平成21年度　卒業研究論文

分散制約最適化問題における Max-Sumアルゴリズムの評価関数の改良の検討

指導教員

松尾　啓志 教授

津邑　公暁 准教授

名古屋工業大学 工学部　情報工学科

平成18年度入学　18115018番

川東 勇輝

平成22年 2月 8日

目 次

第1章

はじめに

近年、無線デバイスやセンサネットワークの発達により、通信網を介して情報を交換しつつ自律・協調的に処理を行うマルチエージェントシステムが注目されている。例えば、地理的に分散して配置されたセンサを用いた自然現象のモニタリングやレスキューロボット等が挙げられる。このようなシステムでは、管理・プライバシー・単一故障・スケーラビリティの問題から、処理を1ヵ所で集中して行うのでは無く、各エージェントで分散して行う事が望まれる。またマルチエージェントシステムにおいては、物理的に分散されたデバイスの動作を協調させること、近傍エージェントとのメッセージ通信のみを用いて最適化を行う必要がある。

このようなマルチエージェントシステムで協調的に解決されるべき代表的な問題の多くは、分散制約充足/最適化問題によって定式化できる。分散制約充足/最適化問題は、マルチエージェントシステムにおける協調問題解決を表す基本的な枠組みであり、マルチエージェントシステムにおける理論的な基礎として、様々な研究がされている。

システムを実装するにあたっては、低電力型の埋め込みデバイスなど、性能が制限されたデバイスが用いられる事が考えられる。つまり、電力や通信バンド幅、メモリ使用量、演算処理能力などに制限が課される可能性がある。したがって分散制約充足/最適化問題を解くアルゴリズムには、性能による制限もできる限り考慮する必要がある。

本論文では、上記のような制限を考慮した分散制約充足/最適化問題の解法である Max-Sum Algorithm[1] とその拡張版である MS-Stable[1] に注目し、それらの評価関数における問題点を挙げ、その改良について検討する。

本論文の構成は以下のとおりである。第 2 章では分散制約充足/最適化問題の基本構成およびその従来手法について述べ、第 3 章では Max-Sum Algorithm を用いた最適化手法について説明および評価として用いていた彩色問題への適用について説明し、第 4 章では提案手法について説明する。第 5 章では実験・評価を行いその結果について考察する。第 6 章では本論文の結論をまとめる。

第2章

分散制約充足/最適化問題

本章では、分散制約充足/最適化問題 (Distributed Constraint Satisfaction/Optimization Problems, DCSP/DCOP) の基本的な形式化、および現在研究されている DCSP/DCOP の解法について説明する。

2.1 分散制約充足/最適化問題

制約充足問題 (CSP) とは、与えられた制約を全て満たすような解を求める問題である。CSP は次のように定義される。

- n 個の変数 $x_1, \ldots, x_n$ が存在する
- 各変数は離散的な値域 $D_1, \ldots, D_n$ と x_i, x_j 間の制約 $c_{i,j} \in C$ を持つ

制約最適化問題 (COP) では、さらに制約に評価関数があり、評価関数 $f_{i,j}$ は変数値の割り当て $\{(x_i, d_i), (x_j, d_j)\}$ についての評価を計算し、その評価が最大となるような変数値の割り当てを求める。分散制約充足/最適化問題 (DCSP/DCOP) では、複数のエージェントに各問題の変数と制約が分散して配置される。各変数は各エージェントの状態を表し、各変数の値はその変数を保持するエージェントのみが決定できる。各エージェントは自エージェントと関係のある制約の情報のみを持つ。自エージェントと制約の繋がったエージェントを近傍エージェントとし、近傍エージェントとメッセージ交換を行いながら解を求める。

2.2 分散制約最適化手法

従来研究における分散最適化問題の解法について説明する。これらの解法は確実に最適解を探索する厳密解法と、近似解を求める非厳密解法に分類される。

2.2.1 厳密解法

最適解を求める厳密解法として、ADOPT(Asynchronous Distribute Constraint Optimization)[5], DPOP(Dynamic Programming Optimisation Protocol)[3] 等が挙げられる。ADOPT と DPOP は、制約網に対して前処理として，深さ優先探索木などの生成木および，それにもとづく疑似木 (pseudo-tree) を生成する。そして疑似木により定義される変数の半順序関係に従った，メッセージ交換型の探索アルゴリズムにより最適解を求める。これらのアルゴリズムは最適となる解を確実に求めることができるが、変数や制約密度などの問題の規模に対して、計算/空間複雑度・総メッセージ数・メッセージサイズ、もしくはそのいずれかが指数関数的に増加する問題が挙げられる。例えば ADOPT では、反復的な探索のための計算時間と総メッセージ数が指数関数的に増加する。DPOP では、前処理で作成した疑似木の幅にメッセージのサイズが依存しており、与えられる問題によってはメッセージサイズ・メモリ使用量などが指数的に増加するためエージェントに用いられるデバイスの性能に制限がある場合には、計算量・メモリ使用量などにおいて問題となる。

2.2.2 非厳密解法

最適解が求まるとは限らないが、比較的少ない計算で近似解を求める解法として、DSA(Distributed Stochastic Algorithm)[4]、Max-Sum Algorithm[1] が挙げられる。

DSA では、各エージェントは自身の持つ制約に関係する近傍エージェントの状態に基づいて、確率的に状態を更新する。確率的に状態を更新する事によって、連続して制約違反となる状態を取る事を避けている。この手法では、エージェントの状態に関する情報のみをメッセージとして送受信するので、通信コストを低く抑えることが出来る。そのため比較的大規模なシステムに適しているといえる。しかし、各エージェントは近傍エージェントの状態のみに基づいて自身の状態を決定しているため、局所的な最適解に収束しやすく、エージェント数や制約が多い複雑なグラフになると解の精度が低下してしまう。Max-Sum Algorithm では、近傍エージェントから隣接する変数がどのような状態を取るべきかというメッセージを、周囲の制約を考慮して送信する。そのメッセージを用いて、各エージェントは周辺関数を計算し、全体として最適である自身の状態を取るため、より解の精度が向上すると考えられる。そこで本研究では、Max-Sum Algorithm とその評価関数を拡張した MS-Stable[1] について注目する。

第3章

Max-Sum Algorithm

本章では、既存手法である Max-Sum Algorithm[1] および DCOP への適用について述べる。

3.1 Max-Sum Algorithm

Max-Sum Algorithm は、情報理論の分野で用いられる Sum-Product Algorithm と Max-Product Algorithm から派生したアルゴリズムである。Max-Sum Algorithm は Max-Product Algorithm の周辺関数を最大化するという点に注目して、DCOP を解くために用いる。これらのアルゴリズムでは、グラフ上のエージェントは関数ノード $f_1, \ldots, f_m$ と変数ノード $x_1, \ldots, x_n$ に分けられる。これらのノードは関数ノード同士、変数ノード同士が接続されないような二部グラフ (factor graph) で接続される。factor graph は次のように表される。

$$F(\mathbf{x}) = \prod_{m=1}^{M} f_m(\mathbf{x}_m) \tag{3.1}$$

例えば、関数 $F = f_1(x_1, x_2)(x_2, x_3)$ は図のように表される。

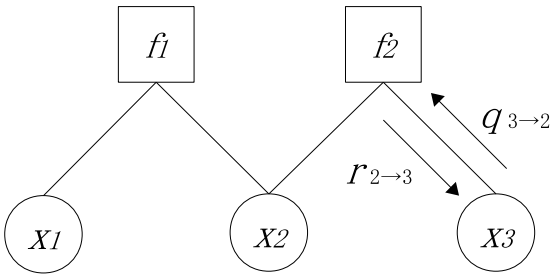

図 3.1: $F = f_1(x_1, x_2) f_2(x_2, x_3)$ の factor graph

Sum-Product Algorithm や Max-Product Algorithm ではこのような二部グラフ上で、関数ノードと変数ノード間でメッセージの交換を行い、変数 x_n が $F(x)$ 全体に与える影響 (周辺関数) を計算する。

一方、Max-Sum Algorithm では各エージェントが内部的に関数ノードと変数ノードを保持するように配置する。各エージェントの状態を変数が表し、エージェントに関係する制約を評価関数が表す。例えば、図 3.2 のようにエージェントが接続されている問題の場合、対応する factor graph は図 3.3 のように表現される。図 3.3 の factor graph では、エージェント 1 の利得はエージェント 1 と 2 の状態、エージェント 2 の利得はエージェント 1 と 2 と 3 の状態、エージェント 3 の利得はエージェント 2 と 3 の状態にそれぞれ基づいていることを表している。また図 3.3 で表されるグラフの利得は式で表すと次のように表せる。

$$\sum_{m=1}^{3} U_m(\mathbf{x}_m) = U_1(x_1, x_2) + U_2(x_1, x_2, x_3) + U_3(x_2, x_3) \tag{3.2}$$

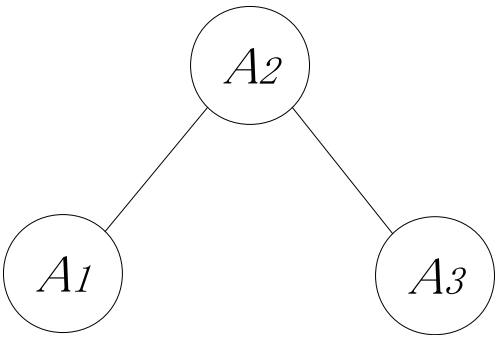

図 3.2: (a) グラフの構造

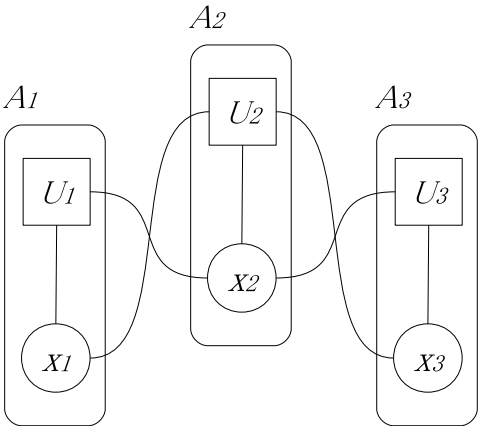

図 3.3: (b) 対応する factor graph

Max-Sum Algorithm ではこのように表されたグラフ上で、関数ノードと変数ノード間でメッセージ交換を行いながら $U_m(\mathbf{x}_m)$ を最大化する $\mathbf{x}$ を求める。Max-Sum Algorithm は大きく 3 つの動作に分けられる。

1. 関数ノードから変数ノードへのメッセージ計算・送信
2. 変数ノードから関数ノードへのメッセージ計算・送信

3. 周辺関数を計算し、全体として最適となる状態を選択

Max-Sum Algorithm では各エージェントが 1〜3 の動作を繰り返し行う。次にこれらの動作について述べる。

3.1.1 変数ノードから関数ノードへのメッセージ

Max-Sum Algorithm において変数ノードから関数ノードへのメッセージは、関数ノードから変数ノードのメッセージの和、すなわち変数が各状態を取った場合の評価に基づいて計算される。このようにして計算されたメッセージは各エージェント (変数) がどの状態を取りやすいかという事表している。変数ノードから関数ノードへのメッセージは以下のように定義される。

変数から関数へのメッセージ:

$$Q_{n\to m}(x_n) = \alpha_{nm} + \sum_{m'\in M(n)\backslash n} R_{m'\to n}(x_n) \tag{3.3}$$

ここで、Max-Sum Algorithm ではエージェントを関数ノードと変数ノードの 2 つに分けて配置するため、グラフは必ずサイクルを含む。そこでメッセージの値が無限に増加しないようにメッセージに α_{nm} を加え、正規化を行う。α_{nm} は次の式を満たすように選ばれる。

$$\sum_{x_n} Q_{n\to m}(x_n) = 0 \tag{3.4}$$

3.1.2 関数ノードから変数ノードへのメッセージ

Max-Sum Algorithm での関数ノードから変数ノードへのメッセージは、送り先の変数が各状態を取った場合に、その制約 (関数) にとってどの程度の利得を得られるかを送り先の変数に送信する。この利得は、評価関数 U と各変数からのメッセージによって計算され、以下のように定義される。

関数から変数へのメッセージ:

$$R_{m\to n}(x_n) = \max_{\mathbf{x}_m\backslash n}\left(U_m(\mathbf{x}_m) + \sum_{n'\in N(m)\backslash n} Q_{n'\to m}(x_{n'})\right) \tag{3.5}$$

また各エージェントにおいて最も計算量が必要であるのは式 (3.5) であり、式 (3.5) は近傍ノードの数のみに関して指数関数的である。そのため Max-Sum Algorithm は問題のサイズやシステム全体のエージェント数などに影響を受けない。

3.1.3 周辺関数の計算

ここでは、変数ノードから関数ノードへのメッセージを用いて周辺関数を計算する。周辺関数は変数 x_n が全体に与える影響を述べていて、以下のように定義される。

$$Z_n(x_n) = \sum_{m \in M(n)} R_{m \to n}(x_n) \tag{3.6}$$

このとき、周辺関数は利得の最大値を計算することができるはずであるが、Max-Sum Algorithm ではグラフの構成上必ずサイクルを含むので、式 (3.5) により正規化を行っている。そのために正確な値は計算できなくなり、周辺関数は以下のように近似値を表す。

$$Z_n(x_n) \approx \max_{\mathbf{x}_m \backslash n} \sum_{m=1}^{M} U_m(\mathbf{x}_m) \tag{3.7}$$

式 (3.7) を満たす引数を求めることで、全体の評価関数の値の合計すなわち全体の利得が最大となるような状態を決定する事が出来る。

3.1.4 アルゴリズムの特長

アルゴリズムの同期について

Max-Sum Algorithm はエージェントの動作に大域的な同期を必要としない。つまり、各エージェントは任意の時刻にメッセージの送信を行うことが出来る。また近傍のエージェントから新しいメッセージを受け取った場合、すぐにそのメッセージを計算に反映させる事ができる。よって各エージェントは、最新の情報である受信メッセージからその時点における自エージェントの最適な状態を推定し、選択することができる。

メッセージの数とサイズ

Max-Sum Algorithm はメッセージとして評価関数の値を定期的に送信する。そのため、エージェントの状態が変化したときのみその状態を送信する DSA[4] と比較するとそ

のメッセージの数およびサイズは大きい。しかし、エージェント数と値域の増加に対してメッセージサイズの増加は線形であるため、厳密解法である ADOPT[5] や DPOP[3] と比較すると、エージェント間で交換されるメッセージの数およびサイズは小さい。

3.2 グラフ点彩色問題への適用

グラフ点彩色問題とは、与えられたグラフに対して隣接する頂点同士が異なる色に彩色されるような、各頂点の組み合わせを求める問題であり、例題として用いられる [1][5]。分散グラフ彩色問題では、グラフの各頂点の色は、その頂点に対応するエージェントによって決定される。以下では頂点の色をエージェントの状態として表す。グラフ上で隣接する頂点に対応する 2 つのエージェントの状態が同じ色を表すとき，それらの状態は衝突する。各エージェントは選択可能な状態 $x_m \in 1, \ldots, c$ から状態を選択する。各エージェントの評価関数 $U_m(\mathbf{x}_m)$ は次のように表現される。

$$U_m(\mathbf{x}_m) = \gamma_m(x_m) - \sum_{i \in N(m) \setminus m} x_m \otimes x_i \tag{3.8}$$

このとき

$$x_i \otimes x_j = \begin{cases} 1 & (x_i = x_j) \\ 0 & (x_i \neq x_j) \end{cases} \tag{3.9}$$

であり、$\gamma_m(x_m) \ll 1$ は、衝突が無い状態での優先度を表し、同じ衝突数である対称解を除くために用いる。Max-Sum Algorithm ではこの評価関数を用いて、衝突の合計数が最小となるような各エージェントの状態を求める。

3.2.1 Max-Sum の評価関数の問題点

Max-Sum Algorithm に式 (3.8) を適用した場合、彩色可能でないグラフ、すなわち制約が密で大域的、局所的な不規則なサイクルを含むようなグラフにおいては、解の精度と収束が悪くなる事が示されている。しかし、これは評価関数 $U_m(\mathbf{x}_m)$ を 3.4 で述べる評価関数に変更することにより解消することが示されている [1]。

3.3 評価関数の拡張

ここではMax-Sumの評価関数を拡張し、周囲の制約をより詳細に考慮するように変更された評価関数について述べる。評価関数$U_m(\mathbf{x}_m)$を次のように変更される。

$$U_m(\mathbf{x}_m) = \gamma_m(x_m) - \sum_{i \in N(m)} \sum_{j \in C(i,m)} x_i \otimes x_j \tag{3.10}$$

ここで、$C(i,m)$は

$$C(i,m) = \{l \in N(m) | l > i \wedge (i \in N(l) \vee l \in N(i))\} \tag{3.11}$$

この拡張された評価関数を適用したものををMS-Stable[1]と呼ぶ。Max-Sum Algorithmの評価関数では、自エージェントと近傍エージェント間の衝突しか考慮していなかったが、拡張した評価関数を用いることで自エージェントと制約のある近傍エージェント同士の衝突も考慮される。これにより詳細な情報において最適な状態が推定されるので、解の精度や収束が向上すると考えられる。自エージェントA_0が隣接エージェントA_1、A_2、A_3と制約があり、隣接エージェント間の制約A_1-A_2、A_2-A_3があるグラフにMax-Sum、MS-Stableの評価関数を適用した場合、A_0において考慮される制約は図3.4、図3.5のようになる。

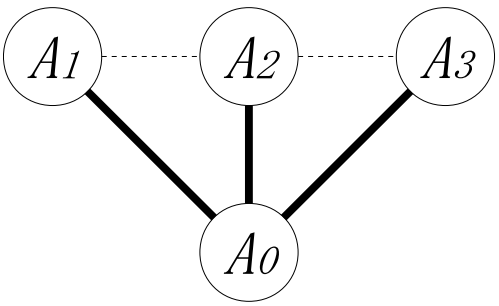

図3.4: A_0によるMax-Sumの評価関数で考慮される制約

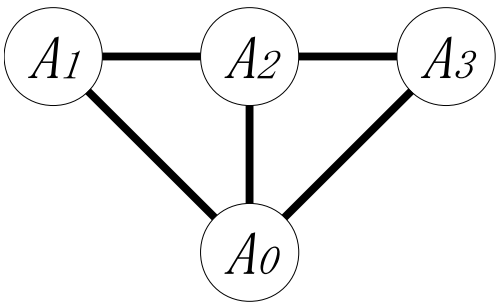

図3.5: A_0によるMS-Stableの評価関数で考慮される制約

3.3.1 拡張された評価関数の問題点

3.1.2 で述べたとおり、Max-Sum Algorithm は近傍ノード数について指数関数的に増加するようなメッセージ計算量を必要とするが、拡張された評価関数を用いることでその計算量はさらに増加する。具体的には、拡張前の Max-Sum の評価関数では式 (3.5) を求めるために、それぞれの近傍エージェントのメッセージから最大となる状態の組み合わせを計算する必要があるため

$$\sum_{i \in N(m) \backslash m} < x_m \times x_i > \tag{3.12}$$

通りの組み合わせから、最大値を求める必要があった。しかし拡張された評価関数を使う事により、近傍エージェント間の制約が考慮され、近傍エージェント間のメッセージの状態が衝突として式 (3.5) の最大値に影響を与えるようになるため、計算すべき組み合わせは最悪の場合

$$\prod_{i \in N(m)} x_i \tag{3.13}$$

通りまで増加してしまう。

第4章

提案手法

ここでは評価関数の変更による解の精度およびメッセージ計算量の調整と、各エージェントの評価関数の適用方法の提案について述べる。

4.1 評価関数の提案

3.3.1 で述べたように、Max-Sum[1] の評価関数は複雑なグラフに適用した場合の解の精度の低下が、MS-Stable[1] の評価関数では近傍エージェント間の全ての制約を考慮する事による計算量の大幅な増加が問題であった。そこで計算量を抑えつつ、解の精度を向上させるために、グループ化を用いて一部の近傍エージェント間の制約を考慮する評価関数について提案する。

4.1.1 近傍エージェントのグループ化

提案する評価関数では、一部の近傍エージェント同士の制約を考慮にいれるため、またメッセージ計算量を削減するために、自エージェントと制約のある近傍エージェントに対してグループ化を行う。グループ化の手順は次の通りである。

1. $N(m)$ から定数 k 個になるまで変数を取り出す
2. 取り出せた変数が k 個なら、それらの変数をグループ化し 1. へ
3. 取り出せた変数が k 個未満なら、それらの変数をグループ化し終了する

このようにして作成したグループに用いて、関数から変数へのメッセージ計算 (式 (3.5)) の際に、各グループ内の変数間の制約も考慮に入れる。グループ内の変数間の制約は、近傍エージェント同士の制約の一部である。提案手法では、グループ化する変数の数 k を増減する事によって、考慮する近傍エージェント間の制約数を調整する。例えば、自

エージェント A_0 が隣接エージェント A_1、A_2、A_3 間と制約があり、隣接エージェント間の制約 A_1-A_2、A_2-A_3 があるグラフとする。ここで自エージェント A_0 に提案手法のグループ化数 $k = 2$、$k = 3$ の評価関数を適用した場合、それぞれ図 4.1、図 4.2 のようになる。Max-Sum Algorithm の評価関数とグループ化数 $k = 2$ の提案する評価関数を用いた場合を比較すると、図 4.1 では自エージェントと隣接エージェント間の制約だけでなく、Group1 に含まれる A_1-A_2 間の制約も考慮されているため、提案手法ではより周囲の制約が考慮されている事が分かる。またグループ化数を増やした場合、すなわち図 4.1 と図 4.2 を比較すると、図 4.2 では Group1 の変数の数が 3 に増加し、Group1 は $\{A_1, A_2, A_3\}$ となり A_2-A_3 間の制約も考慮されるようになる。

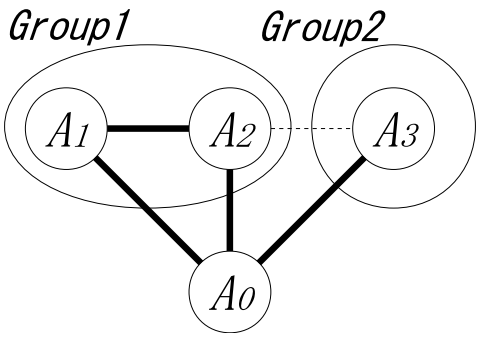

図 4.1: A_0 によるグループ化数 $k = 2$ の場合のグループ化と考慮される制約

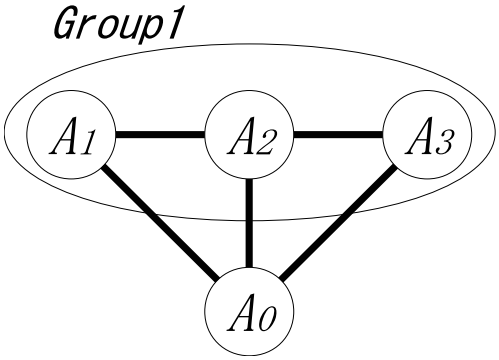

図 4.2: A_0 によるグループ化数 $k = 3$ の場合のグループ化と考慮される制約

以上のようにグループ内の制約を考慮に入れるために、Max-Sum Algorithm における評価関数を次の式 (4.1) に変更する。

$$U_m(\mathbf{x}_m) = \gamma_m(x_m) - \sum_{i \in N(m)} \sum_{j \in L(i,m)} x_i \otimes x_j \tag{4.1}$$

ここで、$L(i, m)$ は

$$L(i,m) = \{l \in N(m) | l > i \wedge (i \in N(l) \vee k \in N(i)) \wedge (G(m,i) = G(m,l))\} \tag{4.2}$$

ここで $G(m, i)$ は関数 f_m による変数 x_i の所属するグループの番号を返す。

4.1.2 提案手法のメッセージ計算量

関数から変数へのメッセージの計算 (式 3.5) において、MS-Stable の評価関数を使用すると最悪の場合、すべての近傍変数からのメッセージ ($Q_{n\to m}(x_n)$) の各値の組み合わせについて計算する必要があった。しかし提案手法では異なるグループに所属する変数ノード間の制約は考慮しないので、各グループで最大となる組み合わせを計算する際にグループ外の変数の状態がグループ内の最大値に影響しない。したがって各グループ毎に独立して最大値を計算することができ、グループ内の変数からのメッセージの各状態における組み合わせを計算するだけでよい。その場合の計算すべき組み合わせの数は

$$\sum_{g\in G(m)} < x_m \times \prod_{i\in g} x_i > \tag{4.3}$$

となる。ここで $G(m)$ は関数 f_m におけるグループの集合である。

4.2 評価関数の適用についての提案

従来の Max-Sum、MS-Stable の彩色問題への適用では、全てのエージェントが同一の評価関数を用いてメッセージを計算する。しかし全てのエージェントが周囲の制約を詳しく調べる、または簡略に調べるというのは極端で、非効率的だと考える。そこで提案手法では、制約網の複雑さに応じて適した評価関数を適用する事を検討する。基本的な方針として、次の 2 つが挙げられる。

1. 複雑な制約網を持つエージェントに周囲の制約を詳細に考慮する評価関数を適用する

2. 複雑な制約網を持つエージェントの周囲のエージェントに、周囲の制約を詳細に考慮する評価関数を適用する

1. の方針は Max-Sum Algorithm を複雑なグラフに適用した場合に解の精度が低下することから、複雑な制約網を簡易に調べている事が性能の低下に繋がっていると考え、複雑な制約網を持つエージェントは詳細に調べる評価関数を用いれば良いと考えた。しかし複雑な制約網を持つエージェントに周囲を詳細に調べる評価関数を適用すると、そのエージェントのメッセージの計算量が大幅に増加してしまい、異なる評価関数を適用したエージェント間での計算量に大きな差が出来てしまう事が問題として挙げられる。

2. の方針は 1. の方針のような評価関数の適用方法を用いるとエージェント間の計算量に大きな差が出来てしまう事から、複雑な制約網を持つエージェントの周囲のエージェントが詳細に周囲の制約を調べることにより、各エージェント間の計算量の差を緩和しつつ、解の精度の向上させる方針である。たとえば図 4.3 のような格子状に制約のあるグラフについて例を示す。図 4.3 のグラフでは A_4 が他のエージェントに比べて複雑な制約状況にあると考えられる。

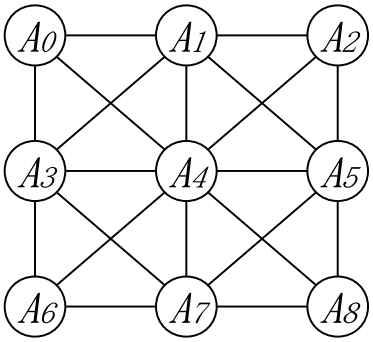

図 4.3: 格子状のグラフ

ここでは方針 1. の例として、A_4 に MS-Stable の評価関数を適用し、他のエージェントには Max-Sum の評価関数を適用した場合の各エージェントによって考慮される制約を表したものが図 4.4 になる。方針 2. の例として、A_4 の周囲のエージェントに MS-Stable の評価関数を、A_4 に Max-Sum の評価関数を適用した場合は図 4.5 になる。図 4.4 と図 4.5 を比較した場合、全体で考慮されている制約数は変わらないが図 4.4 においては考慮される制約が A_4 に偏っている。一方、図 4.5 では考慮される制約が各エージェントに分散されている。

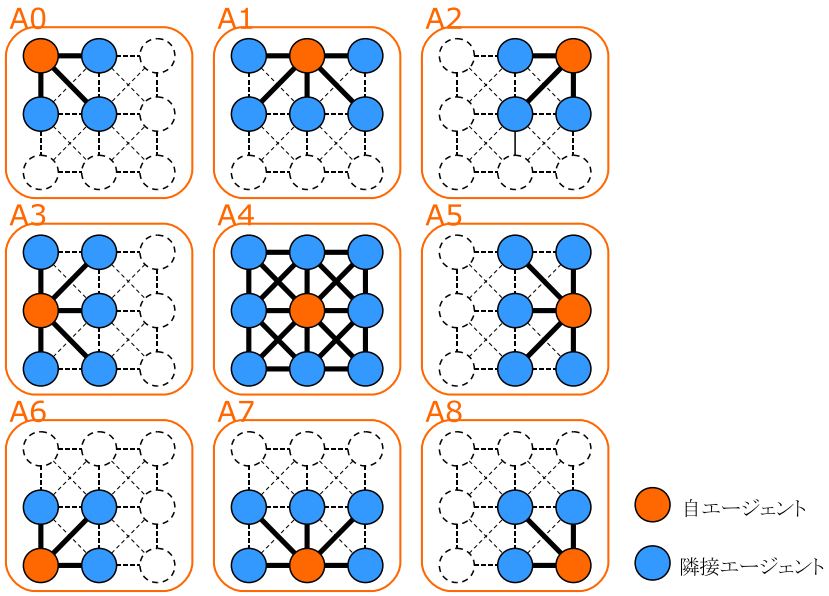

図 4.4: 方針 1. の各エージェントにより考慮される制約

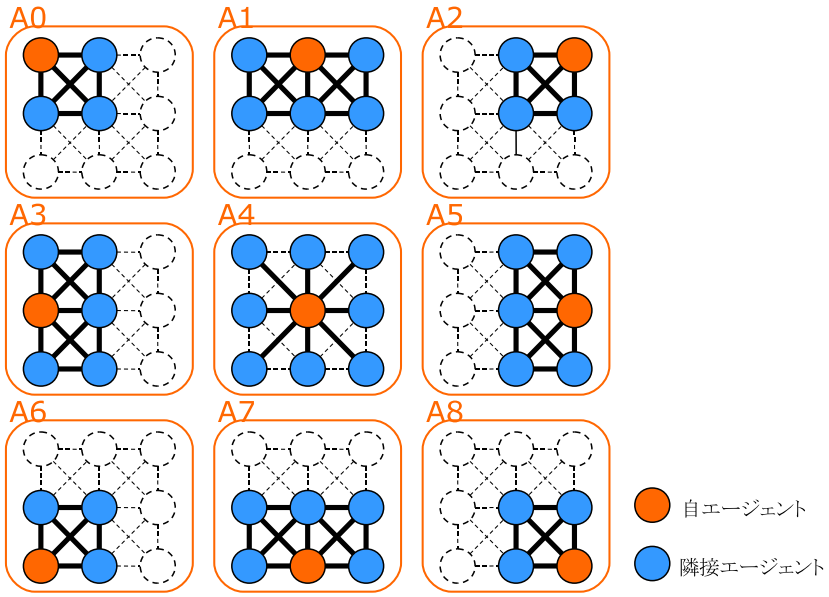

図 4.5: 方針 2. の各エージェントにより考慮される制約

第5章

実験・評価

ここでは彩色問題に、Max-Sum Algorithm[1] における、従来の評価関数と提案手法の評価関数を適用した場合についての評価を示し、その結果について考察する。

5.1 評価方法

実験 1 では彩色不可能な複雑なグラフ、実験 2 では完全グラフ、実験 3 では格子状のグラフを用いて、それぞれのグラフにおいて Max-Sum の評価関数、MS-Stable[1] の評価関数、提案手法の評価関数を適用した場合についての解の精度と計算量を比較した。計測の便宜上、マルチエージェントシステム全体の動作を、「サイクル」を単位として同期した。1 サイクルは、次のように構成される。変数から関数へのメッセージ、関数から変数へのメッセージ、周辺関数の計算をひとつの動作のまとまりとする。この動作のまとまりを、各エージェントが 1 回ずつランダムな順序で行う機会を設けた。隣接する頂点同士が同じ状態を選択した場合に衝突とし、1 サイクル毎に各エージェントの状態の評価を行った。そのサイクルあたりの衝突数の平均を取ったものを、単位時間あたりの衝突数として、平均衝突数とした。本実験では、アルゴリズムの終了までのサイクルを 50 サイクルに固定して行った。計算量においては、式 (3.5) の計算に必要となった、$Q_{n\to m}(x_n)$ の組み合わせの総数を用いた。計算量は各エージェント中、最大の組み合わせ総数であったものの 50 回平均 (最大計算組合せ総数平均) と各エージェントにかかった組み合わせ数の平均の 50 回平均 (平均計算組合せ総数平均) の 2 つを評価した。また変数値の値域の大きさは 3 とした。

5.2 実験 1:彩色不可能なグラフの場合

実験 1 では、彩色不可能であるような複雑なグラフに適用した場合、提案手法が従来手法に比べてどの程度の解の精度と計算量がかかるかを比較した。また提案手法におい

ては、最大グループ化数 k の値を変化させ比較した。ここでは問題のグラフとしてエージェント数を 10 から 20 に変化させ、エージェントの数*3 の制約数であるランダム生成した 50 個のグラフに対して、各手法の評価関数を全てのエージェントに適用し計測した。

5.2.1 実験 1:各手法の平均衝突数および計算量のエージェント数による変化

各手法における平均衝突数を図 5.1 に、50 回試行における最大計算組み合わせ数の平均を表 5.1 に、計算組み合わせ総数の平均を表 5.2 に示す。図 5.1 では、Max-Sum の評価関数を用いた場合が全体的に解の精度が悪く、MS-Stable が良く、提案手法がその中間程度の解の精度となった。またエージェント数が少ない方が、Max-Sum と MS-Stable の解の精度の差が大きく、各提案手法は MS-Stable に近い精度を示した。

最大の計算量は Max-Sum、提案 ($k = 2$)、提案 ($k = 3$)、提案 ($k = 4$)、MS-Stable の順に増加し、MS-Stable は大幅に値が大きくなった。ただし MS-Stable は実装上の都合で常に最悪計算総数となっている。エージェント数が増加するにつれて最大計算量値は増加した。

また平均の計算量についても同様の傾向を示したが、エージェント数による増加の傾向は見られなかった。

表 5.1: 複雑なグラフにおける
最大計算組合せ総数 (50 回試行平均)

エージェント数	10	12	15	18	20
Max-Sum	69	74	79	84	84
MS-Stable	16796	40416	70071	136206	129908
提案 ($k = 2$)	102	109	116	123	124
提案 ($k = 3$)	185	201	221	240	238
提案 ($k = 4$)	431	464	495	506	505

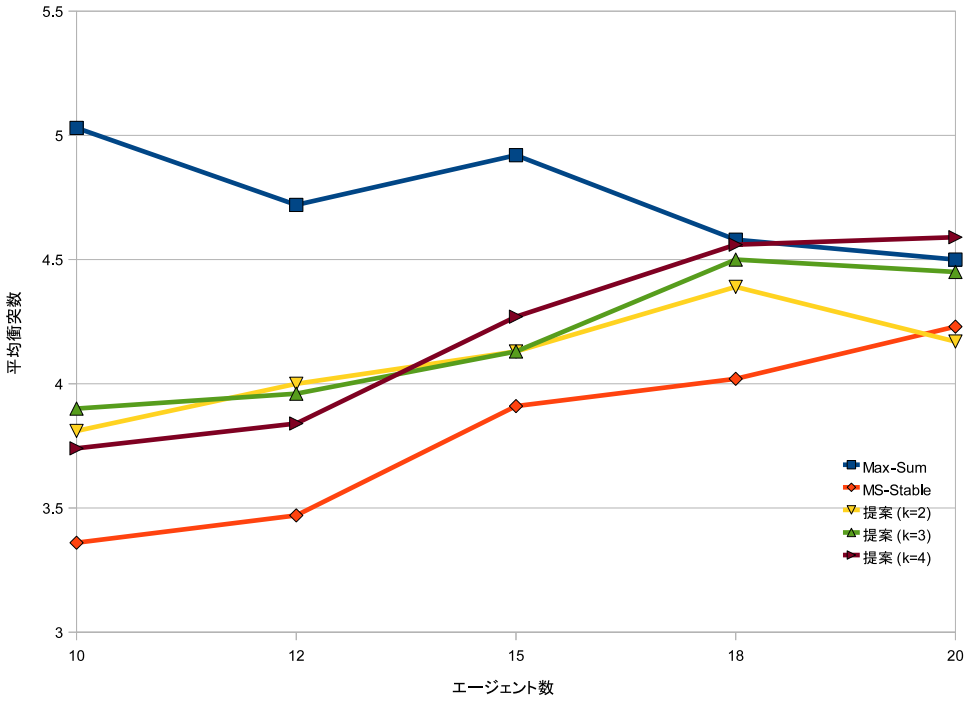

図 5.1: 複雑なグラフにおける平均衝突数

表 5.2: 複雑なグラフにおける
計算組合せ総数 (50 回試行平均)

エージェント数	10	12	15	18	20
Max-Sum	54	54	54	54	54
MS-Stable	4573	6767	9500	13586	13240
提案 ($k = 2$)	78	78	78	78	78
提案 ($k = 3$)	147	146	146	147	147
提案 ($k = 4$)	294	296	304	302	301

5.2.2 実験 1:考察

まず平均衝突数については Max-Sum、提案手法、MS-Stable の評価関数の順に周囲の制約を考慮する度合いが大きくなるため、解の精度も概ねその順によくなったと考えられる。エージェント数が少ない場合は解の差が大きく、エージェント数が多い場合は解の差が少なくなっている事から、評価関数において周囲の制約を詳細に調べても、グラフ全体を考慮しなければ解消できないような制約違反については効果が少ないという事がわかる。さらにエージェント数が少ないほど、制約網が密になっている事も影響している。

最大計算量・平均計算量については、MS-Stable の評価関数の場合、一部のエージェントに制約が偏って配置されると極端に計算量が大きくなってしまうが、提案手法ではグループ化によりその増加がある程度抑えられている事が分かる。

5.3 実験 2:完全グラフの場合

実験 2 では、完全グラフにおいて提案手法と従来手法の解の精度および計算量を比較する。実験 1 により、制約網が疎な場合より密である方が、評価関数で詳細に調べる効果が高い事から、制約網が密である完全グラフを用いて比較した。エージェント数は 4 から 9 まで変化させた。計算量については、完全グラフの場合各エージェントにおける計算量に違いはないので、計算組み合わせ総数 (50 回試行平均) のみを示す。

5.3.1 実験 2:完全グラフの場合の衝突数および計算量のエージェント数による変化

図 5.2 の平均衝突数については実験 1 同様、周囲の制約を詳細に調べる評価関数は平均衝突数が少なく、逆に簡略に調べる評価関数は平均衝突数が多くなり、実験 1 よりその差は顕著になった。提案手法についてはグループ化数が 2 の時よりも、3 と 4 の場合の方がより MS-Stable の結果に近づいた。またエージェント数が増えるにつれ、各手法による平均衝突数の差は大きくなった。

表 5.3 の完全グラフにおける計算量おいても、実験 1 と同様の傾向を示すが、各手法における計算量の差は実験 1 よりも顕著にあらわれている。

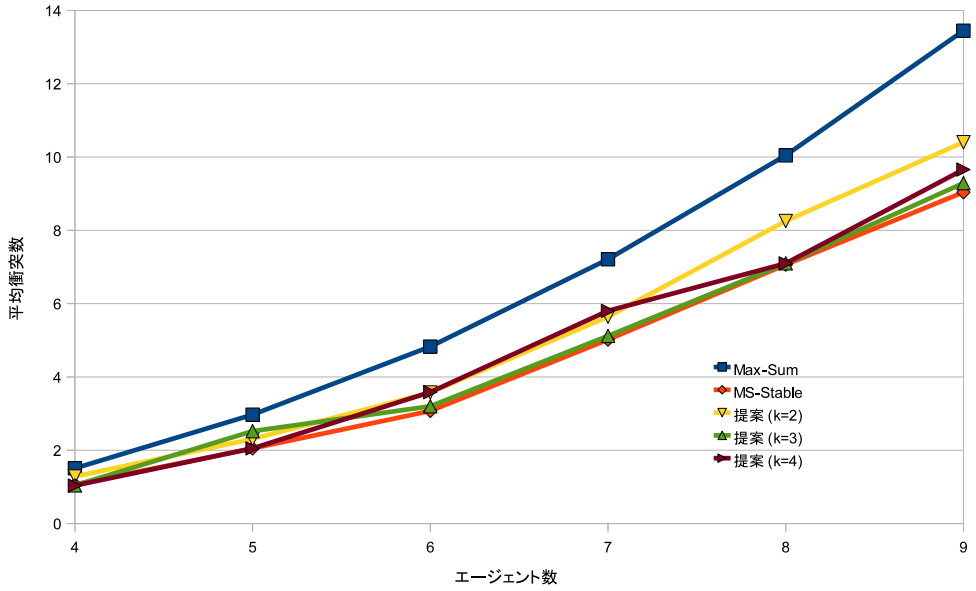

図 5.2: 完全グラフにおける平均衝突数

表 5.3: 完全グラフにおける計算組合せ総数

エージェント数	4	5	6	7	8	9
Max-Sum	27	36	45	54	63	72
MS-Stable	81	243	729	2187	6561	19683
提案 ($k = 2$)	36	54	63	81	90	108
提案 ($k = 3$)	81	90	108	162	171	189
提案 ($k = 4$)	81	243	252	270	324	486

5.3.2 実験 2:考察

完全グラフにおいては、Max-Sum の評価関数と MS-Stable の評価関数で考慮する制約数が大きく差があるので、実験 1 よりも解の精度や計算量に大きく差ができたと考えられる。また提案手法のグループ化個数が 2 個の場合は、3 個、4 個に比べて考慮する制約が少ない事から MS-Stable よりも解の精度は少し悪くなった。

5.4 実験 3:複雑さに偏りのあるグラフの場合

実験 3 では、どのようなエージェントが周囲を詳細に調べる必要があるかを検証するために、各エージェントに異なる評価関数を適用した場合について比較する。ここでは格子状のグラフ (図 4.3) を用いて実験を行った。この問題では、エージェント $4(A_4)$ は他のエージェントに比べて複雑な状況にあると考えられる。この実験では従来の評価関数・提案する評価関数に加え、以下の手法を用いた。

MS-Stable(Max-Sum):
A_4 に MS-Stable の評価関数、それ以外のエージェントは Max-Sum の評価関数

Max-Sum(MS-Stable):
A_4 に Max-Sum の評価関数、それ以外のエージェントは MS-Stable の評価関数

提案 $k = 2$(Max-Sum):
A_4 に提案の評価関数 ($k = 2$)、それ以外のエージェントは Max-Sum の評価関数

提案 $k = 3$(Max-Sum):
A_4 に提案の評価関数 ($k = 3$)、それ以外のエージェントは Max-Sum の評価関数

提案 $k = 4$(Max-Sum):
A_4 に提案の評価関数 ($k = 4$)、それ以外のエージェントは Max-Sum の評価関数

MS-Stable(Max-Sum) は複雑な状況にあるエージェントが周囲の制約を詳細に考慮するという手法、Max-Sum(MS-Stable) は複雑な状況にあるエージェントの周囲のエージェントが、周囲の制約を詳細に考慮するという手法、提案 $k = 2 \sim 4$(Max-Sum) は、MS-Stable(Max-Sum) で MS-Stable の評価関数を使う代わりに、提案手法の評価関数を使った手法である。

5.4.1 実験 3:評価関数を混在させた場合の衝突数と計算量

全エージェントに各評価関数を適用した場合は表 5.4 のようになった。実験 1、実験 2 同様、制約を考慮する数が多いほど解の精度がよくなり、計算量も増加した。エージェントに異なる評価関数を適用した場合は表 5.5 のようになった。

表 5.4: 全てのエージェントが同じ評価関数である場合の平均衝突数と計算量

手法	平均衝突数	最大計算量 (50 回平均)	平均計算量 (50 回平均)
Max-Sum	4.18	72	40
MS-Stable	2.10	19683	2547
提案 $k = 2$	3.62	108	56
提案 $k = 3$	3.51	189	105
提案 $k = 4$	3.09	486	202

表 5.5: エージェントの評価関数が異なる場合の平均衝突数と計算量

手法	平均衝突数	最大計算量 (50 回平均)	平均計算量 (50 回平均)
MS-Stable(Max-Sum)	2.08	19683	2221
Max-Sum(MS-Stable)	3.37	729	368
提案 $k = 2$(Max-Sum)	3.27	108	44
提案 $k = 3$(Max-Sum)	2.83	189	53
提案 $k = 4$(Max-Sum)	2.16	486	86

5.4.2 実験 3:考察

MS-Stable(Max-Sum) と全てのエージェントに MS-Stable を適用した場合の解の精度がほぼ同じになった事、Max-Sum(MS-Stable) のように周囲が詳細に調べる評価関数を適用した場合は、あまり解の精度が向上しなかった事から、周囲のエージェントに比べて制約網が複雑なエージェントが詳細に調べる評価関数を用いる事で良好な結果が得ら

れる事がわかった。また提案 $k = 2 \sim 4$ に比べて、提案 $k = 2 \sim 4$(Max-Sum) の解の精度が向上している事から、制約網が複雑なエージェントのみが一部の制約を考慮する事は効果的であることがわかった。

5.5 実験のまとめと今後の課題

実験 1、2 より、隣接エージェント間の制約の一部を考慮する提案の評価関数により、解の精度と計算量がある程度コントロールできる事がわかった。また実験 3 により異なる評価関数を混在させた場合、複雑な制約網を持つエージェントほど、詳細に調べる評価関数を導入すれば良いことがわかった。そこで今後は、グラフの制約網の特徴により、各エージェントが自律的にグループ化数を判断するような仕組みが必要であると考えられる。

第6章

まとめ

本論文では、分散制約充足/最適化問題の確率的解法である Max-Sum Algorithm[1] を複雑な制約網のグラフに適用した場合の性能の低下が、評価関数による周囲の制約網の考慮する度合いであるという事に注目し、制約の一部を考慮するような評価関数を提案・実装・評価した。実験・評価により、考慮する制約数によって解の精度および計算量がある程度調整可能であり、また複雑な制約網を持つエージェントが周囲の制約を詳細に調べる事の有効性を示した。今後の課題としては、グラフの特徴により、評価関数を自律的に変更するなどのアルゴリズムの改良があげられる。

謝辞

本研究のために多大な御尽力を頂き、日頃から熱心なご指導を賜った名古屋工業大学の松尾啓志教授、津邑公暁准教授、齋藤彰一准教授、松井俊浩助教に深く感謝いたします。

また本研究の際に多くの助言、協力をして頂いた松尾・津邑研究室ならびに齋藤研究室の皆様に深く感謝致します。

参考文献

[1] A.Farinelli, A.Rogers, A.Petcu and N.R.Jennings. Decentralised Coordination of Low-Power Embedded Devices Using the Max-Sum Algorithm. In *Seventh International Conference on Autonomous Agents and Multi-Agent Systems(AAMAS-08),2008.*

[2] D.J.C.MacKay. Information theory, inference, and learning algorithms. *Cambridge University Press, 2003.*

[3] A.Petcu and B.Faltings. DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, (IJCAI'05), pages 266-271, 2005.*

[4] Weixiong Zhang, Guandong Wang and Lars Wittenburg. Distributed stochastic search for constraint satisfaction and optimization. In *AAAI-02 Workshop on Probabilistic Approaches is Search, 2002.*

[5] P.J.Modi, W.Shen, M.Tambe, and M.Yokoo. ADOPT:Asynchronous distributed constraint optimization. Proc.*Autonomous Agents and Multi-Agent Systems, pp.161-168, 2003.*