Vol.2022-MPS-138 No.14
Vol.2022-BIO-70 No.14
2022/6/27

暫時的な優先度を導入したPIBT手法の拡張

藤谷 雪北[1,a)]　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 の基本的な性質を変え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 手法はステップごとに優先度を計算し，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，行き止まりや袋小路のような形状を含むマップでは行き詰まり (デッドロック) が発生するため，環境にその発生を防ぐ制約を設けている．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，更に不要な部分への移動を禁止する拡張を施し，先行研究で求められる条件を緩めても継続的な搬送ができることを述べる．よく知られた既存手法である Token Passing との比較実験を通し，その効率が優位であること，特に一般的な応用で想定されるような運搬箇所に集中や偏りがある場合にその効果が極めて高いことを示す．

キーワード：マルチエージェント，配送問題，優先度計算

Keywords: multiagentsystem, delivery, priority

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する機械やシステムが着目されている．例えば清掃ロボットは，事前情報なしに自動的に部屋のレイアウトを学習しながら清掃をする．また災害時に人が立ち入れない場所で，ロボットが人や物を探索する例も存在する．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，複数のエージェントが協力して共通のあるいは個々の目的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ている．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] などがある．しかし各エージェントの個別の独立した判断に基づく最適行動を同一環境で取ると，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発生頻度も高くなり，複数エージェントによる効率化を妨げる．これを防ぐには，これらの競合を回避する制御が重要である．

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 [3] 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り関連する多くの研究も存在する．MAPD の個々の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に対応する．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 [3], [4], [5]. 例えば [4] では，エージェントのタスク実行順に high-level と low-level の 2 段階で，すでに生成されたプランを変えず，それらと衝突のない経路を生成する Conflict Based Search (CBS) を提案した．[3] では，自動配送用に予め設計された倉庫の環境を想定し，token と呼ばれる共有メモリを参照しながら各エージェントが必要に応じて自律的に新しいプランを生成し，それを token に記録しながら，分散的な制御を達成している．本研究では，局所的な通信により衝突を避け，並列度の高い継続的なタスク実行が可能な *priority inheritance with backtracking*(PIBT) [5] 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，近隣のエージェントのみ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，衝突しないように次の移動位置を決める．PIBT は，各エージェントが自律的かつ局所的な通信のみで次位置を決めるため，エージェント数の増加に対するロバスト性はあるが，環境が 2 重連結グラ

[1] 早稲田大学 基幹理工学研究科 情報理工・情報通信専攻 東京都新宿区大久保 3 丁目 4-1
[a)] y.fujitani@isl.cs.waseda.ac.jp

フ，つまり任意の 2 ノードの組は，その他の任意の頂点を一つ削除してもそれらを結ぶパスが存在しなくてはならない[*1]という制限があり，たとえば袋小路のような形状のパスを含む環境では行き詰まり（デッドロック）が発生する．

そこで本研究では，この制約条件を緩めた環境でも動作するように PIBT を拡張する．具体的には，主領域は 2 重連結グラフ（以下主領域と呼ぶ）ではあるものの，これに一本道あるいは木構造の袋小路が含まれる環境でも適用可能とする．実際に，MAPD における積み上げ積み下ろしは，たとえば複数の搬入搬出口で待機するトラックへの積み込み・積み下ろしと保管エリアとの搬送に相当し，このような形状は多いと考える．ここでは既存の優先度に加え一時的に与えられる暫時的優先度を導入し，袋小路などで行き詰まりが発生しうる場合でも，暫時的優先度を利用して PIBT の基本的なアルゴリズムとその利点を損なうこと無く，デッドロックを回避可能とする．また，いくつかの環境を用意し，提案手法を MAPD でよく使われている *token passing* (TP) [3] と比較し，提案手法の効率性と限界を示す．

2. 関連研究

複数のエージェントが現在地からそれぞれおの目的地まで移動する Multi-Agent Path Finding (MAPF) 問題に対し，衝突の無い各エージェントの経路を生成する (マルチエージェントプランニングの) 研究が古くから存在する [6], [7], [8]．たとえば [6] では各エージェントが，他のエージェントのプランの情報を得た状態で，衝突のない経路を各エージェントごとに生成する Cooperative A*と，それらの拡張アルゴリズムを提案している．また，[8] は，A* を効率化した Enhanced partial expansion A*を提案し，これを MAPF に適用する手法を提案している．

上記の MAPF の繰り返しと捉えられる MAPD での経路生成とその制御の方法には，大きく分けて特定のエージェントが全体を把握し全てのプランを生成する集中制御手法 ([4], [9] など) と，各エージェントが周囲の状況から自律的にプランを生成する分散手法 ([3], [5], [10] など) がある．たとえば [4] では，エージェントのタスク実行順に high-level と low-level の 2 段階で，すでに生成されたプランと衝突のない経路を生成する Conflict Based Search (CBS) を提案した．[9] 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 操作により，エージェントの動きを制御している．しかし，集中制御手法では，エージェント数の増加とともにコストが増加するほか，局所的な競合回避のためプラン生成・修正が関連しない他のエージェントの行動の遅延をもたらす可能性もあり，システム全体の性能に影響する．

他方，[3] では，アマゾンの倉庫のような予めロボットによる自動配送用に設計された倉庫の環境で，token と呼ばれる全エージェントの共有メモリに各エージェントが自身の経路を書き込み，このアクセス権を持ったエージェントが必要に応じて自律的に新しいプランを生成し，ここに記録することで，分散的な制御を達成している．[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．分散制御手法ではコスト増加を回避できる可能性はあるが，他方，環境やタスク選択に関する制約も存在する．応用範囲を拡大させるためには，それらの制約を緩めることが必要となる．本研究でベースとなる PIBT [5] も分散制御手法に属すが，タスク選択の制限が少なく並列度を上げやすいことから，我々は PIBT の適用範囲を拡張させることを目的とする．

3. 準備

3.1 問題設定

n 体の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 $v \in V$ 上に存在でき，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．なお，ノードを必要に応じて追加し，エッジの長さはすべて 1 と仮定する．[*2] PIBT [5] では，G は自己ループや多重辺を含まない 2 重連結グラフを条件としている．各エージェントは，一辺の長さを 1 として長さ 2 の範囲のノードに存在するエージェントと通信可能とする．

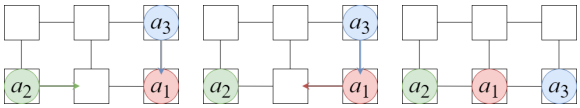
タスク τ は，スタート $\tau_s \in V$ とゴール $\tau_g \in V$ で表現でき，τ を割り当てられたエージェント $a \in A$ は τ_s にある物資を τ_g に運ぶものとする．ここでは単純化のため，τ を与えられたエージェントが τ_s に訪れると物資を受け取り，τ_g に到着すると荷物を降ろし，τ はが完了したとする．本研究では，タスクのスタート τ_s とゴール τ_g は後に定義するタスクが発生しうる場所から選択される．タスクは予め与えられるか，あるいは時間経過に応じて与えられ，それらをエージェントに順に割当てるかエージェントが選択[*3]し，それを実行する MAPD 問題を対象とする．

以下，離散時間を導入し，その単位をステップと呼ぶ．時刻 $t \in \mathbb{N}$ ($\mathbb{N}$ は自然数) におけるエージェント a_i の位置を $v_i(t) \in V$ と表す．PIBT を踏襲し a_i は以下のように，各ステップごとに現在地に留まるか隣接ノードの 1 つに移動できる．

[*1] 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とも言える．

[*2] PIBT では，隣接ノードへの移動を 1 timestep としてほぼ同等の所要時間で移動できることを仮定している．

[*3] TP では，エージェントが選択する必要があるため，実験を考慮し，このような条件とした．



(a) 行き詰まり　(b) 優先度継承　(c) 1 ステップ後

図 1: PIBT における優先度継承

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

エージェントは衝突回避のために同じノードに存在できず，すれ違いもできない．

$$v_i(t) \neq v_j(t) \quad (2)$$

$$v_i(t) \neq v_j(t+1) \vee v_i(t+1) \neq v_j(t) \quad (3)$$

なお同期した行動を想定し，以下のようなローテーションは可能とする．

$$v_i(t+1) = v_j(t) \wedge v_j(t+1) = v_k(t) \wedge \cdots \wedge v_l(t+1) = v_i(t) \quad (4)$$

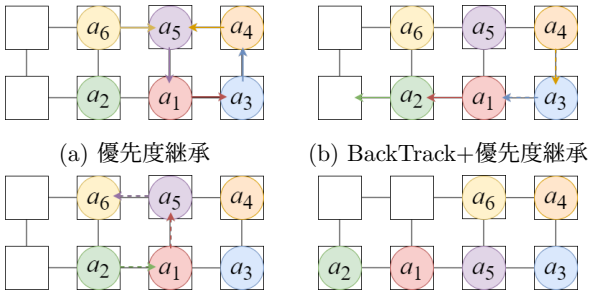
なお，上記の条件から $|V| \geq n$ となる．

3.2 PIBT

既存手法である PIBT [5] は，毎ステップ各エージェントが全順序関係にある優先度を計算し，その優先度の高いエージェントから次の行動（つまり次の移動位置）に順位をつけ，次位置を決定するアルゴリズムである．PIBT は必要に応じて隣接エージェントに優先度継承 (*Priority Inheritance*, PI) を再帰的に行い，エージェント同士のデッドロックを防止している．PI では，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 は次の位置としてその場所を放棄するが，同時に a_j の高い優先度を受け継ぐ．もし別のエージェント a_k からも次位置を指定されて競合が発生し次位置を決定できない時には，結果的に a_i は a_j と a_k の高い優先度を継承し，他方の次位置を放棄させる．

図 1 に継承の例を示す．ここでは，エージェント a_3，a_2，a_1 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_3 が下に進もうとするが，そこには a_1 が存在する．そのため a_1 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_1 よりも優先度の高い a_2 の次位置になっており，競合した行き詰まり状況となる．**図 1b** と**図 1c** は優先度継承した場合であり，a_1 は a_3 の優先度を継承し (図 1b)，a_1 は a_2 よりも優先度が暫時的に高くなり左へ移動できる (図 1c)．

また優先度継承が連続し，バックトラック (backtrack, BT) が発生する可能性もある．たとえば，**図 2** では (同様にエージェントの添字の数が大きいほど優先度は高いものとする)，**図 2a** は，優先度継承を用いてエージェント a_6 の優先度を a_5，a_1，a_3，a_4 の順番で継承するが，ここで行き詰まりが発生する．この状況を回避するために，**図 2b** では 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，優先度継承を解消する．これを受けた a_1 は別のルートがあるため，エージェント a_2 の方向への移動を a_2 に伝え，同時に優先度を a_2 に継承させる．**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．a_3, a_4 は，ここでは移動できずそのまま留まる．



(a) 優先度継承　(b) BackTrack+優先度継承

(c) BackTrack　(d) 1 ステップ後の状態

図 2: PIBT におけるバックトラック

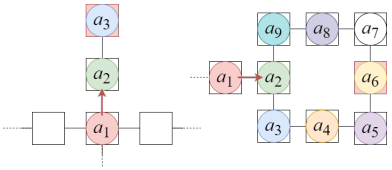
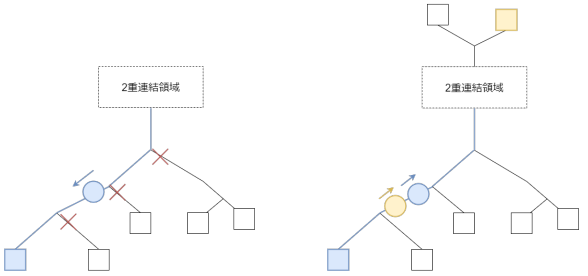


図 3: PIBT が行き詰まる例



(a) 最短経路のみを通る　(b) 2 重連結領域へ押し戻される

図 4: 木構造内におけるエージェントの動き (PIBTTP)

しかし 2 重連結領域という制約を除くと，デッドロックが発生する．**図 3** にその例を示す．ここでは a_1 が最も高い優先度を持つとするが，a_1 は物理的に他のエージェントを押し出せず，進むことができない．既存の PIBT では優先度をタスクのスタート地点あるいはゴール地点を離れてからの時間 (ステップ数) を基準に設定しているため，優先度の順序は変わらず行き詰まりを解決できない．

Vol.2022-MPS-138 No.14
Vol.2022-BIO-70 No.14
2022/6/27

4. 提案手法

本研究では PIBT を拡張し，2 重連結となる主領域のいくつかのノード（接続ノードと呼ぶ）に木構造 (以下，枝と呼ぶ) がついた環境 G においてもタスクを継続的に実行可能とする手法を提案する．一つの接続ノードに複数の木が接続していてもよいが，袋小路であるため木と主領域の接続点は 1 つとする．ここで提案する基本アルゴリズムを，*PIBT with Temporary Priority* (PIBTTP) と呼ぶ．Algorithm 1 に PIBTTP のアルゴリズムを示す．なお，PIBT と同様，エージェント数は主領域のノード数を超えないとする．また，

条件 1: 任意のタスクのスタートとゴールの位置は，同一の木には存在しない．

条件 2: エージェント a_i が主領域にいるなら自分の目的地を含まない木には入らない．

条件 3: エージェント a_i が木構造内にいるなら現在の目的地までの最短経路上のみを移動する．

を仮定する．条件 1 は，たとえば倉庫内の保管場所と搬入・搬出口間の運搬を想定し，木の枝の部分はトラックなどの待機箇所などとすれば自然な条件と考える．条件 2，3 は木構造内では現在の目的地への最短経路以外は通らないことを意味し，例えばその経路上に無い袋小路に入って他のエージェントに影響しないことを仮定している．

まず，ε_i $(i = 1, \ldots, n)$ を 0 以上 1 未満の実数で，$\varepsilon_i \neq \varepsilon_j$ (if $i \neq j$) となるものをランダムに設定する．PIBTTP では各ステップの始めに，次ステップでの行先の決まっていないエージェントの集合 U と，次ステップでエージェントの行くノードの集合 O を用意する [Lines 1,2]．初期状態では $U = A$，$O = \varnothing$ である．次に，各エージェントは各々の優先度を計算する [Lines 3-9]．ここで，$f_i(v)$ をノード v から a_i のさしあたりの目的地（タスクの進行に依存して，スタート地点もしくはゴール地点に相当）までの最短経路長とし，エージェント $a_i \in A$ の基本優先度の $p_i = -f_i(v_i(t)) + \varepsilon_i$，$i$ が木の内部にいてかつそのエージェントの目的地が木の外部にある場合，その優先度 $p_i = 1 + \varepsilon_i$ とする．なお，後者の場合の方が優先度が大きくなり，また ε_i の定義から同一の優先度を持つエージェントは存在しない．次に，$a_i \in U$ で最も高い優先度を持つエージェント a_i に対し，$PIBT(a_i, \perp)$ を実行する [Lines 10-13]．ここで $PIBT(a_i, a_j)$ は，a_i が a_j の優先度を継承する場合を示し，特に継承が無い場合は，$a_j = \perp$ と記す．

エージェント a_i の次の移動先を決定する PIBT(a_i, a_j) では，初めにエージェント a_i の行先候補の集合 C_i を求める [Lines 17-22]．具体的には C_i は，a_i の現在ノード及びそれと隣接したノードのうち移動可能なノードとする．条件 1 から 3 を考慮し，ここで移動できないノードは以下の通りである．

Algorithm 1 PIBTTP

```
 1: U ← A // Agents that do not decide next nodes.
 2: O ← ∅// Nodes already occupied by agent
 3: for a_i ∈ A do
 4:     if a_i が木にいてその目的地がその中の枝に無い then
 5:         p_i ← 1 + ε_i
 6:     else
 7:         p_i ← −f_i(v_i(t)) + ε_i
 8:     end if
 9: end for
10: while U ≠ ∅ do
11:     a ← agent in U with the highest priority
12:     PIBT(a,⊥)
13: end while
14:
15: function PIBT(a_i, a_j)
16:     U ← U \ {a_i}
17:     C_i ← ({v | (v, v_i(t)) ∈ E} ∪ {v_i(t)}
18:     if a_i が 2 重連結領域内 then
19:         C_i ← C_i \ (O ∪ {v_j(t)}∪ 目的地を含まない木)
20:     else
21:         C_i ← C_i \ (O ∪ {v_j(t)}∪ 最短経路外ノード)
22:     end if
23:     while C_i ≠ ∅ do
24:         v_i^* ← arg min_{v∈C_i} f_i(v)
25:         O ← O ∪ {v_i^*}
26:         if ∃a_k ∈ U such that v_i^* = v_k(t) then
27:             if PIBT(a_k, a_i) is VALID then
28:                 v_i(t + 1) ← v_i^*
29:                 return VALID
30:             else
31:                 C_i ← C_i \ O
32:             end if
33:         else
34:             v_i(t + 1) ← v_i^*
35:             return VALID
36:         end if
37:     end while
38:     v_i(t + 1) ← v_i(t)
39:     return INVALID
40: end function
```

(i) 既に a_i よりも優先度の高いエージェントに占領されているノード

(ii) a_i が a_j から優先度継承をしている場合は a_j が現在いるノード

(iii) a_i が主領域にいるなら自分の目的地を含まない木構造内のノード，木構造内にいるなら目的地までの最短経路上に無いノード (**図 4a** 参照)

また C_i の各要素に目的地までの距離に基づく順序を導入する．具体的には，$\forall v \in C_i$ に対し，$f_i(v)$ の値が小さい（目的地に近い）ものほど順位が高くなる．

次に，C_i の中で順位の高いノード v_i^* から順番に，移動可能性を判定する [Lines 23-37]．v_i^* にエージェント a_k がいるならば，PIBT(a_k, a_i) を実行する．a_k が次ステップで行くマスが確定できれば PIBT(a_k, a_i) は $valid$ を返

し，このとき a_i は v_i^* へ進み，PIBT(a_i, a_j) はさらに $valid$ を返す．一方 a_k が次ステップで行くノードが確定できず PIBT(a_k, a_i) が $invalid$ を返す場合，a_i は v_i^* に進めず，C_i から v_i^* を除く．一方，v_i^* にエージェントがいない場合は，a_i は v_i^* に進み，PIBT(a_i, a_j) は $valid$ を返す．最後に，C_i が空集合となり a_i が進めるノードが無い場合，a_i は現在のマスに留まり，PIBT(a_i, a_j) は $invalid$ を返す [Lines 38-39].

PIBTTP では暫時的な優先度 [Line 5] により，木の末端の目的地に到達したエージェントが末端の目的に向かうエージェントによって押し返されることを防止し，必ず主領域へ戻ることを保証する (**図 4b**)．一度主領域へ戻ると，条件 1 から目的地はそれまでの木構造にはないため，元の木に戻ることはない．したがって以下の補題は明らかである．

Lemma 1. 木構造内にある目的地に達したエージェントは，その木から主領域に到達できる．

Lemma 2. 木の内部に目的地を持つエージェントが主領域のその木の接続ノードに到着すれば，その目的地にも到達できる．

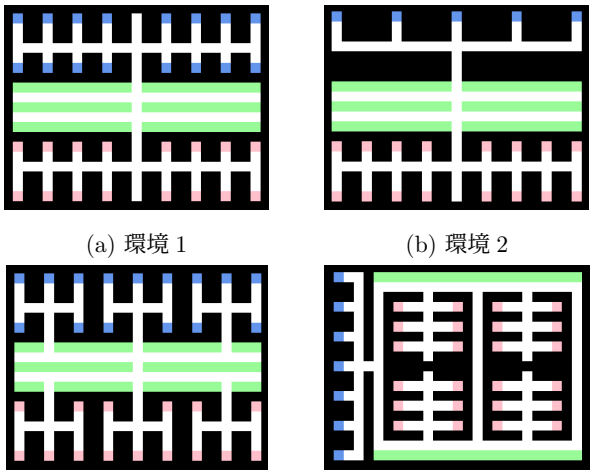
後者の補題はタスク数が有限であれば押し戻しも有限回で終了し，永遠に目的地に到達できないことはない．したがって，新たな目的地を設定したエージェントは必ず主領域に到達でき (Lemma 1)，そのエージェントの目的地が主領域内にあれば，既存の PIBT のアルゴリズムにより目的地への到達性が保障される（条件 2 から押されて木に入ることはない）．また別の木構造の内部に目的地が存在する場合は，PIBT により主領域内でその木の接続ノードへの到達性は保障され，またそこから木の内部の目的地へも到達できる (Lemma 2)．よって，タスクのゴールがある木にあり，そこに達しタスクを完了したエージェントは，(a) 次のタスクのスタートが同一の木に存在しないものを選ぶか，(b) さしあたりの目的地を接続ノードに設定し，そのあとに次のタスクが与えられるとすると，以下の到達性が保障される．

Theorem 1. 環境 G は *2* 重連結を満たす主領域と主領域内の（接続）ノードから延びる木構造からなる領域とする．与えられるタスクは条件 *1* を満たし，その数は有限とすると，A は停止することなく与えられたタスクをすべて完了できる．

5. 評価実験

5.1 実験の目的と設定

本提案手法を評価するために，図**図 5** に示す 4 つの環境で実験し，比較手法 Token Passing (TP) [3] とその効率を比較した．これらの環境の図で，赤いノードはタスクのスタートとなり得る荷物の発生個所（ストレージなど）であり，青いノードは運搬のゴールとなりうる場所である．したがって，これらの環境では，自然に条件 (a) を満たし，各エージェントは任意のタスクを次のタスクとして選択できる．環境 1 では，荷物の搬送場所と搬送先が木構造になっている倉庫を想定している．この環境は木構造の部分が他に比べて深く，PIBTTP では木構造の内部で長い距離の押し戻しの可能性もあるが，一方で，多数のスタートとゴール地点が存在するため TP では多くのタスクの並列行動が可能で，相対的に PIBTTP が不利な環境と言える．環境 2 と 3 は環境 1 と基本的な構造を変えず，環境 2 ではタスクのスタート地点とゴール地点の数をアンバランスとし，環境 3 は木構造の数を増やし，代わりに木の深さを浅くした．環境 4 は，より現実の倉庫に近い環境を想定し，スタートに相当する積み出し地点が倉庫内に密に置かれたラックにあり，そこへのアクセスを木構造とし，ゴールに相当する搬送先はトラックの積み込み場所を想定している．一般にラックの数とトラックなどの搬出入口の数は，アンバランスな数と思われる．なお，全ての環境において PIBT では，行き詰まりを回避できず，適用できない．

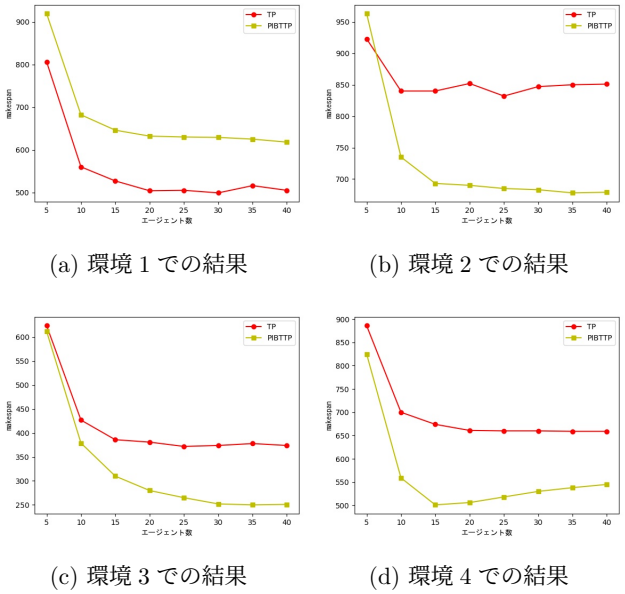


(a) 環境 1　(b) 環境 2　(c) 環境 3　(d) 環境 4

図 5: 実験環境

実験の開始時に，エージェントを主領域の薄緑のセルにランダムに配置し[*4]，また 50 タスクをランダムに生成し，これをタスク集合 $\mathcal{T}$ とする．タスクを持たないエージェントは，実行可能なタスクを $\mathcal{T}$ から取り出す．$\mathcal{T} = \varnothing$ のとき，TP の場合は，さらに選択可能なタスクが残っていない時，自分の初期位置に戻る．タスク集合内の全タスクの完了を 1 試行とし，200 試行した．評価指標は，200 試行に要したステップ数の平均とする．また，環境内のエージェント数は 5 から 40 まで，5 ずつ変化させた．なお，本実験環境では，タスクのスタートとゴール地点を分けていたが，スタートとゴール地点に共になり得る点を追加する場合は，エージェントはタスク選択時に，同一の木の中に

[*4] TP の適用条件を考慮し，薄緑の領域は停止してもルートが確保でき，他のエージェントの妨げになりにくいように設定した．



(a) 環境 1 での結果　(b) 環境 2 での結果

(c) 環境 3 での結果　(d) 環境 4 での結果

図 6: 実験結果

スタート地点を含むタスクを選択しないか，一度，主領域に戻ってから次のタスクを選択するようにすればよい (条件 (a)(b)).

5.2 実験結果

各環境において，エージェント数を変化させたときのタスク完了までの所要時間を**図 6** に示す．環境 1 (**図 6a**) と環境 2 の結果 (**図 6b**) を比較すると，環境 1 は提案手法の性能が比較手法より低いが，環境 2 ではエージェント数が 10 体以上のときには提案手法の性能が比較手法よりも優れ，エージェントの並列性による効果が現れている．比較手法の TP では，エージェントが，スタートやゴール地点の数より多いと同時にエージェントが動作できず，並列性は頭打ちとなる．一方，提案手法はエージェント数や環境のスタートやゴール地点の数にかかわらず，同時に複数のエージェントが同じ目的地を設定できる．他方環境 1 では，スタート・ゴール地点ともに多くかつ同数であるので，TP の制限が緩くなり，並列性も上げられる．一方，PIBTTP でも制限はなく並列度は高いが，木構造内の，特に奥にある目的地に向かうエージェントが，目的地到着前に主領域 (2 重連結領域) まで押し戻されることがある．これが大きなオーバーヘッドとなることが要因と考えられる．

また**図 6c** や**図 6d** から，環境 3 と環境 4 でも提案手法が比較手法よりも効率的である．提案手法では，エージェントは木構造内では目的地までの最短経路上のみを通り，その他の枝や部分木には入らないとしたが，環境 3 と環境 4 は環境 1 と比べ木の深さがやや浅く，また環境 2 では，タスクのスタートとゴール地点が，環境 4 では，タスクのスタート地点を含む木の数が多い．これは短時間で木構造の出入りができること，各エージェントの目的地が存在する木構造が分散しやすく同じ木構造にエージェントが集まりにくいからである．この結果，提案手法で性能低下になりうる木構造内の目的地に向かうエージェントが，すでに目的地に到達し主領域まで戻ろうとする別のエージェントに押し戻される，という状況が少なくなる．特に環境 3 では，スタート・ゴール地点とも多数の木構造に分かれており，(TP でも) 効率が高いことがわかる．

以上の議論から，比較手法と比べて提案手法は，並列性での優位性が見込まれるタスクのスタート地点とゴール地点が多い環境，そして木構造内での衝突頻度が減るような深さの浅い木構造，複数の木構造のある環境において性能が良くなると考えられる．

6. 結論

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を緩めても，適用できる PIBTTP を提案した．提案手法の PIBTTP では暫時的な優先度と移動方向の制限を用いることで，搬送を継続可能であることを確認した．また PIBTTP は PIBT 手法の拡張であるため並列実行性が高く，環境によっては比較手法よりも性能が高いことを確認した．他方，主領域に接続される木構造の深さや枝の数によっても，性能が変化し，特に長い押し戻しが効率を低下させるすることを確認した．今後の課題として，これらの影響を受けにくい手法に取り組む．

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] 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).
[4] 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).
[5] Okumura, K., Machida, M., Defago, X. and Tamura, Y.: Priority inheritance with backtracking for iterative multi-agent path finding, *IJCAI* (2019).
[6] Silver, D.: Coorperative pathfinding, *AIIDE*, Vol. 1, pp. 117–122 (2005).
[7] Standley, T. S.: Finding optimal solutions to cooperative pathfinding problems, *AAAI*, Vol. 1, pp. 28–29 (2010).
[8] Wagner, G. and Choset, H.: Subdimensional expansion for multirobot path planning, *Artif.Intell*, Vol. 219, pp. 1–24 (2015).
[9] Luna, R. and Bekris, K. E.: Push and swap: Fast cooperative path-finding with completeness guarantees, *IJCAI*, pp. 294–300 (2011).
[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).