Vol.2016-ICS-182 No.9
2016/3/2

継続的に発生する優先度つきタスクの効率的割り当て手法の一解法について

飯嶋 直輝[1,a)]　齋藤 健吾[2]　早野 真史[2]　菅原 俊治[2]

概要：本論文では分散環境で継続的に発生する優先度つきタスクの割り当て問題を定義し，この問題の一解法を提案する．近年 IoT（Internet of Things）や電子商取引などのインターネット上のサービスが増加している．これらの問題は，購入者（エージェント）と対象物（オブジェクト），あるいはタスクとそれを処理する計算機（エージェント）間の効果的で効率的な割り当てを発見することと考えられる．しかし，エージェントとタスク（オブジェクト）との割り当ては基本的に組み合わせ問題であり，さらにタスクには優先度や時間制限もあるため，これらを考慮して効率的にタスクの割り当てを決定する必要がある．本研究では，時間とともに継続的にタスクが発生する環境を想定し，優先度と処理効率を考慮しながらも効率的にネットワーク上のエージェントに割り当てる手法を提案する．評価実験のために得られた解と，仮に本課題を整数計画問題として表現し，線形計画ソルバー CPLEX で求められた最適解との比較を試み，提案手法の有用性を示す．

キーワード：タスク割り当て問題，線形計画問題，CPLEX

Naoki Iijima[1,a)]　Kengo Saito[2]　Masashi Hayano[2]　Toshiharu sugawara[2]

1. はじめに

近年，コンピュータ技術を用いたシステムは人々の生活に必要不可欠なものとなっている．例えばインターネット上のサービスや電子商取引，更には道路に設置されたセンサーから交通量や車両の情報を集め，交通情報を提供するサービスなど，応用の幅は多岐にわたる．これらの問題は，購入者（エージェント）と対象物（リソース），あるいはタスクとそれを処理する計算機（エージェント）間の効果的で効率的な割り当てを発見することと考えられる．しかし，これらの技術の発展や新サービスの開発とともに，その情報量や要求タスク数は今後とも増加し，リアルタイム性も要求される．さらに個々のタスクやサービスには優先度や重要度が付与されることもある．従って優先度と処理効率面からのタスクとエージェント間の割り当ての適切さを両立させる効率的かつ効果的な手法を実現する必要がある．

リソース割り当ての問題は，割り当てに必要な情報が既知であれば組み合わせ問題となり，整数 · 線形計画法などの手法を用いて最適な解を求められる [1]．しかし，情報量が増えるにつれて計算量が急激に増加するほか，情報やリソース量が途中で変化すると割り当ての最適解も変わるなどの問題もある [2][3]．

このため現実的な時間内で準最適解を求める研究が多く存在する．例えば [2] ではクラウドコンピューティング環境において，タスクがランダムに発生する環境でバーチャルマシン (VM) をユーザーに効率よく割り当てるアルゴリズムを提案している．ここでは事前に情報がある状態で割り当てた最適解と提案したアルゴリズムとの解の比較をし，最適解に近い値を高速に求めることに成功している．また [4] ではエージェントが複数のリソースに対して価値と希望順位を宣言し，その希望順位を優先しながら全体の合計金額を最大化する問題を定式化し，その準最適解を効率的に求めるアルゴリズムを提案している．この手法では任意の時点でアルゴリズムを中断させたり，新たなリソースやエージェントの情報を追加して再開させることができる性質を持つ．これはリソースをタスク，エージェントをその実行主体と考えると，継続的に与えられるタスクや

1　早稲田大学基幹理工学部情報理工学科
2　早稲田大学基幹理工学研究科情報理工学専攻
a)　n.iijima@isl.cs.waseda.jp

エージェントの処理状況などの変化を迅速に反映させることを示し，実システムで有効な性質と考えられる．

そこで本研究では [4] のモデルを継続的にタスクが発生する環境におけるタスク割り当て問題に対応させ，リソースをタスク，エージェントの希望順位をエージェントのタスクに対する不得意度，また価値をタスクの処理時間として [4] の提案手法を応用した一解法を示す．不得意度に関しては第 3.3 節で述べる．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 節では動的タスク割り当て問題の問題定義と不得意度について述べる．第 4 節では動的タスク割り当て問題における一解法を提案する．第 5 節では提案手法の有用性を示すため，線形計画ソルバー CPLEX と比較した評価実験，考察を行う．最後に結論と今後の課題について述べる．

2. 関連研究

第 1 節で述べたように，効率的なリソース割り当て問題に関する研究は数多くある．例えば [3] ではクラウドソーシング環境において，リクエスタからのタスクを効率よくエージェントに割り当て，リクエスタが得られる利得を最大化する手法を提案している．ここではエージェントの能力は未知とし，幾度かの試行からエージェントの能力を学習し,，それに基づいて割り当てを行うことで，高い利潤を得ることに成功している．しかしここではタスクの利得とエージェントの能力のみに着目しており，エージェントの希望順の概念はない．また [5] ではクラウドコンピューティング環境を想定し，ユーザからのタスクを AHP(階層分析法) を用いて重み付けをし，その重みに従ってタスクを適切に各計算リソースに割り当てるのかを決定している手法を提案している．各タスクの，例えば CPU の必要資源量やコストの上限値等の要望をタスクの評価要素とし，評価指標に重みを付ける．この評価指標に従って，タスクの重みを数値化している．しかしこの方法ではどの評価指標を重視するか，また重みを付けたタスクの計算リソースへの割り当て法については議論されていない．

利得の最大化のほか，公平性に着目した研究も多くある [6][7][8]．割り当ての公平性においては envy-free という概念があり，これは自分への割り当てよりも，他人に与えられた割り当てを自分が割り当てられたほうが効用が大きくならないという概念である．これは割り当てられるものに対する希望順位がない場合にはすべてのエージェントに同じ量の割り当てで解決できるが，希望順位がある場合にはそれを考慮する必要がある．

[9] ではある複製が可能な 1 つの財 (映像データや音楽データ等) に対して複数のエージェントが希望価格を入札し，各エージェントが他のエージェントの割り当てに不満を持たない範囲で利得が最大となる割り当ての研究をしている．この研究ではある閾値となる値段を決め，その値段以上のエージェントが閾値と同じ値段を払って財を割り当てられる．この閾値を得られる利得が最大となるように決定することで不満が出ない範囲の利得が最大となる割り当てを実現している．[10] ではあるひとつの分割できない財（家や高価なもの）とそのほかの分割できる財（金銭など）を複数のエージェント（人など）に分配するときの公平な割り当てを研究している．この研究では分割できない財を A，分割できる財の全体を M，エージェント全体の集合を N としたときに，各エージェントが $A + m_i = (M - m_i)/(|N| - 1)$ となるような分割できる財の値 m_i をもち，この m_i の値が一番小さいエージェントに A と m_i を，それ以外のエージェントに $(M - m_i)/(|N| - 1)$ を割り当てることによって公平な分割を実現している．

3. 問題設定と準備

3.1 Single-Object Resource Allocation with Preferential Orders (SORA/PO) [4]

エージェントの集合を $A = \{1, \ldots, n\}$，リソースの集合を $G = \{1, \ldots, m\}$ とする．G の A への割り当てとは，直積 $G \times A$ の部分集合 $L(\subset G \times A)$ である．各エージェントは多くとも一つのリソースのみ割り当てられるものとする．また $g_L(i)$ を i が割り当てられたリソース，$a_L(g)$ を g が割り当てられたエージェントとし，$g_L(i) = null, a_L = null$ のときは i, j が割り当てられなかったことを意味する．

エージェント i はリソース G に対する素集合族 $G^i_1, \ldots, G^i_{N_i}$(ただし $G^i_m \in G$) をもち，それに対応した i の価値 V^i_m をもつ．G^i_m のリソースは i の第 m 番目の希望の集合を表し，この値が小さいほど希望順位は高い．$B^i = \{(G^i_1, V^i_1), \ldots, (G^i_{N_i}, V^i_{N_i}\})$ を G に対する i の希望と呼ぶ．また割り当て $L \in \Lambda$ における i の割り当てられたリソースの価値を $v_L(i)$，希望順位を $d_L(i)$ とする．ここで，SORA/PO を以下に定義する．

定義

リソースの集合を G とし，$B = \{B^i = \{(G^i_1, V^i_1), \ldots, (G^i_{N_i}, V^i_{N_i})\}\}_{i \in A}$ を希望の集合とするとき，SORA/PO は以下の二つの条件を満たし，価値の総和を最大にする割り当て $L^* \in L$ を求めることと定義する．

$$TV(G, A, B, L^*) = \sum_{(g,i) \in L^*} V^i_{d_{\Lambda^*}(i)} \tag{1}$$

$$\left.\begin{array}{l} \forall (g, i) \in L^* \cup 0 < \forall k < d_{L^*}(i) \\ if h \in G^i_k, a_{L^*}(h) = j then V^j_{d_{L^*}(j)} \geq V^i_k \end{array}\right\} \tag{2}$$

これはエージェント i が割り当てられたリソース g よりも希望順位の高いリソース h が他のエージェント j に割り当てられるとき，j の h に対する価値 $V^j_{d_{L^*}(j)}$ よりも i の h に対する価値 V^i_k が高いことを示している．

[4] では 4 つの手法が提案されているが，ここではそのひとつの LNVF について説明する．基本戦略はエージェン

トに最も価値が高く，希望も高いリソースを割り当て候補とし，その中から割り当てを決定する．具体的には，まず $H(g)$ を $g \in G$ に対して最も高い価値を表明しているエージェントの集合とする．そしてその中で最も高い希望順位を付けられているリソースを割り当て候補とする．ここで割り当て候補がひとつであればそのリソースを割り当てるが，候補が複数ある場合は 2 番目の価値が低い順 (Lowest Next Value First，LNVF) の手法を用いる．これは候補のリソースの中で，価値が 2 番目に低い価値が一番小さいリソースを割り当てる手法である．提案手法では全体の価値を最大化するだけでなく，各エージェントの希望順位を考慮した割り当てができる．また組み合わせ問題における準最適解を効率的に求められること加え，リソースを一つずつ割り当てるという性質から割り当ての途中で新たな希望が発生しても，その希望をその後の割り当てに反映できる．

そこで継続的に優先度つきのタスクが発生する環境における動的タスク割り当て問題を定義し，タスクの優先度，タスクの処理時間に加え，エージェントのタスクに対する不得意度を導入し，これらを考慮した割り当て手法を，[4] の提案手法を応用して提案する．

3.2 動的タスク割当て問題

本研究ではタスクの実行に求められるリソースの構造がエージェントのもつリソースのそれと類似しているときに適切な割り当てと考え，その条件の下で全体の処理時間を最小化する割り当てを目指す．発生したタスクの処理主体をエージェントとし，この集合を $A = \{1, \cdots, n\}$ とする．エージェント i はタスクを処理する能力としてリソース $R_i = \{u_i^1, \cdots, u_i^h\}$ をもつ．ここで h はリソースの種類数を表し，u_i^k はリソースの量を表す．この値が大きいほどエージェントの能力が高いことを表わす．

次にタスクの集合を $G = \{g_1, \cdots, g_m\}$ とおく．タスク $g_j (1 \leq j \leq m)$ は優先度 $P_{g_j} (1 \leq P_{g_j} \leq \gamma, \gamma \geq 1)$ と処理に要求されるリソース $R_{g_j} = \{r_{g_j}^1, \cdots, r_{g_j}^h\}$ をもち，P_{g_j} の値が小さいほど優先度が高いものとする．本研究の問題設定で優先度は，時間的な緊急度ではなく他のタスクとの相対的な重要度を表し，タスク処理におけるデッドラインには余裕のあるものとする．G の A への割り当てとは，直積 $G \times A$ の部分集合 $L (\subset G \times A)$ である．ここでは離散時間の単位として tick を導入し，1 tick 毎にタスクが発生するものとする．タスクの処理の可否と時間を以下のように決める．タスクに対し，1 tick 毎にタスクの各リソースからエージェントの各リソースを減算し，すべてのリソースが 0 以下となったとき処理完了とする．つまり，処理完了までにかかる処理時間 $T_{g_j}^i$ は

$$T_{g_j}^i = \max_{1 \leq k \leq h} \lceil r_i^k / u_{g_j}^k \rceil$$

と表せる．タスクを割り当てられたエージェントは，タスク処理が完了するまで次のタスクを処理できず，ここでは割り当てをしないものとする．このときある時点で未処理のタスクの集合を $G = \{g_1, \cdots, g_m\}$ としたとき，タスク処理時間の総和は

$$T_L = \sum_{i, g_j \in L} T_{g_j}^i$$

と表せる．動的タスク割り当て問題は，タスクの優先度を考慮しつつ全体の処理時間 T_L を最小化する問題と定義する．

3.3 エージェントの不得意度

ここでエージェントのタスク g_j に対する不得意度を定義する．エージェントがタスクを処理する際，エージェントのリソースが無駄なく使用されることを想定する．そのためエージェント i のタスク g_j に対する不得意度を以下のように定義する．

$$C_{g_j}^i = \max_{1 \leq k \leq h} \lceil r_i^k / u_{g_j}^k \rceil - \min_{1 \leq k \leq h} \lceil r_i^k / u_{g_j}^k \rceil$$

これは，タスクの持つリソースの比率とエージェントの持つリソースの比率が異なるのかを表す．この値が大きいほど，エージェントはタスクの要求しているリソースに対して偏ったリソース（または異なるリソース構造）を持つことを示す．

4. 提案手法

4.1 マーケット形式の処理時間宣言

本研究では動的タスク割り当て問題における割り当てタスクの選択において，各エージェントが宣言する各タスクに対する処理時間とタスクの優先度，タスクに対する不得意度を用いて決定する．タスクの集合 G に対し，エージェント i はタスク処理にかかる時間と不得意度を求め，

$$D^i = \{(T_{g_1}^i, C_{g_1}^i), \cdots, (T_{g_m}^i, C_{g_m})\}$$

を宣言する．この情報の集合を処理情報と呼ぶ．処理情報とタスクの優先度を基にして第 4.2 節のアルゴリズムに従って割り当てるタスクを決定する．

4.2 割り当てタスク決定アルゴリズム

本研究では [4] におけるアルゴリズムを拡張し，価値を処理時間，希望順位を不得意度（希望順位は値が小さいほど高いため）と考え，タスクとエージェントのリソース構造の類似性を考慮した効率のよい割り当てを目指す．また優先度に関しては効率を重視しつつ，その中で優先度の高いタスクを選択する．

本アルゴリズムの基本戦略は，まず処理情報の集合 $D = \{D^i | i \in A\}$ の中から各タスクにつき処理時間が最も短く，その中で不得意度が低い処理情報を選び，その処理情報の集合の中から不得意度が最も低い処理情報を抜き

Algorithm 1 タスク割り当て決定アルゴリズム

タスクの集合:$G = \{g_1, g_2, \cdots, g_m\}$
エージェントの集合：$A = \{1, 2, \cdots, n\}$
利得と優先度の組の集合：$D = \{D^1, \cdots, D^n\}$
タスク処理中のエージェントの集合：$A_{exe} = \emptyset$ //初期は空集合
K：保持するタスクの量
while $|G| \geq K$ **do**
  $D_{max} \leftarrow MinProcess(D)$
  $D_{cand} \leftarrow MaxGood(D_{max})$
  $D_{choice} \leftarrow MaxPriority(D_{cand})$
  if $|D_{choice}| = 1$ **then**
    $\alpha \leftarrow GetAgent(D_{choice})$
    $g_\beta \leftarrow GetTask(D_{choice})$
  else
    $D_{choice} \leftarrow D_{choice}$ の中からひとつランダムに選んだタスク
    $\alpha \leftarrow GetAgent(D_{choice})$
    $g_\beta \leftarrow GetTask(D_{choice})$
  end if
  g_β を α に割り当てる
  $G \leftarrow G \setminus g_\beta$
  $A_{exe} \leftarrow A_{exe} \cup \alpha$
end while

出す．ここで抜き出した処理情報が一つの場合は，[4] では LNVF 等の手法を用いて割り当てるタスクを選択するが，ここではタスクの優先度が高く，その中で処理時間が短くなるタスクを選択する．処理時間が同じタスクが複数存在する場合はその中からランダムで一つを選択する．タスク割り当てアルゴリズムを $Algorithm1$ に示す．ここで $MinProcess(D)$ は処理情報の集合 D の中から，各タスクにつき処理時間が最も短く不得意度が低い処理情報の集合を返す関数であり，$MaxGood(D)$ は処理情報の集合 D の各要素に対し，不得意度 $C^i_{g_k}$ が最も低い処理情報の集合を返す関数である．また $MaxPriority(D)$ は処理情報の集合 D の中でタスクの優先度が最も高い処理情報の集合を返す関数であり，$GetAgent(G)$,$GetTask(G)$ はそれぞれ，処理情報の集合 D の要素がひとつのとき，そのエージェントとタスクを返す関数である．

4.3 割り当ての最適解

タスクが継続的に発生する環境において，割り当ての最適解は刻々と変化する．例えば以下のようにエージェント，タスクが存在したとする．

- $i : R_i = \{4, 6, 6\}$,$j : R_j = \{3, 5, 8\}$
- $g_k : R_{g_k} = \{60, 75, 90\}$

このとき $T^i_{g_k} = 15, C^i_{g_k} = 2$ であり，$T^j_{g_k} = 15, C^j_{g_k} = 3$ であるため，同じ処理時間で効率のよい i に g_k が割り当てられる．しかし，次に $g_l : R_{g_l} = \{68, 84, 84\}$ が発生すると $T^i_{g_k} = 14, C^i_{g_k} = 0$ であり，$T^j_{g_k} = 17, C^j_{g_k} = 5$ となる．このとき最も効率がよい割り当ては $L = \{(g_k, j), (g_l, i)\}$ であるが，既に i に g_k を割り当てているため $L = \{(g_k, i), (g_l, j)\}$ となり，効率の悪い割り当てとなる．そこで，提案手法ではある一定のタスクを割り当てずに保持する．タスクを保持することで割り当てるタスクの選択の幅が広げ，効率のよいタスク割り当てを行う．

表 1 実験環境

パラメータ	値
tick	10000
エージェント数	100, 1000
タスクの種類:h	3
発生タスク数	$3 \sim 4, 8 \sim 9$
タスク保持数	100

5. タスク割り当て手法の評価実験

5.1 実験環境

動的タスク割り当て問題においてタスクの発生量やエージェント数を変化させ，様々な環境を想定して評価実験を行う．エージェント，タスクの持つリソースは 3 種類とする．エージェントの持つリソースの各値は最低値を 3，全リソースの合計が 15 ～ 18 となるようランダムに設定し，タスクの要求するリソースは各値の最低値を 30，全リソースの合計が 150 ～ 180 のランダム値で設定した．本実験で設定した実験パラメータを表 1 に示す．

本研究では，提案手法の比較として線形計画ソルバー CPLEX を用いる．しかし本研究では割り当てるタスクの情報が既知ではないため整数計画問題として解くことができない．そこでタスクが一定量貯まる（この値を V とする．ただし V は正の整数）まで待ち，その時点で存在するタスクとエージェントとの割り当てを組み合わせ問題とし，処理時間を最小化する整数計画問題として CPLEX を用いて割り当てを行う．

5.2 結果と考察

本実験では CPLEX を用いた割り当て手法と V タスクを保持して増えた分をひとつずつ割り当てる手法，そして発生したタスクをすぐに割り当てる手法の 3 つの手法を調べた．以下，これらの手法を CPLEX，AOBO (assign one by one)，AOBO/I (assign one by one immediately) と表記する．以下に示す実験結果は 20 回の実験の平均値を取ったものである．

各実験の合計処理時間の推移を**図 1**，**図 2**，**図 3**，**図 4** に示す． 図 1 より CPLEX よりも AOBO，AOBO/I の方が処理時間が短くなったことがわかる．これは，AOBO ではアルゴリズムにより未割り当てのタスクの中で効率よく処理できるエージェントからタスクを割り当てていくためである．また AOBO/I ではタスクを保持しないが，タスクを終了したばかりの効率的なエージェントに直ぐに新しいタスクを割り当てられるためであると考えられる．

一方で図 2 では, CPLEX の処理時間が最も短く, AOBO/I の処理時間が大きく増加した．これは発生タスクが少ない

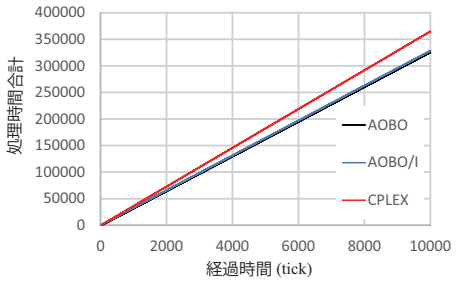


図 1 合計処理時間合計の推移 *agent* : 100, 発生タスク : 3 ∼ 4

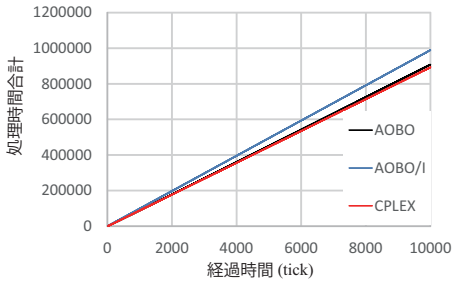


図 2 合計処理時間の推移 *agent* : 100, 発生タスク : 8 ∼ 9

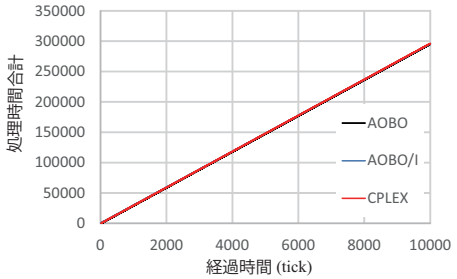


図 3 合計処理時間の推移 *agent* : 1000, 発生タスク : 3 ∼ 4

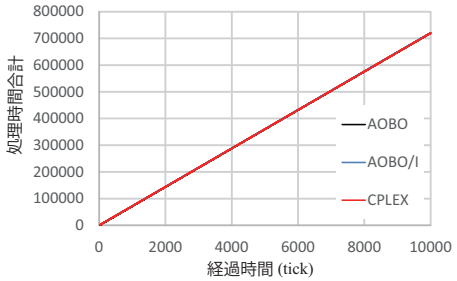
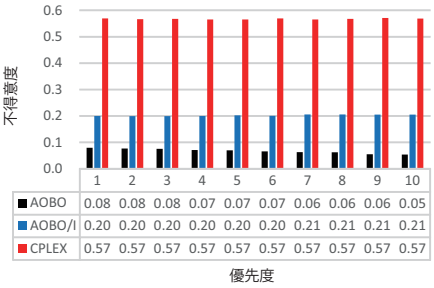


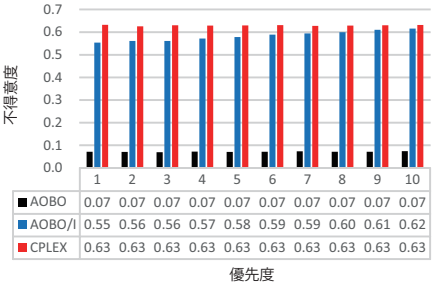
図 4 合計処理時間の推移 *agent* : 1000, 発生タスク : 8 ∼ 9

ときには，タスク割り当て時に未割り当てのエージェントが多く存在するために効率的なエージェントを選んでタスクを割り当てることができるが，タスクが多いときにはタスク処理中のエージェントが増え，少数のエージェントの中から選択しなければならないことが原因である．また図 2 において AOBO/I と AOBO の差が拡大したのは，未割り当てのエージェントが少ないうえに，AOBO/I では発生した少数のタスクに関する処理情報を提示するのに対し，AOBO では保持している多数のタスクに関する処理情報を提示できたためと考えられる．また図 3，図 4 からエージェント数が多いときには 3 つの手法にほとんど差が



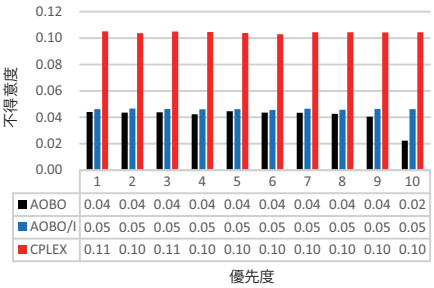
	1	2	3	4	5	6	7	8	9	10
AOBO	0.08	0.08	0.08	0.07	0.07	0.07	0.06	0.06	0.06	0.05
AOBO/I	0.20	0.20	0.20	0.20	0.20	0.20	0.21	0.21	0.21	0.21
CPLEX	0.57	0.57	0.57	0.57	0.57	0.57	0.57	0.57	0.57	0.57

図 5 各優先度のタスクの平均不得意度
agent : 100, 発生タスク : 3 ∼ 4



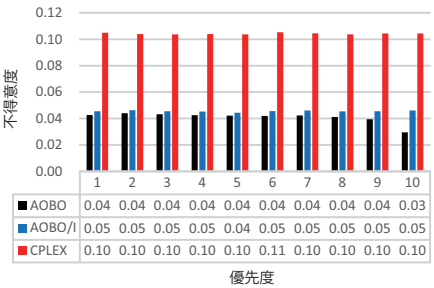
	1	2	3	4	5	6	7	8	9	10
AOBO	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
AOBO/I	0.55	0.56	0.56	0.57	0.58	0.59	0.59	0.60	0.61	0.62
CPLEX	0.63	0.63	0.63	0.63	0.63	0.63	0.63	0.63	0.63	0.63

図 6 各優先度のタスクの平均不得意度
agent : 100, 発生タスク : 8 ∼ 9



	1	2	3	4	5	6	7	8	9	10
AOBO	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.02
AOBO/I	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
CPLEX	0.11	0.10	0.11	0.10	0.10	0.10	0.10	0.10	0.10	0.10

図 7 各優先度のタスクの平均不得意度
agent : 1000, 発生タスク : 3 ∼ 4



	1	2	3	4	5	6	7	8	9	10
AOBO	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.03
AOBO/I	0.05	0.05	0.05	0.05	0.04	0.05	0.05	0.05	0.05	0.05
CPLEX	0.10	0.10	0.10	0.10	0.10	0.11	0.10	0.10	0.10	0.10

図 8 各優先度のタスクの平均不得意度
agent : 1000, 発生タスク : 8 ∼ 9

出ない結果となった．これは未割り当てのエージェントが多いため，多くのエージェントからタスクの割り当てを実行できたからと考えられる．

次に各優先度のタスクを処理したエージェントの不得意度を調べるために，各優先度のタスクの平均不得意度を**図 5**，**図 6**，**図 7**，**図 8** に示す． 図 5 から，AOBO，

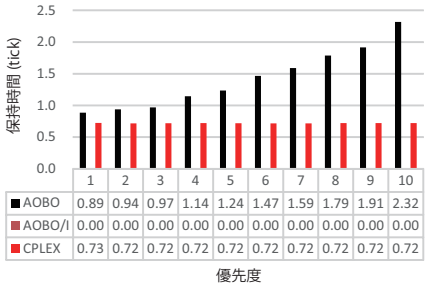


図 9 各優先度のタスクの平均未割り当て時間
agent : 100, 発生タスク : 3 ～ 4

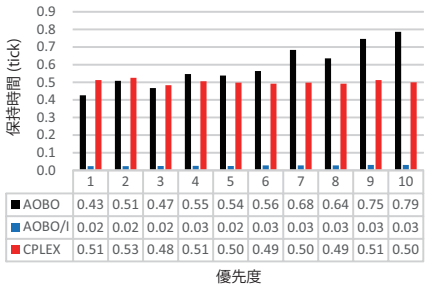


図 10 各優先度のタスクの平均未割り当て時間
agent : 100, 発生タスク : 8 ～ 9

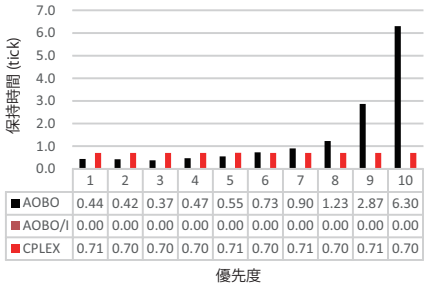


図 11 各優先度のタスクの平均未割り当て時間
agent : 1000, 発生タスク : 3 ～ 4

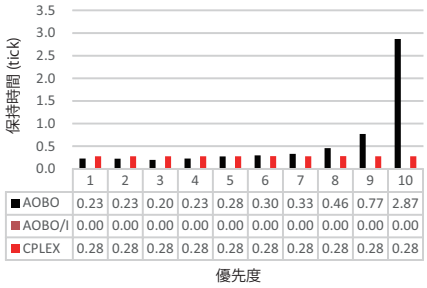


図 12 各優先度のタスクの平均未割り当て時間
agent : 1000, 発生タスク : 8 ～ 9

AOBO/I は CPLEX に比べて不得意度は低い．一方で図 6 では AOBO/I の不得意度は発生タスクが少ない場合と比べて増加している．これは前述の通り，発生タスクの増加により未割り当てのエージェントが減少したことが原因である．一方 AOBO は不得意度が低いエージェントへタスクを割り当てることができた．これらから，一定数タスクを保持することでタスク処理の効率が上昇したといえる．

図 7，図 8 ではタスク発生数の変化による不得意度の変化はない．これは未割り当てエージェントが多いため，タスク発生数が増加しても多くのエージェントが効率よくタスク処理できたためと考えられる．また，総じて CPLEX が不得意度の高いエージェントに割り当てたのは，不得意度ではなく処理時間を最小化することに焦点を当てているため，不得意度が高くても効率的なエージェントにタスクを割り当てるためである．

最後にタスク保持によってタスクが割り当てられなかった時間を調べるため，各優先度のタスクの平均未割り当て時間を**図 9**，**図 10**，**図 11**，**図 12** に示す． 図 9，図 10 から，AOBO では優先度の高いタスクから処理を行うことができている．図 10 図 9 から発生タスクが多いときより発生タスクが少ないほうが未割り当ての時間が増加している．これは発生タスクが少ない方が処理したタスクの総数が少ないためである．図 11，図 12 では下位の優先度のタスクが保持されている時間が極端に増加している．これはエージェント数が多いため効率的なエージェントが増加し，不得意度が低く優先度が高いタスクから割り当てるという提案手法の性質から上位の優先度のタスクが割り当てられたためだと考えられる．

6. まとめと今後の課題

本研究では継続的に発生する優先度つきタスクの割り当て問題を定義し，提案手法と線形計画ソルバー CPLEX を用いた手法との解を比較した．実験ではエージェント数，発生するタスク数を変化させることで各種利得の変化を調べた．結果として提案手法は効率よく優先度の高いタスクを処理できるエージェントにタスクを割り当てることができた．

今後の課題はタスクの発生に遅延が起きたり発生数が変動するなど様々な環境でも実験を行うことである．

参考文献

[1] Shoham, Y. and Leyton-Brown, K.: *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*, Cambridge University Press (2008).

[2] Lee, K., Jeong, J., Yi, Y., Won, H., Rhee, I. and Chong, S.: Max Contribution: An Online Approximation of Optimal Resource Allocation in Delay Tolerant Networks, *Mobile Computing, IEEE Transactions on*, Vol. 14, No. 3, pp. 592–605 (2015).

[3] Ho, C.-J. and Vaughan, J. W.: Online Task Assignment in Crowdsourcing Markets., *AAAI*, Vol. 12, pp. 45–51 (2012).

[4] Saito, K. and Sugawara, T.: Single-object resource allocation in multiple bid declaration with preferential order, *Computer and Information Science (ICIS), 2015 IEEE/ACIS 14th International Conference on*, pp. 341–347 (online), DOI: 10.1109/ICIS.2015.7166617 (2015).

[5] Ergu, D., Kou, G., Peng, Y., Shi, Y. and Shi, Y.: The analytic hierarchy process: task scheduling and resource

allocation in cloud computing environment, *The Journal of Supercomputing*, Vol. 64, No. 3, pp. 835–848 (2013).

[6] Lipton, R. J., Markakis, E., Mossel, E. and Saberi, A.: On approximately fair allocations of indivisible goods, *Proceedings of the 5th ACM conference on Electronic commerce*, ACM, pp. 125–131 (2004).

[7] Aziz, H., Gaspers, S., Mackenzie, S. and Walsh, T.: Fair assignment of indivisible objects under ordinal preferences, *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1305–1312 (2014).

[8] Miltersen, P. B. et al.: Equilibrium Analysis in Cake Cutting, *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)* (2013).

[9] Goldberg, A. V. and Hartline, J. D.: Envy-free auctions for digital goods, *Proceedings of the 4th ACM conference on Electronic commerce*, ACM, pp. 29–35 (2003).

[10] Tadenuma, K. and Thomson, W.: The fair allocation of an indivisible good when monetary compensations are possible, *Mathematical Social Sciences*, Vol. 25, No. 2, pp. 117–132 (1993).