Vol.2021-ICS-204 No.6
2021/9/15

マルチエージェント搬送のための環境制約を緩めたPIBT手法の拡張

藤谷 雪北[1]　山内 智貴[1]　宮下 裕貴[1]　菅原 俊治[1]

概要：本研究では，MAPD 問題の制御手法である Priority Inheritance with BackTracking (PIBT) に暫時的な優先度を導入した拡張 PIBT を提案し，PIBT の能力を下げることなく適用環境の制約を緩めることで適用範囲を拡大すると共に，実験的にその効果を示す．PIBT 手法は毎ステップ優先度を計算し，その優先度の高いエージェントから順番に，次のステップでの移動先を確定させるアルゴリズムである．このアルゴリズムでは，袋小路のような形状を含むマップでは行き詰まりが発生するため，実験環境に対しそれを除外する制約を設けている．そこで本研究では，エージェントに通常の優先度に加えて暫時的な優先度を持たせる拡張を施し，先行研究における環境制約を緩めても搬送を継続できることを実験的に示す．

キーワード：マルチエージェント，配送問題，優先度計算

1. はじめに

近年，人工知能技術の発達により，自律的に状況を判断して行動をする機械やシステムが身近なものとなっている．例えば清掃ロボットは，事前情報なしに部屋のレイアウトを学習し，自動的に清掃をする．また災害時に人が立ち入れないような場所で，ロボットが人や物を探索するという例も存在する．このように自身の周りの環境を観測し，そこから自分の行動を自律的に決めるロボットやシステムをエージェントとしてモデル化し，様々な場面で活用できるように研究が進められている．近年の IoT や通信技術の発達を考慮すれば，将来的にはエージェントが広い場面で求められると考えられる．

このようなエージェントが複雑で広大な環境での利用が期待されると，複数のエージェントが協力して共通の目標を達成するマルチエージェントシステムによる共同作業が求められる．このようなマルチエージェントシステムの応用例として，交通流の制御 [1] や自動倉庫 [2]，飛行機の運行制御 [3] などがある．しかし各エージェントの個別の判断による最適行動をとり同一環境で行動すると，他のエージェントとの衝突やリソースの競合が発生する可能性がある．競合はエージェント数の増加に伴い起きる頻度も高くなるため，複数エージェントによる効率化を実現するには，これらの競合を減らす制御が重要である．

[1] 早稲田大学基幹理工学研究科，東京都
Fundamental Science and Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan

本研究ではマルチエージェントシステムの応用例の 1 つであるマルチエージェント配送 (Multi-Agent Pickup and Delivery, MAPD) 問題 [4] に着目する．MAPD 問題とは，複数のエージェントが特定の環境内で複数の運搬タスクを衝突することなく平行して継続的に実行する問題である．各タスクはスタートとゴール地点で表現でき，スタート地点にある物資をゴール地点まで運ぶこととする．複数のエージェントが互いに衝突を避けながらスタート地点からゴール地点までの経路を探索するマルチエージェント経路探索 (Multi-Agent Path Finding, MAPF) 問題 [5] の繰り返しと考えられる．

MAPD 問題を解決するアルゴリズムはいくつか提案されているが [4], [6], [7]，ここでは priority inheritance with backtracking (PIBT) [7] に着目する．このアルゴリズムでは，各エージェントが毎ステップ優先度を計算し，近隣と通信をしながら優先度の高いエージェントから順番に次に移動する位置を決める．その際，優先度の高いエージェントは次に移動したい位置にいる，自分より優先度の低いエージェントを押し出すことが可能であるが，他方，次の位置に優先度の高いエージェントがいれば，移動をしないか優先順位を下げた次位置に移動する．各エージェントが自律的に移動する次位置を決めるため，エージェント数増加に対する拡張性のある制御方法である．しかしこのアルゴリズムでは，袋小路のような形状を含むマップにおいては行き詰まり（デッドロック）が発生するため，実験環境に対して 2 重連結グラフでなければならないという，環境

制約を設けている．

本研究では，環境制約を緩めた拡張 PIBT を提案する．ここでは PIBT の性質を損なうことなく優先度を変更するとともに，通常の優先度に加え暫時的な優先度を導入し，行き詰まりが発生した際は暫時的な優先度を用いて PIBT を適用することにより，PIBT の環境制約を緩和しても行き詰まりが発生しない制御が可能であることを，実験的に示す．また，その効率などの評価を行う．

2. 関連研究

マルチエージェント配送問題において，エージェントを制御する方法は，特定のエージェントが全体を把握し全ての経路を決定する集中制御と，各エージェントが周囲の状況から自律的に次の行動を決定する分散制御に大きく分けられる．集中制御の研究としては [6] や [8] がある．たとえば [6] では，high-level と low-level の 2 段階で最短経路の探索を行い，タスクの割り当て順に各エージェントがいつどこを通るべきかを考慮し，衝突のない計画する制御方法を提案している．また [8] では，エージェントをゴールに近づける push 操作，2 体のエージェントの位置を交換する swap 操作の 2 つの操作により，エージェントの動きを制御している．

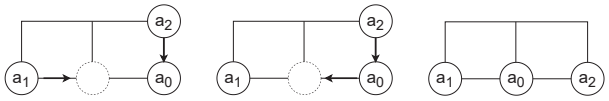
他方，マルチエージェント配送問題に分散制御的なアプローチをとった研究としてとして [4], [7], [9], [10] などがある．たとえば [4] や [9] では，アマゾンの倉庫のような予めロボット運送用に設計された倉庫のような環境で token と呼ばれる全エージェント共通メモリに各エージェントが自身の経路を書き込み，これをエージェント間で参照し合うことにより，分散的にエージェント群の制御をしている．[10] では，現実のロボットにおける大きさや速さといった物理的制約を考慮した環境にて，エージェント群を分散制御する方法を提案している．

3. 準備

3.1 問題設定

エージェントの集合を $A = \{a_1, a_2, \ldots, a_n\}$ と表す．環境は 2 次元ユークリッド空間に埋め込み可能な無向グラフ $G = (V, E)$ とする．エージェントはノード $v \in V$ 上に停止でき，またノード $v, u \in V$ を繋ぐエッジ $(v, u) \in E$ に沿ってノード間を移動できる．PIBT [7] では，グラフ G は自己ループや多重辺を含まず，2 重連結グラフである必要がある．本研究では，この制約を緩める拡張をする．

本研究のタスク τ は，タスク毎に異なるスタート $\tau_{start} \in V$ とゴール $\tau_{goal} \in V$ の 2 つの情報から構成され τ を割り当てられたエージェント $a \in A$ は τ_{start} にある物資を τ_{goal} に運ぶ．ただし簡単のため，エージェント a がタスクのスタート τ_{start} を経由すると，物資を受け取りゴール τ_{goal} に到着すると，タスク τ の実行が完了するものとする．本研究では，タスクのスタート τ_{start} とゴール τ_{goal} はタスクが発生しうる場所からランダムに選択され，各エージェントはそれらのタスクを順に実行する．



(a) 行き詰まり　(b) 優先度継承　(c) 1step 後

図 1: PIBT における優先度継承

$t \in N$ におけるエージェント a_i の位置を $v_i(t)$ と表す．エージェント a_i は以下の式に示すように，自身の現在地に留まるか隣接するノードの 1 つに，移動できる．

$$v_i(t+1) \in \{v \in V | (v_i(t), v) \in E\} \cup \{v_i(t)\} \tag{1}$$

複数のエージェントは，以下の制約式のように同時に同じノードに存在できず，またすれ違うこともできない．

$$v_i(t) \neq v_j(t) \tag{2}$$

$$v_i(t) \neq v_j(t+1) \vee v_i(t+1) \neq v_j(t) \tag{3}$$

なお，以下の式に表すローテーションは可能とする．

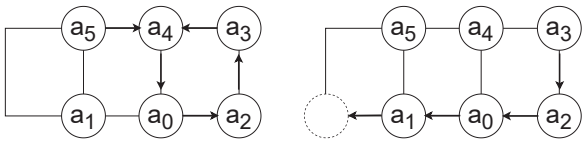
$$v_i(t+1) = v_j(t) \wedge v_j(t+1) = v_k(t) \wedge \ldots \wedge v_l(t+1) = v_i(t) \tag{4}$$

3.2 Priority Inheritance with Backtracking (PIBT)

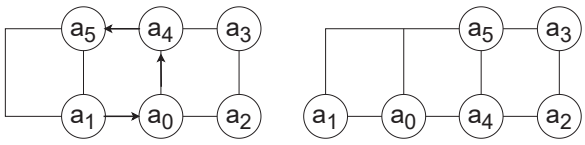
既存手法である PIBT [7] とは，マルチエージェント配送問題におけるエージェントの分散制御アルゴリズムである．これは，毎ステップ各エージェントが優先度を計算し，その優先度の高いエージェントから優先的に次の行動（つまり次の移動位置）を確定するアルゴリズムである．PIBT は優先度継承 (Priority Inheritance) を再帰的に行うことにより，エージェント同士のデッドロックを防止している．優先度継承とは，優先度の高いエージェント a_j が次ステップで移動したいマスに優先度の低いエージェント a_i がいるとき，a_i に a_j の優先度を継承するアルゴリズムである．

図 1 に継承の例を示す．ここでは，エージェント a_2，a_1，a_0 の順番に優先度が高いものとする．図 1a では，最も優先度の高い a_2 が下に進もうとするが，a_0 がそのマスに存在する．このときエージェント a_0 はその位置を空けるために左へ移動する必要があるが，そこには a_0 よりも優先度の高い a_1 が次に進むことになっており，行き詰まりの状況となる．図 1b と図 1c はこの状況に対して優先度継承を適用した場合である．優先度継承により a_0 は a_2 の優先度を継承し，a_0 は a_1 よりも優先度が暫時的に高くなるため，図 1c に示すように，左へ移動できる．

図 2 に PIBT におけるバックトラックの例を示す．ここでもエージェントの添字の数が大きいほど優先度は高いも



(a) 優先度継承　(b) BackTrack+優先度継承



(c) BackTrack　(d) 1step 後

図 2: PIBT におけるバックトラック

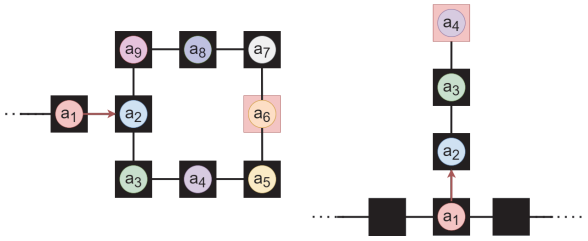
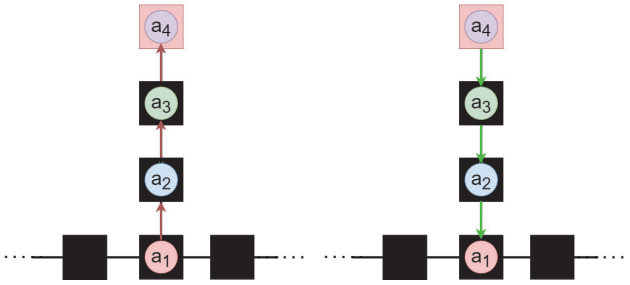


図 3: PIBT が行き詰まる例

のとする．図 2a は，優先度継承を用いてエージェント a_5 の優先度を a_4，a_0，a_2，a_3 の順番で継承するが，ここで行き詰まりが発生する．PIBT のアルゴリズムでは，バックトラックと再帰的な優先度継承によってこの状況を回避している．図 2b では，バックトラック (以降，BT) によって優先度継承の逆向きに，行き詰まりの発生 (*False*) を伝える．これを受けたエージェント a_0 は別のルートがあるため，エージェント a_1 の方向へ優先度を継承させる．図 2c は，優先度継承の結果行き詰まりが発生しない移動ができること (*True*) を，継承の逆向きに伝える．その結果，各エージェントは図 2d に示す方向へ動く．

なおこのアルゴリズムは，環境が 2 重連結グラフ，つまり任意の 2 ノードの組は，その他の任意の頂点が一つ取り除かれてもそれらを結ぶパスがあるなら，必ず全てのエージェントが有限の時間で目的地へ辿り着けることが保証されている．これは，任意の 2 点間に，共有ノードを持たない複数のパスが存在することでもある．また，エージェント数は環境内のノード数以下であれば良い．

しかしこの環境制約を緩めると，PIBT では行き詰まりが発生する．図 3 にその例を示す．これらの例で a_1 が最も優先度は高いものとするが，a_1 は物理的に他のエージェントを押し出せない．特に [7] では，優先度をスタートあるいはゴールを離れてからの時間 (ステップ数) を用いるため，優先度の順序は変わらず，行き詰まりを解決できない．



(a) 優先度の継承　(b) *False* の伝達

図 4: 行き止まり状態における優先度継承

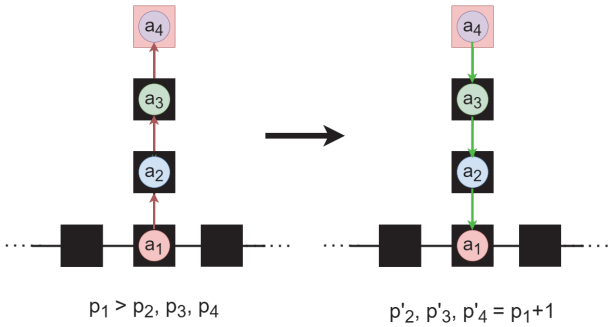


図 5: 暫時的な優先度

4. 提案手法

4.1 暫時的優先度の導入

PIBT では，優先度の高いエージェントが優先度の低いエージェントのいる方向へ移動しようとする際，図 4a に示すように優先度を可能な限り継承する．しかしエージェント a_4 は優先度継承をされるものの，これ以上進むことも伝達も不可能であるため，a_4 に優先度を継承した a_3 に *False* を返し，図 4b に示すように a_1 まで戻り，この状態を繰り返すデッドロック状態となりうる．

そこで，エージェント $a_i \in A$ の PIBT の通常の優先度 p_i に暫時的な優先度 p'_i を持たせる．具体的には，エージェント a_i は通常，自分の優先度 p_i を用いるが，行き詰まりが発生し *False* を返したとき a_i の優先度を暫時的に $p'_i = p_j + 1$ とする．ここで a_j は優先度を継承したエージェントとする．たとえば，図 5 では，各段階で *False* を返した各エージェントの優先度は，$p_1 + 1$ に上がり，このため行き詰まりの状態が繰り返されることなく，*False* を返したエージェントは自分自身に優先度継承をしたエージェントよりも優先度が高くなり，行き詰まりの状況が継続することがなくなる．また暫時的優先度を持ったエージェントは，交差点 (隣接ノードが 3 つ以上存在するノード) に到着すると通常の優先度 p_i に戻るものとする．

本提案手法は，図 6 に示すような 2 重連結グラフに交差点の無い道を 1 本以上繋げた環境で継続的な実行が可能である．エージェント数は，中心となる二重連結領域のノード数を超えないとする．なお，タスクは絶え間なく発生し，

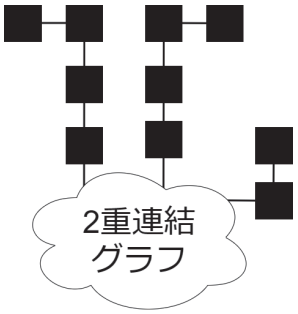


図 6: 本提案手法が想定する環境（複数の道を繋げた 2 重連結グラフ）

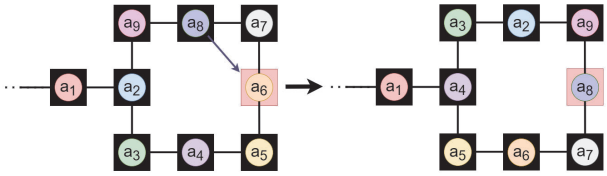


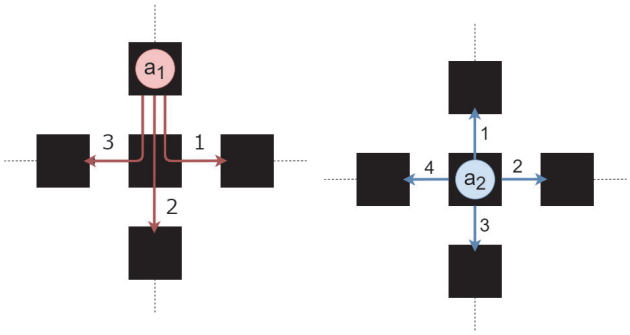
図 7: 予め目的地に近いエージェントが目的地へ移動する

エージェントにはかならずタスクが与えられるものとする（あるいは，各エージェント毎に充電などのための専用の基地を用意し，タスクが無いときには，自分の基地に戻るとする）．

尚，PIBT では優先度として目的地を更新して（つまり新しい目的地に向かって進み始めたとき）からのステップ数（経過時間）を使用しているが，本研究では現在地から目的地までの最短経路長の逆数または -1 倍とする．つまり，目的地に近いほど優先度が高くなる．これは，不必要な暫時的な優先度を発生させないためである．例えば図 3 の左の例で，a_1 に加え a_8 が，現在 a_6 のいるマスを目的地としているとする．目的地を更新してからのステップ数を優先度とすると，a_1 の優先度が a_8 の優先度よりも高くなることがある．このとき，a_1 は a_2 へ，a_2 は a_3 へと継続的に優先度継承を行うが，最終的に a_9 はその場にとどまることも移動も不可能なため *False* を返す．a_8, a_7 以降もこれを繰り返し，この結果 a_2 から a_9 のエージェントは暫時的な優先度を用いる．この暫時的な優先度は a_2 から a_9 まで同じ値（a_1 の優先度 $p_1 + 1$）となり，結果的に目的地に近い a_8 が他のエージェントと同じ優先度となる．このため，a_8 は目的地に到達するために最短の 2 ステップ以上のステップ数が必要となることがあり得る．一方，目的地に近いほど優先度が高くなる設定ではこの時目的地に近い a_8 は a_1 より優先度が高く，図 7 のように他のエージェントを押し回転をすることで，どのエージェントも *False* を発生させることなく，必ず 2 ステップで目的地へ到達可能である．

4.2 他エージェントの目的地を考慮した移動可能性の判定

PIBT では，あるエージェントのいるノードと隣接するノードが複数あるとき，次ステップの移動で好ましい順に隣接ノードへの移動可能性を判定し，移動可能なノードへの移動が確定する．ここで優先度を継承されたエージェントが移動可能判定をする順番について図 8 を例に述べる．図 8a は，エージェント a_1 が交差点に到着した後に移動する優先順位を示している．他方，図 8b は，エージェント a_2 が現在位置から移動する優先順位を示しており，まずはこの順位にしたがって移動可能を判定する．



(a) a_1 の時ステップでの移動優先方向　(b) a_2 の移動優先方向

図 8: エージェントの移動方向優先順位の例（決定前）

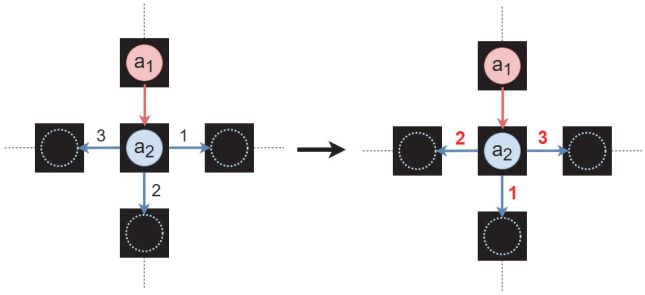
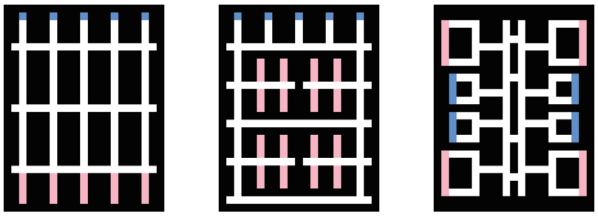


図 9: a_1 の移動方向を勘案した a_2 の移動可能の判定

ここで図 9 の左図のような状況において a_1 の優先度が a_2 の優先度よりも高い場合，a_1 は交差点に優先的に移動するために，交差点にいる a_2 に対して優先度を継承する．a_2 は優先度を継承されているため a_1 のいる上方向へは移動できず，また a_2 がその場に留まることも不可能である．a_2 が移動できるノードは図 8b の順位に基づき右，下，左となり，本来ならばこの順番に移動可能性を判定する．しかしここで工夫として，a_2 の移動可能判定をする際に，優先度の高い a_1 の最も移動したい方向である右方向を最後に判定するようにする (図 9 の右図)．

この工夫を導入しないと，優先度の低いエージェントが高いエージェントに押され続け効率が落ちることがあるが，この工夫によって優先度の低いエージェントが優先度の高いエージェントの移動したい方向を空けるように移動し，競合を早期に解消できる．



(a) 環境 1　(b) 環境 2　(c) 環境 3

図 10: 実験環境

5. 評価実験

5.1 実験 1

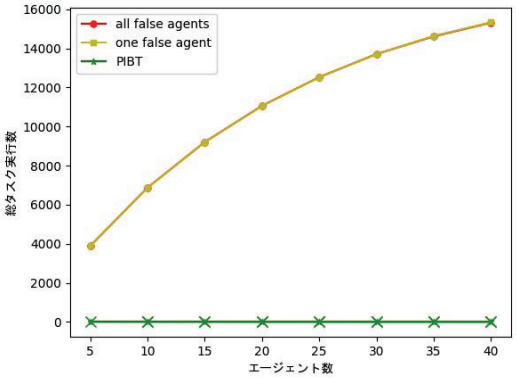
5.1.1 実験の目的と設定

本提案手法を評価するために図 10 に示す 3 つの実験環境で実験した．なお，環境 1 は，本提案手法が想定する環境であり，継続的なタスク実行が可能である．一方，環境 2, 3 は，本提案手法では継続的なタスク実行が必ずしも保証されないため停止する可能性もあるが，その頻度を検証するために実験を行った．また，いずれの環境も PIBT の条件を満たさないため，PIBT 手法の実行で停止する可能性がある．これらの図で，赤ノードはタスクのスタートとなり得る荷物の発生箇所であり，青ノードは運搬の目的地となる．タスクの発生は，本条件のもと，全てのエージェントが常にタスクを担当するような無限列とした．なお，4.2 節の優先度の変更は本実験の提案手法に導入していない．

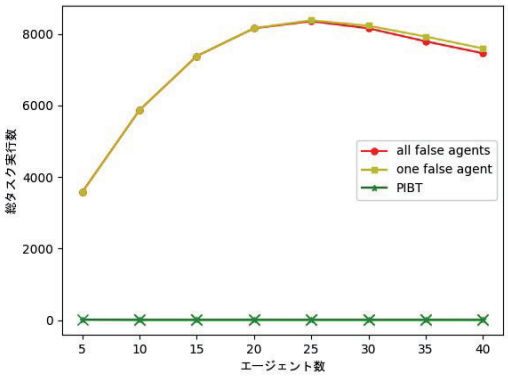
ここでは，これらの複数の環境において，比較のためにオリジナルの PIBT の他に，暫時的な優先度 p'_i を用いるエージェントの選択方法を変えて実験する．第 1 の選択方法は，行き詰まりの結果 *False* を返した全エージェントが暫時的優先度を用いる提案方法である．これをグラフでは all false agents と記す．たとえば，図 4b において，*False* を返した全エージェント a_2, a_3, a_4 の 3 体が暫時的な優先度 p'_i を用いる．第 2 の選択方法は，提案手法を緩めた方法であり，*False* を返したエージェントの内，最終的に他エージェントを押すことのできなかった (つまり最初に *False* を発した) エージェントのみが暫時的優先度を用いる方法である．たとえば図 4b で，*False* を返したエージェントのうち a_4 のみが暫時的な優先度を用いる．これを one false agent と記す．50000 ステップを 1 試行とし，30 試行の平均を実験結果とした．また，エージェント数を 5，10，15，20，25，30，35，40 と変化させた．

5.1.2 実験結果

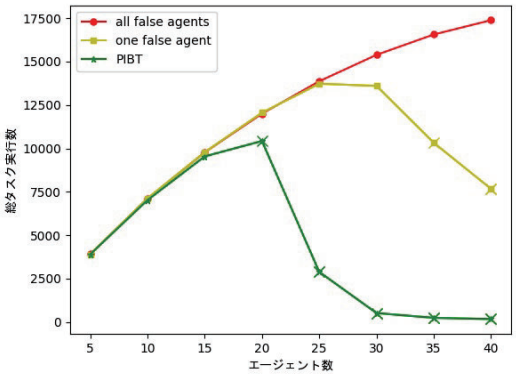
各環境ごとの実験結果を図 11 に示す． 本実験 30 テストのうち 1 テスト以上で途中で行き詰まり（デッドロック）が起きたものには，×がプロットされている．初めに，暫時的な優先度を用いた 2 手法と PIBT による性能差を確認する．環境 1 や環境 2 においては，PIBT によるタスク実行数はほぼ 0 であり，行き詰まりが発生し停止した（数タスクを実行できたが，その後停止する）．これは，タスクのスタートとゴールが，PIBT では対応できない行き止まりのノードにあり，そこに入り込んだエージェントが行き詰まり状態になるためである．一方，提案手法では環境 1 においてタスクの実行が行き詰まりが発生することなく継続的に進むことが確認できる．また環境 2 においては提案手法によるタスク実行でもデッドロックが発生しなかったことが確認できる．環境 3 においても，PIBT ではエージェ



(a) 環境 1 でのタスク実行数の平均値



(b) 環境 2 でのタスク実行数の平均値



(c) 環境 3 でのタスク実行数の平均値

図 11: 実験 1 の結果

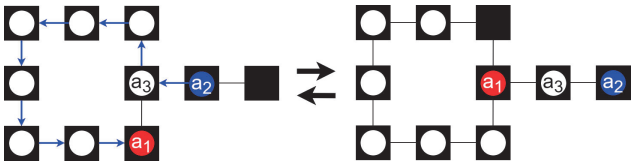


図 12: 環状の行き詰まりの状況

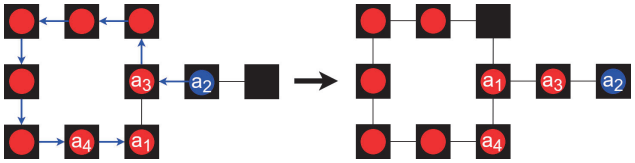


図 13: 継続的に押し返す状況

ント数がある程度増加するとタスク実行数は 0 になる．環境 3 には行き止まりのスタート，ゴールノードはないが，エージェント数の増加に伴い入口が一つの環状路があり，ここで行き詰まりとなる．提案手法ではエージェントが増加しても，タスクを実行できることが確認できるが，提案手法を緩めた “one false agent” では，エージェント数が多いときに，頻度は低いものの途中でデッドロックが発生することがあった．

次に暫時的な優先度を用いるエージェントの選択方法による結果の違いを確認する．環境 1 では，エージェント数が増加するにつれてタスク実行数も単調増加している．また，暫時的な優先度を用いるエージェントの選択方法による結果の違いは見られない．これは環境 1 の端のノードを除けば PIBT 実行可能な環境であること，格子状の路のため，等距離の迂回路が多く存在しエージェント数が増えて混雑しても別経路で目的地へ容易に到達可能なためだと考えられる．

環境 2 では，暫時的な優先度を用いるエージェントの選択数にかかわらず，エージェント数が 25 体の時にタスク実行数の最大となり，エージェント数が 25 体を超えるとタスク実行数は緩やかに減少する．これはエージェント増加による効率増加よりも，特に中心の「キ」の形の路において混雑による優先度の高いエージェントに道を譲る機会が増え，効率低下を招くからであり，環境 2 ではこの負の効果がエージェント数 25 付近で現れたためと考えられる．

環境 3 では，暫時的な優先度を用いるエージェントの選択方法による結果に明確な差がでる．まず，一体のみのエージェントが暫時的な優先度を用いる場合 (one false agent)，そのエージェントのみがそれ自身の目的地の方向へ優先的に移動可能となる．このエージェントは暫時的な優先度を持つ間は優先的な移動ができるが，暫時的な優先度が解除され通常の優先度に戻ると元の位置へ優先度の高いエージェントにより押し戻されることがある．この結果，継続的に優先度の高いエージェントを押し返すことができず，行き詰まりの状況を解消できない．たとえば，図 12 の例では，a_1 は暫時的な優先度として a_2 の優先度 +1 を持つ．よって a_1 は a_3, a_2 を押し出すことが可能だが，その直後に分岐点から離れるときに暫時的な優先度が解除され，a_2 が a_3, a_1 を押し戻すこととなり，その結果図 12 の状態を繰り返す行き詰まりとなる．他方，*False* を返した全エージェントが暫時的な優先度を用いる場合，その全てのエージェントが各自の目的地へ移動しようとするため，継続的に優先度の高いエージェントを押し返すこととなり，元に戻る可能性が下がる．たとえば，図 13 の例では，a_1, a_3, a_4 などの赤色のエージェントは，a_2 の優先度+1 を持つ．ここで a_3 が a_2 を押し出し暫時的な優先度が解除されるが，その後に続いて a_1 も同じ方向へ a_2, a_3 を押し出すことが可能である．a_4 も同様に a_2, a_3, a_1 を押し出し，これが継続する．この結果，暫時的な優先度を用いた全エージェントでこのループ領域から脱出したいエージェントは外の方向へ移動ができ，a_2 を押し返すことで行き詰まりが発生しにくくなる．

5.2 実験 2

5.2.1 実験の目的と設定

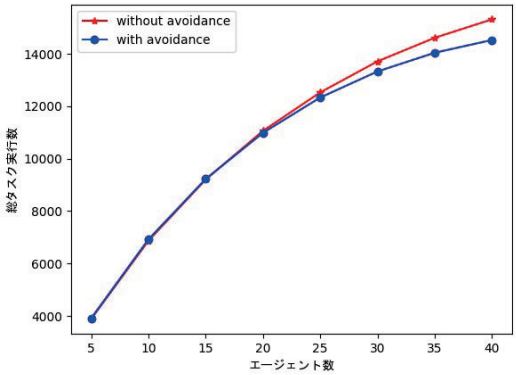
実験 2 では各環境において，4.2 節で提案した優先度の高いエージェントの行き先を勘案した移動可能判定によるタスク実行の効率化への貢献を調査する．なお，実験 1 では，本判定法を導入していないため，実験結果は再掲となる．また暫時的な優先度 p'_i を用いるのは，行き詰まりをした際に *False* を返した全エージェントとする．50000 ステップを 1 試行とし，30 試行の平均を使用した．エージェントは実験 1 と同様に 5，10，15，20，25，30，35，40 体と変化させた．

5.2.2 実験結果

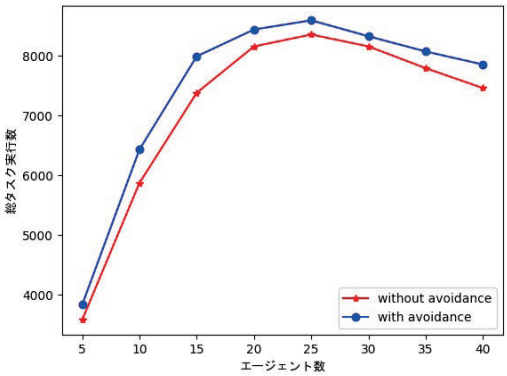
実験 2 の結果を図 11 に示す．実験 1 と同様に，30 テストのうち 1 テスト以上行き詰まりが起きたものには，×がプロットされている．環境 2 では優先度の高いエージェントの行先を勘案して移動可能判定をすることにより，勘案しない時よりもタスク実行数は改善しているのに対し，環境 1，3 ではタスク実行数の悪化が見られた．特に環境 3 においてはエージェント数が 25 体以上で，途中で行き詰まりが発生し，タスク実行数は急激に減少した．

環境 1 では，自分 (a_i) を押すエージェント (a_j) が移動したい方向と，a_i が移動したい方向が一致しているときは，a_i の移動したい方向を最後に移動可能判定をする．結果，自身の移動したい方向以外へ一時退避のように移動する．環境 1 は比較的交差点が少ないため，自分自身の移動したい方向以外へ移動すると大幅に余分な移動を強いられる．環境 1 においては自身を押すエージェントの行き先を勘案した場合の結果が，悪化したと考えられる．

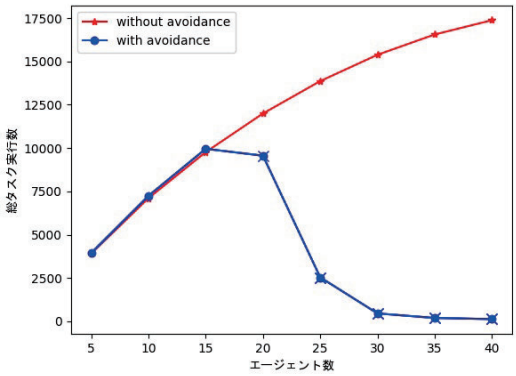
一方，環境 2 では，ほぼ格子上であり交差点（隣接ノードの個数が 3 つ以上のノード）が比較的多く，優先度の高



(a) 環境 1 でのタスク実行数の平均値



(b) 環境 2 でのタスク実行数の平均値



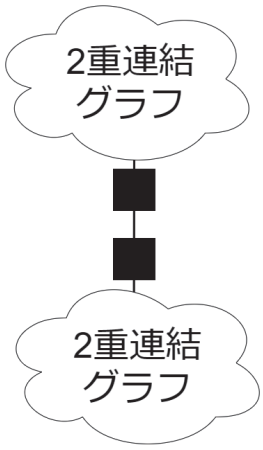
(c) 環境 3 でのタスク実行数の平均値

図 14: 実験 2 の結果

いエージェントに対して低いエージェントが脇道へ避けることで早期に競合を解消したため，タスク実行数が上昇したと考えられる．

環境 3 では自身を押すエージェントの行先を勘案する場合，行き詰まりを再発生させる可能性がある．たとえば，図 13 の右図で a_4 が a_1 をに押している状況において，a_1 は a_4 の移動したい方向である右方向の移動可能判定を最後に行う．a_1 も右に移動したいもののその優先順位を下げ，結果として初めに上方向に対して移動可能判定を行

(a) 2 重連結グラフ＋木構造



(b) 1 本の道を挟んだ 2 重連結グラフ

図 15: 今後の課題となる実行環境

い，これは移動可能であるため，a_1 は上方向に移動する．このようにループ形状を含む環境においては，自身を押すエージェントの行先を勘案して移動可能判定をするとループから抜け出すことができなくなる．このため，環境 3 ではエージェント数が 20 体以上に増えてくるとタスク実行数が急激に減少したと考えられる．

6. まとめ

本研究では，MAPD 問題を解決する PIBT の手法における環境制約 (2 重連結グラフ) に対し，2 重連結グラフに交差点を含まない道を付いた実験環境にするという緩和をしても，暫時的な優先度を導入した拡張 PIBT では搬送を継続可能であることを実験的に示した．また，2 重連結グラフに木構造の付いた図 15a のような環境や，2 重連結グラフ同士が 1 本の道で繋がっている図 15b のような環境において，目的地を制限することでタスクの実行が可能であることを確認した．また，その中で暫時的優先度を用いるエージェントの選択方法や優先度の高いエージェントの行き先の勘案の工夫を取り入れることで，実験環境により性能の変化が起きることを確認した．

今後の課題として，暫時的な優先度を扱うエージェントの選択方法をより工夫することで効率の良いタスク実行を目指す．暫時的な優先度は行き詰まりの解消に繋がるが，複数のエージェントが同じ値の暫時的な優先度を持つ状況では，その中で優先されるエージェントはランダムに選ばれる．これは目的地に近いエージェントの優先度が必ずしも上がるわけではなく，性能向上の余地があるといえる．加えて，暫時的な優先度から通常の優先度に戻る際の条件の工夫の検討も，今後の課題である．暫時的な優先度を持っている間は他のエージェントより優先的に動くことが可能であるが，他のエージェントは移動したい方向へ移動が不可能になることがある．そのため，暫時的な優先度は可能な限り早い段階で通常の優先度に戻す必要がある．また，今後は図 15a や図 15b の環境における継続的なタスク実行を保証するため，暫定的な優先度を持つエージェントの選択方法や，暫時的な優先度から通常の優先度へ戻る際の条件を検討する．加えて既存手法との性能の比較を行い，本提案手法の性能を定量的に評価する．

参考文献

[1] Dresner, K. and Stone, P.: A Multiagent Approach to Autonomous Intersection Management, *J.Artif.Intell.Res*, Vol. 31, pp. 591–656 (2008).
[2] Wurman, P., D'Andrea, R. and Mountz, M.: Coordinating hundreds of cooperative,Autonomous vehicles in warehouses, *AI magazine*, p. 29 (2008).
[3] Morris, R., S Pasareanu, C., Luckow, K. S., Malik, W., Ma, H., Kumar, T. and Koenig, S.: scheduling and monitoring for airport suface operations, *AAAI Workshop:Planning for Hybriid Systems* (2016).
[4] Ma, H., Li, J., Kumar, T. and Koenig, S.: Lifelong multi-agent path finding for online pickup and delivery tasks, *AAMAS*, pp. 837–845 (2017).
[5] Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Satich Kumar, T., Boyarski, E. and Bartak, R.: Multi-Agent pathfinding:Definitions,Variants,and Benchmarks, *AAAI* (2019).
[6] Sharon, G., Stern, R., Felner, A. and Sturtevant, N. R.: Conflict-based search for optimal multi-agent pathfinding, *Artif.Intel*, Vol. 219, pp. 40–66 (2015).
[7] Okumura, K., Machida, M., Defago, X. and Tamura, Y.: Priority inheritance with backtracking for iterative multi-agent path finding, *IJCAI* (2019).
[8] Luna, R. and Bekris, K. E.: Push and swap: Fast cooperative path-finding with completeness guarantees, *IJCAI*, pp. 294–300 (2011).
[9] Farinelli, A., Contini, A. and Zorzi, D.: Decentralized task assignment for multi-item pickup and delivery in logistic scenarios, *AAMAS* (2020).
[10] Ma, H., Honig, W., Kumar, T., Ayanian, N. and Koenig, S.: Lifelong path planning with kinematic constraints for multi-agent pickup and delivery, *AAAI* (2019).