分散制約充足問題に関する研究

横尾　真

(1)

分散制約充足問題に関する研究

横尾 真

序文

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．本論文では，分散協調問題解決の様々な応用問題を定式化する一般的な枠組である分散制約充足問題に関して，筆者がこれまでに行ってきた研究について述べる．

第１章ではまず研究の目的を明確化する．従来の分散協調問題解決の研究では，典型的な応用問題を対象として，問題を解くために必要とされるエージェントおよびエージェント間の相互作用のモデル化を行う実験的なアプローチが中心であり，対象となる問題自体の定式化に関しては十分な考察がなされていなかった．様々な応用問題を表現可能な一般的な枠組を与えることは，領域に依存する異なる仮定のもとに研究されてきた種々のアプローチを比較するため，また，ある領域で得られた結果を，異なる領域の問題で再現するために重要である．本研究の目的は，制約充足問題を分散環境に拡張することにより，分散協調問題解決で議論されてきた様々な問題を定式化する枠組を与え，これらの問題を解く一般的な解法を提供することである．

第２章では，分散制約充足問題の基礎となる制約充足問題に関して，問題の定義と，従来提案されてきた制約充足問題の解法について概観する．さらに，制約充足問題の基本的な解法であるバックトラック型の探索アルゴリズムを改良し高速化した，弱コミットメント探索アルゴリズムを示す．

第３章では，分散制約充足問題の定義を示し，この分散制約充足問題によって，分散資源割当問題，分散解釈問題，分散知識ベースの整合性管理等の様々な分散協調問題解決の応用問題が定式化されることを示す．

第４章，第５章では分散制約充足問題の解法を示す．第４章では解を求める

ための分散探索アルゴリズムを示す．通信 / 制御のボトルネックの回避，処理の並列性の追求，セキュリティ，プライバシーの保持のためには，探索において，情報を一つのエージェントに集中せず，各エージェントが全体的な制御なしに，非同期，並行的に動作できることが望ましい．しかしながら，全体的な制御なしでは一般にアルゴリズムの完全性を保証することが困難となる．本論文では，各エージェントが全体的な制御なしに，非同期，並行的に動作しながらも，アルゴリズムの完全性が保証されることを特徴とする非同期バックトラッキングアルゴリズムを示す．また，非同期バックトラッキングアルゴリズムをベースとして，変数の優先順位を動的に変更することにより，第２章で示した弱コミットメント探索アルゴリズムと同様な性質を持つ非同期弱コミットメント探索アルゴリズムが実現されることを示す．

第５章では探索の効率を高めるための前処理である分散 consistency アルゴリズムを示す．分散 consistency アルゴリズムにおいては，エージェントはあらかじめ変数の領域に関する情報を交換し合い，他の変数との制約により解になり得ない値を取り除くことにより，無駄な探索を減少させる．仮説を用いた真偽値管理システム (Assumption-based Truth Maintenance System, ATMS) を分散化した分散 ATMS を用いることにより，分散 consistency アルゴリズムの実装が可能であることを示す．

現実の問題を分散制約充足問題として定式化した場合，与えられた制約が強過ぎて解が存在しない場合が多く存在する．第６章では，そのような場合に解の定義を拡張し，部分的に制約を満たす不完全な解を求めることについて議論する．具体的には，制約の重要度という概念を導入することにより，部分的に制約を満たす解の評価基準を定式化する．また，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することにより，制約の重要度により定義された評価基準において最適な解を求める非同期段階緩和アルゴリズムを示す．

第７章では，結論として以上についての成果のまとめを行う．

目次

第 1 章

序論

1.1 研究の背景および目的

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る. ごく近い将来には, このような分散計算機環境は個々の家庭にまで普及し, 全世界の計算機が接続された巨大な分散計算機環境が出現することが予想される. 現在のところ, これらの分散計算機環境で動作するプログラム群は, 人手によりかろうじて整合性が保たれている状況であるが, このような巨大な分散計算機環境の複雑性, 多様性に対応するためには, 将来的にはそれぞれのプログラムが自律的に動作し, 他のプログラムと協調しながら問題に対処する技術が不可欠となるだろう.

分散協調問題解決とは, 複数の人間による協調的な問題解決をモデルとし, 複数の人工的な知的エージェントによる問題解決を対象とする人工知能の研究の一分野である. 分散協調問題解決は古くから人工知能の研究者の興味を集めており, 人工知能の分野の中では比較的長い歴史を持っている. 例えば, 分散協調問題解決の研究者の議論の場である国際分散人工知能ワークショップの第一回目は 1980 年に開催されている.

従来の分散協調問題解決の研究は応用志向のものが中心であり, 典型的な応用問題を対象として, 問題を解くために必要とされるエージェントおよびエージェント間の相互作用のモデル化を行い, 実験する中から新たな理論を導くと

いう方向で研究が進められてきた．代表的なモデル化の方法として，結果共有とタスク共有という二種類の協調のモデルが提案されている．これらのモデルはそれぞれ，典型的な応用問題である解釈型の問題と割当型の問題に関する研究から生まれたものであると考えられる．以下，それぞれについて説明する．

解釈型の問題

解釈型の問題とはセンサデータ等をより高レベルの表現に変換する問題であり，音声認識システムである分散型 Hearsay-II [Lesser & Erman 1980] や，地域的に分散されたセンサネットワークによる乗物の軌跡の認識システムである Distributed Vehicle Monitoring Testbed (DVMT) [Lesser & Corkill 1983] 等が知られている．これらの問題では，各エージェントは全体の観測データの一部分のみを持ち，エージェント全体として整合する解釈を得る必要がある．結果共有とは，このような互いに関連する副問題を解いているエージェントが，解の途中結果を適宜交換し合う協調の方式である．エージェントは解釈の途中結果を交換しあうことにより，矛盾を引き起こすような誤った解釈を刈り込み，最終的に全体として整合のとれた解釈を得る．このような結果共有型の協調を行うためには，各エージェントは曖昧なデータを扱い，複数の可能性 (仮説) を同時に管理する必要がある．[Lesser & Corkill 1983; Lesser & Erman 1980] では，このような推論をサポートするために，黒板モデルが用いられている．また，[Mason & Johnson 1989] では，各エージェントが仮説に基づく真偽値管理システム (Assumption-based Truth Maintenance System) を用いて，複数の可能性を管理する方法が提案されている．

結果共有の効率は，各エージェントが全体として共通の解釈に効率良く向かうかどうか，すなわち全体としての統一性 (coherency) が達成できるかどうかに大きく依存する．Partial Global Plan (PGP) [Durfee & Lesser 1987] は 全体としての統一性を妥当な通信コストで実現するために提案されたものであり，各エージェントは問題解決のプランを相互に通信しあう．各エージェントは他のエージェントのプランを把握し，全体

として統一が取れるように自分のプランの調整を行う.

割当型の問題

一方, 割当型の問題とは, タスクや資源を複数のエージェントに割り当てる問題であり, 分散センサネットワーク [Smith & Davis 1981] におけるタスクの割当問題, LAN で接続された計算機ネットワークにおける負荷分散 [Malone *et al.* 1988], 通信ネットワークにおける資源割当問題 [Conry *et al.* 1991] 等が知られている. また, 分散航空管制問題 [Thorndyke, McArthur, & Cammarata 1981] や複数ロボットのプランニング問題 [Georgeff 1983] 等も, 空間的, 時間的な資源を複数エージェントに制約を満たすように割り当てる問題であると考えることができる.

タスク共有とは, 割当問題において複数のエージェントが一つのタスクを共有し, このタスクをサブタスクに分割し分担して問題解決を行う協調の形式である. タスクを分割して複数のエージェントに割当てるためのプロトコルとして, コントラクトネットプロトコル [Smith 1980] が提案されている. コントラクトネットプロトコルは, 人間社会の契約プロセスをモデル化したものであり, まず契約の提案側 (マネージャ) がタスクを提示する. それに対して, 契約の請負側 (コントラクタ) が適当なタスクに対して入札を行い, 提案側が入札から最も望ましいものを選択 (落札) する. このプロトコルでは請負側は入札時に, 提案側は落札時に, それぞれ各自の評価基準で選択を行っていることが特徴であり, このしくみは相互選択 (mutual selection) と呼ばれている. コントラクトネットプロトコルは, 分散センサネットワーク [Davis & Smith 1983], LAN で接続された計算機ネットワークにおける負荷分散 [Malone *et al.* 1988] に用いられている.

マルチステージネゴシエーション [Conry *et al.* 1991] はコントラクトネットプロトコルを発展させたものであり, エージェントの結ぶ複数の契約の間に制約が存在する場合に, 制約を満足する契約方法を発見する

ためのプロトコルである．制約が存在する場合には，一回の入札，落札では契約方法を決定できず，制約が満足されるまでネゴシエーションのステージが繰り返される．マルチステージネゴシエーションは通信ネットワークにおける資源割当問題 [Conry *et al.* 1991] に適用されている．

このように，従来の分散協調問題解決の歴史を振り返ってみると，協調的な問題解決が必要とされる先端的な応用問題を足掛かりとして，協調の様々な側面を浮き彫りにしたパイオニア的な優れた研究が多く存在するが，一つの問題点として，全体に特定の応用領域を対象に研究が行われており，実際に解かれている問題の定式化が不十分で，得られた結果の一般性に乏しいことが挙げられる．

ある分野の研究が成熟すれば，具体的な事例で得られた様々な知見が，より一般的な枠組に整理，統合され，精緻な理論となっていくのが必然であり，分散協調問題解決も例外ではあり得ない．一方，人工知能の研究の歴史を振り返ってみると，人工知能で扱われる問題のほとんどすべては探索という要素を含んでおり，探索問題 [Pearl 1984] という枠組が大きな役割を果たしている．探索問題は，初期状態から目標状態に至るオペレータの系列を発見する問題であり，不必要な詳細を省いた抽象化された問題として定式化されている．また，複数の変数に，変数間の制約を満足するように値を割り当てる制約充足問題 [Mackworth 1992] は探索問題の重要なサブクラスであり，これまでに多くの理論的，実験的研究がなされている．このような一般的な枠組は，領域に依存する異なる仮定のもとに研究されてきた様々なアプローチを比較するため，また，ある領域で得られた結果を異なる領域で再現するために重要である．さらに，このような枠組において一般的に適用できる技術，アルゴリズムは，多くの応用問題において一般的に利用可能な重要な基盤技術となる．分散協調問題解決においても同様な一般的な枠組が渇望されている．

本論文では，これらの状況をふまえ，分散協調問題解決における様々な応用問題を定式化可能な枠組を提案し，その枠組において一般的に適用可能なアルゴリズムを開発することを目的とする．本論文では，制約充足問題を分散環境

に拡張した分散制約充足問題を定義する．分散制約充足問題とは，制約充足問題の構成要素である変数，制約が複数のエージェントに分散された問題であり，従来の分散協調問題解決の分野で，これまで独立の問題として扱われてきた複数知識ベース間の整合性管理問題，分散資源割当問題，分散解釈問題等を定式化することが可能である．さらに，分散制約充足問題を解く種々のアルゴリズムの開発を行う．これらのアルゴリズムは，基本的には集中型の制約充足問題を解くアルゴリズムを分散化したものであり，解を求めるための探索アルゴリズムと，探索に先立つ前処理である consistency アルゴリズムに大別される．従来の集中型の制約充足問題の解法は，問題に関するすべての情報を持っている単独のエージェントにより逐次的に処理が行われることを仮定している．これに対して，本論文で提案する分散制約充足問題の解法は，問題に関して部分的な知識のみを持つ複数のエージェントが非同期に並行して処理を行う．

1.2 論文の構成

本論文は，序論と結論を含めて全体を 7 章で構成している．第 1 章の序論に続いて，第 2 章では分散制約充足問題の基礎となる制約充足問題に関して，問題の定義と従来提案されてきた制約充足問題の解法について概観する．さらに，制約充足問題の基本的な解法であるバックトラック型の探索アルゴリズムを改良し高速化した，弱コミットメント探索アルゴリズムを示す．弱コミットメント探索アルゴリズムにより，従来の探索アルゴリズムと比較して 3 倍から 10 倍程度の高速化が得られることを例題を用いた実験結果，確率モデルを用いた評価により示す．

第 3 章では，分散制約充足問題の定義を示し，この問題によって，分散資源割当問題，分散解釈問題，複数知識ベース間の整合性管理等の様々な分散協調問題解決の応用問題が定式化されることを示す．

第 4 章，第 5 章では分散制約充足問題の解法を示す．第 4 章では解を求めるための分散探索アルゴリズムを示す．通信 / 制御のボトルネックの回避，処理の並列性の追求，セキュリティ，プライバシーの保持のためには，探索において

各エージェントが全体的な制御なしに, 非同期, 並行的に動作できることが望ましい. しかしながら, 全体的な制御なしでは一般にアルゴリズムの完全性を保証することが困難となる. 本論文では, 各エージェントが全体的な制御なしに, 非同期, 並行的に動作しながらも, アルゴリズムの完全性が保証されることを特徴とする非同期バックトラッキングアルゴリズムを示す.

また, 非同期バックトラッキングアルゴリズムをベースとして, 変数の優先順位を動的に変更することにより, 第2章で示した弱コミットメント探索アルゴリズムと同様な性質を持つ, 非同期弱コミットメント探索アルゴリズムが実現されることを示す.

第5章では探索の効率を高めるための前処理である分散 consistency アルゴリズムを示す. 仮説を用いた真偽値管理システムである Assumption-based Truth Maintenance System (ATMS) を分散化した分散 ATMS を用いることにより, 分散 consistency アルゴリズムが実装可能であることを示す.

現実の問題を分散制約充足問題として定式化した場合, 与えられた制約が強過ぎて解が存在しない場合が多く存在する. 第6章では, そのような場合に解の定義を拡張し, 部分的に制約を満たす不完全な解を求めることについて議論する. 制約の重要度という概念を導入することにより, 部分的に制約を満たす解の評価基準を定式化する. また, 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することにより, 重要な制約をできる限り多く満足する最適な解を求める非同期段階緩和アルゴリズムが実現されることを示す.

第7章では, 結論として以上についての成果のまとめを行う.

第 2 章

制約充足問題

2.1 序言

制約充足問題は人工知能の様々な問題を定式化可能な枠組であり，これまでに多くの理論的，実験的な研究が行われて来ている．本章ではまず，制約充足問題の定義を示し，従来の制約充足問題の解法を簡単に紹介する．これらの解法は，探索型の解法と，探索に先立つ前処理である consistency アルゴリズムに大別される．また，探索型の解法は，縦形の系統的な探索アルゴリズムであるバックトラッキングアルゴリズムと，山登り型の探索アルゴリズムである反復改善型アルゴリズムに分類される．本章ではバックトラッキングアルゴリズムと反復改善型の長所を合わせ持った新しい探索アルゴリズムである弱コミットメント探索アルゴリズムを示し，実験結果および確率モデルを用いて，弱コミットメント探索アルゴリズムの有効性を示す．

2.2 問題の定義

制約充足問題 (Constraint Satisfaction Problem) は一般に次のように定義される．n 個の変数 $x_1, x_2, ..., x_n$ と，変数のそれぞれが値を取る有限で離散的な領域 $D_1, D_2, ..., D_n$，および制約の集合が存在する．本論文では，制約の表現方法として次の 2 種類の両方を用いる．

1. 述語によって内包的に表現される．例えば，2 変数 x_i, x_j 間の制約述語 p_{ij} は，直積 $D_i \times D_j$ 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．

2. 許されない値の組合せにより，外延的に表現される[1]．変数 x_i の値が d_i であることを，(x_i, d_i) のように二項組で表すことにする．
制約 $\{(x_i, d_i), (x_j, d_j)\}$ は，$x_i = d_i$ と $x_j = d_j$ が同時に成立できないことを意味する．

これらの表現は互いに交換可能であり，どちらを用いるかは本質的な問題ではない．2 番めの許されない値の組合せで表現された制約を *nogood* と呼ぶ．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，すなわち，すべての制約述語が真となり，かつ，どの nogood に関してもそのスーパセットとならない値の組合せを求めることである．制約充足問題の定義はこのように非常に簡単なものであるが，驚くほどに多くの人工知能で扱われている問題が，この制約充足問題として表現可能である．

制約充足問題の典型的な例題として，n-queens と呼ばれるパズルがよく用いられる (図 2.1)．この例題は，n×n のチェスの盤面に n 個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る．各列には一つのクイーンしか配置できないことは自明であるので，各列のクイーンの位置に対応する変数が存在すると考えれば，図 2.1の 4-queens 問題は，$x_1, \ldots, x_4$ の 4 つの変数に，{1,2,3,4} のいずれかの値を割り当てる問題として表現できる．変数 x_i, x_j 間の制約述語 p_{ij} は $(x_i \neq x_j) \wedge (\mid i - j \mid \neq \mid x_i - x_j \mid)$ として表現される．

もう一つのよく用いられる例題としてグラフの色塗り問題がある (図 2.2)．この問題はノードがリンクで連結されたグラフにおいて，k 個の色を用いて，連結されたノードが異なる色を持つように塗り分ける問題である．この問題は，各ノードに対応する変数が存在し，変数の値が色に対応すると考えれば制約充足

[1]通常の制約充足問題の定義では，制約は変数の許される値の組合せで表現されるのが通例であるが，本論文では探索途中で得られた新しい制約を統一的に扱うのに便利であるため，許されない値の組合せによる表現を用いる．どちらの表現を用いるかは本質的な問題ではない．

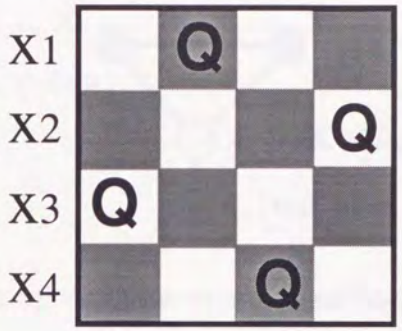


図 2.1: 制約充足問題の例 (4-queens 問題)

問題としてマッピング可能である.

2.3 従来の制約充足問題の解法

2.3.1 探索型の解法

バックトラッキングアルゴリズム

バックトラッキングアルゴリズム [Mackworth 1992] は系統的な縦型の探索アルゴリズムであり, 変数に対して適当な順序で逐次的に値を決定していく. 決定された変数の値の組合せを部分解と呼ぶ. バックトラッキングアルゴリズムは, 部分解に変数を一つ一つ加え, 最終的に完全な解が得られるまで部分解を拡張していく. 部分解に新しい変数を加える際には, 変数の値を部分解との間の制約を満足する値に設定する. そのような値が存在しない場合には, 最も新しく部分解に追加された変数の値を変更する (バックトラック).

単純なバックトラッキングアルゴリズムでは非常に多くの無駄なバックトラック (thrashing と呼ばれる) が生じ, 効率が悪い. thrashing を減少させるため, 後述する consistency アルゴリズムのように探索に先立つ前処理を行う方法, ま

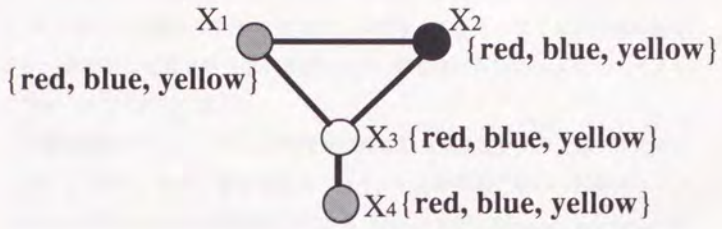


図 2.2: 制約充足問題の例 (グラフの色塗り問題)

た, バックトラッキングアルゴリズムでの部分解に付け加える変数の選択の順序, 変数の値の選択順序をヒューリスティックスを用いて決定する方法 [Dechter & Pearl 1988; Haralick & Elliot 1980; Nudel 1983] 等が提案されている.

これらのヒューリスティックスの中で, 制約充足問題の一つの解を見つける際に特に有効なものとして, 制約違反最少化 (min-conflict) ヒューリスティック [Minton *et al.* 1992] が知られている. このヒューリスティックは, 変数の値を決める際に, 他の変数に与えられた値との間の制約条件違反の個数を最少化するものを選択するというものである. 具体的には, 制約違反最少化ヒューリスティックを用いたバックトラッキングアルゴリズム (制約違反最少化バックトラッキング) では, 変数のすべてに暫定的な初期値が与えられる. 部分解に付け加える変数を選択する際には, 他の変数と制約条件違反を起こしている変数が選択される (暫定的な初期値がすべての制約を満足している変数の値は変更する必要がない). また, 部分解に付け加える変数の値を決定する際に, 部分解との間のすべての制約を満たす値の中から, 部分解に含まれていない変数の暫定的な初期値との間の制約条件違反を最少化するものが選択される.

反復改善型アルゴリズム

反復改善型アルゴリズム [Minton *et al.* 1992; Morris 1993; Selman, Levesque, & Mitchell 1992; Sosič & Gu 1994] は 山登り型の探索アルゴリズムであり,

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．局所的な変更を行う際の基準として，制約違反最少化ヒューリスティックを用いることができる．

反復改善型のアルゴリズムは局所最適解に陥る可能性があり，局所最適解から逃れるために，制約の重みを変更したり，一定時間後に新しい初期値からリスタートする等の方法が提案されている [Morris 1993; Selman, Levesque, & Mitchell 1992].

2.3.2 consistency アルゴリズム

consistency アルゴリズム [Mackworth 1992] は探索に先立つ前処理であり，問題で陽に与えられた制約から間接的に導かれる新しい制約をあらかじめ導き出しておくことにより，最終的な解になり得ないような変数の値を取り除き，thrashing を避けようとするものである．consistency アルゴリズムは k-consistency という概念によって分類される．ある制約充足問題が k-consistent であることの定義は次の通りである．

- 制約を満たす k-1 個の変数の値の組が与えられた場合に，この値の組に対して，任意の k 番目の変数に関して，この値の組と制約を満足する値が存在する．

また，問題が k-consistent であると同時にすべての $j \leq k$ に関して j-consistent である場合に，問題は強 k-consistent であるという．k-consistency アルゴリズムは，与えられた問題を，それと等価な (同じ解を持つ) 強 k-consistent な問題に変換する．明らかに，n 個の変数からなる制約充足問題が強 n-consistent であれば，バックトラックなしで解を得ることが可能となる．しかしながら，強 n-consistency を達成するコストは，ほとんどの場合無駄なバックトラックのコストを上回る [Mackworth 1992]．よって，制約充足問題を解くに当たっての課題は，前処理とバックトラッキングの最適な組合せを発見することであり，様々な

組合せに関する評価がなされている [Dechter & Meiri 1989; Haralick & Elliot 1980; Nudel 1983].

近年, 各種の consistency アルゴリズムは, 仮説を用いた真偽値管理システムである Assumption-based Truth Maintenance System (ATMS) の枠組で説明できることが [de Kleer 1989] により指摘されている. ATMS では, 制約はすべて nogood で表されており, hyper-resolution と呼ばれる操作により, 複数の nogood から新しい nogood を生成する. 変数 x_1 のとり得る値が 1 か 2 のどちらかである場合, $(x_1 = 1) \vee (x_1 = 2)$ が成立する. 変数 x_1 と x_2 の間に, $x_1 < x_2$ なる制約があるとすると, この制約は $\neg(x_1 = 1 \wedge x_2 = 0)$, $\neg(x_1 = 2 \wedge x_2 = 0)$ 等の nogood で表現される. nogood を構成する変数の値の数を nogood の長さと呼ぶ. これらの nogood の長さは 2 である. hyper-resolution とは, これらの nogood と $x_1 = 1 \vee x_1 = 2$ を組み合わせて $\neg(x_2 = 0)$ を導き出す操作であり, 一般には次のように記述される (A_i は $x_1 = 1$ のような命題である).

$$\frac{\begin{array}{l} A_1 \vee \ldots \vee A_n \\ \neg(A_1 \wedge A_{11} \ldots \wedge A_{1m}), \\ \vdots \\ \neg(A_n \wedge A_{n1} \ldots \wedge A_{nm}) \end{array}}{\neg(A_{11} \ldots \wedge A_{1m} \wedge \ldots \wedge A_{n1} \ldots \wedge A_{nm})}$$

hyper-resolution は非常に多くの新しい nogood を生成するため, 適用を制限することが現実的である. 生成する nogood の長さに制限を加えて, 長さが k より小さい nogood を生成する hyper-resolution のみを実行することにすると, 強 k-consistency が達成される.

2.4 弱コミットメント探索アルゴリズム

2.4.1 問題の所在

2.3.1節で説明したバックトラッキングアルゴリズムには以下のような問題点がある.

一回の値の決定の失敗が致命的である: バックトラック型の探索アルゴリズムでは, 探索途中で構成された部分解は, 解の一部分とならないことが判明しない限り修正されない. 解の一部になり得ないような部分解を構成した場合, その部分解を含む解が存在しないことを示すためには, その部分解に関する網羅的な探索が必要となる. 問題が大規模な場合は, そのような網羅的な探索を行うことは事実上不可能である.

一方, 2.3.1節で説明したように, 反復改善型アルゴリズムは, バックトラッキングアルゴリズムと対照的に, 制約をすべて満たす部分解の構成を行わない. よって, 反復改善型のアルゴリズムでは, 網羅的な探索を行わずに同じ変数の値を繰り返し変更することが可能であるため, 一回の値の設定のミスは致命的ではなく, ミスを修正することが可能である. しかしながら, 反復改善型アルゴリズムには次のような問題点がある.

アルゴリズムの完全性が保証されない: ここでは, アルゴリズムの完全性とは, 解が存在する場合には必ず解を求め, 解が存在しない場合には, 解が存在しないことを発見して有限時間でアルゴリズムが終了することを意味する. 例外として, [Morris 1993] に示されている fill アルゴリズムは解が存在する場合には必ず解が得られることが保証されている. しかしながら, このアルゴリズムは類似の完全性が保証されないアルゴリズム (breakout アルゴリズム) よりもはるかに非効率的であることが示されている [Morris 1993]. また, fill アルゴリズムでは, 解が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はできない.

他のヒューリスティックスの導入が困難である: アルゴリズムの完全性は, 大規模な問題を解く場合には理論的な意味しかなく, 現実には問題とならないかも知れない. より現実的な問題点としては, これまでの制約充足問題の長い歴史を持つ研究で得られてきた有用なテクニック, ヒューリスティックスの多くが [Haralick & Elliot 1980; Nudel 1983], 部分解を構成することを前提としたものであり, 反復改善型のアルゴリズムに適用することが難しいことがある.

2.4.2 基本的なアイデア

本節では以下, 部分解に弱くコミットする, 制約違反最少化ヒューリスティックを用いた探索アルゴリズム (弱コミットメント探索アルゴリズム) を提案する. このアルゴリズムはバックトラック型の探索アルゴリズムと, 反復改善型の探索アルゴリズムの中間的な性質を持ち, それぞれの欠点が取り除かれている.

このアルゴリズムの特徴は, 部分解に強くコミットする (解になり得ないことが示されない限り部分解の変更を行わない) バックトラッキングアルゴリズムと対照的に, 部分解に弱くコミットする点である. すなわち, このアルゴリズムでは, 部分解が成長している間は, その部分解を変更しないが, ある変数に関して部分解と制約を満たす値が存在しない場合には, 現在の値の割当を新しい初期値として部分解の構成を最初からやり直す.

このアルゴリズムの利点は次の通りである.

- 値の設定を誤って悪い部分解を構成した場合は, その部分解に関する網羅的な探索を行わずにやり直しが行われるため, 一回のミスが致命的とならない.
- やり直しが生じた際の部分解を記録し, そのような部分解を再び構成しないようにすることにより, アルゴリズムの完全性 (解が存在するなら必ず解を見つけ, 解が存在しないなら, 解が存在しないことを検出すること) が保証される.

表 2.1: アルゴリズムの性質の比較

	weak-commitment	min-conflict BT	iterative-improvement
construct partial solution	yes	yes	no
avoid strong commitment	yes	no	yes
always find a solution if exists	yes	yes	no (except fill algorithm)
stop if no solution exists	yes	yes	no

- 制約を満足する部分解を構成するため, 様々なヒューリスティックスを導入することが可能である.

制約違反最少化バックトラッキングアルゴリズム, 反復改善型アルゴリズム, 弱コミットメント探索アルゴリズムの性質を表2.1にまとめる.

本節では以下, 弱コミットメント探索アルゴリズムの詳細な説明を行う. さらに, 例題に関する実験結果, 確率モデルを用いた評価により, 弱コミットメント探索アルゴリズムの有効性を示す.

2.4.3 アルゴリズムの詳細

図2.3に弱コミットメント探索アルゴリズムを示す. ここでは, すべての制約はnogoodで表現されていることを仮定する. 初期状態では, *vars-left* は, すべての変数と初期値のペアのリスト $\{(x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n)\}$ であり, *partial-solution* は空集合である. このアルゴリズムは変数を一つ一つ *vars-left* から *partial-solution* に移動する.

このアルゴリズムと制約違反最少化バックトラッキングアルゴリズムとの差は図2.3の網掛けで示した部分であり, バックトラッキングアルゴリズムであれ

ばバックトラックが生じ, 部分解中の最も新しく付け加えられた変数の値のみが取り除かれる時点で, 弱コミットメント探索アルゴリズムでは現在の値の割当を新しい初期値として, 部分解の構築が改めてやり直される. 注意すべきことは, やり直しが生じた時点の部分解に含まれる変数は, そのすべてに関して値が変更されるのではない点である. 他の変数すべてと制約を満たしている変数は, 値の変更の対象とならないため, 部分解に含まれている変数の内, 他の変数と制約条件違反が生じている変数のみが変更の対象となる.

このアルゴリズムは, *nogood-list* にやり直しが生じた際の部分解を新しい制約として追加し, そのような部分解を再び構成しないため, アルゴリズムの完全性 (解が存在するなら必ず解を見つけ, 解が存在しないなら, 解が存在しないことを検出すること) が保証される.

バックトラック型のアルゴリズムをベースとして, 部分解に強くコミットせず, 網羅的な探索を行わずに他の可能性を探索するアルゴリズムとして Iterative Broadening アルゴリズム [Ginsberg & Harvey 1990] が提案されている. 弱コミットメント探索アルゴリズムは, Iterative Broadening アルゴリズムで探索の幅が 1 の場合に近いが, Iterative Broadening アルゴリズムでは段階的に探索の幅を広げることによりアルゴリズムの完全性を保証しているのに対し, 弱コミットメント探索アルゴリズムでは探索の幅は 1 のまま変更されず, やり直しが生じた部分解を新しい制約として記録することにより, アルゴリズムの完全性を保証している.

2.4.4 アルゴリズムの実行例

弱コミットメント探索アルゴリズムの実行例を図 2.4に示す. 例題は 4-queens 問題である. 初期状態の暫定的な変数の値は図 2.4 (a) に示す通りである. このアルゴリズムは, まず最初のクイーンの位置を変更し (図 2.4 (b)), 次に 4 番目のクイーンの位置を変更する (図 2.4 (c)). 位置が変更されたクイーンは *partial-solution* に付け加えられる. 図では *partial-solution* に付け加えられたクイーンを黒丸で表している. 図 2.4 (c) の状態では, 3 番目のクイーンに関して *partial-*

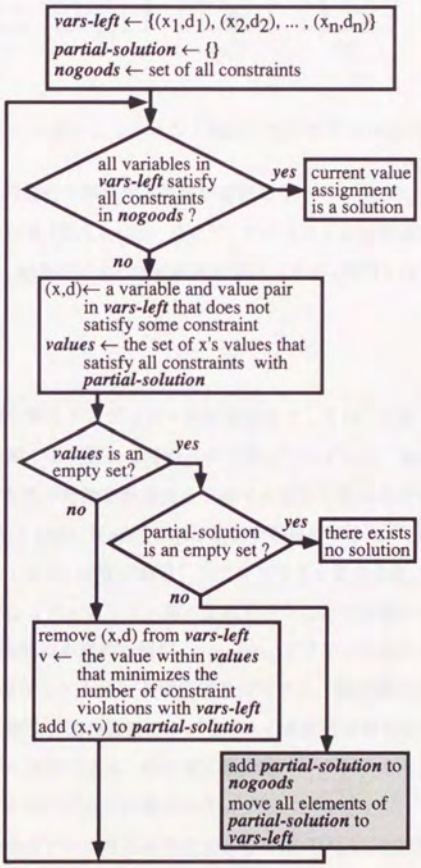


図 2.3: 弱コミットメント探索アルゴリズム

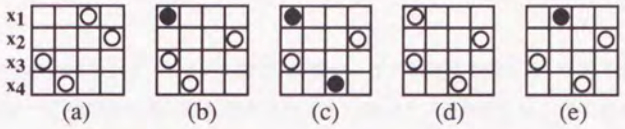


図 2.4: 弱コミットメント探索アルゴリズムの実行例

solution との間の制約を満足する位置が存在しない．このため，*partial-solution* はすべて放棄される (図 2.4 (d))．ここで，アルゴリズムは最初のクイーンの位置を再び変更し，結果的にすべての制約が満足される (図 2.4 (e))．

2.4.5 評価

制約充足問題を解くアルゴリズムの評価方法としては，従来，典型的な例題，ベンチマークに関する実験的な評価が多く用いられている．確率モデルを用いて，アルゴリズムの平均的な計算量を予測する方法も幾つか提案されているが [Haralick & Elliot 1980; Nudel 1983]，解析を容易にするため，非常に単純化したモデルを用いており，性質が類似したアルゴリズム間の比較は可能であるが，性質が大きく異なるアルゴリズム間の比較を行うことは困難である．本節では以下，制約充足問題の典型的な例題 (n-queens，グラフの色塗り問題，3-SAT 問題) に関して，弱コミットメント探索アルゴリズム，制約違反最少化バックトラッキングと反復改善型の探索アルゴリズムの実験的な評価を行う．また，確率モデルを用いた解析により，制約違反最少化バックトラッキングと弱コミットメント探索アルゴリズムの比較を行う．

反復改善型の探索アルゴリズムの代表としては，[Morris 1993] で示されている breakout アルゴリズムを用いる．このアルゴリズムは，局所最適解で止まらないという性質があり，制約違反最少化ヒューリスティックを用いる他の反復改善型のアルゴリズムよりも性能が優れているという実験結果が報告されている [Morris 1993]．しかしながら，このアルゴリズムは複数の局所最適解の間を

巡回する無限ループに陥る可能性があり，アルゴリズムの完全性は保証されない．

評価の基準として，サイクル数と，制約チェックの回数を用いる．サイクル数とは，一回の変数の値の選択およびバックトラック / やり直しを 1 サイクルとしてカウントしたものである．制約チェックの回数とは，制約が存在する一つの変数の値の組合せが制約を満足するかどうか (制約として表現されている許されない値の組合せかどうか) を調べることを一回のチェックとしてカウントしたものであり，計算機の性能に依存しないアルゴリズムの効率を示す評価基準として制約充足アルゴリズムの評価に広く用いられている[2]．弱コミットメント探索アルゴリズムと制約違反最少化バックトラッキングアルゴリズムは，各サイクルにおける処理の内容はほぼ同じであり，サイクル数での比較が可能であるが，これらのアルゴリズムと breakout アルゴリズムとでは，一回のサイクルに要する処理の重さが異なっており，制約チェック数での比較が適当である．制約チェックの回数の測定に当たっては，すべてのアルゴリズムに関して，前回のサイクルでの制約チェックの結果を保存して，差分のみを計算することによる効率化を行っている．

実験を妥当な時間内で終了させるため，各試行に関し，5000 サイクルを上限とし，これを越えた場合は処理を中断する．中断された試行に関しては，サイクル数は 5000 回として，制約チェック数は中断された時点での制約チェック数をカウントする．

n-queens

最初の例題は 2.2節で説明した n-queens 問題である．前述のように，この問題は，$n \times n$ のチェスの盤面に，n 個のクイーンを互いに取り合わないように配置するパズルであり，各列のクイーンの位置を変数とみなすことにより制約充足問題として定式化される．表 2.2に，n=10, 50, 100 の場合の，100 個の異なる

[2]新しく追加された制約 (やり直しが生じた部分解) に関するチェックも，この回数に含まれている．

初期値に関する，解が得られた試行の割合，サイクル数，制約チェック数の平均を示す[3]．$n > 100$ の結果は，ほとんどすべての試行に関してバックトラック / やり直しなしで解が得られているため割愛する．これらの試行では，変数の初期値は [Minton *et al.* 1992] に示されている greedy な方法により，比較的良い (最終的な解に近い) 初期値が与えられている．

表 2.2に示されているように，すべての場合において，弱コミットメント探索アルゴリズムは，制約違反最少化バックトラッキング，breakout アルゴリズムよりも効率的であり，制限時間内に終了しなかった試行は存在しない．また，弱コミットメント探索アルゴリズムと，breakout アルゴリズムを比較すると，サイクル数の差と比較して，制約チェック数の差が大きく，一サイクルに要する処理の重さがこれらのアルゴリズムで異なっていることが示されている．この理由は，値を変更する変数を選択する際に，弱コミットメント探索アルゴリズムでは，制約条件違反を生じている変数から任意のものを選べば良いのに対して，breakout アルゴリズムでは，制約条件違反の個数を減少させるものを選ばなくてはならず，最悪の場合（現在の値の割当が局所最適である場合）には，制約条件違反を生じているすべての変数に関して，制約条件違反の個数を減少させるものがあるかどうかのチェックが必要になるためである．

バックトラック / やり直しが生じない試行に関しては，弱コミットメント探索アルゴリズムと制約違反最少化バックトラッキングの動作は全く同じであり，2 つのアルゴリズムの差は，バックトラック / やり直しが生じた試行に関してのみ現れる．表 2.3に，バックトラック / やり直しが生じた試行の回数の割合，弱コミットメント探索アルゴリズムと制約違反最少化バックトラッキングのこれらの試行に関するサイクル数の平均を示す．表 2.3に示されるように，制約違反最少化バックトラッキングでは，バックトラックが生じた場合 (値の設定を

[3][Minton *et al.* 1992] で示されている 100-queens 問題に関する制約違反最少化バックトラッキングの評価結果では，25 サイクル程度で解が得られることが報告されている．我々の実験では，100 回の試行中，一回だけ制限時間内 (5000 サイクル以内) に解が得られない試行が存在し，この試行が全体の平均に関して支配的となっている．この試行を除いた平均は [Minton *et al.* 1992] の結果とほぼ一致する．

表 2.2: 弱コミットメント探索, 制約違反最少化バックトラッキング, breakout アルゴリズムの比較 (n-queens)

	weak-commitment			min-conflict BT			breakout		
n	ratio	cycles	checks	ratio	cycles	checks	ratio	cycles	checks
10	100%	29.7	2292.8	100%	240.7	15482.2	100%	41.7	7065.0
50	100%	23.6	48593.5	97%	264.2	300175.0	100%	37.9	180393.5
100	100%	27.1	236821.7	99%	76.4	912465.1	100%	38.9	777051.1

表 2.3: バックトラックを生じた試行のサイクル数 (n-queens)

n	ratio of trials with BT	weak-commitment	min-conflict BT
10	80%	35.9	299.7
50	17%	58.2	1473.4
100	2%	96.5	2563.5

ミスした場合) のサイクル数が非常に大きく, 全体の平均に関して支配的な要因となっている.

グラフの色塗り問題

グラフの色塗り問題は 2.2節で示したように, ノードがリンクで連結されたグラフにおいて, k 個の色を用いて, 連結されたノードが異なる色を持つように塗り分ける問題である. これに関しても, [Minton *et al.* 1992] に示されているのと同じ方法を用いて, n 個のノード, m 個のリンクからなる, 解が存在し, グラフが連結されているランダムに生成された問題を用いる. 各ノードの取り得る色の数 k は 3 色, 全体として $m = n \times 2$ 本のリンクが存在する問題に関して, $n = 120, 180, 240$ に関する評価を行う. このパラメータの設定は,

[Minton *et al.* 1992] で用いられた, 変数間の制約の密度が疎で, 制約違反最少化ヒューリスティックが効果的でない問題に対応する. 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). n-queens と同様, 初期値は greedy な方法で設定されている. グラフの色塗り問題に関する評価では, 制約違反最少化バックトラッキングアルゴリズムと弱コミットメント探索アルゴリズムには, *forward checking* と *first-fail principle*[Haralick & Elliot 1980] の 2 種類のヒューリスティックスが導入されている. すなわち, 各変数に関して, 部分解と制約を満たす値のリストを管理し, 値を変更する変数の選択において, 部分解と制約を満たす値の個数が最も少ないものから値の決定を行う. また, 制約を満たす値の個数が 1 つだけになった変数は, 直ちにその値に決定される.

表 2.4に 3 種類のアルゴリズムに関する評価結果を示す. [Minton *et al.* 1992] では, このパラメータ設定では制約違反最少化バックトラッキングが効率的でないことが示されているが, 本論文での評価では, *forward checking* と *first-fail principle* を用いることにより比較的良い結果が得られている. しかしながら, わずかではあるが, 制約違反最少化バックトラッキングで, 値の設定のミスが致命的となり, 制限時間内で解が得られない場合が生じ, 結果的に弱コミットメント探索アルゴリズムの方が効率的となっている. ヒューリスティックスの導入のため, 弱コミットメント探索, 制約違反最少化バックトラッキングのサイクル当たりの制約チェック数は増加し, breakout アルゴリズムとほぼ同等となっているが, サイクル数の削減の効果が大きく, 弱コミットメント探索アルゴリズムは breakout アルゴリズムよりも制約チェック数で 10 倍程度の高速化が得られている. このように, 部分解を構成することを前提とした強力なヒューリスティックスを導入できる点が, 反復改善型の探索アルゴリズムに対する弱コミットメント探索アルゴリズムの大きな利点である.

表 2.4: 弱コミットメント探索, 制約違反最少化バックトラッキング, breakout アルゴリズムの比較 (グラフの色塗り問題)

	weak-commitment			min-conflict BT			breakout		
n	ratio	cycles	checks	ratio	cycles	checks	ratio	cycles	checks
120	100%	28.9	2118.8	99%	78.1	2931.2	100%	198.4	14620.8
180	100%	41.3	3178.9	99%	98.5	5548.5	100%	352.3	32139.3
240	100%	71.9	5988.6	95%	443.57	42164.5	100%	601.2	66892.5

3-SAT 問題

3 番目の例題として, 山登り型の探索アルゴリズムのベンチマークとして広く用いられている 3-satisfiability problem (3-SAT problem) を取り上げる. この問題は n 個の変数に関して真偽値を割り当てる問題であり, 制約として 3 個の変数からなる複数の節が与えられる. これらの制約をすべて満足する変数の真偽値を見つけることが目的である. 制約 (節) の個数を変数の個数で割ったものが, 節の密度と呼ばれ, 問題の難しさにクリティカルに影響することが知られており, 特に節の密度が 4.3 の場合が難しい問題となることが知られている [Mitchell, Selman, & Levesque 1992].

表 2.5に, 節の密度を 4.3 に固定して, 変数の個数 n を変化させた場合の実験結果を示す. [Morris 1993] と同じ方法を用いて, 解が存在する問題をランダムに生成している. 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. 他の問題と同様, 初期値は greedy な方法で設定されている. グラフの色塗り問題と同様, 制約違反最少化バックトラッキング, 弱コミットメント探索アルゴリズムには *forward checking* と *first-fail principle* が導入されている.

表 2.5に示されるように, この問題では制約違反最少化バックトラッキングアルゴリズムが非常に非効率的であるのに対し, 弱コミットメント探索アルゴリ

表 2.5: 弱コミットメント探索, 制約違反最少化バックトラッキング, breakout アルゴリズムの比較 (3-SAT 問題)

	weak-commitment			min-conflict BT			breakout		
n	ratio	cycles	checks	ratio	cycles	checks	ratio	cycles	checks
300	100%	187.7	24357.0	55%	2717.7	1075986.1	100%	828.0	133911.5
500	100%	359.4	47376.0	15%	4428.8	2368672.5	93%	1596.2	352095.8
700	100%	633.2	83345.1	0%	—	—	77%	2740.2	746752.5
900	100%	980.3	132731.7	0%	—	—	70%	3006.8	1032267.3
1100	100%	1246.8	168845.9	0%	—	—	69%	3384.5	1454297.9

ズムは breakout アルゴリズムと比較して n が大きい範囲で 10 倍程度の高速化が得られている. この理由は, 制約違反最少化バックトラッキングでは, 導入されたヒューリスティックスが, バックトラックなしで解が得られる程度に強力でないと効果が現れないが, 弱コミットメント探索アルゴリズムでは段階的に変数の初期値が改善されるため, 弱いヒューリスティックスでも効果があるためであると考えられる.

確率モデルを用いた評価

弱コミットメント探索アルゴリズムの, 制約違反最少化バックトラッキングに対する優位性を説明するために, 次のような簡単なモデルを考える. すなわち, 任意の初期値からスタートして, 制約違反最少化バックトラッキングアルゴリズムで探索を行った場合, バックトラックなしで解が得られる確率 (一度もミスをしない確率) が, 変数の初期値によらず一定の値 p で与えられるとする. また, バックトラックなしで解が得られる場合の探索に要するサイクル数の期待値が n_s で与えられるとする ($n_s \leq n$). 一方, バックトラックを生じる場合, 解が得られるまでに要する探索のサイクル数の期待値を B, そのうち, 最初のバックトラックを生じるまでに要する探索のサイクル数の期待値を n_b ($n_b \leq$

n) とする. 制約違反最少化バックトラッキングアルゴリズムのサイクル数の期待値は, $n_s p + Bq$ で与えられる ($q = 1 - p$).

これに対して, 弱コミットメント探索アルゴリズムを用いて, 改めて部分解の構成を行う場合の探索に要するサイクル数の期待値は, やり直しなしで解が得られる確率 p でその場合のサイクル数は n_s, 1 回のやり直しで解が得られる確率は pq, サイクル数は $n_s + n_b$, 以下同様に, k 回のやり直しで解が得られる確率は pq^k, サイクル数は $n_s + kn_b$ で与えられる. やり直しの回数の分布は, 良く知られた確率 p の幾何分布となり, その期待値は q/p で与えられる. よって, サイクル数の期待値は $n_s + n_b q/p$ で与えられる.

弱コミットメント探索アルゴリズムが制約違反最少化バックトラッキングアルゴリズムより効率的である条件は, $n_s p + Bq > n_s + qn_b/p$ であり, これより $p > n_b/(B - n_s)$ が得られる. $B \gg n_s$ と考えて良く, また, $n_b \leq n$ を用いて, 弱コミットメント探索アルゴリズムがバックトラックアルゴリズムより効率的であるための十分条件として, $p > n/B$ が得られる. すなわち, バックトラックなしで解が得られる確率が, 変数の個数 n とバックトラックを生じる場合のサイクル数の比より大きければよい. これまでの評価結果, 例えば表 2.3から示されるように, 制約違反最少化バックトラッキングでは, 多くの場合にバックトラックなしで高速に解が得られるが, バックトラックが生じた場合のサイクル数は非常に大きくなっている. このため, この条件は多くの場合に満足されると考えられる.

実際には, 制約違反最少化ヒューリスティックを用いた場合, バックトラックなしで解が得られる確率は初期値に依存して変化する. 弱コミットメント探索アルゴリズムでは, 部分解の構成をやり直す際に, その時点での変数の値の割当が新しい初期値とされるため, やり直しを繰り返すことにより, 次第に変数の値は最終的な解に近付いていき, バックトラックなしで解が得られる確率は増加していくことが予想される. このような場合には, 前述の条件が成立しない場合, すなわち, 任意の初期値からスタートした際のバックトラックなしで解が得られる確率が非常に小さい場合でも, やり直しを繰り返すことによって初期

値が改善されていくため, 弱コミットメント探索アルゴリズムが制約違反最少化バックトラッキングアルゴリズムより効率的となることが予想される. 例えば, 3-SAT 問題の n が大きい範囲では, 制約違反最少化バックトラッキングは, 100 回の試行中, 一回もバックトラックなしで解が得られていないが, 弱コミットメント探索アルゴリズムでは解が得られるのは, 初期値の改善の効果のためであると考えられる.

2.4.6 考察

アルゴリズムの計算量

制約充足問題は NP 完全な問題であり, 弱コミットメント探索アルゴリズムの最悪ケースの計算量は変数の個数 n に対して指数的である.

必要とされるメモリ量は, *nogood-list* への追加の回数, すなわちやり直しの回数によって決まり, これも最悪ケースには変数の個数 n に対して指数的となる. 一方, スタックを用いてバックトラック型のアルゴリズムをインプリメントした場合のメモリ量は n に対してリニアであり, 弱コミットメント探索アルゴリズムは, 探索の順序を柔軟に変えながらも完全性を保証しているため, 必要とされるメモリ量が指数オーダになってしまっている. これは前述の fill アルゴリズムに関しても同様で, メモリ量は最悪ケースには変数の個数 n に対して指数的となる.

しかしながら, やり直しの回数は, 解が求まるまでに必要とされるサイクル数を越えることはなく, 実験結果で用いた例題の範囲では, 必要とされるサイクル数は n に対して, 高々自乗の範囲であり, 現実的には *nogood-list* の大きさが問題になることはないと考えられる. また, 他の *nogood-list* の要素を包含するような要素は冗長であり, このような要素を *nogood-list* から取り除くことにより, *nogood-list* の要素数を減少させることが可能である.

さらに *nogood-list* への追加の個数を制限することも可能である. この場合にはアルゴリズムの理論的な完全性は保証されないが, 現実的には問題にならないと考えられる. 実際, *nogood-list* への追加の個数を 10 に制限し, 最も新しい

時点で得られた部分解 10 個のみを記録した場合でも, すべての例題に関して無限ループに陥ることなく解が得られている.

2.5 結言

本章では, 制約充足問題の定義を示し, 従来の制約充足問題の解法の紹介を行った. さらに, バックトラッキングアルゴリズムと反復改善型の長所を合わせ持った新しい探索アルゴリズムである弱コミットメント探索アルゴリズムを示し, 実験結果および確率モデルを用いた評価により, 弱コミットメント探索アルゴリズムの有効性を示した.

第 3 章

分散制約充足問題

3.1 序言

本章では分散制約充足問題の定義を示し, 分散協調問題解決の応用問題がどのようにして分散制約充足問題として定式化されるかを示す.

3.2 問題の定義

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.

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．この理由は，本研究の関心は知的エージェント間の協調にあり，制約充足問題を特定のマルチプロセッサのアーキテクチャで解くことではないためである．

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．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り，すべてのエージェントの変数への値の割当がすべての制約を満たす必要がある．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 (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 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，$\forall x_j$ 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，$\forall p_l$ known(p_l,k) なる制約が，$x_1=d_1,x_2=d_2,\ldots,x_n=d_n$ の元で満足される．

例えば，図 2.1で示した n-queens 問題で，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，この問題は分散制約充足問題として定式化される．この問題を分散 n-queens 問題と呼ぶ (図 3.1)．

3.3 応用問題

本節では，分散協調問題解決で扱われて来た様々な応用問題がどのように分散制約充足問題にマッピングされるかを示す．

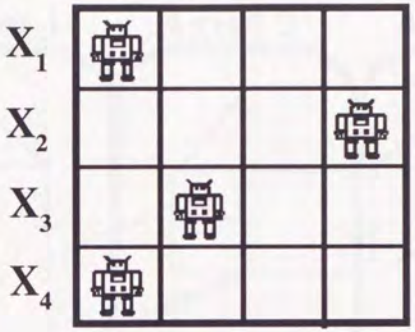


図 3.1: 分散 4-queens 問題

3.3.1 解釈型の問題

1.1節に示したように, 解釈型の問題とは, センサ情報等を, より高レベルの表現に変換する問題であり, 各エージェントがセンサ情報の一部分を担当し, エージェント全体として矛盾のない解釈を得ることが目的である. 入力データの可能な解釈の候補を変数の取り得る値として表すことができれば, 解釈型の問題は分散制約充足問題にマッピングできる. 例えば, 線画の稜線を解釈する問題では, 各頂点の可能な解釈方法は有限であり, 線分で結ばれた頂点の解釈の間には制約があるため, 制約充足問題としてマッピングされることが知られている [Waltz 1975]. 図 3.2に示すように, 複数のエージェントが線画の一部分を担当して解釈を行う問題は分散制約充足問題としてマッピング可能である.

[Mason & Johnson 1989] では, 解釈型の問題を解くために, 第 5 章で示す分散 ATMS と同様なモデルを用いることを提案している. このモデルでは, 各エージェントは自分の複数のセンサデータに関する無矛盾な解釈の組を求める. その際に, 第 5 章で示す分散 ATMS を用いた consistency アルゴリズムを適用することにより, 他のエージェントの解釈と矛盾する解釈を排除することがで

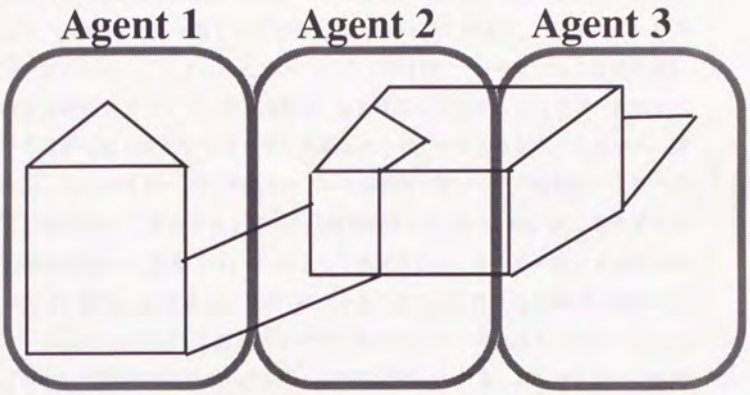


図 3.2: 分散解釈問題の例 (線画の解釈問題)

きる[1].

DVMT[Lesser & Corkill 1983] は DAI の認識型の問題の代表的な例であり, 各エージェントが独立に部分問題を解き, 得られた中間結果を通信し合うこと (結果共有) によって, 解の候補を刈り込み, 最終的に正しい解に到達する方法が提案されている. 各エージェントがまず, 各部分の整合のとれた解釈を求め, それらを交換して第 5 章で示す分散 consistency アルゴリズムにより解の候補を絞り込んでいく解法は, 結果共有の一種と考えられる.

3.3.2 割当型の問題

1.1節に示したように, 割当型の問題とは, 複数のエージェントにタスク, 資源を割り当てる問題である. このような問題でエージェント間の制約が存在する

[1][Mason & Johnson 1989] の応用では, 各エージェントの解釈は, 他のエージェントの解釈とは無矛盾であることが望ましいが, 互いに矛盾する場合もエージェント間の見解の相違として容認しており, エージェント全体の解釈が整合がとれていることを要求する分散制約充足問題とは問題の性質が若干異なっている.

場合には, 各タスク, 資源を変数, その可能な割当方法を変数の値とおくことにより, 分散制約充足問題として定式化が可能である. 例えば, マルチステージネゴシエーションプロトコル [Conry *et al.* 1991] は, 一つのゴールの達成方法に複数の可能性（プラン）がある場合, ある時刻で存在する複数のゴールのすべてを達成可能とするようなプランの組合せを求めることを目的としている. 図3.3に, マルチステージネゴシエーションの研究で用いられた通信ネットワークの例題を示す. このネットワークは複数のサイト (A-1, B-2 等), 各サイトを結ぶ通信リンク (L-5, L-11 等) によって構成される. 各サイトはいくつかの地域的に分割された領域 (A, B 等) に分かれており, それぞれの領域には独立なエージェントが存在し, 領域内のサイトのコントロールを行う. 各エージェントは自分の領域内に関してのみ詳しい情報を持っている. エージェントは接続要求 (ゴールと呼ぶ) に応じてサイトとサイトを結ぶ通信路を確保しようとするが, 各通信リンクの収容可能な通信路の数には制限がある. 各要求 (ゴール) に対して, ゴールを満たすための複数のプランが得られており, 各エージェントはプランの一部 (プランの断片と呼ぶ) を認識している. 表 3.1に, 図 3.3の A-1 と D-1 を接続したいという要求 (goal-1) と, A-2 と E-1 を接続したいという要求 (goal-2) がある場合のプランの断片の例を示す. また, 表 3.2に, これらのプランの断片から構成される全体としてのプランを示す. この問題は, 通信路を確保するという要求 (複数存在する) に対して, 要求を満たし, かつ資源の制限を満足するように各エージェントの持つ資源の割当を決定する問題であり, 分散制約充足問題の特別な場合であると考えられる. この問題を分散制約充足問題として定式化することは容易である.

スケジューリング問題は, 資源, 時間の適切な割当を探索する問題である. スケジューリング問題は一般には最適化問題 (すべての処理が終了するまでの時間を最小化する) として表現されるが, あるデッドラインまでに終了するスケジュールを求める問題等は, 制約充足問題として表現可能である. [Burke & Proccer 1991; Sycara *et al.* 1991] では複数のエージェントの関連するスケジューリング問題を分散制約充足問題として定式化する試みが示されている.

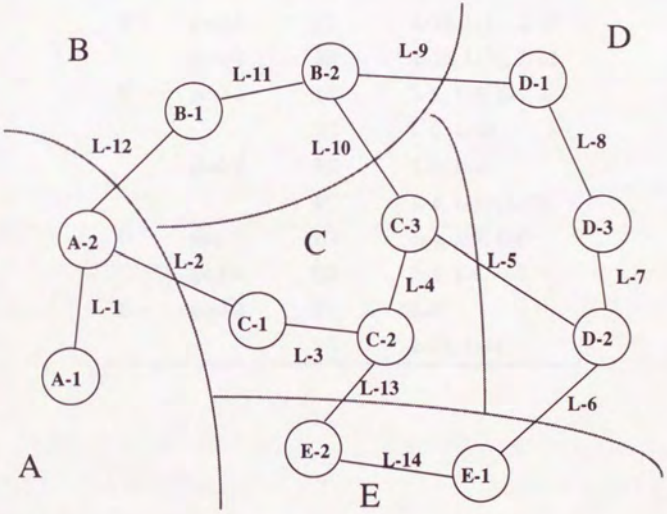


図 3.3: 通信ネットワークの例

表 3.1: プランの断片

Agent	Goal	Plan fragment	Resource Used
A	goal-1	1A	L-1, L-2
		2A	L-1, L-12
	goal-2	3A	L-12
B	goal-1	1B	L-10, L-11, L-12
	goal-2	2B	L-10, L-11, L-12
C	goal-1	1C	L-2, L-3, L-4, L-5
		2C	L-5, L-10
	goal-2	3C	L-5, L-10
		4C	L-4, L-10, L-13
D	goal-1	1D	L-5, L-7, L-8
	goal-2	2D	L-5, L-6
E	goal-2	1E	L-6
		2E	L-13, L-14

表 3.2: プランの候補

goal	plan	plan fragments
goal-1	plan-11	1A, 1C, 1D
	plan-12	2A, 1B, 2C, 1D
goal-2	plan-21	3A, 2B, 3C, 2D, 1E
	plan-22	3A, 2B, 4C, 2E

3.3.3 マルチエージェント真偽値管理システム

マルチエージェント真偽値管理システム [Bridgeland & Huhns 1991] は, 真偽値管理システム [Doyle 1979] の分散版である (図 3.4). 真偽値管理システムでは, データベース中に真偽値が不明のデータが存在し, また, データの真偽値の間に依存関係と呼ばれる制約が存在する. この制約を満足するようにデータの真偽値を決定することが目的である[2]. 真偽値が不明なデータは, 真, 偽の 2 値を取る変数であるとおくことにより, この問題は制約充足問題として表現することができる.

マルチエージェント真偽値管理システムでは, 各エージェントは独立な真偽値管理システムを持つが, 一部のデータに関してエージェント間で共有が行われており, 共有されるデータに関してはエージェント間で真偽値が一致すること (shared consistency) が要求される. 異なるエージェント間に制約が存在するため, この問題は分散制約充足問題とみなすことができる. [Bridgeland & Huhns 1991] では 4.2節で示す同期バックトラッキングと類似のアルゴリズムによりデータの整合性を管理する方法が提案されている.

3.4 結言

本章では分散制約充足問題の定義を示し, 従来, 全く異なる問題として扱われてきた分散協調問題解決の様々な応用問題に関して, 分散制約充足問題として定式化する方法を示した.

[2]この問題は, 真偽値を割り当てる, 割当問題の一種とみなすことが出来る. 一方, 各エージェントが持つデータ間の制約を満たすように解を求める問題と考えれば, 解釈問題の一種であるとみなすことも可能である.

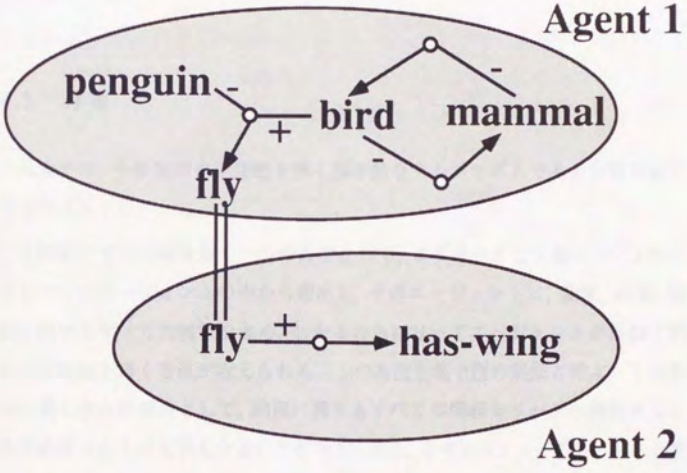


図 3.4: マルチエージェント真偽値管理 システム

第 4 章

分散探索アルゴリズム

4.1 序言

本章では，分散制約充足問題を解く基本的なアルゴリズムである分散探索アルゴリズムを示す．

分散制約充足問題を解く一つの方法として，まずリーダとなるエージェントをすべてのエージェントの中から選出し，そのエージェントに，変数，領域，制約に関するすべての情報を集め，しかるのちにリーダエージェントが単独で制約充足問題を解く方法が考えられる．この方法を集中型の解法と呼ぶ．この方法の明らかな問題点として，問題に関するすべての情報をリーダへ通信するための通信コストが大きくなることがある．また，セキュリティ，プライバシー等の観点から，各エージェントが自分の持っている問題に関する情報をすべて公開し，他のエージェントに送信することが望ましくない場合が存在する．そのような場合には集中型の解法を用いることは不可能であり，情報の集中を行わない分散型の解法が必要とされる．

本章では，まず，通常の集中型の制約充足問題を解くバックトラッキングアルゴリズムを複数のエージェントによりシミュレートする同期バックトラッキングを示す．バックトラッキングアルゴリズムは本質的に逐次的なものであり，同期バックトラッキングアルゴリズムにおいては，エージェントはあらかじめ定められた順序で逐次的に動作する．しかしながら，処理の並列性の追求のた

めには, 探索において各エージェントが全体的な制御なしに, 非同期, 並行的に動作できることが望ましい. 一方, 全体的な制御なしでは一般にアルゴリズムの完全性を保証することが困難となる. 本論文では, 各エージェントが全体的な制御なしに, 非同期, 並行的に動作しながらも, アルゴリズムの完全性が保証されることを特徴とする非同期バックトラッキングアルゴリズムを示す.

また, 非同期バックトラッキングアルゴリズムをベースとして, 第2章で示した弱コミットメント探索アルゴリズムと同様な性質を持つ非同期弱コミットメント探索アルゴリズムが実現されることを示す.

最後に, 例題による実験結果を用いてこれらのアルゴリズムの性能の比較を行う. 実験結果により, 非同期バックトラッキングアルゴリズムの同期バックトラッキングアルゴリズムに対する優位性, および非同期弱コミットメント探索アルゴリズムの非同期バックトラッキングアルゴリズムに対する優位性を示す. また, 集中型の解法と分散型の解法との性能の比較検討を行う.

4.2 同期バックトラッキング

分散制約充足問題においては, 次のような手順で, 制約充足問題でのバックトラッキングアルゴリズムをメッセージ通信によってシミュレートすることができる. エージェント間で, 変数を決める順序に合意がなされている（例えばエージェント $1, 2, \ldots, m$ の順に値を決定する）と仮定する. 各エージェントは一つ前のエージェントから部分解を送信されると, 自分の持つ変数の値を決定して部分解に付け加え, 次の順番のエージェントに部分解を送信する. 送信された部分解と制約を満たす値が存在しない場合はバックトラックメッセージを一つ前のエージェントに送信する.

このアルゴリズムは, 集中型の解法と比較して, 問題のすべての情報を一つのエージェントに通信する必要はないが, エージェント間で変数を決める順序を決定するためにはエージェント間通信が必要であり, また, ある瞬間には一つのエージェントしか動作しないため並列性は生かせない. [Collin, Dechter, & Katz 1991] では, 同期バックトラッキングと類似の, ネットワーク consistency

プロトコルと呼ばれるアルゴリズムが提案されている．このアルゴリズムでは，エージェントは深さ優先探索の木を構成し，*privilege* と呼ばれるトークンを通信し合うことにより，同期，逐次的に動作するが，深さ優先探索木において同じ親ノードを持つ複数のエージェントは並行して動作することが可能である．

4.3 非同期バックトラッキング

本論文で提案する非同期型バックトラッキングでは，各エージェントは順番にではなく，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．問題を簡単化するため，以下の議論では次の仮定をおく．これらの仮定の一般化は容易であり，アルゴリズムの一般性は失われていない．

- 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
- すべての制約は binary (2 変数間) である

4.3.7節で，最初の仮定を一般化し，各エージェントが複数の変数を持つ場合のアルゴリズムを示す．

4.3.1 制約ネットワーク

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，エージェントとその変数について，同じ識別子 (id) を用いることとする．さらに，すべてのリンク (制約) に関して，方向が定められていると仮定する．すなわち，ある制約に関係する 2 つのエージェントのうち，片方がその制約の評価を担当して，もう片方のエージェントは，自分の変数の値を制約を評価するエージェントに送信する．値を送信するエージェントから制約を評価するエージェン

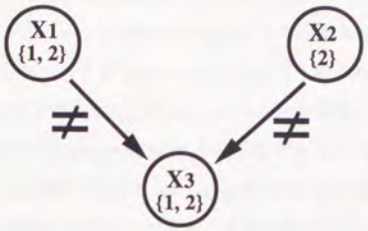


図 4.1: 制約ネットワークの例

トに向かうように, リンクの向きが付けられる. 図4.1の例では, 変数 x_1, x_2, x_3 (領域は $\{1,2\},\{2\},\{1,2\}$) のそれぞれを持つエージェントが存在し, 制約 $x_1 \neq x_3, x_2 \neq x_3$ が存在する. 注意すべきことは, このリンクはエージェント間の論理的な関係 (制約) を表すものであり, エージェント間の物理的な通信ネットワークとは無関係であることである.

4.3.2 アルゴリズムの概要

各エージェントは並行して自分の変数の値を選び, その値を, 外向きのリンクで結ばれたエージェントに対して送信する. その後, 各エージェントはメッセージ待ちの状態となる. 図4.2に, 二種類のメッセージを受けた時のエージェント x_i の動作を記述する. 一つは *ok?* メッセージで, このメッセージは値を送信するエージェントから制約を評価するエージェントに送られるものであり, 選択した値が許容できるかを尋ねるものである (図4.2 (i)). もう一種類のメッセージは *nogood* メッセージで, このメッセージは制約を評価するエージェントから値を送信するエージェントに送られるもので, 制約評価エージェントが制約条件違反を発見したことを示す (図4.2 (ii)).

エージェントは, 他のエージェントから送信された変数の値の割当の集合を管理する. この集合をエージェントの *agent_view* と呼ぶ. エージェント x_1 の値が 1 であると認識されていることを, エージェントの識別子と値のペア $(x_1, 1)$

として表現する．よって，*agent_view* は例えば $\{(x_1,1),(x_2,2)\}$ のように表現される．もし *ok?* メッセージが内向きのリンクから送信された場合，制約評価エージェントは送られたペアを *agent_view* に追加し，現在の自分の変数の割当 ($(x_i,$ *current_value*$)$) と表現する) が，*agent_view* と無矛盾であるかをチェックする．自分の変数の割当が *agent_view* と無矛盾であるとは，エージェントの評価する制約が，自分の値の割当と *agent_view* に示される値の割当の元ですべて真であり，かつ，後述する他のエージェントから通信された *nogood* のスーパセットとならない場合である．もし自分の変数の値が *agent_view* と矛盾する場合は，エージェントは *current_value* を，*agent_view* と矛盾しないように変更する．

エージェントの持つ変数のすべての値が，*agent_view* の部分集合と矛盾する場合，その *agent_view* の部分集合は矛盾しているという．エージェントが自分の *agent_view* の部分集合が矛盾していることを発見した場合，他のエージェントの値の割当が変更される必要があり，エージェントはバックトラックを生じ (図 4.2 (iv))，他のエージェントに対して *nogood* メッセージを送信する．

図 4.2 では，エージェントは送信されたメッセージに対して逐次的に処理を行うように記述が行われているが，実際にはエージェントは複数のメッセージを並行して処理することが可能である．すなわち，複数の *ok?* メッセージおよび *nogood* メッセージを受けて *agent_view* および *nogood_list* を修正した後に，一回だけ **check_agent_view** を実行すれば良い．

4.3.3 処理の無限ループの回避

処理の無限ループとは，複数のエージェントが値の変更を繰り返し，安定した解に到達しない場合である．値の変更のループが存在する場合，処理の無限ループが生じる可能性がある．例えば，変数 x_1 の値の変更が変数 x_2 に影響を及ぼし，x_2 の値が変更され，この変更が変数 x_3 に影響を与え，x_3 の値が変更され，x_3 の値の変更が今度は変数 x_1 に影響を与えるという場合である．このような値の変更のループは，制約ネットワーク上では有向リンクのサイクルとして

when received (**ok?**, (x_j, *value*)) **do** — (i)
add (x_j, *value*) to *agent_view*;
check_agent_view; **end do**;

when received (**nogood**, x_j, *nogood*) **do** — (ii)
add *nogood* to *nogood_list*;
when (x_k,*value*) where x_k is not connected is contained in *nogood* **do**
request x_k to add a link from x_k to x_i
and add (x_k, *value*) to *agent_view*; **end do**;
old_value ← *current_value*; **check_agent_view**;
when *old_value* = *current_value* **do**
send (**ok?**, (x_j, *current_value*)) to x_j; **end do**; **end do**;

procedure **check_agent_view**; — (iii)
when *current_value* and *agent_view* are inconsistent **do**
change *current_value* to a new consistent value; — (iii–a)
when can not find such a value **backtrack**;
change *current_value* to a new consistent value; **end do**;
send (**ok?**, (x_i, *current_value*)) to its outgoing links; **end do**;

procedure **backtrack** — (iv)
nogoods ← {V | V=inconsistent subset of *agent_view*};
when {} is an element of *nogoods* **do**
broadcast to other agents that there is no solution,
terminate this algorithm; **end do**;
for each $V \in$ *nogoods* **do**;
select (x_j, d_j) where x_j has the lowest priority in V; — (iv-a)
send (**nogood**, x_i, V) to x_j;
remove (x_j, d_j) from *agent_view*; **end do**;

図 4.2: メッセージに対する処理 (非同期バックトラッキング)

表現される．リンクの向きは変数の値を変更する順序を示している．すなわち，制約を評価する (リンクが内向きの) エージェントは，制約が満たされない場合に，まず自分の値を変更し，制約を満たす値が存在しない場合に限り，値を送信する (リンクが外向きの) エージェントに *nogood* メッセージを送信する．

ここでは，分散データベースシステムにおけるデッドロック回避手法 [Rosenkrantz, Stearns, & Lewis 1978] と同様に，エージェントのユニークな識別子を用いてサイクルを回避する．すなわち，それぞれのエージェントにユニークな識別子を与え，これらの識別子の辞書式順序でエージェント (ノード) 間の優先順位を定義する (辞書式順序で先行する方が優先順位が高い)．すべてのリンクの向きを，この辞書式順序で決定される優先順位に従い，優先順位の高いノードから低いノードに向くように設定すれば，値の変更のループは存在しない．これは，それぞれの制約に対して，優先順位の高いエージェントが値を送信し，優先順位の低いエージェントが制約の評価を行うことに対応する．さらに，nogood が発見された場合に，nogood を発見したエージェントは，nogood 中の最も優先順位の低いエージェントに対して *nogood* メッセージを送信する (図 4.2の (iv)-a)．

注意すべきことは，このユニークな識別子を用いる方法で必要とされるエージェントの知識は，同期バックトラッキングを行うためにエージェントが持つべき知識と比較して，はるかに局所的であることである．同期バックトラッキングでは，エージェントはあらかじめ定められた逐次的な順序で動作しなければならない．このような順序は，エージェントにユニークな識別子を与えるだけでは得られない．各エージェントは，すべてのエージェント中で，自分の前と次のエージェントがだれかを認識しなければならず，このために自分に最も近い識別子を持つエージェントをすべてのエージェントから探索しなければならない．一方，非同期バックトラッキングで要求されるのは，制約で関連し合うエージェント（多くの場合，全エージェント中のごく一部）中での順序関係のみを認識することである．

4.3.4 非同期な値の変化への対応

以下, エージェントが非同期に動作することにより生じる問題点への対応方法を示す.

agent_view の不整合への対応

エージェントは非同期に値の変更を行うため, *agent_view* は絶えまなく変更される. このため, 制約評価エージェントが *nogood* メッセージを送信した場合, 他のエージェントの値の割当が, 別の制約のためにすでに変更されている可能性がある. 例えば, 図 4.3 (a) で, エージェント x_3 が *nogood* メッセージをエージェント x_2 に送信する場合, メッセージを送信した瞬間に, x_1 の値が, 図に示されていない他の制約のため 2 に変化したと仮定すると, 新しい状況ではもはや x_2 の値を変化させる必要はない. 古い誤った情報に基づいてバックトラックすることにより, 正しい解を見逃してしまう可能性が生じる.

本論文では nogood の通信により, *agent_view* の不整合に対応する. すなわち, エージェントは *nogood* メッセージに, バックトラックを生じる前提となった *agent_view* (もしくはそのサブセット) を付加する. この *agent_view* もしくはそのサブセットは, 最初に与えられた制約から新たに導かれた新しい制約であると考えることができるため, この *agent_view* もしくはそのサブセットを nogood と呼ぶ. この nogood はバックトラックを生じたコンテキストである. メッセージの受け手のエージェントは, メッセージの前提となる nogood が, 現在自分の認識している *agent_view* および自分の値と整合がとれているかを確認して, 整合がとれている場合に限り値の変更を行う (図 4.2の (iii-a)). *nogood* メッセージに付加される nogood は制約条件違反が生じた原因であるため, 非同期型バックトラッキングは制約充足問題における依存関係に基づくバックトラッキング [de Kleer 1987] の機能を含む. 注意すべきことは, 制約充足問題における依存関係に基づくバックトラッキングは, 効率化のための手段であるが, 非同期バックトラッキングにおける nogood の通信は, 非同期な変化のもとで正しい解を求めるために本質的であることである.

新しい制約の追加は，エージェント間に新しい関係が生じることを意味する．例えば図 4.3 (b) で，nogood$\{(x_1,1),(x_2,2)\}$ は，x_1 と x_2 の間の新しい制約とみなすことができる．最初の状態では x_1 と x_2 の間にはリンクは存在しないが，この新しい制約に対応するためには，x_1 と x_2 間に新しいリンクを追加する必要がある．注意すべきことは，制約ネットワークにおけるリンクはエージェント間の論理的な関係を表すものであるため，リンクを追加することはエージェント間に新しい物理的な通信路を設けることを意味するものではないことである．より一般的には，最初に与えられた制約がすべて binary であっても，新たに導かれる制約は 2 つ以上の変数が関係する場合が存在する．そのような場合には，nogood 中の最も優先順位の低いエージェントが制約評価を行い，nogood 中のその他のエージェントと制約評価エージェント間にリンクを追加する．最初に与えられた制約ネットワークに対してリンクを追加する，すなわち最初に陽に表現されていない制約を明らかにするというアイデアは，制約充足問題の consistency アルゴリズムにおいて用いられている．例えば，[Dechter & Pearl 1988] の adaptive consistency アルゴリズムは，与えられた制約ネットワークを，バックトラックなしで解が得られる制約ネットワークに変形する過程において，リンクの追加を行う．

制約チェックへの割り込み

非同期バックトラッキングにおいて，*agent_view* はエージェントが制約を満たす値を探している途中で変化する可能性がある．この場合，変化する以前の *agent_view* に基づいて行った制約評価の結果選ばれた値は，結局新しい *agent_view* に基づいて再評価を行わなければならないため，*agent_view* が変化した場合に処理を続けるのは無駄である．非同期バックトラッキングアルゴリズムでは，制約評価の処理 (図 4.2 (iii-a) で実行される) に，*agent_view* が変化した場合に割り込みをかけることでこの問題に対応する．

さらに，バックマーキング [Gasching 1977] と呼ばれる手法を導入することにより，エージェントが割り込みをかけられた途中の処理を最大限に活用する

ことを可能にする．バックマーキングの基本的なアイデアは以下のとおりである．エージェント i が自分の変数 x_i と，他のエージェントの持つ変数 $x_1, \ldots, x_{i-1}$ の間の制約チェックを行っていると仮定する．もし自分の変数の値 d が，変数 x_j との制約チェックに失敗したとする．その場合，エージェントは値 d に対して j なるマークを付ける．ここで，エージェントが変数 x_k の値が変化したというメッセージを受けたとする．もし値 d がこのメッセージを受ける以前に j とマークされており，かつ $j < k$ ならば，エージェントは，変数 x_k の値が変化しても値 d は制約条件違反を引き起こすと判断できる．なぜならば，制約条件違反を引き起こした変数 x_j の値は変化していないからである．一方，$k < j$ である場合，値 d が新しい *agent_view* と無矛盾であることをしらべる際には，変数 $x_1, \ldots, x_{k-1}$ との制約チェックは省略することができる．なぜならば，これらのチェックは以前に成功しており，これらの変数の値は変化していないためである．このようにして，エージェントは割り込みされた処理からの情報を利用して，不要な制約チェックを省くことが可能となる．

4.3.5 アルゴリズムの実行例

図 4.3 (a) では，エージェント x_3 はエージェント x_1, x_2 から *ok?* メッセージを受け，*agent_view* は $\{(x_1,1),(x_2,2)\}$ となる．x_3 の値 1,2 のどちらもこの *agent_view* と制約を満たさないため，*agent_view* が矛盾であり，*agent_view* から最も優先順位の低いエージェント（この場合はエージェント x_2）に対して *nogood* メッセージを通信し，*agent_view* から $(x_2,2)$ を取り除く．

一方エージェント x_2 は，nogood メッセージを受けて，nogood $\{(x_1,1),(x_2,2)\}$ を記録する．また，この nogood 中には，エージェント x_2 とリンクで結合されていないエージェント x_1 が含まれている．このため，x_1,x_2 間にリンクを追加する必要があり，エージェント x_2 は x_1 に対して，今後の値の変更の通信を要求するメッセージを出し，$(x_1,1)$ を *agent_view* に追加する (図 4.3 (b))．エージェント x_2 は，*agent_view* と自分の値が無矛盾かどうかをチェックする．*agent_view* $\{(x_1,1)\}$ と $(x_2,2)$ は，通信された nogood $\{(x_1,1),(x_2,2)\}$ を含んでおり，矛盾

である．しかしながら，x_2 には他に可能な値がないため，*agent_view* $\{(x_1, 1)\}$ は矛盾である．このため，エージェント x_2 はエージェント x_1 に対して，nogood メッセージを送信する (図 4.3 (c))．

4.3.6 アルゴリズムの健全性と完全性

非同期型バックトラッキングアルゴリズムは，制約を満たす解が存在する時には必ず解を発見してすべてのエージェントがメッセージ待ちの安定状態になり[1]，また，解が存在しない場合は，解が存在しないことを発見して停止する．すべてのエージェントが安定状態に達するのは，すべてのエージェントで制約が満たされている場合のみであり，アルゴリズムの健全性は明らかである．以下，このアルゴリズムが完全であることを示す．

アルゴリズムが停止するのは，空集合が nogood になった場合のみである．nogood は論理的には矛盾を導く変数の値の組合せを意味し，nogood のスーパセットとなるような値の組は解にはなり得ない．空集合が nogood であることは，どのような値の組も解になり得ないことを意味し，明らかに制約を満たす解は存在しない．

以上で，アルゴリズムが安定状態に達した場合は解が得られ，このアルゴリズムが停止する場合には解は存在しないことを示した．以下，このアルゴリズムが有限時間内に安定状態に達するか，もしくは停止することを示す．もしアルゴリズムが決して安定状態に達せず，かつ停止もしない場合には，少なくとも一つのエージェントは値の設定と確認要求を無限に繰り返している（処理の無限ループに陥っている）．以下，数学的帰納法によりこのような場合が存在しな

[1]注意すべきことは，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トが安定状態に達したことを検出することは分散システムにおける重要な問題であり，様々な研究がなされている [Chandy & Lamport 1985]．一方，分散制約充足問題では，制約が満足されていればエージェントは安定状態となるため，優先順位の最も低いエージェントから順に，安定状態となった時点で *agent_view* を通信し，*agent_view* を通信されたエージェントは，通信された *agent_view* が制約を満足していれば自分の *agent_view* をマージして，次のエージェントに通信を行うといった簡単な方法で終了判定が可能である．

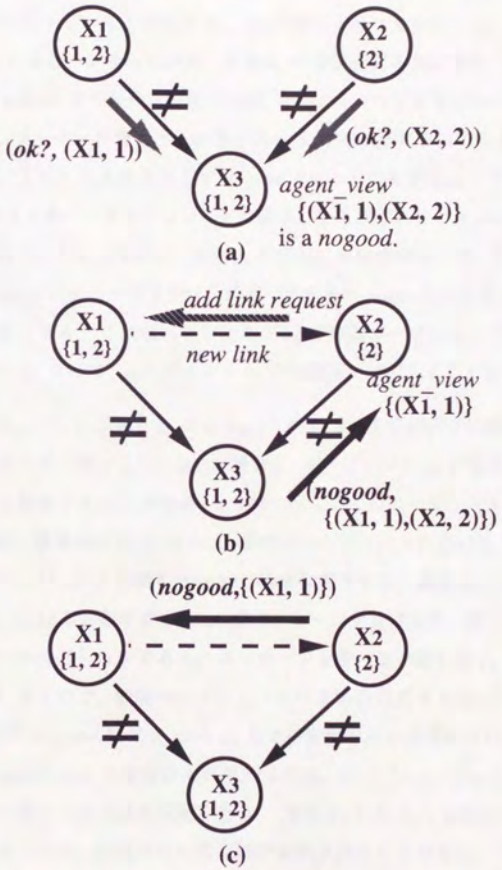


図 4.3: アルゴリズムの実行例 (非同期バックトラッキング)

いことを示す.

まず, すべてのエージェント中で最も優先順位の高いエージェント x_1 が無限ループに入っていると仮定する. x_1 が受けるメッセージは $x_2, \ldots, x_n$ からの *nogood* メッセージのみである. 変数 x_1 の値が変更された場合, x_1 はその値に関して *nogood* メッセージを受けるか, 全くメッセージを受けないかのどちらかである. メッセージを受けない場合は x_1 は安定状態になってしまうので, 無限ループに入っている場合は必ず *nogood* メッセージを受ける. その値に関して *nogood* メッセージを受けると, その値は二度と選ばれることはなく, 新しい値が選ばれる. しかしながら, 変数 x_1 の領域は有限であるため, すべての値に関して *nogood* メッセージを受けた時点で空集合の nogood が生成され, アルゴリズムは終了する. これはエージェント x_1 が無限ループに入っているとの仮定に矛盾する. よって, エージェント x_1 が無限ループに入ることはない.

さらに, エージェント x_1 から x_{k-1} $(k > 2)$ までが安定状態に達している (無限ループに陥いっていない) 場合に, エージェント x_k が無限ループに陥っていると仮定する. この場合, x_1 から x_{k-1} までのエージェントは安定状態であるため, 最終的には x_k から x_n までのエージェントにおいて, x_1 から x_{k-1} までのエージェントに関する *agent_view* は整合する. 変数 x_k の値が変更された場合, x_k はその値を含む *nogood* メッセージを受けるか, 全くメッセージを受けないかのどちらかである. メッセージを受けない場合は x_k は安定状態になってしまうので, 無限ループに入っている場合は必ず *nogood* メッセージを受け, その nogood には x_1 から x_k までの変数のみしか現れない. エージェント間で *agent_view* の整合がとれているため, エージェント x_k は nogood メッセージに従って必ず値の変更を行う. また x_1 から x_{k-1} は安定状態で値が変更されないため, 変更された値は再び選択されることはない. しかしながら変数の値の領域は有限であるため, エージェント x_k は, すべての値に関して nogood メッセージを受けた後, 必ずエージェント x_1 から x_{k-1} までのどれかにバックトラックを行う. これはエージェント x_1 から x_{k-1} までが安定状態であるという仮定に矛盾する. よって, エージェント x_1 から x_{k-1} までが安定状

態である場合に, エージェント x_k は無限ループに陥いることはない. したがって, すべてのエージェントは無限ループに陥ることはなく, 有限時間で停止するか, もしくは安定状態に達する.

4.3.7 複数の変数を持つ場合への拡張

本節では, エージェントが複数の変数を持つ場合に拡張したアルゴリズムを示す. エージェント $1,2,\ldots,m$ が存在し, 各エージェントは複数の変数を持つ. 各変数にはユニークな識別子が与えられていることを仮定する. 変数 x_{ik} の値を value(x_{ik}) と記述する. エージェントは *current_assignments* と呼ばれるリストを管理する. このリストは自分の持つ変数の値の割当のリストである. 各エージェントの持つ変数の初期値は, エージェント内での制約を満足していることを仮定する. 変数の優先順位を変数の識別子 (id) の辞書式順序で定義する. 図 4.4 に, エージェント i が *ok?* もしくは *nogood* メッセージを受けた時の手続きを示す. エージェントは *current_assignments* をチェックし, 制約を満足していない変数のうち, 最も優先順位が低いものの値を変更する. 制約チェックを行う際には, *agent_view* と *current_assignments* 中の, 優先順位がより高い変数との間の制約のみを考慮する. この手続きは, *current_assignments* が *agent_view* と矛盾しなくなるまで続けられる.

4.4 非同期弱コミットメント探索アルゴリズム

本節では以下, 非同期バックトラッキングアルゴリズムをベースとした非同期弱コミットメント探索アルゴリズムを示す.

4.4.1 基本的なアイデア

第 2 章で示した弱コミットメント探索アルゴリズムの基本的な特徴は次の 2 点である.

when received (**ok?**, (*sender_id*, *variable_id*, *variable_value*)) **do**
add (*sender_id*, *variable_id*, *variable_value*) to *agent_view*;
check_agent_view; **end do**;

when received (**nogood**, x_k, *sender_id*, *nogood*) **do**
add *nogood* to *nogood_list*
when (*id*,*x*,*value*) where *id* is not connected is contained in *nogood* **do**
request *id* to add a link from *id* to *my_id*
and add (*id*,*x*,*value*) to *agent_view*; **end do**;
old_value ← value(x_k); **check_agent_view**;
when *old_value* = value(x_k) **do**
send (**ok?**, (i, x_k, value(x_k))) to *sender_id*; **end do**; **end do**;

procedure **check_agent_view**
when *agent_view* and *current_assignments* are not consistent **do**
select a variable x_k in *current_assignments*,
which has the lowest priority within inconsistent variables;
check_agent_view_one (x_k); **end do**;

procedure **check_agent_view_one**(x_k)
if no value in D_k is consistent with
agent_view and *current_assignments* **then**
backtrack (x_k); **check_agent_view**;
else select $d \in D_k$ where d is consistent
with *agent_view* and *current_assignments*;
value(x_k) ← d;
send (**ok?**, (i, x_k, d)) to other agents;
add (i, x_k, value(x_k)) to *current_assignments*; **check_agent_view**; **end if**;

procedure **backtrack**(x_k)
nogoods ← $\{V \mid V =$ inconsistent subset of *agent_view* ∪ *current_assignments*$\}$;
when $\{\}$ is an element of *nogoods* **do**
broadcast to other agents that there is no solution,
terminate this algorithm; **end do**;
for each $V \in$ *nogoods* **do**;
select (j, x_l, d_l) where x_l has the lowest priority in V;
if $j = i$ **then** add V to *nogood_list*; **check_agent_view_one**(x_l)
else send (**nogood**, x_l, i, V) to j;
remove (j, x_l, d_l) from *agent_view*; **end do**; **end if**;

図 4.4: メッセージに対する処理 (複数の変数)

1. 変数の値の選択のヒューリスティックとして制約違反最少化ヒューリスティックを用いる.

2. 通常のバックトラッキングアルゴリズムではバックトラックが生じる時点で, 部分解の構成を最初からやり直す.

これらの特徴のうち, 1 を非同期バックトラッキングアルゴリズムに導入するのは簡単であり, 各エージェントにおいて, *agent_view* と整合する値 (優先順位が高い変数の値と制約を満たす値) が複数存在する場合に, 優先順位が低い変数との間の制約をより多く満たす値を優先して値の割当を行うように変更すれば良い. 一方, 2 を非同期バックトラッキングに導入するのは簡単ではない. 非同期バックトラッキングでは複数のエージェントが非同期, 並行して動作し, 部分解の全体を把握しているエージェントは一般には存在しない. また, バックトラック自体が非同期, 並行して生じる可能性がある. 本論文では以下, バックトラックすべき状態で, 変数の優先順位を動的に変更することにより, 部分解に弱くコミットする分散制約充足アルゴリズムが構築されることを示す.

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. これらの仮定の一般化は容易であり, アルゴリズムの一般性は失われていない.

- 各エージェントは唯一の変数を持つ.
- 制約はすべて binary (2 変数間) である.
- すべてのエージェント間に制約が存在する[2].

次の方法で変数の優先順位を定義する.

- 各変数に対して, その変数の優先順位を表す非負の整数値を定義する. これを変数の優先度と呼ぶ.

[2] 4.3節で示したように, すべてのエージェント間にあらかじめ制約が存在しない場合は, 後述する nogood の送信によって, 最初は無関係であったエージェント間に, 新しい制約が生じる可能性がある.

- 変数の優先順位は, 優先度の大小関係によって決定される (優先度が大きいほど順序関係で上位となる).
- 優先度が同じ変数の順序関係は, 変数の識別子の辞書式順序で決定される.

さらに, この変数の優先度を次のようなルールで変化させる.

- 変数の優先度の初期値はすべて 0 である.
- 優先順位が上位の変数との制約をすべて満足する値が存在しない場合はバックトラックが生じる. バックトラックが生じた場合, バックトラックが生じた変数の優先度を, 他の関連する変数の優先度 +1 に設定する.

このルールにより, バックトラックを契機として, 変数の優先順位の逆転が生じる. このため, これまで優先順位が上位であった変数は, バックトラックが生じた, これまで優先順位で下位であった変数との制約を満足するように値の変更を行う必要が生じ, 部分解の再構築が行われる.

また, 弱コミットメント探索アルゴリズムの完全性は, nogood が単調に増加し, 以前に発見された nogood を繰り返さないことに依存していた. 非同期弱コミットメント探索アルゴリズムで, 同様に完全性を保証するため, 次の処理を行う.

- 各エージェントは自分が以前に送信した nogood を記録する. バックトラックを生じる時点で, 以前に自分が発見した nogood と同じ状況が生じている場合は, 優先順位を変更せず, メッセージの待ち状態になる.

非同期バックトラッキングアルゴリズムでは, 以前に nogood となった状況は基本的には繰り返すことはない. しかしながら, メッセージの遅れにより, あるエージェントの認識する他のエージェントの状態が, 以前に nogood となった状態と一時的に同じとなってしまう可能性がある. そのような状況で, 実際には安定しては存在しない状態に反応し, 不要な処理を行うと, 予期せぬ無限

ループに陥る可能性がある．nogood を記録し，同じ nogood を繰り返し送信しないことにより，nogood が単調に増加し，処理が無限ループに陥らないことが保証される．

4.4.2 アルゴリズムの詳細

本アルゴリズムでは，基本となる非同期バックトラッキングアルゴリズムと同様，エージェントは非同期に並行して値を決定し，その値を関連するエージェントに送信した後，メッセージ待ちの状態になり，以降，到着したメッセージに関する処理を行う．

図 4.5に 2 種類のメッセージ *ok?*, *nogood* 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，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次の通りである．

- エージェント間で，現在の変数値と優先度 (p_j) が通信される (図 4.5 (i))．通信の対象は優先順位が下位のエージェントのみではなく，関連するすべてのエージェントである．

- 優先順位の判断は通信された優先度を用いて行う．また，現在の値と *agent_view* の整合性のチェックは，*agent_view* 中の，より優先順位の高い変数のみを対象とする．現在の値と *agent_view* が整合が取れていない場合，エージェントは *agent_view* が整合が取れ，かつ *agent_view* 中の自分の変数より優先順位が低いものとの間の制約条件違反を最少化するように値の選択を行い，他のエージェントに通信を行う (図 4.5 (ii))．

- 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，値の変更を要求し，また，前述のルールに従って優先度を変更し，他のエージェントに通信を行う (図 4.5 (iii))．非同期バックトラッキングアルゴリズムでは，nogood の通信相手は nogood 中の最も優先順位が低いエージェントであったが，非同期弱コミッ

トメント探索アルゴリズムでは，将来的に優先順位が変更される可能性があるため，nogood 中のすべてのエージェントに対して通信が行われる．ただし，同じ nogood を以前に送信している場合は，優先順位の変更を行わず，メッセージの待ち状態になる．

図 4.5 では，エージェントは送信されたメッセージに対して逐次的に処理を行うように記述が行われているが，非同期バックトラッキングアルゴリズムと同様，実際にはエージェントは複数のメッセージを並行して処理することが可能である．すなわち，複数の *ok?* メッセージおよび *nogood* メッセージを受けて *agent_view* および *nogood_list* を修正した後に，一回だけ **check_agent_view** を実行すれば良い．

4.4.3 アルゴリズムの実行例

具体例を図 4.6を用いて示す．例題は第 3 章で示した分散 4-queens 問題であり，目的は，4×4 のチェスボード上に，4 つのクイーンを互いに取り合わないようにおくことである．この問題は領域が {1,2,3,4} である 4 つの変数からなる制約充足問題として定式化される．ここでは，各エージェントが一つのクイーンに対応し，それぞれ自分の変数の値を決定しようとしている．

初期状態での変数の値の割当は図 4.6 (a) の通りであり，この値が互いに通信される．図中の変数名の隣の括弧内の数字は変数の優先度を表す．各変数の優先度はすべて 0 であり，変数の優先順位は識別子の辞書式順序で決められる．よって，x_4 が値を変更しようとするが，可能な値が存在しない．x_4 は nogood メッセージ $\{(x_1,1),(x_2,4),(x_3,2)\}$ を他のエージェントに送り，優先度を 1 とし，値を制約条件違反の個数を最少化する値 3 に変更し，新しい優先度と変更された値を他のエージェントに通信する (図 4.6 (b))．

x_4 の優先度が変更されたため，x_3 が値を変更しようとするが，可能な値が存在しない．x_3 は nogood メッセージ $\{(x_1,1),(x_4,3)\}$ を他のエージェントに送り，制約条件違反の個数を最少化する値 1,2 からランダムに値を選択し，値が 1 となる (図 4.6 (c))．ここで，x_1 が値を 2 に変更し，解が得られる (図 4.6 (d))．

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

when received (**nogood**, x_j, *nogood*) **do**
add *nogood* to *nogood_list*;
check_agent_view;**end do**;

procedure **check_agent_view**
when *agent_view* and *current_value* are not consistent **do**
if no value in D_i is consistent with *agent_view* **then**
backtrack;
else select $d \in D_i$ where *agent_view* and d are consistent
and d minimizes the number of constraint violations; — (ii)
current_value $\leftarrow$ d;
send (**ok?**, (x_i, d, *current_priority*)) to other agents; **end if**; **end do**;

procedure **backtrack** — (iii)
nogoods $\leftarrow$ $\{V \mid V =$ inconsistent subset of *agent_view*$\}$;
when $\{\}$ is an element of *nogoods* **do**
broadcast to other agents that there is no solution,
terminate this algorithm; **end do**;
when no element of *nogoods* is included in *nogood_sent* **do**
for each $V \in$ *nogoods* **do**;
add V to *nogood_sent*
for each (x_j, d_j, p_j) in V **do**;
send (**nogood**, x_i, V) to x_j; **end do**; **end do**;
set_new_priority;
select $d \in D_i$ where d minimizes the number of constraint violations;
current_value $\leftarrow$ d;
send (**ok?**, (x_i, d, *current_priority*)) to other agents;

procedure **set_new_priority**
$p_{max} \leftarrow max_{(x_j,d_j,p_j) \in agent\_view}(p_j)$;
current_priority $\leftarrow 1 + p_{max}$;

図 4.5: メッセージに対する処理 (非同期弱コミットメント探索)

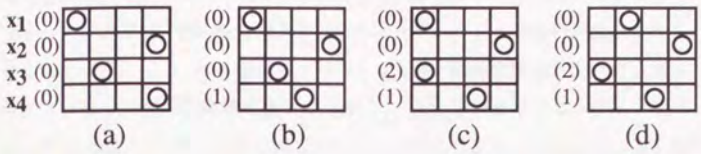


図 4.6: アルゴリズムの実行例 (非同期弱コミットメント探索)

実際には, x_1 の値が 1 の場合は制約を満たす値は存在しない. 非同期バックトラッキングアルゴリズムでは, 網羅的な探索を行わないと最も優先順位の高い x_1 の値は変更できないが, 非同期弱コミットメント探索アルゴリズムでは, 優先順位が変更されることにより, 網羅的な探索を行わずに値の変更が可能である.

4.4.4 アルゴリズムの完全性

非同期弱コミットメント探索アルゴリズムにおいては, 優先順位の変更は, 新しい nogood が得られた場合にのみ行われる. 可能な nogood の個数は有限であるため, 優先度の変更は無限に繰り返されることはない. このため, アルゴリズムを実行した場合には必ず, ある時点から以降には優先度の変更が生じないことになる. 以下, 非同期弱コミットメント探索アルゴリズムにおいて, ある時点から以降, 優先度の変更が生じない場合に, 以下の状況が生じないことを示す.

(i) あるエージェントにおいて制約が満足されないが, すべてのエージェントがメッセージ待ちの状態になる.

(ii) メッセージの送信が繰り返され, 永久に安定状態に達しない (処理の無限ループ).

まず, (i) の状況が生じている場合, 制約が満足されないエージェントが少なくとも 2 つ存在するため, 優先順位が k 番目のエージェントで, 優先順位がよ

り上位のエージェントとの制約が満足されず, 優先順位が 1,...,k-1 番目までのエージェントでは, 優先順位がより上位のエージェントとの制約が満足されているとしよう. ここで, k 番目のエージェントが, 制約を満足しないにも関わらずメッセージ待ちの状態に達するのは, 1,...,k-1 番目までのエージェントに対して nogood メッセージを送信している場合のみである. これは 1,...,k-1 番目までのエージェントで制約が満足されているという仮定に矛盾する. また, もっとも優先順位の高いエージェントで制約が満たされないことはあり得ない. よって, (i) の状況は生じ得ない.

また, 優先順位の変更が生じなければ, 非同期弱コミットメント探索アルゴリズムは, 非同期バックトラッキングアルゴリズムと基本的な振舞いは同一であり, 非同期バックトラッキングアルゴリズムの完全性より, 処理の無限ループは生じない. よって (ii) の状況も生じ得ない. このため, 非同期弱コミットメント探索アルゴリズムでは, すべてのエージェントにおいて制約が満足され, エージェントはメッセージ待ちの安定状態に達する (解が得られる), もしくは, あるエージェントで空集合の nogood が得られ, アルゴリズムが終了する.

4.4.5 アルゴリズムの計算量

制約充足問題は一般には NP 完全問題であり, 非同期弱コミットメント探索アルゴリズムの最悪ケースの計算時間, メッセージ数は, 共に変数の個数 n に対して指数的となる. また, 必要とされるメモリ量は, 記録される nogood の個数によって決まり, これも最悪ケースには変数の個数 n に対して指数的となる. 最悪ケースのメモリ量が指数的になってしまうことは, 完全性を保証しつつ, 探索の順序を柔軟に変更するためには避けられないことである.

現実的には, 弱コミットメント探索アルゴリズムと同様, 最も新しく得られた定数個の nogood のみを記録する方法が考えられる. この場合は, いくつかの nogood の間を巡回する無限ループに陥る可能性があり, 理論的なアルゴリズムの完全性は保証されないが, 実際の問題では非常に多くの nogood を巡回するループが生じることは稀であると考えられる. 実際, 各エージェントが保

存する nogood の個数を 10 個に制限した場合, 4.5節で実験を行った, すべての例題で無限ループに陥ることなく解が得られている.

4.4.6 関連する研究

[Sycara *et al.* 1991] では, texture と呼ばれる概念に基づいた, 分散スケジューリング問題のためのヒューリスティックスが議論されており, *dynamic search rearrangement* [Purdom 1983] に似た, より制約の強い部分から値の割当を行う方法が示されている. [西部ほか 1993] では, あらかじめ各変数の領域に関する情報を交換し, 前処理を行うことにより得られた統計的な情報を元に, 静的な順序付けを行う方法が示されている. これらの方法と比較して, 本論文で提案する非同期弱コミットメント探索は単純であり, 処理のオーバーヘッドが小さいことが予想される.

[Ferber 1989] では, *Computational Ecosystem* のエージェントの行動基準として, 他のエージェントとる状態のすべてと制約を満たす自分の状態が存在しない場合に, 最も古い時点で通信されたエージェントの状態を, 誤っているものと仮定して無視するという, 非同期弱コミットメント探索と似たアイデアが提案されている.

4.5 評価

本節では, これまでに示した種々のアルゴリズムの効率の比較を行う. 本論文では以下のような評価方法を用いる.

- 離散イベントのシミュレータを用いて, エージェントの並行動作のシミュレートを行う. 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, 送信されたいくつかのメッセージを受理し, 局所的な計算を行い, 他のエージェントに対してメッセージを送信するごとにクロックの値を一つ増加させる (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. このシミュレータ

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.

この評価基準での1サイクルは, 各エージェントが外界の状況を認識し, 認識した状況に対応する行動 (変数の値) を選択し, それを通信するという処理の単位に対応している. 分散アルゴリズムの評価基準として, 通信されるメッセージの数 (メッセージ複雑度) が用いられることが多い. しかしながら, 本論文で示した分散制約充足アルゴリズムでは, 複数のメッセージを並行して処理することが可能であり, メッセージの個数は実際の処理時間に直接的には影響しないため, メッセージの個数は評価基準として適切ではない.

以下, この評価基準を用いて, まず同期バックトラッキングと非同期バックトラッキングの比較を示し, 次に非同期バックトラッキングと非同期弱コミットメント探索アルゴリズムの比較を示す. さらに, 問題に関する情報を一つのエージェントに集中させて問題を解く集中型の解法と, 分散型の解法との比較検討を行う.

4.5.1 同期バックトラッキングと非同期バックトラッキングの比較

非同期バックトラッキングでは, 各エージェントは並行して処理を行うため, 同期バックトラッキングと比較して高速となることが予想されるが, どの程度の高速化が得られるかは, エージェント間の制約の強さが影響すると考えられる. すなわち, エージェント間の制約が弱い場合には, 各エージェントが並行して値の設定を行っても, 比較的容易に解に到達できると考えられるが, エージェント間の制約が強い場合には, 優先順位の高いエージェント間の制約が完全に満足されない限り, 優先順位の低いエージェントは制約を満たす値を設定できず, 結果的に非同期バックトラッキングアルゴリズムは逐次的なアルゴリズムに近くなってしまうことが予想される.

これらの仮説を検証するため, 第3章で示した分散 n-queens 問題を用いた実験的評価を行う. 各エージェントには一つのクイーンが割り当てられ, この割り当てられたクイーンを一つの列の中に置こうとする. よって, n-queens 問題は, n 個のエージェントのネットワークによって問題が解かれる. エージェント

間の制約は，それぞれのクイーンが取り合わないことである．n-queens 問題では，n が大きくなると変数／エージェント間の制約が相対的に弱くなるという性質がある．

図 4.7に，n を変化させた場合の同期バックトラッキングアルゴリズムと非同期バックトラッキングアルゴリズムと比較を示す．これらのアルゴリズムでは，各エージェントは優先順位の高いエージェントの設定した変数の値と制約を満たすものから，ランダムに値の選択を行っている．グラフ中には 100 回の試行の平均のサイクル数を示す．

予想通り，n が大きくなると非同期バックトラッキングと同期バックトラッキングの差は大きくなり，$n > 18$ の範囲では 2 倍程度のスピードアップが得られている[3]．

従来の分散協調問題解決の研究の多くは，各エージェントが，独立性の高い局所問題を解く場合を対象としている．この実験結果は，各局所問題が緩く結合され，独立性が高い場合には，各エージェントが非同期，並行的に処理を行うことが，逐次的なコントロールを行うよりも効率的であるという，具体的なアプリケーションに基づいた従来の分散協調問題解決の研究で得られていた結果と一致しており，分散制約充足問題という形式的な枠組を用いて，従来の分散協調問題解決の研究で得られた結果を定式化することができることを示している．

4.5.2 非同期バックトラッキングと非同期弱コミットメント探索アルゴリズムの比較

非同期バックトラッキングアルゴリズムと非同期弱コミットメント探索アルゴリズムの関係は，バックトラッキングアルゴリズムと弱コミットメント探索アルゴリズムの関係と同様であり，2.4.5節で示した確率モデルによる議論が同様に適用できる．よって，非同期弱コミットメント探索アルゴリズムでは一回の値の設定のミスが致命的とならず，非同期バックトラッキングアルゴリズム

[3]この評価では同期バックトラッキングでの順番を決定するためのコストは考慮されていない．

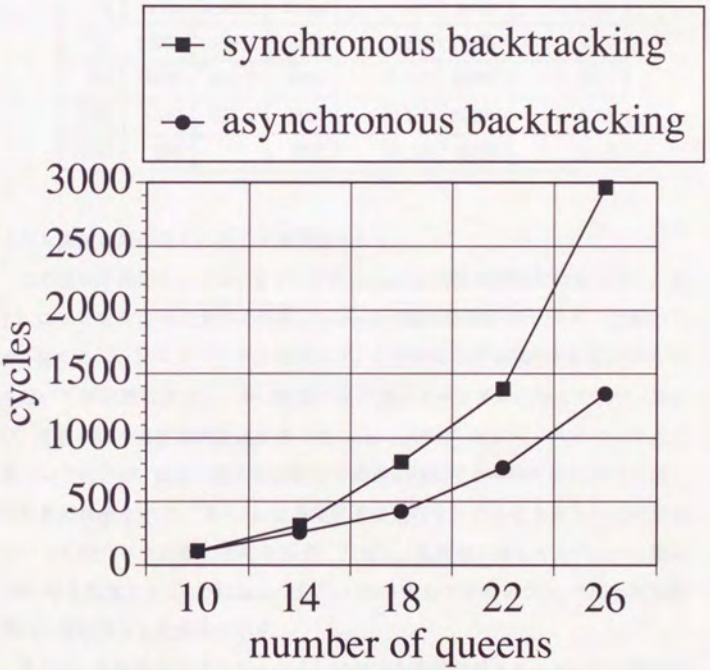


図 4.7: 非同期, 同期バックトラッキングの比較 (分散 n-queens)

表 4.1: 非同期バックトラッキング, 非同期弱コミットメント探索の比較 (分散 n-queens)

	asynchronous backtracking		min-conflict only		asynchronous weak-commitment	
n	ratio	cycles	ratio	cycles	ratio	cycles
10	100%	105.4	100%	102.6	100%	41.5
50	50%	662.7	56%	623.0	100%	59.1
100	14%	931.4	30%	851.3	100%	50.8
1000	0%	—	16%	891.8	100%	29.6

よりも高速に解が得られることが予想される.

この仮説を検証するため, まず, 分散 n-queens 問題を用いた評価を行う. 表 4.1 に n を変化させた場合の分散 n-queens 問題の実験結果を示す. 比較のため, 全くヒューリスティックスを用いず, 変数の値の変更順序を変数の識別子を用いて固定的に決定し, かつ変数の値の選択をランダムに行った場合, および, 変数の値の選択は制約違反最少化 (min-conflict) ヒューリスティックスに基づいて行うが, 変数の値の変更順序を変数の識別子を用いて固定的に決定した場合の結果を示す. 表 4.1には各項目 100 個のランダムに生成された初期値についてのサイクル数の平均を示す. ただし, 各問題に関して, サイクル数は 1000 回を限度として, それ以上のものは 1000 として平均をとり, 平均と制限時間内に解が得られた割合を示す.

さらに, 具体的なアプリケーションにおける非同期弱コミットメント探索の効果を調べるため, [西部ほか 1993] で用いられている通信ネットワークにおける分散資源割当問題での評価を行う. 図 4.8に, 分散資源割当問題の例を示す. この例では, NTT の全国の交換局から, 他の交換局に対して回線割当要求が発生しており, 各割当要求に対して, いくつかの回線割当の候補が与えられている. 目的は回線容量の制約を満たすように各要求に対する割当を決定することである. この問題は各割当要求を変数, 割当の候補を変数の取り得る値とおく

ことにより，制約充足問題として定式化でき，また，各割当要求に関して，その要求の処理を担当するエージェントが存在すると仮定すれば分散制約充足問題として定式化することができる．NTT 光ネットワークシステム研究所で作成されたネットワーク構成管理データベース [山口ほか 1989] を元に，10 個の回線接続要求があり，各要求に関して 50 通りの候補が与えられている問題を 10 個作成し[4]，これらの問題に関して，10 通りのランダムに発生させた初期値からスタートして問題を解いた結果の平均 (合計 100 個の平均) を表 4.2に示す．この問題においても，サイクル数は 1000 回を限度として，それ以上のものは 1000 として平均をとっている．

これらの実験により得られた知見は次の通りである．

- 予想通り，これらの例題に関しては，非同期弱コミットメント探索の効果は劇的であり，非同期バックトラッキングでは現実的な時間で解が得られないような大規模な問題でも，解を求めることが可能になっている．制約違反最少化ヒューリスティック単独では，若干の探索の削減効果はあるものの，大規模な問題には対応できない．

- 弱コミットメント探索を用いない場合，初期値の選び方によって，探索効率が大きく左右される．例えば，ネットワークの資源割当問題では，制約違反最少化ヒューリスティックのみを用いた場合，63% の，1000 サイクル以下で解が得られた場合のみの平均は 92.8 に過ぎないが，残りの 37% ではサイクル数が 1000 以上になってしまい，非常にばらつきが大きい．優先順位で上位の変数の初期値が，最終的な解の一部となる場合には比較的容易に解が得られるが，そうでない場合には，優先順位で上位の変数の値が解になり得ないことを示すのに多くの網羅的な探索が必要となり，サイクル数の上限を越えてしまう場合が多くなる．一方，弱コミットメント探索を用いた場合，初期値の選び方はクリティカルではなく，初期値が最終的な解から大きく隔たっている場合でも，変数の値は段階的に最終

[4]この候補は，要求を満足する経路のうち，通過する交換局の少ないものから 50 本が選ばれている．

表 4.2: 非同期バックトラッキング, 非同期弱コミットメント探索の比較 (ネットワークの資源割当問題)

asynchronous backtracking		min-conflict only		asynchronous weak-commitment	
ratio	cycles	ratio	cycles	ratio	cycles
32%	984.8	63%	428.4	100%	17.3

的な解に近付いていき, 制限時間内に解が得られている.

- 変数の優先順位がエージェントの権限の構造を示すと考えれば, 優先順位を変更しない場合の権限の構造は固定的であり, 優先順位が上位のエージェントにおける判断ミス (悪い初期値) は致命的である. 一方, 非同期弱コミットメント探索での権限の構造は動的に, より強く制約を受けているエージェントの優先順位が高くなるように変更され, ある特定のエージェントの判断ミスは全体に大きな影響を及ぼすことはない. 制約違反最少化ヒューリスティックは, エージェントの他のエージェントへの思いやりに対応すると考えることができ, 変数の優先順位を動的に変更することはエージェント間の公平性に対応すると考えることができる. このような直観的に自然な概念が, 現実にアルゴリズムの高速化を導くことは興味深い.

4.5.3 集中型の解法との比較

問題を分散した複数のエージェントで解くことに意味があるのか, 情報を一つのエージェントに集めて解く方が効率的ではないかという疑問は, 分散協調問題解決の研究に対して常に投げ掛けられる問いである. 分散制約充足問題に関して, 分散型の解法が正当化される状況として, 以下のような場合が挙げられる.

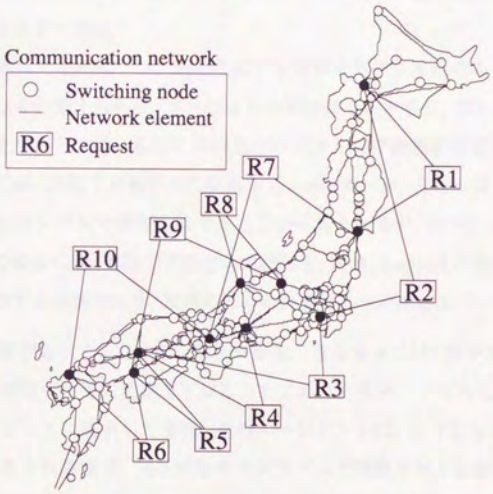


図 4.8: ネットワークの資源割当問題の例

各エージェントが問題に関する情報すべてを, 他のエージェントに対して公開することが望ましくない場合：

各エージェントが独立な組織／個人等を代表している場合には, セキュリティ, プライバシー等の観点から, 集中型の解法を用いることが不適切となる可能性がある. そのような場合には, 効率にかかわらず分散型の解法を用いざるを得ない.

情報を集中するためのコストが, 情報を集中することによって得られる効率化よりも大きい場合：

分散制約充足問題では, 制約に関する情報を集中するためのコストが問題になると考えられる. エージェントが均質である場合, 例えば同種の計算機上で動作する, もしくは同じプログラミング言語を実行可能である場合には, 述語で表現された制約を, binary データ, もしくはソースプログラムのレベルで情報交換することが可能となるが, エージェントが非均質な場合には, 述語で表現された制約を, 一旦, nogood の形式に展開して通信する必要があり, 展開のための計算コストが生じる.

一方, 情報を集中するためのコストが無視できる場合には, 集中型の解法は, 原理的に分散型の解法の性能を下回ることはありえない. すなわち, 情報を一つのエージェントに集中した場合, そのエージェントは, 必ずしも単一のプロセッサからなる計算機で, 逐次的なアルゴリズムで問題を解く必要はなく, 並列計算機を用いることも可能である. このため, 分散型の解法が非常に効率的であるとすれば, エージェント数と同じ数のプロセッサを持ち, プロセッサ間で非常に高速な通信が可能な並列計算機で同じアルゴリズムを実行することにより, 集中型で (並列計算機を用いて), 分散型のアルゴリズムと少なくとも同等な性能を得ることが可能となる.

非同期弱コミットメント探索アルゴリズムに関しては, 以下の２つの条件が満たされた場合, 情報を集中するコストを考慮しない場合でも, 集中型の解法と比較して大きな性能の差はないと考えられ, 情報を集中するコストが無視できない場合には, 集中型の解法より効率的となることが予想される.

1. 各サイクルでの計算時間が, 各サイクルでの通信に要する時間と比較して十分小さい.

2. サイクル数が集中, 逐次型の解法と比較して少ないか同程度である.

各サイクルでの通信に要する時間とは, エージェントが自分の値の変更と nogood に関する情報を送信し, 他のエージェントから送信された値の変更と nogood に関する情報を受信し, *agent_view* および *nogood_list* を修正するのに要する時間である. 1 番目の条件により, 各サイクルでの計算時間が通信に要する時間と比較して十分に大きければ, 通信に要する時間は処理時間全体に対して支配的なファクタとはならない. また, 分散型のアルゴリズムは, 処理の並列性を生かすことが可能であるが, 集中, 逐次型のアルゴリズムのように大局的な情報に基づく制御を行うことは難しく, 結果的に集中, 逐次型のアルゴリズムよりもサイクル数が多くなってしまう可能性があり, そのような場合には集中, 逐次型のアルゴリズムの方が明らかに効率的である.

以下, 弱コミットメント探索アルゴリズムと, 非同期弱コミットメント探索アルゴリズムに関して, これらの条件が成立するかどうかを検討する. 表 4.3に, n-queens 問題／分散 n-queens 問題を, それぞれ弱コミットメント探索アルゴリズム, 非同期弱コミットメント探索アルゴリズムで解いた場合のサイクル数, および各サイクルでの制約チェック数の合計[5]を示す. 表中にはランダムに生成した 100 通りの初期値に関する平均を示している[6].

表 4.3に示されているように, n=10 の場合は, 弱コミットメント探索アルゴリズムの方がサイクル数は少ないが, n=50 で同程度となり, n が大きくなるにつれてサイクル数の差が大きくなっている. この理由は, 同期バックトラッキングと非同期バックトラッキングの比較の場合と同様, n-queens 問題では, n が大きくなるにつれて変数／エージェント間の制約が相対的に弱くなることに

[5]非同期弱コミットメント探索アルゴリズムに関しては, 各サイクルに関して, 各エージェントの制約チェック数の最大値の合計を示す.

[6]初期値がランダムであるため, 初期値が最終的な解に比較的近い表 2.2の結果よりもサイクル数は多くなっている.

表 4.3: 弱コミットメント探索, 非同期弱コミットメント探索アルゴリズムの比較 (n-queens)

	weak-commitment		asynchronous weak-commitment	
n	cycles	checks	cycles	checks
10	32.5	2258.8	41.5	3447.1
50	55.1	98001.8	59.1	130846.2
100	90.0	629316.3	50.8	443899.4
1000	559.7	433266340.0	29.6	23342112.0

表 4.4: 弱コミットメント探索, 非同期弱コミットメント探索アルゴリズムの比較 (ネットワークの資源割当問題)

weak-commitment		asynchronous weak-commitment	
cycles	checks	cycles	checks
32.7	13685.4	17.3	7780.5

起因していると考えられる. また, n が大きくなるにつれて各サイクルでの処理時間 (制約チェック数) は, ほぼ n の自乗で増加していく.

また, 表 4.4に, 4.5.2節で示したネットワークの資源割当問題を, 弱コミットメント探索, 非同期弱コミットメント探索アルゴリズムで解いた場合のサイクル数, および各サイクルでの制約チェック数の合計を示す. この例題でも, 処理の並列性が生かせるため, 非同期弱コミットメント探索アルゴリズムの方が弱コミットメント探索アルゴリズムよりもサイクル数, 制約チェック数とも小さくなっている.

次に, 前述の 1 の条件について検討を行う. n-queens 問題に関しては, 制約チェック数は大きいが, 制約が単純で種々の最適化手法を用いることが可能で

あり, 弱コミットメント探索アルゴリズムの実際の計算時間は数秒程度とすることができる. このため, 分散 n-queens 問題では, 各サイクルでの通信に要する時間が, 各サイクルでの計算時間と比較して十分小さいという仮定は成立しないことが予想される.

以下, 現実のインプリメンテーションでの, ネットワークの資源割当問題に関する各サイクルでの通信コストと制約チェックの比について考察を行う. [Gray 1988] では分散システムにおける通信のコストに関する考察がなされており, LAN では通信を行うための CPU での処理時間が支配的であるのに対して, 広域ネットワーク (WAN) では帯域幅が支配的な要因となり, 現状 (1990 年の時点) では 100byte のメッセージを一つ送信するのに要する時間が, LAN では 0.27msec, WAN では 31msec 程度となるという考察がなされている. 広域ネットワークを前提とすると, ネットワークの資源割当問題では, 各サイクルでエージェントは, ほぼ n×2=20 回のメッセージの送受信を行うとすれば, 各サイクルで通信に要する時間は 600msec 程度となる. 一方, 通信ネットワークの資源割当問題では制約は複雑であり, 一回の制約チェックに要する時間は, SPARC station10 上の Lisp での実行時間で 10msec 程度である. よって, 各サイクルでの通信に要する時間は, おおむね 60 回の制約チェックに相当すると考えることができる. この問題では, 各サイクルでの制約チェック数は 4000 回程度であり, 各サイクルでの通信に要する時間は, 計算時間の 2% 以下となり, ほとんど問題とならないと考えられる.

図 4.9に, 一回の制約チェックに要する時間を一単位時間 (time unit) として, 各サイクルでの通信に要する時間を変化させた場合のネットワークの資源割当問題での弱コミットメント探索アルゴリズムと非同期弱コミットメント探索アルゴリズムの処理時間の比較を示す. 各サイクルでの通信コストが 60 単位時間である場合には, 非同期弱コミットメント探索アルゴリズムの性能は弱コミットメント探索アルゴリズムの性能を上回っていることが示されている.

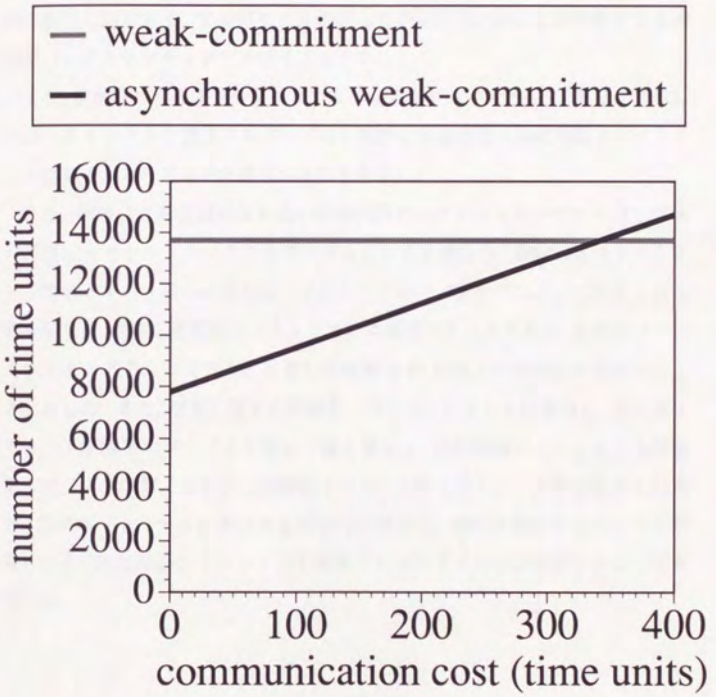


図 4.9: 通信コストの影響 (ネットワークの資源割当問題)

4.6 結言

本章では，分散制約充足問題の基本的な解法である分散探索アルゴリズムを示した．本章ではまず，バックトラッキングアルゴリズムを分散制約充足問題に適用することを検討し，各エージェントが全体的な制御なしに，非同期，並行的に動作しながらも，アルゴリズムの完全性が保証されることを特徴とする非同期バックトラッキングアルゴリズムを示した．

また，非同期バックトラッキングアルゴリズムをベースとして，第2章で示した弱コミットメント探索アルゴリズムと同様な性質を持つ非同期弱コミットメント探索アルゴリズムが実現されることを示した．

また，例題による実験結果を用いて非同期バックトラッキングアルゴリズムの同期バックトラッキングアルゴリズムに対する優位性，非同期弱コミットメント探索アルゴリズムの非同期バックトラッキングアルゴリズムに対する優位性を示した．特に，非同期弱コミットメント探索アルゴリズムは，非同期バックトラッキングアルゴリズムと比較して例題で20倍以上の高速化が得られることを示した．また，問題に関する情報を一つのエージェントに集中し，弱コミットメント探索アルゴリズムを用いて解く場合と，非同期弱コミットメント探索アルゴリズムで解く場合の比較検討を行い，広域ネットワークを前提とした場合，通信ネットワークにおける資源割当問題では，情報を集中するコストを無視しても，非同期弱コミットメント探索アルゴリズムの方が高速であることを示した．

第 5 章

分散 consistency アルゴリズム

5.1 序言

本章では consistency アルゴリズムの分散制約充足問題への適用方法を示す. 制約充足問題で提案されている各種の consistency アルゴリズムは, 各変数に関する処理が逐次的に行われるため, 単純に分散制約充足問題へ適用することは難しい. 一方, これらの consistency アルゴリズムと等価な, 仮説に基づく真偽値管理システムである ATMS(Assumption-based Truth Maintenance System) を用いた処理では, nogood が単調に増加し, 最終的に得られる結果は処理の順序に依存しないという, 分散制約充足問題に適用するのに望ましい性質を持っている.

本章ではまず, 各エージェントが独立な ATMS を持ち, 仮説に基づくデータと仮説間の制約条件 (nogood) を通信し合う, 分散協調問題解決のモデルである分散 ATMS について説明する. さらに, 分散 ATMS を用いた分散 consistency アルゴリズムの実現方法を示す.

5.2 分散 ATMS の概要

5.2.1 ATMS

ATMS は問題解決システムが参照するデータの真偽値の整合性管理を行う一種のデータベースであり, 次の 2 つによって特徴付けられる. 以下, それぞ

れについて説明する.

1. 複数の環境（environment）を持つことによる多重世界機能

2. 矛盾の効率的な回避

ATMS は複数の環境を管理する. 1つの環境はいくつかの仮説の組合せであり, 仮説の組合せの包含関係（サブセット, スーパセットの関係）によってラティスを作る（図 5.1）. データ α が環境 E の仮説 $\{H_1, H_2, \ldots, H_n\}$ から導かれるとき, データ α は環境 E で成立するという. 1つの環境は, その仮説の組がすべて真となるような1つの世界に対応すると考えられる. データ α が環境 E で成立するとき, α は E のスーパセットとなるような環境でも同様に成立する. このため, データ α が成立する, サブセット, スーパセットの関係にある複数の環境は, そのうちの最も小さい環境によって代表させることができる. ATMS はすべてのデータに, そのデータが成立する極小の環境の集合を関連付ける. この環境の集合をラベルと呼ぶ. ラベルが環境の集合であるのは, 同じデータが異なった仮説の組から導かれ得るためである. データが成立する環境をラベルという形で表現することにより, 環境間でのデータの共有が実現される. データ α がいくつかのデータ $\beta_1, \beta_2, \ldots, \beta_n$ を前提として導かれるとき, α のラベルは $\beta_1, \beta_2, \ldots, \beta_n$ のラベルから計算される. すなわち, α がどの世界で成立するかは, 前提が成立する世界によって決定される. このデータ間の依存関係, $\beta_1, \beta_2, \ldots, \beta_n \rightarrow \alpha$ をジャスティフィケーションと呼ぶ.

環境 E で矛盾が生じた場合, E は nogood（制約条件違反）として登録される. E のスーパーセットであるすべての環境も同様に nogood となる. nogood となった環境はすべてのデータのラベルから取り除かれる. このことは矛盾によって以前の推論結果が取り消されることを意味する. 問題解決システムは nogood となった環境では推論を行わない.

5.2.2 分散 ATMS

本論文で提案する分散協調問題解決のモデルでは, 複数のエージェントが独立に ATMS を持ち (図 5.2), 以下の情報を交換し合う[1]. これらの情報は各エージェントの管理する ATMS の環境に影響を与える.

推論結果とそのラベル： 各エージェントの推論結果は, ラベルが付けられて他のエージェントに送信される. 推論結果が送信され, その推論結果の成立する環境が受信側のエージェントにとって未知である場合には, 適切な新しい環境が作られる. 例えば図 5.1で, ある動物が ungulate (有蹄類) であるというデータが送信され, 受信側のエージェントがそのデータの成立する環境 $\{H_1, H_4\}$ を管理していなかった場合, 新たに環境 $\{H_1, H_4\}$ が作られる.

nogood： nogood が他のエージェントから送信された場合, 例えば図 5.1で, nogood$\{H_3\}$ が送信された場合, この nogood のスーパーセットの環境, すなわち $\{H_1, H_3\}$, $\{H_2, H_3\}$, $\{H_1, H_2, H_3\}$ は nogood となる.

ATMS を用いた推論を行い, 仮説に基づく推論結果と nogood を互いに通信し合うことにより, 各エージェントは, 他のエージェントの推論結果を利用し, 5.2.1節の ATMS の特徴を生かして推論を行うことが可能となる.

5.3 分散 ATMS を用いた分散 consistency アルゴリズム

各エージェントは次の処理を行うことにより, k-consistency を達成できる.

- 各エージェントは, 変数の取りうる値を仮説として宣言する.
- 互いに関連する変数を持つエージェント同士で, 仮説 (変数のとり得る値) をすべて通信しあう.

[1]同様なモデルが [Mason & Johnson 1989] によって提案され, 解釈型のエキスパートシステムへの適用が検討されている. 本研究は [Mason & Johnson 1989] とは独立に進められたものである.

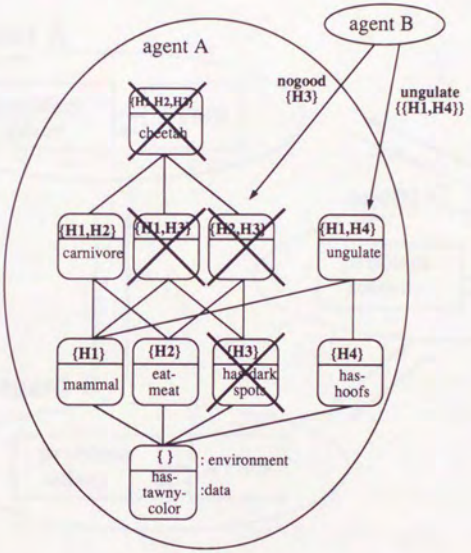


図 5.1: 環境のラティスとエージェント間通信

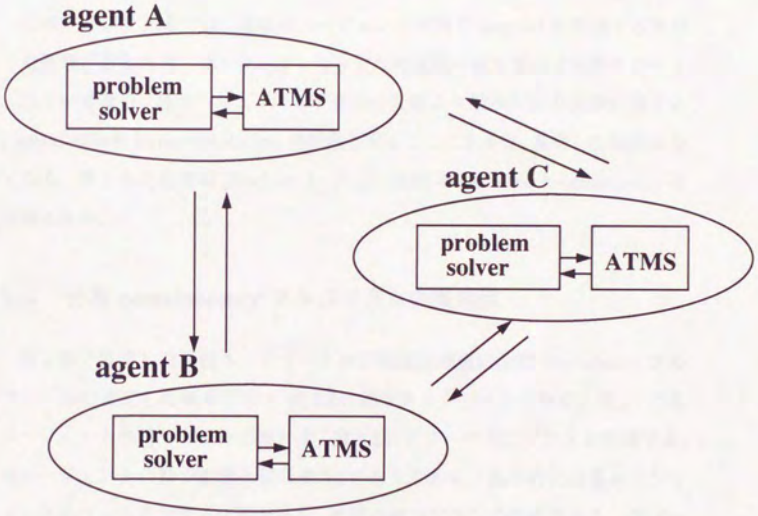

agent A
problem solver
ATMS
agent C
problem solver
ATMS
agent B
problem solver
ATMS

図 5.2: 分散 ATMS

- 制約を用いて nogood を生成し, hyper-resolution ルールを用いて, 新しい, 長さが k より小さい nogood を生成する. 生成した nogood を関連するエージェントに送信する.

- 新しく送信された nogood から, hyper-resolution ルールを用いて新しい nogood を生成する. この処理を新しい nogood が得られなくなるまで繰り返す.

このアルゴリズムでは, 複数のエージェントが同じ nogood を生成する重複した処理が行われる. バックトラッキングでの変数の値を変更する順序が与えられている場合, 各エージェントは, 自分の変数より順序が前の変数に関する nogood のみを hyper-resolution の対象とすることにすると, 重複した処理はなくなる. 得られた結果は [Dechter & Pearl 1988] の directed k-consistency と等価となる.

5.4 分散 consistency アルゴリズムの適用例

第 3 章で説明した通信ネットワークの資源割当問題に分散 consistency アルゴリズムを適用した例を示す. 図 5.3に通信ネットワークの例を, 表 5.1に各エージェントの持つプランの断片を, 表 5.2にグローバルなプランを再掲する. 各エージェントの持つ変数と値は表 5.3の通りである. 基本的には各エージェントは各ゴールに対する変数を持ち, 変数の値はプランの断片である. 各ゴールの起点となっているエージェント以外は, 必ずしもすべてのゴールに対してプランの断片を割り当てる必要はないため, ゴールに対して貢献しないという選択を表す idle という値を持つ.

各通信リンクは唯一の通信路のみを収容可能であると仮定すると, 同じエージェントの持つ, 同じ資源を用いる変数の値の間に制約が生じる. 例えば, プランの断片 1B と 2B は同じ資源を用いているため同時には成立し得ない. すなわち, {(B-goal1, 1B), (B-goal2, 2B)} は nogood である. 異なるエージェントの持つ変数間の制約は, プランの断片間の接続関係によって生じる. 例

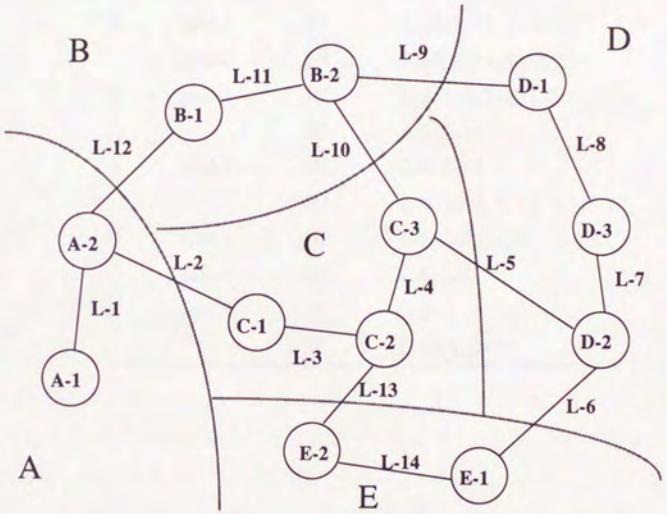


図 5.3: 通信ネットワークの例

表 5.1: プランの断片

Agent	Goal	Plan fragment	Resource Used
A	goal-1	1A	L-1, L-2
		2A	L-1, L-12
	goal-2	3A	L-12
B	goal-1	1B	L-10, L-11, L-12
	goal-2	2B	L-10, L-11, L-12
C	goal-1	1C	L-2, L-3, L-4, L-5
		2C	L-5, L-10
	goal-2	3C	L-5, L-10
		4C	L-4, L-10, L-13
D	goal-1	1D	L-5, L-7, L-8
	goal-2	2D	L-5, L-6
E	goal-2	1E	L-6
		2E	L-13, L-14

表 5.2: プランの候補

goal	plan	plan fragments
goal-1	plan-11	1A, 1C, 1D
	plan-12	2A, 1B, 2C, 1D
goal-2	plan-21	3A, 2B, 3C, 2D, 1E
	plan-22	3A, 2B, 4C, 2E

表 5.3: エージェントの持つ変数と値

agent	variable	value
A	A-goal1	1A, 2A
	A-goal2	3A
B	B-goal1	1B, idle
	B-goal2	2B, idle
C	C-goal1	1C, 2C, idle
	C-goal2	3C, 4C, idle
D	D-goal1	1D
	D-goal2	2D, idle
E	E-goal2	1E, 2E

えば, A がプランの断片 3A を選択した場合, B は 2B を選択する必要があり, {(A-goal2,3A), (B-goal2, idle)} は nogood である. エージェントが nogood を交換し, 強 2-consistency を達成する場合, エージェントは長さ 2 の nogood を用いて, 長さ 1 の nogood のみを新しく生成する. 例えば, 変数 A-goal2 の値は 3A のみであるため, nogood {(A-goal2,3A), (B-goal2, idle)} より nogood {(B-goal2, idle)} が得られ, B-goal2 の可能な値は 2B のみとなる. このため, nogood {(B-goal1,1B), (B-goal2, 2B)} より nogood {(B-goal1,1B)} が得られ, B-goal1 の可能な値は idle のみとなる. 以下同様に, 2A, 2C, 1D, 3C, 2D, 1E 等が nogood となり, 最終的には各変数の値は唯一に決定される.

5.5 評価

制約充足問題と同様, 分散制約充足問題においても分散 consistency アルゴリズムの効果は問題によって大きく異なり, 問題に応じて分散 consistency アルゴリズムとバックトラッキングアルゴリズムを適切に組み合わせることが重要である.

例えば, 8-queens では, 弱い consistency アルゴリズム (k が小さい k-consistency アルゴリズム) の適用は全く効果がなく, 新しい nogood は生成されない. 一方, k が大きい, より強力な consistency アルゴリズムは大量の nogood を生成する. 一方, 線画の認識問題で, 唯一の解が存在する場合は, 多くの場合 2-consistency (arc-consistency) を達成するだけで, 唯一の解が求められる. また, [Conry *et al.* 1991] に示されている通信ネットワークの資源割当の例題 (制約が強すぎて解がない) は, 本論文で示した分散 2-consistency を達成するアルゴリズムにより, 解が存在しないことがことが示される.

多くのランダムに生成した問題に関する実験結果から, 次のような傾向が示されている.

- 分散 2-consistency アルゴリズムによる無駄なバックトラックの削減効果は, 分散 2-consistency アルゴリズムの処理のコストとほぼ同等であり, $n > 2$ の分散 consistency アルゴリズムの処理コストは, バックトラックの削減効果を大きく上回る.

図 5.4に一例を示す. 図 5.4では, 乱数を用いて生成された問題に関して, 直接, 非同期バックトラッキング, 同期バックトラッキングで解いた場合と, 分散 2-consistency アルゴリズムを適用した後に, 非同期バックトラッキング, 同期バックトラッキングで解いた場合のサイクル数の比較が示されている. この問題は, 変数およびエージェントの個数 を 10 (各エージェントは唯一の変数を持つ), 変数の領域の大きさを 5 (すべての変数で同一), 制約の個数 15 (すべての制約は binary) を与えることにより, 制約がある変数の組が乱数により決定され, さらに, その変数の値の組に対して, 確率 0.4 で制約が成立するように, 乱数を用いて制約が生成されている (100 個の問題に関する平均を示す).

[Dechter & Meiri 1989] によれば, 通常の集中型の制約充足問題では, 2-consistency アルゴリズムは thrashing を減少させるために有効であるが, $n > 2$ である consistency アルゴリズムは処理のコストが thrashing の削減量を上回ることが報告されている. 一方, 分散制約充足問題では図 5.4に示されるように, 分散 2-consistency アルゴリズムはサイクル数を減少させるのにあまり有効で

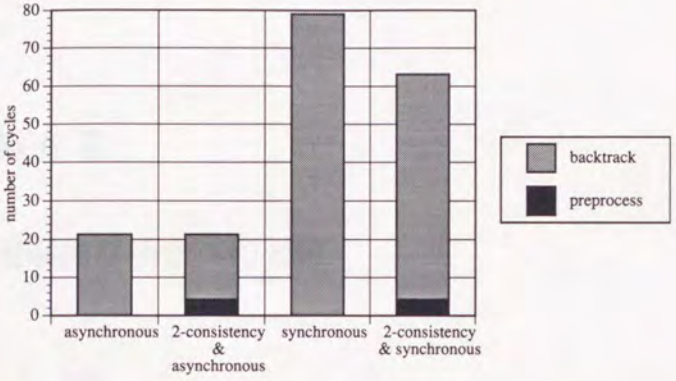


図 5.4: 分散 2-consistency アルゴリズムの効果

はない. この理由は次のように説明できる. 同期, または集中型のバックトラッキングでは, 変数の値は逐次的に決定される. 後の方で決定される変数に関して, 強い制約が存在する場合, そのような制約の影響を consistency アルゴリズムによりあらかじめ伝播することは, 無駄なバックトラックを減らすために非常に有効である. 一方, 非同期バックトラッキングでは, すべての変数の値は非同期, 並行して決定されるため, そのような強い制約が存在する場合, その制約の影響は直ちに伝播される. よって, 分散 consistency アルゴリズムの効果は, 同期バックトラッキングの場合と比較して相対的に小さくなる.

5.6 結言

本章では各エージェントが独立な ATMS を持ち, 仮説に基づくデータと, 仮説間の制約条件を通信し合う, 分散協調問題解決のモデルである分散 ATMS を示し, 分散 ATMS を用いた分散 consistency アルゴリズムの実現方法を示した. さらに, 非同期バックトラッキングによる探索と探索に先立つ前処理である分散 consistency アルゴリズムの適切な組合せに関して議論を行った.

第 6 章

過制約な場合への対応

6.1 序言

本章では, 分散制約問題において制約が強過ぎて解が存在しない場合に, 次善の解を求める方法を示す. 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 [Descotte & Latombe 1985; Freeman-Benson, Maloney, & Borning 1990; Fruder 1989], 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. 本章では, 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. 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.

6.2 過制約である分散制約充足問題の定式化

制約の重要度を導入した制約充足問題の定義は次の通りである.

- 変数の集合 $x_1, \ldots, x_n$ 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.
- 変数間の制約の集合が存在する.
- 制約 p_k に対して, その制約の重要性を示す正の実数値 r_k が定義される. この値を制約の重要度と呼ぶ (r_k は大きいほどその制約が重要であることを示す).
- 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 r 以上の制約をすべて満たすのに対し, S' が重要度 r 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.

過制約である分散制約充足問題は, 第 3 章と同様に変数と制約が複数のエージェントに分散された制約充足問題として定義される.

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, 問題のデザイナによって主観的な基準で与えられていることを仮定している. *locally-predicate-better* と呼ばれる同様な基準が [Freeman-Benson, Maloney, & Borning 1990] で, シミュレーションのグラフィカルディスプレイにおける制約に対処するために用いられている. もちろん, これ以外の解の基準, 例えば満たされる制約の重要度の合計を最大化する等の基準を用いることも考えられる. しかしながら, 通常は解の基準の詳細度と最適解を求めるための労力の間にはトレードオフが存在する. 本論文で用いる解

の基準は, 妥当な労力で直観的に自然な解を与えることができると考えられる. 6.5節で解の条件の一般化に関して議論する.

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, 前述の定式化における選好関係は, 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 6.1に問題の例を示す. この問題では, 2.4節で示した n-queens 問題で n が 3 の場合であり, 3×3 のチェスボード上の各列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る. この問題は, 各列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 $((x_i \neq x_j) \wedge (|x_i - x_j| \neq |i-j|)) \vee (x_i - x_j)^2 + (i-j)^2 > k$ の時に真となる.

図 6.1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 $p_k(x_i) \equiv x_i \leq k$ を導入する. k 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 k 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

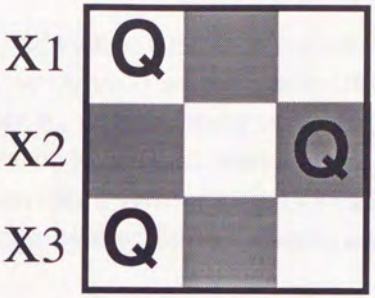


図 6.1: 3-queens 問題

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.

6.3 非同期段階緩和アルゴリズム

6.3.1 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. このアルゴリズムを非同期段階緩和アルゴリズムと呼ぶ.

- ある閾値 (初期値は適当な下界値か 0) を設定し, その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. 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内, 最も小さい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[1].

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.

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を可能とするアルゴリズムについて説明する.

6.3.2 nogood の依存関係を導入した非同期段階緩和アルゴリズム

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 6.1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, nogood N_k に対して, N_k の重要度を示す正の値 r 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 r であることは, 重要度が r 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.

[1]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.

- 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, 先述の 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 r であれば, 新しい閾値は r 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 r より小さく設定しても, すなわち, 重要度が r 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 r 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 6.1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は寄与しておらず, 新しい閾値は 1/4 となるべきことが分かる.

無駄な再計算を避ける:

閾値が変更され制約が緩和される際には, nogood の重要度を導入しない場合は, 閾値を変更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 6.1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.

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ち, 制約はすべて binary (2 変数間) であるという仮定をおく. 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 6.2 (i) にエージェント x_i で実行される初期化手続きを, 図 6.2 (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通りである.

- 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 6.2 (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.

- 第 4章の非同期バックトラッキングアルゴリズムと同様, 各制約に対して, 変数の識別子の辞書式順序により, 制約の評価を行うエージェントが決定され, 制約の方向が決定される (値を通信するエージェントから制約を評価するエージェントに向かうようにリンクの向きが決定される).

when initialized do — (i)
  select $d \in D_i$ where $F_i(\{(x_i, d)\}) \leq threshold$; — (i - a)
  $current\_value \leftarrow d$;
  send (**ok?**, (x_i, d)) to outgoing links; **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, importance_value*) **do** — (iii)
  add (*nogood, importance_value*) to *nogood_list*;
  when (x_j,d_j) where x_j is not connected is in *nogood* **do**
    request x_j to add a link from x_j to x_i;
    add (x_j,d_j) to *agent_view*; **end do**;
  $old\_value \leftarrow current\_value$;
  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**
    $threshold \leftarrow new\_threshold$;
    send (**revise_threshold**, x_i,*new_threshold*) to other agents except x_j;
  end do; **end do**;

procedure **check_agent_view** — (v)
  when $F_i(agent\_view \cup \{(x_i, current\_value)\}) > threshold$ **do**
    if there exists $d \in D_i$ where $F_i(agent\_view \cup \{ (x_i, d)\}) \leq threshold$ **then**
      $current\_value \leftarrow d$; — (v - a)
      send (**ok?**, (x_i, d)) to outgoing links;— (v - b)
    else backtrack; **check_agent_view**; **end if**; **end do**;

procedure **backtrack** — (vi)
  $V \leftarrow$ a subset of *agent_view* where $\min_{d \in D_i} F_i(V \cup \{(x_i, d)) > threshold$;
  $new\_importance\_value \leftarrow \min_{d \in D_i} F_i(V \cup \{(x_i, d)\})$ — (vi - a)
  if $V=\{\}$ **then** $threshold \leftarrow new\_importance\_value$;
      send (**revise_threshold**, x_i, *threshold*) to other agents; — (vi - b)
  else select (x_j, v_j) from V where j has the lowest priority;
      send (**nogood**, x_i, V, *new_importance_value*) to x_j;— (vi - c)
      remove (x_j, v_j) from *agent_view*; **end if**;

図 6.2: メッセージに対する処理 (非同期段階緩和アルゴリズム)

エージェントは, 外向きのリンクで結ばれた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 6.2 (i-b,v-b))).

- 一方, エージェントは内向きのリンクで結ばれたエージェントから送信された値を *agent_view* に記録する (図 6.2 (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 6.2 (v)).

- 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 6.2 (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通りである.

- 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 6.2 (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 がすべての制約を満足する場合には 0 を返し, その他の場合には, 満足されない制約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 6.2 (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 6.2 (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 6.2 (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.

アルゴリズムの実行例 図6.1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6.3では，エージェントをノードで，エージェント間の制約をリンクで表している．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 (問題の対称性より，この制限を行っても最適解は失われない)．

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 6.3 (a))．

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，他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 6.3 (b))．

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 6.3 (c))．

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 6.3 (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．

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により，本アルゴリズムの完全性も保証

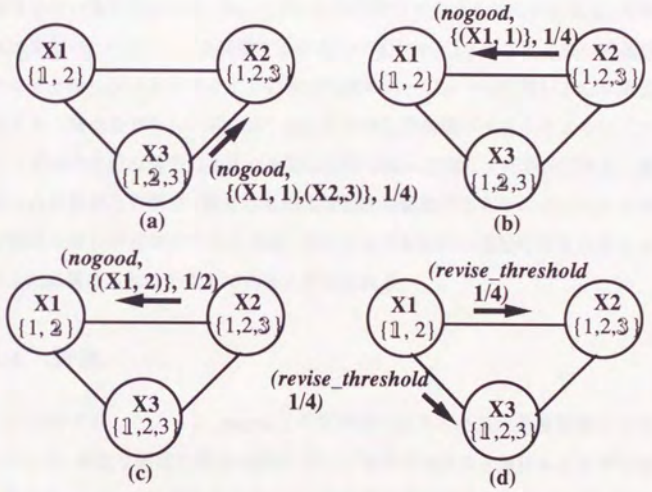


図 6.3: アルゴリズムの実行例 (非同期段階緩和アルゴリズム)

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，かつ新しい閾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，閾値は制約の重要度の最大値に達し，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．

本アルゴリズムの最悪ケースの計算量を考察するために，4.5節で示した評価基準を用いる．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サイクルとし，あるサイクルで送信されたメッセージは次のサイクル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サイクル数は変数の数 n 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．サイクル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ことであると考えられる．

6.4 評価

6.3.2節で示したように，nogood の重要度を導入した非同期段階緩和アルゴリズムは，無駄な閾値に関する探索，および無駄な再計算を避けることが可能になるため，nogood の重要度を用いない基本的な非同期段階緩和アルゴリズムよりも効率的となることが予想される．本節では以下，例題に関する実験結果により，この仮説を検証する．例題として，分散型の n-queens 問題において，変数の取り得る値を，$i = 1,\ldots,n/2$ に関しては $\{1,\ldots,n/2\}$ に，$i = n/2+1,\ldots,n$ に関しては $\{n/2+1,\ldots,n\}$ に制限し，過制約とした問題を用いる (n は偶数であることを仮定する)．

図 6.4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サイクル数を示す．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，必要とされるサイクル数は 1/5 程度に減少している．

図 6.5に，過制約である分散 12-queens 問題における，前述の 2 つのアルゴリズムにおける閾値の，サイクル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，サイクルが進むにつれて段階的に増加して閾値 $=1/72\approx 0.014$ 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 $=1/162\approx 0.0062$ 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 サイクルで終了しているが，基本 AIR では 249 サイクルが必要とされる．

6.5 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少化する解を与える．一方，例えば局

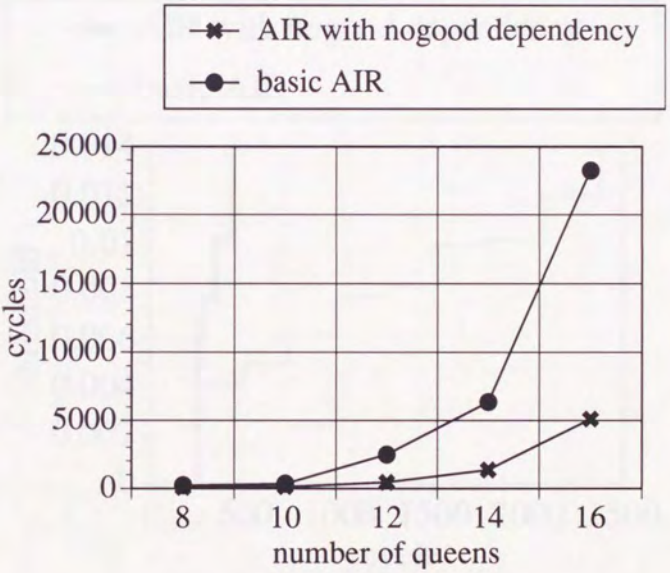


図 6.4: nogood の依存関係の効果 (分散 n-queens)

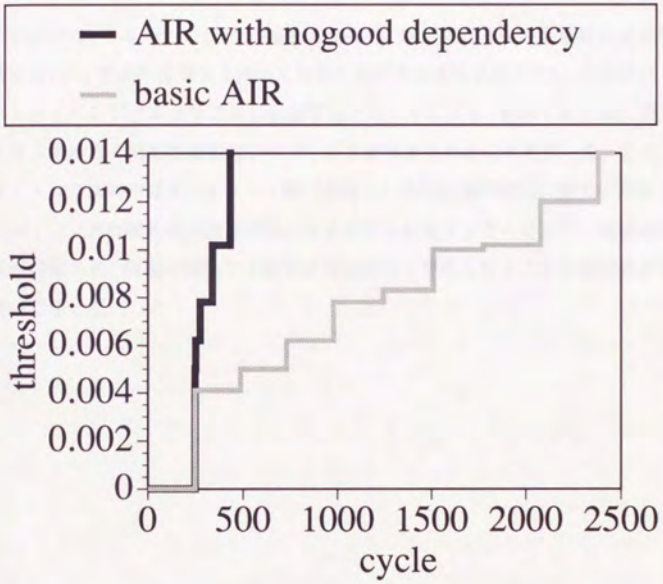


図 6.5: 分散 12-queens 問題での閾値の変化

所的な評価関数の合計値を最小化する場合には，現在の状態の評価値が閾値より悪いかどうかは局所的には判断できず，他のエージェントとの間で評価値の通信が必要となる．このように局所的な観点で枝刈りが可能であることが，本論文で用いる解の基準の一つの利点である．

6.6 結言

本章では，与えられた制約が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．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．

第 7 章

結論

本論文は，分散制約充足問題に関して筆者がこれまで行ってきた研究成果をまとめたものである．以下にその成果を要約し，最後に今後の課題を述べる．

第 1 章では，分散協調問題解決の研究の背景について述べるとともに，様々な応用問題を表現できる一般的な枠組の必要性を示した．次いで，制約充足問題を分散環境に拡張することにより，分散協調問題解決で議論されてきた様々な問題を定式化する枠組を与え，これらの問題を解く一般的な解法を提供するという本論文の基本方針を示し，論文の構成について述べた．

第 2 章では，分散制約充足問題の基礎となる制約充足問題に関して，従来の探索アルゴリズムを改良し高速化した，弱コミットメント探索アルゴリズムを示した．弱コミットメント探索アルゴリズムは，アルゴリズムの完全性が保証され，制約を満たす部分解を構成するため様々なヒューリスティックスの導入が可能であり，悪い決定を網羅的な探索なしに修正可能である等の，従来の探索アルゴリズムの問題点を克服し，長所を兼ね備えていることを示した．さらに，従来の探索アルゴリズムと比較して 3 倍から 10 倍程度の高速化が得られることを例題を用いた実験結果，確率モデルを用いた評価により示した．

第 3 章では，分散制約充足問題の定義を示し，この問題によって，解釈型，割当型の問題，複数知識ベース間の整合性管理等の様々な分散協調問題解決の応用問題が表現されることを示した．

第 4 章では分散制約充足問題の解を求めるための非同期バックトラッキング

アルゴリズムを示した. このアルゴリズムでは, 各エージェントは非同期, 並行的に, 全体をコントロールするエージェントなしに, 局所的な知識に基づいて動作し, かつアルゴリズムの完全性が保証されることを示した. また, 非同期バックトラッキングアルゴリズムをベースとして, 第2章で示した弱コミットメント探索アルゴリズムの分散版である非同期弱コミットメント探索アルゴリズムを実現する方法を示した. 例題による実験結果を用いて非同期バックトラッキングアルゴリズムの同期バックトラッキングアルゴリズムに対する優位性, 非同期弱コミットメント探索アルゴリズムの非同期バックトラッキングアルゴリズムに対する優位性を示した. 特に, 非同期弱コミットメントアルゴリズムは, 非同期バックトラッキングアルゴリズムと比較して例題で20倍以上の高速化が得られることを示した.

第5章では各エージェントが独立な, 仮説を用いたデータの真偽値管理システムであるATMSを持ち, 仮説に基づくデータと, 仮説間の制約条件を通信し合う, 分散協調問題解決のモデルである分散ATMSを示した. さらに, 分散ATMSを用いて探索の効率を高めるための前処理である分散consistencyアルゴリズムが実装されることをを示した. さらに, 非同期バックトラッキングによる探索と分散consistencyアルゴリズムの適切な組合せに関して議論を行った.

第6章では, 与えられた制約が強過ぎて解が存在しない場合に, 次善の解を求める方法を示した. 制約の重要度という概念を導入することにより, 制約充足問題の解の定義を拡張し, 部分的に制約を満たす解の評価基準を定式化した. また, 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することにより, 重要な制約をできる限り多く満足する最適な解を求める非同期段階緩和アルゴリズムが実現されることを示した. さらに, 探索途中で得られた新しい制約に関して, 最初に与えられた制約との依存関係を管理することにより, 無駄な探索を避けることが可能になり, 5倍程度の高速化が得られることを実験結果を用いて示した.

分散計算機環境の普及と, 人手によるプログラムの整合性の維持の困難さか

ら，様々な応用分野で“分散”と“協調”というキーワードが流行語のように用いられるようになっており，分散協調問題解決の研究に対する期待は非常に大きいものがある．それに対して，従来の分散協調問題解決の研究で得られた技術で，一般性を持ち，有効性およびその限界が明確化されたものは残念ながら多いとは言い難い．言うまでもなく，分散協調問題解決のすべての応用問題が分散制約充足問題で表現できる訳ではなく，分散制約充足問題で表現できる問題であっても，本論文で示した一般的なアルゴリズムがすべての場合で有効である訳ではない．しかしながら，本研究は分散協調問題解決において，十分大きな広がりを持つ応用分野をカバーする枠組を示し，一般的な解法を示している点で貴重な成果であり，今後の分散協調問題解決の研究の方向に大きなインパクトを与えるものであると確信している．実際，筆者の研究以降，分散探索／分散制約充足は，国際分散人工知能ワークショップで独立したセッションが設けられ，分散協調問題解決の重要な一分野となっている．

本研究の次のステップとして，分散制約充足問題の枠組のさらなる拡張，特に問題自身の動的な変化を導入することが挙げられる．現実世界でエージェントが直面する問題は，問題を解いている間にも刻一刻と変化する可能性がある．このような動的な変化を定式化し，一般的なアルゴリズムを開発することは，非常に長い歴史を持つ集中型の人工知能における探索／制約充足の研究においても未だに達成されていない困難な課題である．しかしながら，そのような動的な環境こそ，今後，分散協調問題解決の技術が最も必要とされる場面であり，本研究をベースとして新たな検討を続けていきたい．

謝辞

本論文をまとめるにあたり, 東京大学工学部電子情報工学科 渕 一博 教授には終始 懇切丁寧なる御指導と, 御教示を賜わりました. ここに謹んで深謝の意を表します. また, 内容について御指導, 御助言を賜わりました東京大学工学部電子情報工学科 斉藤 忠夫 教授, 石塚 満 教授に心から感謝致します. また, 東京大学工学部電気工学科 田中 英彦 教授には, 本研究を論文としてまとめるに際し, 少なからぬ御教授を賜わりました. 深く感謝致します. 東京大学工学部生産技術研究所 喜連川 優 助教授, 東京大学工学部電子情報工学科 相田 仁 助教授には有益な御討論, 御助言を賜わりましたことをここに記し, 心から御礼申し上げます. また, 東京理科大学基礎工学部電子応用工学科 伊藤 紘二 教授には, 有益な御教授, 御助言を頂くと共に, 本研究を論文としてまとめるにあたり激励の御言葉を頂きました. 心から感謝致します.

本研究は, NTT コミュニケーション科学研究所において, 多くの方々の御指導を得て行われました. 研究の機会を与えて頂くと共に, 日頃から御指導, 御援助を頂く 河岡 司 所長, 大里 延康 グループリーダに深謝致します. また, 本研究の期間中に御指導頂いた 大阪大学基礎工学部情報工学科 西川清史 教授 (前 NTT コミュニケーション科学研究所 所長), NTT コミュニケーション科学研究所 中野 良平 グループリーダに厚く御礼申し上げます. また, 本研究の期間中に, 直接の上司として御教授頂いた 京都大学工学部情報工学教室 石田 亨 教授 (前 NTT コミュニケーション科学研究所 主幹研究員) には, 本研究を進めるにあたり, 終始 丁寧なる御指導, 御助言を賜わりました. ここに深く感謝致します. また, また, ミシガン大学において御指導を頂いた Edmund H. Durfee 準教授に感謝致します. また, 日頃から御討論頂く 桑原 和宏 主任研究員, 松

原 繁夫 氏 をはじめとするNTTコミュニケーション科学研究所の皆様に感謝致します.

また, 本研究はその過程で, 様々な方々からプログラムやデータの提供を受けました. NTT 光ネットワークシステム研究所 藤井 伸朗 主幹研究員, 依田 育生 研究主任には, ネットワーク構成管理データベースを提供して頂きました. また, NTT 通信網研究所 西部 喜康 研究主任には, ネットワークの資源割当問題のデータを提供して頂くと共に, 様々な御助言を賜わりました. ここに記して深謝致します.

参考文献

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

[Burke & Proccer 1991] Burke, P. and Proccer, P.: A Distributed Asynchronous System for Predictive and Reactive Scheduling, *Artificial Intelligence in Engineering*, Vol. 6, No. 3, pp. 106–124 (1991).

[Chandy & Lamport 1985] 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).

[Collin, Dechter, & Katz 1991] Collin, Z., Dechter, R., and Katz, S.: On the Feasibility of Distributed Constraint Satisfaction, *IJCAI-91*, pp. 318–324 (1991).

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

[Davis & Smith 1983] Davis, R. and Smith, R. G.: Negotiation as a Metaphor for Distributed Problem Solving, *Artificial Intelligence*, Vol. 20, No. 1, pp. 63–109 (1983).

[de Kleer 1987] de Kleer, J.: Dependecy-directed Backtracking, in S.C.Shapiro, ed., *Encyclopedia of Artificial Intelligence*, pp. 47–48, Wiley-Interscience Publication, New York (1987).

[de Kleer 1989] de Kleer, J.: A Comparison of ATMS and CSP Techniques, *IJCAI-91*, pp. 290–296 (1989).

[Dechter & Meiri 1989] Dechter, R. and Pearl, J.: Network-based Heuristics for Constraint Satisfaction Problems, *Artificial Intelligence*, Vol. 34, No. 1, pp. 1–38 (1988).

[Dechter & Pearl 1988] Dechter, R. and Meiri, I.: Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems, *IJCAI-89*, pp. 271–277 (1989).

[Descotte & Latombe 1985] Descotte, Y. and Latombe, J.: Making Compromises among Antagonistic Constraints in Planner, *Artificial Intelligence*, Vol. 27, No. 1, pp. 183–217 (1985).

[Doyle 1979] Doyle, J.: A Truth Maintenance System, *Artificial Intelligence*, Vol. 12, pp. 231–272 (1979).

[Durfee & Lesser 1987] Durfee, E. H. and Lesser, V. R.: Using Partial Global Plans to Coordinate Distributed Problem Solvers, *IJCAI-87*, pp. 875–883 (1987).

[Ferber 1989] Ferber, J.: Eco Problem Solving, How to solve a problem by interactions, *9th Workshop on Distributed Artificial Intelligence*, pp. 113–128 (1989).

[Freeman-Benson, Maloney, & Borning 1990] Freeman-Benson, B., Maloney, J., and Borning, A.: An Incremental Constraint Solver, *Communications of the ACM*, Vol. 33, No. 1, pp. 54–62 (1990).

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

[Gasching 1977] Gasching, J.: A General Backtrack Algorithm That Eliminates Most Redundant Tests, *IJCAI-77*, pp. 457 (1977).

[Georgeff 1983] Georgeff, M.: Communication and Interaction in Multi-Agent Planning, *AAAI-83*, pp. 125–129 (1983).

[Ginsberg & Harvey 1990] Ginsberg, M. L. and Harvey, W. D.: Iterative Broadening, *AAAI-90*, pp. 216–220 (1990).

[Gray 1988] Gray, J.: The cost of messages, *7th ACM Symposium on Principles of Distributed Systems*, pp. 1–7 (1988).

[Haralick & Elliot 1980] Haralick, R. and Elliot, G. L.: Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence*, Vol. 14, pp. 263–313 (1980).

[Lesser & Corkill 1983] 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).

[Lesser & Erman 1980] Lesser, V. R. and Erman, L. D.: Distributed Interpretation: A Model and Experiment, *IEEE Transactions on Computers*, Vol. 29, No. 12, pp. 1144–1163 (1980).

[Mackworth 1992] Mackworth, A. K.: Constraint Satisfaction, in S.C.Shapiro, ed., *Encyclopedia of Artificial Intelligence*, pp. 285–293, Wiley-Interscience Publication, New York (1992), second edition.

[Malone *et al.* 1988] Malone, T., R.E.Fikes, Grant, and M.T.Howard: Enterprise: A Market-like Task Scheduler for Distributed Computing Environ-

ments, in B.A.Huberman, ed., *The Ecology of Computation*, pp. 177–205, North-Holland (1988).

[Mason & Johnson 1989] Mason, C. and Johnson, R.: DATMS: A Framework for Distributed Assumption Based Reasoning, in Gasser, L. and Huhns, M., eds., *Distributed Artificial Intelligence vol.II*, pp. 293–318, Morgan Kaufmann (1989).

[Minton *et al.* 1992] 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).

[Mitchell, Selman, & Levesque 1992] Mitchell, D., Selman, B., and Levesque, H.: Hard and Easy Distributions of SAT Problem, *AAAI-92*, pp. 459–465 (1992).

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

[西部ほか 1993] 西部喜康, 桑原和宏, 石田亨, 横尾真: 分散制約充足の高速化と通信網回線設定への適用, 電子情報通信学会論文誌, Vol. J76-D-II, No. No.10, pp. 2204–2213 (1993).

[Nudel 1983] Nudel, B.: Consistent Labeling Problems and their Algorithms: Expected Complexities and Theory-based Heuristics, *Artificial Intelligence*, Vol. 21, pp. 135–178 (1983).

[Pearl 1984] Pearl, J.: *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley (1984).

[Purdom 1983] Purdom, P.: Search Rearrangement Backtracking and Polynomial Average Time, *Artificial Intelligence*, Vol. 21, (1983).

[Rosenkrantz, Stearns, & Lewis 1978] Rosenkrantz, D., Stearns, R., and Lewis, P.: System Level Concurrency Control for Distributed Database Systems, *ACM Trans. on Database Systems*, Vol. 3, No. 2, pp. 178–198 (1978).

[Selman, Levesque, & Mitchell 1992] Selman, B., Levesque, H., and Mitchell, D.: A New Method for Solving Hard Satisfiability Problems, *AAAI-92*, pp. 440–446 (1992).

[Smith 1980] Smith, R. G.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, *IEEE Transactions on Computers*, Vol. 29, No. 12, pp. 1104–1113 (1980).

[Smith & Davis 1981] Smith, R. G. and Davis, R.: Frameworks for Cooperation in Distributed Problem Solving, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 11, No. 1, pp. 61–70 (1981).

[Sosič & Gu 1994] Sosič, R. and Gu, J.: Efficient Local Search with Conflict Minimization: A Case Study of the n-Queens Problem, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 5, pp. 661–668 (1994).

[Sycara *et al.* 1991] 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).

[Thorndyke, McArthur, & Cammarata 1981] Thorndyke, P., McArthur, D., and Cammarata, S.: AUTOPILOT: A Distributed Planner for Air Fleet Control, *IJCAI-81*, pp. 171–177 (1981).

[Waltz 1975] Waltz, D.: Understanding Line Drawing of Scenes with Shadows, in Winston, P., ed., *The Psychology of Computer Vision*, pp. 19–91, McGraw-Hill (1975).

[山口ほか 1989] 山口 治男, 藤井 伸朗, 山中 康史, 依田 育生: ネットワーク構成管理データベース, *NTT R & D*, Vol. 38, No. 12, pp. 1509–1518 (1989).

[Yokoo 1993a] Yokoo, M.: Constraint Relaxation in Distributed Constraint Satisfaction Problem, *5th International Conference on Tools with Artificial Intelligence*, pp. 56–63 (1993).

[Yokoo 1993b] Yokoo, M.: Dynamic Variable/Value Ordering Heuristics for Solving Large-Scale Distributed Constraint Satisfaction Problems, *12th International Workshop on Distributed Artificial Intelligence*, pp. 407–422 (1993).

[Yokoo 1994] Yokoo, M.: Weak-commitment Search for Solving Constraint Satisfaction Problems, *AAAI-94*, pp. 313–318 (1994).

[横尾 1994] 横尾 真: 弱コミットメント探索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).

[Yokoo & Durfee 1992] Yokoo, M. and Durfee, E. H.: Distributed Search Formalisms for Distributed Problem Solving: Overview, *11th International Workshop on Distributed Artificial Intelligence*, pp. 371–390 (1992).

[Yokoo *et al.* 1992] Yokoo, M., Durfee, E. H., Ishida, T., and Kuwabara, K.: Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving, *Proceedings of the Twelfth IEEE International Conference on Distributed Computing Systems*, pp. 614–621 (1992).

[横尾ほか 1992] 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).

[横尾, 石田 1990] 横尾 真, 石田 亨: ATMS を用いた分散制約充足問題の解法, 情報処理学会論文誌, Vol. 31, No. 1, pp. 106–114 (1990).

[Yokoo, Ishida, & Kuwabara 1990] Yokoo, M., Ishida, T., and Kuwabara, K.: Distributed Constraint Satisfaction for DAI Problems, *10th International Workshop on Distributed Artificial Intelligence* (1990).

本論文に関する原著論文

学術論文

1. 横尾 真, 石田 亨, "ATMS を用いた分散制約充足問題の解法," 情報処理学会論文誌, Vol. 31, No. 1, pp. 106-114, 1990.

2. 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. J75-D-I, No.8, pp. 704-713, 1992.

3. 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.

4. 横尾 真, "分散制約充足問題における制約緩和," 情報処理学会論文誌 (採録決定済み).

国際会議

1. Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara, "Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving," *12th International Conference on Distributed Computing Systems (ICDCS-92)*, pp. 614-621, 1992.

2. Makoto Yokoo, "Constraint Relaxation in Distributed Constraint Satisfaction Problem," *5th International Conference on Tools with Artificial Intelligence*, pp. 56–63, 1993.

3. Makoto Yokoo "Weak-commitment Search for Solving Constraint Satisfaction Problems," *12th National Conference on Artificial Intelligence (AAAI-94)*, pp. 313–318, 1994.

国際ワークショップ

1. Makoto Yokoo, Toru Ishida, and Kazuhiro Kuwabara, "Distributed Constraint Satisfaction for DAI Problems," *10th International Workshop on Distributed Artificial Intelligence*, Chapter 18, 1990.

2. Makoto Yokoo and Edmund H. Durfee, "Distributed Search Formalisms for Distributed Problem Solving: Overview," *11th International Workshop on Distributed Artificial Intelligence*, pp. 371-390, 1992.

3. Makoto Yokoo, "Dynamic Variable/Value Ordering Heuristics for Solving Large-Scale Distributed Constraint Satisfaction Problems," *12th International Workshop on Distributed Artificial Intelligence*, pp. 407-422, 1993.

研究会, 大会等

1. 横尾 真, 石田 亨, "分散問題解決における ATMS の利用," 第 2 回人工知能学会全国大会, pp. 141-143, 1988.

2. 横尾 真, "分散 ATMS を用いた探索問題の解法," 第 3 回人工知能学会全国大会, pp. 167-170, 1989.

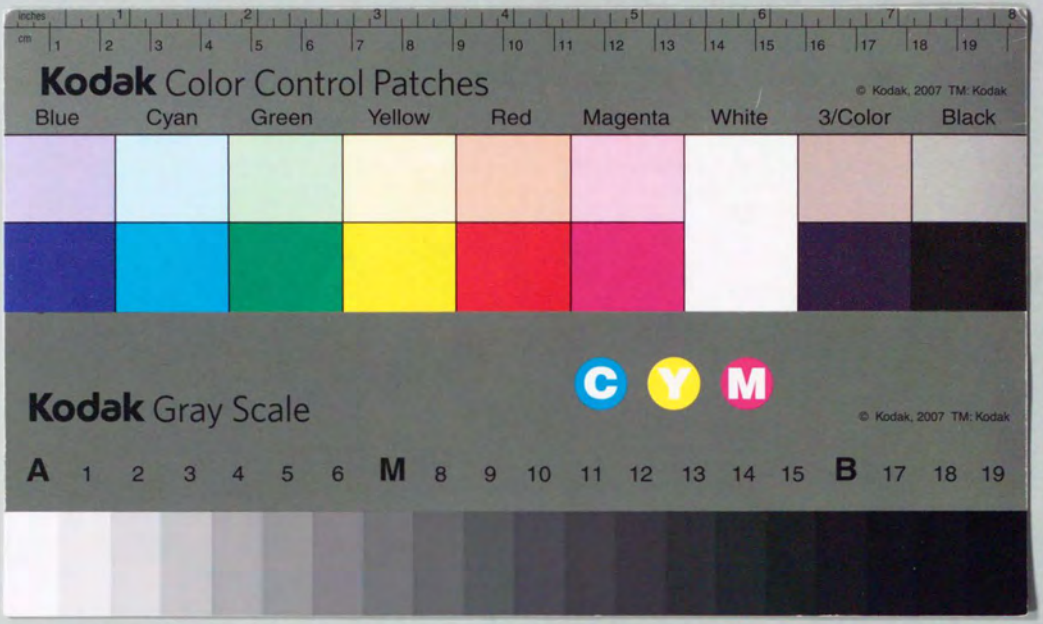
3. 横尾 真, "分散制約充足アルゴリズム," 電子情報通信学会コンピューティング基礎研究会, COMP89-90, pp. 9-16, 1989.

4. 横尾 真, "分散制約充足とその周辺 - 分散人工知能の基盤技術 -" 電気関係学会東海支部連合大会シンポジウム講演, 1993.

5. 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”, 電子情報通信学会 技術研究報告, Vol.93, No.424, 1994.

図一覧

表一覧


inches
cm
Kodak Color Control Patches
© Kodak, 2007 TM: Kodak
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
C
Y
M
Kodak Gray Scale
© Kodak, 2007 TM: Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19