercantile Marine

ことにより局所最適から脱出することを可能とする，分散制約充足問題を解く反復改善型の探索アルゴリズムを提案している[6]．

本論文では，分散制約充足問題を解く新しい反復改善型のアルゴリズムである分散 breakout アルゴリズムを提案する．本アルゴリズムは集中型の制約充足問題を解く反復改善型の探索アルゴリズムの一種である breakout アルゴリズムに触発されたものであり，以下のような特徴を持つ．

- エージェント全体として局所最適に陥ったことを検出するのではなく，より弱い条件である準局所最適を検出し，制約の重みの変更を行う．

分散グラフの色塗り問題に関する実験結果により，分散 breakout アルゴリズムは既存のアルゴリズム（非同期弱コミットメント探索アルゴリズム）と比較して，制約の疎な問題，制約の密な問題に関しては性能向上は得られないが，制約の密度が中間的な非常に難しい問題のインスタンスに関して，15 倍程度の高速化が得られることが示される．

本論文では以下，分散制約充足問題の定義について簡単に説明し（2 章），分散 breakout アルゴリズムについて解説する（3 章）．さらに，例題による実験結果により本アルゴリズムの有効性を示し（4 章），本アルゴリズムおよび既存の分散制約充足アルゴリズムの性質等について考察する（5 章）．

2. 分散制約充足問題

制約充足問題（CSP）は一般に次のように定義される．n 個の変数 $x_1, x_2, \ldots, x_n$ と，変数のそれぞれが値をとる有限で離散的な領域 $D_1, D_2, \ldots, D_n$，および制約の集合が存在する．本論文では制約は述語によって内包的に定義されるとする．すなわち，制約 $p_k(x_{k1}, \ldots, x_{kj})$ は，直積 $D_{k1} \times \ldots \times D_{kj}$ に対して定義され，これらの変数の値が互いに整合のとれている場合に真となる．制約充足問題の解を求めることは，すべての制約を満足する変数の値の組を求めることである．制約充足問題は一般に NP 完全であり，最悪の場合の計算量は変数の個数に対して指数的となることが予想される．しかしながら，探索を効率化し平均的な計算時間を小さくするための様々なヒューリスティックス，探索手法が提案されており，大規模で解が少数しか存在しない難しい問題においても，現実的な時間内で解を得ることが可能となっている[9),11),15),16)]．

分散制約充足問題とは，制約充足問題の変数が複数のエージェントに分散された問題である．本論文では以下のエージェント間通信のモデルを仮定する．

- エージェント間通信はメッセージ通信によってなされる．
- エージェントは，他のエージェントのアドレスを知っている場合に限りそのエージェントにメッセージを送信できる．
- メッセージの遅延は有限であるが，遅延時間の上限は分かっていない．
- 任意の 2 つのエージェントの組合せに関して，送信されたメッセージの順序は保存される．

各エージェントは自分の持つ変数の値を決定しようとするが，異なるエージェントの持つ変数間に制約がある．エージェントの目的は，エージェント間の制約をすべて満足する値の割当を見つけることである．形式的には，エージェントの集合 $\{1, 2, \ldots, m\}$ が存在し，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（$\mathrm{belongs}(x_j, i)$ と書く）．制約に関する情報も同様にエージェント間に分散される．エージェント i が制約 p_k を知っていることを $\mathrm{known}(p_k, i)$ と書く．

次の場合に，分散制約充足問題が解けたという．

- 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．そして，すべてのエージェント k について，$\forall p_l$ $\mathrm{known}(p_l, k)$ なる制約が，$x_1 = d_1, x_2 = d_2, \ldots, x_n = d_n$ の下で満足される．

以下，アルゴリズムの説明を行う際に，簡単のため次の仮定をおく．これらの仮定を緩和し，アルゴリズムを一般の場合に拡張することは容易である．

- 各エージェントの持つ変数は唯一である．
- 各エージェントは自分に属する変数が関係する制約をすべて知っている．
- エージェント間の制約はすべて二項関係（binary）である．

すべての制約が binary である分散制約充足問題はネットワークを用いて表現できる．すなわち，変数がネットワークのノードに対応し，ノード間のリンクが制約に対応する．また，各エージェントは唯一の変数を持つと仮定しているため，各ノードはエージェントに対応すると考えることもできる．以下，エージェント i と変数 x_i に関して共通の識別子 x_i を用いる．すべてのエージェントおよび変数にはユニークな識別子が与えられていると仮定する．

各エージェントに対して，リンクで接続されたエージェントの集合をそのエージェントの近傍と呼ぶ．また，2 つのエージェント間の距離を，2 つのエージェントを結ぶ最短の経路に含まれるリンクの個数として定

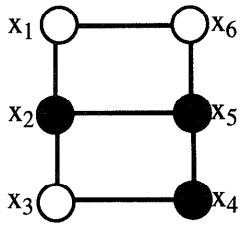

図1 制約ネットワークの例
Fig. 1 Example of a constraint network.

義する．たとえば，**図1**は，分散グラフの色塗り問題の例であり，各ノードに対応するエージェントが，リンクで結ばれたエージェントと異なる色になるように自分の色を決定しようとしている．この例では，エージェントのとりうる色は白と黒の2種類である．図1では，$x_1, x_2, \ldots, x_6$ の6つのエージェントが存在し，x_1 の近傍は $\{x_2, x_6\}$ であり，x_1 と x_4 との距離は3である．

3. 分散 breakout アルゴリズム

本章ではまず，通常の制約充足問題を解く breakout アルゴリズムについて説明し，さらに，同様なアルゴリズムを分散制約充足問題において実現する方法を示す．

3.1 breakout アルゴリズム

breakout アルゴリズム[10]は，反復改善型アルゴリズム[9),11),12)]の一種であり，すべての変数から構成される，いくつかの制約を満足しない不完全な解を構成し，その不完全な解を局所的な変更により改善していくことにより完全な解を得る．個々の制約に関して重みが定義され（初期値は1），違反している制約の重みの和を不完全な解の評価値とする（初期状態では重みが1であるため，この評価値は違反している制約の個数に等しい）．breakout アルゴリズムは，個々の変数の値を評価値が減少するように（制約条件違反の個数が減少するように）1つ1つ変更していく．このように制約条件違反の個数を減少させる方向に値を変更するという方針は制約違反最少化ヒューリスティック[9)]と呼ばれる．どの変数の値を変更しても全体として評価値が減少しない場合，その状態は局所最適であるという．局所最適に陥った場合，breakout アルゴリズムでは，その状態で違反している制約に関して重みを1増加させる．このように評価関数を変更することにより，近傍の状態が現在の状態よりも評価値が良くなり，局所最適から脱出することが可能となる．breakout アルゴリズムは非常に単純であるが，他の反復改善型アルゴリズム[9),11)]と比較して効率が優れていることが文献10)に示されている．

3.2 基本的なアイデア

前節で示した breakout アルゴリズムを分散制約充足問題に適用するにあたって，以下の問題点が存在する．

- ある時点で唯一のエージェントのみが値を変更できるようにすると処理の並列性が生かせない．一方，制約で関連する（近傍の）複数のエージェントが同時に値を変更すると評価値が改善されない可能性がある．
- エージェント全体として局所最適に陥っていることを判定するためにはエージェント全体での情報交換が必要となる．

これらの問題点を解決するため以下のような方法をとる．

- 近傍のエージェント間で評価値の可能な改善量を交換し合い，最も改善量の大きいエージェントのみに対して値の変更を許す．
- 局所最適を検出するのではなく，近傍との通信のみで検出できる，より弱い条件である準局所最適を検出し，重みの変更を行う．

本アルゴリズムでは，近傍のエージェント間で以下の2種類のメッセージが通信される．

(i) (ok?, 変数名，値)
(ii) (improve, 変数名，改善量，評価値，終了判定カウンタ値)

ok?メッセージは現在の値の割当を通知するメッセージであり，improve メッセージは可能な改善量を通知するメッセージである．近傍のエージェントで improve メッセージが交換され，最も改善量が大きいエージェントのみが値を変更することにより，制約で関連する複数のエージェントが同時に値を変更することを避け，かつ，制約で直接関連しないエージェントが並行して値に変更することを可能にする．

また，エージェント x_i から見て，現在の値の割当が準局所最適であることを以下のように定義する．

- エージェント x_i の持つ変数が満足しない制約が存在し，x_i および x_i の近傍のすべてのエージェントで可能な改善値が0である．

明らかに，エージェント全体として局所最適であれば，制約を満足しない任意のエージェントから見て，この状態は準局所最適であるが，逆は成り立たない．たとえば図1の状態は，エージェント x_1 から見て準局所最適であるが，エージェント全体としては局所最適ではない（x_5 が値を変更することにより改善が可能である）．実際には局所最適でない準局所最適の状態で

重みを変更することは，評価関数を必要以上に歪めることになり，解への収束に悪影響をもたらす可能性がある．準局所最適で重みを変更することの影響に関しては4章で評価を行う．

これらの方法は，基本的には著者らの以前の反復改善型の探索アルゴリズムに関する研究[6]で用いられているものと同等であるが，本アルゴリズムでは文献6)のように準局所最適でエージェント間で提携を構成するのではなく，各エージェントが独立に管理している，制約に関する重みを変更する．このためエージェント間での提携の構成のための合意形成が不要であり，アルゴリズムはより簡単になっている．

3.3 アルゴリズムの詳細

本アルゴリズムでは，各エージェントは初期状態ではランダムに値を決定し，最初のok?メッセージを送信する．近傍のすべてのエージェントからok?メッセージを受け取ったら現在の評価値，可能な改善量を計算してimproveメッセージを送信する．

図2 に2種類のメッセージ（ok?, improve）を受けたときのエージェント x_i で起動される手続きを示す．エージェントはok?メッセージを待つモード（**wait_ok? mode**, 図2(i)）とimprove?メッセージを待つモード（**wait_improve mode**, 図2(ii)）を繰り返す．

wait_ok? mode では，x_i は受け取った他のエージェントの変数の値の割当をエージェント/変数名と値のペア（e.g., (x_j, d_j)）で表現し，*agent_view* に格納する．近傍のすべてのエージェントからok?メッセージを受けると，現在の評価値，評価値の可能な改善量を計算し，improveメッセージを送信して **wait_improve mode** に移行する．

アルゴリズム中で用いられている状態変数の意味は以下のとおりである．

can_move: そのエージェントが値を変更する権利を持つかどうかを示すものあり，近傍の他のエージェントの方が改善量が大きい場合，もしくは改善量が等しく，他のエージェントの方が名前のアルファベット順で先行する場合に *false* に書き換えられる．

quasi_local_minimum: 現在の状態が x_i から見て準局所最適であるかを判定するものであり，他のエージェントの改善量が正であれば *false* に書き換えられる．

my_termination_counter: アルゴリズムの終了判定に用いるものであり，エージェントからの距離が *my_termination_counter* 以内のエージェン

```
wait_ok? mode — (i)
when received (ok?, x_j, d_j) do
  counter ← counter + 1;
  add (x_j, d_j) to agent_view;
  when counter = number_of_neighbors do
      send_improve; counter ← 0;
      goto wait_improve mode; end do;
  goto wait_ok mode; end do;

procedure send_improve
  current_eval ← evaluation value of current_value;
  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) to neighbors;

wait_improve? mode — (ii)
when received (improve, x_j, improve,
                    eval, termination_counter) do
  counter ← counter + 1;
  my_termination_counter ←
    min(termination_counter, my_termination_counter)
  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;
  when counter = number_of_neighbors do
    send_ok; counter ← 0; clear agent_view;
    goto wait_ok mode; end do;
  goto wait_improve mode; end do;

procedure send_ok
  when consistent = true do
    my_termination_counter ←
        my_termination_counter +1;
    when my_termination_counter = max_distance do
      notify neighbors that a solution has been found;
          terminate the algorithm;
    end do; end do;
  when quasi_local_minimum = true do
    increase the weights of constraint violations;
  end do;
  when can_move = true do
    current_value ← new_value; end do;
  send (ok?, x_i, current_value) to neighbors;
```

図2 メッセージに対する処理
Fig. 2 Procedures for receiving messages.

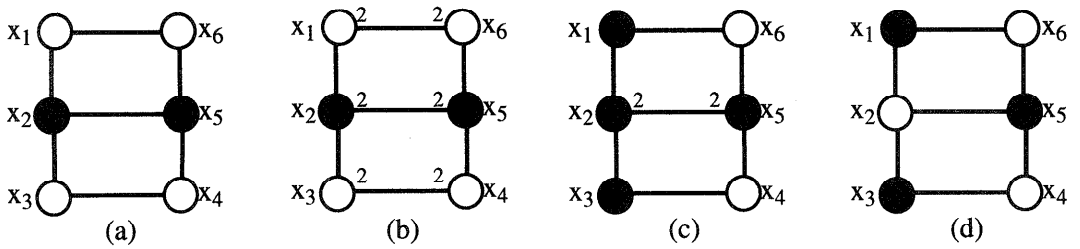

図3 アルゴリズムの実行例
Fig. 3 Example of algorithm execution.

トすべてで制約が満足されていることを示す．各エージェントは他のエージェントに対する距離の最大値，もしくは最大値の適当な上界値（*max_distance*）を知っていることを仮定する．*my_termination_counter* の値がこの上界値と等しくなれば，すべてのエージェントで制約が満足されたことが確認できる．この終了判定方法の正しさは，x_i で *my_termination_counter* が k から $k+1$ に増加するのは，x_i および x_i の近傍のエージェントですべての制約が満足され，かつ x_i の近傍のエージェントすべてで *my_ternimation_counter* の値が k 以上であることから，数学的帰納法により証明することができる．

近傍のすべてのエージェントから improve メッセージを受け取った時点で，状態変数 *quasi_local_minimum* が真であれば制約の重みを変更する．各エージェントは制約の重みを独立に管理しているため，重みの変更に関してエージェント間で合意をとる必要はない．状態変数 *can_move* が真であれば値を変更し，そうでなければ値を変更せず，ok?メッセージを通信して **wait_ok? mode** に移行する．

3.4 アルゴリズムの実行例

具体例を**図3**を用いて示す．例題は分散グラフ色塗り問題であり，各ノードに対応するエージェントが，リンクで結ばれたエージェントと異なる色になるように自分の色を求める問題である．エージェントのとりうる色は白と黒の2種類である．

初期値は図3(a)のようになっていると仮定する．各エージェントはこの初期値を ok?メッセージにより通信しあう．通信された初期値を元に，各エージェントは改善量を計算し，improve メッセージを交換する．初期状態では制約の重みはすべて1である．この場合はどのエージェントも改善が不可能である．よって，現在違反している制約（x_1 と x_6，x_2 と x_5，x_3 と x_4 の間の制約）の重みを1増加し，ok?メッセージを交換し，改めて改善値を通信する（図3(b)）．ここで，x_1，x_3，x_4，x_6 で改善値が1，x_2，x_5 で0となるため，x_1，x_3，x_4，x_6 中で，近傍のエージェント間でアルファベット順で最も先行する x_1 および x_3 が値を白から黒に変更し，ok?メッセージの交換がなされる（図3(c)）．この時点では，x_2 の改善値は4，他の改善値が0であるため，x_2 が黒から白に値を変更し，すべての制約が満足される（図3(d)）．この後，ok?メッセージと improve メッセージの交換が繰り返され，各エージェントの *my_ternimation_counter* の値が与えられた上限値に達した時点で解が得られたことが確認され，処理が終了する．

4. 評　　価

本論文では以下のような計算モデルを用いて評価を行う．すなわち，各エージェントは自分自身のクロックを管理し，送信されたメッセージを受理し，局所的な計算を行い，メッセージを送信するごとにクロックの値を1つ増加させる（これを1サイクルと呼ぶ）．送信されたメッセージは次のサイクルで他のエージェントで受信されるとする．シミュレータを用いて，解を得るのに必要とされるサイクル数を測定する．1サイクルはエージェントが外界の状況を認識し，状況への対応方法を決定し，通信を行うという一連の行動を意味する．分散 breakout アルゴリズム（distributed breakout，DB）では，ok?メッセージの受信と処理，improve メッセージの送信で1サイクル，improve メッセージの受信と処理，ok?メッセージの送信で1サイクルを要し，あるエージェントが1回の値の変更を行うのは，たかだか2サイクルに1回である．

比較の対象として，非同期弱コミットメント探索アルゴリズム[17),18)]（asynchronous weak-commitment search，AWC），山登り+提携アルゴリズム[6)]（hill-climbing+coalition，HCC）を用いる．HCC では，博愛的な戦略[6)]を用い，提携の大きさが5エージェント以上になった場合に新しい初期値から探索を再開することとする．

表1 制約が疎な問題での評価 ($k=3, m=n\times 2$)
Table 1 Evaluation with "sparse" problems ($k=3, m=n\times 2$).

	DB		DB+BC		AWC		HCC	
n	ratio	cycles	ratio	cycles	ratio	cycles	ratio	cycles
90	100%	150.8	100%	230.4	100%	70.1	98%	533.5
120	100%	210.1	100%	253.4	100%	106.4	100%	538.4
150	100%	278.8	100%	344.5	100%	159.2	99%	1074.2

また，準局所最適で重みの変更を行うことの影響を見るため，エージェントが近傍だけでなく他のすべてのエージェントと通信を行い，本当の局所最適の場合にのみ重みの変更を行うアルゴリズム（distributed breakout with broadcasting, DBB）の評価結果を合わせて示す（近傍以外のエージェントからの通信結果は局所最適の判定のみに用い，他の処理は通常の分散breakoutとまったく同等である）．

以下，前述の分散グラフの色塗り問題を用いて評価を行う．この問題は，移動体通信での周波数帯割当問題（隣接する区域では干渉を避けるため同じ周波数帯を使用できない）等を表現していると考えることができる．グラフの色塗り問題はエージェント/変数の個数 n，各エージェントのとりうる色の数 k，エージェント間のリンクの個数 m の3つのパラメータで表現される．

まず，各エージェントのとりうる色の数 k を3，リンクの個数 $m=n\times 2$ 本の場合の，$n=90,120,150$ に関する評価を行う☆．このパラメータの設定は，文献9)で用いられた，変数間の制約の密度が疎である問題に対応し，制約違反最少化ヒューリスティックが効果的でないことが示されている．分散グラフの色塗り問題では，10個の異なる問題に関して，それぞれ10通りの初期値に関して試行を行う（合計100回の試行を行う）．各試行に関して，変数の初期値は乱数を用いて決定する．すなわち，とりうる色の数が3色であるため，確率1/3のさいころを用いて，3色のどの色を初期値とするかを決定する．実験結果を**表1**に示す．実験を妥当な時間内で終了させるため，サイクル数の上限を10000に設定し，この上限を超えた試行は打ち切り，サイクル数を10000とカウントして平均を求める．また，制限時間内で解が得られた試行の割合を表中に示す．

さらに，他のパラメータは同一で，リンクの個数 $m=n\times 2.7$ 本の場合，$m=n\times(n-1)/4$ の場合の評価を**表2**，**表3**にそれぞれ示す．リンクの個数が $m=n\times 2.7$ の場合のパラメータ設定は，文献2)により，非常に難しい問題のインスタンスを作り出すことが示されている．リンクの個数が $m=n\times(n-1)/4$ の場合は，エージェント間の制約の密度が密である場合に相当する．

さらに，$k=4$，リンクの個数 $m=n\times 4.7$ 本の場合の，$n=60,90,120$ に関する評価を**表4**に示す．このパラメータ設定も文献2)により，非常に難しい問題のインスタンスを作り出すことが指摘されている．

これらの評価から以下の知見が得られる．

(1) 制約が疎な問題，密な問題に関しては，非同期弱コミットメント探索アルゴリズムが最も高速である．一方，$k=3$，$m=n\times 2.7$ および $k=4$，$m=n\times 4.7$ の難しい問題に関しては，分散breakoutアルゴリズムが最も効率的である．

(2) 本来の局所最適でなく，準局所最適で値の変更を行うことの悪影響は，制約が疎な問題，密な問題に関してはまったくなく，逆に性能が向上している場合もある．一方，難しい問題に関しては準局所最適で値の変更を行う悪影響は存在し，たとえば $n=120$，$k=4$，$m=n\times 4.7$ の場合，準局所最適で重みの変更を行うことにより，正しい局所最適で重みの変更を行うアルゴリズムと比較してサイクル数は40%近く増加している．

(3) 山登り+提携アルゴリズムは全体に他のアルゴリズムより効率が悪く，特にクリティカルな問題で性能が悪い．

以下，これらの結果が得られた理由を解析する．

(1)のような結果が得られる理由として，制約が疎な問題，密な問題は比較的容易に解を求めることができ，分散breakoutアルゴリズム，山登り+提携アルゴリズムのように近傍間で制御を行うオーバヘッドは探索の削減量に対して引き合わないことが考えられる．分散breakoutアルゴリズムは各エージェントは2サイクルで値の変更をたかだか1回行い，山登り+提携アルゴリズムでは3サイクルにたかだか1回の値の変更を行うのに対して，非同期弱コミットメント探索は

☆ 文献9)で示されている方法に従い，解が存在し，グラフが連結されている問題のみを生成している．

表2 クリティカルな問題での評価（$k=3, m=n\times 2.7$）
Table 2 Evaluation with "critical" problems ($k=3, m=n\times 2.7$).

	DB		DB+BC		AWC		HCC	
n	ratio	cycles	ratio	cycles	ratio	cycles	ratio	cycles
90	100%	517.1	100%	397.1	97%	1869.6	66%	5305.7
120	100%	866.4	100%	693.0	65%	6428.4	37%	7788.2
150	100%	1175.5	100%	687.7	29%	8249.5	19%	8874.0

表3 制約が密な問題での評価（$k=3, m=n\times(n-1)/4$）
Table 3 Evaluation with "dense" problems ($k=3, m=n\times(n-1)/4$).

	DB		DB+BC		AWC		HCC	
n	ratio	cycles	ratio	cycles	ratio	cycles	ratio	cycles
90	100%	31.2	100%	31.2	100%	9.9	100%	65.6
120	100%	34.6	100%	34.4	100%	9.3	100%	70.2
150	100%	34.9	100%	34.9	100%	9.6	100%	70.7

表4 クリティカルな問題での評価（$k=4, m=n\times 4.7$）
Table 4 Evaluation with "critical" problems ($k=4, m=n\times 4.7$).

	DB		DB+BC		AWC		HCC	
n	ratio	cycles	ratio	cycles	ratio	cycles	ratio	cycles
60	100%	591.3	100%	497.1	100%	1733.6	61%	19953.8
90	100%	1175.8	100%	691.8	83%	14897.3	26%	32923.9
120	100%	2218.1	100%	1616.7	25%	34771.6	9%	38028.1

1サイクルに1回の値の変更が可能である．これに対して，非常に難しい問題では近傍のエージェント間で制御を行い，慎重に値を決定することが効果的であることが示されている．

一方，集中型の制約充足問題では，非常に難しい問題（節の密度が4.3の3-SAT問題等）においても，弱コミットメント探索の方がbreakoutアルゴリズムよりも効率的であることが示されている[15),16)]．この違いの原因は何であろうか？

1つの理由として，集中型の弱コミットメント探索アルゴリズムでは，forward-checking[5)]等の変数の順序付けのヒューリスティックが導入され，また，先読みを行うことにより，ただちに失敗に至るような値の選択を避けることが可能となっていることがあげられる．非同期弱コミットメント探索において，分散breakoutアルゴリズムと同様な近傍間の制御を導入することにより，forward-checkingのような変数/エージェントの順序付けを行うことは可能ではあるが，複数のエージェントに変数と制約に関する知識が分散された状況では，値の決定の影響を先読みすることは難しい．

さらに，集中型のbreakoutアルゴリズムは弱コミットメント探索アルゴリズムと比較して，評価値を改善するように値の変更を行わなければならないため，一般に1回の値の変更に要する処理が多く，最悪の場合（局所最適の場合）には，すべての変数に関して，改善が可能かどうかをチェックする必要がある．一方，分散breakoutアルゴリズムでは，各変数に関する処理は複数のエージェントにより並行して行われる．すなわち，breakoutアルゴリズムには並列に実行可能な処理が多く，分散されたエージェントでの実行に適している．

(2)の結果が得られる理由として，制約が疎な場合，あるエージェント x_i から見て現在の状況が準局所最適であるが，全体としては局所最適でない場合，他の改善可能なエージェントが値の変更を行っても，その影響は x_i に対して比較的小さく，結局は x_i が重みの変更をしなければならないことが考えられる．また，制約が密な問題では局所最適，準局所最適に陥ることが非常に稀である．難しい問題に関しては準局所最適で値の変更を行う悪影響は存在するが，エージェント全体で通信を行う場合，近傍のみで通信を行う場合と比較して，このパラメータ設定で10倍以上のメッセージが必要となる．このメッセージ通信のコストを考えると，準局所最適で値の変更を行うことの性能劣化は十分許容可能であると考えられる．

(3)の結果が得られる理由として，難しい問題では提携が大きくなる傾向があり，設定された提携の大きさの上限（実験では5）では小さすぎることが考えられる．しかしながら，提携の上限を大きくすると制約チェックの回数が多くなり，アルゴリズム全体として

の性能は劣化する．

5. 考　　察

分散breakoutアルゴリズムの1つの問題点は，アルゴリズムの完全性が保証されないことであり，特に，解が存在する問題に関しても，処理の無限ループが生じる可能性があり，解を求めることは保証されない．非同期弱コミットメント探索アルゴリズム，山登り+提携アルゴリズムでは，解が存在する場合には必ず解を求め，解がない場合には解がないことを発見することが保証されている．

一方，分散breakoutアルゴリズムの利点として，通信がつねに近傍との間のみで行われることがあげられる．非同期弱コミットメント探索アルゴリズムでは，探索途中で得られた新しい制約の記録を行う場合（この処理は完全性を保証するためには必須である）には，初期状態では直接の関連がないエージェント間で情報の交換を行う必要が生じうる．また，山登り+提携アルゴリズムでも，初期状態では直接の関連がない複数のエージェントが同じ提携に属する可能性があり，問題に関する情報を委譲する必要が生じる．

さらに，分散breakoutアルゴリズムでは終了判定の処理がアルゴリズム中に組み込まれているが，他の2つのアルゴリズムでは終了判定のための別のアルゴリズム[1)]を起動する必要がある．

文献4)では，コネクショニストアーキテクチャを用いてbreakoutアルゴリズムに似た分散型のアルゴリズムを実装している．この方法では近傍のclusters（エージェントに相当する）が同時に値を変更することを禁止していないため，ネットワークが発振する可能性がある☆．また，ネットワーク全体として一定のステップの間に変化がない場合に局所最適であるとの判断を行っている．このような大局的な情報は分散制約充足問題で用いることは困難である．

6. むすび

本論文では分散制約充足問題を解くための新しいアルゴリズムである分散breakoutアルゴリズムを提案した，本アルゴリズムは集中型の制約充足問題を解くための反復改善型のアルゴリズムであるbreakoutアルゴリズムに触発されたものであり，各エージェントは近傍のエージェントと現在の変数の値の割当，可能な改善方法を交換し，エージェント全体として制約条件違反の個数が減少するように値の変更を行う．また，エージェント全体として局所最適に陥ったことを検出するのではなく，近傍との通信のみで検出できる，より弱い条件である準局所最適を検出し，制約の重みを変更することにより準局所最適から脱出する．実験的な評価により，本アルゴリズムは既存のアルゴリズムと比較して，非常に難しい問題のインスタンスに関して効率的であることが示された．

今後の課題として，他の例題，特に移動体通信での周波数割当問題のような現実的なアプリケーションでアルゴリズムの性能の比較検討を行うこと，また，アルゴリズムの性能に関する理論的な考察を行うことがあげられる．

謝辞　本研究の最初のアイデアはMultiagent Research community in Kansai（MARK）のワークショップでのディスカッションで得られた．有益なご討論をいただいたMARKのメンバの皆様とMARKをサポートいただいている（株）けいはんなのご厚意に感謝します．また，日頃からご指導，ご討論いただくNTTコミュニケーション科学研究所の松田所長，同研究所計算機科学部の服部部長，大阪大学豊田教授，東京工業大学山田助教授に感謝します．

参考文献

1) 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).
2) Cheeseman, P., Kanefsky, B. and Taylor, W.: Where the really hard problems are, *Proc. 12th International Joint Conference on Artificial Intelligence*, pp.331–337 (1991).
3) Conry, S.E., Kuwabara, K., Lesser, V.R. and Meyer, R.A.: Multistage Negotiation for Distributed Constraint Satisfaction, *IEEE Trans. Systems, Man and Cybernetics*, Vol.21, No.6, pp.1462–1477 (1991).
4) Davenport, A., Tsang, E., Wang, C.J. and Zhu, K.: GENET: A Connectionist Architecture for Solving Constraint Satisfaction Problems by Iterative Improvement, *Proc. 12th National Conference on Artificial Intelligence*, pp.325–330 (1994).
5) Haralick, R. and Elliot, G.L.: Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence*, Vol.14, pp.263–313 (1980).
6) Hirayama, K. and Toyoda, J.: Forming Coalitions for Breaking Deadlocks, *Proc. 1st international Conference on Multiagent Systems*,

☆ cluster間の通信が非常に高速で，各clusterが非同期に動作する場合には，発振することは稀であると考えられる．

pp.155–162 (1995).
7) Huhns, M.N. and Bridgeland, D.M.: Multiagent Truth Maintenance, *IEEE Trans. Systems, Man and Cybernetics*, Vol.21, No.6, pp.1437–1445 (1991).
8) Mackworth, A.K.: Constraint Satisfaction, *Encyclopedia of Artificial Intelligence*, Shapiro, S.C. (Ed.), pp.285–293, Wiley-Interscience Publication, New York (1992).
9) Minton, S., Johnston, M.D., Philips, A.B. and Laird, P.: Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems, *Artificial Intelligence*, Vol.58, No.1–3, pp.161–205 (1992).
10) Morris, P.: The Breakout Method for Escaping From Local Minima, *Proc. 11th National Conference on Artificial Intelligence*, pp.40–45 (1993).
11) Selman, B., Levesque, H. and Mitchell, D.: A New Method for Solving Hard Satisfiability Problems, *Proc. 10th National Conference on Artificial Intelligence*, pp.440–446 (1992).
12) Sosič, R. and Gu, J.: Efficient Local Search with Conflict Minimization: A Case Study of the n-Queens Problem, *IEEE Trans. Knowledge and Data Engineering*, Vol.6, No.5, pp.661–668 (1994).
13) Sycara, K.P., Roth, S., Sadeh, N. and Fox, M.: Distributed Constrained Heuristic Search, *IEEE Transactions on Systems, Man and Cybernetics*, Vol.21, No.6, pp.1446–1461 (1991).
14) 横尾 真，エドモンド H. ダーフィ，石田 亨，桑原和宏：分散制約充足による分散協調問題解決の定式化とその解法，電子情報通信学会論文誌，Vol.J-75 D-I, No.8, pp.704–713 (1992).
15) Yokoo, M.: Weak-commitment Search for Solving Constraint Satisfaction Problems, *Proc. 12th National Conference on Artificial Intelligence*, pp.313–318 (1994).
16) 横尾 真：弱コミットメント戦略を用いた制約充足問題の解法，情報処理学会論文誌，Vol.35, No.8, pp.1540–1548 (1994).
17) Yokoo, M.: Asynchronous Weak-commitment Search for Solving Distributed Constraint Satisfaction Problems, *Proc. 1st International Conference on Principles and Practice of Constraint Programming* (*Lecture Notes in Computer Science 976*), pp.88–102, Springer-Verlag (1995).
18) 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).
19) Yokoo, M., Durfee, E.H., Ishida, T. and Kuwabara, K.: Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving, *Proc. 12th IEEE International Conference on Distributed Computing Systems*, pp.614–621 (1992).

(平成 9 年 7 月 28 日受付)
(平成 10 年 3 月 6 日採録)

横尾 真（正会員）

1962 年生．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．1990～1991 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研究所に勤務．分散人工知能，制約充足問題に関する研究に従事．分散探索，分散制約充足問題等に興味を持つ．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．人工知能学会，日本ソフトウェア科学会，AAAI 各会員．

平山 勝敏（正会員）

1967 年生．1990 年大阪大学基礎工学部制御工学科卒業．1995 年同大学院基礎工学研究科博士後期課程修了．同年神戸商船大学商船学部助手．1997 年同大学商船学部講師，現在に至る．博士（工学）．制約充足問題，マルチエージェントシステムに関する研究に従事．人工知能学会，日本ソフトウェア科学会，AAAI 各会員．