Vol. 36 No. 2 情報処理学会論文誌 Feb. 1995

分散制約充足問題における制約緩和

横尾 真†

分散制約充足問題は分散協調問題解決の様々な問題を表現できる枠組として近年注目を集めている．本論文では，制約の重要度という概念を導入して分散制約充足問題の枠組の拡張を行い，制約が強過ぎて解が存在しない過制約である分散制約充足問題に関して，制約を部分的に満たす解の定式化を行う．さらに，分散制約充足問題を解くアルゴリズムである非同期バックトラッキングアルゴリズムを繰り返し適用して，重要度の低い制約を段階的に緩和するアルゴリズム（非同期段階緩和アルゴリズム）により，定義された基準で最適な解を求めることが可能であることを示す．本アルゴリズムでは，通常の逐次的なバックトラッキングアルゴリズムとは対照的に，エージェントは非同期に並行して動作するが，アルゴリズムの完全性，解の最適性は保証される．さらに，本アルゴリズムにおいて，制約条件違反（nogood）と制約との間の依存関係を管理することにより，無駄な計算を避けることができ，例題において5倍程度の速度向上が得られることを示す．

Consrtaint Relaxation in Distributed Constraint Satisfaction Problem

Makoto Yokoo†

The distributed constraint satisfaction problem (DCSP) formulation has recently been identified as a general framework for formalizing various Distributed Artificial intelligence problems. In this paper, we extend the DCSP formalization by introducing the notion of importance values of constraints. With these values, we define a solution criterion for DCSPs that are over-constrained (where no solution satisfies all constraints completely). We show that agents can find an optimal solution with this criterion by using the *asynchronous incremental relaxation algorithm*, in which the agents iteratively apply the *asynchronous backtracking algorithm* to solve a DCSP, while incrementally relaxing less important constraints. In this algorithm, agents act asynchronously and concurrently, in contrast to traditional sequential backtracking techniques, while guaranteeing the algorithm completeness and the solution optimality. Furthermore, we show that, in this algorithm, agents can avoid redundant computation and achieve a five-fold speed-up in example problems by maintaining the dependencies between constraint violations (nogoods) and constraints.

1. まえがき

分散協調問題解決とは人工知能の研究の一分野であり，複数の人工的な知的エージェントの協調的な問題解決を対象とする．我々は文献 9), 10) において，分散協調問題解決を定式化する一般的な枠組を与えるために，分散協調問題解決を分散制約充足問題とみなすというアイデアを提案した．分散制約充足問題は非常に一般的な問題であり，文献 1), 5), 8) 等で，分散資源割当問題，分散スケジューリング問題，マルチエージェント Truth maintenance のそれぞれが，この分散制約充足問題として定式化されることが指摘されている．我々は文献 10) において，与えられた制約をすべて満足する解を求める分散アルゴリズムである非同期バックトラッキングアルゴリズムの開発を行った．

一方，集中型の制約充足問題の研究において，多くの現実の問題では，制約が強過ぎて解が存在しないため，制約を緩和することにより，適当な解を求めることが重要であることが指摘されており[2)-4)]，分散制約充足問題を拡張して過制約で解が存在しない問題に対応することは重要な課題である．

与えられた問題の制約を緩和する場合，緩和した問題の解に関して，その良さを測るなんらかの基準が必要である．本論文では，制約の重要度という概念を導入して分散制約充足問題の枠組を拡張し，解の良さの基準を定式化する．

さらに，分散制約充足問題を解くアルゴリズムである非同期バックトラッキングアルゴリズムを繰り返し

† NTT コミュニケーション科学研究所
NTT Communication Science Laboratories

適用して，重要度の低い制約を段階的に緩和するアルゴリズム（非同期段階緩和アルゴリズム）により，定義された基準で最適な解を求めることが可能であることを示す．本アルゴリズムでは，通常の逐次的なバックトラッキングアルゴリズムとは対照的に，エージェントは完全に非同期に並行して動作するが，アルゴリズムの完全性，解の最適性は保証される．さらに，本アルゴリズムにおいて，制約条件違反（nogood）と制約との間の依存関係を管理することにより，無駄な計算を避けることができることを示す．

本論文では以下，制約の重要度が導入された集中型の制約充足問題の定式化および制約緩和アルゴリズムについて紹介した後（2章），制約の重要度が導入された分散制約充足問題の定式化を示し（3章），非同期段階緩和アルゴリズムの説明を行い（4章），実験結果を示す（5章）．また，本アルゴリズムのより一般的な解の条件への適用に関して議論する（6章）．

2. 集中型の制約充足問題

2.1 定式化

制約の重要度を導入した集中型の制約充足問題の定義は次のとおりである．

- 変数の集合 $x_1, \cdots, x_m$ が存在する．変数 x_i は有限で離散的な領域 D_i から値を取る．
- 有限個の変数間の制約の集合 P が存在する．制約 $p_k(x_{k1}, \cdots, x_{kj})$ は直積 $D_{k1} \times \cdots \times D_{kj}$ 上で定義された述語であり，対応する変数の値が互いに整合が取れている場合に真となる．
- 制約 p_k に対して，その制約の重要度を示す正の実数値 c_k（c_k は大きいほどその制約が重要であることを示す）が定義される．
- 目的は，より重要な制約をより多く満たす解を求めることである．すなわち，二つの解 S, S' が存在し，S が重要度が c 以上の制約をすべて満たすのに対し，S' が重要度 c 以上の制約のいずれかを満たさない場合に，S が S' よりも選好される．目的はどのような解によってもより選好されない解を求めることである．

現実の問題を制約充足問題として定式化する場合，すべての制約が同様に重要であるとは考え難く，どの制約を緩和するべきかに関するなんらかの基準が存在すると仮定することは自然である．この定式化では，そのような基準が制約の重要度として表現されていると仮定している．より複雑な解の評価基準を用いること，例えば満足される制約の重要度の和を用いる，あるいは制約緩和の基準として全順序ではなく半順序関係を用いることも可能である．しかしながら，解の評価基準の複雑さ／表現能力は，その基準の元で最適な解を求めるコストに影響する．この重要度を用いた評価基準は，基準の複雑さ／表現能力と解を求めるコストの適当な妥協点を与えていると考えられる*．

変数への値の割当 S に関して，$F(S)$ を S が満足しない制約の重要度の最大値を返す（S がすべての制約を満足する場合は0を返す）評価関数として定義することにより，この定式化における選好関係が，この評価関数の評価値の大小関係と同値となる（評価値が小さい方が望ましい）．よって，目的はこの評価値を最小化する解を求めることと置き換えられる．

図1に問題の例を示す．この問題では，3×3のチェスボード上の各行に一つのクイーンを互いに取り合わないように配置する問題であり，各行のクイーンの位置に対応する変数 x_1, x_2, x_3 に $\{1, 2, 3\}$ の値を割り当てる制約充足問題として定式化できる．この問題は明らかに過制約である．二つのクイーンの制約条件違反の深刻さは，クイーンの距離の二乗に反比例すると仮定し，以下のように制約を定義する．

- 変数 x_i と x_j の間に，重要度 $1/k$ ($k \in \{1, 2, 4, 8\}$) の制約 $p_{ij,k}$ が存在する．この制約は $((x_i \neq x_j) \wedge (|x_i - x_j| \neq |i-j|)) \vee (x_i - x_j)^2 + (i-j)^2 > k$ の場合に真となる．

図1のクイーンの配置は，重要度が 1/4 より大きい制約をすべて満足しているため，この解の評価値は 1/4 であり，どのような配置も，評価値は 1/4 以上であるため，この配置が最適な解となる．

この定式化により，変数の取り得る値の数が多く，すべての可能な値があらかじめ数え挙げられていない場合に対応することが可能となる．例えば，変数 x_i の領域が，10,000 以下の素数の集合である場合，変数の領域をすべて求めることはコストがかかる．ここで，変数の値に関する付加的な制約，$p_c(x_i) \equiv x_i \leq c$

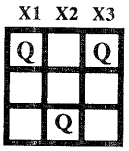


図1 3クイーン問題
Fig. 1 Three-queens Problem.

* 6章で，この解の条件の一般化に関して議論する．

を導入する．c の取り得る値は例えば {1000, 2000, 3000, ⋯, 10, 000} であり，これらの制約の重要度は c であるとすると，これらの付加的な制約は，やむをえない場合にのみ重要でない制約から順に緩和が行われるため，不要な変数の値を求めることを回避することが可能となる．このように，問題が過制約でない場合でも，必要な場合は緩和可能な付加的な制約を加えることにより，効率的に問題を解くことが可能となる場合が存在する．

2.2 アルゴリズム

集中型の制約緩和問題の解を求めることは，評価関数の値を最小化する解を探索することであり，A* や Depth-first Branch & Bound 等の，一般の状態空間探索の手法が適用可能である[7]．文献 4) では，Depth-first Branch & Bound アルゴリズムを用いて，与えられた解の基準を最適化する．集中型の制約充足問題の解を求める方法が示されている．

3. 分散制約充足問題の定式化

3.1 通信モデル

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時間であるが，遅延時間の上限は分かっていない．
- 任意の二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．

3.2 分散制約充足問題

分散制約充足問題は，変数と制約が複数のエージェント間に分散された制約充足問題である．

- エージェント $1, \cdots, n$ が存在する．
- それぞれの変数は，いずれか一つのエージェントに属する．
- 各エージェントは自分の変数が引数となっているすべての制約の集合 P_i と，P_i の他の引数を持つエージェントのアドレスを知っている．

図1の例題で，3個のエージェントが存在し，各エージェントがそれぞれ担当する行に一つのクイーンを配置しようとしていると仮定すれば，この問題は分散制約充足問題として表現できる．分散制約充足問題においても，集中型の制約充足問題と同様に制約の重要度および解の評価関数が定義される．エージェントの目的は評価関数の評価値を最小化する解を求めることである．

4. 非同期段階緩和アルゴリズム

4.1 基本アルゴリズム

非同期バックトラッキングアルゴリズムは，与えられた制約をすべて満足する解が存在すればその解を見つけ，存在しない場合には，解が存在しないことを発見することができる．よって，次のような方法により，繰り返し非同期バックトラッキングアルゴリズムを適用して，制約緩和を行った場合の最適解を求めることができる．

- ある閾値（初期値は適当な下界値か 0）を設定し，重要度がその閾値以下の制約をすべて緩和し，重要度がその閾値よりも大きな制約のみを考慮して非同期バックトラッキングを用いて解を求める．与えられた閾値よりも重要度が大きな制約をすべて満たす解が存在しない場合，現在考慮している制約のうち，最も小さい重要度を持つものを緩和し，その重要度を新しい閾値として，新しい閾値よりも重要度が大きな制約のみを考慮して，非同期バックトラッキングを用いて解を求めることを繰り返す*．

別の方法として，制約を段階的に強化していく，Depth-first Branch & Bound 的なアルゴリズムが考えられる．しかしながら，制約を段階的に強化していく方法は，変数の領域に関して付加的な制約を加えている場合には不適切である．なぜならば，このアルゴリズムでは最初に，可能な限り制約を緩和した状態で解を求めるため，最終的な解となり得ない変数の値の多くを求める可能性がある．一方，本アルゴリズムでは制約は必要な場合にのみ緩和されるため，不要な値を求めることはない．

---

* この通信モデルは，物理的な通信ネットワークの形状が完全グラフである（すべてのエージェント間に直接の物理的な通信リンクが存在する）ことを仮定するものではない．多くの並列/分散アルゴリズムの研究においては，物理的な通信ネットワークの形状が重要な意味を持つが，本研究では信頼できる低レベルの通信ネットワークが存在することを前提として，実際の物理的ネットワークのインプリメンテーションについては考慮しない．

* この最小値はエージェント間の適当なメッセージ通信によって得られると仮定する．

以下，この基本となるアルゴリズムに対して，エージェント間で通信される制約条件違反に関する情報(nogood)に，その制約条件違反が成立する条件(nogoodの重要度)を付加するという拡張を行い，多くの無駄な計算を避けることが可能とした，非同期段階緩和アルゴリズムについて説明する．

4.2 nogoodの依存関係を導入した非同期段階緩和アルゴリズム

4.2.1 nogoodの依存関係

非同期バックトラッキングアルゴリズムでは，エージェントは制約条件違反に関する情報(nogood)を交換し合う．nogoodは制約条件違反を引き起こす変数の値の組である．例えば，図1の問題で，もし$x_1=1$で$x_2=3$であると，x_3にはこれらの値の組合せと制約を満たす値は存在しないため，値の組合せ$\{(x_1,1),(x_2,3)\}$はnogoodである．nogoodのスーパセットとなるような値の組合せは最終的な解にはなり得ない．もし空集合からなるnogoodが発見された場合，どのような値の組合せも最終的な解になり得ず，制約を満たす解は存在しない．

本論文ではnogoodを一般化し，nogoodN_kに対して，N_kの重要度を示す正の値cを付加する．nogoodの重要度は，そのnogoodの成立に寄与している制約とnogoodとの依存関係を示す．すなわち，nogoodの重要度がcであることは，重要度がc以下の制約がすべて緩和された場合に，このnogoodが無効となることを示す．nogoodの重要度は次のように定義される．

- nogoodの重要度は，そのnogoodが制約条件違反であることに寄与している制約の重要度の最小値で与えられる．

例えば，前述のnogood$\{(x_1,1),(x_2,3)\}$は，すべての制約が満足されなければならないという条件の元で制約条件違反であるが，いくつかの制約が緩和されれば制約条件違反ではなくなる．すなわち，$\{(x_1,1),(x_2,3),(x_3,1)\}$は，重要度が1/4より大きい制約をすべて満たすため，もし重要度1/4の制約を無視することができれば制約条件違反ではなくなる．同様に，$\{(x_1,1),(x_2,3),(x_3,2)\}$，$\{(x_1,1),(x_2,3),(x_3,3)\}$は，それぞれ重要度1/2，1の制約を無視することにより，制約条件違反ではなくなる．よって，nogood$\{(x_1,1),(x_2,3)\}$の重要度は1/4となる．これは，もし重要度1/4の制約を無視することができれば，$\{(x_1,1),(x_2,3)\}$が解の一部となる可能性があるが，閾値が1/4より小さく，重要度1/4の制約が緩和されていない場合には$\{(x_1,1),(x_2,3)\}$を含む解は存在し得ないことを示している．

このnogoodの重要度を用いて，次のように無駄な計算を避けることが可能となる．

無駄な閾値に関する探索を避ける：

空集合であるnogoodが発見された場合，このnogoodの重要度がc_kであれば，新しい閾値はc_kとするべきであることが分かる．nogoodの重要度は，このnogoodに寄与している制約の重要度の最小値であり，新しい閾値をc_kより小さく設定しても，すなわち，重要度がc_kより小さい制約を緩和しても，このnogoodはまだ有効であり，解は得られない．よって，重要度がcより小さい制約を緩和して探索を行うことは無駄である．

例えば図1の例で，閾値が0の場合（すべての制約を考慮した場合），問題は過制約で空集合からなるnogoodが得られるが，このnogoodの重要度は1/4となる．よって，重要度が1/8の制約が存在するにも関わらず，この制約は現在の閾値の元で解が得られなかったことに寄与しておらず，新しい閾値は1/4となり，重要度が1/4以下の制約は緩和されるべきであることが分かる．

無駄な再計算を避ける：

閾値が変更され，制約が緩和された場合には，nogoodの重要度を導入しない場合は，制約を緩和する以前の計算で得られたnogoodはすべて廃棄し，改めて計算を最初からやり直すことが必要であった．一方，nogoodの重要度を導入した場合は，現在の閾値よりも重要度が大きいnogoodは，新しい閾値の元で有効であり，いくつかの制約を緩和した後でも廃棄する必要はない．このようなnogoodを用いて無駄な再計算を避けることが可能となる．

例えば図1の例で，nogood$\{(x_1,1)\}$の重要度は1/4で，nogood$\{(x_1,2)\}$の重要度は1/2である．閾値が1/4に増加した場合に，前者は新しい閾値の元でもはや有効でないが，後者はまだ有効である．

4.2.2 アルゴリズムの詳細

以下のアルゴリズムの説明の際，簡単のため，各エージェントは唯一の変数を持つという仮定をおく．この仮定を緩和して，以下のアルゴリズムをエージェントが複数の変数を持つ場合に拡張することは容易である．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．

本アルゴリズムでは，各エージェントは順番にでは

```
when initialized do — (i)
  select d ∈ D_i where F_i({(x_i, d)}) ≤ threshold; — (i - a)
  current_value← d;
  send (ok?, x_i, d) to receiver_list; end do; — (i - b)

when received (ok?, x_j, d_j) do — (ii)
  add (x_j, d_j) to agent_view;
  check_agent_view; end do;

when received (nogood, x_k, nogood, condition) do — (iii)
  add (nogood, condition) to nogood_list;
  when (x_j, d_j) where x_j ∉ sender_list is in nogood do
      request x_j to add x_i to its receiver_list;
      add x_j to sender_list; add (x_j, d_j) to agent_view; end do;
  old_value ← current_value;
  check_agent_view;
  when old_value = current_value do
    send (ok?, x_i, current_value) to x_k; end do; end do;

when received (revise_threshold, x_j, new_threshold) do — (iv)
  when new_threshold > threshold do
    threshold ← new_threshold;
    send (revise_threshold, x_i, new_threshold)
         to other agents except x_j; end do; end do;

procedure check_agent_view — (v)
  when F_i(agent_view ∪ {(x_i, current_value)}) > threshold do
    if there exists d ∈ D_i
        where F_i(agent_view ∪ {(x_i, d)}) ≤ threshold then — (v - a)
      current_value ← d;
      send (ok?, x_i, d) to receiver_list;— (v - b)
    else backtrack; check_agent_view; end if; end do;

procedure backtrack — (vi)
  nogoods ← {V | V ⊂ agent_view and
              min_{d∈D_i} F_i(V ∪ {(x_i, d)}) > threshold};
  for each V ∈ nogoods do;
    new_condition ← min_{d∈D_i} F_i(V ∪ {(x_i, d)}) — (vi - a)
    if V={} then threshold ← new_condition;
      send (revise_threshold, x_i, threshold) to other agents; — (vi - b)
    else select (x_j, v_j) from V where j is the largest;
      send (nogood, x_i, V, new_condition) to x_j;— (vi - c)
      remove (x_j, v_j) from agent_view; end if; end do;
```

図 2 メッセージ受信で起動される手続き
Fig. 2 Procedures for receiving messages.

なく，並行，非同期に値を決定し，関連する他のエージェントに決定した値を送信する．**図 2**(i)にエージェント x_i で実行される初期化手続きを，図 2(ii)，(iii)，(iv)のそれぞれに，3種類のメッセージ *ok?*，*nogood*，*revise_threshold* を受けた時のエージェント x_i で起動される手続きを示す．

これらの手続きの主な内容は次のとおりである．

- 各エージェントは最初，並行して自分の変数の値を選び，その値を関連するエージェントに対して送信する(図 2(i))．その後，各エージェントはメッセージ待ちの状態となる．
- エージェント x_i は，制約で関連し合うエージェントを，その id を用いて 2 種類のグループに分類する．一つは自分の id よりも小さな id を持つもの (*sender_list*) であり，もう一つは自分の id よりも大きな id を持つもの (*receiver_list*) である．エージェントは，*receiver_list* に含まれるエージェントに対して自分の変数の値 (*current_value*) を通信する (図 2(v-b, i-b))．
- 一方，エージェントは *sender_list* に含まれるエージェントから送信された値を *agent_view* に記録する(図 2(ii))．エージェントはこの *agent_view* と制約を満足するように自分の値を変更する (図 2(v))．このように，変数の id によって *ok?* メッセージの送信方向を決定することは，処理の無限ループを避けるために必要とされる．詳細は文献 10) を参照されたい．
- エージェントが，*agent_view* に関して，制約を満足する値を発見できない場合には，エージェントは，*agent_view* 中のエージェントに対して *nogood* メッセージを送信することにより，値の変更を要求する (図 2(vi-c))．

本アルゴリズムと，基本となる非同期バックトラッキングアルゴリズムとの違いは以下のとおりである．

- エージェントは制約をすべて満たす値を選択するのではなく，現在の閾値より重要度が大きな制約のみを満足する値を選択する (図 2(i-a, v-a))．本アルゴリズムでは，nogood のチェックを導入した，部分解の評価値を求める関数 F_i を用いている．$F_i(S)$ は，S が P_i 中の制約をすべて満たし，かつ *nogood_list* 中のどの nogood のスーパセットでもない場合には 0 を返し，その他の場合には，P_i 中で満足されない制約と S のサブセットである nogood の重要度の最大値を返す．
- *nogood* メッセージにその nogood の重要度が付加される (図 2 (iii, vi-c))．新しい nogood を生成する際に，可能な値に関する F_i の最小値を取ることにより，新しい nogood の成立条件の計算がなされる (図 2(vi-a))．
- 空集合からなる nogood が発見された際に，エージェント間で *revise_threshold* メッセージが交換され，閾値の変更がなされる (図 2(iv, vi-b))．

本アルゴリズムでは，各エージェントは完全に非同期に並行して動作し，現在の閾値より重要度が大きい制約をすべて満たす解が得られた時点で，すべてのエージェントがメッセージ待ちの安定状態に達する*．

* 注意すべきことは，全体としてすべてのエージェントが安定状態に達したことを各エージェントが知る方法はアルゴリズム中に含まれていないことである．すべてのエージェンが安定状態に達したことを知り，アルゴリズムを終了させるためには，文献 6) 等で提案されている終了判定アルゴリズムをエージェントが起動する必要がある．

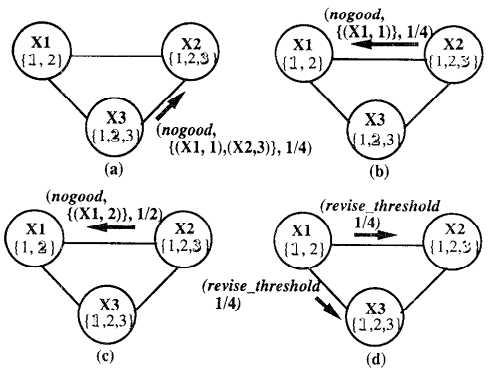


図 3 アルゴリズムの実行例
Fig. 3 Example of algorithm execution.

4.2.3 アルゴリズムの実行例

図1の例題を用いてアルゴリズムの実行経過の概要を示す（**図 3**）．図3では，エージェントを丸で，エージェント間の制約を線で表している．簡単のため，x_1 の領域は {1, 2} のみに制限されていると仮定する（問題の対象性より，この制限を行っても最適解は失われない）．

Step 1：まず，各エージェントは $x_1=1$，$x_2=3$，$x_3=2$ のように変数の値を設定し，*ok?* メッセージを送信する．このメッセージを受け，エージェント x_2 では $x_1=1$ と制約を満たすため値の変更は行われない．一方，エージェント x_3 では，*agent_view* $\{(x_1, 1), (x_2, 3)\}$ と制約を満たす解が存在しないため，エージェント x_2 に対して nogood メッセージ（nogood, $\{(x_1, 1), (x_2, 3)\}$, 1/4）が送られる（図3（a））．

Step 2：エージェント x_2 では，この nogood メッセージを受けて値を変更しようとするが，ほかに可能な値が存在しないため，x_1 に対して nogood メッセージ（nogood, $\{(x_1, 1)\}$, 1/4）が送られる（図3（b））．

Step 3 この nogood メッセージを受けて，エージェント x_1 は値を1から2に変更する．しかしながらこの値と制約を満たす x_2 の値は存在せず，x_1 に対して nogood メッセージ（nogood, $\{(x_1, 2)\}$ 1/2）が送られる（図3（c））．

Step 4：x_1 の領域を {1, 2} に制限しているため，この nogood メッセージを受けて，x_1 の可能な値はなくなり，空集合からなる nogood が生成される．この nogood の重要度は 1/4 である（1/4 と 1/2 の小さい方）．ここでエージェント x_1 で閾値が 1/4 に変更され，この変更を伝えるメッセージが他のエージェントに通信される（図3（d））．ここで，$x_1=1$，$x_2=3$ となり，閾値が 1/4 に変更されているため，今回は x_3 は1となることが可能で，エージェントは安定状態になり，現在の変数への値の割当が最適な解となる．

4.2.4 アルゴリズムの完全性と計算量

本アルゴリズムのベースとなっている非同期バックトラックアルゴリズムの完全性[10]により，本アルゴリズムの完全性も保証される．制約の個数は有限であり，よって制約の重要度の取り得る値も有限である．本アルゴリズムは，非同期バックトラッキングアルゴリズムを繰り返し適用し，一回の適用により閾値の増加が必ず生じ，かつ新しい値は制約の重要度の取り得る値となっている．よって，有限回の繰り返しにより，閾値は最大の制約の重要度に達し，この場合はすべての制約が緩和されるため必ず解が得られ，エージェントはメッセージ待ちの安定状態に達する．

本アルゴリズムの計算量を考察するために，次のような計算モデルを用いる．すなわち，各エージェントがすべてのメッセージを読み，内部の処理を行い，メッセージを送信するのを一ステージとし，あるステージで送信されたメッセージは次のステージにおいて他のエージェントに受けとられると仮定する．残念ながら，一回の繰り返しにおける非同期バックトラッキングアルゴリズムの最悪の場合のステージ数は変数の数 m に関して指数的である．繰り返しの回数の上界値は，異なる制約の重要度の個数で与えられる．ステージ数が最悪の場合が指数的になるのは，NP 完全である制約充足問題を対象としているため避けられないものであると考えられる．

5. 実験結果

本章では，nogood の重要度を導入することによる無駄な計算の削減効果を例題に関する実験結果により示す．実験では前章で述べた計算モデルを用いる．例題として，分散型の n-queens 問題において，問題を過制約とするため，変数の取り得る値を，$i=1, \cdots, n/2$ に関しては $\{1, \cdots, n/2\}$ に，$i=n/2+1, \cdots, n$ に関しては $\{n/2+1, \cdots, n\}$ に制限した問題を用いる（n は偶数であることを仮定する）．

図 4 に，n を変化させた場合の，基本となる非同期段階緩和アルゴリズム（Basic Asynchronous Incremental Relaxation, Basic AIR）と nogood の依存関

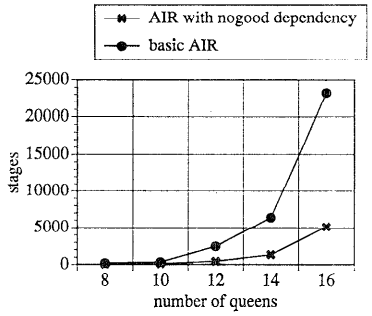


図 4 n クイーン問題で必要とされるステージ数
Fig. 4 Required stages for over-constrained distributed n-queens Problem.

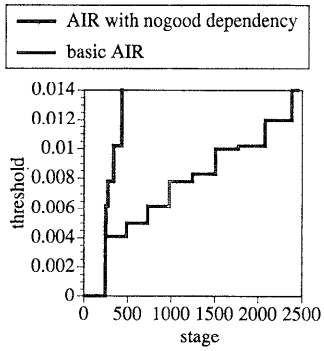


図 5 12 クイーン問題での閾値の変化
Fig. 5 Threshold Change in over-constrained distributed 12-queens Problem.

係を導入した非同期段階緩和アルゴリズム (AIR with nogood dependency) との，過制約な n-queens 問題を解くのに必要とされるステージ数を示す．基本 AIR アルゴリズムでは，nogood の重要度を導入していないため，無駄な閾値に関する探索を行い，かつ，解が得られない場合は，それまでに得られた nogood を廃棄する必要があるため，以前の計算の結果を利用することができない．図から分かるように，nogood の依存関係を導入した AIR アルゴリズムは，基本 AIR アルゴリズムと比較して，必要とされるステージ数は 1/5 程度に減少している．

図5に，過制約である分散 12-queens 問題における，前述の二つのアルゴリズムにおける閾値の，ステージの増加における変化の様子を示す．閾値の初期値は 0 であり，ステージが進むにつれて段階的に増加して閾値＝1/72≈0.014 となり，解が得られる．nogood の依存関係を導入した AIR アルゴリズムでは，基本 AIR アルゴリズムと比較して無駄な閾値に関する探索を行わないことが示されている．また，以前の計算の結果を有効利用することにより，各閾値での探索に要する時間が削減されていることが示されている．例えば，閾値＝1/162≈0.0062 に関する探索は，nogood の依存関係を導入した AIR ではわずかに 23 ステージで終了しているが，基本 AIR では 249 ステージが必要とされる．

6. 考　　察

本論文では，制約緩和における解の条件を，より重要な制約をより多く満たす解として定義したが，本論文で提案されたアルゴリズムは，より一般的な解の条件においても適用可能である．本アルゴリズムは，各エージェントの持つ局所的な評価関数 F_i に関して，これらの局所的な評価関数の最大値（最悪値）の最小化を行っている（$\max_{i=1,\ldots,m} F_i(S)$ を最小化する解 S を求めている）．F_i の定義自体は本アルゴリズムの本質的な部分と無関係であり，F_i として任意の関数を用いることができる．F_i の定義を変更することにより，本アルゴリズムを他の解の条件に用いることが可能となる．例えば，F_i をエージェント i の関係する制約条件違反の個数と定義すれば，本アルゴリズムは各エージェントに関する制約条件違反の個数の最大値を最小化する解を与える．

7. おわりに

本論文では，与えられた制約が強過ぎて解が存在しない分散制約充足問題において，制約の重要度を用いて制約を緩和する基準を定式化し，非同期バックトラッキングアルゴリズムに拡張を加えることにより，制約を最大限に満たす解を求める非同期段階緩和アルゴリズムが与えられることを示した．本アルゴリズムでは，逐次的な通常のバックトラッキングアルゴリズムとは対照的に，エージェントは非同期に並行して動作するが，アルゴリズムの完全性，解の最適性は保証される．また，本アルゴリズムではエージェント間で通信される制約条件違反に関する情報（nogood）に，その制約条件違反が成立する条件を付加することにより，無駄な計算が排除され，例題において 5 倍程度の速度向上が得られることを実験結果を用いて示した．

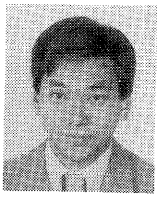
今後の研究課題として，本アルゴリズムを通信ネットワークにおける資源割当問題等の大規模な現実の問題に適用することが挙げられる．

謝辞 本研究の機会を与えて下さったNTTコミュニケーション科学研究所の河岡 司 所長，大里 延康 グループリーダに感謝いたします．また，草稿の段階で有益なコメントを頂いた京都大学の石田 亨 教授に感謝いたします．

参 考 文 献

1) Conry, S. E., Kuwabara, K., Lesser, V. R. and Meyer, R. A.: Multistage Negotiation for Distributed Satisfaction, *IEEE Trans. Syst. Man Cybern.*, Vol. 21, No. 6, pp. 1462 1477 (1991).
2) Descotte, Y. and Latombe, J. C.: Making Compromises among Antagonistic Constraints in Planner, *Artif. Intell.*, Vol. 27, No. 1, pp. 183-217 (1985).
3) Freeman-Benson, B. N., Maloney, J. and Borning, A.: An Incremental Constraint Solver, *Comm. ACM*, Vol. 33, No. 1, pp. 54-62 (1990).
4) Fruder, E. C.: Partial Constraint Satisfaction, *Proc. of IJCAI-89*, pp. 278-283 (1989).
5) Huhns, M. N. and Bridgeland, D. M.: Multiagent Truth Maintenance, *IEEE Trans. Syst. Man Cybern.*, Vol. 21, No. 6, pp. 1437-1445 (1991).
6) Chandy, K. M. and Lamport, L.: Distributed Snapshots: Determining Global States of Distributed Systems, *ACM Trans. Comput. Syst.*, Vol. 3, No. 1, pp. 63-75 (1985).
7) Pearl, J.: *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley (1984).
8) Sycara, K. P., Roth, S., Sadeh, N. and Fox, M.: Distributed Constrained Heuristic Search *IEEE Trans. Syst. Man Cybern.*, Vol. 21, No. 6, pp. 1446-1461 (1991).
9) Yokoo, M., Ishida, T. and Kuwabara, K.: Distributed Constraint Satisfaction for DAI Problems, Huhns, M. ed., *Proc. of 10th Workshop on Distributed Artificial Intelligence*, Chapter 18 (1990).
10) Yokoo, M., Durfee, E., Ishida, T. and Kuwabara, K.: Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving, *Proc. of 12th IEEE International Conference on Distributed Computing Systems*, pp. 614-621 (1992).

(平成4年11月13日受付)
(平成6年11月17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研究所に勤務．分散 AI，制約充足問題に関する研究に徒事．分散探索，分散制約充足問題等に興味を持つ．1992年人工知能学会論文賞．ソフトウェア科学会，人工知能学会，AAAI 各会員．