Extending PIBT to support Heterogeneous Robot Fleets

Arjo Chakravarty[1], Michael X. Grey[2], M. A. Viraj J. Muthugala[3] and Mohan Rajesh Elara[3]

***Abstract*— Priority Inheritance and Backtracking (PIBT) has demonstrated significant success as a heuristic for large-scale Multi-Agent Path Finding (MAPF), but its application has been limited to scenarios where all robots move at uniform speeds. This work addresses the critical question of extending PIBT's capabilities to heterogeneous fleets, where robots may vary in size and speed. We introduce a novel collision-checking-based reservation system integrated within the WinPIBT framework and a new backtracking mechanism. Heterogeneous PIBT (Het-PIBT) effectively scales to hundreds of robots of various sizes. Our findings confirm that this scheme enables the application of heterogeneous MAPF to large robot populations. To facilitate further research, we also provide a new set of heterogeneous MAPF benchmark scenarios along with a Python package for the generation of such problems and visualization tools. However, we observe that HetPIBT, similar to its homogeneous counterpart, can yield suboptimal solutions. We provide an open source implementation of the algorithm in Rust.**

I. INTRODUCTION

The problem of Multi-Agent Path Finding (MAPF) is critical for large-scale logistics, warehousing, and autonomous vehicle systems [1]. Solving MAPF at the scale demanded by modern industrial applications—often involving thousands of agents—requires extremely fast, high-throughput planners. Heuristic-based, rule-following approaches, such as Priority Inheritance and Back Tracking Planner (PIBT) [2] and its derivatives like LaCAM [3], have successfully demonstrated real-time planning for thousands of homogeneous robots on modest hardware.

However, there is a growing interest in having heterogeneous fleets, composed of agents with diverse footprints (e.g., small mobile robots versus large AGVs) and varying dynamics or maximum velocities. Although existing approaches like Conflict-Based Search (CBS) extensions (e.g., LA-CBS [4]) can accommodate such heterogeneity through modular low-level planners, their reliance on global tree searches leads to an unpredictable branching factor and failure to scale beyond a few hundred agents. Crucially, highly scalable heuristics like PIBT fundamentally rely on the assumption of uniform agent size and identical travel dynamics, making them unsuitable for deployment in heterogeneous environments.

This paper addresses the core challenge of achieving high-throughput MAPF for heterogeneous fleets. We introduce Het-PIBT, a novel extension of the PIBT framework that retains the priority-based scalability while rigorously guaranteeing collision-free paths for multi-sized agents with asynchronous movement. Het-PIBT leverages an extension of the WinPIBT reservation system [5] to model time-space conflicts, enabling the planner to account for the physical size and variable travel time of each agent. This approach allows us to bridge the critical gap between high-scalability and real-world applicability in MAPF. Our key technical contributions are:

- A general reservation-based extension to PIBT that enables safe path planning for heterogeneous footprints and varying asynchronous velocities.
- A conservative multi-graph space-time collision model to accurately detect conflicts between differently sized agents.
- A dependency-based time reconstruction mechanism integrated into the planning loop, enabling safe motion planning despite inconsistent travel speeds and ensuring the integrity of reservations.
- A Heterogeneous MAPF benchmark set and an open-source solver to facilitate future research in this under-explored domain.

The remainder of this paper is structured as follows: Section II reviews related work in MAPF scalability and heterogeneity. Section III details the formulation of the Heterogeneous MAPF problem. Section IV presents the core Het-PIBT algorithm. Section V presents our benchmark generation pipeline and Section VI presents our results and associated discussion.

II. BACKGROUND AND RELATED WORK

A. Multi-Agent Path Finding (MAPF) and Scalability

The fundamental challenge in Multi-Agent Path Finding (MAPF) lies in efficiently finding a set of collision-free paths for N agents from their start to their goal locations. Given that finding an optimal solution is NP-hard, the history of MAPF is characterized by methods that consider the trade-offs between optimality guarantees and computational speed.

Initial scalable attempts often relied on simple priority-based systems; however, these suffered from inherent incompleteness and were prone to deadlocks. More rigorous approaches emerged through compilation methods– transforming MAPF into SAT [6] [7] or MILP–but these were limited to small graphs and low agent counts [7] [2].

This research is supported by the National Robotics Programme under category National Robotics Programme 2.0, LEO 1.0: A New Class of Bed Making Robot, Award No. M25N4N2028, and also supported by A*STAR under its RIE2025 IAF-PP programme, Modular Reconfigurable Mobile Robots (MR)2, Grant No. M24N2a0039.

[1]Arjo Chakravarty is with Intrinsic LLC and Singapore University of Technology and Design, Singapore. arjoc@intrinsic.ai, arjo_chakravarty@mymail.sutd.edu.sg

[2]Michael X. Grey is with Intrinsic LLC mxgrey@intrinsic.ai

[3]M. A. Viraj J. Muthugala and Mohan Rajesh Elara are with Singapore University of Technology and Design, Singapore. {viraj_jagathpriya,rajeshelara}@sutd.edu.sg

The landscape shifted significantly with the introduction of Conflict-Based Search (CBS) [8]. CBS achieves optimality by employing a high-level search over conflicts and a low-level path planner. While extensions like Extended Enhanced CBS (EECBS) [9] have pushed the envelope by relaxing the optimality constraints, the scalability of CBS is fundamentally limited by the unpredictable branching factor of its conflict tree, making it unsuitable for thousands of conflicting agents.

To achieve industrial-scale throughput, the focus shifted to high-speed heuristics. PIBT (Priority Inheritance and Backtracking) [2] and its successors, such as LaCAM [3], represent the current state-of-the-art for throughput, enabling planning for thousands of robots in real time. Separately, learning-based planners [10] have also been explored to achieve high scale, often trained on the results of fast heuristic solvers like PIBT or LaCAM. However, their primary challenge remains the difficulty in providing strong, verifiable collision-free guarantees necessary for safety-critical real-world deployment. Our work operates within the domain of rule-based, deterministic planners that provide guaranteed collision-free paths.

B. Heterogeneity in MAPF

Heterogeneous robot fleets are increasingly seen as a tool to unlock new capabilities in brown field industrial settings. Having the ability to mix robots from different vendors can unlock new capabilities in many cases. For instance, in a warehouse you may have large autonomous pallet jacks being used to move heavy items while a smaller AMR might be used for sorting items.

One of the initial pieces of work towards heterogeneity in MAPF was Large Agent MAPF (LA-MAPF) [4]. In Large-Agent MAPF, rather than modeling agents as a single point, individual agents have different sizes. An extension to CBS called LA-CBS was proposed, where the low level A* in CBS was replaced with an A* that is aware of agent footprints. More recent efforts in this direction have extended this to kinodynamics and task heterogeneity [11] by allowing CBS to incorporate a variety of different low-level planners (e.g., diffusion-based or RRT-based) instead of being restricted to A*.

Although CBS provides a flexible way to implement heterogeneous systems by modularly swapping its low-level components, its main downside is an unpredictable branching factor, which can lead to performance stagnation. To address this, iterative alternatives like PIBT offer an appealing performance guarantee, while maintaining completeness for multi-agent pickup and delivery on any bi-connected graph. However, as discussed in the following section, PIBT makes the critical assumption that every agent moves in unison during each time step. This may not always be true in LA-MAPF as one agent might move more slowly than another. Our work seeks to answer the question: Is there an iterative extension of PIBT suitable for agents with heterogeneous footprints and dynamics?

C. PIBT

PIBT was first introduced by Okumura Et al. in 2022. Unlike traditional MAPF solvers, PIBT does not focus on optimality. Rather, it is a configuration generator: given a biconnected graph, PIBT will give another valid configuration.

PIBT generates new configurations using a priority based scheme. First, it ranks agents according to their priorities. Priorities are calculated based on the distance from the final goal. A common cause of deadlocks in priority based systems is Priority Inversion. Priority inversion occurs when a lower priority agent blocks the progress of a high priority agent. In PiBT, blocking agents inherit the priority of the agent that needs to move. In the event that we reach a deadlock while performing priority inheritance, we backtrack by picking another agent to have a higher priority [2]. PIBT itself is extremely fast as it is a greedy algorithm picking the next best configuration according to an approximate cost function. Given a biconnected graph, PIBT will always output a safe configuration as a next step. However, PIBT assumes that all robots behave according to the same dynamics, making it unsuitable for heterogeneous systems.

D. WinPIBT

WinPIBT [5] is an extension to PIBT that extends the look up of PIBT from 1 step ahead to n-steps ahead. The key idea in WinPIBT is that instead of just looking one step ahead, one maintains a reservation table of robot trajectories. Higher priority robots reserve their trajectories in this table as they find new trajectories that do not collide with previously reserved trajectories. In the event that no solution is found, back tracking can be performed to help find a solution. WinPIBT is particularly useful when dealing with tight constrained passages such as narrow aisles in a warehouse. In our work, we adapt the longer lookup horizon of WinPIBT to accommodate variability in the time steps of different robots.

E. Reservation Systems

Fleet management systems often use reservation subroutines to be more effective at managing contention between agents. They are often used to ensure that robots do not collide when parked. They may also be used for emergency parking and resource allocation [12]. PIBT can be framed as running a reservation system at every time step. This insight is critical for us to be able to scale PIBT to heterogeneous fleets. We propose a simple reservation system that takes into account robot footprints and configurations.

F. Benchmarks in the context of MAPF

One of the most commonly used benchmarks for MAPF are the maps and scenarios described in [1]. More recent additions to the field include POGEMA, which provides scripts to generate instances of homogeneous MAPF so that one can train RL models [13]. Amazon runs an annual competition for the fastest MAPF solver. In 2024, the competition introduced a new variation of MAPF with Turns [14]. All of

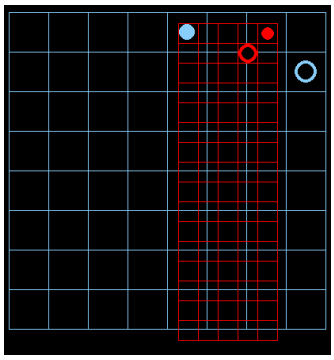

Fig. 1: Example instance of Heterogeneous Agents. Here the red dot represents an agent and the blue dot another. Their goals are represented by the circles. Each agent can move a different amount per time step. In particular the red agents footprint is smaller but its velocity is also lower.

these benchmarks and simulations assume a uniform cell size with uniform speeds across all robots.

Although heterogeneity is not often addressed in the field of MAPF, the Multi-Agent Reinforcement Learning field has a number of gyms that exercise this capability. Recent work in this domain includes VMAS, which provides a set of scenarios in which multiple agents must cooperate to achieve their end goals [15]. In fact, VMAS covers a much broader scope of multi-agent cooperation.

Another work of interest here is REMROC which provides a nice end-to-end framework for MAPF in unstructured environments [16]. REMROC's key feature is the fact that it simulates the entire navigation stack of each robot along with some crowd. However, running such an end-to-end test is expensive, and for the purpose of studying scalability up to thousands of agents, simulating the entire physical stack introduces variables that obfuscate analysis.

III. PROBLEM STATEMENT

This paper aims to introduce a derivative of Priority Inheritance with Backtracking (PIBT) tailored for heterogeneous multi-agent systems. An illustrative example of this system is provided in Figure 1.

Given a set of Agents A_i^g that move on a graph G^g, where each agent has a different start position and end position $(s_i^g, e_i^g) \in G^g$, find a collision free trajectory for each agent from s_i^g to e_i^g

A collision-free trajectory for an agent is a sequence of discrete locations (vertices in the graph) over time, ensuring that:

- The agent's path starts at its designated start position and ends at its goal position.
- At no point in time are two agents in collision with each other.
- The trajectory respects the specific movement capabilities and constraints of each agent type (specifically, speed and footprint in this case).

We define G' as the base occupancy grid with static obstacles. The cell size of G' must be equal to or smaller than the smallest graph G^x.

To better categorize agent types within a heterogeneous system, we introduce the term "fleet". A fleet refers to a group of robots (agents) that share identical characteristics, including their dynamics, size (footprint), and the specific graph or sub-graph on which they operate. Consequently, agents with different dynamics or footprints are considered to belong to different fleets.

IV. PROPOSED ALGORITHM

A. Handling Heterogeneity

To handle heterogeneity, we need to consider the fact that traditionally PIBT assumes that at each time step agents can fully clear out of the cell they currently occupy. In heterogeneous scenarios, this is not always the case. A robot that moves slowly might not be able to evacuate a cell within a single timestep of another robot. Thus, we have to employ WinPIBT's approach [5]. Instead of looking ahead just one timestep into the future, we extend the look-ahead to be the area of the cell that needs clearing. However, that alone is not sufficient as the start time of the highest priority agent will be affected by the amount of time it takes to clear the cell. This is demonstrated by Figure 2.

Decisions are made sequentially according to the priority of an agent. We first determine the path for the highest-priority agent without regard for lower-priority positions; these positions are then updated by treating the highest-priority agent's path as a hard constraint. We recursively travel down all blocking agents by asking them to move out and pick the next best location. However, unlike vanilla PiBT we don't know that a child agent can move out in one time step, so we simply build up a dependency graph of who needs to move out. We then backtrack and fill the timing information up. This is similar to the causal-PIBT's approach, where we first try to figure out the order of the motion and then execute the motion [17].

B. Reservation System

We represent the reservation system as P in our subsequent discourse. P is defined as the map of agents A_i^g to their trajectories ξ. Due to the possibility that robots have different sizes, each fleet may have its own occupancy graph representation, but these graphs may overlap in Cartesian space (see Figure 1). We define a mapping T_g between every cell in G^g and the base occupancy grid G' as shown in equation (1). A single cell on a graph can map to multiple occupied cells on the base occupancy grid.

$$T_g : (c^g) \rightarrow G' \tag{1}$$

Here we use c^g to refer to a cell on the graph G^g. Two cells c_1, c_2 in between two graphs G^1, G^2 overlap if $T_{G^1}(c_1) \cap T_{G^2}(c_2) \neq \{\}$. A trajectory refers to a timed series of cells of a graph (see 2).

$$\xi^i = [(c^i_{t_0}, t_0), (c^i_{t_0+\Delta}, t_0+\Delta), (c^i_{t_0+2\Delta}, t_0+2\Delta), ...] \tag{2}$$

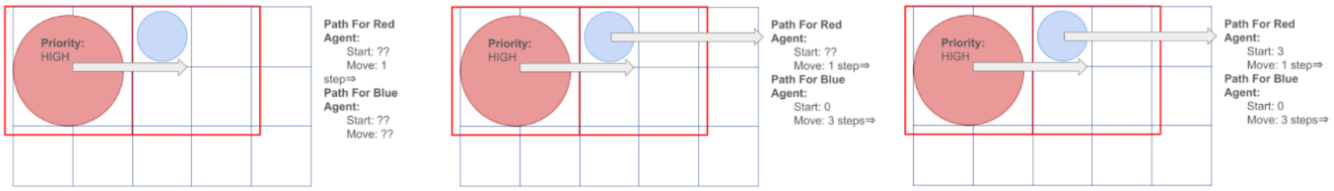

Fig. 2: Resolving timing dependencies later in order for the agents to safely complete an action

Where Δ is the uniform timestep used across all agents when planning.

A path, on the other hand, refers to a series of cells with no timing information (see 3).

$$\Pi = [c_1, c_2, ...] \tag{3}$$

We say two trajectories ξ^1, ξ^2 collide if there exists a t such that there are common nodes (see (4))

$$T_1(c_t^1) \cap T_2(c_t^2) \neq \{\} \tag{4}$$

Or the agents swap positions. That is, given two trajectories ξ^1 and ξ^2, there is a situation at time t such that $T_1(c_{t+\Delta}^1) \cap T_2(c_t^2) \neq \{\}$ and $T_1(c_t^1) \cap T_2(c_{t+\Delta}^2) \neq \{\}$.

When adding a trajectory ξ to P we must ensure that none of the trajectories $\xi^* \in P$, collide with ξ. We achieve this via space time search when generating new trajectories. See IV-E for more details.

C. High level iterative process

The high level iterative search is largely identical to the iterative process used in WinPIBT [5]. At each time step, we choose the agent with the highest priority that does not have a trajectory allocated and then try to assign it a trajectory using Algorithm 1. If the ordering fails, we backtrack and try picking another agent as a high priority agent. We call this the Priority Traversal Search.

D. Priority traversal search

The priority determination is described by the Algorithm 1. We greedily pick the next best location for the agent, while making sure that we do not violate the trajectories in P.

In the event a low priority agent blocks a high priority agent, as seen in Figure 2, we perform priority inheritance by creating "Keep Out Zones". These "Keep Out Zones" define a region that the low priority agent must vacate so higher priority agents can move in. Once we know that we have a valid set of movements that free up space for high priority agents, we then backtrack and figure out at what time the agents should move. This is represented by BACKTRACKANDRESERVE, in Algorithm 1. The back tracking is done via Figure 2.

As it takes longer for a small agent to move out of a large agent's way, we adjust the look-ahead depth d based on the ratio of their sizes.

GETNEXTLOCATIONS represents the Per-Agent Search described in IV-E. Its goal is to enumerate the best path for the current agent under consideration to leave the "Keep Out Zone" and which unassigned agents need to move out in order for the current agent to be able to move to its goal. The path which the current agent takes is added to the "Keep Out Zone" as all agents that block this area need to inherit priority and leave the "Keep Out Zone".

Algorithm 1 Reservation Algorithm

function PRIORITYTRAVERSAL(a_i: Agent, P: Reserved Trajectories, t: Time)
  `keep_out` $\leftarrow \{\}$ ▷ Cells which the current agent must avoid
  d $\leftarrow 1$ ▷ The search depth for WinPiBT
  $Q \leftarrow \{(\{a_i\}, d, \texttt{keep\_out})\}$
  while $Q \neq \{\}$ **do**
    $a_i, d,$ `keep_out` $\leftarrow Q.\texttt{pop}()$
    for $b, \Pi \leftarrow \texttt{GetNextLocations}(a, P, t, \texttt{keep\_out}, d)$ **do**
      b : List of blocking agents
      Π : Path of each agent
      if $b.\texttt{len} = 0$ **then**
        `BackTrackAndReserve`(P)
        return
      end if
      if $b.\texttt{len} > 1$ **then**
        continue ▷ While possible to do more than one, this will lead to exponential state-space explosion
      end if
      if $a_i.size > b_i.size$ **then**
        $d = \lceil \frac{b_i.size}{a_i.size} \rceil$
      end if
      $Q.\texttt{push}((b, d, \texttt{keep\_out} \cup \Pi))$
    end for
  end while
  return ▷ Failed to reserve anything stay put
end function

E. Per Agent Path Search

To get successor locations for each agent, we rely on Algorithm 2. The goal of this function is to return a list of agents that need to move out and the relevant paths that the agents need to take. We perform space-time BFS so that the agent inheriting the priority of the higher priority agent

leaves the keep out zone. We order potential destinations by their distance from the desired goal for the current agent.

To make swaps work, we break ties by the number of agents that would need to be moved out to allow the current agent some free space. This process yields a path rather than a full trajectory for blocked agents; the complete, timed trajectory is then reconstructed using the method shown in Figure 2. Blocked agents will effectively act as static obstacles until the trajectories of the blocking agents are determined.

Algorithm 2 Per-agent Search Algorithm

function GETNEXTLOCATIONS(a_i: Agent, P: Reserved Paths, d: Depth, K: path-finding successful if we are not in this location, t:current time)
  $Q \leftarrow \{(a_i.pos, t)\}$
  `solutions` $= \{\}$
  `affects` $= \{\}$
  while $Q \neq \{\}$ **do**
    `pos, time` $\leftarrow Q$.`pop_front()`
    if $pos \notin K$ **then**
      `solutions.add(path_to(pos))`
    end if
    if `time` $> d$ **then**
      continue
    end if
    for `neighbour` of `pos` **do**
      if moving from `pos` to `neighbour` collides with a trajectory in P after `time` **then**
        continue
      end if
      if moving from `pos` to `neighbour` collides with the last know position of a_j in P **then**
        if `time` $> a_j$`.last_time` **then**
          `affects[pos]` $= a_j$
        end if
      end if
    end for
  end while
  sort `solutions` by distance to goal for a_i and then by number of affected agents
  return solutions and affected agents by solution
end function

V. BENCHMARKS

After some survey, we determined that there is not a sufficiently representative set of benchmarks for heterogeneous fleets. We reuse Stern Et al.'s MAPF benchmarks for maps, but we generate our own agent footprints and starting positions [1]. The occupancy map for each scenario is projected onto the grid cells of every fleet. In order to generate valid agent paths, we randomly select a start position for each agent and then set a goal for each agent. We make sure that each goal can be reached from its respective start position.

Our benchmarks are designed to answer these key questions:

- How does HetPIBT scale with the number of robots of the same kind?
- How does HetPIBT scale with the number of robot types?
- How far from optimal are the solutions generated by HetPIBT?

We implement our solver in Rust and provide an open source package[1].

A. Benchmark Design

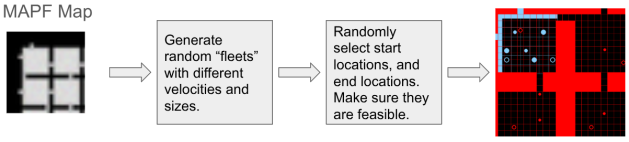

Fig. 3: Benchmark Generation Pipeline

Figure 3 shows how the benchmarks are generated. We reuse the MAPF benchmark maps [1]. We overlay a set of different grids that are dynamically generated to represent the motion of a robot moving from a starting point to a goal at a certain speed. An important consideration is making sure that the goal can be reached from its associated starting point.

We generate 10 instances per map using this methodology and save them as files. The benchmark scenarios are available in a repository[2]. Each scenario is formatted as a plain text file with lines specifying the agent, the agent's fleet, the fleet's footprint, the fleet's velocity, and start and goal positions. To ensure that the benchmarks capture conflicts between different fleets, all navigation graphs cover the same area but may have different velocity and cell size components.

A Python package is provided to generate further benchmark scenarios and visualization tools to support future research.

VI. RESULTS AND DISCUSSION

A. Scaling in Robot Dimension

In the event that there is only one fleet, HetPIBT and WinPIBT are identical. This means that for one fleet we can easily search for thousands of robots. As the number of fleets increases, there is an additive performance penalty. We discuss this in more depth in the next section. Despite this penalty, figure 4 shows that we can plan for 700 robots across 14 different fleets within a minute.

B. Scaling in terms of multiple fleets

Figure 4 illustrates the scaling capabilities of Heterogeneous PIBT (HetPIBT) in relation to the number of fleets. The results demonstrate that HetPIBT effectively scales to 14

[1] https://github.com/arjo129/pibt_rs
[2] https://github.com/arjo129/pypibt/tree/main/het_bench

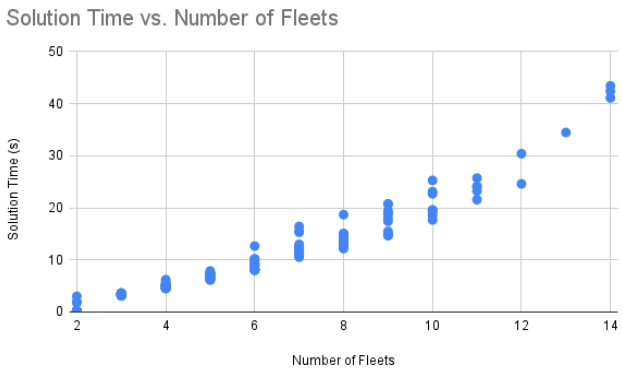

Fig. 4: Solution time for HetPIBT given number of fleets. Each fleet consists of 55 robots, each nav-graph is of a different size. The environment was based on the `room-64-64-8` map.

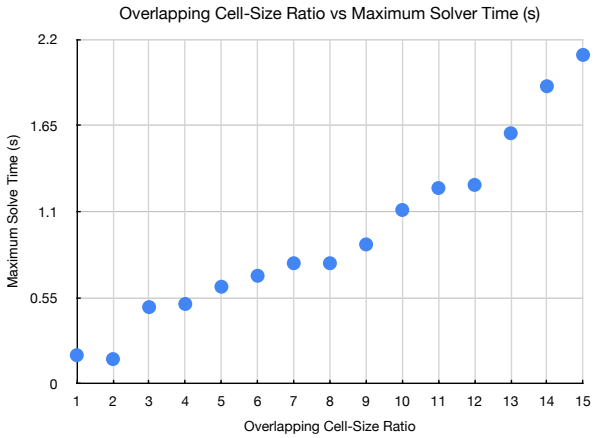

Fig. 5: Overlapping Cell-Size Ratio vs time to solve. We keep the number of fleets at 3 and the number of agents per fleet at 55.

fleets, each composed of 55 robots, for a total of 770 robots within a single instance. However, it is important to note that increasing the number of fleets incurs a significantly higher cost compared to increasing the number of robots within existing fleets.

One thing to note is that the "Overlapping Cell-Size Ratio" is very important for the performance of HetPIBT. This describes the size ratio between the largest and the smallest set of robots. Figure 5 shows the effect that the overlapping ratio has with respect to the maximum solver time. For this set of experiments, we ran the solver 100 times at each cell size ratio across a set of different random scenarios. We plot the maximum time the solver takes to solve the instance against the overlapping cell size ratio. As the size difference in individual cells across the fleets' occupancy graphs increases, we observe that the maximum solver time increases. This results from an increase in planning complexity as larger robots will tend to be obstructed by an increasing number of smaller robots as the difference in cell size grows.

TABLE I: CCBS vs HetPIBT

Scenario	HetPIBT		CCBS	
	Len.	Time (s)	Opt. Len.	Time (s)
room-64-64-8.0	**375**	**2.87**	**375**	15.1
room-64-64-8.2	**105**	**0.96**	–	TLE
room-64-64-8.3	**138**	**1.21**	–	TLE
room-64-64-8.5	**398**	**1.81**	–	TLE
room-64-64-8.6	358	**2.51**	**188**	17.2
room-64-64-8.7	**133**	**2.09**	**133**	28.1
room-64-64-8.8	503	**2.04**	**122**	29.0
room-64-64-8.9	664	**2.05**	**322**	28.1

C. Comparing baselines and sub-optimality

The best baseline we have for multi-agent path finding in heterogeneous domains is Conflict Based Search (CBS). Particularly, we rely on a CCBS based approach using the Open-RMF MAPF library, which supports heterogeneous agents. While it may be difficult to scale CBS to 1000 robots, we can still compare how far from optimality HetPIBT is for smaller scales. This is important because PIBT is known to produce suboptimal trajectories. The instances we used in this case have 9 robots spread across 3 fleets. Each fleet has a footprint which is about 3 times the size of the next smaller fleet. We give an upper limit of 1 minute for each solver to solve each instance. Based on our results in Table I, we can see that HetPiBT has a much higher success rate than CCBS but can at times give suboptimal solutions. This is particularly true in dense scenarios with more conflicts.

VII. CONCLUSION AND FUTURE WORKS

Our work presents a novel extension of the PIBT algorithm that enables large-scale heterogeneous MAPF. This approach addresses a critical gap in the original PIBT which was limited to homogeneous agents with uniform speeds and sizes. By introducing a collision-checking-based reservation system, HetPIBT can successfully plan paths for hundreds of robots with different footprints and dynamics.

The results demonstrate that HetPIBT scales effectively with an increasing number of fleets handling up to 14 fleets each composed of 55 robots for a total of 770 agents in under a minute. Crucially, the overlapping cell-size factor has a significant impact on performance. For practical purposes, we recommend using the approach taken in E-PIBT [18] and limiting the number of agents that can be affected by a single agent's move.

While HetPIBT provides a viable solution for large-scale heterogeneous problems, the observed sub-optimality presents a clear avenue for future work.

Future research could focus on the following:

- **Improving Solution Optimality:** The sub-optimality of HetPIBT suggests a need for an optimizer. Integrating HetPIBT with a hierarchical planner, such as a large neighborhood search (LNS) or a similar approach, could help to improve solution quality while retaining the scalability of the base algorithm.
- **Alternative Prioritization Schemes:** The current priority scheme in HetPIBT is based on the distance to the

final goal. Exploring more sophisticated prioritization schemes could potentially address sub-optimality.

- **Data generation:** Using HetPIBT could be of interest to generate feasible trajectories for heterogeneous systems to seed ML based models.

REFERENCES

[1] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *Symposium on Combinatorial Search (SoCS)*, pp. 151–158, 2019.

[2] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artificial Intelligence*, p. 103752, 2022.

[3] K. Okumura, "Lacam: Search-based algorithm for quick multi-agent pathfinding," in *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

[4] D. Atzmon, Y. Zax, E. Kivity, L. Avitan, J. Morag, and A. Felner, "Generalizing multi-agent path finding for heterogeneous agents," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 11, no. 1, 2020, pp. 101–105.

[5] K. Okumura, Y. Tamura, and X. Défago, "winpibt: Extended prioritized algorithm for iterative multi-agent path finding," *arXiv preprint arXiv:1905.10149*, 2019.

[6] P. Surynek, "Problem compilation for multi-agent path finding: a survey." in *IJCAI*, 2022, pp. 5615–5622.

[7] P. Surynek, A. Felner, R. Stern, and E. Boyarski, "Sub-optimal sat-based approach to multi-agent path-finding problem," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 9, no. 1, 2018, pp. 99–105.

[8] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370214001386

[9] J. Li, W. Ruml, and S. Koenig, "Eecbs: A bounded-suboptimal search for multi-agent path finding," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 14, 2021, pp. 12 353–12 362.

[10] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.

[11] R. Veerapaneni, A. Tang, H. He, S. Zhao, V. Shah, Y. Cen, Z. Ji, G. Olin, J. Arrizabalaga, Y. Shaoul, *et al.*, "Conflict-based search as a protocol: A multi-agent motion planning protocol for heterogeneous agents, solvers, and independent tasks," *arXiv preprint arXiv:2510.00425*, 2025.

[12] A. Chakravarty, M. X. Grey, M. V. J. Muthugala, and M. R. Elara, "Time-ordered ad-hoc resource sharing for independent robotic agents," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 10 081–10 088.

[13] A. Skrynnik, A. Andreychuk, A. Borzilov, A. Chernyavskiy, K. Yakovlev, and A. Panov, "Pogema: A benchmark platform for cooperative multi-agent pathfinding," in *The Thirteenth International Conference on Learning Representations*, 2025.

[14] S.-H. Chan, Z. Chen, T. Guo, H. Zhang, Y. Zhang, D. Harabor, S. Koenig, C. Wu, and J. Yu, "The league of robot runners competition: Goals, designs, and implementation," in *ICAPS 2024 System's Demonstration track*, 2024.

[15] M. Bettini, R. Kortvelesy, J. Blumenkamp, and A. Prorok, "Vmas: A vectorized multi-agent simulator for collective robot learning," in *International Symposium on Distributed Autonomous Robotic Systems*. Springer, 2022, pp. 42–56.

[16] L. Heuer, L. Palmieri, A. Mannucci, S. Koenig, and M. Magnusson, "Benchmarking multi-robot coordination in realistic, unstructured human-shared environments," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14 541–14 547.

[17] K. Okumura, Y. Tamura, and X. Defago, "Time-independent planning for multiple moving agents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 299–11 307.

[18] E. Yukhnevich and A. Andreychuk, "Enhancing pibt via multi-action operations," *arXiv preprint arXiv:2511.09193*, 2025.