Vol.2015-ICS-178 No.4
2015/3/2

デッドライン付きタスクを対象とした効率的チーム編成手法の提案

川口 竜太郎[1,a)]　早野 真史[1,b)]　菅原 俊治[1,c)]

概要：近年のネットワークやロボット技術の発展により，複数のエージェントが協力あるいは調整して実現できるタスクの重要性が増し，それらを効果的に処理させる手法が着目されている．このようなタスクの実現には，そのタスクを構成する各サブタスクを適切な能力を持つエージェントに割り当てる必要がある．さらにこれらのタスクには実時間性を求められるものも多く，デッドラインを考慮することも必要となる．マルチエージェントシステムの研究では，このような資源割り当て問題をチーム編成問題と考えられた多くの研究が存在する．本研究では，タスクにはデッドラインあること，また一つのタスクをチームで処理するのに一定の処理時間を要することを想定し，タスク処理効率の向上だけでなくエージェントがチームに必要以上に拘束される時間を減らすチーム編成手法を提案する．そのために，学習パラメータによりエージェントが自律的に適切なエージェントとチームを組み，チーム編成の効率を上げる学習手法と，エージェントがチーム履歴から自分の組めるチームの能力を推定してタスク処理の時間を見積もり，処理のできそうなタスクだけを選択し無駄なチーム編成の試みを減らす学習手法を提案する．評価実験から，チーム編成の成功率を上げるとともにエージェントのチームへの不要な拘束時間を減らし，システムとしてのタスク処理効率の向上が見込めることを示す．

1. はじめに

近年のネットワークやロボット技術の発展により，複数のエージェントが協力あるいは調整して実現するタスクの重要性が増している．このようなタスクの例としてコンピュータネットワークでは，グリッドコンピューティング [1] やサービスコンピューティングがあり，また災害時におけるレスキューロボット [6] の例もある．例えば，サービスコンピューティングではサービス（タスク）を構成する各サービス要素（サブタスク）を分散した計算機に割り当てて処理させることでサービスを実現させており，レスキューロボットでは災害現場での各作業（サブタスク）を複数のロボットに割り当てることで一つの作業（タスク）を実現する [6], [8]．分散システムにおける計算機やロボットを表すエージェントは，それぞれ異なる能力を持つため，タスクを成功させるには適切な能力を持つエージェントにサブタスクを割り当てる必要がある．

さらに上記で述べたタスクには実時間性を求められるものも多く，デッドラインを考慮した割り当てが必要となる．例えば災害時におけるタスクはデッドラインをもち，それまでに完了しないと人命に影響を及ぼしたり，建築物の崩壊などの新たな災害を起こしかねない．インターネット上のサービスもタイムリーかつ迅速なサービス提供はサービスの質では重要な要因であり，何らかの実時間性は必須である．

マルチエージェントシステムにおいてこのような資源割り当て問題を複数のタスクに対するエージェントのチームを複数同時に実現するチーム編成問題あるいは提携形成問題あるいは市場モデルにもとづく割り当て問題と考えられた多くの研究が存在する [4], [5], [10], [11]．例えば [5] では，エージェントがチーム編成の役割としてリーダかメンバを選択する．各エージェントは学習によりチームとして組むべき相手を学習し効率化しているが，リアルタイム性などは考慮していない．

そこで，本研究では [5] のモデルを拡張し，タスクの処理にリアルタイム性を導入し，タスクにデッドラインがあることを想定した場合においてチーム編成の効率化，及びタスク処理量の向上をめざす．また，タスク処理にはチームで一定時間を要するため，エージェントはチームにある一定時間を拘束される．そこでエージェントがチームに必要以上に拘束される時間を減らすようになるべく均等な能力をもつもので構成し，処理時間の無駄を小さくすること

[1] 早稲田大学
Waseda University
[a)] r.kawaguchi@isl.cs.waseda.ac.jp
[b)] m.hayano@isl.cs.waseda.ac.jp
[c)] sugawara@waseda.jp

もめざす．具体的には，エージェントは自分の役割とチームを組むべきエージェントを学習し，迅速かつエージェントの能力・資源利用効率がバランスしたチームを組むようにする．また，過去のチーム編成の経験から，自分の組めるであろうチームの能力を推定することで，デッドラインまでに処理することが難しいタスクの処理を断念し，無駄なチーム編成の実施も防ぐ．

本論文の構成は以下の通りである．まず，第 2 節で関連研究を述べ，第 3 節で基本的なモデルとチーム編成について述べる．第 4 節では，エージェントの学習手法とタスク処理の見積もり方法，さらにそれらを用いたチーム編成の過程について提案する．第 5 節では評価実験を行い，その結果から，(1) エージェントが組めるチームの能力を見積もり，デッドライン以内にタスクを完了させる確率を高めること，(2) チームメンバの能力を均等化し，能力の高いエージェントを不要に拘束しない，などを実現させ，システムの効率化を図れることを示す．第 6 節で結論と今後の課題を述べる．

2. 関連研究

第 1 節で述べたような，マルチエージェントシステムによる資源割り当てを考慮したチーム編成問題，提携形成問題，市場モデルにもとづく割り当て問題は多く存在する．例えばコントラクトネットプロトコル [11] では，エージェントがマネージャーか契約者の役割を持ち，マネージャーはエージェント群にタスクを広報し，契約者は広報されたタスクに対して入札する．その後マネージャーは入札エージェントから最適なエージェントを決め，そのエージェントに対してタスクを割り当てる．しかし，マネージャーは全契約者に広報メッセージを送るため，大規模な環境ではメッセージによる負荷が大きいだけでなく，メッセージの待ち時間が増える，競合するエージェントが増えるなどの課題がある．[10] では，システム全体の提携による効用値が最大となる提携構造を求めているが，その効用値はあらかじめ既知としている．また，組合せ問題であり，タスク数やエージェント数に応じて指数関数的に計算量は増大する．[2] では，エージェントが過去のチーム編成の結果から組むべきチームを学習し，タスクを処理するチーム編成の効率化を行っている．しかし，システム内に存在するタスクは無限に存在するため，一度も処理されないタスクが生じることがあり，現実的ではない．

[4], [5] では，複数のサブタスクで構成されるタスクがキューから順番に与えられるモデルで，タスクを処理するチーム編成の効率化を行っている．ここでは，エージェントがリーダとメンバの役割を持つ．各エージェントは学習パラメータを用いて，エージェントの獲得する報酬や他のエージェントとのメッセージ交換の結果から学習していき，適切な役割を選択する．しかし，タスクの処理においてリアルタイム性を考慮しておらず，また，タスクの処理時間は同一としている．実際のタスクにはデッドラインが存在し，このようなタスクは決められた時刻までに処理する必要がある．また，要求リソース量が多いタスクはより処理に時間を要することが考えられる．

タスクのリアルタイム性を考慮したマルチエージェントシステムの研究もある．[7] では，タスクの処理コストが時間と共に増加する環境におけるタスクの割り当て問題を対象としている．ここでは，タスクの処理コストは指数的に増加すると仮定し，処理コストが無限大にならないように効率的にタスク処理を行う手法が提案されている．また，タスクが継続的に発生するような動的な環境変化にも対応できるように，リアルタイムでタスクの再割り当てを行う手法も提案されている．しかし，エージェントの能力は均一としているため，エージェントの能力を考慮したタスク割り当ては行われていない．エージェントはそれぞれ異なる能力を持つのが一般的であるため，能力を考慮した効率的な割り当てを行うべきと考えられる．

[9] では，空間的，時間的制約がある際のチーム編成に関して述べられている．タスクにはデッドライン（時間的制約）と，存在位置（空間的制約）があり，エージェントがタスクまで移動してチームでタスクを処理する．エージェントはチームのサイズとタスクの処理時間を最小にするようにチームを選ぶことでタスクの処理率を向上させている．[3] では，ランダムな位置に置かれているデッドラインのあるタスクを，ロボットがグループで共通のゴールまで運ぶことを目的として，効率の良いグループ編成を行っている．しかし，[9] と [3] では，タスクがすでに与えらえている状況でチーム編成が行われており，新たにタスクが発生するような動的な変化には対応できない．

3. 基本設定とモデル

3.1 エージェントとタスク

エージェントの集合を $A = \{1, \ldots, n\}$ とおき，n はエージェントの数を表す．各エージェントは自分の処理能力に相当したリソースを持ち，エージェント i のリソースを $H_i = \{h_i^1, \ldots, h_i^p\}$ とおく．ここで p はリソースの種類数を表し，h_i^k は非負の整数とする．この数値が大きいほど，エージェントは高いタスク処理能力を持つ．

一定時間ごとにキューに追加されるタスク T は複数のサブタスクで構成されており，そのサブタスクの集合を $S_T = \{s_1, \ldots, s_m\}$ とおく．ここで m はタスクに含まれるサブタスクの数を表す．また，タスク T とそれを構成するサブタスクの集合 S_T を同一視することもある．各サブタスク s_i は処理するために要求されるリソースを持ち，そのリソースを $R_{s_i} = \{r_{s_i}^1, \ldots, r_{s_i}^p\}$ とおく．

ここでは，離散時間の単位を導入し，それを $tick$ と呼ぶ．各タスク T はある時間までに処理されなければならな

い締め切り時刻を表すデッドライン d_T をもつ．デッドライン d_T をもつタスク T を構成するサブタスク s の処理は次のように行われる．サブタスク s が要求する各リソースからエージェントが持つ各リソースを単位時間ごとに減算する．サブタスク s の各リソースをデッドライン d_T までに 0 以下にしたとき，サブタスク s の処理が完了する．例えば，$h_i^1 = 2, h_i^2 = 4$ のリソースをもつエージェント i が $r_s^1 = 4, r_s^2 = 9$ のサブタスク s を処理する時間は 3 となる．エージェント i がサブタスク s を処理する時間を t_s^i とすると，以下の 2 式を満たす t_s^i が存在するとき，エージェント i はデッドライン d_T までにサブタスク s を完了できる．

$$r_s^k - h_i^k \times t_s^i \leq 0 \tag{1}$$

$$t_s^i \leq d_T \tag{2}$$

また，あるタスク T の部分集合 $S \subseteq S_T$ に対し，エージェント i が以下の 2 式を満たす t_s^i が存在するとき，エージェントは S のサブタスクをすべて処理できる．

$$\bigwedge_{s \in S} (r_s^k - h_i^k \times t_s^i \leq 0) \tag{3}$$

$$\sum_{s \in S} t_s^i \leq d_T \tag{4}$$

もし，タスク T がデッドライン d_T を超えても完了できない場合は，タスク T の処理の失敗として廃棄する．

タスクに含まれるサブタスクをデッドラインまでに全て完了したとき，タスクの処理に成功し，エージェントには報酬が与えられる．本研究では，与えられる報酬をタスクに含まれるサブタスクの要求リソースの和とし，

$$u_T = \sum_{s \in S_T} \sum_{k=1}^{p} r_s^k \tag{5}$$

とする．この報酬はチーム内で配分されるが，エージェントは利己的とし，獲得報酬を最大にするように行動する．

3.2 チーム編成

エージェントはチームを編成してタスクを処理する．タスク T を処理するチームを $C = (G, \sigma, T)$ と表す．ここで，G はタスク T を処理するエージェントの集合，σ は各サブタスク s の割り当て関数を表し，$G \subseteq A, \sigma(s) = i \in G$ となる．σ についてチーム内の任意のエージェント $\forall i \in G$ が以下の 2 式を満たす t_s^i が存在するとき，C はタスク T を処理でき，チーム編成に成功したとする．

$$\bigwedge_{s \in \sigma^{-1}(i)} (r_s^k - h_i^k \times t_s^i \leq 0) \tag{6}$$

$$\sum_{s \in \sigma^{-1}(i)} t_s^i \leq d_T \tag{7}$$

エージェントはチームを編成する際，リーダかメンバの役割を選択する．ここで，リーダはタスクを共に処理するチームを作り，チームに参加しているメンバ及び自分自身にサブタスクを分配する．メンバは参加を依頼されたチームに参加し，サブタスクを処理する．

まず，エージェントは後で述べるタスク選択方法に従って，キューからマークの無いタスク T を一つ選び，そのタスクにマークする．また，自分に来ているチーム参加提案メッセージからメッセージ m を一つ選ぶ．エージェントはマークした T と選択した m をもとに，リーダになるかメンバになるか役割を決める．詳しい役割選択方法は後述する．

リーダになる場合，マークしたタスク T を共に処理するエージェントを選ぶ．このエージェントをメンバ候補と呼ぶ．メンバ候補を選択後，そのメンバ候補にチーム参加提案メッセージを送り，チームへの参加を依頼する．メンバ候補を選択できないときは，チーム編成の実施を断念し，次の時刻に再度役割選択を行う．詳しいメンバ候補の決定は後述する．

メンバになる場合，m を送ったリーダにチーム参加提案受諾メッセージを返し，チーム参加を表明する．m 以外のメッセージにはチーム参加提案拒否メッセージを返し，チーム参加を断る．その後，リーダはチーム参加を表明したエージェントをチームのメンバに加え，チームで T を処理できる割り当て関数 σ を生成可能かを判定する．このような σ が生成可能であれば，それに基づいて T の各サブタスク s_i を割り当て，各メンバにチーム編成の成功の通知とサブタスクの処理の依頼をする．このとき，サブタスクを割り当てないメンバが現れたときにはチーム編成の失敗を通知し，チームから外す．このようなエージェントは，リーダがチーム編成失敗の確率を下げるために，参加提案メッセージをやや多めに送るときに発生する．σ を生成できない場合は，チームの各メンバにチーム編成の失敗を通知し，チームを解散する．具体的なサブタスクの割り当て方法は後述する．

チーム編成に成功した場合，エージェントは自分に割り当てられたサブタスクを処理する．チーム内の全エージェントがサブタスクの処理を完了するまでの時間をチーム処理時間と呼ぶ．その時間がチームとしてタスクを処理する時間であり，その時間までチームを解散できないものとした．そのため，エージェントは自分に割り当てられたサブタスクの処理を早く終えてもチーム内の他のエージェントがサブタスクの処理を終えていなければ，チームに拘束される．そのためエージェントの不要な待ち時間をなるべく少なくすることも効率化につながる．

タスク処理完了後，チームは式 (5) で表された報酬を受け取り，これをリーダがチーム内のエージェントに分配する．まず，リーダが一定割合の報酬を受け取り，残りの報酬をメンバが処理したサブタスクのリソースの割合で分配する．詳細は次節で述べる．

4. 提案手法

本研究ではデッドラインを導入したモデルにおいて，タスクの処理効率を上げ，同時にエージェントがチームに必要以上に拘束される時間を減らすチーム編成をめざす．本モデルでは，タスクの処理後に得られる報酬はタスクのリソースの合計値とした．この報酬は，リーダーに与えられ，それをリーダがチーム内に配分する．各エージェントはこの報酬の配分を最大化するよう行動する．そのためにここでは，エージェントは過去のチーム編成の履歴にもとづいて 3 つのパラメータを学習し，行動を決定する．以下では，その 3 つの学習パラメータ，チーム履歴を用いたタスク選択方法，役割選択方法，メンバ候補の決定方法，サブタスクの割り当て方法について述べる．

4.1 欲張り度

欲張り度はチームとしてタスクの処理が完了し，リーダが報酬を分配するときの自分自身の報酬の取り分を表す値である．エージェント i の欲張り度を $g_i(0 \leq g_i \leq 1)$ と表す．チームに報酬 u_T が与えられたとき，リーダ i が獲得する報酬 u_i は $u_i = g_i \times u_T$ となる．その後 i はメンバの処理リソース量に応じて，各メンバ $j \in G \setminus i$ に報酬を以下の式で公平に分配する．

$$u_j = (u_T - u_i) \times \frac{\sum_{s \in \sigma^{-1}(j)} u_s}{\sum_{s \in S_T \setminus \sigma^{-1}(i)} u_s} \tag{8}$$

ここで u_s はサブタスクのリソース量に相当し，$u_s = \sum_{k=1}^{p} r_s^k$ である．g_i が大きいと，リーダ i が獲得する報酬は増加するが，メンバ j に分配する報酬は減少するため，合理的エージェントはリーダ i からのチーム参加提案メッセージを受諾しにくくなる．

欲張り度 g_i はチーム編成の成否によって，以下の式で更新する．

$$g_i = \alpha_g \times \delta + (1 - \alpha_g) \times g_i \tag{9}$$

ここで，$\alpha_g(0 \leq \alpha_g \leq 1)$ は欲張り度の学習率，δ はチーム編成の成否を表し，チーム編成に成功した場合は $\delta = 1$ とし，チーム編成に失敗した場合は $\delta = 0$ とする．

4.2 信頼度

信頼度はリーダが持つ値で，以下の 2 つの目的をもつ．

(1) リーダ i のチーム参加提案メッセージをエージェント j が受託してくれる可能性

(2) エージェント j がリーダ i をチームに不要に拘束しない時間割合

リーダ i のエージェント j に対する信頼度を $e_{i,j}$ と表す．この値が高いと，リーダ i はそのエージェント j が参加依頼を受託する可能性が高く，かつ必要以上に i をチームに拘束しないとして信頼する，そのため，積極的にチーム参加提案メッセージを送ったり，リソース量の大きいサブタスクを割り当てるようになる．

信頼度 $e_{i,j}$ は以下の式で更新する．

$$e_{i,j} = \alpha_e \times \mu \times \delta + (1 - \alpha_e) \times e_{i,j} \tag{10}$$

ここで，$\alpha_e(0 \leq \alpha_e \leq 1)$ は信頼度の学習率，δ はチーム参加提案返答メッセージの結果を表し，チーム参加を受諾した場合，$\delta = 1$ とし，チーム参加を拒否した場合，$\delta = 0$ とする．また，チーム参加提案返答メッセージが返ってこなかった場合はチーム参加を拒否したとし，$\delta = 0$ とする．

エージェント j がリーダ i よりサブタスクの処理時間が長いとき，リーダ i をチームに拘束したとして，その分の拘束時間を割り引いた値で学習する．μ はその割引率を表し，

$$\mu = \frac{t_i}{t_j} \tag{11}$$

とする．ここで，t_i はリーダ i のサブタスク処理時間，t_j はエージェント j のサブタスク処理時間を表す．ただし，エージェント j がリーダ i よりもサブタスクの処理時間が短いときは，リーダ i を必要以上にチームに拘束していないため，$\mu = 1$ とする．

4.3 報酬期待度

報酬期待度はメンバが持つ値で，メンバ i がリーダエージェント j から提示された単位時間あたりの報酬に対する実際に受け取る単位時間あたりの報酬の割合を表す値である．メンバ i のエージェント j に対する報酬期待度を $d_{i,j}$ と表す．エージェント i がリーダエージェント j からチーム参加提案メッセージ m を受託した時，得られる単位時間あたりの報酬の期待値を以下の式で表す．

$$\frac{\sum_{s \in S(m)} u_s}{\sum_{s \in S(m)} t_s} \times d_{i,l(m)} \tag{12}$$

ここで，$l(m)$ はメッセージ m を送ったリーダ，$S(m)$ はメンバ i がリーダ $l(m)$ から処理を依頼されたサブタスクの集合，t_s はエージェント i がサブタスク s の処理を完了する時間を表す．また，報酬期待度 $d_{i,j}$ は以下の式で更新する．

$$d_{i,j} = \alpha_d \times \frac{\frac{U}{t_{team}}}{\frac{\sum_{s \in S(m)} u_s}{\sum_{s \in S(m)} t_s}} + (1 - \alpha_d) \times d_{i,j} \tag{13}$$

ここで，U は実際に受け取った報酬，t_s はエージェント i によるサブタスク s の処理時間，t_{team} はチームとしての処理時間（全チームメンバーが処理を完了するまでに要した時間），$\alpha_d(0 \leq \alpha_d \leq 1)$ は報酬期待度の学習率を表す．すなわち，リーダ j から提示された単位時間あたりの報酬に対する実際に受け取った単位時間あたりの報酬を学習す

る．ここで，チーム編成に失敗した場合は報酬を受け取らなかったとし，$\frac{U}{t_{team}} = 0$ として学習する．

4.4 チーム履歴を用いたタスク選択方法

リーダエージェントがマークしたタスクを共に処理するメンバ候補を探すとき，デッドラインが迫っており，デッドラインまでに処理できるメンバ候補が見つからない場合がある．このようなタスクを処理しようとするとキューに存在するその他のタスクのデッドラインも迫るため，処理の断念も必要である．そこで，エージェントは過去のチーム編成の履歴から，自分の組めるチームの能力を推定し，タスクを処理できるか見積もることで，キュー内のタスクを選択する際，成功できないと判断したものは，その選択を断念する仕組みを導入する．エージェントはキュー Q 内に存在するタスク T を選択するとき，以下の式で表す過去のチーム履歴の平均リソース Ave からタスクをデッドライン d_T までに処理できるか見積もる．

$$Ave = \frac{\sum_{C \in P_i} H_C}{|P_i|} \tag{14}$$

ここで，P_i はエージェント i が保持するチーム編成の履歴の集合，$|P_i|$ はエージェント i が保持するチーム編成の履歴の数，H_C は P_i に含まれるチーム $C = (G, \sigma, T)$ のエージェント集合 G のリソース和であり，

$$H_C = \sum_{j \in G} \sum_{k=1}^{p} h_j^k$$

を表す．比較してタスクを処理できると判定すれば，そのタスクをマークし，そうでなければ次のタスクの見積もりに進む．これによりある程度，自分達の能力を勘案して処理時間が掛かるタスクを諦め，処理可能と思われるタスクのみを選択する．なお，エージェントには最近 N_o 回の成功したチーム編成の履歴を保持させることにした．タスク処理見積もりのアルゴリズムを**図 1** に示す．

Require: Ave: 過去のチームの平均リソース
P_i: エージェント i が保持するチーム履歴
Q: キュー内に存在するタスクの集合
R_T: タスク T に含まれるサブタスクのリソース和
d_T: タスク T のデッドライン
Ensure: $T_{possible}$: 処理できると見積もったタスク

```
for all T ∈ Q do
  if P_i = ∅ then
    T_possible ← T //チーム履歴がない場合はタスク T を処理できるとする
    break;
  else
    time ← ⌈R_T/Ave⌉
    if time ≤ d_T then
      T_possible ← T //タスク T を処理できるとする
      break;
    end if
  end if
end for
```

図 1 タスク処理の見積もり方法
Fig. 1 Algorithm for Task Estimation

4.5 役割選択

エージェントはチーム編成を行う際，リーダになるかメンバになるか役割を選択する．このとき，エージェント i はリーダとしてキュー内のマークしたタスク T を処理した際に得られる単位時間あたりの期待報酬 E_i^{leader} と，メンバとしてチーム参加提案メッセージ m と共に送られてきたサブタスクを処理した際に得られる単位時間あたりの期待報酬 E_i^{member} を比較して役割を決める．

$$E_i^{leader} = \frac{\sum_{s \in T} u_s}{t'_{team}} \times g_i \tag{15}$$

$$E_i^{member} = \frac{\sum_{s \in S(m)} u_s}{\sum_{s \in S(m)} t_s} \times d_{i,l(m)} \tag{16}$$

ここで，t'_{team} は第 4.4 節で述べたチーム履歴に基づくタスク T の処理時間の見積もり，t_s はエージェント i によるサブタスク s の処理時間，$l(m)$ はメッセージ m を送ったリーダ，$S(m)$ はメンバ i がリーダ $l(m)$ から処理を依頼されたサブタスクの集合を表す．この 2 つの値を比較し，$E_i^{leader} \geq E_i^{member}$ ならばリーダを選択し，$E_i^{leader} < E_i^{member}$ ならばメンバを選択する．ただし，キューにマークするタスクが存在しない場合は $E_i^{leader} = 0$ とし，チーム参加提案メッセージを一つも受信していない場合は $E_i^{member} = 0$ とする．$E_i^{leader}, E_i^{member}$ の値がどちらも 0 の場合は，次の時刻に再度役割選択を行う．また，チーム履歴がない場合は，t'_{team} の見積りができず，E_i^{leader} を求められない．その場合はランダムに役割を選択する．

4.6 メンバ候補の選択

リーダは信頼度を用いて，チームに参加するメンバ候補 M とサブタスク割り当て仮関数 $\sigma'(s)$ を決定する．ここで，$M \subset A$，$\sigma'(s) = i \in M$ である．まず，リーダは処理できるだけのサブタスク s を自分に割り当て，そのサブタスクの集合を $S_T \subset T$ とする．その後，残りのサブタスクを処理するメンバ候補を選び，そのサブタスクの集合を $S'_T = T \setminus S_T$ とする．まず最初に S'_T の各サブタスクをリソースの大きい順にソートする．これは信頼度の高いエージェントになるべくリソースの大きいサブタスクを割り当てるためである．リーダは $s \in S'_T$ につき s をデッドラインまでに処理できるメンバエージェントを L 体選んでいく．この L をメッセージ重複度と呼び，リーダは一つのサブタスク s につき，L 体のエージェントにチーム参加提案メッセージを送る．リーダはこのメンバエージェントを ε-greedy 戦略で信頼度の高い順に選び，メンバ候補とサブタスク割り当て仮関数 $\sigma'(s)$ を決める．すなわち，各サブ

Require: T: タスク
　A: エージェントの集合
　i: リーダ
　L: 各サブタスクごとに選択するメンバ候補数
　S'_T: リーダ以外が処理するサブタスクの集合
　d_T: タスク T のデッドライン
Ensure: M: メンバ候補
　$\sigma'(s)$: サブタスク割り当て仮関数
1: $M = \emptyset$
2: S'_T をリソースの降順にソート
3: **for all** $s \in S'_T$ **do**
4:　$X = \emptyset$
5:　**for** $a \in A \setminus i$ **do**
6:　　$X \leftarrow X \cup \{s$ を d_T までに処理できるエージェント $\}$
7:　**end for**
8:　**if** $X = \emptyset$ **then**
9:　　*break*; //メンバ候補を探せなかったとし，チーム編成を断念
10:　**end if**
11:　$selected \leftarrow 0$
12:　**while** $selected < L$ **do**
13:　　$j \leftarrow X$ の中から $e_{i,j}$ の値で ε-greedy によって選択したエージェント
14:　　**if** $j \in M$ **then**
15:　　　*continue*;
16:　　**else**
17:　　　$M \leftarrow M \cup j$
18:　　　$\sigma'(s) \leftarrow \sigma'(s) \cup j$
19:　　　$selected \leftarrow selected + 1$;
20:　　**end if**
21:　**end while**
22: **end for**

図 2　メンバ候補の選択方法
Fig. 2　Algorithm for Member Selection

タスク s につき，ε の確率でメンバエージェント L 体をランダムに選び，$1-\varepsilon$ の確率で信頼度の大きいメンバエージェント L 体を選ぶ．リーダが S'_T の全てのサブタスクのメンバ候補を決定できたら，メンバ候補にチーム参加提案メッセージを送る．もし，$s \in S'_T$ をデッドラインまでに処理できるメンバエージェントが見つからなければチーム編成の実施を断念し，次の時刻に再度役割選択から行う．メンバ候補選択のアルゴリズムを**図 2** に示す．

4.7　サブタスクの割り当て

リーダがチーム参加提案返答メッセージを受信し，チーム参加を受諾したエージェントが決まると，それらのエージェントにサブタスク割り当て仮関数 $\sigma'(s)$ を用いてサブタスクを割り当てる．メンバ候補の中で，チーム参加を受諾したエージェントの集合を M' と表す．まず，サブタスク s を割り当て仮関数 $\sigma'(s)$ に従って M' の中のエージェントに割り当てる．ただし，メンバ候補を選ぶ際，一つのサブタスク s につき，L 体のメンバ候補を選びチーム参加提案メッセージを送るため，サブタスク s の処理を複数のエージェントが受諾する可能性がある．その場合は，

表 1　実験におけるパラメータ
Table 1　Parameter Values

パラメータ	値
エージェント数 $\|A\|$	200
リソースの種類数 p	2
タスクを追加する時間間隔 N	10tick
Ntick ごとに発生するタスク数 λ	2（ポアソン分布）
タスクに含まれるサブタスク数 $\|S_T\|$	3 〜 5 のランダム
エージェントの各リソース h_i^k	1 〜 6 のランダム
サブタスクの各リソース r_i^k	60 〜 600（60 刻み） のランダム
タスクのデッドライン d_T	100 〜 400（10 刻み） のランダム
1 サブタスクごとに選択するメンバ候補数 L	2
チーム履歴を保持する上限数 N_o	10
ε-greedy で用いる ε	0.05

表 2　学習パラメータ
Table 2　Learning parameters and Initial Values

パラメータ	値
欲張り度 g_i	0.5
信頼度 $e_{i,j}$	0.5
報酬期待度 $d_{i,j}$	0.5
α_g	0.05
α_e	0.05
α_d	0.05

ε-greedy 戦略により，提案手法で述べた信頼度の高いエージェントにサブタスク s を割り当てる．また，L 体全てのエージェントに拒否された場合は，M' の中からデッドラインまでに処理に余裕があり，信頼度の高いエージェントにサブタスク s を割り当てる．全てのサブタスクの割り当てが決まった後，M' 内のサブタスクを割り当てられていないエージェントはチームから外す．これは，チームに不要に拘束させないためである．もし，全てのサブタスク S'_T の割り当てが完了したら，チーム編成は成功となり，それ以外は失敗となる．

5.　実験と考察

5.1　実験環境

本モデル上で，提案手法の有効性を示すために，評価実験を行う．実験におけるバラメータを**表 1**〜**表 2** に示す．なお，本研究では Ntick ごとに平均 λ のポアソン分布に従いタスクが生成され，キューに追加される．また，今回サブタスクの各リソースを 60〜600 の 60 刻みで設定したが，これは全エージェントが各サブタスクを処理する時間を各サブタスクのリソース量に依存させるためである．これにより，どのエージェントもサブタスクを処理する時間は各サブタスクごとに異なる．

5.2　比較手法

提案手法の評価のために以下の 3 つの比較手法を導入した．

見積もり学習のみ手法

本提案手法の欲張り度，信頼度，報酬期待度の学習を行わない手法を見積もり学習のみ手法とする．つまり，すべての学習率 α_g, α_e, α_d を 0 とした場合の提案手法である．そのため，メンバ候補の選択，リーダからのチーム参加提案メッセージの選択，エージェントへのサブタスクの割り当てはエージェントのリソース量のみに基づいて行う．ただし，4.4 節で述べたチーム履歴を用いたタスク処理の見積もりの学習は行う．

ランダム手法 1

チームの編成の過程において欲張り度，提案受託期待度，報酬期待度の学習を行わず，さらにエージェントのリソースを考慮せずランダムにメンバー選択する手法をランダム手法 1 とする．そのため，メンバ候補の選択，リーダからのチーム参加提案メッセージの選択，エージェントへのサブタスクの割り当てはすべてランダムに選択する．ただし，4.4 節で述べたチーム履歴を用いたタスク処理の見積もりの学習は行う．

ランダム手法 2

上記のランダム手法 1 で，チーム履歴を用いたタスク処理の見積もりを行わない手法をランダム手法 2 とする．このとき，キュー内のタスクを選択する際は，常にマークのついていない先頭のタスクを選択することとなる．

本実験では，タスク処理成功数とチーム編成実施断念回数の時間推移，エージェント一体あたりのサブタスク処理時間とチーム処理時間の平均差分を調べた．なお，チーム編成実施断念回数は，チーム編成の過程においてリーダがタスク処理の見積もりを誤ったため，サブタスクを処理するメンバ候補を探せず，そこでチーム編成の実施を断念する回数である．また，エージェント一体あたりのサブタスク処理時間とチーム処理時間の平均差分 t_{diff} は以下の式で求められるものである．

$$t_{diff} = \frac{t_{team} - t_s^i}{|G|} \tag{17}$$

ここで，t_{team} はチーム処理時間，t_s^i はエージェント i によるサブタスク s の処理時間，$|G|$ はチーム内のエージェント数を表す．

5.3 実験結果

500tick ごとのタスク処理成功数，500tick ごとのチーム編成実施断念回数，エージェント一体あたりのサブタスク処理時間とチーム処理時間の平均差分を**図 3**〜**図 5** に示す．

図 3 において，提案手法と見積もり学習のみ手法を比較すると，提案手法が 25000〜50000tick あたりまで上昇し，見積もり学習のみ手法が 75000tick あたりまで上昇している．これは，エージェントが欲張り度，信頼度，報酬期待度の 3 つのパラメータを学習し，適切なエージェントとチームを組めたことを示している．また，図 3 において，見積もり学習のみ手法，ランダム手法 1 は提案手法ほどではないが，値が上昇している．これはランダム手法 2 と比較すると明らかであるが，チーム履歴を用いたタスク処理の見積もりによる効果が表れている．すなわち，処理の難しいタスクは最初から選択せず，処理できそうなタスクから処理するようになったと考えられる．

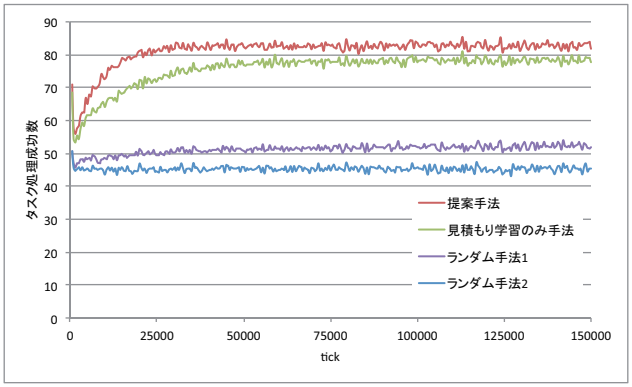

図 3 500tick ごとのタスク処理成功数
Fig. 3 Number of Completed Tasks per 500 tick

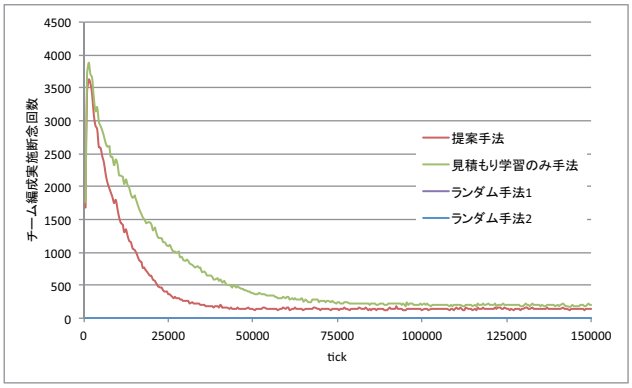

図 4 500tick ごとのチーム編成実施断念回数
Fig. 4 Number of Team Formation Failures due to Incorrect Estimation per 500 tick

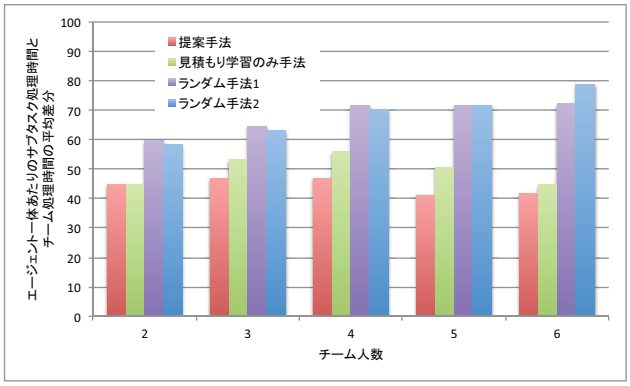

図 5 エージェント 1 体あたりのサブタスク処理時間とチーム処理時間との平均差分
Fig. 5 Average Difference between Time when Agents Finish Executing Subtasks and that when Finish as Teams

次に，**図 4** において，提案手法と見積もり学習のみ手法を比較すると，提案手法が見積もり学習のみ手法よりも速く収束し，値も提案手法の方が小さい．これより，エージェントが 3 つのパラメータを学習することで，早い段階でタスク処理の見積もり精度を高めることができ，より早く無駄なチーム編成の実施を防ぐことができたと言える．また，ランダム手法 1 とランダム手法 2 の値は常に 0 である．これはエージェントのリソースを未知とし，リソースを考慮せずランダムにメンバ候補を選択するため，チーム編成の実施を断念することがないからである．

最後に，図 5 から，全てのチーム人数の場合において，提案手法の値が最も小さくなっている．これより，提案手法による 3 つのパラメータの学習を行うことで，エージェントはあまり自分を拘束しないようなエージェントとチームを組むようになり，エージェントが必要以上にチームに拘束されることが少なくなったと考えられる．しかし，ランダム手法 1 とランダム手法 2 を比較すると，値の差はあまり見てとれない．したがって，提案手法のタスク処理の見積もりによる効果はないと考えられる．

5.4 考察

図 3 において，見積もり学習のみ手法とランダム手法 1 の値は時間と共に上昇している．これはタスク処理の見積もりによる効果の表れであるが，時間と共に上昇した理由は，エージェントがチーム編成の度にチーム履歴を更新するため，チーム編成の回数が増えていくにつれてタスク処理の見積もりの精度が上がったためだと考えられる．

また，値がどの手法も最初の 500tick までは急激に下がっている．これは，本実験で設定したデッドラインによるものだと考えられる．実験が開始された最初は，キュー内にデッドラインの迫っているタスクはあまり存在しないため，キュー内のタスクを処理しやすいが，時間が経過していくにつれキュー内にデッドラインの迫っているタスクが増えると，タスク処理が困難になる．これが 500tick まで値が急激に下がる原因であると考えられる．図 4 においても，提案手法と見積もり学習のみ手法の値が最初の 500tick までは急激に上がっているが，これも同様な理由であると考えられる．

6. 結論

本研究では，デッドラインのあるタスクをチームとして処理する条件の下，効率的なチーム編成手法を提案した．エージェントに役割を持たせ，学習することにより，チーム編成を効率化し，タスク処理率を上げた．さらに，エージェントは自分を不要に拘束しないようなエージェントとチームを組めるようになり，拘束時間も減少できた．また，チーム履歴からタスクの処理を見積もる方法を提案した．処理の難しいタスクは無理に処理しないようにすることで，無駄なチーム編成の実施回数を減らし，その分タスクの処理率向上に成功した．

今後は，タスクの優先度の付与，タスクの負荷の変化など，多様な条件のもとでのチーム編成を検証する．また，大規模化やエージェントのリソースを未知とし，代わりに学習を導入することも考える．

参考文献

[1] Berman, F., Fox, G. and Hey, A. J.: *Grid computing: making the global infrastructure a reality*, Vol. 2, John Wiley and sons (2003).
[2] Génin, T. and Aknine, S.: Coalition formation strategies for self-interested agents in task oriented domains, *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, Vol. 2, IEEE, pp. 205–212 (2010).
[3] Guerrero, J. and Oliver, G.: Multi-robot coalition formation in real-time scenarios, *Robotics and Autonomous Systems*, Vol. 60, No. 10, pp. 1295–1307 (2012).
[4] Hamada, D. and Sugawara, T.: Autonomous decision on team roles for efficient team formation by parameter learning and its evaluation, *Intelligent Decision Technologies*, Vol. 7, No. 3, pp. 163–174 (2013).
[5] Hayano, M., Hamada, D. and Sugawara, T.: Role and member selection in team formation using resource estimation for large-scale multi-agent systems, *Neurocomputing*, Vol. 146, No. 0, pp. 164–172 (2014).
[6] Nanjanath, M., Erlandson, A. J., Andrist, S., Ragipindi, A., Mohammed, A. A., Sharma, A. S. and Gini, M.: Decision and coordination strategies for robocup rescue agents, *Simulation, Modeling, and Programming for Autonomous Robots*, Springer, pp. 473–484 (2010).
[7] Parker, J. and Gini, M.: Tasks with cost growing over time and agent reallocation delays, *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 381–388 (2014).
[8] Ramchurn, S. D., Farinelli, A., Macarthur, K. S. and Jennings, N. R.: Decentralized Coordination in RoboCup Rescue, *Computer journal*, Vol. 53, No. 9, pp. 1447–1461 (2010).
[9] Ramchurn, S. D., Polukarov, M., Farinelli, A., Truong, C. and Jennings, N. R.: Coalition formation with spatial and temporal constraints, *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 3-Volume 3*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1181–1188 (2010).
[10] Shehory, O. and Kraus, S.: Methods for task allocation via agent coalition formation, *Artificial Intelligence*, Vol. 101, No. 1, pp. 165–200 (1998).
[11] Smith, R. G.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, *Computers, IEEE Transactions on*, Vol. 100, No. 12, pp. 1104–1113 (1980).