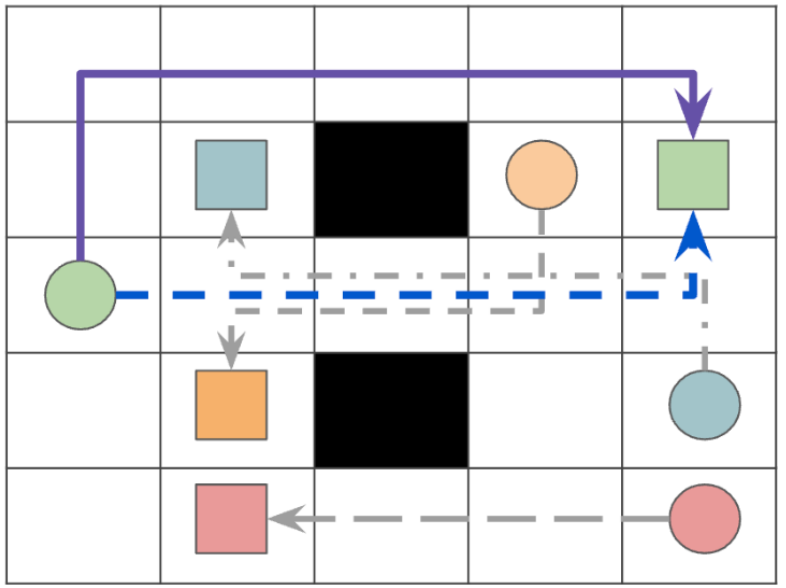
2025.05.14 理論談話会#4

Traffic Flow Optimisation for Lifelong Multi-Agent Path Finding

Chen, Z., Harabor, D., Li, J., & Stuckey, P. J. (2024). Proceedings of the AAAI Conference on Artificial Intelligence.

交通・都市・国土学研究室

M1　重村 奏画



0. 目次

01.

はじめに

1-1. 本研究の概要

目的：大規模MAPFに均衡配分を導入することで衝突回避時の渋滞を緩和し、効率を向上

MAPF (Multi-Agent Path Finding) とは？

複数エージェントがグリッドマップ上を移動し、各目的地に衝突なく到達する経路計画。

総移動距離 (SIC=Sum of Individual Costs) を最小化。

MAPFの課題

既存手法 (PIBT等) は各エージェントにとって個別に最適な経路を計画 (A*等)

↓

エージェント数が数千になると衝突回避のため渋滞が発生し、効率低下が著しい

↓

交通工学の均衡配分を導入して渋滞緩和を目指す

▲物流倉庫で自動制御される商品運搬用ロボット

◀ドローンの群飛行

1-2. 本研究の概要

新規性

- 衝突回避のため待機することで発生する渋滞を、交通工学から均衡配分の考え方を導入して軽減
  - 頂点コストと辺コスト (対向流コスト) を定義し、MAPFにおける渋滞を定式化
  - 既存のPIBT (Okumura et al. 2022) やLaCAM* (Okumura 2023) に時間非依存のGuide Pathを導入

有用性

- 従来のMAPFではエージェント数は数百が限界だったのに対し、数千での計算を可能に

信頼性

- 最大12,000エージェント、4種類のマップで検証し、ベースライン手法に対する優位性を示している
- One-shot MAPFで解の品質、Lifelong MAPFでスループットを検証

ACT-X：https://research-er.jp/projects/view/1199593

02.

背景

2-1. 関連研究① PIBT (Priority Inheritance with Back Tracking, Okumura et al. 2022)

各々が次に進みたいセルを予約し、優先度の高いエージェントから進むアルゴリズム

アルゴリズム

ゴールまでの最短距離が短くなる隣接セルを選択し、
競合が存在すれば予約

↓

エージェントの優先度順にそのセルに移動
(優先度の高い他者が移動するまで待機)

▲優先度の高い他者が通るまで待機

利点

ルールベースで個々のエージェントの最短経路のみを考慮するため高速。
Deadlockを引き起こすことがない。

欠点

One-shot MAPFでは、あるエージェントのゴールが他者の経路上にある場合、Livelockが発生。
Lifelong MAPFでは、ゴールに到着すると新しいタスクが与えられるため上記の問題は発生しづらい。
個々の最短経路のみ考慮するため渋滞しやすい。

2-2. 関連研究② LaCAM* (Lazy Constraints Addition search for MAPF, Okumura 2023)

PIBTを繰り返し用いてより総コストが小さい解を計算可能なメタアルゴリズム

アルゴリズム

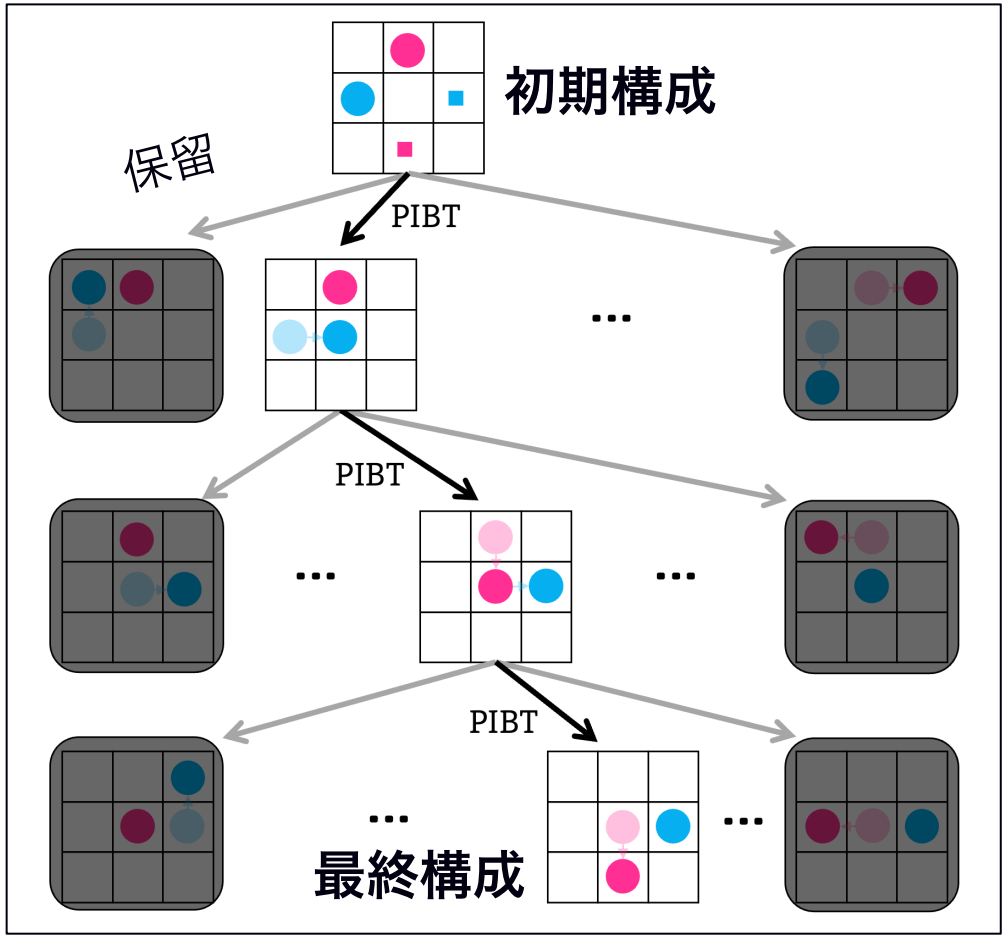
PIBTを用いて全エージェントの”構成”の遷移を探索

↓

Operator Decomposition (Standley 2010) で
部分展開を行って組み合わせ爆発を避ける

利点

イテレーションを繰り返すことで最適解に収束するため、
PIBT単体よりも解の品質が高くなる



https://kei18.github.io/note/posts/mapf-tutorial/

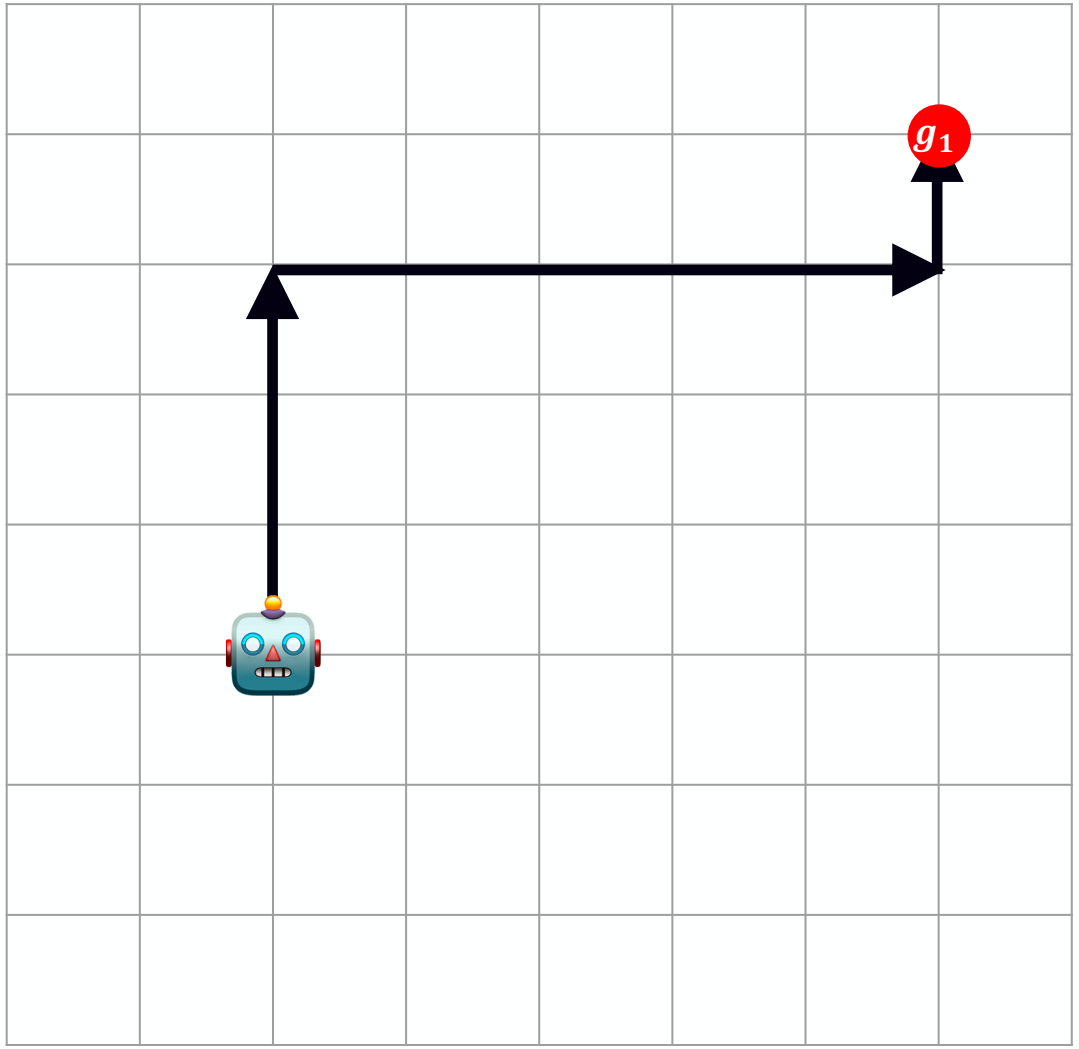
03.

問題定義

3-1. MAPFの問題定義

複数エージェントがグリッドマップ上を移動し、各目的地に衝突なく到達するための経路

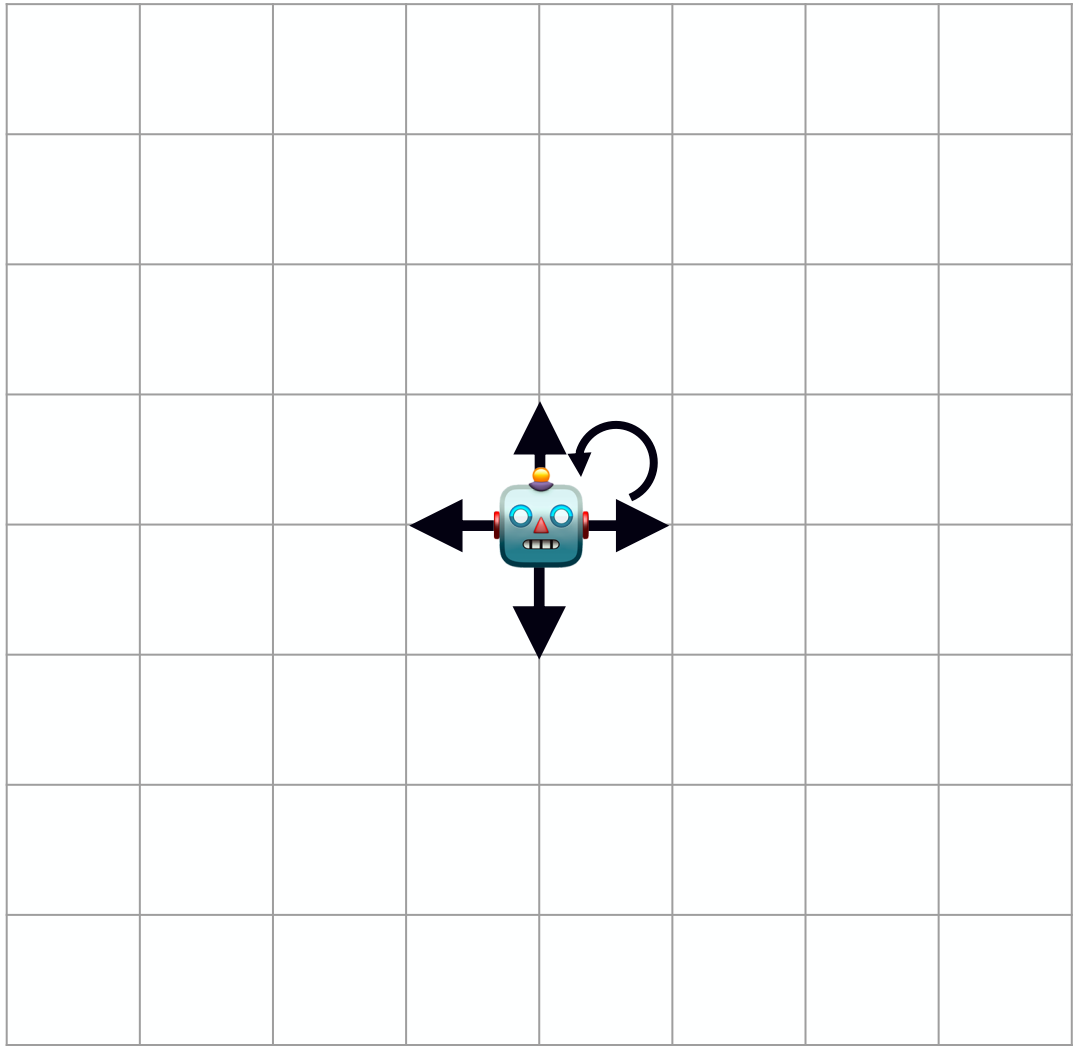
- 無向グリッドグラフ $G = (V, E), \mathbf{v} \in V, e \in E$
- エージェント集合を $\{a_1 \ldots a_k\}$ とし、
  各 a_i はスタートs_i とゴール g_i を持つ
- 行動は東西南北・待機 (すべてコスト1)



3-1. MAPFの問題定義

複数エージェントがグリッドマップ上を移動し、各目的地に衝突なく到達するための経路

- 無向グリッドグラフ $G = (V, E), \mathbf{v} \in V, e \in E$
- エージェント集合を $\{a_1 \dots a_k\}$ とし、
  各 a_i はスタートs_i とゴール g_i を持つ
- 行動は東西南北・待機 (すべてコスト1)



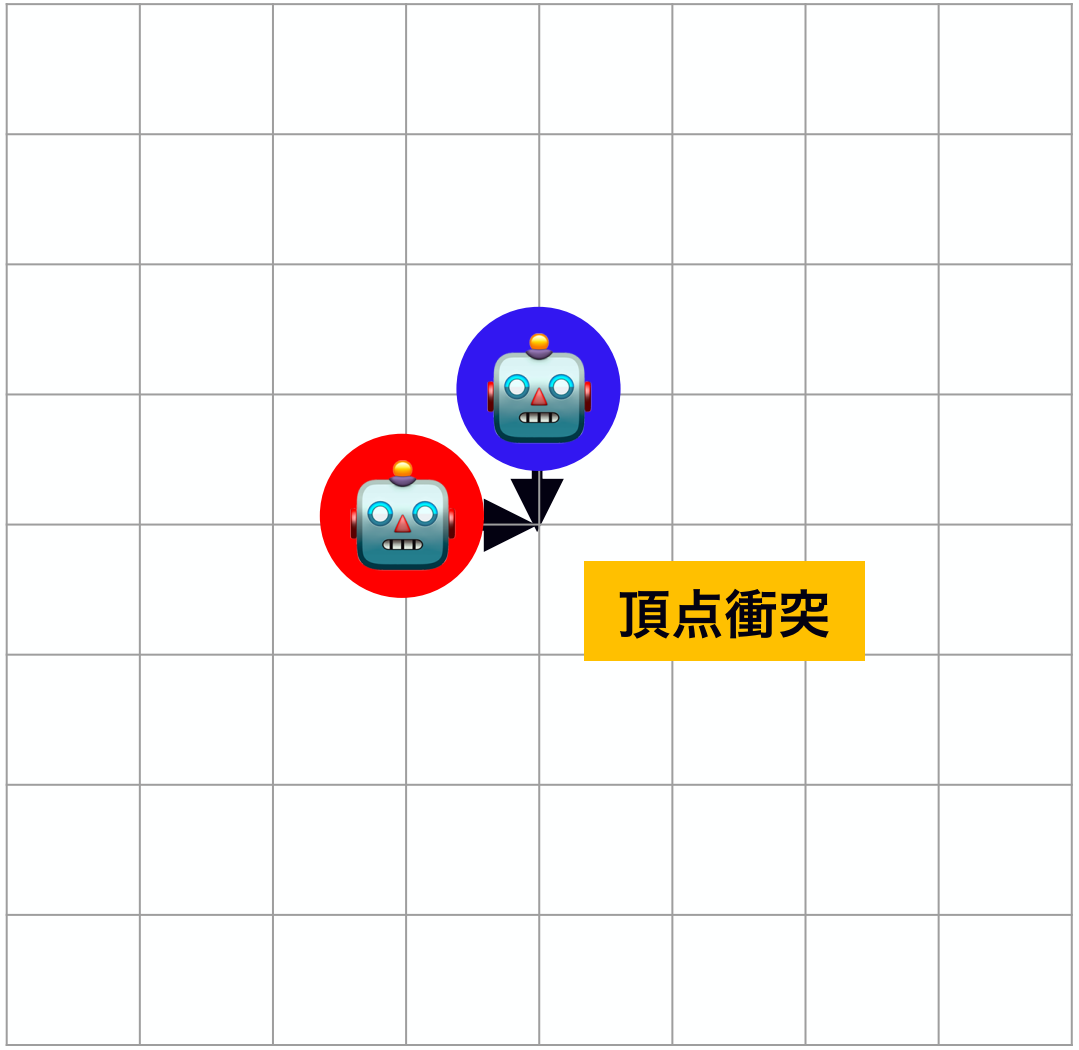
3-1. MAPFの問題定義

複数エージェントがグリッドマップ上を移動し、各目的地に衝突なく到達するための経路

- 無向グリッドグラフ $G = (V, E), \mathbf{v} \in V, e \in E$
- エージェント集合を $\{a_1 \ldots a_k\}$ とし、
  各 a_i はスタートs_i とゴール g_i を持つ
- 行動は東西南北・待機 (すべてコスト1)

① 頂点衝突 $\langle a_i, a_j, v, t \rangle$：同時に同じ頂点に存在

② 辺衝突 $\langle a_i, a_j, e, t \rangle$：同時に同じ辺を逆方向に移動



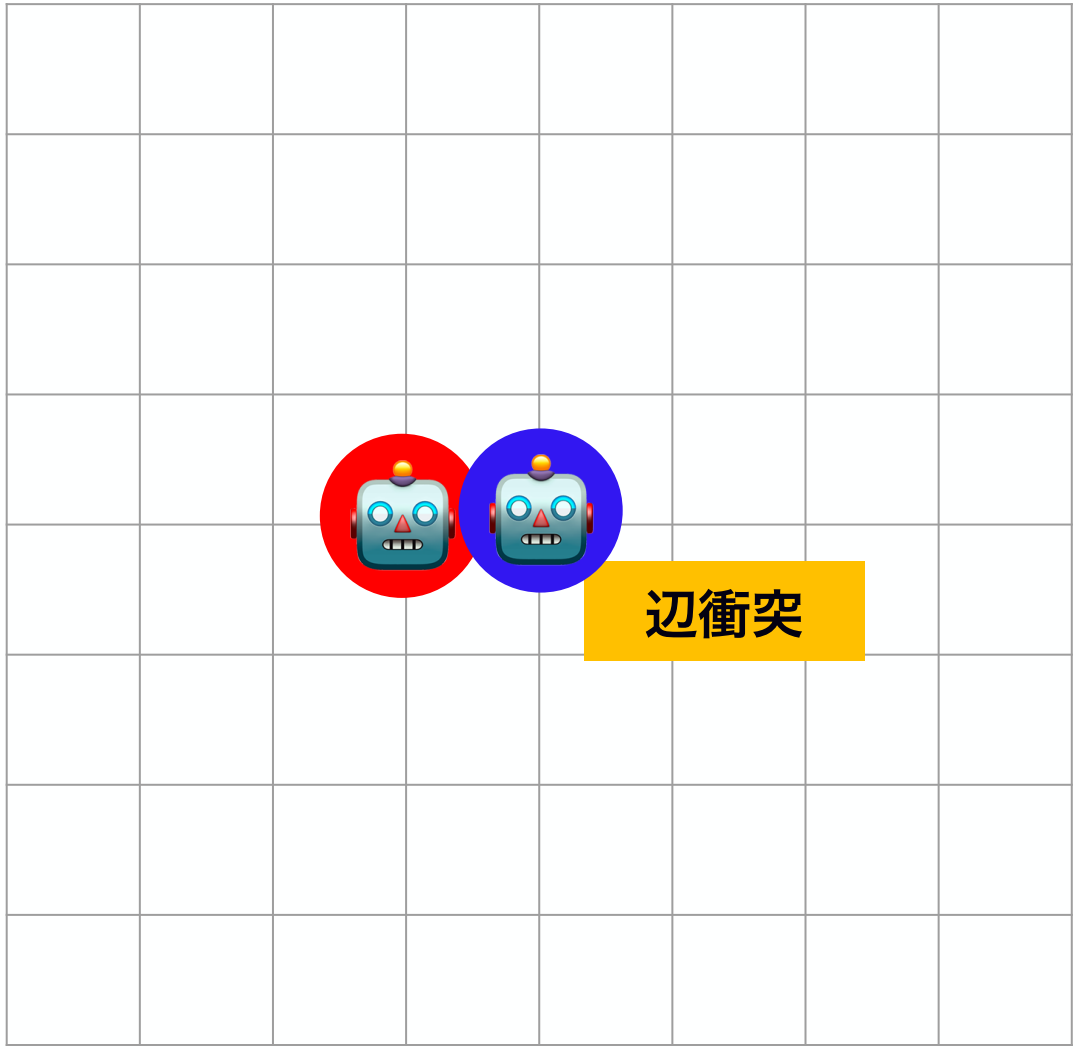
3-1. MAPFの問題定義

複数エージェントがグリッドマップ上を移動し、各目的地に衝突なく到達するための経路

- 無向グリッドグラフ $G = (V, E), \mathbf{v} \in V, e \in E$
- エージェント集合を $\{a_1 \dots a_k\}$ とし、
  各 a_i はスタートs_i とゴール g_i を持つ
- 行動は東西南北・待機 (すべてコスト1)

① 頂点衝突 $\langle a_i, a_j, v, t \rangle$：同時に同じ頂点に存在

② 辺衝突 $\langle a_i, a_j, e, t \rangle$：同時に同じ辺を逆方向に移動



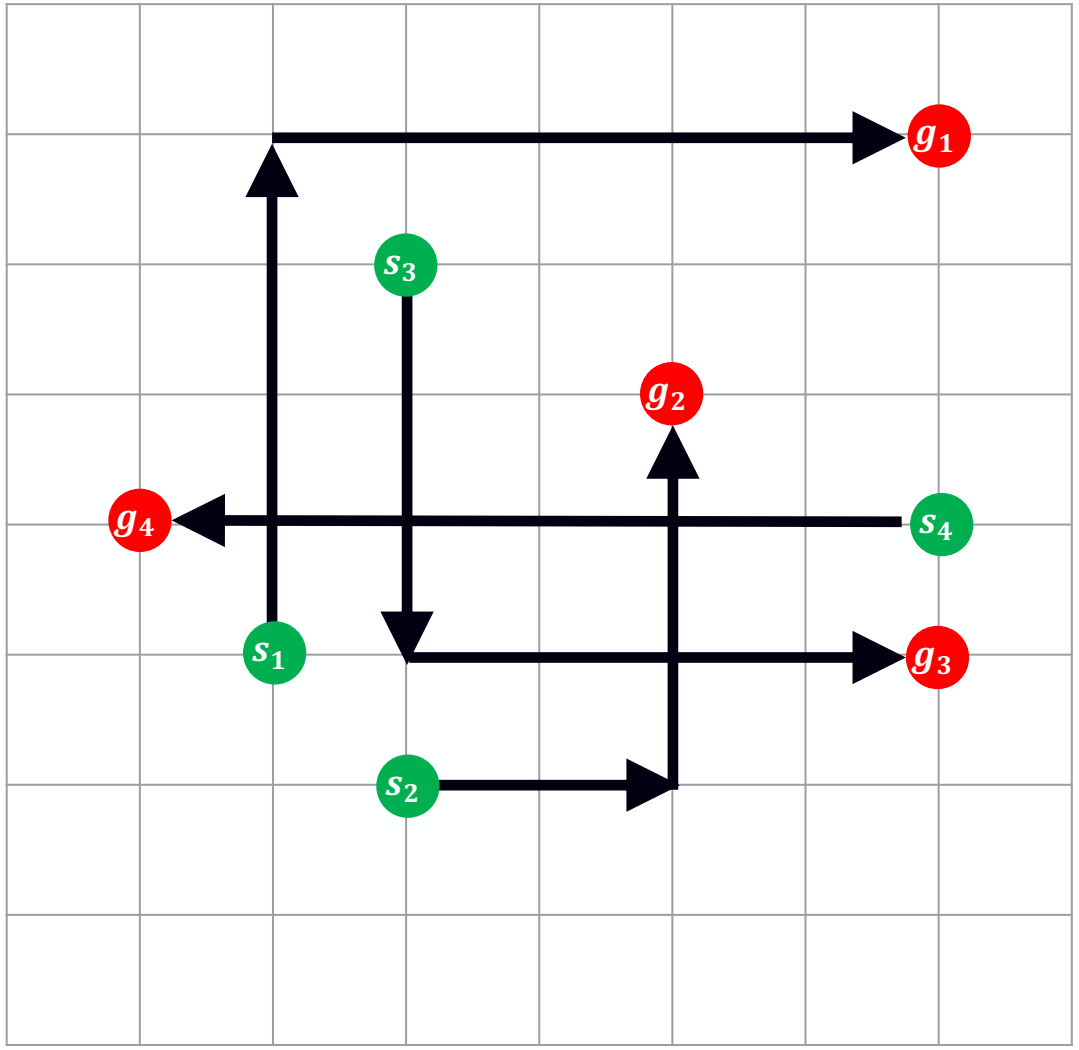
3-1. MAPFの問題定義

複数エージェントがグリッドマップ上を移動し、各目的地に衝突なく到達するための経路

- 無向グリッドグラフ $G = (V, E), \mathbf{v} \in V, e \in E$
- エージェント集合を $\{a_1 \dots a_k\}$ とし、
  各 a_i はスタートs_i とゴール g_i を持つ
- 行動は東西南北・待機 (すべてコスト1)

① 頂点衝突 $\langle a_i, a_j, v, t \rangle$：同時に同じ頂点に存在

② 辺衝突 $\langle a_i, a_j, e, t \rangle$：同時に同じ辺を逆方向に移動



One-shot vs. Lifelong

① One-shot： $s_i \rightarrow g_i$に移動し、衝突なくg_iに留まる
→総移動距離 (SIC=Sum of Individual Costs) の最小化

② Lifelong： g_iに到着すると新たなゴールg_i'が割り当てられ、時刻Tに達するまで繰り返す
→タスクの完了数の合計 (スループット) の最大化

04.

MAPFにおける交通渋滞の考慮

4-1. 交通流割当問題 (TAP=Traffic Assignment Problem)

どのエージェントも経路変更で移動コストを短縮できないUE (User Equilibrium) を目指す

類似点

複数のエージェント (車両) が共有のマップ (道路網) 上を移動し、目的地へ到達することを目指す

相違点

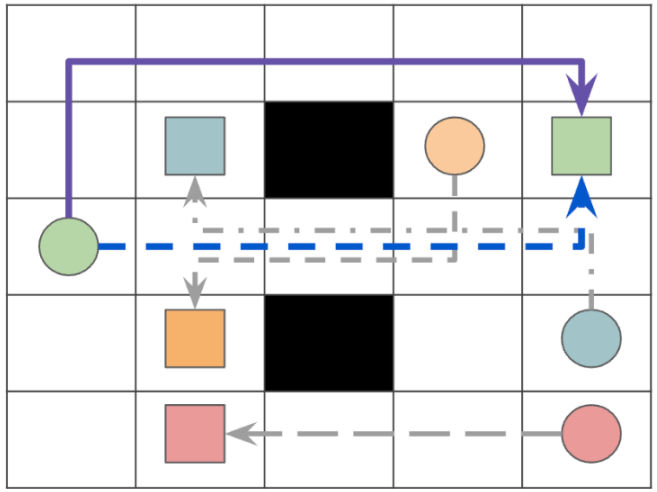
TAPでは同じ頂点・辺 (道路) を同時に複数エージェントが通行できるのに対して、MAPFでは衝突として扱われる

PIBTの課題

個々の最短距離のみを考慮するためボトルネックに集中して渋滞

↓

迂回させるため、時間に依存しないGuide Pathを導入



4-2. 渋滞コストの定義

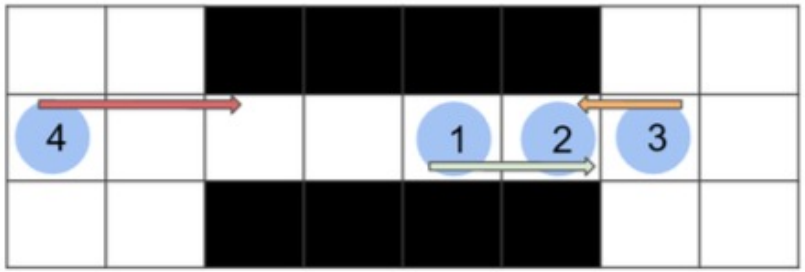
- 頂点v_1からv_2への交通流 (エージェント数) をf_{v_1,v_2}とする

頂点渋滞c_v (Vertex Congestion)

- 頂点vに流入するエージェント数を
  $n = \sum_{v' \in V:(v',v) \in E} f_{v',v}$ として、
  $c_v = \frac{n \times (n-1)}{2}$ がベストケースで発生する
  待機コストの合計
- エージェントあたり $p_v = \lceil \frac{c_v}{n} \rceil = \lceil \frac{n-1}{2} \rceil$

対向流渋滞c_e (Contraflow Congestion)

- 辺 $e \equiv (v_1, v_2) \in E$ に対して $c_e = f_{v_1,v_2} \times f_{v_2,v_1}$



- 渋滞コストを二部構成の $(c_e, \underline{1} + p_{v_2})$ として、順番に最小化する
  free-flow (渋滞がない時)
- $1 + c_e + p_{v_2}$ と $1 + p_{v_2}$ についても実験

4-3. Guide Pathの生成

初期経路計画で渋滞コストを算出

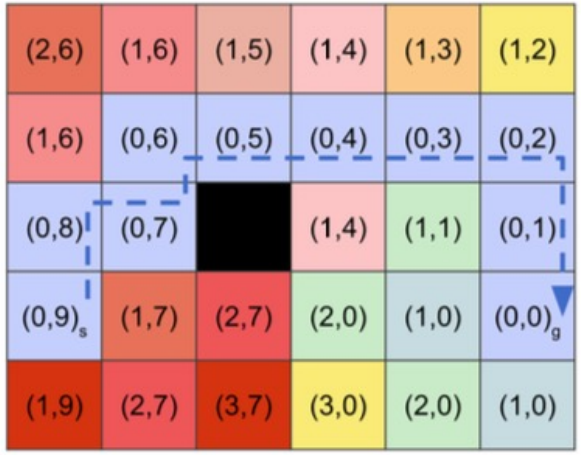
↓

終了条件 (最大反復数) を満たすまで経路の更新を繰り返す

- 渋滞コスト $(c_e, 1 + p_{v_2})$ に基づいて最短経路 $\pi[a]$を FOCAL Search (準最適なA*) を用いて計算

↓

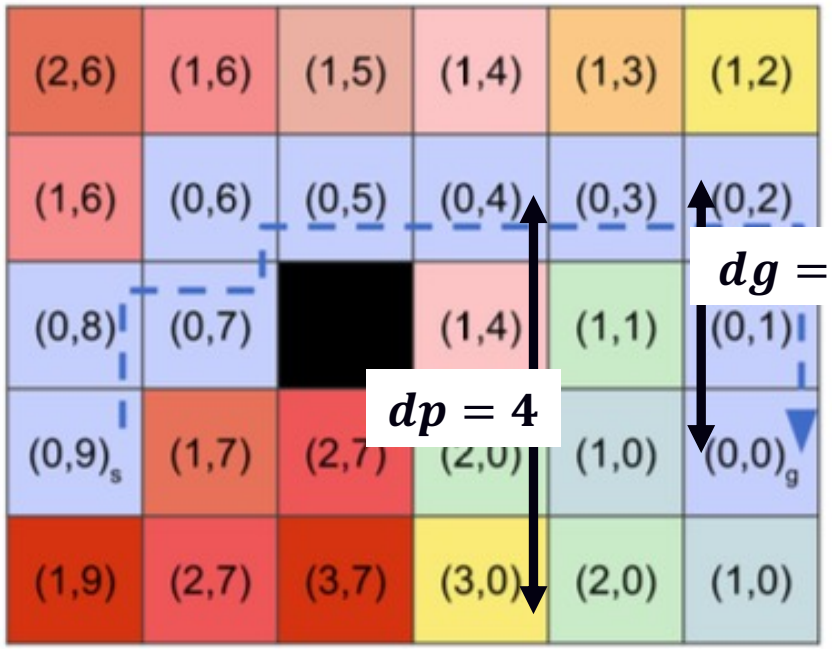
生成した経路をGuide PathとしてPIBTと融合




```
1  FindPaths(A, V, E, s, g)
2      for (v1, v2) ∈ E do
3          f[v1, v2] ← 0; f[v2, v1] ← 0;
4      for a ∈ A do
5          π[a] ← SP(s_a, g_a, f, V, E, w);
6          for i ∈ 2..|π[a]| do
7              f[π[a][i − 1], π[a][i]] + +;
8      return π
9
10 PathRefinement(π)
11     while not meet termination condition do
12         Choose subset S ⊂ A ;
13         π ← Replan(π, S, A, f, V, E) ;
14
15 Replan(π, S, A, f, V, E)
16     for a ∈ S do
17         for i ∈ 2..|π[a]| do
18             f[π[a][i − 1], π[a][i]] − − ;
19     for a ∈ S do
20         π[a] ← SP(s_a, g_a, f, V, E, w);
21         for i ∈ 2..|π[a]| do
22             f[π[a][i − 1], π[a][i]] + +;
23     return π
```

4-3. Guide Heuristicsの導入

- Guide Heuristics: $h_i(v) = (dp, dg)$
  dp: vからGuide Pathへの最短距離 (free-flow)
  dg: Guide Pathを使用した場合のゴールg_iへの最短距離
  隣接する頂点を並び替えるために使用
- 計算には幅優先探索 (BFS=Breadth First Search) を使用してキャッシュ
- 渋滞コストはすでに前段のGuide Pathの計算で考慮されているため、再考慮はしない



$h_{i(v)} = (dp, dg)$

$dist(v, g_i)$に対して$h_i(v)$を用いることで高速化

4-4. One-shot MAPF→Lifelong MAPFへの適用

- Lifelong MAPF では、エージェントがゴールに到着すると新しいタスクが与えられ状況が変化するため、これに対応するためのオンライン処理が必要

初期化フェーズ (Lazy Initialisation)

全エージェントのガイドパスとヒューリスティックを一度に計算すると時間がかかる
→毎タイムステップ一定数のエージェントに対してのみ、Guide Path, Heuristicsを計算

* 計算されなかったエージェントはPIBTを適用

更新フェーズ

ゴールに到着し、新しいゴール$g[a]$を割り当てられた場合、当該エージェントのGuide Pathを更新

```
1  GuidedPlanStep(A,π, t, g, ϕ)
2      //Initialising;
3      if ∃a ∈ A, π[a] = ∅ then
4          A'←A;
5          if Relax then A'← SelectNotInitialized(A);
6          π ← FindPaths(A',V,E,ϕ,g);
7      //Updating;
8      for ∀a ∈ A, g[a] changed, π[a] ≠ ∅ do
9          π ← Replan(π, a, A,f,V,E);
10     //Refining;
11     if ∀a ∈ A, π[a] ≠ ∅ ∧ Refine then
         PathRefinement(π);
12     for ∀a ∈ A where π[a] has changed do h_a ←
         Get(π[a]) ;
13     //PIBT with Guide Heuristic;
14     θ ← PlanStep(A);
15     return θ;
```

4-4. One-shot MAPF→Lifelong MAPFへの適用

- Lifelong MAPF では、エージェントがゴールに到着すると新しいタスクが与えられ状況が変化するため、これに対応するためのオンライン処理が必要

初期化フェーズ (Lazy Initialisation)

全エージェントのガイドパスとヒューリスティックを一度に計算すると時間がかかる
→毎タイムステップ一定数のエージェントに対してのみ、Guide Path, Heuristicsを計算

* 計算されなかったエージェントはPIBTを適用

更新フェーズ

ゴールに到着し、新しいゴール$g[a]$を割り当てられた場合、当該エージェントのGuide Pathを更新

```
1  GuidedPlanStep(A,π, t, g, ϕ)
2      //Initialising;
3      if ∃a ∈ A, π[a] = ∅ then
4          A'←A;
5          if Relax then A'← SelectNotInitialized(A);
6          π ← FindPaths(A',V,E,ϕ,g);
7      //Updating;
8      for ∀a ∈ A, g[a] changed, π[a] ≠ ∅ do
9          π ← Replan(π, a, A,f,V,E);
10     //Refining;
11     if ∀a ∈ A, π[a] ≠ ∅ ∧ Refine then
         PathRefinement(π);
12     for ∀a ∈ A where π[a] has changed do h_a ←
         Get(π[a]) ;
13     //PIBT with Guide Heuristic;
14     θ ← PlanStep(A);
15     return θ;
```

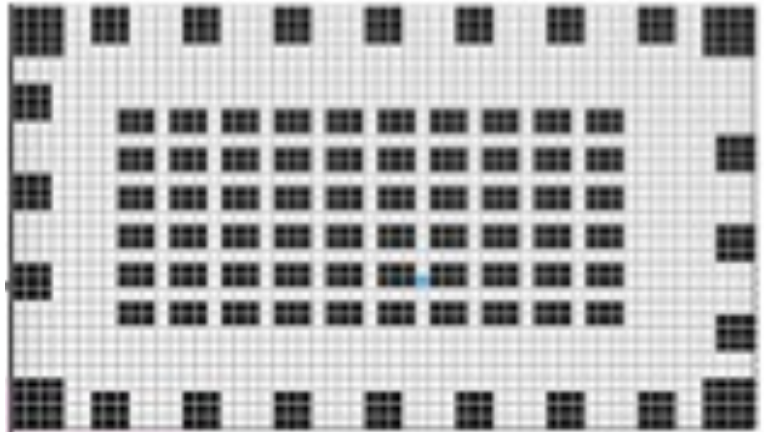
05.

実験結果

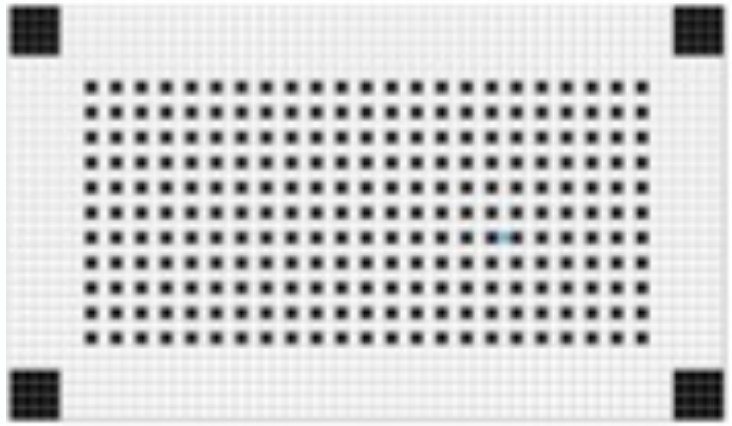
5-1. 実験設定

- 比較対象アルゴリズム：PIBT, LaCAM*, RHCR
- 実験環境：Nectar Cloud VM, 32 AMD EPYC-Rome CPU, 64GB RAM
- タイムアウト：One-shotは60秒、Lifelongは各タイムステップで10秒

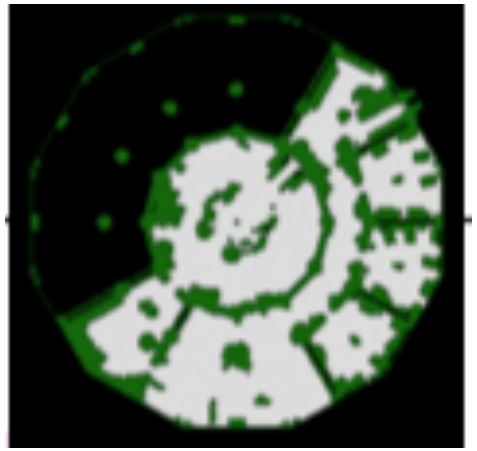
Warehouse



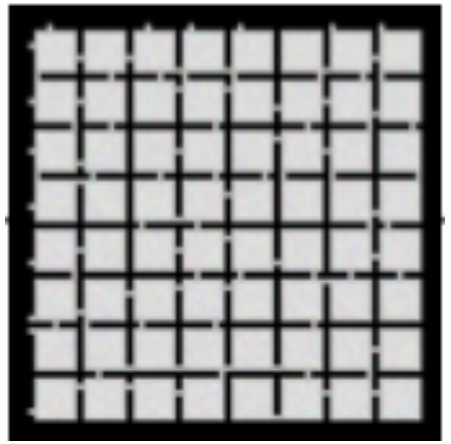
Sortation



Game



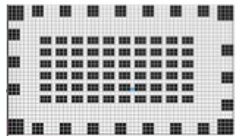
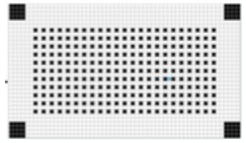
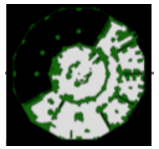
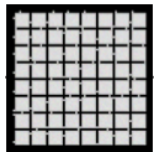
Room



*白が通行可能セル

5-1. 実験設定

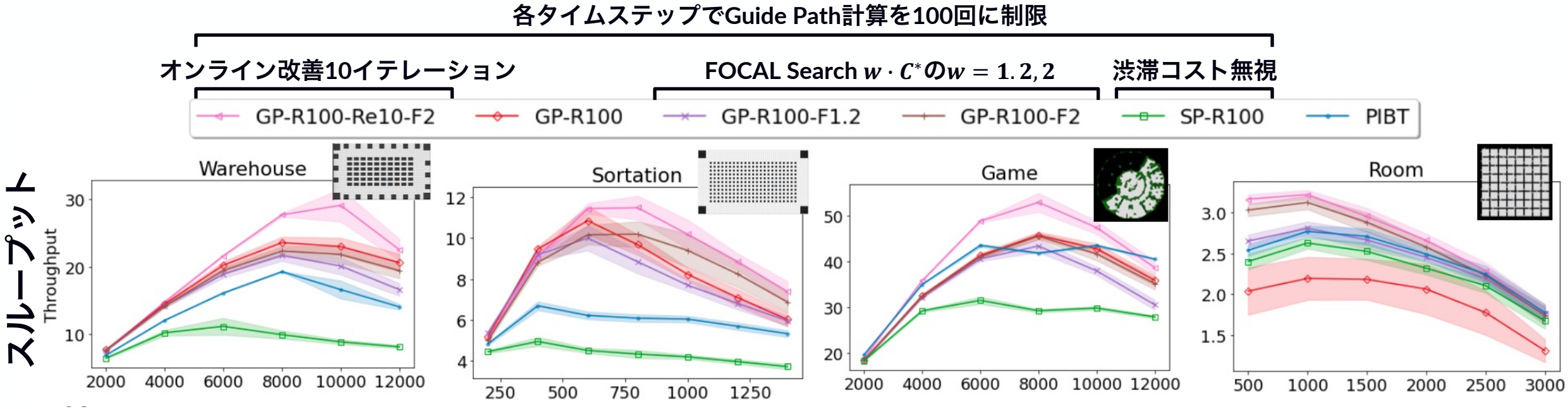
- 比較対象アルゴリズム：PIBT, LaCAM*, RHCR
- 実験環境：Nectar Cloud VM, 32 AMD EPYC-Rome CPU, 64GB RAM
- タイムアウト：One-shotは60秒、Lifelongは各タイムステップで10秒

	グリッドサイズ	エージェント数	タイムステップ数	セル占有率
 Warehouse	500×140	2000-12000	3200	31%
 Sortation	33×57	200-1400	450	92%
 Game	194×194	2000-12000	1940	90%
 Room	64×64	500-3000	640	93%

5-2. 実験結果｜Lifelong MAPF

- GP-R100 (基本提案手法): PIBT と比較して、Warehouse, Sortation, Game マップで顕著なスループット向上。Room マップでは効果が薄い (一本道が多く代替路が少ないため)
- GP-R100-F2 (FOCAL Search w=2): Room マップでもスループットが向上→適度な迂回許容が効果的
- GP-R100-Re10-F2 (オンライン改善あり): 全てのマップ、全てのエージェント数で最大スループット

Guide Pathにより、エージェント数が増加してもスループットが低下し始めるPeak Agent Densityを右側にシフト



5-3. 実験結果｜Lifelong MAPF

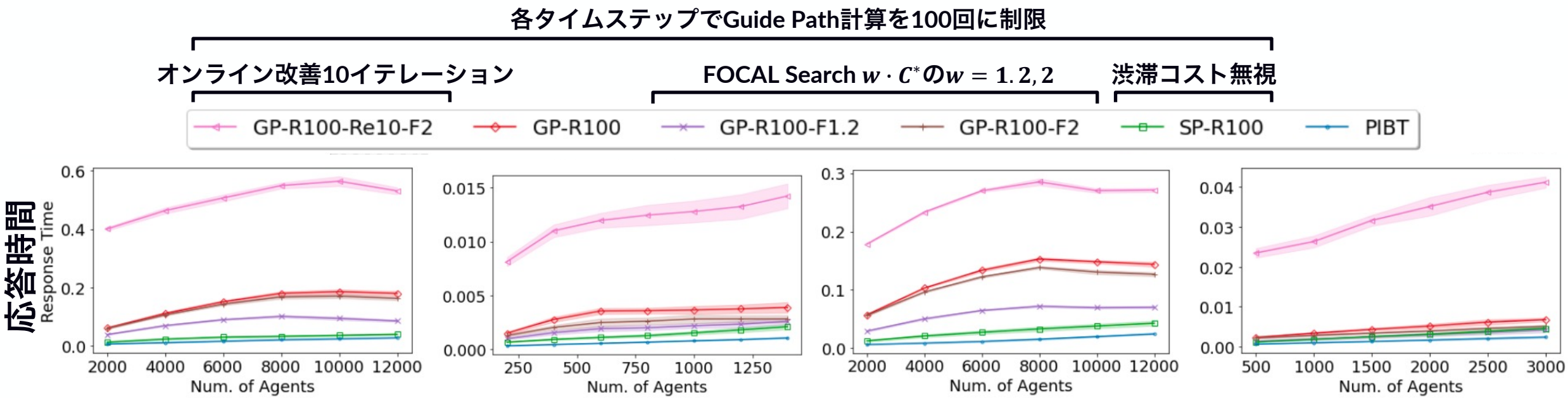
- Guide Path計算のオーバーヘッドがあるものの、安定した応答時間（0.5秒以内）を維持
- 初期化コストを分散 (Lazy Initialisation) することで、応答時間のスパイクを防げている

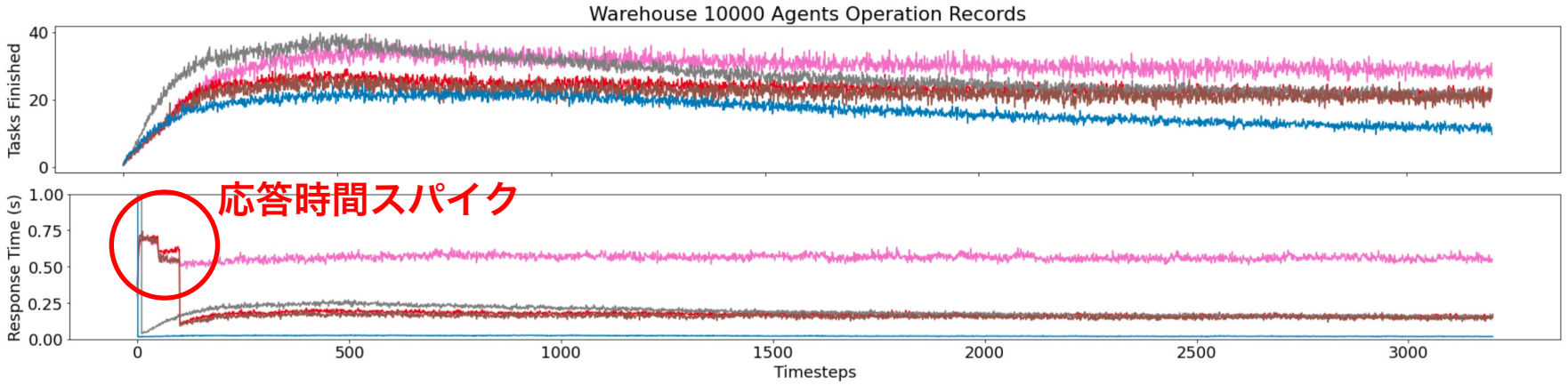
各タイムステップでGuide Path計算を100回に制限

オンライン改善10イテレーション

FOCAL Search $w \cdot C^*$の$w = 1.2, 2$

渋滞コスト無視





5-4. 実験結果｜Lifelong MAPF & RHCR

- Guided PIBTは、Lifelong MAPFの有力な先行研究であるRHCRと比較しても、スケーラビリティと効率で優位性を示した
- RHCR (Rolling-Horizon Collision Resolution): 問題を時間窓で区切り、その窓内の衝突のみを解決するアプローチ
  (RHCR は 1000 エージェントを超える規模へのスケーリングが困難なため、比較的小規模な Sortation マップ (200-1000 エージェント) で比較)

Metric	ALG	200	400	600	800	1000
スループット TP	RHCR-ECBS	5.7±0.1	9.1±0.1	9.1±0.2	5.2±0.3	3.5±0.2
	RHCR-PBS	**6.0±0.1**	**11.1±0.1**	6.5±0.6	5.2±0.5	3.5±0.3
	GP-R100-Re10-F2	5.0±0.1	9.1±0.1	**11.4±0.3**	**11.5±0.6**	**10.2±0.7**
応答時間 R-Time per Timestep	RHCR-ECBS	**0.005±0.0**	0.038±0.002	0.78±0.121	1.987±0.009	2.0±0.0
	RHCR-PBS	0.023±0.0	0.29±0.016	1.997±0.015	2.01±0.006	2.006±0.003
	GP-R100-Re10-F2	0.008±0.0	**0.011±0.001**	**0.012±0.001**	**0.012±0.001**	**0.013±0.001**
R-Time per Planner Run	RHCR-ECBS	0.026±0.0	0.191±0.012	3.905±0.619	9.936±0.048	10.001±0.0
	RHCR-PBS	0.114±0.002	1.449±0.078	9.985±0.074	10.051±0.028	10.03±0.013
	GP-R100-Re10-F2	**0.008±0.0**	**0.011±0.001**	**0.012±0.001**	**0.012±0.001**	**0.013±0.001**

5-5. 実験結果｜Lifelong MAPF & 渋滞コスト比較

- 渋滞コスト関数を変化させて性能への影響を検証
- 2部構成の $(c_e, 1+p_{v_2})$ が計算効率と性能のバランスが良いアプローチといえる

渋滞コスト関数

$1 + p_{v_2}$

頂点のみ

$1 + c_e + p_{v2}$

Han and Yu 2022

Guide Path計算後
ヒューリスティック

ALG	Warehouse (8000)		Sortation (600)		Game (8000)		Room (1000)	
	TP	R-Time (s)	*TP*	R-Time (s)	*TP*	R-Time (s)	*TP*	R-Time (s)
GP-R100	**23.6±5.8**	0.18±0.083	10.9±3.5	0.004±0.002	45.7±8.6	0.153±0.041	2.2±1.5	0.003±0.006
GP-R100-F2	22.3±5.6	0.168±0.08	10.2±3.4	0.002±0.002	45.5±8.6	0.138±0.036	3.1±1.8	0.003±0.004
GP_v-R100	14.9±4.2	0.084±0.059	9.1±3.2	0.002±0.001	41.0±7.7	0.092±0.029	2.9±1.7	0.002±0.004
GP_v-R100-F2	14.8±4.2	0.114±0.076	9.0±3.2	0.002±0.002	41.1±7.7	0.086±0.026	2.9±1.7	0.002±0.004
GP_{svc}-R100	22.9±5.7	0.17±0.083	**11.8±3.8**	0.003±0.002	**46.5±8.7**	0.138±0.037	3.1±1.8	0.003±0.005
GP_{sui}-R100	20.0±5.2	0.076±0.029	8.8±3.1	0.002±0.001	44.3±8.4	0.06±0.015	2.7±1.6	0.002±0.002
GP_{sui}-R100-F2	21.9±5.5	0.149±0.069	11.2±3.6	0.003±0.002	45.6±8.5	0.124±0.032	3.1±1.8	0.003±0.005
TH_v	timeout	timeout	10.9±3.5	0.05±0.008	timeout	timeout	**3.5±1.9**	0.187±0.026
TH_v-R100	timeout	timeout	10.7±3.5	0.011±0.004	31.1±11.6	0.27±0.221	3.4±1.9	0.023±0.015
PIBT	19.3±5.0	**0.021±0.092**	6.2±2.6	**0.001±0.0**	41.9±7.7	**0.015±0.065**	2.8±1.7	**0.001±0.002**

スループット　応答時間

5-6. 実験結果｜One-shot MAPF & LaCAM*

- 多くのマップでGuided LaCAM*はLaCAM*より小さいSICを達成 (特にエージェント密度が高いWarehouseで効果的)
- 実行時間 60 秒のうち、最初の 30 秒をGuide Pathの計算と改善に割り当て、残りの 30 秒で LaCAM* が解探索を行うガイドパス計算には FOCAL Search を用いた

評価指標

- Relative SIC (RC): SIC_{guided}/SIC_{lacam}
  (1 より小さいほど Guided LaCAM* の方が解の質が良い)
- Solved: 60 秒以内に解を見つけられたインスタンス数

Map	Agents	RC		Solved		
		F1.2	F2	F1.2	F2	LC*
Warehouse	2000	0.982	1.467	24	24	24
	4000	0.971	1.166	24	24	24
	6000	0.942	1.106	24	24	24
	8000	0.852	1.009	24	24	24
	10000	0.770	0.954	24	24	24
	12000	-	-	0	0	24
Sortation	200	0.965	0.992	22	24	24
	400	0.860	0.888	22	24	24
	600	0.754	0.751	22	24	24
	800	0.728	0.695	21	24	24
	1000	0.759	0.735	22	24	24
	1200	0.815	0.817	22	24	24
	1400	0.903	0.906	22	24	24
Game	2000	1.058	1.084	19	24	24
	4000	1.070	1.121	19	24	24
	6000	1.072	1.099	19	24	24
	8000	1.029	1.053	6	3	22
	10000	-	-	0	0	14
Room	500	0.904	0.888	22	24	24
	1000	0.890	0.804	22	24	24
	1500	0.921	0.801	22	24	24
	2000	0.948	0.828	20	15	19
	2500	-	-	0	0	1

06.

結論

6. 結論

- 従来は近視眼的な経路のみを考慮していたMAPFに渋滞という概念を頂点/辺と分けて導入し、これらを考慮したGuide Pathという比較的シンプルなアルゴリズムによって計算速度と解の品質の両方を向上
- 多様な実験シナリオによってアルゴリズムの有効性を検証
- 現在はFocal Searchで迂回距離と渋滞回避のバランスを取っているが、これをより明示的に最適化できる目的関数を追求する

所感

- 逆にこれまでMAPFで最速だったものが順番待ちをするようなアルゴリズムだったことに驚いた
- 卒論はUAVの飛行軌道を連続値の (x, y, z) 座標で最適化していたが、ベースラインとして離散ボクセルでの比較も必要
- ヒューリスティック vs. MLでどちらが勝つか

> To address the bottleneck issue, newer work in heuristic search has proposed using non-uniform edge costs or guidance [4], [43]. Future ML MAPF works should exploit this literature and be amenable to such works [35].

Veerapaneni, R., Jakobsson, A., Ren, K., Kim, S., Li, J., & Likhachev, M. (2024). Work Smarter Not Harder: Simple Imitation Learning with CS-PIBT Outperforms Large Scale Imitation Learning for MAPF. arXiv preprint arXiv:2409.14491.

- 奥村さんのMAPF記事は非常にわかりやすい

https://kei18.github.io/note/posts/mapf-tutorial/