Vol.2022-ICS-205 No.4
2022/2/21

マルチエージェント搬送問題のための グラフ理論を活用したデッドロック回避手法の提案

山内 智貴[1,a)]　宮下 裕貴[1,b)]　菅原 俊治[1,c)]

概要：本研究では迷路状の制限された環境でも輸送効率を向上するため，*multi-agent pickup and delivery*（MAPD）問題に対してグラフ理論を活用したデッドロック回避手法 *standby-based deadlock avoidance*（SBDA）を提案する．複数エージェントが衝突せずに資材を繰り返し回収・運搬する MAPD 問題が注目されているが，従来の MAPD アルゴリズムの多くは自動倉庫のような特別に設計されたグリッド状の環境を想定する．それらの環境にはエージェントが長時間滞在できる集配場所が多く，グリッド内の移動の自由さから，衝突回避のための迂回路も豊富である．一方，災害現場や建設現場のような迷路状の環境には集配場所が少なく，それらの数が偏るため，多くのエージェントが集配場所に集中する結果，輸送効率の悪化や立ち往生，デッドロックに陥りやすい．SBDA はグラフ理論の *articulation-point-finding* アルゴリズムを用いてリアルタイムに決定される待機ノードを使用し，エージェントが有限時間そこに滞在することを保証する．我々は実験により，提案手法が従来手法の輸送効率を上回ることを示した．

キーワード：マルチエージェント搬送問題，マルチエージェント経路計画，デッドロック回避

1. はじめに

近年，実世界の複雑かつ膨大なタスクに対するマルチエージェントシステムの活用が注目されている．例えば自動倉庫での搬送ロボット [10] やライドシェアリングサービス [13]，複数ドローンによる輸送システム [2] などがある．しかし単純にエージェント数を増やすと，冗長な移動や衝突などのリソース競合の発生により，かえって非効率にもなる．したがって全体性能の改善には，エージェント間の悪影響を回避する協調行動が不可欠である．特に我々のアプリケーションである，重くて大きな資材を運ぶ大型の搬送ロボットをエージェントとした，制限された環境に適用する集配システムでは衝突回避は必須である．

この問題は，集荷と配送のタスクが個々のエージェントに同時に割り当てられる *multi-agent pickup and delivery*（MAPD）問題として定式化される．割り当てられたエージェントは資材置き場に移動して指定された資材を積み込み，それを必要な場所に配送して積み下ろす必要がある．したがって実行するべきタスクが多数存在する MAPD 問題は，複数エージェントが目的地までの衝突しない経路を生成する *multi-agent path-finding*（MAPF）の反復と考えられる．MAPF 問題は最適解の取得が一般に NP 困難であり [6]，そのため MAPD 問題は更に時間がかかる．

MAPF/MAPD 問題に焦点を当てた研究は数多くあり [3], [4], [5], [8]，それらの結果は実世界のアプリケーションで活用されている．これらのアプリケーションでは，衝突やデッドロックのない計画を生成することが中心的な課題となる．例えば [8] は，エージェントがデッドロックを回避するために局所的な通信と優先順位に基づいて短い時間内の行動を決定する *priority inheritance with backtracking*（PIBT）を提案した．[5] はタスクの集配場所を含む *endpoint* を独占的に保持する *holding task endpoints*（HTE）[3], [4] を用いて衝突を回避する *token passing*（TP）を提案した．[4] は，各エージェントが固有の駐車場への *dummy paths* を常に予約することで，同じ endpoint を持つタスクを並列実行可能にする *reserving dummy paths*（RDP）を導入した．

しかし，これらの手法は我々の対象環境では使用できない．例えば PIBT では，完全性を保証するために環境が二重連結である必要があるが，我々の環境はこの要求を満たさない．HTE と RDP では効率的な移動のため，エージェントが任意の時間滞在できる endpoint と，長さが同等の迂回路を多く持つグリッド状の環境を想定する．このよう

[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
[a)] t.yamauchi@isl.cs.waseda.ac.jp
[b)] y.miyashita@isl.cs.waseda.ac.jp
[c)] sugawara@isl.cs.waseda.ac.jp

な要求は自動倉庫のような特別に設計された環境では可能だが，我々の環境は災害現場や建設現場のようなアドホックで迷路状の環境であり，通常集配場所が少なく，それらの数も偏るため，一部の endpoint 付近で局所的に混雑する可能性がある．さらに迂回路の数は限られており，距離が非常に長くなることが多い．

そこで我々は，迷路状の環境でもエージェントが同じ endpoint を持つタスクを並列実行可能にすることで輸送効率を向上させるデッドロック回避手法 *standby-based deadlock avoidance*（SBDA）を提案する．我々の対象環境には endpoint が非常に少ないが，この手法を TP に統合した．SBDA では，エージェントが目的地（すなわち endpoint）付近で有限時間待機することを保証した待機ノードを導入する．待機ノードの集合はエージェントが滞在を予約する度に変化するが，グラフ理論の *articulation-point-finding*（APF）アルゴリズム [9] を用いてリアルタイムで効率的に決定できる．他エージェントが目的地として到着済み，または移動中の endpoint に向かって移動する場合，SBDA により，エージェントは立ち往生やデッドロック回避のために endpoint 付近に残る待機ノードで一時待機してから順番に目的地に向かう．我々は様々な実験条件下でベースラインとして HTE を用いた手法と比較して，提案手法の性能を評価した．そして迷路状の制限された環境において，提案手法がベースライン手法の性能を上回ることを示す．なお，本稿は [12] を簡略化したものである．

2. 準備

2.1 問題設定

MAPD 問題はエージェント集合 $A = \{1, \dots, M\}$, タスク集合 $\mathcal{T} = \{\tau_1, \dots, \tau_N\}$, 2 次元ユークリッド空間に埋め込み可能な無向連結グラフ $G = (V, E)$ から構成される．ノード $v \in V$ は場所，エッジ $(u, v) \in E$ $(u, v \in V)$ はエージェントが u と v の間で移動可能な通路に相当する．エッジ (u, v) の長さは $l(u, v)$ と表記する．ノード v_1 と v_2 の間の距離は v_1 から v_2 への最短経路に現れるエッジの長さの和として定義される．Endpoint は G の dead-end に設定されると仮定する．我々のエージェントは前方にピッカーを持つ大型フォークリフト型の自律ロボットで，重い資材（500kg〜1ton）を運び，特定のノードで特定の方向にピッカーを使うことで資材を積み込み（load）または積み下ろし（unload）できる．我々は離散時間 $t \in \mathbb{Z}^+$（$\mathbb{Z}^+$ は正の整数集合）を導入する．

エージェント $i \in A$ に対して，時刻 t における i の向き $o_i^t \in \mathbb{Z}^+$ と進行方向 $d_i^t \in \mathbb{Z}^+$ を $0 \leq o_i^t, d_i^t < 360$ として D 刻みで定義する．ここで $o_i^t = 0$ と $d_i^t = 0$ は G の北側の向きと方向を示す．また，可能な向きの集合を $\mathcal{D}$ と表記する．例えば $D = 90$ とすると $\mathcal{D} = \{0, 90, 180, 270\} \ni o_i^t, d_i^t$ となる．簡単のため $D = 90$ と仮定するが，D は環境構造に応じた任意の数を持てる．

エージェントは任意のノード上で *move*，*rotate*，*wait*，*load*，*unload* の動作を実行できる．これらの持続時間をそれぞれ，長さ $l = l(u, v)$，自転角度 $\theta \in \mathbb{Z}^+$，待ち時間 t を用いて $T_{mo}(l)$，$T_{ro}(\theta)$，$T_{wa}(t)$，T_{ld}，T_{ul} で表わす．時刻 t に i が v にいるとする．*move* により，i は自転によって向き o_i^t を適切に変えた後，エッジ (u, v) に沿って u へ前進または後退する．*rotate* により，i は o_i^t から時計回り (D) または反時計回り $(-D)$ に D 度自転する．すなわち v にて $o_i^{t+T_{ro}(D)} = o_i^t \pm D$ となる．エージェント i は $t = 0$ の開始位置である固有の駐車ノード $park_i \in V$ を持ち [4]，i が実行すべきタスクを持たない場合はそこに戻って留まる．駐車ノードは図 2 にて赤い四角で表現される．

タスク τ_j はタプル $\tau_j = (\sigma_{\tau_j}^{ld}, \sigma_{\tau_j}^{ul}, \phi_{\tau_j})$ で指定される．$\sigma_{\tau_j}^{ld} = (v_{\tau_j}^{ld}, o_{\tau_j}^{ld})$ $(\in V \times \mathcal{D})$ は資材 ϕ_{τ_j} を積み込む位置と向き，$\sigma_{\tau_j}^{ul} = (v_{\tau_j}^{ul}, o_{\tau_j}^{ul})$ $(\in V \times \mathcal{D})$ は ϕ_{τ_j} を積み下ろす位置と向きとする．エージェントが資材の積み込み・積み下ろしを行う場合，ピッカーの方向を考慮して特定の方向を向く必要がある．エージェントは $\mathcal{T}$ の全タスクを衝突やデッドロックなしに完了する必要があり，その後 $park_i$ に戻る．

2.2 Token Passing

提案する SBDA はいくつかの従来の MAPD アルゴリズムに適応可能だが，本論文では MAPD インスタンスを効率的に解くために TP に統合できることを示す．TP [5] は，エージェントが自らタスクを選択し，トークンの情報を用いて経路を生成する，有名な MAPD アルゴリズムである．トークンは全エージェントの現在の経路，現在エージェントに割り当てられているタスク，およびエージェントに割り当てられていない残りのタスクを含む同期された共有メモリブロックである．

MAPD 問題の endpoint の集合 V_{ep} $(\subset V)$ は，タスクの集配可能地点である全 *task endpoint* と，各エージェントの初期位置 (すなわち $park_i$) を含む *non-task endpoint* から構成される．TP はどのエージェントも，他エージェントの現在のタスクのための移動を妨げることなく endpoint に任意の有限時間滞在できることを仮定する．task endpoint の集合を V_{tsk} $(\subset V_{ep})$ で表す．全 endpoint の位置は事前にエージェントに与えられると仮定する．したがって，集配場所が *open* な endpoint，つまり現在のトークンにおいて実行中の計画の endpoint として現れないタスクをエージェントが一つ選択すると，トークンの内容を見て衝突しない経路が生成される．そして TP に用いられる HTE は，衝突回避のために task endpoint を保持する．

全ての MAPD インスタンスが解決可能ではないが，*well-formed* MAPD インスタンスは常に解決可能である [1]．TP では，MAPD インスタンスは (a) タスク数が有限であり，(b)non-task endpoint の数がエージェント数以上で，(c)

任意の 2 つの endpoint 間に他の endpoint を横切らない経路が存在する場合に限り，well-formed である [5]．Well-formed MAPD インスタンスでは，エージェントはいつでも non-task endpoint（例えば $park_i$）に移動でき，他エージェントとの衝突回避のために必要なだけそこに滞在できる．この行動により，過密な環境を避けるためにエージェント数を減らせるかもしれない．Endpoint に関する仮定は明らかにこれらの要件を成立させる．ただし複数エージェントは endpoint が重なるタスクを並列実行できないため，我々が検討する迷路状の環境では，このアプローチは効率が大幅に低下する．

3. 提案手法

3.1 状態管理トークン

我々は，迷路状の環境下でもエージェントが同じ endpoint を持つタスクを並列実行可能にすることで，輸送効率を向上させる新しいデッドロック回避手法 SBDA を提案する．プランニングやエージェント，待機ノードの状態管理や，他エージェントとの競合検出のため，我々はトークン [5] と *synchronized block of information* [11] の拡張である状態管理トークン（*status management token*, SMT）を導入する．ここで競合とは，複数エージェントが同時に同一ノード $v \in V$ を占有するか，同一エッジを横切る状況と定義する．SMT には *reservation table*（RT）と *task execution status table*（TEST），*standby-node status table*（SST）が含まれる．RT は衝突しない計画の生成に用いる有効な予約データであるタプル $(v, [s_v^i, e_v^i], i)$ の集合で，$[s_v^i, e_v^i]$ はエージェント i のノード v に対する占有時間とする．$e_v^i < t_c$（t_c は現在時刻）のとき，タプルは期限切れとなって RT から削除される．TEST はタプル (τ, v, i) の集合で，τ は現在 i が実行中のタスク，v は τ で指定された積み込みまたは積み下ろしノードである．したがって，i がタスク τ を選択したときに 2 つのタプル (τ, v_τ^{ld}, i) と (τ, v_τ^{ul}, i) が TEST に追加され，i が v_τ^{ld} または v_τ^{ul} に到着したときに該当エントリは除去される．SST は全待機ノードの状態管理に使用され，SBDA によって動的に参照・変更される．SST の構造は 3.2 節で説明する．

SMT は共有可能なメモリ領域であり，TP のトークンと同様に特定の時間に 1 エージェントのみが排他的にアクセスできる．SMT は性能の若干のボトルネックになるが，我々が想定するロボットの移動速度は速くないため，現実的なエージェント数（例えば図 2 に示す実験環境では 30 エージェント以下）では相互排除によるオーバーヘッドに要する時間は無視できる程度である．

3.2 待機可能ノードの検出

待機ノードとは，直感的には TP の non-task endpoint と同様に，エージェントが割り当てられたタスクの endpoint に向かう途中で有限時間待つことが保証されるノードであり，動的に識別できる．駐車ノードを除き，non-task endpoint は事前に与えられないと仮定する．*articulation point*（AP）と，必要な時に待機ノードとして使用される待機可能ノードを定義する．

Definition 3.1. G において，ノード自体と関連するエッジを除去すると G の連結領域が分割されて連結成分の数が増える場合，そのノードを articulation point とする．

Definition 3.2. G において，*AP* でも *endpoint* でも *dead-end* でもないノードは G の待機可能ノードとする．

$G = (V, E)$ における全待機可能ノードの集合を $\mathcal{S}_{psn}(G)$ $(\subset V)$ とする．この定義から次の命題は明らかである．

Proposition 3.3. $G = (V, E)$ が連結グラフかつ $\mathcal{S}_{psn}(G) \neq \varnothing$ の場合，G から $\mathcal{S}_{psn}(G)$ の任意のノードを除去して生成した部分グラフは連結グラフである．

そのため，あるエージェントが待機可能ノードに留まっても，他エージェントはそのノードを通らずに目的地に到着する経路を生成できる．したがって以下の系が得られる．

Corollary 3.4. 待機ノードを予約したエージェントは，他エージェントの移動を妨げずに有限時間そこに留まれる．

全待機可能ノードの集合 $\mathcal{S}_{psn}(G)$ は計算量 $O(|V| + |E|)$ [7] の Tarjan's algorithm [9] などの APF アルゴリズムを用いて効率的に識別できる．G_t を時刻 t における G の修正された部分グラフとして，以下のように待機ノードを定義する．

Definition 3.5. エージェント $i \in A$ が滞在を予約したとき，待機可能ノード $v \in \mathcal{S}_{psn}(G_t)$ は待機ノードとなる．

エージェントが待機ノード v から離れた場合，v は待機ノードではなくなる．我々はエージェントが予約している全待機ノードの集合を $\mathcal{S}_t = \{(v, i)) \mid i \in A$ は v を待機ノードとして予約 $\}$ とする．

エージェントが待機ノードへの滞在予約を作成した場合（予約の作成方法は 3.4 節にて後述），他エージェントはそのノードを通過できなくなる．これは待機ノード（と関連するエッジ）の除去により，グラフ G の構造が一時的に修正されることを意味する．時刻 t における修正されたグラフを $G_t = (V_t, E_t)$ と表記する．したがって G_t の待機可能ノードの集合は $\mathcal{S}_{psn}(G_t)$ で示され，これも APF アルゴリズムを用いて効率的に識別される．なお $G = G_0$ とする．

エージェントが MAPD 問題のタスクを開始する前に，SBDA は各 endpoint v_{tsk} に対して関連待機可能ノードの集合 $s(v_{tsk})$ を生成する．$\forall v_{tsk}$ $(\in V_{tsk})$ に対して，SBDA は以下の式を用いて $s(v_{tsk})$ $(\subset \mathcal{S}_{psn}(G))$ を計算する．

$$s(v_{tsk}) = \{v_{psn} \in \mathcal{S}_{psn}(G) | dist(v_{psn}, v_{tsk}) \leq \alpha\}$$

ここで $dist(v_1, v_2)$ は v_1 と v_2 の距離，すなわち両ノード間の最短経路長とする．パラメータ α (≥ 0) は，endpoint v_{tsk} から待機ノードまでの距離の閾値である．なお，$v_{tsk}, v'_{tsk} \in$

Algorithm 1 Task selection by agent i

```
1: function SelectTask(i)
2:     T' = Set of tasks satisfying Cond. 1.
3:     // where τ = (σ_τ^ld, σ_τ^ul, φ_τ), and σ_τ^ld = (v_τ^ld, o_τ^ld)
4:     if T' ≠ ∅ then
5:         τ* ← arg min_{τ∈T'} dist(v_c^i, v_τ^ld) // v_c^i: current location
6:         T ← T \ τ*; return τ*
7:     else return false
8:     end if
9: end function
```

V_{tsk} に対して $s(v_{tsk}) = \varnothing$ かつ $s(v_{tsk}) \cap s(v'_{tsk}) \neq \varnothing$ の可能性がある．同様に，t における v_{tsk} に対する待機可能ノードの集合を $s_t(v_{tsk}) = s(v_{tsk}) \cap \mathcal{S}_{psn}(G_t)$ で示す．したがって，$s(v_{tsk}) = s_0(v_{tsk})$ は $G\ (= G_0)$ に対する初期の待機可能ノードの集合である．また関連待機可能ノードに含まれない待機可能ノードの集合を以下で定義する．

$$\mathcal{S}^c_{psn}(G_t) = \mathcal{S}_{psn}(G_t) \setminus \bigcup_{v \in V_{tsk}} s_t(v)$$

$\mathcal{S}^c_{psn}(G_t)$ の要素を t における自由待機可能ノードと呼ぶ．なお $\mathcal{S}^c_{psn}(G_t) = \varnothing$ の可能性がある．

待機ノードに関する情報は SST に格納される．t における SST は (1) 待機可能ノードの初期集合 $\mathcal{S}_{psn}(G)$，(2) endpoint とその待機可能ノードの全ペアの初期集合 $\{(v_{tsk}, s(v_{tsk})) \mid v_{tsk} \in V_{tsk}\}$，(3) $(v_{sn}, i) \in \mathcal{S}_t$ の形式で表される，待機ノードとそのノードを予約するエージェント i のペアの集合，(4) $\mathcal{S}^c_{psn}(G)$ の自由待機可能ノードに一時的に留まるエージェント i の集合 *crowded list*，$CL\ (\subset A)$ で構成される．

3.3 タスク選択プロセス

エージェントが MAPD 問題のタスクを開始した後，SBDA におけるエージェントは SMT の待機可能ノードに基づいて実行するタスクを選択し，必要に応じて待機ノードを選択することで目的地を決定，そこへの経路を生成する．エージェントはこのプロセスの間，現在の SMT に排他的にアクセスする．

待機可能ノードに基づくタスク選択プロセス SelectTask(i) の擬似コードを Algorithm 1 に示す．これは $\mathcal{T} \neq \varnothing$ のときにのみ適用され，さもなければエージェント i は駐車ノード $park_i$ に戻る．$v_{psn} \in \mathcal{S}_{psn}(G_t)$ に対して，全エージェントが現在の全計画に対して最後に v_{psn} を通過する時刻を $e^*_{v_{psn},t}$ とする．これは SMT の RT にある要素 $(v_{psn}, [s^j_{v_{psn}}, e^j_{v_{psn}}], j)$ によって計算でき，RT にそのような要素が存在しない場合は $e^*_{v_{psn},t} = t_c$（t_c は $e^*_{v_{psn},t}$ 計算時の時刻）と設定する．

エージェント i は時刻 t に以下の条件（Cond. 1）を満たすタスク $\tau = (\sigma^{ld}_\tau, \sigma^{ul}_\tau, \phi_\tau) \in \mathcal{T}$ を選択する．ここで v^i_c は i の現在地，$\sigma^{ld}_\tau = (v^{ld}_\tau, o^{ld}_\tau)$，$\sigma^{ul}_\tau = (v^{ul}_\tau, o^{ul}_\tau)$ とする．

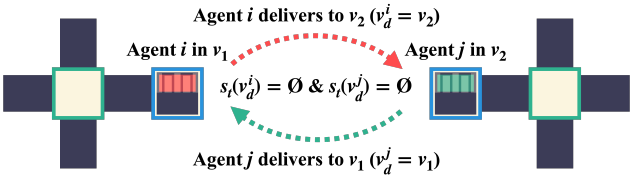


図 1: デッドロック状態（$v_1, v_2 \in V$）

Algorithm 2 Destination decision by agent i

```
1: function DecideDest(i, v_d, v_c)
2:     // v_d: v_τ^ld, v_τ^ul or park_i,
3:     // v_τ^ld: pickup, v_τ^ul: delivery, park_i: parking node
4:     if i ∈ CL then remove i from CL
5:     end if
6:     if v_d satisfies one of Cond. 2, and v_d is open then
7:         return v_d
8:     else if v_c ∈ s(v_d) then return v_c
9:     else
10:        G_t^* = G_t ∪ {v_c} // if v_c ∉ S_psn, G_t = G_t^*
11:        V'_psn(G_t^*) = {v ∈ S_psn(G_t^*) | e_v^* − t_c ≤ δ}
12:        if S = V'_psn(G_t^*) ∩ s(v_d) ≠ ∅ then
13:            v_d ← arg min_{v∈S} (e_v^* − t_c)
14:        else if S' = V'_psn(G_t^*) ∩ S^c_psn(G_t^*) ≠ ∅ then
15:            v_d ← arg min_{v∈S'} dist(v_d, v)
16:            CL = {i} ∪ CL
17:        else
18:            v_d ← park_i // i returns to its parking nodes.
19:        end if
20:    end if
21:    return v_d
22: end function
```

(1) i が駐車ノードに滞在している場合（つまり $v^i_c = park_i$），CL が空である．

(2) 積み込みノード v^{ld}_τ が *open* である（つまり RT にそのノードが endpoint として現れない），または v^{ld}_τ が空ではない関連待機可能ノードを持ち（つまり $s_t(v^{ld}_\tau) \neq \varnothing$），かつ $e^*_{v_{psn}} - t_c \leq \delta$ となるような $\exists v_{psn} \in s_t(v^{ld}_\tau)$ がある．

(3) $TEST$ に現れる v^{ul}_τ を目的地とするタプルの数より $|s_t(v^{ul}_\tau)| + 1$ が大きい．

ここで i は $e^*_{v_{psn}}$ より前に v_{psn} で滞在を開始できないため，$\delta\ (\geq 0)$ は待機ノードの予約に対する時間マージンの閾値パラメータとする．これらの条件を満たすタスクの集合を $\mathcal{T}'$ とする（2 行目）．

次に i は最も近い，すなわち G_t における $dist(v^i_c, v^{ld}_{\tau^*})$ が最小のタスク τ^* を選ぶ（4–6 行目）．ここで v^i_c は i の現在地とする．もし i がそのようなタスクを見つけられなかった場合，i は $park_i$ に戻る．しかし，i は $park_i$ に戻る途中や滞在中に，他エージェントがタスクを完了することで，いくつかの endpoint や待機ノードが open になっているかどうか時々確認できる．

3.4 待機ノードの活用

タスク選択プロセスの後，エージェント i は選択したタ

スク τ の v_τ^{ld} または v_τ^{ul} への経路生成のため，経路計画プロセスを呼び出す．しかし，これらのノードは open ではない可能性が高く，必要に応じて i は代わりに一時的な目的地への経路を生成する．したがって経路を生成する前に，i は現在時刻 t_c に次の実際の目的地を決定するために Algorithm 2 の DecideDest(i,v_d^i,v_c^i) 関数を呼び出す．ここで v_d^i は v_τ^{ld}，v_τ^{ul}，$park_i$ のいずれかを示す i の目的地で，v_c^i は endpoint，待機ノード，駐車ノードのいずれかを示す i の現在地である．

Algorithm 2 を説明する前に，$\forall i \in A$ に対して，主に endpoint への横入りを防ぐ条件（Cond. 2）を導入する．

(1) Endpoint に直接向かうための閾値 β ($> \alpha$) に対して，$dist(v_c^i, v_d^i) \leq \beta$ となる．
(2) $s_t(v_d^i)$ に向かうエージェントがいない．
(3) $v_d^i == park_i$

これらの条件のいずれかを満たせれば i は可能な限り直接 v_d^i に向かい，さもなければ v_d^i の待機ノードに向かう．なお β は横入りによって他エージェントが待機ノードで長く待たされることを防ぐ閾値でもある．

Algorithm 2 を簡単に説明する．v_d が Cond. 2 のいずれかを満たし，かつ v_d が *open* の場合，i は v_d を目的地に決定する（6–7 行目）．なお $park_i$ は常に open である．v_c が v_d の待機ノードで，かつ v_d が open でない場合は v_c を返し，もうしばらくそこに留まる（8 行目）．そうでない場合，i は一時的な目的地として v_d の待機可能ノードへの移動を試みる．したがって，i はまず RT と SST を参照して $V'_{psn}(G_t^*)$ を計算する（11 行目）．$G_t^* = G_t \cup \{v_c\}$ は G の部分グラフとする（10 行目）．他エージェントが δ 時間以内に通過する待機可能ノードがあれば，最も適切なノードを選択する（12–13 行目）（Cond. 1 の (2) 参照）．そのような待機ノードが存在しない場合，i は代わりに $\mathcal{S}^c_{psn}(G_t^*)$ の自由待機可能ノードの選択を試みる．そのようなノードを選択できた場合，i は環境の混雑によって $s(v_d)$ の要素を選択できないことを示す CL に追加される（14–16 行目）．さもなければ駐車ノードに戻る（18 行目）．なお i が待機ノードを v_d として選択・予約する場合（13，15 行目），(v_d, i) が SST の $\mathcal{S}_t$ に追加されるが，i が現在の待機ノードから離れる場合（7 行目），(v_c, i) は $\mathcal{S}_t$ から削除される．

自由待機可能ノードや駐車ノードでの一時退避（14–18 行目）は，エージェント i と j が同時に積み下ろしノードと現在地の交換を試み，かつ両ノードが関連待機可能ノードを持たない場合に起こるデッドロックの回避を可能にする．図 1 は目的地が open になるか $s_t(v_d) \neq \varnothing$ になるまで 2 つのエージェントが待機する例を示し，この状況はいずれかの目的地が変わらないと解決できない．しかし，いずれかのエージェントが他のノードを一時的な目的地に設定すると，積み下ろしノードの一つが open になり，もう一つのエージェントは移動を開始できる．エージェント i が τ で指定された endpoint の代わりに待機ノード v_{sn} に到着した後，i は SMT へのアクセス権を取得できるとき，実際の endpoint にいつ移動できるか識別するために DecideDest 関数を呼ぶ．次に i は経路探索アルゴリズムを用いて次の目的地への経路を生成して v_{sn}^i を解放し，$\mathcal{S}_t$ から対応するエントリ (v_{sn}^i, i) を削除して SMT のロックを解除，最後に v_{sn} を離れる．

SMT には同時に 1 つのエージェントしかアクセスできないため，エージェント i が待機ノード v_{sn} を予約した後に経路を生成する他エージェントは，v_{sn} の通過を禁止される．しかし，エージェント i が v_{sn} を予約する前に経路を生成した j がそこを通過する場合，i は j が v_{sn} を通過した後に v_{sn} での待機を開始する必要がある．したがって，i はどこかで待機することで適切に遅れて v_{sn} に到着しなければならず，これは他エージェントの他の遅延を起こす可能性がある．このような待ち時間の短縮のため，SBDA アルゴリズムでは，Cond. 1 (2) を導入している．

最後に提案手法が我々の well-formed MAPD インスタンスに対して完全であることを示す．SBDA では 2.2 節で前述の TP の well-formed 条件 (c) を以下のように修正する．

(c') 任意のエージェントが時刻 t に計画を生成するとき，endpoint または t における待機可能ノードの任意の 2 ノード間に，他の endpoint または t に登録された待機ノードを横切らない経路が存在する．

これまで説明したアルゴリズムは明らかに条件 (c') を常に満たす．実際，エージェント i が待機可能ノードから待機ノードを選択すると，グラフ G の構造は一時的に変更される．しかし SBDA において，別のエージェントは待機ノードや endpoint を通らずに目的地までの経路を常に見つけられる．これは，i が次のタスクや，タスクの目的地の代わりに待機ノードを選択するときに，SBDA が APF アルゴリズムを用いて $\mathcal{S}_{psn}(G_t)$ を生成するためである．さらに TP と同様に，エージェントは排他的に現在の SMT にアクセスし，順番に経路を生成する．したがって次の定理が得られる．なお証明の詳細は我々の [12] を参照されたい．

Theorem 3.6. *SBDA は全ての well-formed MAPD インスタンスを解く．*

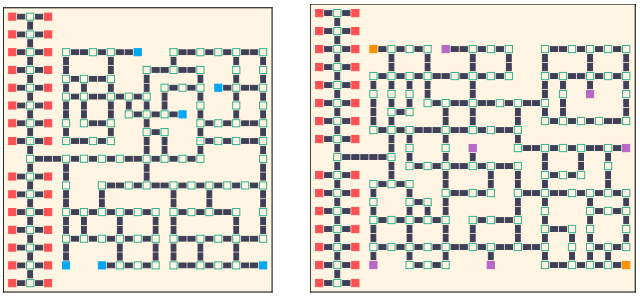
4. 実験・議論

4.1 実験設定

MAPD 問題に対する提案手法の実行性能を評価するため，2 つの異なる環境下で実験を行い，HTE をベースラインとした結果と比較した．HTE では，エージェントは積み込み・積み下ろしノードが他エージェントが現在実行中のタスクの endpoint と重ならず，積み込みノードが現在地 v_c^i に最も近いタスクを選択する．次にエージェントは積み込みノードへの経路を生成し，続いて積み込みノードから積

表 1: 実験に使用するパラメータ値

説明	パラメータ	値
エージェント数	M	2 to 30
タスク数	N	100
向き・方向の刻み幅	D	90
move の持続時間（長さ 1 あたり）	$T_{mo}(1)$	10
rotate の持続時間	$T_{ro}(D)$	20
load と *unload* の持続時間	T_{ld}, T_{ul}	20
wait の持続時間	$T_{wa}(t)$	t
待機ノード予約の時間マージン	δ	100
直接 endpoint に向かうための閾値	β	20



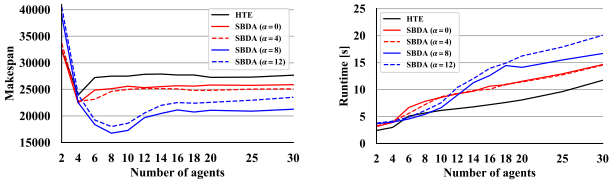
(a) 環境 1　(b) 環境 2

図 2: 赤: 駐車ノード，青: task endpoint，オレンジ: 集荷限定ノード，紫: 配送限定ノード，緑: ノード，黒: エッジ）

み下ろしノードへの経路を生成する．そのようなタスクが選択できない場合は駐車ノードに戻る．両手法の経路探索アルゴリズムとして，TP でも使用されている space-time A^* を使用した．最初の環境（環境 1）は建設現場を想定した，task endpoint が少ない（$|V_{tsk}| = 6$）迷路状の環境である（図 2a）．ノードはエッジの端と交差点に設定される．エージェントはノードでのみ自転，待機，資材の積み込み・積み下ろしができる．Task endpoint は図 2a の青い四角で示され，エージェントはそこで資材の積み込みと積み下ろしの両方を行える．図のエッジの切れ目は長さ 1 のブロックを表す．

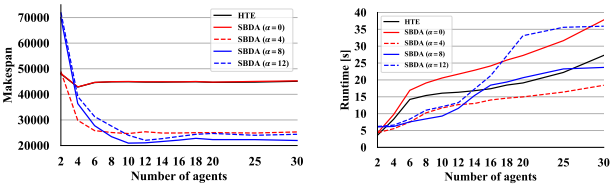
2 番目の環境（環境 2）も task endpoint が少ない（$|V_{tsk}| = 8$）迷路状の環境で，図 2b に示すように集配場所の数が偏っている．この図では，オレンジ色の四角はエージェントが資材の積み込みのみできるノード，紫色の四角は資材の積み下ろしのみできるノードである．エージェントの初期位置は，両環境ともに赤い四角で示される駐車ノードにランダムに割り当てられる．最初に青，オレンジ，紫の四角から実験設定に従って集配場所をランダムに選んで 100 個のタスクを生成し，$\mathcal{T}$ に追加する．なお環境 1 と 2 は明らかに二重連結ではない．

提案手法を評価するため，$\mathcal{T}$ の全タスクの完了に必要な時間である *makespan* と，全エージェントによる全タスクに対するタスク選択，目的地決定，経路探索の合計 CPU



(a) Makespan　(b) Runtime

図 3: SBDA と HTE の比較（環境 1）



(a) Makespan　(b) Runtime

図 4: SBDA と HTE の比較（環境 2）

時間である *runtime* を計測した．Makespan は輸送効率，runtime は計画効率を示す．その他のパラメータ値は表 1 に示す．我々の実験は 3.00-GHz Intel 8-Core Xeon E5 with 64 GB of RAM で行った．以下の実験結果は，異なるランダムシードによる 50 試行の平均値である．我々の実験のサンプル動画は `https://youtube.com/playlist?list=PLKufA_6vumDUO1_hYNWihEN4IGlTh_oZ8` 参照とする．

4.2 性能比較実験

環境 1 と 2 において，提案手法 SBDA とベースライン手法 HTE の性能を比較した．環境 1 の結果を図 3 に示す．特に，輸送効率が最も高かった $\alpha = 8$ の結果に注目する．なお SBDA のパラメータ α や Cond. 1，Cond. 2 の効果に関する詳細な議論は我々の [12] を参照されたい．

図 3a は，SBDA が HTE に比べて大幅に makespan を削減することを示す．特にエージェント数が $M = 8$ の場合，SBDA は HTE に対して makespan を約 39%削減できる．HTE では，エージェントは同じ endpoint を持つタスクを同時に実行できない．したがって，環境 1 では $|V_{tsk}| = 6$ なため，M が大きくても 3〜4 エージェントしかタスクを並列実行できない．対して SBDA では，待機ノードを有効活用することで積み込み・積み下ろしノードが実行中の他タスクと重複するタスクを複数エージェントが同時に実行できるため，タスク実行の並列性が高くなり，makespan が大幅に改善される．例外として，$M = 2$ のとき，SBDA の makespan は HTE よりも高い．これは，M が task endpoint の数 $|V_{tsk}|$ に比べて極めて小さいとき，エージェントは endpoint が重複しないタスクを容易に見つけ，待機ノードを考慮せずとも常に集配タスクを同時に実行できるためである．

図 3b を見ると，SBDA の runtime は HTE と比較してわ

ずかに増加することがわかる．HTE における唯一の CPU 使用率は，タスク選択と混雑していない環境で衝突のない経路を生成する経路計画によるものである．したがって，エージェント数が増えても runtime は大幅に増加せず，タスク選択の確認にのみ使用される．一方 SBDA では，多くのエージェントが実行するタスクを選択し，主に endpoint や待機ノードまでの経路を多数生成できるため，runtime が長くなる．しかし並列実行を考慮すると，1 エージェントあたりが費やす CPU 時間は大きくない．

環境 2 の実験結果（図 4）は，SBDA が HTE と比べて大幅に性能を改善でき，makespan が半分以下になることを示す．例えば $M = 10$ のとき，SBDA は HTE に対して makespan を約 53%削減した．これは環境 1 のときよりも大きな改善である．環境 2 には積み込みノードが 2 つしかないため，HTE では同時に作業するエージェント数が制限され，makespan が下がらない．SBDA の場合，多くのエージェントが少数の集荷位置に向かって無制限に移動するのではなく，集荷位置に近い待機ノードで自動的に待機して順番に移動することで，同時に作業するエージェント数を増やせる．さらに，Cond. 1 が作業領域へ多くのエージェントが入りすぎることを防ぎ，Cond. 2 が待機ノードで待機中のエージェントを無視した endpoint への横入りを防ぐため，過度な混雑を防ぎ，高い効率性を実現している．

5. むすび

迷路状の環境でも輸送効率を向上させるために，我々は MAPD 問題に対してグラフ理論を活用したデッドロック回避手法 SBDA を提案した．SBDA の中心となるアイデアは，エージェントが滞在しても環境の接続性が保たれる待機ノードの活用である．さらに，待機ノードはグラフ理論で用いられる articulation-point-finding アルゴリズムによって低い計算コストで識別でき，それによってリアルタイムに発見できる．SBDA は任意の有限時間待つことが保証される待機ノードを効果的に活用することで，well-formed MAPD インスタンスに対する完全性を保証する．我々が想定するアプリケーションに基づき，制限された迷路状の環境において，提案手法を有名な従来手法である HTE と比較評価した．実験結果より，SBDA が従来手法の性能を大幅に上回ることを実証した．

今後は SBDA の実アプリケーションに対する柔軟性や有用性向上のため，待機ノードの選択に用いるパラメータ α の適切な値をグラフ構造から決定する方法を検討する．

謝辞 本研究は JSPS 科研費 17KT0044 と 20H04245 の助成を受けたものです．

参考文献

[1] Čáp, M., Vokřínek, J. and Kleiner, A.: Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures, *Twenty-Fifth International Conference on Automated Planning and Scheduling*, pp. 324–332 (2015).

[2] Krakowczyk, D., Wolff, J., Ciobanu, A., Meyer, D. J. and Hrabia, C.-E.: Developing a Distributed Drone Delivery System with a Hybrid Behavior Planning System, *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, Springer, pp. 107–114 (2018).

[3] Li, J., Tinka, A., Kiesel, S., Durham, J. W., Kumar, T. S. and Koenig, S.: Lifelong Multi-Agent Path Finding in Large-Scale Warehouses, *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1898–1900 (2020).

[4] Liu, M., Ma, H., Li, J. and Koenig, S.: Task and Path Planning for Multi-Agent Pickup and Delivery, *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1152–1160 (2019).

[5] Ma, H., Li, J., Kumar, T. and Koenig, S.: Lifelong multi-agent path finding for online pickup and delivery tasks, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 837–845 (2017).

[6] Ma, H., Tovey, C., Sharon, G., Kumar, T. S. and Koenig, S.: Multi-agent path finding with payload transfers and the package-exchange robot-routing problem, *Thirtieth AAAI Conference on Artificial Intelligence* (2016).

[7] Nuutila, E. and Soisalon-Soininen, E.: On finding the strongly connected components in a directed graph, *Information Processing Letters*, Vol. 49, No. 1, pp. 9–14 (online), DOI: https://doi.org/10.1016/0020-0190(94)90047-7 (1994).

[8] Okumura, K., Machida, M., Défago, X. and Tamura, Y.: Priority Inheritance with Backtracking for Iterative Multi-agent Path Finding, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, pp. 535–542 (online), DOI: 10.24963/ijcai.2019/76 (2019).

[9] Tarjan, R.: Depth-first search and linear graph algorithms, *SIAM journal on computing*, Vol. 1, No. 2, pp. 146–160 (online), DOI: https://doi.org/10.1137/0201010 (1972).

[10] Wurman, P. R., D'Andrea, R. and Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses, *AI magazine*, Vol. 29, No. 1, pp. 9–9 (online), DOI: 10.1609/aimag.v29i1.2082 (2008).

[11] Yamauchi, T., Miyashita, Y. and Sugawara, T.: Path and Action Planning in Non-uniform Environments for Multi-agent Pickup and Delivery Tasks, *European Conference on Multi-Agent Systems*, Springer, pp. 37–54 (online), DOI: 10.1007/978-3-030-82254-5_3 (2021).

[12] Yamauchi, T., Miyashita, Y. and Sugawara, T.: Standby-Based Deadlock Avoidance Method for Multi-Agent Pickup and Delivery Tasks, *arXiv preprint arXiv:2201.06014* (2022).

[13] Yoshida, N., Noda, I. and Sugawara, T.: Multi-agent Service Area Adaptation for Ride-Sharing Using Deep Reinforcement Learning, *International Conference on Practical Applications of Agents and Multi-Agent Systems*, Springer, pp. 363–375 (online), DOI: https://doi.org/10.1007/978-3-030-49778-1_29 (2020).