分散不完全制約充足問題
Distributed Partial Constraint Satisfaction Problem*

平山 勝敏 † 横尾 真 ‡

† 神戸商船大学
〒 658 神戸市東灘区深江南町 5-1-1
E-mail: hirayama@ti.kshosen.ac.jp
‡NTT コミュニケーション科学研究所
〒 619-02 京都府相楽郡精華町光台 2-2
E-mail: yokoo@cslab.kecl.ntt.co.jp

概要

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

1 はじめに

AI における多くの問題は制約充足問題（Constraint Satisfaction Problem: CSP）として記述できるため，これまでに多くの研究者が，その性質やアルゴリズムに関する理論的，実験的な研究を行なってきた．しかしながら，近年，AI がより現実的な問題を扱う機会が増えるにつれ，CSP として記述できる問題にも限りがあることが指摘され，その枠組を拡張する試みがなされている．

[14] では，計算機ネットワーク上の複数のエージェントが関わる問題を記述する一般的な枠組として分散制約充足問題（distributed CSP）が提案されている．分散 CSP では，CSP における変数および制約が複数エージェントに分散されており，エージェントが互いに情報交換しながら，すべての制約を満たすことが要求される．マルチエージェントシステムにおける多くの問題，例えば，分散解釈問題 [8]，分散資源配分問題 [3]，分散スケジューリング問題 [10]，マルチエージェント TMS[6] などは分散 CSP として記述することができる．

一方，デザイナが現実の問題を CSP として記述する際，制約が強過ぎて解がないという場合がよくあるが，このような過制約な CSP に対し，従来の制約充足アルゴリズムは，ただ解がないことを発見するのみであった．そのため，多少の制約は違反してよいから何らかの値が欲しい場合には，デザイナが不要な制約を削るなどして，CSP が過制約にならないよう注意深く記述する必要がある．このような過制約な CSP を扱う枠組の一つとして，[4] では，不完全制約充足問題（Partial CSP）が提案されている．不完全 CSP では，もとの過制約な CSP の制約を許容できる範囲内で緩和し，その緩和して得られた問題の解を得ることが要求される．

マルチエージェントシステムにおける問題においても，それを分散 CSP として記述した場合に過制約となることは十分に考えられる．そのため，分散 CSP

*The english version of this material is published in *Principles and Practice of Constraint Programming – CP97*, LNCS 1330, pp.222–236 (Springer-Verlag)

に不完全 CSP のアイデアを導入して，過制約な分散 CSP を扱うことができるように枠組を拡張することは有益であると考えられる．本論文では，過制約な分散 CSP を扱うことができる分散不完全制約充足問題（Distributed Partial CSP）を提案する．また，分散不完全 CSP の特殊な場合として，分散最大制約充足問題（Distributed Maximal CSP）を提案し，それを解くアルゴリズムとして，同期型分枝限定法，反復分散ブレイクアウト法を提案する．

次節以降の本論文の構成は次の通りである．2節と3節で分散 CSP と不完全 CSP の概要を説明する．また，4節で分散不完全 CSP を定義し，次にその構成要素を特定して分散最大 CSP という部分クラスを導入する．5節では分散最大 CSP を解くアルゴリズムを与え，6節でそれらのアルゴリズムを実験で評価する．最後に7節をまとめとする．

2 分散制約充足問題

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

一方，分散 CSP は，CSP における変数や制約が複数エージェントに分散された問題だと見なすことができる．形式的には次のように定義される [14]．

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

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

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

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

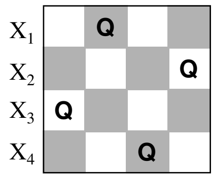

図 1: Example of CSPs（4-queens problem）

図 1は，CSP の例として良く用いられる n-queens 問題で n=4 とおいた場合であるが，各クイーンに対応する独立なエージェントが存在し，それらのエージェントが自分のクイーンの位置を決定しようとしていると考えれば，この問題は分散 CSP として記述できる．

3 不完全制約充足問題

不完全 CSP は，過制約な CSP を扱う枠組の一つであり，形式的には次のように記述される [4]．

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

ここで，P は，一般に過制約で解がない CSP である．U は，P の変数の潜在的な値の集合でユニバース（universes）という．$(PS, \leq)$ は問題空間（problem space）とよばれ，PS は，一般に P を何らかの方法で緩和した CSP の集合で，その要素間には半順序 $\leq$ が導入されている．また，問題空間上では，PS の要素である CSP と P との間の距離を測る距離関数（distance function）M が定義され，さらにその距離に関して必要値（necessary bound）N と充分値（sufficient bound）S が定義される．不完全 CSP の解とは，PS の要素である CSP のうち，P との距離が必要値 N 未満の充足可能な CSP とその解のことである．また，それらの解のうち P との距離が S 以下であるものが見つかると探索を終了することができる．なお，P との距離が最小となる解を不完全 CSP の最適解といい，そのときの距離を最適距離という．

不完全 CSP の重要な部分クラスの一つに最大 CSP（Maximal CSP）がある．最大 CSP では，P から制約を取り除いて得られる可能な CSP の集合が問題空間となり，取り除いた制約の数がその問題と P の距離となる．最大 CSP の最適解を求めることは，取り除く制約の数が最小である充足可能な問題とその解を求めることだが，これは，P の変数への割り当てのうち，違反する制約の数が最小のものを求めることに相当する．

4 分散不完全制約充足問題

4.1 動機

マルチエージェントシステムにおける問題においても，それを分散 CSP として記述した場合に過制約とな

ることは十分に考えられる．

例えば分散解釈問題[8]では，個々のエージェントがセンサデータの一部を担当して可能な解釈を生成し，それらを交換しながら全体として無矛盾な解釈を構成する．しかし，センサデータのノイズ等の原因によりエージェントが誤った解釈を生成し，そのため全体として無矛盾な解が存在しないという可能性がある．

また，分散資源配分問題を解く手法の1つであるマルチステージネゴシエーション[3]では，エージェントがゴール（変数）とゴールを達成するための複数のプラン（値域）をもち，また，プラン実行時に必要な資源の利用に関してエージェント間で競合が存在する（制約）という設定のもとで，全エージェントのゴールを達成するプランを求める．この場合，エージェントのゴールや資源の状態によっては，過制約な問題になる可能性がある．

分散CSPを解く従来のアルゴリズム[14][13][5][15]は，制約を全て充足する変数への値の割り当てを解として求めるが，そもそも問題にそのような割り当てがない場合には，単にその事実を発見するのみであり，変数への値の割り当てに関しては何も保証しない．そのため，多少の制約は違反しても構わないから何らかの値が欲しい場合には，従来の分散CSPの枠組では不十分である．

[12]では，過制約な分散CSPに対し，制約の重要度を反映した制約階層[1]を導入し，充足可能な最適な階層を求めるという問題が提案されている．それに対して，本論文では，[12]の問題も扱える、より一般的なアプローチとして分散不完全制約充足問題（Distributed Partial CSP）を提案する．[12]の問題や後述する分散最大CSPは分散不完全CSPの部分クラスとして位置付けることができる．

4.2 問題の定義

分散不完全CSPは，次のように記述される．

- エージェントの集合 $1, 2, \ldots, m$ が存在する．
- 任意のエージェント i は不完全CSP:

  $$\langle (P_i, U_i), (PS_i, \leq), (M_i, (N_i, S_i)) \rangle$$

  をもつ．
- 個々のエージェントの距離から，エージェント全体の距離を求める大域距離関数 $G : \Re^m \rightarrow \Re$ が定義される

ここでエージェント i に対する不完全CSPのうち，(P_i, U_i) はそれぞれ，もとの過制約なCSPとその変数に対するユニバース，また，$(PS_i, \leq)$ は問題空間，$(M_i, (N_i, S_i))$ はそれぞれ，距離関数と P_i との距離の必要値と充分値である．これらはエージェントごとに設定される以外は，3節で説明した通りである．

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

本論文では，分散不完全CSPのいくつかの設定を次のように具体化した分散最大CSP（Distributed Maximal CSP）を導入する．任意のエージェント i について，

- P_i からいくつかの制約を除いたCSPが PS_i の要素となる．可能な取り除き方すべてについて PS_i の要素が対応する．
- PS_i 内の問題 P'_i と P_i の距離は，除いた制約の数である．
- エージェント i の距離を d_i としたとき，$G(d_1, d_2, \ldots, d_m) = \max_i d_i$ である．

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

分散最大CSPは，潜在的な応用の可能性やアルゴリズム設計の容易さの点からも重要な問題だと考えられる．もちろん，他のアプローチとして，例えば，分散最大CSPの他の設定はそのままにして，大域距離関数を $\max_i d_i$ ではなく $\sum_{i=1}^{m} d_i$ とした問題も考えられる．しかし，この問題には，制約違反数の和は最小だが，違反した制約が特定のエージェントに集中するような最適解を許したり，効率のよいアルゴリズムの設計が困難である，などの問題点がある．

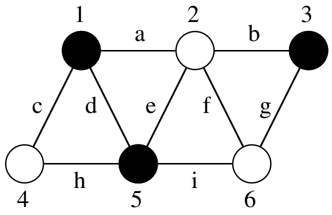

図 2: Distributed 2-coloring problem

4.3 例

図2の分散色塗り問題を用いて，分散最大 CSP を具体的に説明する．この図でグラフの頂点は変数（エージェント），辺が変数間の制約を表し，互いに辺でつながっている頂点どうしが異なる色になるように，全体を黒と白の 2 色で塗り分ける．エージェントは 1 つの頂点を担当するが，その頂点につながっている辺で表される制約のみを知っているとする．例えばエージェント 1 は制約 $\{a, c, d\}$ のみを知っている．この場合，エージェント 1 の最初の問題 P_1 は，値域が｛黒，白｝である変数 1 と制約 $\{a, c, d\}$ である．変数への値の割り当てが図 2の通りであるとき，エージェント 1 の制約違反数は 1 なので現在の距離は 1 になる．他のエージェントについては，エージェント 2，エージェント 5，エージェント 6 の距離が 1，エージェント 3 とエージェント 4 の距離は 0 である．今，すべてのエージェントの距離の必要値 N_i が 2 だとすると，図 2の割り当てで分散最大 CSP の解が得られている．しかし，1 だとすると，エージェント 3 とエージェント 4 以外のエージェントの距離が 1 未満でないため，この割り当てで解は得られていない．なお，この割り当てでのエージェント全体の距離は，個々のエージェントの距離の最大値で 1 になる．この例では，エージェントがどのように値を割り当てても全体の距離を 1 未満にすることはできないので，この割り当ては最適解を与え，最適距離は 1 になる．

5 分散最大制約充足アルゴリズム

次に，分散最大 CSP の最適解を求めるアルゴリズムを開発する[1]．

もっとも単純なアルゴリズムとしては，集中型のアルゴリズムが考えられる．集中型のアルゴリズムでは，まずエージェント間でリーダーを選出し，すべてのエージェントの問題をリーダーに集め，リーダーがそれらの問題を最大 CSP を解く要領で解く．集中型のアルゴリズムは大域的な情報を参照できるという利点があり，効率面では優位となる可能性がある．しかし，マルチエージェント環境においては，問題を集めるコストが問題になるとともに，プライバシーやセキュリティの観点からも，特に個人に関する情報を含むような問題（個人のスケジュールなど）を集めることは望ましいとはいえない．そこで本論文では，問題が分散された状態のまま実行される分散型のアルゴリズムとして，同期型分枝限定法（Synchronous Branch and Bound）と反復分散ブレイクアウト法（Iterative Distributed Breakout）を提案する．

5.1 仮定

アルゴリズムの設計に際して，次のようなエージェント間通信のモデルを想定する．これらは，一般の非同期通信システムとして妥当なものである．

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

さらに，対象とする問題のクラスを次のように限定する．これは，アルゴリズム設計の単純化のために設けたものであり，このクラスの問題に対して設計したアルゴリズムをより広いクラスの問題に対応できるように拡張することは容易である．

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

5.2 同期型分枝限定法

同期型分枝限定法は，分枝限定法を複数エージェントでシミュレートしただけの最も素朴なアルゴリズムである．このアルゴリズムでは，前もって変数順序（エージェント順序）と値順序が固定され，パスとよばれる一つの部分解（全変数の部分集合に対する値の割り当て）をエージェント順序に従って順番に送り，徐々にそれを伸ばしていく．動作の概略は次の通りである．

- エージェント順序で 1 番目にあたるエージェントがアルゴリズムを起動し，自分がもつ変数に対して 1 番目の値をパスとして，次のエージェントに送る．
- エージェント順序が 1 つ前であるエージェントからパスを受け取ると，それを保存し，そのパスと自分の変数の 1 番目の値を合わせて評価して，エージェント全体の距離の下界値を見積もる．下

[1]最適解を求めるために，任意のエージェント i について $S_i = 0$ が成り立つと仮定する

界値が現時点での上界値未満であれば，パスとその値を新たなパスとして次のエージェントに送る．上界値以上であれば次の値で同様の評価を行うが，これを繰り返して値が尽きれば前のエージェントにパスを送り返す．

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

パスとは，実際には，変数，変数の値，その値が原因で生じている違反制約の数の 3 つ組のリストである．よって，エージェント全体の距離の下界値は，パス内の変数ごとに求められている違反制約の数のうちの最大値として求められる．また，上界値は過去に得られた全体解の評価値（エージェント全体の距離）のうちの最小値として求められる．アルゴリズムの詳細を図 3に示す．

同期型分枝限定法は，エージェント間で逐次的に分枝限定法の動作をシミュレートし，複数エージェントに分散している探索空間をしらみ潰し的に探索する．そのため，必ず最適解を見つけるというアルゴリズムの完全性は保証される．しかし現状では，一つのパスを伸ばしていくだけであり，パスを受け取ったエージェント以外は何の動作も行わないため，複数エージェントによる並列実行の利点は生かされていない．

5.3 反復分散ブレイクアウト法

5.3.1 分散ブレイクアウト法

分散ブレイクアウト法 [15] は分散 CSP を解くアルゴリズムで，その特徴は，複数のエージェントが互いの動作を相互排除しながら並列に山登り法 [5] を実行すること，エージェントが疑似局所最適解に陥ったときの手続きとしてブレイクアウト法 [9] を採用していることの 2 点である．

分散ブレイクアウト法では，エージェントは最初に自分の変数に任意の値を割り当て，その割り当てを近傍（同じエージェント間制約を共有しているエージェントの集合) に $ok?$ メッセージとして送る．以降，エージェントは次の手続きを繰り返す．

- 近傍の全エージェントから $ok?$ メッセージを受け取ると，自分がもつ変数の現在の割り当てを評価する．評価値（コスト）は，その割り当てのもとで違反している制約の重み和とする．ここで制約の重みとは，制約ごとに定義された整数で，その初期値は 1 である．また，別の値に割り当てを変更した場合のコストの改善量の最大値（$improve$ とよぶ）を計算し，その値を $improve$ メッセージとして近傍に送る．

- 近傍の全エージェントから $improve$ メッセージを受け取ると，それぞれの $improve$ を自分のものと比較し，もしも相手の値の方が大きければ，自分の次回の割り当て変更は行わない．また，$improve$ の値が同じ場合は，両エージェントの識別子を用いてタイブレイクする．近傍の全エージェントとの比較が終った段階で，割り当て変更が可能なエージェントは変数の割り当てを変更する．そして割り当てを変更しなかったエージェントも含めて，すべてのエージェントが現在の割り当てを近傍に $ok?$ メッセージとして送る．

これを繰り返すだけで分散 CSP の解が求められることもあるが，多くの場合，あるエージェントが疑似局所最適解に陥る．疑似局所最適解とは，あるエージェントの現在の割り当てのコストが 0 より大きく（つまり，違反制約が 1 つ以上あり），しかも，割り当てをどう変更してもコストが下がらない状態をいう．分散ブレイクアウト法では，疑似局所最適解にいるエージェントが割り当てを変更する場合，その時点で違反している制約の重みを上げ，以降に同じ状態になったとしても，そこからすぐに出られるように評価関数を修正しておく．

分散ブレイクアウト法では，あるエージェントが割り当てを変更する際，近傍の動作は抑制されるが，それ以外のエージェントの動作は抑制されないので，個々の動作の確実性を保証しながら，全体としては並列に割り当てが変更される．そのため，解が存在する非常に困難な問題に対して，特に効率がよいことが示されている [15]．また，解を発見したことを検出するアルゴリズムが埋め込まれていることも分散ブレイクアウト法の利点である．従来のアルゴリズムでは，その検出に，分散スナップショット [2] などの別の手続きを呼び出すことが前提にされていた．一方，分散ブレイクアウト法の欠点は完全性が保証されないことである．すなわち，解が存在したとしてもそれを見つけることができるとは限らないし，また，解がないことを判定できない．

5.3.2 反復分散ブレイクアウト法

反復分散ブレイクアウト法では，分散最大 CSP に対し，分散ブレイクアウト法を繰り返し適用する．基本的な動作は次の通りである．まず，任意のエージェント i の不完全 CSP の必要値 N_i に共通の値 ub を代入し，分散ブレイクアウト法を実行する．この分散ブレイクアウト法は，それぞれのエージェントの距離が N_i 未満になると終了し，そのことをあるエージェントが検出する．検出したエージェントは次の N_i の値を

```
procedure initiate
    d_i ⟵ first value in domain;
    send (token, [[x_i, d_i, 0]], n_i) to the next agent;

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

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

procedure send_token
    if next ≠ 'exhausted' then
        if i = the last agent then
            next_to_next ⟵ next;
            while next_to_next ≠ 'exhausted' do
                best_path ⟵ new_path;
                n_i ⟵ max nv_j in best_path;
                when n_i ≤ s_i do
                    terminate the algorithm; end do;
                next_to_next ⟵ get_next(domain minus all elements up to next_to_next);
            end do;
            send (token, previous_path, n_i) to the previous agent;
        else
            send (token, new_path, n_i) to the next agent; end if;
    else
        if i = the first agent then
            terminate the algorithm;
        else
            send (token, previous_path, n_i) to the previous agent; end if; end if;

procedure get_next(domain)
    if domain = nil then
        return 'exhausted';
    else
        d_i ⟵ first value in domain;
        new_path ⟵ nil;
        counter ⟵ 0;
        if check(previous_path) then
            return d_i;
        else
            return get_next(domain minus d_i); end if; end if;

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

図 3: Synchronous Branch and Bound. Variable/agent and value orderings are given in advance, and both n_i and s_i for $\forall i$ have uniform values as their initial values. The initial value of s_i should be zero when searching for an optimal solution. The procedure should be initiated only by the first agent in the ordering.

$ub-1$ にするように他のエージェントにメッセージを伝播させる．これを受け取ったエージェントは再度分散ブレイクアウト法を実行し，以下同様に繰り返す．この繰り返しは，あるエージェントが $\forall i(N_i = 1)$ のときの解を発見したときに終了する．

反復分散ブレイクアウト法では，分散ブレイクアウト法と同じ $ok?$ メッセージと $improve$ メッセージが用いられ，また，それぞれを受け取ったときに実行される手続きも分散ブレイクアウト法の手続きの一部を変更したものになっている．そのため，以下では主な変更部分についてのみ述べ，分散ブレイクアウト法と共通する部分については [15] に譲る．

- 実行前に，各エージェントがもつ不完全 CSP の共通の必要値 ub を決める．ub は，$ok?$ メッセージ，$improve$ メッセージとともにやり取りされる．

- 近傍の全エージェントから $ok?$ メッセージを受け取ったとき，エージェント i は，自分が知っている制約のうち違反している制約の数が必要値 N_i 未満ならば，現在の割り当てのコストを零にする．N_i 以上ならば分散ブレイクアウト法と同様に違反制約の重み和としてコストを計算する．

- 分散ブレイクアウト法では，あるエージェントの $termination\_counter$ という内部変数の値が，エージェントを頂点，エージェント間の論理的な通信リンクを辺とするグラフの直径以上になるとき，分散 CSP が解けていることが保証されている．反復分散ブレイクアウト法では，同じ条件のとき，すべてのエージェントの不完全 CSP が $\forall i(N_i = ub)$ のもとで解けていることが保証される（証明の概略は後述）．このことを検出したエージェントは，現在の N_i の値を 1 だけ下げ，新たな ub として $ok?$ メッセージ，$improve$ メッセージで他のエージェントに送る．

- 新しい値 ub を受け取ったエージェントは，その値を自分の必要値に代入すると同時に，$broadcast$ という内部変数を真にする．この変数が真の間は，全エージェントの必要値を 1 減少させるための言わば過渡状態にあると解釈でき，その間は，$termination\_counter$ の値が零になる

反復分散ブレイクアウト法のアルゴリズムの詳細を図 4に示す．

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

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

6 評価

同期型分枝限定法と反復分散ブレイクアウト法の性能を実験で評価する．

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

評価に使用したのは，ランダムバイナリ分散 CSP のうち $n = 10$，$m = 10$，$p_1 \in \{18/45, 27/45, 36/45, 45/45\}$，$p_2 \in \{0.8, 0.9\}$ というクラスの問題である．このクラスのランダムバイナリ CSP は最大 CSP の問題として特に困難な部類に入る [7] [2]．

実験は，複数エージェントの並行動作をシミュレートする離散事象シミュレータの上で行う．このシミュレータでは，マネージャという仮想的なエージェントを想定する．マネージャは離散的な時を刻む時計をもち，また，エージェント間のメッセージの配送を一手に引き受ける．全体の動作は，次の一連の処理を 1 サ

[2] 最大 CSP と分散最大 CSP では問題が異なるため，分散最大 CSP 用のアルゴリズムを評価するのに，これらのクラスの問題が適当かどうかは不明である．しかし，予備実験により，それらは分散最大 CSP の問題としても充分困難なものであると確認できた．

```
procedure initiate
    current_value ⟵ the value randomly chosen from domain;
    send (ok?, x_i, current_value, n_i) to neighbors;
    goto wait_ok? mode;

wait_ok? mode
when i received (ok?, x_j, d_j, ub) do
    counter ⟵ counter + 1; add (x_j, d_j) to agent_view;
    when ub ≠ n_i do
        n_i ⟵ min(n_i, ub); broadcast ⟵ true; end do;
    if counter = number_of_neighbors then
        when n_i ≤ s_i and broadcast = false do
            terminate the algorithm; end do;
        send_improve;
        counter ⟵ 0; broadcast ⟵ false;
        goto wait_improve mode;
    else
        goto wait_ok? mode; end if; end do;

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

wait_improve mode
when i received (improve, x_j, improve, eval, termination_counter, ub) do
    counter ⟵ counter + 1;
    my_termination_counter ⟵ min(termination_counter, my_termination_counter);
    when ub ≠ n_i do
        n_i ⟵ min(n_i, ub); broadcast ⟵ true; end do;
    when improve > my_improve do
        can_move ⟵ false; quasi_local_minimum ⟵ false; end do;
    when improve = my_improve and x_j precedes x_i do
        can_move ⟵ false; end do;
    when eval > 0 do
        consistent ⟵ false; end do;
    if counter = number_of_neighbors then
        when n_i ≤ s_i and broadcast = false do
            terminate the algorithm; end do;
        send_ok; counter ⟵ 0; broadcast ⟵ false; clear agent_view;
        goto wait_ok? mode;
    else
        goto wait_improve mode; end if; end do;

procedure send_ok
    when consistent = true and broadcast = false do
        increment my_termination_counter;
        when my_termination_counter = diameter do
            n_i ⟵ n_i - 1; my_termination_counter ⟵ 0; end do; end do;
    when quasi_local_minimum = true do
        increase the weights of violated constraints; end do;
    when can_move = true do
        current_value ⟵ new_value; end do;
```

図 4: Iterative distributed breakout. Both n_i and s_i for $\forall i$ have uniform values as their initial values. The initial value of s_i should be zero when searching for an optimal solution. The procedure should be initialized by each agent before receiving any message.

problem class	median cycle	mean optimal distance
$\langle 10, 10, 18/45, 0.8\rangle$	3500	1.0
$\langle 10, 10, 18/45, 0.9\rangle$	18262	2.0
$\langle 10, 10, 27/45, 0.8\rangle$	46247	2.4
$\langle 10, 10, 27/45, 0.9\rangle$	499841	3.4
$\langle 10, 10, 36/45, 0.8\rangle$	336416	3.6
$\langle 10, 10, 36/45, 0.9\rangle$	1985700	5.0
$\langle 10, 10, 45/45, 0.8\rangle$	3435984	4.9
$\langle 10, 10, 45/45, 0.9\rangle$	21834077	6.0

表 1: Median cycles for the Synchronous Branch and Bound for finding an optimal solution.

イクルとした繰り返しとなる．(1) エージェントから他のエージェントに向けて送信されたメッセージをマネージャがすべて集める，(2) マネージャが時計の時刻を 1 単位すすめる，(3) マネージャが集めたメッセージを宛先となるエージェントに届ける，(4) 個々のエージェントが，受け取ったメッセージに対して処理をおこない，他のエージェントにメッセージを送信する．このシミュレータ上で 2 つのアルゴリズムを実現し，サイクル数（マネージャの時計が刻んだ時間) をコストとして評価する．

6.1 最適解のコスト

まず，最適解を得るコストを評価する．

同期型分枝限定法では最適解を得ることが保証されているため，それを得るまでのサイクル数を求めることができる．実験では，ランダムバイナリ分散 CSP の各問題クラスに対して 25 例をランダムに生成し，それぞれに同期型分枝限定法を適用した．なお，同期型分枝限定法の変数順序は，conjunctive width ヒューリスティックスの width/domain-size の場合 [11] に従って決定し，値順序は辞書式順序とした．また，任意のエージェント i の必要値 N_i には，初期値として，解く例に対応する制約グラフの最大次数より 1 だけ小さい値を代入する．同期型分枝限定法では，N_i をパスの評価値の上界値として使うが，パスの評価値は制約グラフの最大次数より大きくなることがないためである．また，充分値 S_i は零とした．表 1は，各問題クラスごとに，最適解を得るまでにかかったサイクル数の中央値，および，最適距離の平均値を示している．同期型分枝限定法で最適解を見つけるには全般にコストがかかり，特に最適距離が大きい問題に対して，かなり厳しい結果になっている．最適距離が大きい場合，探索の途中で枝刈りに有効ないい上界値を得にくいため，探索する空間が広くなりがちである．同期型分枝限定法は逐次的な探索を行うので，探索する空間が広くなれば，それが直接サイクル数に影響する．

一方，反復分散ブレイクアウト法は完全ではないため，最適解に達しない場合があり，同期型分枝限定法と同様には評価できない．しかし，どれくらい最適解への到達に失敗するかを調べることは可能である．今回の実験では，$p_1 = 27/45$，$p_2 = 0.8$ の 25 例のそれぞれにつき，初期値を変えて 10 回ずつ合計 250 回に渡って反復分散ブレイクアウト法を実行してみた．なお，各エージェントの必要値と充分値の設定については同期型分枝限定法の場合と同じである．結果は，同じ問題に対し同期型分枝限定法が最適解を発見したサイクル数までに最適解に達することができたのは 30 回であり，残り 220 回はすべて失敗した．

6.2 Anytime Curves

次に，同期型分枝限定法と反復分散ブレイクアウト法の最適解に至るまでの動作を評価するために Anytime Curves を描く．Anytime Curves とは，アルゴリズムをスタートさせて以後，時間経過と共に中間結果がどのように改善されるかをグラフ化したものである．ここでは，x 軸にアルゴリズムをスタートさせてからのサイクル数，y 軸に大域距離関数の値（エージェントの制約違反数の最大値）をとり，各アルゴリズムごとに，あるサイクル数までに得られた解のうち，もっとも良い解の大域距離の値をプロットする．図 5の実線は，$n = m = 10$，$p_1 = 27/45$，$p_2 = 0.9$ の 1 つの例に対する同期型分枝限定法の Anytime Curve を示している．このクラスでは，この例に対して，同期型分枝限定法はもっとも短いサイクル数で最適解を見つけている．また，破線は，同じ例に対する反復分散ブレイクアウト法の Anytime Curve である．反復分散ブレイクアウト法では，初期値を変えて 10 回試行し，その平均をプロットしている．

図 5から明らかなように，同期型分枝限定法の Anytime Curve は，最終的には最適距離まで減少していくものの，減少する速度はかなり遅い．一方，反復分散ブレイクアウト法の場合は，非常に早い段階で急減し，以降，最適距離の近く（準最適距離とよぶ）でほぼ一定となり，それ以上は下がらない．これらと同

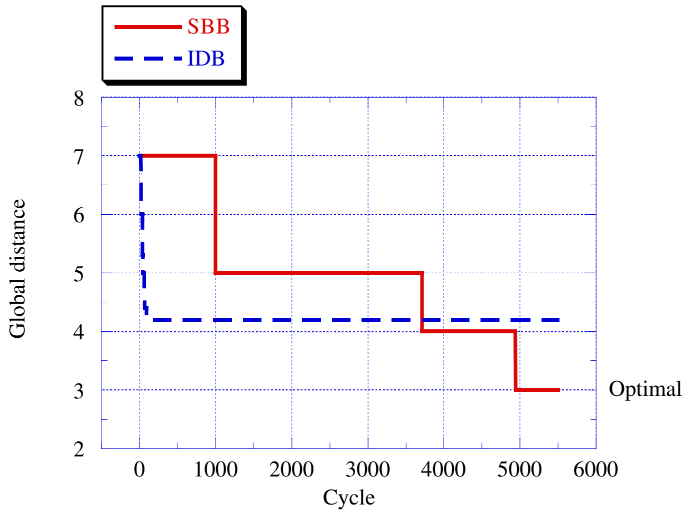

図 5: Anytime curve for an instance of $\langle 10, 10, 27/45, 0.9 \rangle$

様のことは，この例のみならず，他のクラスの他の例についても観察された．また，各クラスの問題について，先ほどと同様に例を 1 つ決めて，その例に対して同期型分枝限定法と反復分散ブレイクアウト法のそれぞれが準最適距離に達するまでのサイクル数を求めた．表 2にその結果を示す．これを見ると，すべての場合において，反復分散ブレイクアウト法の方が速く準最適距離に達していることがわかる．

反復分散ブレイクアウト法の Anytime Curve が最初に急減する理由は次の通りである．反復分散ブレイクアウト法では，全エージェント共通の必要値として ub を決め，各エージェントが自分の距離が ub 未満になるように分散ブレイクアウト法を実行し，全エージェントが ub 未満になると，次は $ub - 1$ にして . . . と繰り返すわけだが， ub の値が大きいときには，分散ブレイクアウト法がほとんど疑似局所最適解に出会うことなく終了し，すぐに次の $ub - 1$ の場合をスタートできる．そのため，反復分散ブレイクアウト法の初期段階（ ub の値が大きいとき）には，大域距離が急速に減少する．

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

7 まとめ

本論文では，過制約な分散 CSP を扱う枠組として分散不完全 CSP を提案した．また，過制約な分散 CSP への具体的な対処法として，分散不完全 CSP の部分クラスにあたる分散最大 CSP を取り上げ，それを解くアルゴリズムとして同期型分枝限定法と反復分散ブレイクアウト法を開発した．

今回は，分散最大 CSP という方法で過制約な分散 CSP を扱ったわけだが，これは，分散資源配分問題のような問題に対しては有効かもしれないが，例えば，分散解釈問題などに対してはあまり意味がないかもしれない．つまり，分散最大 CSP だけで過制約な分散 CSP をすべて扱えるわけではなく，今後，さらに別の部分クラスも検討する必要があると考えられる．今後の課題は，アルゴリズムの実現可能性等を考慮しながら，応用範囲のできるだけ広い分散不完全 CSP の部分クラスを探すこと，また，分散最大 CSP に対して，さらに効率のよいアルゴリズムを開発することである．

参考文献

[1] A. Borning, B. Freeman-Benson, and M. Wilson. Constraint hierarchies. In *Lisp and Symbolic Computation*, Vol. 5, pp. 223–270, 1992.

[2] K. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transaction on Computer Systems*, Vol. 3, No. 1, pp. 63–75, 1985.

[3] S. E. Conry, K. Kuwabara, V. R. Lesser, and R. A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1462–1477, 1991.

[4] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, Vol. 58, No. 1–3, pp. 21–70, 1992.

[5] K. Hirayama and J. Toyoda. Forming coalitions for breaking deadlocks. In *Proceedings of*

problem class	nearly-optimal (optimal)	cycle for IDB	cycle for SBB
$\langle 10, 10, 18/45, 0.8\rangle$	2.2 (1)	100	417
$\langle 10, 10, 18/45, 0.9\rangle$	3.6 (2)	46	585
$\langle 10, 10, 27/45, 0.8\rangle$	3.6 (2)	508	2052
$\langle 10, 10, 27/45, 0.9\rangle$	4.2 (3)	196	3716
$\langle 10, 10, 36/45, 0.8\rangle$	4.6 (3)	3416	6360
$\langle 10, 10, 36/45, 0.9\rangle$	6.0 (4)	344	200145
$\langle 10, 10, 45/45, 0.8\rangle$	5.8 (4)	438	258753
$\langle 10, 10, 45/45, 0.9\rangle$	7.3 (6)	90	45696

表 2: Cycles to find nearly-optimal solutions

First International Conference on Multi-Agent Systems, pp. 155–162, 1995.

[6] M. N. Huhns and D. M. Bridgeland. Multiagent truth maintenance. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1437–1445, 1991.

[7] J. Larrosa and P. Meseguer. Phase transition in max-csp. In *Proceedings of the Twelfth European Conference on Artificial Intelligence*, pp. 190–194, 1996.

[8] V. R. Lesser and D. D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, Vol. 4, No. 3, pp. 15–33, 1983.

[9] P. Morris. The breakout method for escaping from local minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 40–45, 1993.

[10] K. P. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 6, pp. 1446–1461, 1991.

[11] R. J. Wallace and E. C. Freuder. Conjunctive width heuristics for maximal constraint satisfaction. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 762–768, 1993.

[12] M. Yokoo. Constraint relaxation in distributed constraint satisfaction problem. In *5th International Conference on Tools with Artificial Intelligence*, pp. 56–63, 1993.

[13] M. Yokoo. Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In *Principles and Practice of Constraint Programming –CP95*, Vol. 976 of *Lecture Notes in Computer Science*, pp. 88–102. Springer-Verlag, 1995.

[14] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of the Twelfth IEEE International Conference on Distributed Computing Systems*, pp. 614–621, 1992.

[15] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of Second International Conference on Multi-Agent Systems*, pp. 401–408, 1996.