平成20年度　　卒業研究論文

分散制約最適化問題のための Max-Sum アルゴリズムの評価と改良に関する検討

指導教員
松尾 啓志 教授
津邑 公暁 准教授

名古屋工業大学　情報工学科
平成17年度入学　17115140番

松浦加奈子

平成21年 2月10日

目 次

第1章

はじめに

近年、ネットワークを介して自律的に処理を行うマルチエージェント環境が注目されている。これは、通信コスト・管理・プライバシーなどの理由から、処理を一ヶ所で集中して行うのでは無く、分散した状態で行えることが求められているためで、例えば、分散センサ網における観測資源の割当 [8] や、レスキューロボット [5] などの研究がなされている。また、マルチエージェント環境で起こり得る問題を定式化した分散制約充足/最適化問題に関する様々な研究が行われている。

マルチエージェントシステムにおける課題として、物理的に分散されたデバイスの動作を協調させること、近傍にある少数の他エージェントとの通信のみを用いて最適化を行うことなどが挙げられる。さらに、システムを実装するにあたっては、低電力埋め込み型のデバイスなど、その性能に制限があるデバイスが用いられることが十分に考えられる。つまり、電力やバンド幅、メモリ資源や通信の安定性など様々な制限が課される可能性がある。従って、分散制約充足/最適化問題の解法ではこのような制限を考慮する必要がある。

そこで本論文では、上記で挙げた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] に注目する。その性能を平均衝突数およびメッセージサイクルの観点から評価し、グラフの特徴と Max-Sum Algorithm の性能の関連性について考察することで改良の検討を行う。

本論文は以下のように構成する。第 2 章では分散制約充足/最適化問題の基本構造及びその手法について述べ、第 3 章で Max-Sum Algorithm を用いた最適化手法につい

て説明する。第 4 章では評価方法として用いられている彩色問題への適用について説明し、第 5 章で実験評価及びその結果を述べ考察する。第 6 章で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) の基本的な構成及び、現在研究されている DCSP/DCOP の手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、求める解が満たすべきいくつかの制約を与えそれらを全て満たすような解を求める問題で、次のように定義される。n 個の変数 x_1,...,x_n とその各々の値域 D_1,...,D_n、及び変数 x_i, x_j 間の制約 $c_{i,j} \in C_s$ から構成される。各 D_i は有限で離散的である。制約に違反しない値 D_i を各変数 x_i に割り当て、全ての制約が満たされる変数の値の集合を求めたとき CSP の解が得られたことになる。また制約最適化問題 (COP) は、制約に対応した評価関数 $f_{i,j}$ により各変数値の割当 $\{(x_i, d_i), (x_j, d_j)\}$ についての利得を評価し、その利得の合計値を最適化する割当を求める問題である。

分散制約充足問題 (DCSP) 及び分散制約最適化問題 (DCOP) は、各問題の変数と制約が複数のエージェントに分散された問題である。各変数は各エージェントの状態を表す。また、各変数の値はその変数を持つエージェントのみによって決定される。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ち、他のエージェントとメッセージ通信を行うことで全ての制約を満たす解を探索する。

DCOP は DCSP では記述することが難しい最適化問題問題を扱うものとしてマルチエージェント分野で重要な位置づけにあり、本研究では主に DCOP について検討する

ものとする。

DCSP/DCOP で取り上げられる代表的な問題として、グラフ彩色問題、分散タスクスケジューリング問題 [9]、分散センサ網における観測資源の割当 [8] などが挙げられる。グラフ彩色問題は DCSP/DCOP を解く上で基礎となり、アルゴリズムのベンチマークとして頻繁に用いられる。分散タスク資源スケジューリング問題及び分散センサ網における観測資源の割当は、実際的・応用的な問題として扱われている。

2.2 分散制約最適化手法

現在研究されている DCOP の関連研究について説明する。DCOP は最適解を探索する厳密解法と、近似解を探索する確率的解法に大きく分けられる。

2.2.1 厳密解法

最適解を探索する厳密解法として、OptAPO[6]、ADOPT(Asynchronous Distributed Constraint Optimization)[4]、DPOP(Dynamic Programming Optimisation Protocol)[3] 等が挙げられる。OptAPO は、仲介エージェントが総合的な問題の一部を計算する、すなわち部分的な集中型アプローチを用いている。また、ADOPT や DPOP は共に制約グラフに対してあらかじめ深さ優先木 (DFS 木) に配置し pseudo tree を作成するような予備処理を行う。そして木の辺に沿ってメッセージ交換を行い情報を得て最適値を求める。

これらのアルゴリズムは確実に最適解を導くことが出来る一方で、エージェントの処理が変数や制約密度などの問題の規模に対して、計算/空間複雑度およびそれに付随する総メッセージ数/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ことが問題として挙げられる。例えば ADOPT は反復的な探索処理のための計算時間と総メッセージ数が、DPOP は部分解のコストを計算するための計算、記憶、メッセージサイズがそれぞれ指数関数的に増加する。これらの増加は作成した pseudo tree の induced width と各変数の変数値の数に依存しており、システムの規模が大きい程顕著である。従って、エージェントに用いられるデバイスの性能に制限が設けられて

いる場合には深刻な問題となる。

2.2.2 非厳密解法

一方で、最適解が得られるとは限らないが、比較的少ない計算で近似解を得る非厳密解法としてDSA(Distributed Stochastic Algorithm)[7]、Max-Sum Algotithm[1]が挙げられる。

DSAでは、各エージェントはそのコストに影響を及ぼす近隣のエージェントの状態のみに基づいて、確率的にその状態を更新する。

この手法では、その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信する。そのため、その通信コストは少なく抑えることができる。従って、比較的大規模な分散アプリケーションにも適しているといえる。しかし、各エージェントはその近隣エージェントの状態のみに基づいて状態を決定するため局所最適解に陥り易く、エージェント数や制約が多い複雑なグラフであるほどその解の精度は低くなる。

Max-Sum Algorithmも近隣エージェントから受け取ったメッセージの情報に基づいてその状態を決定するが、そのメッセージは、送信元エージェントの周囲の情報に基づいた評価関数である。そのため、各エージェントはより広範囲の情報を元にその状態を決定することができる。これにより、その解の精度は高くなることが期待される。

そこで本研究では、Max-Sum Algorithmに注目する。

第3章

Max-Sum Algorithm

本章では、Max-Sum Algorithm 及びその特徴について述べる。

3.1 Sum-Product / Max-Product Algorithm

Max-Sum Algorithm は、Sum-Product Algorithm から派生したアルゴリズムである。まず、初めに Sum-Product Algorithm 及びその派生である Max-Product Algorithm について説明する。

3.1.1 Sum-Product Algorithm

Sum-Product Algorithm は情報理論の分野で用いられており、図 3.1 のような二部グラフ (factor graph) 上での確率伝搬アルゴリズムである。グラフ中のノードは関数ノード f_1,...,f_m と変数ノード x_1,...,x_n に分かれていて、関数ノードは変数ノード、変数ノードは関数ノードとそれぞれ接続されている。この factor graph は、式 (3.1) の様な式を表すことが出来る。

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

関数 F は、n 個の変数 $\mathbf{x}$=$\{x_1,...,x_n\}$ を引数として、m 個の関数の結果として定義される。従って、図 3.1 では関数 $F = f_1(x_1, x_2)f_2(x_2, x_3)$ という式を表している。Sum-Product Algorithm では、各エージェントの周辺関数を計算するために接続された関数・変数ノード間でメッセージの伝達を行う。周辺関数は、変数 x_n が関数 $F(\mathbf{x})$ 全体

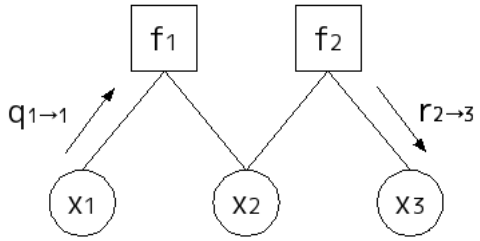

図 3.1: factor graph

に及ぼす影響を述べていて、次のように与えられる。

$$z_n(x_n) = \sum_{\{x_{n'}, n' \neq n\}} F(\mathbf{x}) \tag{3.2}$$

メッセージは関数 f_m から変数 x_n へのメッセージ $r_{m\to n}$ と、変数 x_n から関数 f_m へのメッセージ $q_{n\to m}$ の 2 つの形式を持つ。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:

$$q_{n\to m}(x) = \prod_{m' \in M(n)\backslash m} r_{m'\to n(x_n)} \tag{3.3}$$

関数から変数へのメッセージ:

$$r_{m\to n}(x) = \sum_{\mathbf{x}_m \backslash n} \left(f_m(\mathbf{x}_m) \prod_{n' \in N(m)\backslash n} q_{n'\to m}(x_{n'}) \right) \tag{3.4}$$

$N(m)$ は関数 f_m に接続された変数のインデックスのセット、$M(n)$ は変数 x_n に接続された関数のインデックスのセットを表す。

木構造などサイクルを含まないグラフでは、Sum-Product Algorithm は各変数に対する正確に周辺関数の値を計算し、ステップ数がグラフの幅に等しくなった後メッセージが収束するような更新ルールを持つ [2]。この更新ルールに従って、各葉ノードすなわち近傍ノード数が 1 であるノードはその親ノードに対してメッセージを送信することによってプロセスを開始する。各ノードはメッセージを受け取る度に等式 (3.3)、(3.4) に従って新しいメッセージを計算し送信する。変数ノードがその全ての近傍ノードからメッセージを受け取っている場合、以下の等式に従って正確な周辺関数の値が計算

出来る。

$$z_n(x_n) = \prod_{m \in M(n)} r_{m \to n}(x_n) \tag{3.5}$$

3.1.2 Max-Product Algorithm

Max-Product Algorithm は Sum-Product Algorithm から派生したアルゴリズムで、関数 $F(x)$ を最大化するような $\mathbf{x}$ の値を計算する。そのメッセージは、以下のように Sum-Product Algorithm のメッセージにおける合計関数を最大化関数に置き換えて定義される。

変数から関数へのメッセージ:

$$q_{n \to m}(x) = \prod_{m' \in M(n) \backslash m} r_{m' \to n(x_n)} \tag{3.6}$$

関数から変数へのメッセージ:

$$r_{m \to n}(x) = \underset{\mathbf{x}_m \backslash n}{max} \left(f_m \left(\mathbf{x}_m \right) \prod_{n' \in N(m) \backslash n} q_{n' \to m} \left(x_{n'} \right) \right) \tag{3.7}$$

Sum-Product Algorithm と同様に Max-Product Algorithm においても式 (3.5) を用いて周辺関数を計算するが、その目的は周辺関数そのものを計算するということよりも $max_x F(\mathbf{x})$ を計算することである。周辺関数は以下のように表される。

$$z_n(x_n) = \underset{\mathbf{x}_m \backslash n}{max} \prod_{m=1}^{M} f_m(\mathbf{x}_m) \tag{3.8}$$

この周辺関数から導かれる結果は、サイクルを含まないグラフにおいては最適である。

3.1.3 サイクルを含む場合のアルゴリズムの動作

先に述べたように Sum-Product Algorithm 及び Max-Product Algorithm における結果は、サイクルを含まないグラフにおいては最適でかつ収束することが保証されている。しかし、サイクルを含むグラフにもしばしば利用されることが考えられる。その

場合、伝播しているメッセージから導かれる周辺関数の値は近似解を表す。

$$z_n(x_n) \approx \max_{\mathbf{x}_m \backslash n} \prod_{m=1}^{M} f_m(\mathbf{x}_m) \tag{3.9}$$

また、サイクルを含むことでメッセージは反復して伝播されるため、それらの値は無限に増加する可能性がある。従ってメッセージが更新される際に値の正規化を行う必要がある。そこで、変数から関数へのメッセージを以下のように変更する。

変数から関数へのメッセージ:

$$q_{n \to m}(x_n) = \alpha_{nm} \prod_{m' \in M(n) \backslash m} r_{m' \to n}(x_n) \tag{3.10}$$

このとき、α_{nm} は係数で、次式を満たすように選ばれる。

$$\sum_{x_n} q_{n \to m}(x_n) = 1 \tag{3.11}$$

この正規化によって $max_{\mathbf{x}} \prod_{m=1}^{M} f_m(\mathbf{x}_m)$ の正確な値が計算できなくなる可能性があるが、その引数は計算することが可能である。

3.2 Max-Sum Algorithm

Max-Sum Algorithm は、Max-Product Algorithm の「関数 $F(x)$ を最大化するような $\mathbf{x}$ の値を計算する」という点に注目し、DCOP を解くためにそれを用いる。

3.2.1 グラフ構成

Max-Product Algorithm を DCOP に用いるために factor graph を用いて問題を表現する。具体的には、変数ノードが各エージェントの状態を、関数ノードがその利得を求める評価関数を受け持つことで「評価関数 U を最大化するような状態の集合 $\mathbf{x}$ を計算する」ことを実現する。例えば、図 3.2-(a) のようにエージェントが接続されている場合、対応する factor graph は図 3.2-(b) のように表される。この factor graph 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

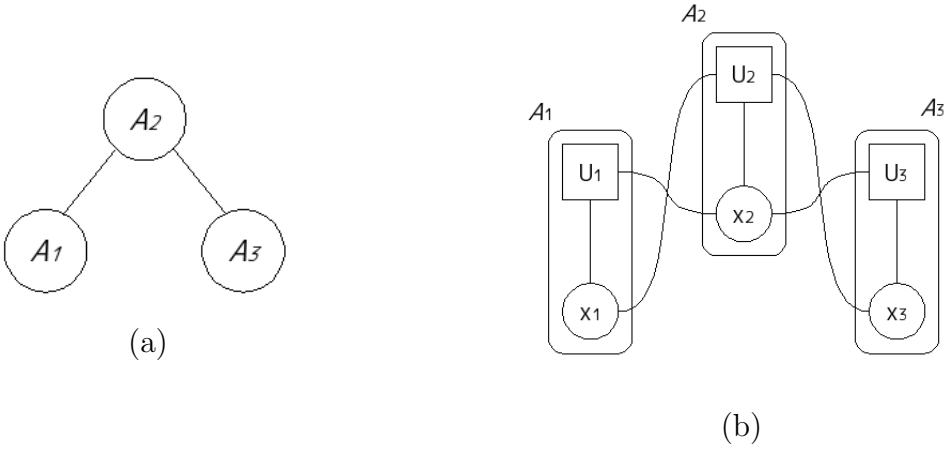

図 3.2: (a) グラフ構造と (b) 対応する factor graph

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ということを表している。

このような factor graph で表すとき、その利得の合計 U_{sum} は次式で求められる。

$$U_{sum} = \sum_{m=1}^{M} U_m(\mathbf{x})_m \tag{3.12}$$

3.2.2 メッセージ

Max-Sum Algorithm ではそのメッセージを対数で処理するため、以下のように定義する。

$$\begin{aligned} R_{m\to n} &= \log r_{m\to n} \\ Q_{n\to m} &= \log q_{n\to m} \\ Z_n(x_n) &= \log z_n(x_n) \\ U_m(\mathbf{x}_m) &= \log f_m(\mathbf{x}_m) \end{aligned}$$

これらを式 (3.6)(3.7) に適用すると以下のような式が得られる。

変数から関数へのメッセージ:

$$Q_{n\to m}(x_n) = \alpha_{nm} + \sum_{m'\in M(n)\backslash m} R_{m'\to n}(x_n) \tag{3.13}$$

このとき α_{nm} は係数で、次式を満たすように選ばれる。

$$\sum_{x_n} Q_{n\to m}(x_n) = 0 \tag{3.14}$$

関数から変数へのメッセージ:

$$R_{m\to n}(x_n) = \underset{\mathbf{x}_m \backslash n}{max} \left(U_m\left(\mathbf{x}_m\right) + \sum_{n' \in N(m)\backslash n} Q_{n'\to m}\left(x_{n'}\right) \right) \tag{3.15}$$

さらに、これらのメッセージを用いて各変数の周辺関数を計算する。

$$Z_n(x_n) = \sum_{m \in M(n)} R_{m\to n}(x_n) \tag{3.16}$$

このとき、$Z_n(x_n) = max_{\mathbf{x}_m \backslash n} \sum_{m=1}^{M} U_m(\mathbf{x})$ となるべきである。しかし、Max-Sum Algorithm においては、グラフの構造上必ずサイクルを含む。そのため、周辺関数は以下のように近似解となる。

$$Z_n(x_n) \approx \underset{\mathbf{x}_m \backslash n}{max} \sum_{m=1}^{M} U_m(\mathbf{x}) \tag{3.17}$$

式 (3.17) を満たす引数を見付けることで、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ことが出来る。各エージェントの動作はそのエージェントの近傍エージェントの数のみに関して指数関数的となり、問題サイズや pseudo tree の induced width などシステム全体の規模には影響を受けない。

3.2.3 アルゴリズムの動作

Max-Sum Algorithm は形式的な更新スケジュールを必要としない。すなわち、各エージェントは送信するメッセージを任意の時刻に初期化出来る。また近隣エージェントからのメッセージを受け取ったときは、いつでも送信メッセージを更新することが出来る。また、式 (3.17) で述べた計算は、近隣エージェントから受け取った最も新しいメッセージを用いることで、いつでも実行することが出来る。そのため、常にエージェントは自エージェントの最適な状態の推定が可能である 。

アルゴリズムを動作させた場合の最終的な状態として、一般的に以下の状態が考えられる。

1. 全てのエージェントの状態が最適もしくは近似解の一定の状態に収束し、そのメッセージも収束する
2. エージェントの状態は上記 1 のように収束するが、メッセージは更新するごとに変化しつづけ伝播し続ける
3. エージェントの状態及びメッセージが共に収束せず周期的な動作を行う。

これらを考慮すると、アルゴリズムの終端規則として以下の二つの異なる終端規則が用いられる。

1. メッセージが各々のエージェントの状態の変化を収束させる、もしくは状態の変化を収束させてメッセージが収束するまでメッセージを伝播させる
2. 各エージェントごとに一定回数メッセージを送信する

3.2.4 メッセージサイズ

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 と比較するとそのメッセージサイズは大きくなる。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[4] や DPOP[3] と比較するとエージェント間で交換される総メッセージ数、あるいは記憶複雑度及びメッセージサイズは少ないものとなる [1]。

第4章

彩色問題への適用

Max-Sum Algorithm の評価のために分散グラフ彩色問題が用いられている。本章では Max-Sum Algorithm の彩色問題への適用ついて説明する。

4.1 彩色問題

グラフ彩色問題とは、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なるように彩色を行う問題である。これは基礎的な組合せ探索/最適化問題で、センサネットワークのための協調アルゴリズムの評価のための例題としても用いられている。分散グラフ彩色問題ではエージェントはグラフ中のノードに配置され、自エージェントと辺で接続された他エージェントの状態と衝突する、すなわち同じ色を選ぶことがないように、可能な状態の集合 $x_m \in 1, ..., c$ から自エージェントの状態を選択する。

各エージェントの評価関数 $U_m(\mathbf{x}_m)$ は次式で表現される。

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

このとき、

$$x_m \otimes x_i = \begin{cases} 1 & (x_i = x_j \\ 0 & (x_i \neq x_j) \end{cases} \tag{4.2}$$

である。また、$\gamma_m(x_m) \ll 1$ は、衝突が無い状態での優先度を表す。この評価関数を Max-Sum Algorithm に用いることで、衝突の合計数を最小とするような各エージェントの状態を求める。

アルゴリズムの具体的な動作例を図 4.1 に示す。ステップ 0 は、各エージェントの優先度を表している。ステップ 1 からはそれぞれ関数から変数へのメッセージ (式 (3.15))、変数から関数へのメッセージ (式 (3.13))、そして各エージェントについての周辺関数の計算 (式 (3.17)) を表している。

0:
$$\gamma_1(x_1) = [\mathbf{0.1}, -0.1]$$
$$\gamma_2(x_2) = [-0.1, \mathbf{0.1}]$$
$$\gamma_3(x_3) = [-0.1, \mathbf{0.1}]$$

1:
$$R_{1\to2}(x_2) = max_{x_1}(U_1(x_1, x_2) + Q_{1\to1}(x_1)) = [-0.1, 0.1]$$
$$R_{1\to1}(x_1) = max_{x_2}(U_1(x_1, x_2) + Q_{2\to1}(x_1)) = [0.1, -0.1]$$

2:
$$Z_1(x_1) = R_{1\to1}(x_1) + R_{2\to1}(x_1) = [\mathbf{0.1}, -0.1]$$
x_1 は状態 0 を選択

3:
$$Q_{2\to3}(x_2) = R_{1\to2}(x_2) + R_{2\to2}(x_2) = [-0.1, 0.1]$$
$$Q_{2\to1}(x_2) = R_{2\to2}(x_2) + R_{3\to2}(x_2) = [0, 0]$$
$$Q_{2\to2}(x_2) = R_{1\to2}(x_2) + R_{3\to2}(x_2) = [-0.1, 0.1]$$

4:
$$R_{2\to3}(x_3) = [0.2, -0.2]$$
$$R_{2\to1}(x_3) = [0.2, -0.2]$$
$$R_{2\to2}(x_3) = [-0.1, 0.1]$$

5:
$$Z_2(x_2) = [-0.2, \mathbf{0.2}]$$
x_2 は状態 1 を選択

6:
$$Q_{3\to2}(x_2) = R_{3\to3}(x_3) = [0, 0]$$
$$Q_{3\to3}(x_3) = R_{2\to3}(x_3) = [0.2, -0.2]$$

7:
$$R_{3\to2}(x_2) = [-0.1, 0.1]$$
$$R_{3\to3}(x_3) = [0, 0]$$

8:
$$Z_3(x_3) = [\mathbf{0.2}, -0.2]$$
x_3 は状態 1 を選択

図 4.1: Max-Sum Algorithm の動作例 [1]

第5章

実験及び結果

本章では、前章で述べた彩色問題を用いて Max-Sum Algorithm の評価実験を行った。また、実験結果からグラフの特徴と Max-Sum Algorithm の性能の関連性について考察した。

5.1 評価基準

Max-Sum Algorithm の指標として、平均衝突数及びメッセージが収束する、すなわちエージェントが更新するメッセージが前回送信したメッセージと同一となるまでにかかるサイクル数を用いた。平均衝突数は単位時間あたりの制約違反数を表す。また、関数から変数へのメッセージを送信する、変数から関数へのメッセージを送信する、周辺関数を計算する、という一連の動作を 1 サイクルと定義した。

Max-Sum Algorithm は、彩色可能なグラフに適用された場合、DSA と比較すると約 5 倍の精度を示している [1]。しかし、彩色可能ではないグラフに適用した場合、制約網の構造によってはその精度が低くなり、またサイクル数が増加する傾向があることが示されている。これらのことは、あるエージェントと直接接続されている近隣エージェントとの間の制約のみでなく、エージェントに接続されている近隣エージェント間に制約がある場合は、その制約も考慮するような評価関数に変更することによって、解決することがわかっている [1]。しかし、3.2.2 で述べたとおり、Max-Sum Algorithm は近傍ノード数について指数乗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、変更した評価関数を用いることで、そのメッセージの計算量はさらに増加してし

まう。

本論文では、制約網の構造によってはその平均衝突数が増加し収束率が低下する傾向にあるということに注目し、グラフの構造と規模に対する平均衝突数及びサイクル数について Max-Sum Algorithm の特性を評価した。

5.2 実験 1: 環状および直線状のグラフ

Max-Sum が有効な問題の種類を特定するために、エージェント数が少ないグラフを用いて評価を行った。実験 1 では、グラフを用いてそのサイクル数の評価を行う。以下の二種のグラフを用いた。

1. エージェントが環状に接続されているグラフ (リング)
2. エージェントが一直線に接続されているグラフ (線)

実験では三色での彩色実験を行い、エージェント数が 3〜10 の場合の平均衝突数、及び、制約数が 3〜10 である場合の平均サイクル数についてそれぞれ計測を行った。1 つのグラフにつき 50 回の計測を行った。

5.2.1 実験 1: 結果

実行結果を図 5.1、図 5.2 に示す。線状のグラフの計測結果は、平均衝突数と平均サイクル数が、共に制約数が増加するに従って増加していることが分かる。しかし、リング状のグラフにおいてはエージェント数が 3 の場合のみ平均衝突数及び平均サイクル数が増加した。

5.2.2 実験 1: 考察

実験 1 のより、エージェント数が 3 のリング状のグラフにアルゴリズムを適用したときに平均衝突数及びサイクル数が増加するという結果が得られた。ここで、エージェント数 3 のリング状のグラフは、3 頂点の完全グラフであることに注目する。実験 1 では三色で彩色している。つまり、各エージェントが選択できる状態数は 3 である。こ

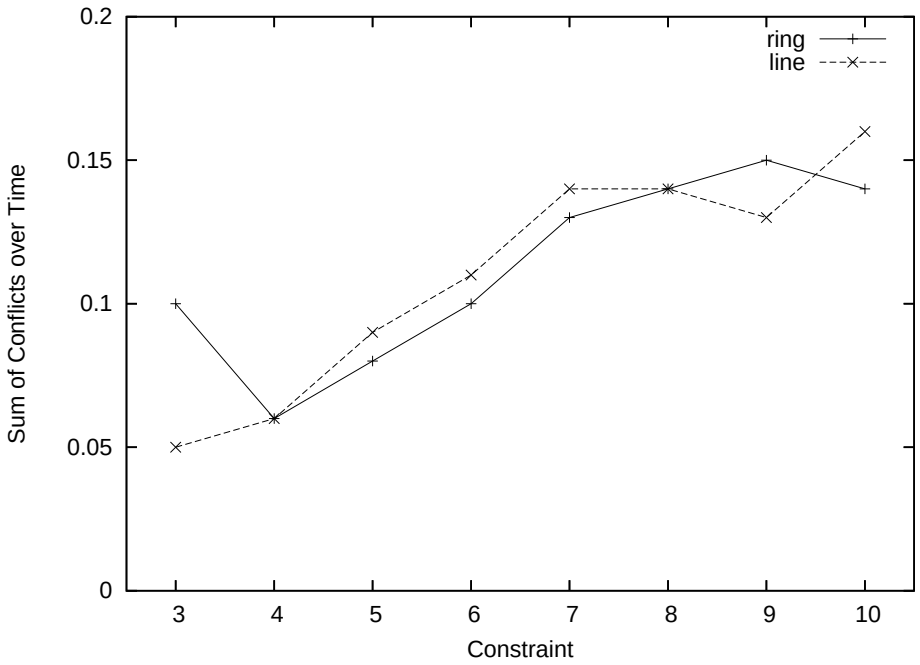

図 5.1: 実験 1: 平均衝突数 (Ring, Line)

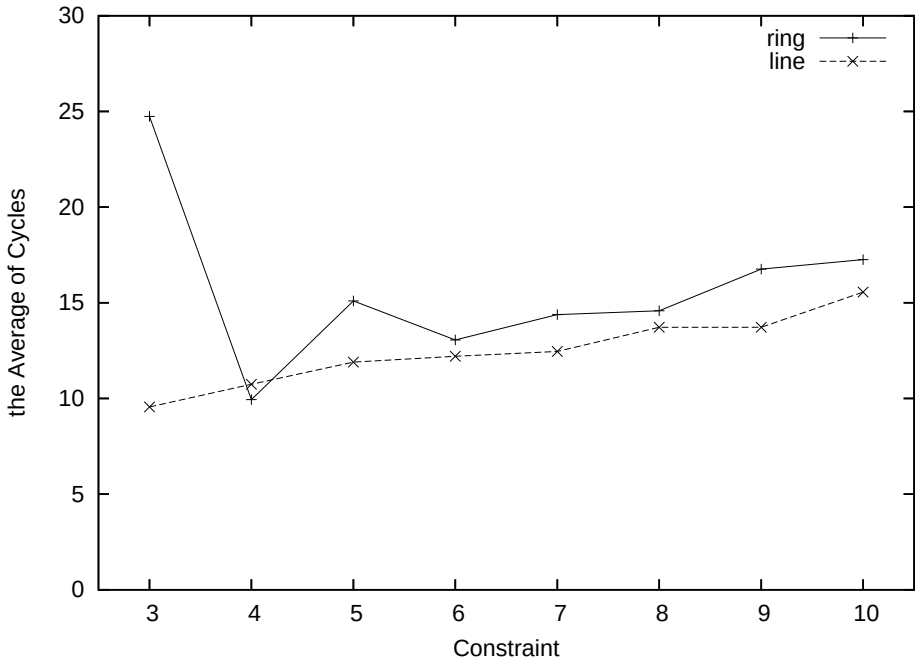

図 5.2: 実験 1: 平均サイクル数 (Ring, Line)

れは、このグラフを彩色するのに必要な最低数である。そのため、解となる状態を選択することが難しくなり、平均衝突数及びサイクル数が増加することが考えられる。よって、完全グラフを含む場合に平均衝突数及びサイクル数が増加すると推測できる。

5.3 実験 2: 完全グラフ

完全グラフを含む場合に平均衝突数及びサイクル数が増加することを検証した。

実験 1 で用いたリング状のグラフ中に制約を加えることで完全グラフを 1 つずつ追加し、平均衝突数及び平均サイクル数を計測する。このとき、グラフ中の完全グラフ同士が干渉しないよう、一定以上間隔を空けて完全グラフを配置した。エージェント数は 20 とした。

実験は 3 色完全グラフ、及び 4 色完全グラフを用いて行い、それぞれ同じ制約数で完全グラフを含まないグラフと比較を行った。

5.3.1 実験 2: 結果

3 色完全グラフを増加させた場合の結果を図 5.3、図 5.4 に、4 色完全グラフを増加させた場合の結果を図 5.5、図 5.5 に示す。実験の結果、どちらのグラフも共に同じ制約数でも完全グラフを含む場合の方が、含まない場合と比較して平均衝突数及びサイクル数は共に大きくなった。また、完全グラフを含むグラフと含まないグラフの平均衝突数、サイクル数それぞれの差を表 5.1、表 5.2 に示す。この表より、制約数が増えるにつれて二つのグラフの平均衝突数及びサイクル数の差が大きくなっていることが分かる。つまり、これらの数の増加は、制約数の増加に影響しただけではなく、完全グラフを含むことにも影響しているといえる。

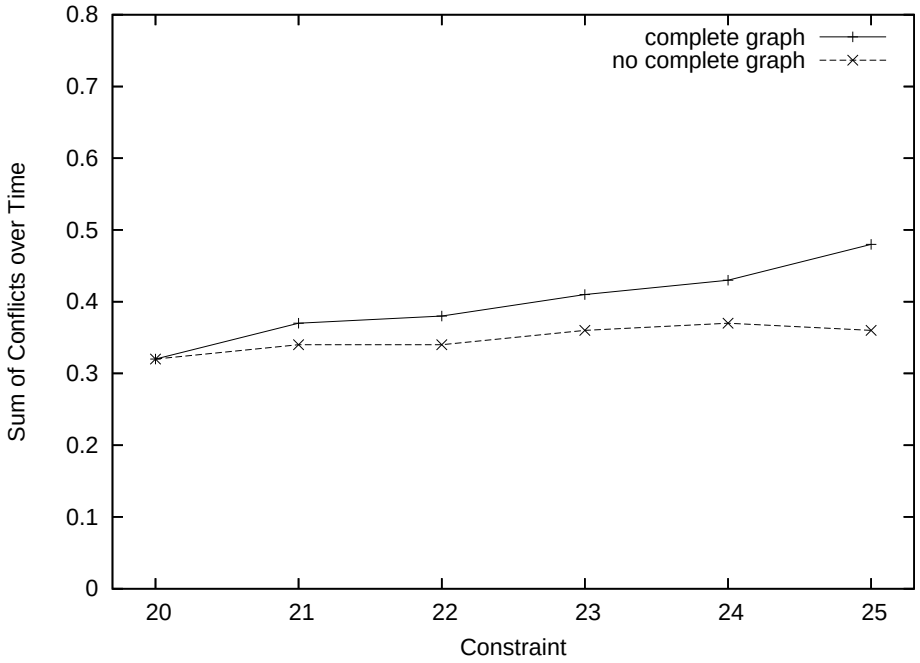

図 5.3: 実験 2:平均衝突数 (3 色)

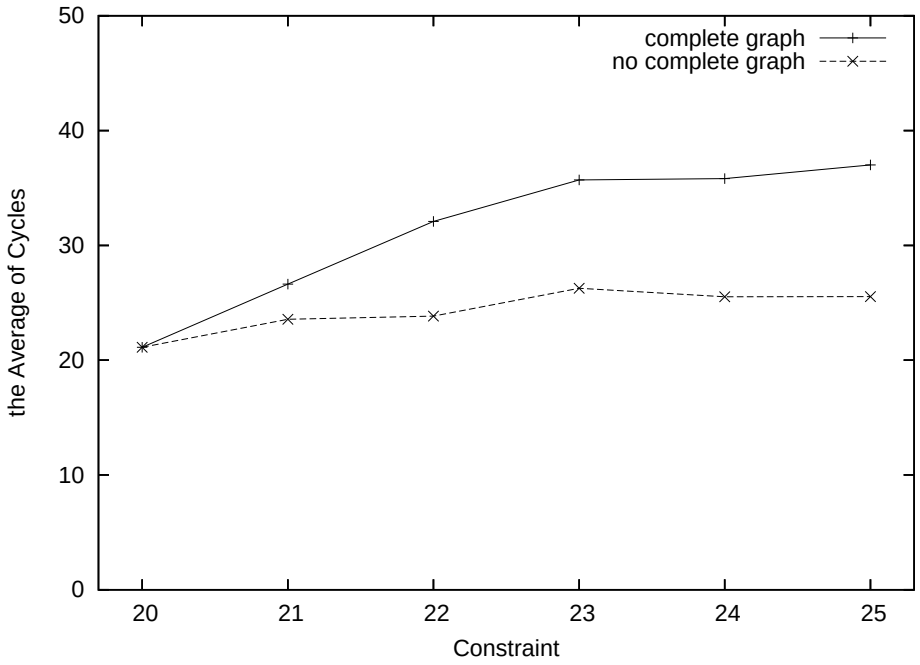

図 5.4: 実験 2:平均サイクル数 (3 色)

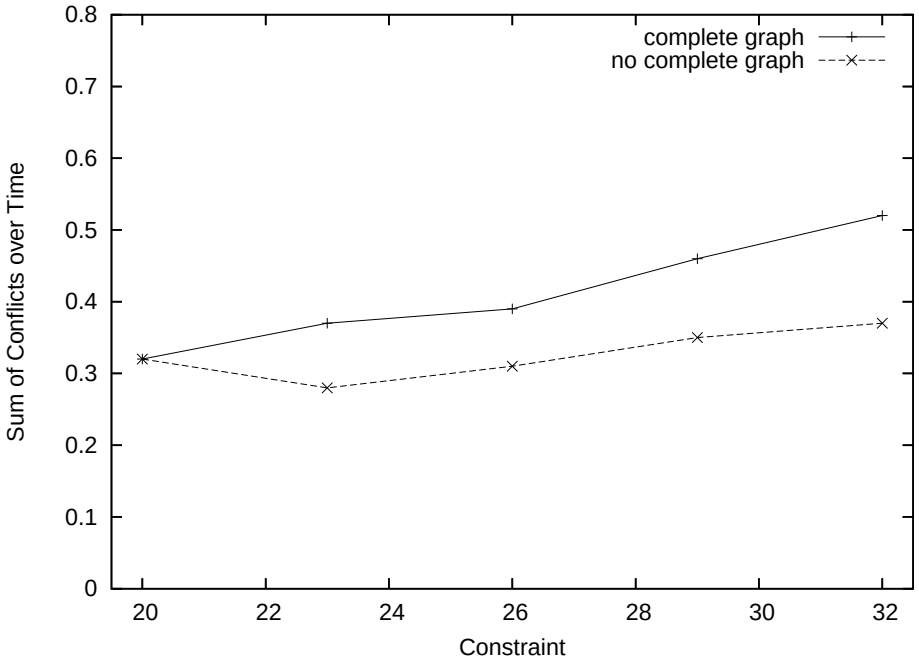

図 5.5: 実験 2:平均衝突数 (4 色)

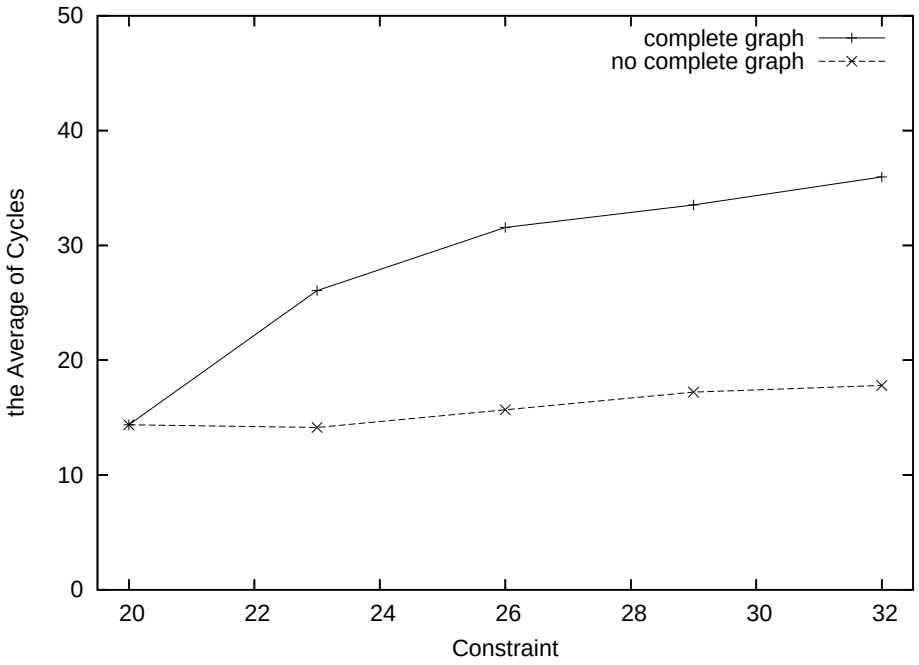

図 5.6: 実験 2:平均サイクル数 (4 色)

制約数	平均衝突数	サイクル数
21	0.03	3.60
22	0.04	8.24
23	0.05	9.44
24	0.06	10.03
25	0.12	11.46

表 5.1: 実験 2: 差 (3 色)

制約数	平均衝突数	サイクル数
23	0.09	11.92
26	0.08	15.88
29	0.11	16.3
32	0.15	18.16

表 5.2: 実験 2: 差 (4 色)

5.4 グラフの特徴による Max-Sum Algorithm の性能とその応用

実験結果より、完全グラフが含まれることが探索効率に影響し、効率が低下する傾向にあることが示された。従って、5.1 で述べたような周辺のサイクルを考慮する評価関数を導入することの有効性の根拠が示された。しかし、完全グラフとなる箇所全てについて、周辺のサイクルを考慮する評価関数を導入することはメッセージ及び計算量の点より効率的ではないといえる。そこで今後は、完全グラフ中の一部のサイクルのみを考慮するような手法を導入するために必要なグラフの特徴の解析を行うことが必要と考えられる。

第6章

まとめ

本論文では、マルチエージェントシステムの実装に伴う制限を考慮、かつ解の精度を保つことができる分散制約充足/最適化問題の確率的解法である Max-Sum Algorithm に注目し、その評価、改良を検討した。制約網の構造による精度および収束性の低下に注目し、そのグラフの特徴とアルゴリズムの探索処理の性能との関連性を調べた。単純なグラフを用いて実験・評価を行いグラフの特徴を考察することで、完全グラフとなる箇所を含むグラフにおいて性能が低下することを示した。今後の課題としては、制約網の特徴と規模の影響に関するより詳細な解析、および、解析に基づくアルゴリズムの改良などが挙げられる。

謝辞

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

また、本研究に際して多くの助言、協力をして頂いた松尾・津邑研究室ならびに斎藤研究室の皆様に深く感謝致します。

参考文献

[1] A.Petcu and N.R.Jennings A.Farinelli, A.Rogers. 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] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. *9th Int. Joint Conf. on Artificial Intelligence,2005.*

[4] Milind Tambe Pragnesh Jay Modi, WeiMinShen and Makoto Yokoo. An asynchronous complete method for distributed constraint optimization. *Autonomous Agents and Multi-Agent Systems,2003.*

[5] M.Gini D.Hougen P.Rybski, S.Stoeter and N.Papanikolopoulos. Effects of limited bandwidth communications channels on the control of multiple robots. *In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 369-374*, 2001.

[6] R.Mailler and V.Lesser. Solving distributed constraint optimization problems using cooperative mediation. *In Proceedings of 3rd International Joint Conference on Autonomous Agents and MultiAgent Systems(AAMAS'04),pages 438-445,2004.*

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

[8] Zhao Xing Weixiong Zhang, Guandong Wang and Lars Wittenburg. Distributed stochastic search and distributed breakout:properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence*, 2005.

[9] 工藤知宏 田中良夫 関口智嗣竹房あつ子. 計算資源とネットワーク資源を同時確保する予約ベースグリッドスケジューリング. *SACSIS 2006-05, pp93-100*, 2006.