F-033

MAPD 問題における滞在の予約と移動予測時間を用いるタスク割り当て

Reservation of Stay and Task Allocation with Estimated Travel Time for MAPD Problem

下川 真典 *　　松井 俊浩 *
Masanori Shimokawa　　Toshihiro Matsui

1　はじめに

Multi-Agent Path Finding (MAPF) 問題は，既知の環境において複数のエージェントが互いに衝突することなくそれぞれの目的地に向かう経路を求める問題である．この問題は，物流倉庫内の搬送ロボット [1]，航空機の牽引車両 [2] やビデオゲーム内のキャラクター [3] など様々な分野へ応用されている．単一の MAPF 問題では，時間の経過とともに新しいタスクが発生する物流倉庫における集配のような連続的な問題を表現できないため，MAPF 問題を発展させた，Multi-Agent Pickup and Delivery (MAPD) 問題が提案されている [4]．

MAPD 問題は，絶えず発生するタスクの経路を，エージェントに順次割り当てる問題である．この問題は，今日既に活用されている物流倉庫内の搬送ロボットの高度化を目的として広く研究されている．MAPD 問題の解法として，最適解法の Conflict Based Search (CBS) [5] や，準最適解法の Cooperative A* (CA*) [3] に基づく手法が提案されている．CBS は，エージェントごとの最短経路を計算する下位層の探索と，下位層で計算した経路が衝突していないかを確認する上位層の二つに分けて衝突のない経路を計算する．CA*は時間軸と二次元位置座標からなる三次元空間を表すグラフ上で A* アルゴリズム [6] により探索し衝突の回避と経路の探索を同時に計算する．

全ての MAPD 問題が解を持つとは限らないため，解の存在を保証する well-formed な問題とその解法が示されている [7, 4]．Well-formed な問題では，エージェントが他のエージェントの経路を妨げない退避位置で留まることが許される．Well-formed な MAPD 問題の解法として，Token Passing (TP) が提案されている [4]．TP は，継続的に発生するタスクを各エージェントに割り当て，エージェントの退避位置を考慮しつつ CA*を用いて求めた，タスク間の衝突がない経路を予約した後で，タスクを実行する．また，経路の終点に達したエージェントは，新しいタスクが割り当てられるか，他のタスクの妨げになり退避する必要が生じるまで，その位置に滞在する．しかし TP では，経路の終点に滞在していたエージェントが退避する際に，他のエージェントの経路の支障となったり，冗長な移動が生じる場合があると考えられる．

本研究では，経路終端の占有状況をエージェント間で整合させる方法の改善と，冗長な移動を削減するためにタスク間の連続性を考慮するタスク割り当てを提案する．これらにより，各タスクの集荷経路長を削減することにより，タスクが生成されてから完了するまでの時間である service time と，エージェントの総移動経路長の削減を目指す．提案手法が，集荷経路長，service time，総移動経路長，全タスクが完了するまでの時間である makespan を TP よりも削減することを，実験により示す．また，各時刻のタスク数に対して，エージェントの数が十分であるならば，従来手法に比べて提案手法の効果が比較的大きいことを示す．

*名古屋工業大学 Nagoya Institute of Technology

2　研究の背景

本章では，複数エージェントの経路計画問題である MAPF 問題とそれを継続的な問題に発展させた MAPD 問題について述べる．また，全ての MAPD 問題が解を持つとは限らないことと，MAPD 問題の解を保証する well-formed な問題およびその解法について述べる．

2.1　MAPF 問題

Multi-Agent Path Finding (MAPF) 問題は，複数のエージェントが既知の環境下で現在位置からそれぞれの目的地まで互いに衝突することのない経路を求める問題である．人工知能，ロボット工学，制御分野で広く研究されてきた基本的な MAPF 問題は，単一の問題であり，継続的な経路の発生は考慮されない．エージェントの運動制約 [8] や，エージェントの移動に伴う遅延時間 [9] など実世界の特性を考慮する研究があるが，これらは単一の問題である．

2.1.1　MAPF 問題の定義

MAPF 問題は，$\langle A, G, T, EP\rangle$ からなる．$A = \{a_1, a_2, ..., a_m\}$ はエージェントの集合，$G = \langle V, E\rangle$ は頂点 $v \in V$，頂点間の辺 $e(v, v') \in E$ $(v \neq v')$ からなるグラフ，T は離散時刻の集合，$EP = \{(v_i^s, v_i^e)|a_i \in A\}$ は各エージェントの始点と終点の集合である．解はエージェントの経路の集合 $P = \{P_i|a_i \in A\}$ であり，$P_i = (v_t^i, v_{t+1}^i, ..., v_{t+k}^i)$ である．ただし，全ての経路の衝突を回避する必要がある．経路の衝突がないことは以下のように定義される．全てのエージェント $\forall a_i \in A, a_j \in A, a_i \neq a_j$ について，任意の時刻 $\forall t \in T$ における経路の頂点 $v_t^i \in P_i, v_t^j \in P_j$ は $v_t^i \neq v_t^j$ である．また，$v_t^i \neq v_{t+1}^j \wedge v_{t+1}^i \neq v_t^j$ である．経路の衝突を回避しつつ，できるだけ短い経路を求めることが MAPF

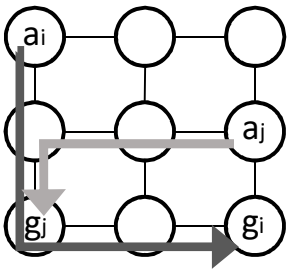

図 1　MAPF 問題におけるエージェント a_i，a_j のそれぞれの目的地 g_i，g_j までの経路

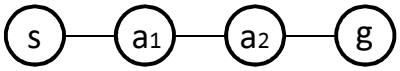

図 2　MAPD の解が無い例 (エージェント a_1, a_2 と，タスクの集荷位置 s, 配達位置 g)

問題の目的である．

基本的な MAPF 問題は，図 1 のようなグリッド世界で表される問題を対象とする．エージェント a_i，a_j にはそれぞれの目的地 g_i，g_j があり，互いの衝突を回避する経路を計算し移動する．全エージェントがそれぞれの目的地に達する時間を最小化する．

2.2　MAPD 問題

基本的な MAPF 問題では継続的な経路の発生は考慮されないため，物流倉庫内での搬送ロボットのように継続的に新しいタスクをエージェントに割り当てる状況は，直接的に表現できない．従来研究では，MAPF 問題を継続的な問題に発展させた MAPD 問題を提案している [4].

2.2.1　MAPD 問題の概要

MAPF 問題を継続的に発生するタスクに対して経路を割り当てる問題に発展させた問題を，Multi-Agent Pickup and Delivery (MAPD) 問題と呼ぶ．MAPD 問題の解法には，動的かつ継続的な，衝突のない経路の割り当てと探索が必要である．新たなタスクの始点と終点は任意の時刻に生じるため，事前の割り当ても探索もできない．したがって，解法の実行中に生じる経路について，随時割り当てと探索をする．MAPD 問題の基本的な目的の一つは各タスクに要する時間の最小化であり，タスクが生成されてから完了するまでの時間である service time で評価する．また，エージェントの総移動経路長や，全タスクの終了に要する時間の最小化も目的とされる．

2.2.2　MAPD 問題の定義

MAPD 問題は $\langle A, G, T, V_{ep}, V_{tsk}, TSK\rangle$ からなる．A, G, T は MAPF と同様，エージェントの集合，グラフ，離散時刻の集合である．$V_{ep} \subset V$ はエージェントが滞在できる頂点の集合であり，$V_{tsk} \subset V_{ep}$ は滞在できる頂点の集合に含まれる，タスクの始点と終点の集合である．$TSK = \{(v_j^s, v_j^g) | v_j^s, v_j^g \in V_{tsk}\}$ は V_{tsk} に含まれる始点と終点を持つタスクの集合である．a_i に割り当てられたタスクの集合を TSK_i とすると，$\forall i \neq i', TSK_i \cap TSK_{i'} = \emptyset$ である．MAPD 問題の解も経路の集合 $P = \{P_{i,\tau_j} | a_i \in A, \tau_j \in TSK\}$ であり，エージェント a_i に割り当てられたタスク τ_j の経路を P_{i,τ_j} と表す．MAPF 同様，全ての経路が衝突してはならない．

MAPF 問題を拡張した部分である，タスクについて説明する．MAPD 問題では，集荷位置 $v_j^s \in V_{tsk}$ と配達位置 $v_j^g \in V_{tsk}$ をもつタスク τ_j が随時，未完了のタスク集合 TSK に追加される．エージェントは，タスクを実行していない $free$ の状態とタスクを実行している $occupied$ の状態の二つに分かれる．タスクは $free$ の状態のエージェントに割り当てられる．$\tau_j \in TSK$ を割り当てられたエージェント a_i は，現在位置から v_j^s を経由して v_j^g に移動する．a_i が v_j^s にたどり着いたとき，τ_j は実行中となり，TSK から取り除かれる．このとき，a_i は $occupied$ の状態となる．そして，v_j^g にたどり着いたとき τ_j は完了となり，a_i は $free$ の状態となる．タスクの集荷位置に到達した時点でエージェントは $occupied$ の状態となるため，集荷位置に達するまでは，他のタスクを割り当てることができる．

2.3　well-formed な問題

全ての MAPD 問題の解が求まるとは限らない．図 2 のような二体のエージェント a_1，a_2 とタスクの集荷位置 s，配達位置 g の関係を考える．エージェント a_1，a_2 は互いの経路を妨げとなり，集荷位置または配達位置のいずれかに到達できないため，どちらのエージェントに割り当ててもこのタスクは実行できず，解は無い．解の存在を保証する well-formed な問題が示されている [7, 4].

2.3.1　Well-formed な MAPD 問題の定義

従来研究では，解の存在が保証される well-formed と呼ばれる問題を対象としている [4]．well-formed な問題は，エージェントが他のエージェントの経路を妨げることなく，滞在できる位置である endpoint を含む．Endpoint のうち，タスクの集荷，配達位置を task endpoint と呼び，それ以外のエージェントの初期位置などを non-task endpoint と呼ぶ．Endpoint の集合を V_{ep}，task-endpoint の集合を V_{tsk} と表し，non task endpoint の集合を $V_{ep} \backslash V_{tsk}$ と表す．エージェントの数を m，non-task endpoint の数を $|V_{ep} \backslash V_{tsk}|$ とするとき，well-formed な MAPD 問題は以下のように定義される．

Definition 1. *Well-formed* な *MAPD* 問題

1. タスクの数は有限．
2. $|V_{ep} \backslash V_{tsk}| > m$
3. どの二つの *endpoint* 間にも，他の *endpoint* を経由しない経路が存在する．

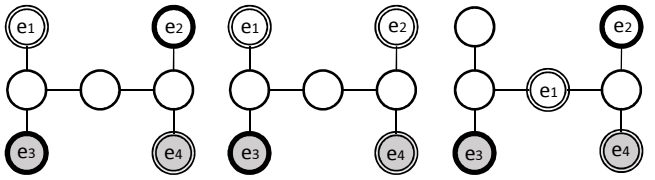

図 3　Well-formed な問題とそうでない問題 (二重丸：task endpoint，太線丸：non-task endpoint，塗りつぶした丸：エージェント)

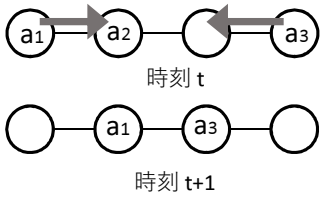

図 4　エージェントの行き場がなくなってしまう例 (エージェント a_1，a_2，a_3)

Well-formed な MAPD 問題には少なくともエージェントの数 $m+1$ の endpoint が必要である．図 3 において，(a) は well-formed な問題である．(b) はエージェントの数より non-task endpoint の数が少ないため，well-formed な問題ではない．(c) は，endpoint e_2 と e_3，e_3 と e_4 の間には他の endpoint を経由しない経路が存在しないため well-formed な問題ではない．

2.4　従来手法：Token Passing

従来研究では，MAPD 問題の解法である Token Passing (TP) が提案されている [4]．この解法は，システムを管理する中央処理により，継続的にタスクが追加され，token と呼ばれる共有メモリを更新する．エージェントが順番に token にアクセスし，CA*を用いて経路の探索と各時刻の頂点の占有を予約してゆく分散処理の手法である．Well-formed な MAPD 問題では，TP アルゴリズムを用いて endpoint 間の経路を各エージェントに割り当てれば，エージェント同士が衝突しない経路が得られることが証明されている [4].

2.4.1　経路の終点における滞在

TP において，経路の終点まで移動したエージェントは，その場に滞在し続けるという規則のもとで経路探索を行う．このような規則が必要となる状況の例を図 4 に示す．時刻 t において，a_1，a_3 は経路に沿って移動しており，a_2 は経路の終点に到達する．次の時刻 $t+1$ において，a_2 の存在が考慮されないと仮定すると，$t+1$ で a_1 が移動し，隣接頂点に a_3 が移動してくる場合，a_2 は行き場を失ってしまう．このような状況を回避するために，TP アルゴリズムでは，エージェントは経路の終端の頂点に到達した後，その場に滞在し続ける．

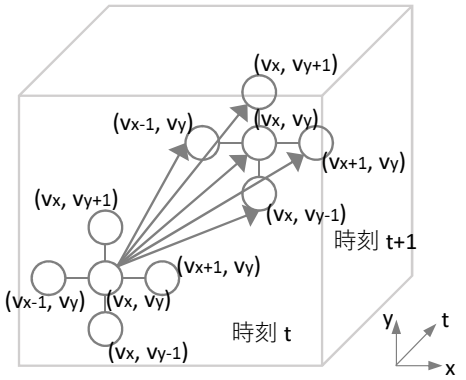

図 5　三次元空間上の A*アルゴリズムの近傍頂点の展開

2.5　経路計算：Cooperative A*

TP アルゴリズムは経路計算に Cooperative A* (CA*) を用いる [3]．CA*は二次元の位置座標と将来の時刻からなる離散的な三次元空間を表すグラフ上で，A*アルゴリズム [6] により経路探索する．二次元の位置座標系では，各時刻においてエージェントはその場に留まるか，隣接頂点に移動する．三次元空間上を時刻 t と頂点 v の座標 x, y を用いて $(t,\, v_x,\, v_y)$ と表す．図 5 の，三次元空間を表すグラフのように，$(t,\, v_x,\, v_y)$ から展開される近傍の頂点は二次元位置座標 $(v_x,\, v_y)$ とその近傍の頂点に対応する時刻 $t+1$ における頂点，

$$
\begin{aligned}
&(t+1,\, v_x,\, v_y)\\
&(t+1,\, v_x-1,\, v_y)\\
&(t+1,\, v_x+1,\, v_y)\\
&(t+1,\, v_x,\, v_y-1)\\
&(t+1,\, v_x,\, v_y+1)
\end{aligned}
$$

である．

TP アルゴリズムは，token に予約された各エージェントの経路を避けるように，新たな経路を CA*により探索する．探索空間上の各頂点を時刻 t と頂点 v の組 $(t,\, v)$ で表す．ある時刻 t において，複数のエージェントが同じ頂点 v に存在してしまうならば，衝突が起こるため，そのような近傍の頂点は展開されない．また，エージェント a_1 が状態 $(t,\, v)$ から $(t+1,\, v')$ に移動し，エージェント a_2 が状態 $(t,\, v')$ から $(t+1,\, v)$ に移動しようとする場合は，同じ辺を逆向きに移動する衝突が起きてしまう．従って，このような状況にある近傍の頂点も展開されない．

2.6　従来手法の問題点

TP において，経路の終点でエージェントが滞在し続けるという規則は，エージェントの経路探索の順序によっては他のエージェントの経路と衝突してしまう可能性がある．また，タスクの割り当ては，エージェントが token にアクセスする順に依存する貪欲法であるため，冗長な移動を含む可能性がある．

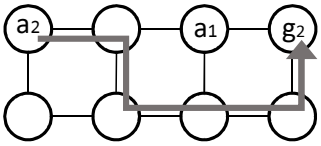

図 6　滞在する想定による衝突回避の例

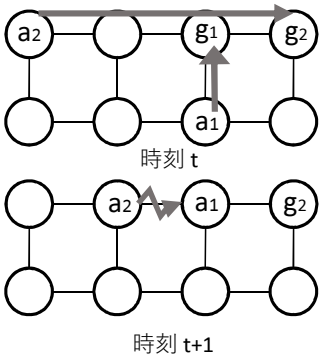

図 7　滞在する想定による衝突の例

2.6.1　経路の終点での滞在に伴う問題点

TP では，エージェントが経路の終点に到達した後，滞在し続ける規則のもとで経路探索を行う．図 6 のようにエージェント a_1 が先に g_1 への経路を計算し，その後 a_2 が g_2 への経路を計算する場合は，a_2 は a_1 が滞在していることが分かっているため，a_1 を避けた経路が割り当てられている．しかし，図 7 のように，先に a_2 が g_2 への経路を計算し，次に a_1 が g_1 への経路を計算する場合，タスクの経路は衝突していないが，滞在する予約は a_2 の経路と衝突してしまう．次の時刻に a_1 が token にアクセスし，新しいタスクを割り当て，経路を探索する場合であっても，他のエージェントの経路によって近傍頂点が埋まっているならば，a_1 は行き場を失ってしまう．

2.6.2　貪欲法による冗長な移動

token にアクセスしたエージェント a_i は割り当て候補の TSK' に含まれる，現在位置に最も近い集荷位置をもつタスク τ_j を自身に割り当てる．タスクの割り当てが無い複数のエージェントが同時刻に token へのアクセスを要求するとき，どのエージェントが先に token へアクセスするかは特に定められていない．エージェント a_1，a_2 が同時刻に token を要求するとき，先に token へアクセスした a_1 が割り当てたタスク τ の集荷位置が a_1 の現在地よりも a_2 の現在位置に近い場合，a_1 が τ を集荷する経路は a_2 が集荷する経路よりも長くなる場合があると考えられる．

このような問題を緩和することを意図した TP の拡張として Token Passing with Task Swaps (TPTS) が提案されている [4]．この解法は，エージェント a_1 が受け持っているタスク τ の集荷位置に移動している間に，より集荷位置に近いエージェント a_2 が存在すれば，a_2 に τ を割り当てる．タスクがなくなった a_1 は現在位置が endpoint でなければ，endpoint へ退避する経路を計画する．TP 同様，TPTS も endpoint から endpoint の経路であれば，衝突のない経路を計画できる．しかし，タスクがなくなる位置は必ずしも endpoint であるとは限らないため，全ての解を求めることはできない．また，a_1 が τ の集荷位置に向かう移動と，退避位置に向かう移動は無駄になる可能性がある．

3　タスク間のエージェントの冗長な行動を削減する提案手法

本章では，TP の二つの問題点の改善手法を提案する．先ず，経路の終点に滞在するエージェントによる他のエージェントの経路の阻害を改善するために，あらかじめ滞在することも予約する手法を提案する．次に，タスクの総移動経路長の削減を目指して，集荷経路長を削減するために，各エージェントに割り当てられるタスク間の連続性を考慮する割り当て手法を提案する．

3.1　従来手法の課題

TP アルゴリズムでは，エージェントが経路の終点に滞在する規則を用いる．滞在する位置を将来通過するエージェントの経路の予約が既にある場合，その経路を考慮しないため，他のタスクの経路を阻害する状況が起きると考えられる．

また，複数のエージェントが同時刻に token へのアクセスを要求するとき，同じタスクを先にアクセスしたエージェントに割り当てると，後からアクセスしたエージェントに割り当てるよりも，そのタスクの集荷経路長が長くなる場合があると考えられる．TP を拡張した TPTS においても，エージェントが移動し始めてから，他のエージェントへのタスクの受け渡しを考えると，冗長な移動となる場合や，解がない場合がある．

3.2　提案手法

先ず，経路の終点で滞在するための，既に予約されている経路も考慮する，滞在の予約手法を提案する．

次に，タスクの集荷経路長を削減するために，エージェントの移動の予測時間に基づいて，タスク間の連続性を考慮するタスク割り当てを提案する．

提案手法の基本的な流れは TP と同じである．token にアクセスした a_i は割り当て候補の TSK' から現在位置に集荷位置がもっとも近いタスクを選択する．ここで，提案手法では，現在位置から集荷位置まで移動する時間 t_i の推定値を A*アルゴリズムにより得られる経路長に対応する値とする．次に，タスクが割り当てられている他のエージェント a_k が受け持っているタスク τ_k を完了させるまでの時間 t_k を求める．さらに，τ_k を完了した位置から τ_j の集荷位置まで移動する時間 t'_k の推定値を A*より得られる経路長に対応する値とする．$t_i > t_k + t'_k$ のとき，タスク τ_j は他のエージェントが集荷する方が時間がかからないと期待して a_i に割り当てず，T' に含まれる別のタスクの割り当て

を試みる．タスクが割り当てられた場合，現在位置から集荷位置を経由して配達位置へ向かう経路を探索する．ここで提案手法では，経路の終点で滞在できるか確認する．滞在できない場合，タスク τ_j の割り当てを解除し，TSK' に含まれる別のタスクの割り当てを試みる．滞在の予約ができた場合のみ，token に経路を予約する．これらの処理を a_i にタスクが割り当てられるか，TSK' が空になるまで繰り返す．TSK' が空になった場合，その場に滞在するか，退避する．その場に滞在する処理は TP と同様である．退避する処理は，TP 同様，最も近い non-task endpoint への退避を試みる．ここで提案手法では，選択した退避位置に滞在する予約ができた場合のみ退避する．予約ができない場合は，他の退避位置の選択を試みる．これらの処理を a_i に退避位置が割り当てられるまで繰り返す．

3.2.1 経路の終点における滞在の予約

Well-formed な MAPD 問題では，endpoint はエージェントが滞在できる位置と定義されているが，単に通り過ぎることもできる．これにより TP では，タスクの経路の割り当てを行うエージェントは，他のエージェントがそれぞれの経路の終点で滞在する規則にもとづいて，経路の割り当てを行う．しかし，滞在の予約は既にある経路を考慮していない．そこで本研究では，エージェントが現在位置から集荷位置を経由して配達位置まで向かう衝突のない経路を探索したのち，経路の終点で滞在する予約ができる場合のみ，そのタスクをエージェントに割り当てる．予約ができない場合には，そのタスクを割り当てず，別のタスクの割り当てを試みる．退避の際も同様に，退避位置までの衝突のない経路を計画したのち，退避位置で留まり続ける予約ができる場合のみ，その退避位置へ退避する．予約ができない場合には，他の退避位置への退避を試みる．Well-formed な問題の定義より，non-task endpoint の数はエージェントの数より多いため，退避位置が割り当てられないことはない．これらの予約のために，時刻ごとのエージェントの頂点の占有情報を保存する，予約表を用いる．この予約表を利用して，CA*に基づき経路探索する．

図 8 のように a_1 の経路が既に予約表に保存されており，a_2 に経路を割り当てる状況を考える．(a) の場合，a_2 の経路の終点は $(t+2,\, v_{13})$ であり，すべての予約についての最大時刻である $t+4$ まで v_{13} で滞在する予約をする．このとき，滞在の予約は他のエージェントの経路を妨げていないため，このタスクの経路を割り当てることができる．(b) の場合，a_2 の経路の終点は $(t+2,\, v_5)$ である．v_5 で滞在の予約を試みるが，$t+4$ で a_1 の経路を妨げてしまう．この場合は，a_2 へこのタスクを割り当てない．次に，図 9 のように a_1 と a_2 の経路と滞在の予約が $t+4$ までされているとき，a_3 が予約表の大きさ $t+4$ を超える経路を計算する状況を考える．この場合，a_3 が経路を探索する際，$t+5$ で

エージェント＼時刻	t	t+1	t+2	t+3	t+4
a_1	v1	v2	v3	v4	v5 (終点)
a_2	v11	v12	v13 (終点)	v13 (滞在)	v13 (滞在)

(a) 滞在の予約成功

エージェント＼時刻	t	t+1	t+2	t+3	t+4
a_1	v1	v2	v3	v4	v5 (終点)
a_2	v3	v4	v5 (終点)	v5 (滞在)	v5 (滞在×)

(b) 滞在の予約失敗

図 8　予約表による滞在の予約の成功例と失敗例

展開済み　展開

エージェント＼時刻	t	t+1	t+2	t+3	t+4	t+5
a_1	v1	v2	v3	v4	v5 (終点)	v5 (滞在)
a_2	v11	v12	v13 (終点)	v13 (滞在)	v13 (滞在)	v13 (滞在)
a_3	v21	v22	v23	v24	v25	v26

図 9　予約表の大きさを超えたときの滞在の予約

近傍頂点を展開する前に，その時刻の予約表を a_1，a_2 の両方の経路の終点，v_5 と v_{13} で予約することで，a_3 はその時刻に a_1，a_2 が滞在する頂点を近傍頂点で展開しない．これにより，衝突を回避できる．

3.2.2 集荷経路長の削減

集荷経路の冗長な移動を削減するために，他のエージェントのタスクの完了時刻と移動の予測時間をタスクの割り当ての際に考慮する．

TP 同様，エージェント a_i の現在位置から最も集荷位置が近いタスク τ_j の割り当てを試みる．このとき，a_i の現在位置から集荷位置まで移動する時間 t_i の推定値を A*アルゴリズムにより得られる経路長に対応する値とする．次にタスクが割り当てられている他のエージェント a_k を考慮する．先ず，a_k が割り当てられているタスク τ_k を完了させるまでの，現時刻からの時間を t_k とする．次に，τ_k の配達位置から τ_j の集荷位置まで移動する時間 t'_k の推定値を A*により得られる経路長に対応する値とする．$t_i > t_k + t'_k$ であれば，a_k が τ_j をより短い時間で集荷することを期待し，a_i には τ_j を割り当てず，TSK' に含まれる他のタスクの割り当てを試みる．

この割り当てにより，TP の課題である同時刻に複数のエージェントが token を要求する場合でも，タスクの集荷位置により近いエージェントを割り当てることができる．

4　評価と考察

本章では，実験により提案手法の有効性を評価する．TP にあらかじめ滞在の予約を行う手法とタスク間の連続性を考慮したタスク割り当て手法を加えた解法を，continuity-TP (coTP) と呼ぶ．

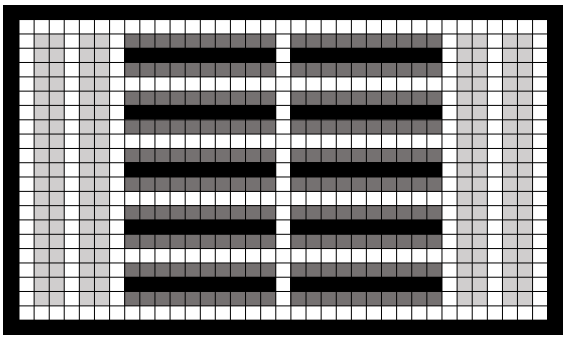

図 10　倉庫を模した問題 (黒：棚 (外枠は除く)，濃灰色：task endpoint，薄灰色：non-task endpoint)

4.1　実験の問題設定

MAPD 問題は物流倉庫内の搬送ロボットへ応用を目的とした問題である．そのため，問題の中央に棚が並ぶような倉庫を模した問題で TP と coTP を評価した．図 10 は既存研究 [4] と同じ，四近傍のグリッドの問題である．図 10 において，外枠の黒い部分を除いた位置が棚である．棚に隣接する濃灰色の部分が task endpoint，薄灰色の部分が non-task endpoint である．この環境に 500 のタスクを生成し，タスクの集荷，配達位置は task endpoint からランダムに選択した．タスクの生成頻度である task frequency (TF) は 1，2，5，10 とした．エージェントの数 (NoA) は 20，30，40，50 とし，エージェントの初期位置は non-task endpoint からランダムに選択した．評価尺度は，タスクが生成されてから完了するまでの時間ステップ数 service time (ST)，全タスクが完了するまでの時間ステップ数 makespan (MS)，単位時間ステップ数あたりの実際の計算時間 computation time (CT)，エージェントあたりの総移動経路長 (total move)，集荷経路長 (pickup)，退避経路長 (evacuation) である．Service time は計 500 のタスクの平均，総移動経路長，集荷経路長，退避経路長はエージェントあたりの平均である．これらについてそれぞれのタスクの生成頻度とエージェント数で 10 回ずつ試行した平均を示す．

4.2　実験結果

Task frequency が 1 と 2 のときの結果と，5 と 10 のときの結果をそれぞれ表 1， 2 に示す．

Service time

全てのタスク頻度，エージェント数において，TP よりも coTP の方が小さい値となった．タスクの頻度が大きい場合，タスクが生成されてから割り当てられるまでの待ち時間が大きくなり，service time の値は大きくなる傾向であった．最大のパラメータの値であるタスク頻度 10，エージェント数 50 のとき約 7%削減された．タスク頻度が小さい場合はエージェントが各タスクに迅速に対応できるため，service time の値は小さくなる傾向であった．タスク頻度 1，エージェント数 50 のとき，TP に対して約 45%削減された．

Makespan

表 1　実験結果 (タスク頻度 1，2)

TF	NoA	手法	ST	MS	CT $[ms]$	total move	pick up	evacuation
1	20	TP	76.5	649.4	30.5	575.3	176.6	0.97
		coTP	67.8	631.1	44.6	555.6	153.9	2.61
		ratio	0.886	0.972	1.46	0.966	0.871	2.70
	30	TP	44.4	586.7	57.2	488.4	222.9	2.99
		coTP	33.6	559.2	118.8	424.6	143.6	20.41
		ratio	0.757	0.953	2.08	0.868	0.644	6.83
	40	TP	48.8	601.4	152.3	458.9	256.1	18.25
		coTP	30.0	551.5	200.5	318.4	97.8	24.51
		ratio	0.615	0.917	1.32	0.694	0.382	1.34
	50	TP	51.7	597.4	289.7	411.7	136.7	31.59
		coTP	28.8	552.1	276.1	251.0	73.2	22.44
		ratio	0.557	0.924	0.953	0.610	0.535	0.71
2	20	TP	167.3	602.0	28.3	532.6	133.8	0.92
		coTP	156.9	579.7	47.6	511.9	113.5	1.27
		ratio	0.938	0.962	1.68	0.961	0.848	1.36
	30	TP	101.1	474.8	57.4	384.5	112.9	1.66
		coTP	89.8	436.5	91.4	360.8	87.3	2.92
		ratio	0.889	0.919	1.59	0.938	0.773	1.76
	40	TP	71.3	416.2	102.0	313.7	106.1	2.76
		coTP	60.3	372.1	161.8	288.9	77.6	5.62
		ratio	0.846	0.894	1.59	0.921	0.731	2.04
	50	TP	61.3	401.2	125.8	279.9	111.7	4.32
		coTP	48.5	352.8	288.0	248.5	73.8	11.18
		ratio	0.791	0.879	2.29	0.888	0.661	2.59

表 2　実験結果 (タスク頻度 5，10)

TF	NoA	手法	ST	MS	CT $[ms]$	total move	pick up	evacuation
5	20	TP	225.7	590.0	25.7	522.8	124.6	1.13
		coTP	220.5	576.5	61.0	507.3	109.3	1.50
		ratio	0.977	0.977	2.37	0.970	0.877	1.33
	30	TP	157.7	451.4	51.5	371.3	98.3	2.03
		coTP	149.1	428.1	111.2	353.2	78.1	2.60
		ratio	0.945	0.948	2.16	0.951	0.795	1.28
	40	TP	123.8	395.9	78.4	297.2	86.6	2.47
		coTP	115.3	361.0	179.6	276.8	63.6	4.25
		ratio	0.931	0.912	2.29	0.931	0.734	1.73
	50	TP	105.9	371.1	111.5	254.3	81.6	4.57
		coTP	94.1	331.9	284.9	229.9	55.3	5.44
		ratio	0.889	0.894	2.56	0.904	0.678	1.19
10	20	TP	249.1	595.7	27.9	523.7	127.9	1.03
		coTP	243.9	583.0	75.6	509.7	111.3	2.33
		ratio	0.979	0.979	2.71	0.973	0.870	2.27
	30	TP	178.9	442.8	55.8	369.7	97.1	1.58
		coTP	172.5	432.3	138.8	352.8	79.0	2.57
		ratio	0.964	0.976	2.49	0.954	0.814	1.63
	40	TP	144.6	393.5	95.6	296.9	87.9	2.95
		coTP	138.8	365.6	251.2	275.9	62.8	4.14
		ratio	0.960	0.929	2.63	0.929	0.714	1.40
	50	TP	126.3	368.7	126.6	252.0	80.0	4.75
		coTP	118.0	318.1	284.7	230.4	54.4	5.56
		ratio	0.934	0.863	2.25	0.914	0.680	1.17

全てのタスク頻度，エージェント数において，TP よりも coTP の方が小さい値となった．タスク頻度が低

い場合，全タスクがシステムに追加されるのに時間がかかるため，makespan の値は大きく，また，TP に対して 10%未満の削減であった．しかし，タスク頻度が大きい場合は makespan の値は小さくなる傾向があり，TP に対しての削減も大きくなり，タスク頻度 10，エージェント数 50 のとき，最大で約 14%削減された．

Computation time

ほとんどのタスク頻度，エージェント数において TP よりも coTP の方が大きな値となった．coTP は集荷経路長の予測等，TP と比べて処理が多いため，1 ステップあたり約 1.3 から 2.7 倍の計算時間がかかった．計算時間と，エージェント-共有メモリ間の通信時間が，搬送ロボットの移動や荷物の積み下ろしにかかる時間と比較して，十分小さく，計算時間の増加量が，移動時間の削減量に吸収されれば，適用できる可能性がある．TP による単位ステップあたりの計算時間を CT_{TP}，coTP による単位ステップあたりの計算時間を CT_{coTP}，makespan を MS_{coTP} とし，TP によるエージェントの総移動距離を D_{TP}，coTP によるエージェントの総移動経路を D_{coTP} とする．実際の搬送ロボットの移動速度を V_r とすると，coTP が適用できる可能性がある搬送ロボットの移動速度の範囲は，

$$V_r < \frac{D_{TP} - D_{coTP}}{(CT_{coTP} - CT_{TP})MS_{coTP}}$$

である．グラフの頂点間の距離を 1m と仮定すると，task frequency=1，エージェント数 40 のとき，
$CT_{TP} = 152, CT_{coTP} = 200, D_{TP} = 458, D_{coTP} = 318, MS_{coTP} = 551$ より $V_r < 5.28\ [m/sec]$
また，task frequency=10，エージェント数 50 のとき，
$CT_{TP} = 126, CT_{coTP} = 284, D_{TP} = 252, D_{coTP} = 230, MS_{coTP} = 318$ より $V_r < 0.44\ [m/sec]$
となった．Amazon Robotics Kiva の移動速度は約 1.7 $[m/s]$ である[1]．搬送ロボットの移動速度をこの速度と仮定すると，パラメータによっては適用できる可能性がある．

総移動経路長

全てのタスク頻度，エージェント数において，TP よりも coTP の方が小さい値となった．どのタスク頻度でもエージェントの数が増えるにつれて，エージェントに割り当てられるタスク数の平均が減少するため，総移動経路長は削減される傾向であった．TP に対しての削減率はタスク頻度が小さいほど大きい傾向であった．システム中に残っているタスクの数に対してエージェントの数が十分であれば，各タスクに集荷経路が短いエージェントを割り当てることができるため，総移動経路長が削減された．タスク頻度 1，エージェント数 50 のとき最大で約 39%削減された．

集荷経路長

全てのタスク頻度，エージェント数において，TP よりも coTP の方が小さい値となった．どのタスク頻度においても，エージェントの数が増えるにつれて，集荷経路長は削減される傾向であった．総移動経路長と同様に，システム中のタスク数に対してエージェントの数が十分であれば，タスクの集荷位置により近いエージェントを割り当てることができた．それにより，タスク頻度 1，エージェント数 50 のときに TP に対して最大で約 47%削減された．また，図 11 はタスク生成頻度 1，エージェント数 50 のときのタスクあたりの集荷経路長の一例を示している．coTP は経路の密集状況により，集荷に時間がかかる場合もあるが，全体として集荷経路長は削減された．TP では集荷経路長が 15 以上のタスクが非常に多かったことに対し，coTP では 15 未満のタスクがほとんどであった．

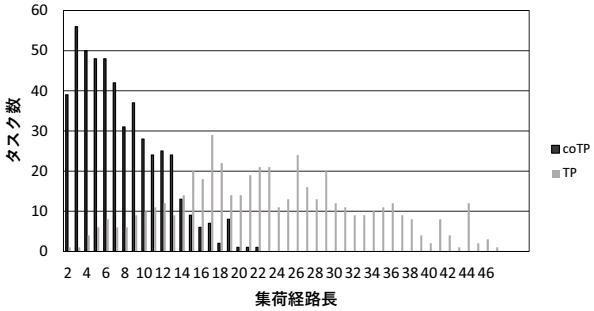

図 11　1 タスクあたりの集荷経路長の一例 (タスク頻度 1，エージェント数 50)

退避経路長

ほとんどのタスク頻度，エージェント数において，TP よりも coTP の方が大きい値となった．これは，タスクの割り当ての際に他のエージェントを考慮することで，他のエージェントにタスクを任せ，退避する回数が増加した影響である．タスク頻度 2，エージェント数 50 のとき，TP の退避回数が約 24 回であるのに対して提案手法の退避回数は約 65 回であった．しかし，タスク頻度 10，エージェント数 50 のとき，TP の退避回数が約 25 回であるのに対して提案手法の退避回数は約 30 回であった．つまり，タスク頻度が増えるにつれて，多くのエージェントがタスクを割り当てられる状況が続き退避する回数が減ったことにより，TP に対しての増加が抑えられた．

実験結果により，システムに残っているタスクの数に対し，エージェントが十分であれば，提案手法の効果が大きいことが分かった．また，本実験で用いた解法の実装は実験的なものであり，改善の余地がある．

5　まとめ

本研究では，時系列的に発生する集配タスクに対する，複数エージェントの経路計画問題である Multi-Agent Pickup and Delivery (MAPD) 問題の解法 TP の改善手法を提案した．TP アルゴリズムは，エージェントが経路の終点で滞在し続ける規則に基づき経路探索する．これは，滞在する位置を将来通過する既存の経路を考慮しないため，他のタスクの経路を阻害する状況が起きると考えられる．さらに，タスクを割り当てることができるエージェントを貪欲的な順番で列挙し，最も

[1] https://www.itmedia.co.jp/news/articles/1612/06/news107.html (公開日 2016.12.6 閲覧日 2020.6.19)

集荷位置が近いタスクを割り当てるため，各タスクの集荷位置に最も近いエージェントが割り当てられるとは限らない．このように，集荷経路に冗長さがあることが TP の問題点として挙げられる．そこで本研究では，経路の終点で滞在する予約が他のエージェントの経路の妨げにならないタスクまたは退避位置のみを割り当てる手法を提案した．また，冗長な移動の削減のために，タスクの割り当ての際にエージェントの移動の予測時間に基づいて，タスクに対して適切なエージェントを選択する手法を提案した．実験結果により，提案手法は TP よりも service time，makespan，総移動経路長，集荷経路長を削減することが示された．また，システム内のタスクの数に対してエージェントの数が十分であれば，各タスクに対して集荷経路が短いエージェントを割り当てることができるため，提案手法の効果が大きいことが分かった．単位時間ステップ当たりの，計算時間のオーバヘッドは，従来手法よりも提案手法の方が大きいが，実際の搬送ロボットの移動や積み降ろしにかかる時間と，移動時間の削減量よりも，計算時間のオーバヘッドが十分に小さければ，適用の可能性があると考えられる．実験結果を用いて，搬送ロボットの移動速度を推定すると，パラメータによっては適用可能な値であることが分かった．今後の課題として，より適切な endpoint の配置，退避経路長を削減しつつ次の集荷にスムーズに繋がる退避位置の選択が挙げられる．また，TP アルゴリズムにおける経路計画は，準最適解法である CA*を用いているが，問題のスケールアップのために，CBS に提案手法を適用する模索も挙げられる．

参考文献

[1] P.R. Wurman, R. D'Andrea, and M. Mounts. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *AI Magazine*, pages 29(1):9–20, 2008.

[2] Robert Morris, Corina Pasareanu, Kasper Luckow, Waqar Malik, Hang Ma, T. K. Satish Kumar, and Sven Koenig. Planning, scheduling and monitoring for airport surface operations. In *AAAI-16 Workshop on Planning for Hybrid Systems*, pages 608–614, 2016.

[3] D. Silver. Cooperative pathfinding. In *Arltificial Intelligence and Interactive Digital Entertainment*, pages 117–122, 2005.

[4] Hang Ma, Jiaoyang Li, T.K.Aatish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 837–845, 2017.

[5] E. Boyarski, A. Felner, R. Stern, G. Sharon, and D. Tolpin. Improved conflict-based search algorithm for multi-agent pathfinding. In *International Joint Conference on Arttificial Intelligence*, pages 740–746, 2015.

[6] Hart P.E., Nilsson N.J., and Raphael B. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems Science and Cybernetics*, pages 100–107, 1968.

[7] M. Cap, J. Vokrinek, and A. Kleiner. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *International Conference on Automated Planning and Schedulin*, pages 324–332, 2015.

[8] Wolfgang Hönig, T. K. Satish Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian, and Sven Koenig. Multi-agent path finding with kinematic constraints. In *International Conference on Automated Planning and Scheduling*, pages 477–485, 2016.

[9] Hang Ma, T. K. Satish Kumar, and Sven Koenig. Multi-agent path finding with delay probabilities. In *AAAI Conference on Artificial Intelligence*, pages 3605–3612, 2017.