Vol.2021-ICS-204 No.2
2021/9/15

不均一環境におけるマルチエージェント搬送問題のための効率的な経路・動作計画アルゴリズムの提案

山内 智貴[1,a)]　宮下 裕貴[1,b)]　菅原 俊治[1,c)]

概要：複数エージェントが衝突せずに，ある保管場所から各目的地まで繰り返し資材を運ぶ *multi-agent pickup and delivery*（MAPD）問題が注目されているが，従来の MAPD アルゴリズムは特別に設計された環境を前提とすることで，制約条件を考慮しない単純で均一のモデルを使用する．したがってこのような従来アルゴリズムは，より複雑で制限された環境でエージェントが移動する必要がある現実的なアプリケーションに適用できない．例えば災害現場や建設現場では，エージェントや運搬資材のサイズ，通路幅によってエージェントの経路や向きは厳しく制限される．そこで本研究ではまず，不均一環境に適用するために MAPD 問題を拡張した N-MAPD 問題を定式化する．次に，環境制約を満たす衝突のない経路を効率的に生成するため，N-MAPD アルゴリズムである *path and action planning with orientation*（PAPO）を提案する．PAPO は我々の N-MAPD 問題の定式化において，エージェント・資材・ノードのサイズ，通路幅を考慮して，進行方向だけでなく自転のコストやタイミングと同様にエージェントの向きも考慮するアルゴリズムである．我々はシミュレーション環境を用いて PAPO の性能を実験的に評価し，不均一環境において，最適ではないが許容可能な経路を効率的に生成できることを示した．

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，不均一環境

1. はじめに

近年，実世界の複雑で膨大なタスクに対する *multi-agent systems*（MAS）の活用が注目されている．例えば，自動倉庫での自動搬送ロボット [16]，空港での自律的な航空機牽引車 [11]，複数ドローンによる配送システム [4] などが挙げられる．しかし単純にエージェント数を増やすだけでは，タスク割当の冗長性や，衝突などのエージェント間のリソース競合の発生が原因でかえって非効率にもなる．したがって実際に使用するには，相互の悪影響を回避して，性能を向上させる協力・協調行動が必要不可欠である．特に我々の対象アプリケーションである，重くて大きなサイズの資材を運ぶロボットをエージェントとした，制限環境下での集配システムでは衝突回避は必須である．

このような問題は複数エージェントに複数タスクを割り当てる *multi-agent pickup and delivery*（MAPD）問題として定式化される．各タスクにおいて，エージェントは資材の保管場所に移動して指定された資材を積み上げ，必要な場所に配送して資材を積み下ろす必要がある．MAPD 問題はプランニングの観点では，複数エージェントが始点と終点の間で衝突のない最適または許容可能な経路を生成する *multi-agent path-finding*（MAPF）の繰り返しと考えられる．これは MAPD 問題では要求されるタスク数が多く，エージェントは結果的に MAPF 問題を 1 つずつ繰り返し，衝突の可能性を回避する必要があるためである．なお，MAPF 問題は最適解の導出が一般に NP 困難であり [10]，MAPD 問題は更に時間がかかる．それでも MAPD 問題の許容可能な解を効率的に得る必要がある．

MAPF/MAPD 問題に着目した研究は数多くあり [7], [9], [14]，その成果は実際のアプリケーションで使用されている．しかしこれらの研究のアプリケーションは，通常，均一な通路幅を持つグリッドとして記述され，エージェントや運搬資材のサイズを無視し，自転や進行方向などのエージェントの操作に関する制約がない特別に設計された環境を前提とする．一方，我々の対象アプリケーションは，建設現場でのピッカーを前方に備えたフォークリフト型の自律エージェントによる運搬や，災害現場での自律アームロボットによる救助など，より複雑で独自の制約を持つ．例えば，建設現場では様々な幅の通路がある．更に，エージェントは自身よりも広い幅の，重くて大きな資材を運ぶことが多い．障害物によりエージェントが自転できない場

[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

所もある．このことは，環境中の場所にはそれぞれ制約があり，エージェントは運搬中の資材の有無で異なる経路を取るべきであることを示唆する．このような制約により，単純で均一な環境の想定は現実的ではない．更に，前日にはなかった新しい壁が作られたり，翌日には通路に新しい資材が設置・除去されたりするため，通路幅やトポロジーは容易に変化する．これは膨大な量の学習データを必要とする学習手法が望ましくないことを示す．

本研究では，まず上記のような複雑な状況を想定してMAPD 問題を拡張した *multi-agent pickup and delivery in non-uniform environment*（N-MAPD）問題を定式化する．例えば建設現場での集配問題では，ノードサイズや通路幅の関係で，エージェントは特定のノードや通路での活動を禁止されることがある．同様に，エージェントは自身の（運搬資材の幅やサイズを含む）サイズを考慮する必要があるが，本研究では，エージェントは自転せずに上下左右に移動できる．このような環境では，衝突回避のための制約だけでなく，移動のための追加の制約を考慮する必要がある．エージェントやエッジ，ノードのサイズに関する制約を考慮しない単純な最短経路は，例えば経路のエッジが非常に狭く，資材を持ったエージェントが通過するには向きを変える必要があるが，追加の自転はかなりの時間を消費する可能性があるため，許容できない場合がある．

そこで我々は，このような環境で衝突のない経路や動作列を生成するために，N-MAPD アルゴリズムである *path and action planning with orientation*（PAPO）を提案する．PAPO は我々の N-MAPD 定式化において，速度や進行方向だけでなく，エージェントの向きや時間コスト（持続時間），自転のタイミングなどを考慮したアルゴリズムである． 更に，環境制約を満たすためには待機（同期），迂回，行動順序の変更などの競合回避方法をエージェントが適切に判断する必要があるため，競合解消のための高度なプロセスを考慮する．続いて様々な実験設定の下，我々のシミュレーション環境を用いてナイーブな集中型手法の結果と比較し，PAPO の性能を実験的に評価する．我々の提案アルゴリズムは，最適ではないが許容可能な経路を生成できることを示す．また，エージェント数が全体の性能に与える影響についても調査する．我々の定式化は，マルチエージェント災害救助問題のように，物理的制約を考慮する必要がある他の状況でも使用できる．なお，本稿は [17] を簡略化したものである．

2. 関連研究

MAPF/MAPD 問題は様々な観点から研究される [13]．例えばエージェント間の協調構造に着目すると，中央集権型と分散型の 2 種類に分けられる．前者には，例えば MAPF の *conflict-based search algorithm*（CBS）[14] やその拡張 [2] がある．これは各エージェントが独立して経路を決定する *low-level search* と，エージェント間の競合を解消して動作列を生成する *high-level search* からなる 2 段階の探索である．いくつかの分散型手法は，ある制限の下で完全性を保証する [9], [12]．例えば [9] は MAPD において，エージェントが自分にタスクを割り当て，同期した共有メモリブロックであるトークンの情報を用いて経路を生成する *token passing*（TP）を提案する．しかし，これらの研究はエージェントのサイズや通路幅，移動時間，個々のエージェントの速度などの制約を無視した単純な環境を前提とするため，実世界での応用には限界がある．

これに対して [3], [6], [8] は，エージェントの自転，サイズ，速度差をモデル化した．例えば [8] はエージェントの進行方向や自転を考慮するために，*safe interval path planning with reservation table*（SIPPwRT）と呼ばれる新しい組合せ探索アルゴリズムを用いて TP を拡張した TP-SIPPwRT を提案した．しかし，これも通路幅や距離が均一な，特別に設計された環境に基づく．したがって，これを我々の対象環境に直接適用することはほぼ不可能である．軌道計画の分野には計画時の運動学的制約に着目した研究 [1], [5] があるが，我々の研究は，雑然とした狭い環境で MAPD の反復タスクをより効率的に完了することを目的とする点で異なる．

また，エージェントの状態への向きの追加や環境制約の考慮によって，従来アルゴリズムを我々の N-MAPD 問題に適用できる．従来アルゴリズムの経路計画は空間次元と時間次元の 2 次元探索空間を持つため，様々な通路幅やエージェントサイズに関わる方位次元を加えると，N-MAPD 問題の探索空間は非常に大きくなる．そのため，この 3 次元探索空間において従来アルゴリズムのように最適解を得るためにナイーブ探索を行うと，計算コストが増大する．我々の知る限り，離散グラフ上の経路計画において，空間，エージェント，運搬資材の形状やサイズに起因する行動制約の下で競合解消を伴う研究は存在しない．

3. 問題設定と背景

通路（エッジ）幅，エージェントや資材のサイズ，エージェントの持続時間などを導入して，従来の MAPD 問題を拡張した N-MAPD 問題を定義する．

3.1 N-MAPD

N-MAPD 問題はエージェント集合 $A = \{1, \ldots, M\}$，タスク集合 $\mathcal{T} = \{\tau_1, \ldots, \tau_N\}$，$x$ 軸と y 軸で記述される 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 $G = (V, E)$ で構成する．ノード $v \in V$ は環境内の位置に対応し，エッジ $(u, v) \in E$ $(u, v \in V)$ は位置 u と v の間でエージェントが移動可能な通路に対応する．ノード v は幅 W_v と長さ L_v を持つ．(u, v) の幅と距離はそれぞれ W_{uv} と $\mathrm{dist}(u, v)$ で表し，距離はユークリッド空間における u

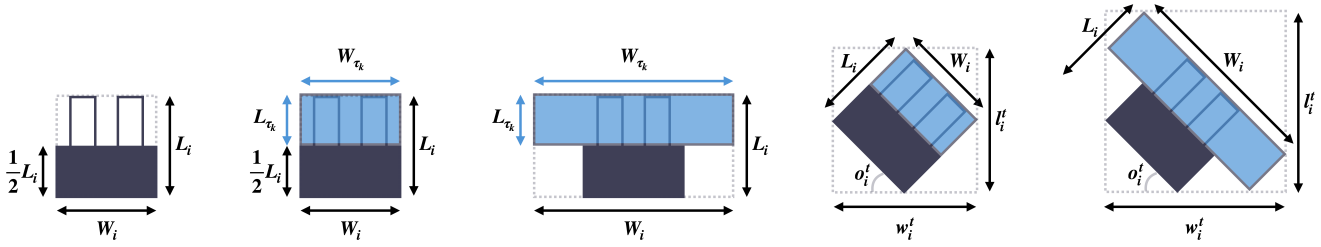


図 1: エージェントの（運搬資材を含む）サイズと向きの定義

と v の中心間の長さとして定義する．我々はエージェントを，ピッカーを前方に備えたフォークリフト型の，重い資材を運搬するロボットと仮定する．資材はラックベース上にあり，エージェントは特定のノードで特定の方向に向かってピッカーを使うことで積み上げ（load）または積み下ろし（unload）できる．離散時間 $t \in \mathbb{Z}^+$（$\mathbb{Z}^+$ は正の整数集合）を導入し，隣接地への移動や自転，待機，資材の積み上げ・積み下ろしなどのエージェントの動作には一定時間が必要とする．図 2 に環境の例を示す．

エージェント $i \in A$ は，幅 W_i と長さ L_i で指定されるサイズを持つ．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．例えば $D = 90$ とすると，向き・方向は $o_i^t, d_i^t = 0, 90, 180, 270$ の 4 つになる． D は環境構造に応じて任意の数を持てるが，単純化のため $D = 90$ と仮定する．ここでは G の北側の向き・方向を 0，つまり $o_i^t = 0$, $d_i^t = 0$ と定義する．また，可能な向きの集合を，$D = 90$ の場合は $\mathcal{D} = \{0, 90, 180, 270\}$ と表記する．時刻 t における i の x 軸の長さ w_i^t と y 軸の長さ l_i^t は以下で計算される．

$$w_i^t = |L_i \sin o_i^t| + |W_i \cos o_i^t|$$
$$l_i^t = |W_i \sin o_i^t| + |L_i \cos o_i^t|$$

タスク τ_k に関連する資材は対応するサイズを持ち，幅と長さを W_{τ_k} と L_{τ_k} とする．エージェント i がそれを積み上げる時，i のサイズは一時的に以下のように変化すると仮定する．

$$W_i = \max(W_{\tau_k}, W_i)$$
$$L_i = \max(L_{\tau_k} + \gamma L_i, L_i)$$

ここで γ $(0 \leq \gamma \leq 1)$ は，エージェントの車体とフォーク部分の長さの比とする．図 1 は $\gamma = 0.5$ のときの，資材を持つ/持たないエージェントのサイズと向きの例を示す．

エージェントは，通路幅とエージェントの長さ・幅を考慮する制約を考慮して，環境の中を移動する必要がある．通路やノード，エージェントの幅やサイズなど，環境で定義される制約を環境制約と呼ぶ．

エージェントは *move*，*rotate*，*wait*，*load*，*unload* の動作を実行できる．u と v の間の移動距離 $l = \mathrm{dist}(u, v)$，自転角度 $\theta \in \mathbb{Z}$，待ち時間 t を用いて，*move*，*rotate*，*wait*，*load*，*unload* の持続時間をそれぞれ $T_{mo}(l)$, $T_{ro}(\theta)$, $T_{wa}(t)$, T_{ld}, T_{ul} と表記する．例えば $l = 1$ の場合，我々の実験設定では $T_{mo}(1) = 10$ となる．時刻 t に i が v にいるとする．*move* では，エッジに十分な幅があれば，i は向き o_i^t を変えずにエッジ (u, v) に沿って u に移動する．*rotate* では i は v に留まり，ノードサイズが十分であれば，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 $o_i^{t+T_{ro}(D)} = o_i^t \pm D$ となる．エージェント i は，$t = 0$ の出発地である固有の駐車位置 $park_i \in V$ [7] を持っており，エージェントは実行タスクがない限りそこに戻って滞在する．駐車位置は図 2 の赤い四角で示される．

タスク τ_j はタプル $\tau_j = (\nu_{\tau_j}^{ld}, \nu_{\tau_j}^{ul}, W_{\tau_j}, L_{\tau_j}, \phi_{\tau_j})$ によって指定される．ここで $\nu_{\tau_j}^{ld} = (v_{\tau_j}^{ld}, o_{\tau_j}^{ld})$ $(\in V \times \mathcal{D})$ は資材 ϕ_{τ_j} を積み上げる位置と向きを，$\nu_{\tau_j}^{ul} = (v_{\tau_j}^{ul}, o_{\tau_j}^{ul})$ $(\in V \times \mathcal{D})$ は ϕ_{τ_j} を積み下ろす位置と向きを，W_{τ_j} と L_{τ_j} はそれぞれ ϕ_{τ_j} の幅と長さを示す．エージェントが資材の積み上げ・積み下ろしをする際には，ピッカーの向きや資材の形状を考慮して，特定の方向を向く必要がある．エージェントは衝突や環境制約の違反なく，$\mathcal{T}$ 内の全タスクを完了することが求められる．i は $\mathcal{T}$ 内の全タスクを完了した時，充電のため $park_i$ に戻る．

3.2 Well-Formed N-MAPD

全 MAPD インスタンスが解けるわけではないが，*well-formed* MAPD インスタンスは常に解ける [9]．MAPD インスタンスは，(a) タスク数が有限であり，(b) 各エージェントの駐車位置がタスクの全ての集配位置と異なり，(c) 任意の 2 点のスタート/ゴールの間に，他エージェントのスタート/ゴールを横切らない経路が存在する場合に限り，well-formed である．N-MAPD の環境制約を反映するため，条件 (c) を次のように修正する．(c') 任意の 2 点のスタート/ゴールの間に，他エージェントのスタート/ゴールを横切らない実行可能な経路が存在する．ここで実行可能な経路とは，環境制約を考慮した MAPF インスタンスに解が存在することを意味する．Well-formed (N-)MAPD インスタンスでは，エージェントは他エージェントとの衝突回避のため，必要なだけ駐車位置に戻って滞在できる．この行動によって，過度に混雑した環境におけるエージェント数を減らせる．現実の MAPD インスタンスの多くは

建設現場のものも含めて well-formed なため，well-formed MAPD インスタンスは全 MAPD インスタンスの現実的なサブクラスだが，N-MAPD の条件 (c') に関する議論が必要である．これについては後述する．

4. Path and Action Planning with Orientation

我々は N-MAPD 問題に対して，衝突のない経路とそれに続く計画，すなわち不均一環境で目的地に到達するための動作列を生成する PAPO を提案する．この提案手法では，エージェントは競合の検出・解消のために *synchronized block of information*（SBI）を用いる．SBI は *reservation table*（RT）[15] と *task execution status table*（TEST）で構成される．TEST はタプル (τ, v, i) の集合で，ここで τ は i が現在実行中のタスク，v は τ で指定された積み上げまたは積み下ろしノードとする．したがって，1 タスクへのエージェント割当につき，2 つのタプルが TEST に格納される．RT と TEST の詳細は後述する．SBI は中央のメモリ領域に格納され，この領域には同時に 1 エージェントしかアクセスできない．同期した共有メモリが性能のボトルネックになることはよくあるが，ここでのロボットの動きは排他制御によるオーバーヘッド時間と比べて遅い．そのため，現実的なエージェント数（例えば 100 エージェント以下）では，このような共有メモリの排他制御はエージェントの行動に影響を与えないと仮定する．PAPO アルゴリズムは 2 段階構造を持つ．*two-stage action planning*（TSAP）では，エージェントはまず目的地までの多数の最短経路を生成し，次に各経路に沿った動作列候補の集合を構築する．そして *conflict resolution with candidate action sequences*（CRCAS）では，エージェントは SBI 内の他エージェントの承認済み計画を参照して競合を検出・解消し，衝突のない（承認された）動作列を生成する．

4.1 Two-Stage Action Planning (TSAP)

TSAP の第 1 段階では，エージェント $i \in A$ は現在地 v_s^i から目的地 v_d^i までの上位 $N_K \in \mathbb{Z}^+$ 個の最短経路を生成する．v_d^i は通常，タスクの処理状況に応じて，積み上げ，積み下ろし，または駐車位置である．経路は隣り合うノードの任意のペアが E のエッジであるノード配列 $r = \{v_0(= v_s^i), v_1, \dots\}$ で定義され，r の距離は $\sum_{l=1}^{|r|} \mathrm{dist}(v_{l-1}, v_l)$ である．上位 N_K 個の最短経路を得るアルゴリズムはいくつかあるが，ここではダイクストラ法を用いた Yen's algorithm [18] を使用する．この段階では，幅やサイズの制約は考慮せず，$G = (V, E)$ のトポロジー構造のみを使用する．

第 2 段階では，第 1 段階で生成された経路 $\forall r_k \in \{r_1, \dots, r_{N_K}\}$ とパラメータ $N_P \in \mathbb{Z}^+$ に対して，i は環境制約を違反せずに経路 r_k に沿って移動するために，*wait* を使わずにコスト最小上位 N_P 個の動作列を構築する．ここでコストは動作列完了までの持続時間を示す．第 2 段階について詳しく説明する．まず経路 r_k に沿った動作列構築のため，経路 r_k に対して，G からエージェント i の重み付き状態グラフ $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$（以下添え字は省略）を生成する．ノード $\nu = (v, o) \in \mathcal{V}$ $(\subset V \times \mathcal{D})$ は i の位置と向きに対応しており，$\mathcal{V}$ の要素を状態ノードと呼ぶ．エッジ $(\mu, \nu) \in \mathcal{E}$ $(\mu, \nu \in \mathcal{V})$ は，μ から ν に遷移するための *move* または *rotate* に対応する．エッジの重み $\omega(\mu, \nu)$ は対応する動作に必要な時間として定義する．i が状態 ν に到達する時刻 t_ν を用いて探索空間を (ν, t_ν) で示し，コスト最小上位 N_P 個の動作列を生成するために A^* 探索を用いた Yen's algorithm を適用する．ここでヒューリスティック関数 h は，現在地と v_d^i の距離 l と，現在の向きと v_d^i で必要な向きの差 θ を用いて以下のように定義する．

$$h(l, \theta) = T_{mo}(l) + T_{ro}(\theta) \tag{1}$$

関数 h は動作や環境の制約を一切考慮しないため，明らかに許容的である．v_d^i は資材の積み上げまたは積み下ろし位置なため，v_d^i で必要な向きは通常，i のタスクで決まる．

第 2 段階では，エージェント i は通路幅や運搬資材と自身のサイズを考慮して，環境制約違反になる状態ノードを剪定できる．t_1 に i の状態が $\nu = (v, o)$ で，(ν, μ) によって $t_2 = t_1 + \omega(\nu, \mu)$ に $\mu = (u, o')$ への状態遷移を予定しているとする．ここで $w_i^{t_2} > W_u$ または $l_i^{t_2} > L_u$ の場合はノードとエージェントサイズの制約違反なため，μ は剪定される．同様に *move* の (ν, μ)（つまり $o = o'$ かつ $(u, v) \in E$）は，向き o と方向 d によって $W_{uv} < |l_i^{t_1} \sin d_i^{t_1}| + |w_i^{t_1} \cos d_i^{t_1}|$ を満たす場合は不可能である．したがって ν の後の μ も剪定される．最後に (ν, μ) が *rotate*（つまり $v = u$ だが $o \neq o'$）の場合，対応する動作が不可能になることがある．自転制約違反，すなわち v のサイズ，W_v と L_v（両方またはどちらか）が不十分な場合，μ も剪定される．この状況は W_v，L_v と，t_1，t_2 間の w_i^t，l_i^t の比較で確認できる．これらの値は，例えばエージェントの形状が正方形の場合，i の向きが $45, 135, 225, 315$ の時に最大となる（図 1 参照）．

TSAP の後，総持続時間で昇順ソートされた，最大 $N_K \cdot N_P$ 個の順序付き動作列集合 $P_i = \{p_1, \dots, p_{N_K \cdot N_P}\}$ が生成される．その先頭要素，すなわち最小コスト計画は明らかに最良候補だが，他エージェントの計画との競合により選択されない可能性がある．これは次の段階の CRCAS で検討される．結果として計画 p_i は対応する経路 r に沿って生成されるため，この関係を $r = r(p_i)$ と表記する．

4.2 Conflict Resolution of Candidate Action Sequences (CRCAS)

CRCAS では，エージェント i は P_i 先頭の計画を選択し，それと他エージェントの承認済み計画との間の競合検出を試みる．この競合検出は SBI へのアクセスで実現する．次

Algorithm 1 Conflict Resolution part of PAPO

```
1: function CRCAS(P_i)
2:     // P_i is generated by TSAP(N_K, N_P).
3:     C_max ← Maximal duration of the plan in P_i
4:     while true do
5:         if P_i = ∅ then return false
6:         end if
7:         P_i is sorted by duration
8:         p_1 ← the first (shortest duration) plan in P_i
9:         C_1 ← duration of p_1 // Shortest duration in P_i
10:        c_1 ← (⟨i, j⟩[t'_s, t'_e], v') // The first conflict in p_1 by
    comparing with the entries in RT.
11:        if c_1 = null then return p_1
12:        end if
13:        C ← all conflicts occurring at v' // so c_1 ∈ C
14:        c_f ← final element (⟨i, k⟩, [t_s, t_e], v) in C after sorted
    by occurrence order.
15:        u ← e^k_v − s^i_v + 1, C_1 = C_1 + u
16:        if C_1 ≥ C_max + β then // β: tolerance parameter.
17:            P_i ← P_i \ {p_1} // abandon p_1
18:        end if
19:        wait(u) is inserted in p_1 before reaching v with the
    modification strategy.
20:        p_1 in P_i is replaced to the modified p_1 if p_1 ∈ P_i.
21:    end while
22: end function
```

に競合回避のため計画を修正し，修正後の計画に置き換える（P_i はソートされる）．この修正の基本方針では，承認済み計画は修正しない．P_i 先頭要素が衝突しない場合，それが CRCAS の出力となる．したがって，SBI の RT へ関連する予約データが追加され，実行が承認される．RT の構造を以下に説明する．

我々は複数エージェントが同一ノード $v \in V$ を同時に占有する状態を競合と定義する．i が時刻 s^i_v に v から隣接ノード u に移動を始め，v と u に対する i の占有区間をそれぞれ $[s^i_v, e^i_v]$，$[s^i_u, e^i_u]$ と示す場合，$e^i_v = s^i_v + T_{mo}(l)/2$，$s^i_u = e^i_v$ とする．ここで $l = \mathrm{dist}(v, u)$ とする．同様に，i が時刻 s^i_v に v で *rotate*，*wait*，*load*，*unload* を始める場合，対応動作の v への占有区間は $[s^i_v, e^i_v = s^i_v + T_*]$ で表せる．ここで T_* は動作の持続時間である．更に，エージェントは物理的なサイズを持つため，安全のためにこれらの区間に一定のマージン $\lambda \geq 0$ を加える．そのため，例えば $s^i_v \leftarrow s^i_v - \lambda$，$e^i_v \leftarrow e^i_v + \lambda$ となる．これらの計算から，計画 $p_k \in P_i$ より以下に示す占有リストを生成できる．

$$((v_0, [s_{v_0}, e_{v_0}], i), (v_1, [s_{v_1}, e_{v_1}], i), \ldots, (v^i_d, [s_{v^i_d}, e_{v^i_d}], i))$$

ここで $r(p_k) = \{v_0, v_1, \ldots, v^i_d\}$ はソート済み集合とする．

2 台のエージェント $i, j \in A$ によるノード v に対する 2 つの占有区間が交差 $[t_s, t_e]$ を持つとき，競合が発生し，それをタプル $c = (\langle i, j\rangle, [t_s, t_e], v)$ で表す．他エージェントの承認済み計画の占有リストは SBI の RT に格納される．エージェント i の計画は，同一ノード v で別のエージェント k の承認済み計画と別の競合を起こす可能性がある．しかし，SBI に格納されている計画は承認済みなため，競合の交差は 2 エージェント間でのみ現れる．したがって，2 つの競合 $c_1 = (\langle i, j\rangle, [t^1_s, t^1_e], v)$ と $c_2 = (\langle i, k\rangle, [t^2_s, t^2_e], v)$ が存在する場合，$[t^1_s, t^1_e] \cap [t^2_s, t^2_e] = \varnothing$ となる．

RT は，承認済み計画の占有リストの要素かつ期限が切れていない $(v, [s^i_v, e^i_v], i)$ の集合である．したがって，$e^i_v < t_c$（t_c は現在時刻）のとき，その要素は RT から削除される．エージェント i の計画 p が承認されたとき，i は占有リストの全要素を RT に登録する．

CRCAS アルゴリズムの全体的な流れを Algorithm 1 の疑似コードを用いて説明する．関数実行中，エージェントは SBI の RT に排他的にアクセス可能と仮定する．エージェント i は TSAP で P_i を生成したとき，まず P_i の先頭要素 p_1 に対して占有リストを計算する．次に訪問ノード順 $r(p_1) = \{v_1, v_2, \cdots, \}$ に従って，i は RT から最初の要素が v_l のリストを検索し，これらのリストを比較して競合検出を試みる．競合がなければ p_1 が CRCAS の結果となり，承認済み計画として RT に登録される（11 行目）．ここで c を訪問順に v で検出された最初の競合とする．c は $c = (\langle i, j\rangle, [t^c_s, t^c_e], v)$ で表され，j は p_1 と競合する承認済み計画を持つエージェント，$[t^c_s, t^c_e]$ は両エージェントの v での滞在時間の交差である．v で別のエージェントとの別の競合も起こりうるため，v での全競合の集合を C とする．その後，i は C を発生時刻順にソートし，C の末尾要素を c_f とする（14 行目）．次に，k が v を出発した後に i が到着するように，次節で説明する計画修正戦略を用いて，i は p_1 に *wait* を挿入する（15 行目と 19 行目，ここで e^k_v は k が v を出発する時刻）．したがって，少なくとも検出された競合は回避できる．しかし，*wait* の追加により，修正された計画の持続時間が実行には大きすぎる場合，p_1 は放棄される（16 行目と 17 行目）．ここで β は選択された計画を保持するための許容パラメータである．その後 P_i は持続時間で再ソートされ，P_i の先頭要素 p_1 に競合が発生しなくなるまで i は同じプロセスを繰り返す．

このプロセスは最終的には停止する．このプロセスが永遠に続くと，計画実行のための持続時間が $C_{max} + \beta$ 以上になるため計画が P_i から削除され，最終的に P_i が空になるためである．このとき関数 CRCAS は *false* を返す．つまり，パラメータ N_K，N_P，β を使って P_i から経路を生成できない．この場合，エージェント i は緩和された条件で，すなわち N_K，N_P，β（全てまたはいずれか）の増加した値で生成した P_i を用いて CRCAS を再び呼び出す．

4.3 計画修正戦略

本研究における計画修正戦略は，計画 p_i で最初の競合が予想されるノード v' より前のどこに $wait(u)$ を挿入するか決めることである．$r(p_i) = \{v_0, \ldots, v_l(= v'), \ldots, v_n(= v^i_d)\}$ とすると，v_k $(0 \leq k \leq l-1)$ を出る直前に動作を追加で

Algorithm 2 Path and Action Planning with Orientation

```
1: function PAPO(i, v_s, v_d) // in agent i
2:     // v_s: current location, v_d: destination.
3:     while true do
4:         P_i ← TSAP(N_K, N_P)
5:         p ← CRCAS(P_i)
6:         if p ≠ false then return p
7:         end if
8:         (N_K, N_P, β) ← RelaxParam(N_K, N_P, β)
9:     end while
10: end function
```

きる．例えば v_{l-1} に追加できるが，これは別の競合を起こす可能性がある．追加された *wait* は消去されないため，別の競合を避けるために更なる *wait* を挿入して，結果的に遅延が長くなったり，過度に長い待機のために失敗したりする．逆に i が v_0 を出るときに挿入すれば，後述のタスク選択プロセスで i が現在のタスク τ を選択したときにその積み上げ・積み下ろしノードが TEST に格納済みで，また well-formed N-MAPD の条件 (c') によって他エージェントはノード v_0 を通過しないため，i は失敗なく競合を回避できる．しかし，これは他エージェントの活動をしばらくロックすることを意味し，全体の性能が低下する可能性がある．この点は更なる議論が必要だが，$wait(u)$ を v_k $(k = \max(0, l-3))$ に追加する戦略を暫定的に採用した．

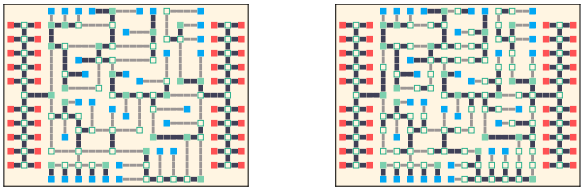
4.4 タスク選択と PAPO プロセスの流れ

エージェント i のタスク選択プロセスは $\mathcal{T} \neq \varnothing$ のときにのみ実行され，そうでなければ，i は自分の駐車位置 $park_i$ に戻る．エージェント i はこのプロセスの間，SBI の現在の TEST に独占的にアクセスする．Well-formed N-MAPD の仮定 (c') より，i は積み上げ・積み下ろしノードである v_τ^{ld} と v_τ^{ul} が現在の SBI の TEST に現れないタスク $\tau = (\nu_\tau^{ld}, \nu_\tau^{ul}, W_\tau, L_\tau, \phi_\tau)$ にのみ注目する．次に i は，A^* 探索で用いたヒューリスティック関数 h の値が最小のタスク τ^* を選択する．そのようなタスクが見つからない場合，i は $park_i$ に戻り，しばらくそこに留まる．これは他エージェントがその間にタスクを完了することで，i が再びタスク選択できる可能性があるためである．

エージェント i がタスク τ_i を選択した後，衝突のない動作列を生成するため PAPO プロセスを開始する．PAPO プロセスの疑似コードを Algorithm 2 に示す．ここで v_s は現在地，v_d はタスクの進行状況に応じて v_τ^{ld}，v_τ^{ul}，$park_i$ のいずれかとする．まず i が TSAP と CRCAS プロセスによって計画 p の生成に成功した場合，i は p に従って行動し，$(\tau, v_d, i) \in \mathcal{T} \times V \times A$ は TEST から削除される．そうでなければ，i は計画生成条件緩和のため，パラメータ N_K，N_P，β の閾値を変更する緩和関数 **RelaxParam** を呼び，計画プロセスを再実行する．i が既に何度かパラメータを緩和している場合，この関数は τ_i の実行を諦め（つまり τ_i は $\mathcal{T}$ に戻される），駐車位置に戻るため $v_d = park_i$ を設定する．この状況は現在の環境サイズと比べて，エージェント数が多すぎて環境内を動き回れない場合に起こる可能性がある．

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	1 to 40
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
安全間隔	λ	5



(a) 環境 1　(b) 環境 2

図 2: 実験環境（赤：駐車位置，青：集配位置，緑：小ノード，中空の緑：大ノード，灰：狭いエッジ，黒：広いエッジ

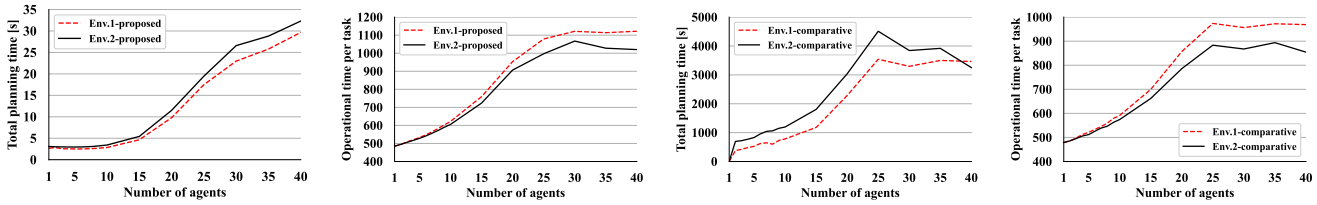
N_K，N_P，β の初期値の決め方とそれらの値の増やし方について，いくつかの戦略が考えられる．N_K，N_P，β の値が小さいと効果的な計画を生成できる可能性が高いが，同様に計画生成を失敗する可能性も高い．したがって，これらの初期値は小さく設定し，エージェントが衝突のない計画を構築できない場合に値を増やすのが良いと思われる．しかし，パラメータ値を頻繁に変更すると計画効率も低下することを考慮する必要がある．このパラメータを増加し続ければ，他エージェントによる承認済み計画の実行が進むことで RT の占有リストは時間とともに失効して制約が徐々に除去されるため，衝突のない計画が得られる（すなわち完全性が保証される）．

5. 実験・議論

5.1 実験設定

N-MAPD 問題実行のため，我々の提案手法の性能を評価した．N-MAPD タスク $\tau \in \mathcal{T}$ に応じて，エージェントは大小いずれかの資材を運ぶ．ここで小型資材 ϕ_τ の幅と長さは $W_\tau = 0.5$，$L_\tau = 0.25$，大型資材のそれらは $W_\tau = 1.0$，$L_\tau = 0.25$ とする．全エージェントのサイズは同一で，$W_i = 0.5$，$L_i = 0.5$，$\gamma = 0.5$ で指定される．大型資材と小型資材の数はともに $N/2$ とする．

続いて，2 つの異なる環境で実験を行った．環境 1 は建設現場を想定した迷路状の環境で，他の作業のためのスペー



(a) Planning time (b) Operational time (c) Planning time (d) Operational time

図 3: 環境 1 と 2 の比較

スや壁，柱などの障害物があり，そのために通路幅やノードサイズなどの環境制約がある（図 2a）．ノードは交差点やエージェントが資材を積み上げ・積み下ろしするエッジの端に設定する．エージェントはノードでのみ自転・待機できると仮定する．ノードには，図 2 にて緑色の四角で示される，幅と長さが $W_v = 1.0$，$L_v = 1.0$ の小ノード v と，緑色で中空の四角で示される $W_v = 1.5$，$L_v = 1.5$ の大ノード v の 2 種類がある．そのため，大型資材を持つエージェントは小ノードでは自転できない．同様にエッジにも，灰色のエッジで示される，幅が $W_{uv} = 0.5$ の狭いエッジ (u, v) と，黒い太いエッジで示される $W_{uv} = 1.0$ の広いエッジの 2 種類がある．エッジの切れ目はその長さを表し，1 ブロックは長さ 1 を示す．そのため，大型資材を持つエージェントは狭いエッジの通過前に自転が必要になる可能性がある．

環境 2 は環境 1 と同じだが，エッジ上にノードを追加した．これらのノードは通常，狭いエッジと広いエッジをつなぐもので，大型資材を持つエージェントは狭いエッジを通過するためにこのノードで自転が必要になる可能性がある（図 2b）．交差点での自転は他エージェントの動きを妨げる可能性があるため，これらのエッジ上のノードは待機動作を減らせる．エージェントの初期位置は駐車位置からランダムに割り当てられる．同様に青い四角からランダムに選択された積み上げ・積み下ろし位置より 100 個のタスクが最初に生成され，$\mathcal{T}$ に追加される．

提案手法を評価するため，*operational time per task*，すなわち 1 タスク完了までの平均時間（単に *operational time* と呼ぶ）と，$\mathcal{T}$ の全タスクに対する *total planning time*（単に *planning time* と呼ぶ）を測定した．Operational time は生成された計画の質の評価に使用され，planning time は計画効率を表す．これらの実験におけるその他のパラメータ値を表 1 に示す．実験は 3.00-GHz Intel 8-Core Xeon E5 with 64-GB RAM で行った．以下の実験結果は異なるランダムシードを用いた 100 試行の平均値である．

5.2 性能比較実験

環境 1 と 2 にて，提案手法とナイーブな集中型手法の性能を比較した．この比較手法は，他エージェントの計画によって予約済みのノードを一定時間，一時的な障害物と仮定することで，最適（最短）で衝突のない動作列を順番に生成する集中型プランナーである．生成された動作列は $\{move, rotate, wait, load, unload\}$ で構成され，状態はノード v，エージェントの向き o，ノード v でエージェントが向き o に到達する時刻 t のタプル $\nu = (v, o, t)$ で表される．全ての持続時間は提案手法と同じに設定される．この手法は，既に実行中の計画は修正されないと仮定することで，最適な計画を生成できる．提案手法のパラメータ初期値を $N_K = 3$，$N_P = 3$，$\beta = 100$ とし，緩和関数 **RelaxParam** は呼び出すたびに $N_K \leftarrow N_K + 1$，$N_P \leftarrow N_P$，$\beta \leftarrow \beta * 2.0$ とパラメータ値を増やす．

環境 1 と 2 における planning time と operational time を図 3 に示す．全タスク完了に非常に長時間を要したため，比較手法の結果は 10 試行の平均値となった．図 3b と 3d は，提案手法の operational time が比較手法に比べて約 10%増加することを示すが，図 3a と 3c は，いずれの環境においても提案手法の全タスクに対する planning time が比較手法に比べて大幅に短縮されることを示す．比較手法はナイーブ探索で最適解を求めるため，生成される動作列の質は高いかもしれないが，実環境で使うには planning time が非常に長くなる．また本論文では割愛したが，パラメータの変更で提案手法の operational time は改善できる．

環境 1 と 2 の結果を比較すると（図 3a，3b），環境 2 における提案手法の operational time は環境 1 のそれより小さかったが，planning time はその逆だった．これはいくつかのエッジへのノードの追加が原因である．*rotate* と *wait* では現在のノードに留まるため，そのノードが交差点の場合は他エージェントの動作を妨げる可能性が高い．しかし，自転や待機可能なノードの追加により，エージェントはそのような障害のある状態を減らせる．実際 $M = 25$ のときの提案手法では，環境 1 で検出された競合数 82135.07 に対して，環境 2 で検出された競合数は 64743.1 だった．これによりエージェントの動きがスムーズになり，operational time が $M = 25$ のとき約 82（8%）改善されたと考えられるが，環境 2 の planning time は約 2 秒（11%）だけ長くなった．なお $T_{mo}(1) = 10$ なため，この operational time の 82 の改善は 8.2 ブロック分の追加の実行長を意味する．

図 3c，3d に示すように，比較手法では $M = 25$ のとき，環境 2 の operational time が環境 1 に比べて約 90（9%）短くなったが，環境 2 の planning time は約 971 秒（27%）増加した．またエージェント数が少ないとき（例えば $M = 2$ や 3）でも planning time の大幅な増加が生じた．これはノード数の増加が計画効率に大きく影響することを意味する．対して，提案手法は planning time の増加を抑えつつ operational time を改善できる（図 3a）．以上より，提案手法はノード数の増加に十分にロバストであると言える．

6. まとめ

本論文では，不均一環境における MAPD（N-MAPD）問題に対して衝突のない計画を生成可能な計画手法 *path and action planning with orientation*（PAPO）を提案した．N-MAPD 問題は，ノード（場所），エージェント，資材のサイズや通路幅によってエージェントの動作や向きが厳しく制限される，より複雑で不均一な環境をモデル化する．提案手法をナイーブな比較手法と比較して評価した．実験結果は，提案手法が実アプリケーションにおいて最適ではないが適度な長さの許容可能な経路を生成可能で，ノード数の増加に対して十分なロバスト性を持つことを示した．

今後は輸送効率の更なる改善のため，well-formed N-MAPD の条件 (c') の緩和を検討する．条件 (c') により，エージェントは他エージェントのスタート/ゴールを横切れないため，環境内で同時にタスクを実行できるエージェント数の上限はタスクが生成されるノード数に依存する．

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] Alonso-Mora, J., Beardsley, P. and Siegwart, R.: Cooperative Collision Avoidance for Nonholonomic Robots, *IEEE Transactions on Robotics*, Vol. 34, No. 2, pp. 404–420 (online), DOI: 10.1109/TRO.2018.2793890 (2018).

[2] Bellusci, M., Basilico, N. and Amigoni, F.: Multi-Agent Path Finding in Configurable Environments, *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 159–167 (2020).

[3] Ho, F., Salta, A., Geraldes, R., Goncalves, A., Cavazza, M. and Prendinger, H.: Multi-Agent Path Finding for UAV Traffic Management, *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 131–139 (2019).

[4] 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).

[5] Li, J., Ran, M. and Xie, L.: Efficient Trajectory Planning for Multiple Non-Holonomic Mobile Robots via Prioritized Trajectory Optimization, *IEEE Robotics and Automation Letters*, Vol. 6, No. 2, pp. 405–412 (online), DOI: 10.1109/LRA.2020.3044834 (2021).

[6] Li, J., Surynek, P., Felner, A., Ma, H., Kumar, T. K. S. and Koenig, S.: Multi-Agent Path Finding for Large Agents, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, No. 01, pp. 7627–7634 (online), DOI: 10.1609/aaai.v33i01.33017627 (2019).

[7] 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).

[8] Ma, H., Hönig, W., Kumar, T. S., Ayanian, N. and Koenig, S.: Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 7651–7658 (online), DOI: 10.1609/aaai.v33i01.33017651 (2019).

[9] 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).

[10] 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).

[11] Morris, R., Pasareanu, C. S., Luckow, K., Malik, W., Ma, H., Kumar, T. S. and Koenig, S.: Planning, scheduling and monitoring for airport surface operations, *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence* (2016).

[12] 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).

[13] Salzman, O. and Stern, R.: Research Challenges and Opportunities in Multi-Agent Path Finding and Multi-Agent Pickup and Delivery Problems, *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1711–1715 (2020).

[14] Sharon, G., Stern, R., Felner, A. and Sturtevant, N. R.: Conflict-based search for optimal multi-agent pathfinding, *Artificial Intelligence*, Vol. 219, pp. 40–66 (online), DOI: 10.1016/j.artint.2014.11.006 (2015).

[15] Silver, D.: Cooperative Pathfinding, *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'05, AAAI Press, pp. 117–122 (2005).

[16] 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).

[17] 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).

[18] Yen, J. Y.: Finding the k shortest loopless paths in a network, *management Science*, Vol. 17, No. 11, pp. 712–716 (online), DOI: 10.1287/mnsc.17.11.712 (1971).