Anytime Single-Step MAPF Planning with Anytime PIBT

Nayesha Gandotra*[1], Rishi Veerapaneni*[1], Muhammad Suhail Saleem[1], Daniel Harabor[2], Jiaoyang Li[1], Maxim Likhachev[1]

[1] Carnegie Mellon University [2] Monash University

{nayeshag, rveerapa, msaleem2, jiaoyanl, mlikhach}@andrew.cmu.edu, daniel.harabor@monash.edu

Abstract

PIBT is a popular Multi-Agent Path Finding (MAPF) method at the core of many state-of-the-art MAPF methods including LaCAM, CS-PIBT, and WPPL. The main utility of PIBT is that it is a very fast and effective single-step MAPF solver and can return a collision-free single-step solution for hundreds of agents in less than a millisecond. However, the main drawback of PIBT is that it is extremely greedy in respect to its priorities and thus leads to poor solution quality. Additionally, PIBT cannot use all the planning time that might be available to it and returns the first solution it finds. We thus develop Anytime PIBT, which quickly finds a one-step solution identically to PIBT but then continuously improves the solution in an anytime manner. We prove that Anytime PIBT converges to the optimal solution given sufficient time. We experimentally validate that Anytime PIBT can rapidly improve single-step solution quality within milliseconds and even find the optimal single-step action. However, we interestingly find that improving the single-step solution quality does not have a significant effect on full-horizon solution costs.

1 Introduction

Efficient and collision-free navigation is a fundamental challenge for teams of robots operating in shared environments such as factory floors, warehouses, or autonomous vehicle hubs. The field of Multi-Agent Path Finding (MAPF) develops planning algorithms that enable multiple robots to move from their start locations to designated goals without colliding with or obstructing each other.

Priority Inheritance with Backtracking (PIBT) is a popular MAPF algorithm that has gained attention due to its simplicity, speed, and effectiveness (Okumura et al. 2022). PIBT is a one-step, greedy algorithm that quickly determines the next action for agents to execute and can generate a one-step plan for hundreds of agents in less than a millisecond. Due to its efficiency, PIBT has been integrated into various MAPF frameworks. For example, it is used for full-horizon planning in LaCAM (Okumura 2023b), collision-shielding for learned local MAPF policies in CS-PIBT (Veerapaneni et al. 2024b), and as the initial solution generator under tight runtime constraints in WPPL (Jiang et al. 2024).

While PIBT's greedy nature enables exceptional speed, it also introduces a key limitation: poor solution quality. In its current form, PIBT always computes a solution as quickly as possible (usually sub-millisecond), regardless of the available time budget which could range to hundreds or thousands of milliseconds. However, real-life applications typically have longer planning times. A common paradigm for such applications is "anytime" planning where a planner outputs an initial solution fast and then iteratively improves it until it reaches the timeout, where it then returns the best solution found. PIBT in its current form cannot do this and cannot take advantage of additional planning time to improve its solution.

To this extent, we propose Anytime PIBT, which quickly finds a single-step solution identical to PIBT and then continually refines the solution improving its quality with additional planning time. Anytime PIBT achieves this by searching through the all possible actions for agents in an efficient anytime manner.

Anytime PIBT utilizes two main insights. First, we interpret the single-step MAPF problem as a recursive problem where we can employ a Depth-First Search (DFS). This allows us to explore through all actions while saving intermediate solutions and pruning. Second, we exploit the semi-independent structure of MAPF problems. At any given time, not all agents interact or affect one another's actions. By identifying these interacting agents and grouping them, we break the joint action space into smaller, independent subspaces, making Anytime PIBT more efficient.

We theoretically prove that Anytime-PIBT algorithm converges to the optimal single-step solution given sufficient planning time. Additionally, we empirically evaluate its performance on both small and large problem instances (ranging from 20 to 1000 agents) and find that it can consistently improve the single-step cost. However, Anytime PIBT does not significantly improve full-horizon costs compared to PIBT when used with LaCAM and LaCAM*. Still, given the popularity of PIBT, we believe that Anytime PIBT has potential widespread use and promising future directions.

2 Preliminaries and Related Work

Multi-Agent Path Finding (MAPF) involves finding collision-free paths for a set of N agents, denoted as $i = 1, \ldots, N$, where each agent must travel from its start location s_i^{start} to its goal location s_i^{goal}. In the standard 2D MAPF setup, the environment is discretized into grid

*These authors contributed equally.

cells. Agents can move to adjacent cells in any cardinal direction or remain stationary in their current cell. A valid solution consists of a set of paths $\Pi = \{\pi_1, ..., \pi_N\}$ where $\pi_i^0 = s_i^{start}$, $\pi_i^T = s_i^{goal}$ with T representing the maximum timestep of all agents' paths. To ensure validity, the solution must avoid vertex collisions (when two agents occupy the same cell at the same timestep, i.e., $\pi_i^t = \pi_j^t$ for $i \neq j$) and edge collisions (when two agents swap positions between consecutive timesteps, i.e., $\pi_i^t = \pi_j^{t+1} \wedge \pi_i^{t+1} = \pi_j^t$). The standard objective in optimal MAPF is to find a solution Π that minimizes the total cost $|\Pi^{0:T}| = \sum_{i=1}^{N} |\pi_i^{0:T}| = \sum_{i=1}^{N} \sum_{t=0}^{T-1} c(s_i^t, s_i^{t+1})$. In this work, we assume every action to be of unit cost, $c(s_i^t, s_i^{t+1}) = 1$, except when an agent remains at its goal (in which case the cost is zero).

While the above describes the standard formulation of the full-horizon MAPF problem, this work focuses on a simplified variant: the single-step MAPF problem. Here, the objective is to determine the next single actions for agents that minimizes the total cost for all agents to reach their goals, under the assumption that after executing the first action, each agent follows its individual optimal path to the goal. The latter part ignores potential interactions with other agents and therefore is just each agent's individual optimal path to the goal. We note that all performant 2D MAPF methods compute a backward Dijkstra's for each agent where $h_i^*(s) = c^*(s, s_i^{goal})$. Thus our objective is to compute the first action for all agents that minimizes $\Pi^{0:1} = \sum_{i=1}^{N} (c(s_i^0, s_i^1) + h_i^*(s_i^1))$.

2.1 PIBT

PIBT is a single-step planner that starts out by assigning each agent a priority. It then sequentially plans agents (in descending order of priority) with each agent reserving its next action/location. If an agent R_i reserves a location that collides with the single-step path of a previously planned (i.e. higher priority) agent, R_i is not allowed to try that action and must try its next preferred action. If R_i instead moves into a lower priority agent R_k, the other lower priority agent R_k is required to plan to make way for R_i (hence "inherits" the higher priority). This priority inheritance repeats until an action is found. If R_k cannot compute a valid action, it then chooses to stay in place and requires R_i to reserve a different action. This "backtracking" behavior enables PIBT to remain effective in congested scenarios where agents must try out different actions.

PIBT is greedy in respect to the provided single-agent heuristic and agent priorities, e.g., the highest priority agent always prefers to minimize its heuristic, even if that would delay all the other agents. Thus PIBT is not single-step optimal, i.e., does not minimize $|\Pi^{0:1}|$ and can return arbitrarily bad solutions.

2.2 PIBT in Other Methods

PIBT is popular as it is an extremely fast single-step MAPF solver (< 1 millisecond for hundreds of agents). In particular, the LaCAM solvers, i.e., LaCAM (Okumura 2023b), LaCAM* (Okumura 2023a), Engineering LaCAM* (Okumura 2024), search over the joint configuration space and leverage PIBT as their fast joint configuration successor generator. The different methods search over the joint space in different manners with LaCAM using a Depth-First Search (DFS), LaCAM* using anytime A*, and Engineering LaCAM* employing many tools including parallelism. These methods all rely on PIBT for its speed.

PIBT's collision-free single-step behavior has also been used in CS-PIBT as a smart "collision shield" for post-processing learnt local MAPF policy predictions (Veerapaneni et al. 2024b). Since single-step learnt policies could make errors and predict actions that lead to collisions, they propose using PIBT informed by the model predictions to produce collision-free actions.

Traffic Flow (Chen et al. 2024) improves PIBT's performance in Lifelong MAPF (where agents are continually assigned new goals) by computing heuristics online informed by agent congestion. This decreases the greedy single-step behavior of PIBT and was shown to improve performance.

2.3 Anytime MAPF Algorithms

Anytime algorithms find initial solutions very fast and then improve them over time, allowing them to be queried at "anytime" where they return the best solution found so far. Anytime Focal search (Cohen et al. 2018) replaced the high-level focal search of BCBS (Barer et al. 2014) with an anytime focal search, allowing it to improve the solution quality over time. However, it is unclear how such anytime conflict-based methods would scale to 100s of agents where finding an initial solution itself could take seconds.

A more general popular approach for anytime behavior is to use Large Neighborhood Search (LNS) (Shaw 1998) in MAPF. LNS is a broad local search method that tries to find better solutions by taking an existing solution, destroying part of it, and then repairing it. MAPF-LNS (Li et al. 2021) and MAPF-LNS2 (Li et al. 2022) explore different algorithms for generating the initial MAPF solution, destroying, and repairing and showed significant improvement over Anytime EECBS. Using LNS however does not have any guarantees on reaching the optimal solution.

Anytime algorithms are particularly useful when the time cutoff is very small. The Robot Runners competition (Chan et al. 2024) required planning for 100s-1000s of agents in one second. The winning team developed Windowed Parallel PIBT-LNS (WPPL) which used PIBT on weighted graphs to obtain an initial windowed solution (of length 5-15 depending on the instance) and then parallel LNS to improve the solution (Jiang et al. 2024). Planning Interleaved with Execution (Zhang et al. 2024) improves an initial solution (found by LaCAM*) using LNS while simultaneously executing the current best-found path.

2.4 Disjoint Agent Groups

One core aspect of our proposed Anytime PIBT is planning over groups of agents rather than all agents together. This idea is inspired by two main works that utilize this idea.

Operator Decomposition (OD) (Standley 2010) uses independence detection to iteratively divide a MAPF problem of

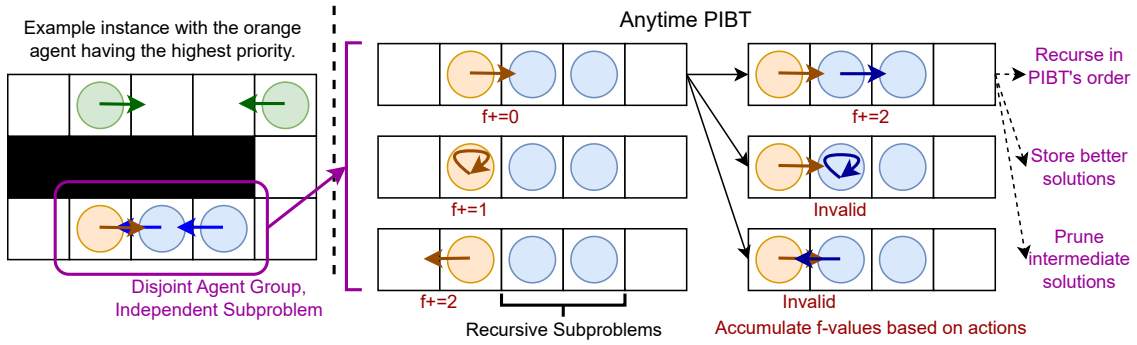


Figure 1: The left example shows six agents with preferred actions denoted in arrows. The orange agent with higher priority would push back the blue agents, when an optimal single-step plan would have the orange agent move back instead. Anytime PIBT first detects initial disjoint agent groups through an initial modified PIBT call. Then Anytime PIBT solves each group by recursing through possible actions and agents in PIBT's order. It stores encountered better solutions and prunes intermediate solutions based on the accumulated penalty.

all agents into a set of smaller MAPF problems $M < N$ where each smaller problem contains a set of conflicting agents. The WinC-MAPF framework formalizes this notion and defines Disjoint Agent Groups (DJAGs) where agents in different disjoint agent groups do not affect each other's solutions (Veerapaneni et al. 2024a).

Definition 1 (Disjoint Agent Groups)**.** *Given a configuration transition $s_{1:N} \rightarrow s'_{1:N}$, and a set of disjoint agent groups $\{Gr^i\}$, we have the property that for each agent R_j with transition $s_j \rightarrow s'_j$ in disjoint agent group Gr^i, there cannot exist another agent in a different group Gr^k that blocked R_j from picking a better path.*

The main implication of DJAGs is that we can plan for each DJAG separately (as agents in different groups do not affect each other). We use this idea in Anytime PIBT and detect and solve DJAGs, which significantly improves the anytime behavior.

Additionally, finding the optimal solution for each DJAG ensures that we find the global optimal solution (as the solution for each DJAG captures interactions between agents internally, but agents in different DJAGs do not interact with each other). Thus, individually finding, optimizing, and recalculating DJAGs (if agents have new conflicts when replanning) guarantees eventually finding the optimal solution (see Theorem 1).

3 Anytime PIBT

Anytime PIBT utilizes two main insights. First, as depicted in the middle of Figure 1, we view the single-step MAPF as a recursive problem; planning for N agents means assigning a location for one agent and then planning for the rest $N-1$ agents. Given N agents, each with 5 actions, this means we could consider all 5^N options through our recursive tree. We then save intermediate solutions in the tree, and can prune out intermediate branches whose solution is worse than our best found so far. We use PIBT's priority inheritance and backtracking to order our recursive calls. Thus, Anytime-PIBT at its core is a DFS through the action tree that employs standard solution saving and pruning.

Algorithm 1: Anytime PIBT

Parameters: Current states $s^t_{1:N}$, Agent Priorities $AP_{1:N}$, Timeout T_{out}
Output: $\pi^{t+1}_{1:N}$

procedure AnytimePIBT($s^t_{1:N}$, $AP_{1:N}$, T_{out})
  $\pi^{t+1}_{1:N}$ = Null, GroupQ = $\emptyset$
  PIBTwithGrouping($s^{1:N}$, $AP_{1:N}$) ▷ *Populates $\pi^{t+1}_{1:N}$ and GroupQ*
  $\pi^{*t+1}_{1:N} = \pi^{t+1}_{1:N}$ ▷ *Current best solution*
  while !GroupQ.empty() and not timeout **do**
    Gr = GroupQ.pop() ▷ *Group class, see Alg 3*
    T_{group} = TimePerGroup($|Gr.AoP|$, GroupQ)
    $\pi^{t+1}_{\forall i \in Gr.AoP}$ = Null ▷ *Clear for replanning*
    AnytimePIBT-R(Gr, $Gr.AoP$, Null, $s^t_{1:N}$, 0, T_{group}) ▷ *Alg 3, updates $\pi^{*t+1}_{i \in Gr}$, earlyExit, NewGroup*
    if NewGroup $\neq Gr$ **then**
      GroupQ.removeNotDisjointWith(NewGroup)
      GroupQ.push(NewGroup)
    else if earlyExit **then** ▷ *Did not finish group*
      GroupQ.push(Gr) ▷ *We can revisit this later if given enough time*
  return $\pi^{*t+1}_{1:N}$

function PIBTwithGrouping($s^t_{1:N}$, $AP_{1:N}$)
  for agent $k \in$ argsort($AP_{1:N}$) **do**
    if π^{t+1}_k is Null **then** ▷ *If no move planned*
      PIBT-Gr(k, $s^t_{1:N}$) ▷ *Alg 2, updates $\pi^t_{1:N}$*

function TimePerGroup(K, GroupQ)
  totalAgentsInGroups = $\sum_{Gr^i \in \text{GroupQ}} |Gr^i.AoP|$
  return getTimeLeft()$*K/$totalAgentsInGroups

However, this recursive structure disregards the semi-independence of agents and requires recursing through / resolving identical subproblems repeatedly. Thus, our second insight is to use the concept of disjoint agent groups from

Algorithm 2: PIBT with Grouping Recursive Function, red denotes modifications to regular PIBT

Parameters: Agent k, Current states $s^t_{1:N}$
Globals and Side Effects: $\pi^{t+1}_{1:N}$ being populated, Group() updates disjoint agent groups

1: **procedure** PIBT-Gr(k, $s^t_{1:N}$)
2: **for** $s^{t+1}_k \in$ Neighbors(s^t_k) **do** ▷ *Sort by* $c(s^t_k, s^{t+1}_k) + h^*(s^{t+1}_k)$
3: **if** s^{t+1}_k is invalid **then** ▷ *Ignore obstacles*
4: **continue**
5: **if** $\exists j$ s.t. $\pi^{t+1}_j = s^{t+1}_k$ or $\pi^{t+1}_j = s^t_k \wedge \pi^t_j = s^{t+1}_k$ **then** ▷ *Vertex or Edge collision*
6: Group(k, j)
7: **continue**
8: $\pi^{t+1}_k = s^{t+1}_k$
9: **if** $\exists$ agent $j \neq k$ with $s^t_j = s^{t+1}_k$ **then**
10: Group(k, j)
11: **if** PIBT-Gr(j, $s^t_{1:N}$) **then**
12: **return** Success
13: π^{t+1}_k =Null
14: **else**
15: **return** Success
16: **return** Failure

the WinC-MAPF framework (Veerapaneni et al. 2024a) and decompose the single-step MAPF problem into smaller disjoint agent groups (as seen in left of Figure 1). This enables us to scale to more agents and have better anytime performance. Given these two insights (high-level DFS and grouping), we now describe Anytime PIBT using the pseudocode in Algs 1, 2, and 3.

Initial PIBT with Grouping Call Anytime PIBT (Alg 1) starts with a modified PIBT call (line 3) that keeps track of agents that interact with each other and define disjoint agent groups. The only modification to regular PIBT is that the recursive helper PIBT call (PIBT-Gr, Alg 2) groups agent k when its action is blocked by a higher priority agent j (line 5) or when agent k bumps into another agent j (line 9). Agents k and j with such interactions belong to the same group as they affect each other's ability to pick their best single-step path. On the flip side, agents without such interactions are not in the same agent group.

Note that our grouping scheme might contain extra agents (e.g., k bumping into j does not necessarily mean j blocks k if they both move in the same direction). However, this is still a valid disjoint agent group and would be fast to plan for later on as these agents move on their optimal actions (and thus we would find the optimal solution immediately and would spend negligible time planning for these agents).

The "Group" function is a simple union that merges agents that have been grouped. We highlight that grouping is just bookkeeping and adds negligible time. Thus after the end of the initial PIBT call, we have an initial solution identical to PIBT, as well as disjoint groups of agents. In Figure 1, this would lead the bottom three agents to be in a single group and the green agents to not be in any group.

Algorithm 3: Anytime PIBT Recursive Function

Parameters: Group Gr, Current States $s^t_{1:N}$, Timeout T_{out}
Group Class: Gr has agents $Gr.AoP$ and best f-value $Gr.F_b$ (default ∞)
Globals and Side Effects: Preplanned $\pi^{t:t+1}_{\forall i \notin Gr.AoP}$, updates Paths $\pi^{t+1}_{\forall i \in Gr.AoP}$, Best solution $\pi^{*t+1}_{1:N}$, "earlyExit" (default True), and "NewGroup" (defaults $Gr.AoP$)

1: **procedure** AnytimePIBT-R(Gr, Current AoP, Agent k, $s^t_{1:N}$, Accumulated f-costs F_c, T_{out})
2: **if** k is Null **then** ▷ *If no particular agent to plan*
3: $k = AoP$.top() ▷ *Pick from AoP by priority*
4: $AoP = AoP \setminus k$ ▷ *Less agents for recursive call*
5: **for** $s^{t+1}_k \in$ Neighbors(s^t_k) **do** ▷ *Sort by* $c(s^t_k, s^{t+1}_k) + h^*(s^{t+1}_k)$
6: **if** getCurrentPlanningTime() $> T_{out}$ **then**
7: **return** ▷ *Time cutoff, set earlyExit to True*
8: **if** s^{t+1}_k is invalid **then**
9: **continue**
10: **if** $\exists j$ s.t. $\pi^{t+1}_j = s^{t+1}_k$ or $\pi^{t+1}_j = s^t_k \wedge \pi^t_j = s^{t+1}_k$ **then** ▷ *Vertex or Edge collision*
11: Group(k, j) ▷ *Updates NewGroup*
12: **continue**
13: $F_{next} = F_c + c(s^t_k, s^{t+1}_k) + h^*(s^{t+1}_k)$
14: **if** $F_{next} \geq Gr.F_b$ **then** ▷ *Prune if not better*
15: **return**
16: $\pi^{t+1}_k = s^{t+1}_k$
17: **if** AoP is $\emptyset$ **then** ▷ *If all agents planned*
18: $Gr.F_b = F_{next}$; $\pi^{*t+1}_{\forall i \in Gr.AoP} = \pi^{t+1}_{\forall i \in Gr.AoP}$
19: **else**
20: **if** $\exists$ agent $j \neq k$ with $s^t_j = s^{t+1}_k$ **then**
21: Group(k, j) ▷ *Updates NewGroup*
22: nextAgent = j ▷ *Priority inheritance*
23: **else**
24: nextAgent = Null
25: AnytimePIBT-R(Gr, AoP, nextAgent, $s^t_{1:N}$, F_{next}, T_{out})
26: π^{t+1}_k =Null

3.1 Solving a Group with Anytime PIBT

After generating the groups, we then iterate through each group and call AnytimePIBT-R (Alg 3) on the group. AnytimePIBT-R at its core is a DFS that goes through all agents in the group and their possible actions. We first describe it without the notion of grouping, e.g., imagine that the group with corresponding agents to plan (AoP) contains all agents. We will revisit grouping afterwards.

Similar to PIBT-Gr, AnytimePIBT-R starts by going through the states for agent k (line 5) and ignores invalid states (line 8) or states that conflict with previously planned agents (line 10).

One main difference is that it accumulates the f-value of all planned agents by adding the $c(s^k, s') + h^*(s')$ of the chosen state to the current accumulated sum (line 13). If this f-value is greater or equal than the current best solution's F_b, we can immediately prune the rest of the DFS branch as we are guaranteed that moving future agents will only

increase the f-value (line 14). One subtlety is that we sorted s' by increasing $c(s^k, s') + h^*(s')$, and thus can prune out the following neighbor states s'' with larger $c(s^k, s'') + h^*(s'')$, allowing us to *return* as opposed to *continue*.

If we do not prune, we update π_k^{t+1} accordingly (line 16). If we have finished planning for all agents, we are guaranteed that the accumulated f-value is smaller than F_b (since it was not pruned before) and we thus update the best solution and f-value (line 18).

If we have not finished planning and bump into another agent, we choose that to be the next agent to plan (line 20) following PIBT's priority inheritance logic. We then recurse (line 25) which goes through the other agents and their action. Note if we did encounter another agent, the recursive call will pick the next (highest priority) agent in AoP.

Finally, we reset π_k^{t+1} and proceed to consider the next action. We repeat this logic for all actions and therefore consider all possibilities of movements for each agent.

In summary, Anytime PIBT is a DFS that systematically goes through agents and their actions while keeping track of f-values and saving or pruning solutions accordingly. The ability to stop the DFS at anytime (line 6) allows Anytime PIBT to return the best solution found at any time.

3.2 Updating Groups on the Fly

Anytime PIBT described so far searches over actions for agents in the group while having the rest of the agents move following their original PIBT solution. Thus, an agent $k \in Gr.AoP$ (where $Gr.AoP$ are the agents in the k's group Gr) cannot conflict with an agent $j \notin Gr.AoP$ inside AnytimePIBT-R, as denoted by the preplanned $\pi^{t:t+1}_{\forall i \notin Gr.AoP}$ in Alg 3 "Globals". Put in other words, when planning for $Gr.AoP$, the agents not in $Gr.AoP$ are assumed to be following their existing paths and must be avoided.

This can introduce suboptimality issues. For example it is possible that an agent k and j are initially in different groups. However, during the Anytime PIBT call to the group containing k, k could consider an action that conflicts with $\pi_j^{t:t+1}$. This action would not be pursued (Alg 3, line 10). Likewise, planning for j could have the same issue. This means that our planning procedure is missing possible actions of considering both agent k and j moving informed by each other. On the flip side, if k and j were in the same group, a joint movement would not be pruned and instead be considered through the recursive logic. Thus, to maintain optimality, we must update our disjoint agent groups so that k and j are grouped together. Note this in turn means that their respective groups should be merged too.

We do this by keeping track of the grouped agents in AnytimePIBT-R (line 10). However, we do not terminate early and instead continue the current AnytimePIBT-R call to make progress and encounter other potential agents. Then in AnytimePIBT after AnytimePIBT-R return (Alg 1, line 9), AnytimePIBT adds a new found group (if it exists) and removes non-disjoint groups (e.g., the old groups that just got merged into the new group). This group (i.e., its corresponding agents) will be replanned later on.

Diving Time Across Groups Given a set of groups to solve, a naive anytime implementation would allocate T_{out} to each AnytimePIBT-R grouped call and terminate once the cumulative time exceeds T_{out}. However, this would prioritize earlier groups. Thus in Alg 1 (line 7) an additional optimization is to allocate time based on the group's size. Other methods are possible but we found this to work well.

3.3 Theoretical Properties

Lemma 1. *Given a sufficiently large timeout T_{out}, AnytimePIBT-R finds the optimal single-step solution $\pi^{t+1}_{\forall i \in Gr.AoP}$ for agents in $Gr.AoP$ given fixed $\pi^{t:t+1}_{\forall i \notin Gr.AoP}$ that should not be conflicted with.*

Proof. Anytime PIBT considers all valid transitions for each agent except those that conflict with previously planned agents (lines 10) or are pruned (lines 14). Transitions conflicting with previously planned agents (populated by $\pi^{t:t+1}_{i \notin Gr.AoP}$ as well as agents planned in the current recursive call) can be safely skipped. In respect to pruning, since $c(s_k^t, s_k^{t+1}) + h(s_k^{t+1}) \geq 0$ for all possible s_k^t, s_k^{t+1} and agents k, if an intermediate call has $F_c \geq F_{best}$, it will never lead to a better solution and thus can be pruned. Therefore Anytime PIBT will eventually find the optimal single-step solution $\pi^{t+1}_{\forall i \in Gr.AoP}$ satisfying it does not conflict with $\pi^{t:t+1}_{\forall i \notin Gr.AoP}$. □

Theorem 1. *Given a sufficiently large timeout T_{out}, Anytime PIBT will eventually find the optimal single-step solution for all agents.*

Proof. Suppose Anytime PIBT does not find the optimal single-step solution. Then there must exist some agent k whose chosen action is worse than its action in the optimal solution. According to Lemma 1, AnytimePIBT-R will optimally plan agents in $Gr.AoP$ given fixed $\pi^{t:t+1}_{\forall i \notin Gr.AoP}$, so it must be the case that k's optimal action is blocked by $\pi_j^{t:t+1}$ of some $j \notin Gr.AoP$.

Since $Gr.AoP$ is defined by a disjoint agent group, this means that k and j are in different disjoint agent groups. However, j would be detected in AnytimePIBT-R (Alg 3, line 10) and k, j would be grouped together. Then, Anytime PIBT would plan for this new group (Alg 1, line 10), which contradicts the earlier statement of k and j being in different disjoint agent groups. Thus, AnytimePIBT will eventually find the optimal single-step solution. □

3.4 Anytime PIBT Tiebreak

As we will see in the next section, Anytime PIBT actually has a worse success rate than PIBT as its optimal one-step action means that higher priority agents may not push away lower priority agents and can get stuck. Therefore, we implemented a variant where agents only consider their actions that have the best individual f-value, i.e., we search over tiebreaking between agent's best actions. We call this variant Anytime PIBT Tiebreak. This does not ensure global optimality but does retain regular PIBT properties (e.g., that the highest priority agent will make progress towards its goal at every step if the path to the goal is biconnected).

4 Experiments

We evaluate Anytime PIBT on using the standard MAPF benchmark maps (Stern et al. 2019) with varying numbers of agents and anytime cutoffs. Since Anytime PIBT is a drop in replacement for PIBT, we also evaluate its effect in LaCAM and LaCAM* which both internally use PIBT. All instances were run with a 60 second cutoff time (i.e., sum of all planning times across all iterations).

Our experiments aimed to answer two questions. First, how much does Anytime PIBT improve single-step performance compared to PIBT? Second, how much does Anytime PIBT improve full horizon planning compared to using PIBT (e.g., by itself or in LaCAM, LaCAM*)? Our results find that we perform quite well in finding single-step solutions and many times can find optimal single-step solutions. Interestingly, we see that this however does not translate to any meaningful full horizon solution quality gains.

4.1 Single-Step Solutions

Figure 2 shows an example run of Anytime PIBT with 1 second per-timestep cutoff on map den520d with 500 agents. At every timestep, we record and plot the initial normalized f-value (blue) and the final normalized f-value after Anytime PIBT finishes (orange). Note that the initial value (blue) is the result of the initial PIBT call. The normalized f-value subtracts out the lowerbound f-value (i.e., the sum of each agent's individual minimum f-value), so a value of zero means that all agents are going on their individual optimal action. We see that Anytime PIBT is able to consistently improve the f-value compared to the initial (PIBT) result.

Additionally, in this scenario, Anytime PIBT never reached the 1 second cutoff time, which means that it was able to find the optimal single-step solution for every timestep.

Figure 3 shows a histogram of the per-timestep across different Anytime PIBT deadlines in *milliseconds* for 25 scenes on den520d with 500 agents. The x-axis denotes PIBT's f-value minus Anytime PIBT's f-value, e.g. a change of 4 means that Anytime PIBT improved (decreased) the f-value by 4 compared to PIBT's initial solution. We see that Anytime PIBT can make a non-trivial improvement with 0.1 milliseconds (red) and has more substantial single-step improvements within more time (purple, green). We generally saw similar patterns in Figures 2 and 3 across the different maps and agents with Anytime PIBT having larger impacts on more crowded instances.

We found that grouping was a crucial part to Anytime PIBT's fast performance. Without grouping, Anytime PIBT's number of recursive calls increases exponentially with the number of agents such that in most cases the algorithm would not finish planning a single-step for 100 agents within 1 minute. This was due to the fact that without agent groups, finding the optimal action for an agent low in the priority list requires recursing through all the possible actions for all the agents above it in the priority list, which is highly inefficient if the higher agents are unrelated to the lower one. Hence, our group logic is critical for Anytime PIBT efficiency.

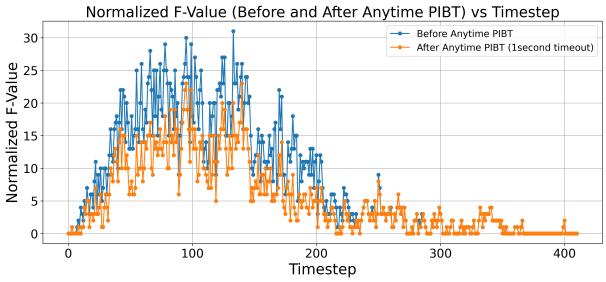


Figure 2: We visualize the per timestep normalized f-values for 500 agents on map den520d. The normalized f-value is the solution f-value minus the lower-bound f-value (e.g. sum of each agent's best action). At every timestep, we plot the initial PIBT solution value (blue) and the solution after running Anytime PIBT for 1 second (orange).

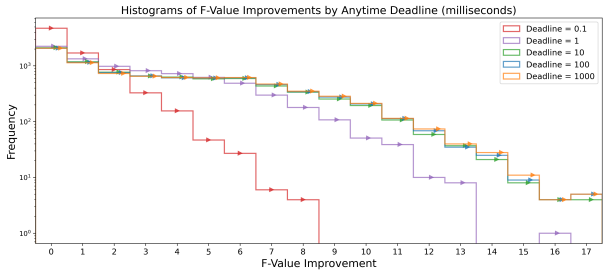


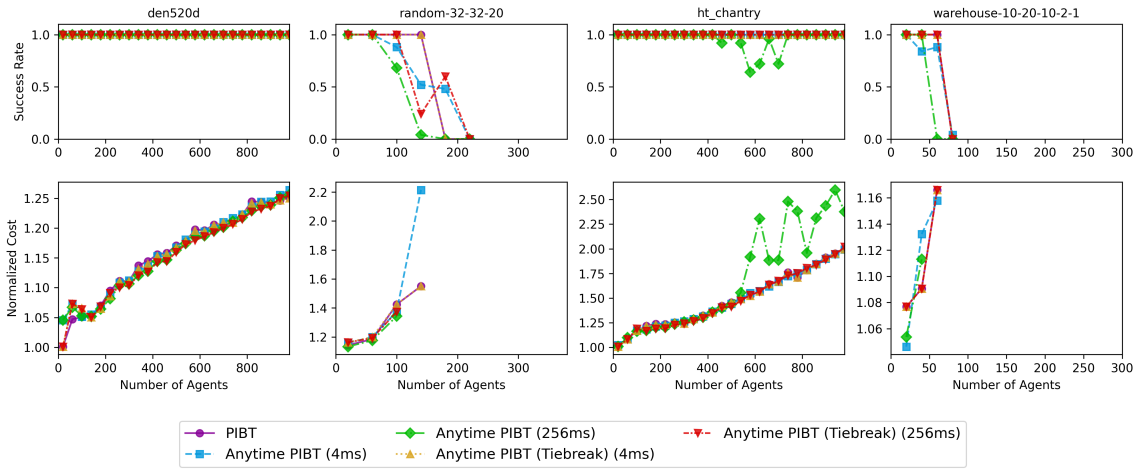
Figure 3: We run Anytime PIBT and store the f-value improvements compared to PIBT's initial solution at different timeouts. Deadlines are in additional *milliseconds* to the initial PIBT call (e.g., deadline of 0 is identical to PIBT).

4.2 Full-Horizon Solutions

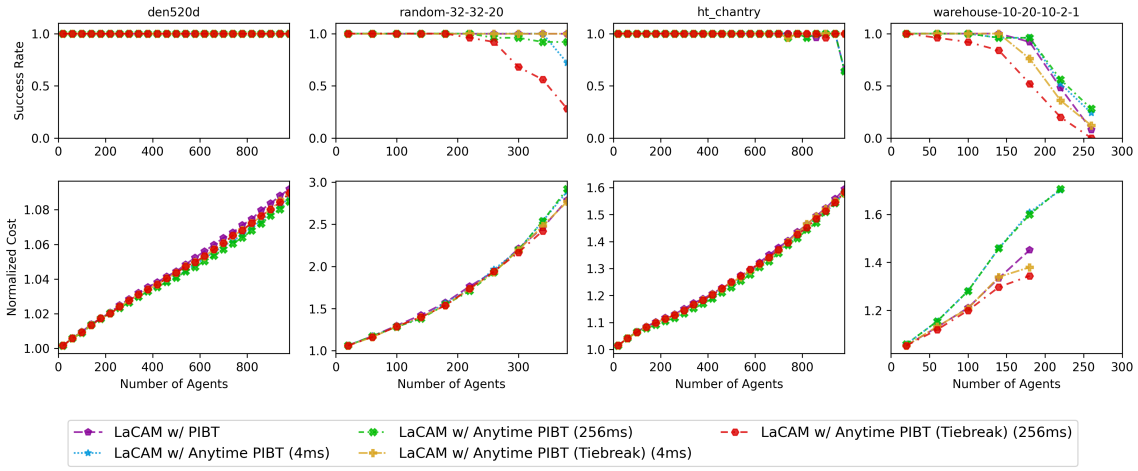
Figure 4 shows the performance of Anytime PIBT vs PIBT across a variety of maps as standalone planners along as a configuration generator in LaCAM and LaCAM*. For each variant (standalone, in LaCAM, in LaCAM*), we compare using PIBT, normal Anytime PIBT with a 4 or 256 ms per-step cutoff, and Anytime PIBT Tiebreak with identical cutoffs. We plot the mean normalized cost (total cost divided by lower bound) for instances with a 50+% success rate.

PIBT vs Anytime PIBT Comparing PIBT and Anytime PIBT in Fig 4a, we see that Anytime PIBT has a perfect success rate on den520d and ht_chantry, but worse performance than PIBT on random-32-32-20 and warehouse-10-20-10-2-1. This reveals that PIBT's greedy nature actually helps performance compared to Anytime PIBT. For example, when a low priority agent is blocking a high priority agent in a corridor, Anytime PIBT could find the optimal single-step solution which is for both agents to wait (in deadlock), whereas PIBT will greedily drive the higher priority agent forward. Anytime PIBT Tiebreak, on the other hand, does not suffer from this issue and has similar success rate as PIBT.

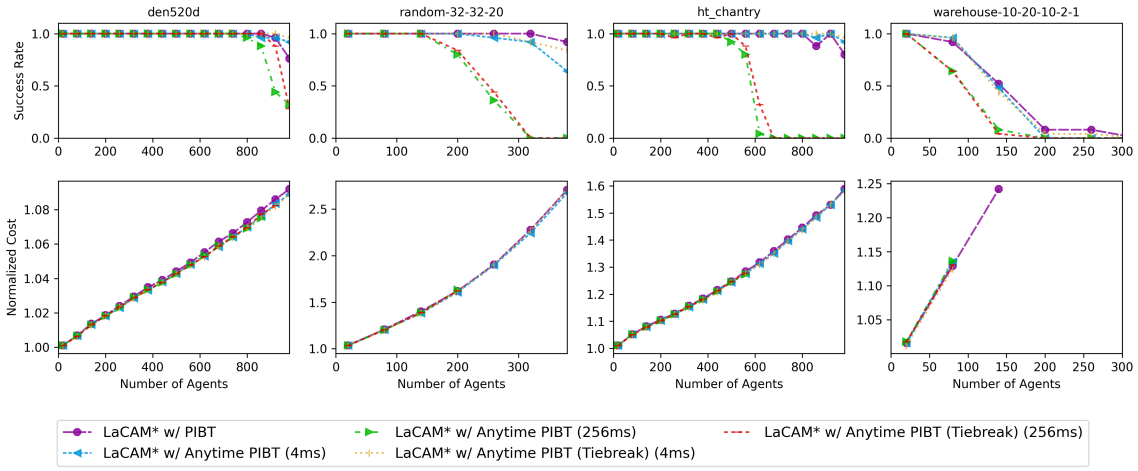
We additionally see that Anytime PIBT (with and without tiebreaking) is able to have small but noticeable improvement on overall solution quality. Even though this change is small, it is possible that this is the best that a single-step planner can do (without additional information) as the per timestep improvements seen in Fig 3 are on the order of 1-17 for 500 agents when the total per timestep cost is on the



(a) Standalone PIBT vs Anytime PIBT



(b) PIBT vs Anytime PIBT in LaCAM



(c) PIBT vs Anytime PIBT in LaCAM*

Figure 4: Evaluating PIBT vs Anytime PIBT

order of 500 (so we would expect only few percentage improvement based on this data).

Anytime PIBT in LaCAM(*) We first focus on LaCAM results (Fig 4b). We observe that the solution quality are

again slightly better with Anytime PIBT, especially with Tiebreaking. However, performance is noticeable worse in warehouse-10-20-10-2-1. We hypothesize this occurs as more agents are stuck in deadlock and accumulate cost with Anytime PIBT as the action generator (which is reflected in the standalone Anytime PIBT performance).

We now focus on LaCAM* results (Fig 4c). Since LaCAM* is an anytime solver, using Anytime PIBT as a configuration generator means that we now tradeoff evaluating many "poorer" quality high-level nodes or evaluating fewer "higher" quality high-level nodes. For example, with a 60 second timeout and assuming the configuration generator takes a full 256 milliseconds per call, LaCAM* would generate at most 60/0.256 = 234 high-level nodes. LACAM* with a 4 millisecond timeout could generate 15,000 nodes instead. This discrepency explains why increasing the Anytime PIBT cutoff decreases the success rate in the more congested instances (e.g., ht_chantry). Looking closely at the cost, Anytime PIBT does decrease the cost slightly for the first three maps and marginally increases it for the last. However, overall the change in cost is small.

The overall lack of performance gain of Anytime PIBT in LaCAM and LaCAM* was quite unexpected, and implies a large gap between single-step solvers and the LaCAM methods. In particular, since LaCAM is very dependent on its single-step configuration generator, common intuition would think that improving the single-step action configuration would improve overall performance. Instead, even though Anytime PIBT consistently increases single-step performances over PIBT (Section 4.1), this does not translate to gains with the LaCAM methods.

5 Conclusion and Future Work

PIBT is an extremely fast, scalable, and popular single-step solver. However, PIBT's speed comes at the expense of solution quality. An ideal anytime single-step solver would be as fast as PIBT but improve its solution over time. Anytime PIBT does exactly this.

Anytime PIBT interprets single-step MAPF as a recursive problem and therefore uses an anytime DFS to search through agents' actions. Anytime PIBT is able to save intermediate solutions and efficiently prune branches. Additionally, Anytime PIBT utilizes disjoint agent groups to decompose the entire recursive problem into smaller problems, and is able to dynamically update groups and eventually find the optimal single-step solution.

Experimentally, we find that Anytime PIBT is indeed able to improve single-step costs with milliseconds, and can even find the optimal single-step solution in many instances. However, counterintuitively these single-step improvements do not manifest themselves in full-horizon solutions when using Anytime PIBT inside LaCAM or LaCAM*. Still, we believe that Anytime PIBT is important as the first efficient anytime and eventually optimal single-step algorithm and see several avenues for future work.

Anytime PIBT and OD: One interesting observation to OD (Standley 2010) is that Anytime PIBT could interpreted as an anytime version of OD that uses a depth first search and priority inheritance instead of an A* search. This perspective interprets a single recursive call of PIBT as generating an intermediate OD state moving only a single agent. PIBT itself is then searching using OD and a greedy depth first search, and is intelligently picking the next agent to expand using priority inheritance. Anytime PIBT's disjoint agent groups could also be interpreted as a version of OD's independence detection. The connection between OD and PIBT has not been described before and future work could benefit by further exploring this perspective. In particular, OD uses A*, PIBT uses DFS, while Anytime PIBT uses anytime DFS with pruning. It is possible that a different search algorithm using this perspective could be promising work.

Integration with group cost functions: MAPF algorithms like CBS (Sharon et al. 2015) and PIBT (and therefore methods that build off them like EECBS (Li, Ruml, and Koenig 2021) and LaCAM*) only natively support individual agent costs. Thus, incorporating a cost that is a function of the configuration of several agents (e.g., a formation cost) is not easily feasible in their frameworks. Since Anytime PIBT efficiently searches the joint configuration, it would be possible to modify Anytime PIBT to incorporate and search over these group costs.

Single-Step Cost vs Full-Horizon: As seen in our experimental results, although we obtain single-step cost improvements, our overall full-horizon solution is not significantly better. Thus, broadly investigating how different single-step solvers and their properties affect full-horizon planning (e.g., in LaCAM) would be fruitful work. Another avenue is to incorporate Anytime-PIBT with works such as (Chen et al. 2024) which explicitly compute heuristics online that incorporate congestion to avoid deadlock in the future.

Overall, Anytime PIBT is a promising single-step anytime algorithm with many potential future applications and extensions.

References

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Seventh Annual Symposium on Combinatorial Search*.

Chan, S.-H.; Chen, Z.; Guo, T.; Zhang, H.; Zhang, Y.; Harabor, D.; Koenig, S.; Wu, C.; and Yu, J. 2024. The League of Robot Runners Competition: Goals, Designs, and Implementation. In *ICAPS 2024 System's Demonstration track*.

Chen, Z.; Harabor, D.; Li, J.; and Stuckey, P. J. 2024. Traffic Flow Optimisation for Lifelong Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(18): 20674–20682.

Cohen, L.; Greco, M.; Ma, H.; Hernández, C.; Felner, A.; Kumar, T. K. S.; and Koenig, S. 2018. Anytime Focal Search with Applications. In Lang, J., ed., *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 1434–1441. ijcai.org.

Jiang, H.; Zhang, Y.; Veerapaneni, R.; and Li, J. 2024. Scaling Lifelong Multi-Agent Path Finding to More Realistic

Settings: Research Challenges and Opportunities. In *Proceedings of the International Symposium on Combinatorial Search*, volume 17, 234–242.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021. Anytime Multi-Agent Path Finding via Large Neighborhood Search. In Dignum, F.; Lomuscio, A.; Endriss, U.; and Nowé, A., eds., *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, 1581–1583. ACM.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9): 10256–10265.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(14): 12353–12362.

Okumura, K. 2023a. Improving LaCAM for Scalable Eventually Optimal Multi-Agent Pathfinding. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, 243–251. ijcai.org.

Okumura, K. 2023b. LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(10): 11655–11662.

Okumura, K. 2024. Engineering LaCAM*: Towards Real-time, Large-scale, and Near-optimal Multi-agent Pathfinding. In Dastani, M.; Sichman, J. S.; Alechina, N.; and Dignum, V., eds., *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024*, 1501–1509. International Foundation for Autonomous Agents and Multiagent Systems / ACM.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310: 103752.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219: 40–66.

Shaw, P. 1998. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In Maher, M. J.; and Puget, J., eds., *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings*, volume 1520 of *Lecture Notes in Computer Science*, 417–431. Springer.

Standley, T. S. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. In Fox, M.; and Poole, D., eds., *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, 173–178. AAAI Press.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. *Symposium on Combinatorial Search (SoCS)*, 151–158.

Veerapaneni, R.; Saleem, M. S.; Li, J.; and Likhachev, M. 2024a. Windowed MAPF with Completeness Guarantees. arXiv:2410.01798.

Veerapaneni, R.; Wang, Q.; Ren, K.; Jakobsson, A.; Li, J.; and Likhachev, M. 2024b. Improving Learnt Local MAPF Policies with Heuristic Search. *International Conference on Automated Planning and Scheduling*, 34(1): 597–606.

Zhang, Y.; Chen, Z.; Harabor, D.; Bodic, P. L.; and Stuckey, P. J. 2024. Planning and Execution in Multi-Agent Path Finding: Models and Algorithms. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34(1): 707–715.

A Quick Summary

Recommended background readings: Readers new to PIBT or LaCAM should read Okumura et al. (2022) or Okumura (2023b) respectively. Readers new to Disjoint Agent Groups are recommended to read Veerapaneni et al. (2024a).

Motivation in respect to prior work: PIBT is an extremely fast and effective single-step planner that is used in several other methods (e.g., LaCAM). However, PIBT is very greedy (which leads to poor solution quality) and cannot leverage extra planning time as it returns the first solution it finds. Ideally, we would have an "anytime" version of PIBT that can leverage additional planning time to improve its solution cost. Our goal is to do exactly this.

A.1 Intended Takeaways

Our main contribution is designing Anytime PIBT, which finds the same initial solution as PIBT but then uses additional planning time to improve costs. We prove how Anytime PIBT will eventually find the optimal single-step solution, and empirically show how it can indeed do so even with hundreds of agents on certain instances. Anytime PIBT has a few algorithmic/theoretical insights and interesting experimental results which are our main takeaways:

1. Viewing single-step MAPF as a recursive process where planning for N agents means we can plan for 1 one and get a smaller $N-1$ single-step MAPF instance. From this perspective, PIBT employs a DFS that returns the first solution it found. Thus, we can get anytime behavior in Anytime PIBT by employing an Anytime DFS.

2. The problem with the recursive perspective is that it grows exponentially with the number of agents, and thus performs poorly in practice. Our second key insight is that we can decompose our MAPF problem into groups of independent agents (formally defined as disjoint agent groups) and solve those separately. This makes Anytime PIBT run significantly faster.

3. Anytime PIBT will provably find single-step optimal actions given sufficient time (and in practice can indeed find single-step optimal actions in less than a second).

4. Anytime PIBT has a consistent very small improvement in solution costs (e.g., less than 2 percent) when used for full horizon by itself or with LaCAM and LaCAM*. This is interesting as PIBT is typically thought of as a very greedy single-step planning, so our intuition is that improving the single-step solution cost would have a larger impact on overall costs.

5. Finding optimal single-step paths can lead to poorer success rates as the optimal single-step solution as agents resting on their goal can become obstacles as pushing from their goal increases the cost. Another perspective is that PIBT's behavior of high priority agents pushing through low priority agents is long term beneficial despite it having high single-step solution costs. We therefore tested an Anytime-PIBT with tiebreaking that maintains priorities and only checks through agents best actions, which does result in a similar success rate to PIBT while still improving costs.