db-LaCAM: Fast and Scalable Multi-Robot Kinodynamic Motion Planning with Discontinuity-Bounded Search and Lightweight MAPF

Akmaral Moldagalieva[1], Keisuke Okumura[2,3], Amanda Prorok[2], Wolfgang Hönig[1]

[1]Technical University of Berlin, Germany
[2]University of Cambridge, UK
[3]National Institute of Advanced Industrial Science and Technology (AIST), Japan
moldagalieva@tu-berlin.de, {ko393,asp45}@cst.cam.ac.uk, hoenig@tu-berlin.de

Abstract

State-of-the-art multi-robot kinodynamic motion planners struggle to handle more than a few robots due to high computational burden, which limits their scalability and results in slow planning time. In this work, we combine the scalability and speed of modern multi-agent path finding (MAPF) algorithms with the dynamic-awareness of kinodynamic planners to address these limitations. To this end, we propose discontinuity-Bounded LaCAM (db-LaCAM), a planner that utilizes a precomputed set of motion primitives that respect robot dynamics to generate horizon-length motion sequences, while allowing a user-defined discontinuity between successive motions. The planner db-LaCAM is resolution-complete with respect to motion primitives and supports arbitrary robot dynamics. Extensive experiments demonstrate that db-LaCAM scales efficiently to scenarios with up to 50 robots, achieving up to ten times faster runtime compared to state-of-the-art planners, while maintaining comparable solution quality. The approach is validated in both 2D and 3D environments with dynamics such as the unicycle and 3D double integrator. We demonstrate the safe execution of trajectories planned with db-LaCAM in two distinct physical experiments involving teams of flying robots and car-with-trailer robots.

1 Introduction

Kinodynamic motion planning addresses the problem of finding collision-free trajectories that are dynamically feasible between start and goal states. This formulation makes it more challenging than geometric planning since it requires accounting for robot dynamics and actuation limits. For instance, when car dynamics are neglected, the resulting state sequences may be physically infeasible. A car cannot rotate in place and must translate while steering to change its heading, making arbitrary state transitions impossible.

In multi-robot settings, the planner must not only enforce each robot's dynamics, but also ensure collision avoidance among robots. The problem is more challenging if robots operate in close proximity to each other in a cluttered space. To date, various approaches have been proposed to incorporate robot dynamics into multi-robot planning (Kottinger, Almagor, and Lahijanian 2022; Tajbakhsh, Biegler, and Johnson 2024; Moldagalieva et al. 2024). Although these approaches can handle robots with complex dynamics (Moldagalieva, Ortiz-Haro, and Hönig 2025), they remain computationally expensive and do not scale well to large teams.

In parallel, multi-robot motion planning, in its simplified form, can be formulated as a Multi-Agent Path Finding (MAPF) problem. In the MAPF domain, the world is represented as a graph, where robots move between vertices in one step. Over the years, remarkable progress has been achieved in developing efficient and scalable solutions to MAPF problems (Sharon et al. 2015; Barer et al. 2014; Okumura et al. 2022; Okumura 2023b). While these methods can handle thousands of robots (Okumura 2023b,a), they neglect robot dynamics, modeling robots as 2D points that move in a discrete grid world. As a result, planned trajectories are not executable on real robot platforms.

This work integrates the scalability and efficiency of MAPF algorithms with the dynamic-awareness of kinodynamic planners. At its core lies the discontinuity-bounded principle (Hönig, Ortiz-Haro, and Toussaint 2022), which relaxes strict continuity between consecutive motion primitives. Allowing bounded discontinuity when selecting successor motions makes the search tractable by enabling primitive reuse and maintaining a finite set of states. By combining this idea with lightweight MAPF coordination, we propose discontinuity-Bounded LaCAM (db-LaCAM), a dynamics-agnostic multi-robot motion planner capable of generating dynamically feasible and collision-free trajectories efficiently.

The proposed planner db-LaCAM is a search-based planner that extends LaCAM (Okumura 2023b) to the continuous domain. As LaCAM is built upon PIBT (Okumura et al. 2022), db-LaCAM incorporates its dynamic counterpart, db-PIBT, to produce feasible motion sequences over a fixed-length horizon. This integration enables db-LaCAM to leverage db-PIBT for efficient local coordination while performing long-horizon search to avoid livelocks and improve scalability. The heuristic guidance within db-LaCAM relies on a hierarchical Expansive Space Tree (EST) (Phillips, Bedrossian, and Kavraki 2004) that approximates the cost-to-go in continuous, dynamically-aware settings.

Theoretically, db-LaCAM is resolution-complete with respect to motion primitives. Empirically, we show that db-LaCAM can solve challenging multi-robot motion planning problems efficiently, see Fig. 1. For instance, db-LaCAM solves a 50 unicycle robots-instance in 20 s, whereas state-of-the-art planners db-CBS (Moldagalieva et al. 2024), db-ECBS (Moldagalieva, Ortiz-Haro, and Hönig 2025) fail.

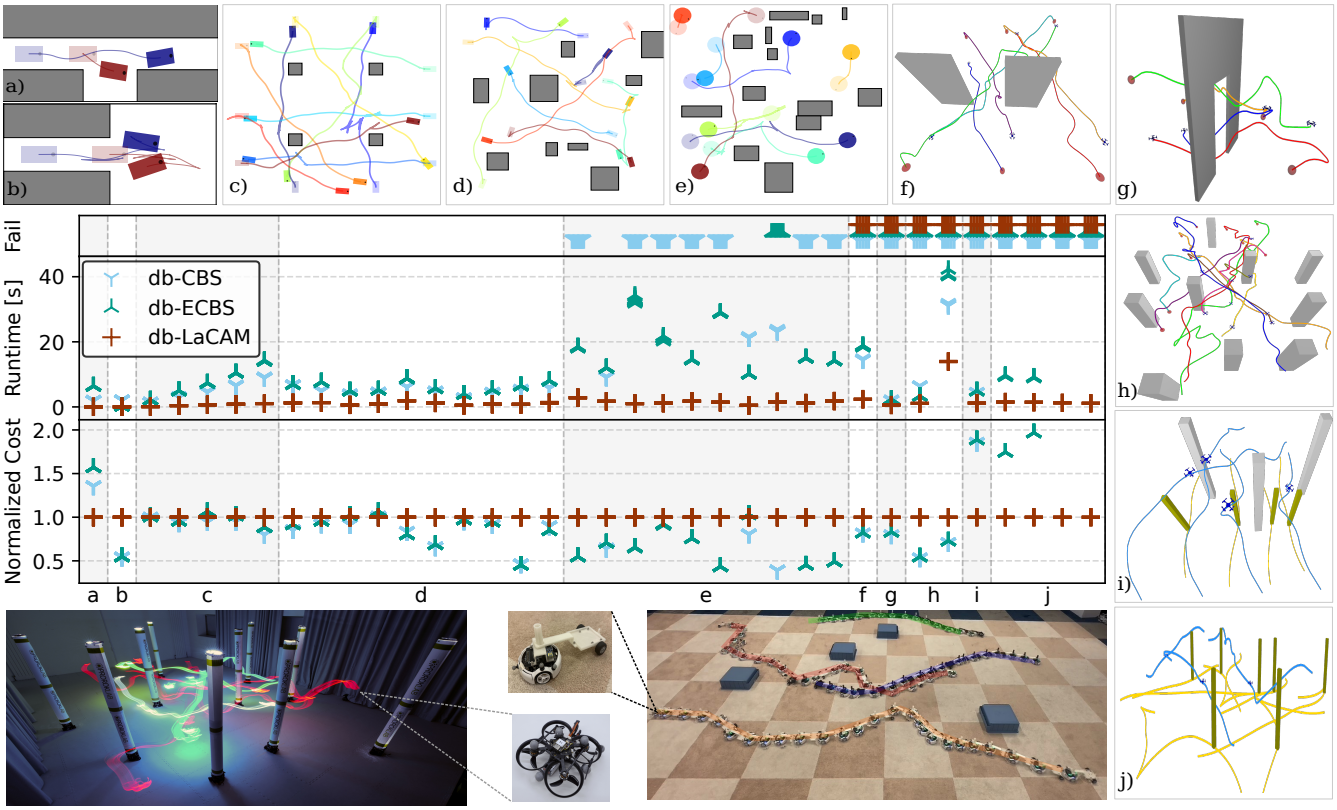

Figure 1: Performance and demonstration of db-LaCAM. Example problem setups (top and third rows) and corresponding quantitative comparisons (second row) over 35 instances grouped into ten representative environments (alternating shaded regions) over 10 trials. Each point represents a single trial outcome, while panels show failure rate, runtime, and normalized cost, respectively. Labels denote the evaluated environments: (a) *alcove*, (b) *at goal*, (c) *circle-n10*, (d) *random-n8*, (e) *random-n8-u_s*, (f) *passage-n6*, (g) *door-n4*, (h) *forest-n10*, (i) *swap-hetero-n8*, (j) *random-hetero-n8*. Example *random-hetero-n8* denotes a *random* problem with a heterogeneous team of eight robots. Instances (f–i) correspond to 3D environments. The bottom row illustrates real-world demonstrations with a team of ten flying robots and four car-like robots with trailers.

2 Related Work

This section reviews related work in multi-agent path finding (MAPF) and multi-robot kinodynamic motion planning.

Multi-Agent Path Finding MAPF assumes a discrete state space represented as a graph. A robot can move from one vertex along an edge to an adjacent neighbor in one step; robots cannot occupy the same vertex or traverse the same edge at the same time step. Solving MAPF optimally is NP-hard (Yu and LaValle 2013); therefore, optimal solvers struggle with scalability (Sharon et al. 2015). Several alternative methods have been suggested to address this scalability issue, including suboptimal alternative methods (Barer et al. 2014; Okumura et al. 2022; Okumura 2023b). Existing MAPF solvers are highly scalable and provide solutions efficiently, yet they ignore robot dynamics. As a result, their solutions can be infeasible to deploy on real robot platforms.

Multi-Robot Kinodynamic Motion Planning For multi-robot kinodynamic motion planning, single-robot planners applied to the joint space can be used (Li, Littlefield, and Bekris 2016), but do not scale well beyond a few robots. Better scalability can be achieved by adapting MAPF optimal solvers (Sharon et al. 2015) to kinodynamic motion planning by combining them with Model Predictive Control (Tajbakhsh, Biegler, and Johnson 2024), integrating them with sampling-based planners (Kottinger, Almagor, and Lahijanian 2022), or incorporating motion primitives (Moldagalieva et al. 2024; Moldagalieva, Ortiz-Haro, and Hönig 2025; Shankar, Okumura, and Prorok 2025; Guo and Yu 2024). In the latter case, hand-designed primitives can fail to generalize to complex robot dynamics and do not cover the full set of feasible motions, which can reduce planning efficiency (Shankar, Okumura, and Prorok 2025; Guo and Yu 2024). Additionally, these methods incur a high computational burden, leading to slow runtimes. An efficient planner can be achieved by combining Mixed-Integer Linear Programs with control theory (Chen et al. 2021). This approach needs safe regions around computed trajectories, so the controller can track them. These safe regions can be large, which makes the method incomplete. Overall, existing multi-robot kinodynamic planners remain slow and scale poorly as the number of robots increases.

3 Preliminaries

In this section, we first formally define the problem and its objective considered in this work (Sec. 3.1). We then briefly review MAPF algorithms that form the basis of our approach (Sec. 3.2) and define the notion of a motion primitive, a key component of our planning framework (Sec. 3.3).

3.1 Problem Definition

We consider a team of N heterogeneous robots. The state of the ith robot is given as $\mathbf{x}^{(i)} \in \mathcal{X}^{(i)} \subset \mathbb{R}^{d_{x^{(i)}}}$, which is actuated by controlling actions $\mathbf{u}^{(i)} \in \mathcal{U}^{(i)} \subset \mathbb{R}^{d_{u^{(i)}}}$. The workspace the robots operate in is $\mathcal{W} \subseteq \mathbb{R}^{d_w}$ ($d_w \in \{2, 3\}$), the collision-free space is $\mathcal{W}_{\text{free}} \subseteq \mathcal{W}$.

We assume that each robot $i \in \{1, \dots, N\}$ has dynamics $\dot{\mathbf{x}}^{(i)} = \mathbf{f}^{(i)}(\mathbf{x}^{(i)}, \mathbf{u}^{(i)})$. With zero-order hold discretization, motion can be framed as

$$\mathbf{x}_{k+1}^{(i)} \approx \text{step}(\mathbf{x}_k^{(i)}, \mathbf{u}_k^{(i)}) \equiv \mathbf{x}_k^{(i)} + \mathbf{f}^{(i)}(\mathbf{x}_k^{(i)}, \mathbf{u}_k^{(i)})\Delta t, \quad (1)$$

where Δt is sufficiently small to ensure that the Euler integration holds.

We denote by $\mathbf{X}^{(i)} = \langle \mathbf{x}_0^{(i)}, \mathbf{x}_1^{(i)}, \dots, \mathbf{x}_{K^{(i)}}^{(i)} \rangle$ the sequence of states of the ith robot sampled at times $0, \Delta t, \dots, K^{(i)}\Delta t$ and by $\mathbf{U}^{(i)} = \langle \mathbf{u}_0^{(i)}, \mathbf{u}_1^{(i)}, \dots, \mathbf{u}_{K^{(i)}-1}^{(i)} \rangle$ the sequence of actions applied to the ith robot for times $[0, \Delta t), [\Delta t, 2\Delta t), \dots, [(K^{(i)} - 1)\Delta t, K^{(i)}\Delta t)$.

Our goal is to move a team of N robots from their start states $\mathbf{x}_s^{(i)} \in \mathcal{X}^{(i)}$ to their goal states $\mathbf{x}_g^{(i)} \in \mathcal{X}^{(i)}$ as fast as possible while avoiding collisions and respecting robot dynamics. This problem can be formulated as:

$$\min_{\{\mathbf{X}^{(i)}\}, \{\mathbf{U}^{(i)}\}, \{K^{(i)}\}} \sum_{i=1}^{N} K^{(i)} \quad (2)$$

$$\text{s.t.} \begin{cases} \mathbf{x}_{k+1}^{(i)} = \text{step}(\mathbf{x}_k^{(i)}, \mathbf{u}_k^{(i)}) & \forall i \; \forall k, \\ \mathbf{u}_k^{(i)} \in \mathcal{U}^{(i)}, \;\; \mathbf{x}_k^{(i)} \in \mathcal{X}^{(i)} & \forall i \; \forall k, \\ \mathcal{B}^{(i)}(\mathbf{x}_k^{(i)}) \in \mathcal{W}_{\text{free}} & \forall i \; \forall k, \\ \mathcal{B}^{(i)}(\mathbf{x}_k^{(i)}) \cap \mathcal{B}^{(j)}(\mathbf{x}_k^{(j)}) = \emptyset & \forall i \neq j \; \forall k, \\ \mathbf{x}_0^{(i)} = \mathbf{x}_s^{(i)}, \;\; \mathbf{x}_{K^{(i)}}^{(i)} = \mathbf{x}_g^{(i)} & \forall i, \end{cases}$$

where $\mathcal{B}^{(i)} : \mathcal{X}^{(i)} \to 2^{\mathcal{W}}$ is a function that maps the state of the ith robot to a collision shape. The objective is to minimize the sum of the arrival times of all robots.

Example 1. *Consider a car with trailer robot with state* $\mathbf{x} = [x, \; y, \; \theta_1, \; \theta_2]^\top \in \mathbb{R}^2 \times (S^1)^2$*, where* (x, y) *is the car position,* θ_1 *is the car orientation and* θ_2 *is the trailer orientation. The control is* $\mathbf{u} = [v, \; \phi]^\top \in \mathcal{U} \subset \mathbb{R}^2$*, with linear velocity* v *and steering angle* ϕ*. Let* L *be the car wheelbase and* L_h *the hitch length. The dynamics are* $\dot{\mathbf{x}} = [v\cos(\theta_1), v\sin(\theta_1), \frac{v}{L}\tan\phi, \frac{v}{L_h}\sin(\theta_1 - \theta_2)]^\top$.

3.2 PIBT and LaCAM

PIBT *(priority inheritance with backtracking)* (Okumura et al. 2022) is a scalable and suboptimal MAPF algorithm. The method performs one-timestep planning among agents following the priority of the agents. In each timestep, agents update priorities, and the planner assigns next positions sequentially to avoid conflicts with high-prioritized agents. When there is no location left for a lower-prioritized agent, then priority inheritance takes place, enforcing the higher-prioritized robot to move out of the way. PIBT is a greedy algorithm, which is guided by cost-to-go heuristics. Its greedy nature can cause deadlocks, making the planner incomplete.

The complete planner LaCAM *(lazy constraints addition search)* (Okumura 2023b) addresses these limitations of PIBT. LaCAM is a search-based MAPF solver that works in two levels. At the high level, LaCAM searches over configurations of all agents. At the low level, it generates constraints for each high-level node. Constraints are used to specify which locations are occupied by which agent in the next configuration. The high-level search is guided by PIBT and proceeds lazily: instead of generating all valid successors, it only generates one feasible successor each time the high-level node is invoked.

The scalability and speed of PIBT and LaCAM motivate extending them to kinodynamic motion planning. Such an extension introduces new challenges: continuous state spaces, constraints from robot dynamics, and harder heuristic computation, as Euclidean distance no longer reflects true cost-to-go. The proposed planner, db-LaCAM, addresses these challenges via motion primitives, enabling efficient, constraint-respecting planning in the continuous domain.

3.3 Motion Primitive

A motion primitive is a sequence of states and controls that fulfill the dynamics of the system given in Eq. (1). Formally,

Definition 1. *A motion primitive is a tuple* $\langle \mathbf{X}, \mathbf{U}, K \rangle$*, consisting of state sequences* $\mathbf{X} = \langle \mathbf{x}_0, ..., \mathbf{x}_K \rangle$ *and control sequences* $\mathbf{U} = \langle \mathbf{u}_0, ..., \mathbf{u}_{K-1} \rangle$ *which obey the dynamics* $\mathbf{x}_{k+1} = \text{step}(\mathbf{x}_k, \mathbf{u}_k)$.

Motion primitives can be generated by discretizing the state space of the robot (Shankar, Okumura, and Prorok 2025; Hou et al. 2025), or by solving a two-point boundary value problem with nonlinear optimization (Hönig, Ortiz-Haro, and Toussaint 2022).

4 Approach

We begin by describing db-PIBT (Sec. 4.1), the core component of our planner db-LaCAM. We then present db-LaCAM (Sec. 4.2), followed by the techniques developed for heuristic estimation (Sec. 4.3), motion clustering (Sec. 4.4), livelock detection and resolution (Sec. 4.5), and finally discuss the properties of db-LaCAM (Sec. 4.6).

4.1 db-PIBT

Kinodynamic motion planner db-PIBT extends PIBT (Okumura et al. 2022) to the continuous domain. The planner db-PIBT (Alg. 1) searches over motion primitives to plan a fixed-horizon state sequence and supports arbitrary robot dynamics with pre-computed primitives; major changes are marked in blue.

At a high level, db-PIBT incrementally assigns dynamically feasible motion segments to robots in descending priority order (Line 3), ensuring that each selected motion re-

mains collision-free with respect to already planned trajectories. The priority order of robots is based on the distance to the goal, with the farthest receiving the highest priority. Here, we rely on a user-defined *metric* $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ to measure the distance between two states.

Within this high-level loop, a recursive procedure is called for each robot. The procedure `db-PIBT` explores the set of valid motion primitives $\mathcal{M}^i$ for the given robot i (Line 6) to select feasible successors. Each candidate motion m is checked for potential conflicts: *(i)* collisions with high-priority robot motions (Line 7); *(ii)* collisions with lower-priority robots that are not yet planned (Line 11). For the second case, we assume that each lower-priority robot j has a set of valid motion primitives $\mathcal{M}^j$ (Line 10). When the candidate motion m conflicts with any motion in $\mathcal{M}^j$, then db-PIBT recursively invokes itself (Line 12) to tentatively assign a valid motion to robot j. This recursive call checks whether a consistent set of motions can be found for both robots. If no valid combination exists, the candidate motion m is discarded, and the algorithm proceeds to test the next motion in $\mathcal{M}^i$. Once a valid motion primitive is found, db-PIBT updates the reserved motions vector $\mathcal{T}$ with the found motion for robot i (Line 9) and returns `VALID` (Line 14).

While db-PIBT is effective, its greedy nature can lead to livelocks in tightly constrained scenarios, highlighting the need for long-term planning, which db-LaCAM provides.

Algorithm 1: db-PIBT

1: **input:** robots N, motion sets $\boldsymbol{M} = (\mathcal{M}^1, \dots, \mathcal{M}^N)$
2: **output:** motions $\mathcal{T}$ (each element is initialized with $\perp$)
3: **for** $i \in N$ **do**; **if** $\mathcal{T}[i] = \perp$ **then** db-PIBT$(i, \boldsymbol{M}, \mathcal{T})$
4: **return** $\mathcal{T}$
5: **procedure** `DB-PIBT`$(i, \boldsymbol{M}, \mathcal{T})$
6: **for** $m \in \mathcal{M}^i$ **do** ▷ loop over sorted motions
7: **if** $\exists m_k \in \mathcal{T}$ **s.t.** `Collide`(m, m_k) **then**
8: **continue** ▷ collision with planned robot
9: $\mathcal{T}[i] \leftarrow m$ ▷ reserve the motion
10: **for** $\mathcal{M}^j \in \boldsymbol{M}$ **do**
11: **if** $\exists m_j \in \mathcal{M}^j$ **s.t.** `Collide`(m, m_j) **then**
12: **if** db-PIBT$(j, \boldsymbol{M}, \mathcal{T})$ = INVALID **then**
13: **continue** ▷ priority-inheriting robot failed
14: **return** VALID
15: **return** INVALID

4.2 db-LaCAM

The planner db-LaCAM is a search-based kinodynamic planner that builds on LaCAM (Okumura 2023b) and db-PIBT to account for robot dynamics. The high-level search of db-LaCAM is given in Alg. 2. The algorithm can work with arbitrary robot dynamics. Its major changes compared to LaCAM are marked in blue.

The planner db-LaCAM starts the search by creating two sets: $Open$ that stores nodes to be expanded, and $Explored$ to keep track of already expanded nodes (Line 4). The initial node, given as Q_{init}, is initialized with a set of starting states for all robots, an empty set of constraints, and

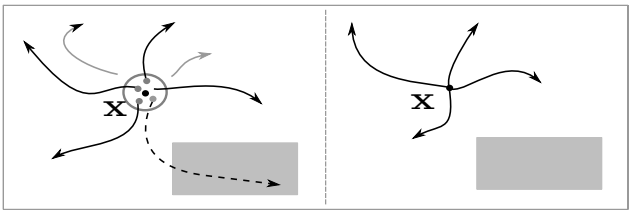

Figure 2: Visual representation of motion primitive samples. Given state $\mathbf{x}$, applicable motions (black edges) start within a discontinuity lower than $\alpha\delta$ (gray circumference). Motions in collision with the environment are discarded (dashed edges). The action sequences of applicable motion primitives are used to forward-propagate the state $\mathbf{x}$.

motion primitives (Line 6). At each iteration, the top node from $Open$ is removed and expanded. If the state of this node is within the user-defined δ_g distance to goal states, the solution is recovered by backtracking each node's stored motion primitives and returned (Line 12). Otherwise, the search proceeds by querying a set of valid motion primitives via `Process_Motions`. Motions generated with `Process_Motions` update the constraint tree of the current node Q with `Set_Constraint_Tree` (Line 16). Each constraint defines which motion is selected for each robot in the subsequent planning horizon (details are in the appendix). The `db-PIBT` then assigns priorities to robots and sequentially plans for each robot, using the updated constraint tree to guide motion selection and maintain inter-robot consistency as shown in Alg. 1. The output of `db-PIBT` is a next horizon, collision-free sequence of states for each robot (Line 17). If all robots move successfully, then this configuration creates a new high-level node Q' (Line 21) and adds it to the $Open$ set (Line 22) After each successful high-level search iteration, the robots are optionally checked for livelock (Line 23).

The `Process_Motions` procedure is a key component of the algorithm and also its most computationally expensive step. It involves several subroutines for generating, validating, and ranking motion candidates as shown in Alg. 2. First, it finds applicable motions. A motion m is considered applicable at state $\mathbf{x}$ if its start state $m.\mathbf{x}_s$ is within $\alpha\delta$ away from it, where α is a user-defined parameter and δ is the discontinuity bound (Line 27). For efficient search, we adopt a k-d tree, $\mathcal{T}_m$, to index the start states of all provided motion primitives (Line 3). Once applicable motions are selected, their action sequences are applied to the current states to perform forward propagation (Line 28). An example of resulting motions is illustrated in Fig. 2. Second, it computes the cost-to-go $h(m.\mathbf{x}_f)$ for each rolled-out motion m (Line 29); details are in Sec. 4.3. Finally, it performs a motion clustering (Line 30), as described in Sec. 4.4.

4.3 Heuristics Estimation

The core component of db-LaCAM, db-PIBT, requires an accurate estimation of the cost-to-go h for each motion to guide the search towards the goal. In the discrete domain, PIBT uses the shortest path length to compute the h for each

Algorithm 2: db-LaCAM

input: robots N, starting states $\{\mathbf{x}_s^{(i)}\}$, goal states $\{\mathbf{x}_g^{(i)}\}$, motion set $\boldsymbol{M}$
params: goal threshold δ_g
$\mathcal{T}_m \leftarrow$ `Nearest_Neighbor_Init`$(\boldsymbol{M})$ ▷ use start states of motions
`Initialize` $Open$, $Explored$
$C_{\text{init}} \leftarrow \langle\ parent : \perp, who : \perp, where : \perp, motion : \{\}\ \rangle$ ▷ no constraint
$Q_{\text{init}} \leftarrow \langle state : \mathbf{x}_s,\ tree : C_{\text{init}},\ motions : \{\}\rangle$
$Open.push(Q_{\text{init}})$; $Explored[\mathbf{x}_s] \leftarrow Q_{\text{init}}$
while $Open \neq \emptyset$ **do**
  $Q \leftarrow Open.top()$ ▷ high-level node
  $\mathbf{x} \leftarrow Q.state$ ▷ starting state for the horizon
  if `distance`$(\mathbf{x}, \mathbf{x}_g) \leq \delta_g$ **then**
    return backtrack(Q)
  if $Q.tree = \emptyset$ **then** $Open.pop()$; **continue**
  $C \leftarrow Q.tree.pop()$ ▷ constraint tree
  $\boldsymbol{M'} \leftarrow$ `Process_Motions`$(\mathbf{x}, \mathbf{x}_g, \mathcal{T}_m)$
  `Set_Constraint_Tree`(Q) ▷ update $Q.tree$
  $\mathcal{T}' \leftarrow$ `db-PIBT`$(N, \boldsymbol{M'})$ ▷ Alg. 1
  $\mathbf{x}' \leftarrow \mathcal{T}'.\mathbf{x}.back()$ ▷ final state for the horizon
  if $\mathbf{x}' = \perp$ **then continue**
  if $Explored[\mathbf{x}'] \neq \perp$ **then continue**
  $Q' \leftarrow \langle state : \mathbf{x}',\ tree : C_{init},\ motions : \mathcal{T}'\rangle$
  $Open.push(Q')$; $Explored[\mathbf{x}'] \leftarrow Q'$
  `Livelock_Detection(Q)` ▷ optional step
return NO_SOLUTION
procedure PROCESS_MOTIONS$(\mathbf{x}, \mathbf{x}_g, \mathcal{T}_m)$
  for $i \in N$ **do**
    $\mathcal{M}^i \leftarrow$ `Nearest_Neighbor_Query`$(\mathcal{T}_m^i, \mathbf{x}^i)$ ▷ applicable motions with discontinuity up to $\alpha\delta$
    $\mathcal{M}_r^i \leftarrow$ `Rollout_Motions`$(\mathbf{x}^i, \mathcal{M}^i)$
    `HEST`$(\mathbf{x}_g^i, \mathcal{M}_r^i)$ ▷ compute heuristics Sec. 4.3
    $\mathcal{M}_c^i \leftarrow$ `Cluster_Motions`$(\mathcal{M}_r^i)$ ▷ Sec. 4.4
  return $\mathcal{M}_c$ ▷ clustered motions for each robot

grid cell. However, in a continuous domain where robots exhibit different dynamics, this metric can no longer provide a reliable cost-to-go estimate. The h can be pre-computed using a single robot planner in a reverse manner from the goal state towards the start state. This method has two limitations. First, it requires inverse dynamics, which are often intractable to compute since the system is forward-propagated via actions, and recovering actions backward from the final state would need inverting the dynamics model. Second, it is computationally expensive – especially in large environments, as it explores large portions of the environment that are often irrelevant for finding a solution.

Hierarchical EST We propose the Hierarchical Expansive Spaces Trees (*HEST*) for efficient estimation of h. *HEST* adapts Guided EST (Phillips, Bedrossian, and Kavraki 2004), a single-robot sampling-based planner that iteratively expands the most promising state until the goal state is reached.

HEST implements a nearest-neighbor table of explored states with their cost-to-go values for efficient h-value retrieval during expansions. It operates on two hierarchical levels. At the high level, it runs Guided EST in reverse from the goal to the start to obtain a coarse estimate of the heuristic h. At the low level, it conducts a forward search with Guided EST from a given state towards the goal, using the high-level estimate to prioritize expansions.

Alg. 3 computes motion-wise heuristic estimates for robot i using precomputed nearest-neighbor tables. For each motion in $\mathcal{M}^i$ (Line 5), the algorithm queries the reverse search table H_r^i to find the closest stored state (Line 6). If no nearby state exists within threshold Δ, it performs a forward estimation using `EST` (Line 8) and updates the forward table H^i; otherwise, it reuses the nearest neighbor's heuristic value (Line 10). This structure enables *HEST* to focus computation where it matters most–refining heuristics locally while reusing information from high-level exploration. An ablation study in Sec. 5.3 examines heuristic methods.

Algorithm 3: HEST for a single robot i

1: **input:** goal state $\mathbf{x}_g^i$, motions $\mathcal{M}^i$
2: **output:** heuristic values h for each motion in $\mathcal{M}^i$
3: **params:** distance threshold Δ
4: **global:** H_r^i, H^i ▷ Nearest-neighbor table for reverse and forward searches
5: **for** $m \in \mathcal{M}^i$ **do**
6: $\mathbf{x}_n \leftarrow$ `Nearest_Neighbor_Query`$(m.\mathbf{x}_f, \mathcal{H}_r^i)$
7: **if** $\mathbf{x}_n = \emptyset \vee$ `distance`$(m.\mathbf{x}_f, \mathbf{x}_n) > \Delta$ **then**
8: $h(m.\mathbf{x}_f) \leftarrow$ `EST`$(m.\mathbf{x}_f, \mathbf{x}_g, \mathcal{H}^i)$ ▷ update H^i
9: **else**
10: $h(m.\mathbf{x}_f) \leftarrow h(\mathbf{x}_n)$ ▷ assign nearest neighbor's h

4.4 Motion Primitives Clustering

Let $\mathcal{M}^i = \langle m_1,\ m_2,\ \ldots\ m_M\rangle$ be a set of valid motions for the robot i. As the number of motions in $\mathcal{M}^i$ grows, the search in db-PIBT becomes less efficient and slows down. A straightforward approach to reduce the size is to sort motions in ascending order of their heuristic h values and select the top n motions. However, this can lead to a set of motions that are very similar to each other, which might hinder the planner. Based on the motion primitives design, we develop two clustering techniques to have diverse motions.

Goal-Oriented Clustering (GOC) GOC aims to sort motions in an ascending order of h-values $h_1 \leq h_2 \leq \cdots \leq h_M$. For a cluster $\mathcal{K}$ starting at h_l, it includes all motions $m \in \mathcal{M}_i$ that satisfy $|h - h_l| \leq \iota$, where $\iota = \rho \cdot (h_{\max} - h_{\min})$ and ρ is a hyperparameter. Once clusters are formed, n elements per cluster are chosen using one of: *(i)* vanilla selection–all elements; *(ii)* deterministic selection–top n elements; *(iii)* weighted selection–sampling with probability proportional to their values. This clustering strategy enables fast goal attainment but may cause livelock when robots must pass closely near their goals. As shown in Fig. 3 (bottom), both robots may prefer forward motions with lower h values, blocking each other.

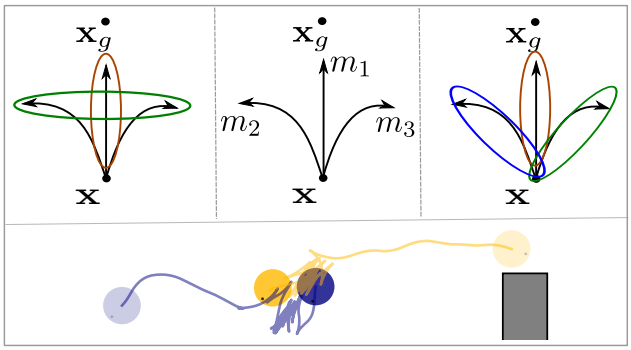

Figure 3: Top row: motion primitive clusters — middle: original motions, left: GOC, right: SC-GOC, with colors for each cluster. Motions toward left and right have similar distance to the goal state, yielding close h-values. Bottom row: example livelock between two robots.

Space-Cover and Goal-Oriented Clustering (SC-GOC) SC-GOC aims to provide a set of diverse motions for a better space cover. It first forms multiple intermediate clusters $\mathcal{K}_{int}$ based on spatial proximity. Specifically, it picks the motion m_r from $\mathcal{M}^i$ with the minimum h-value and sets it as the reference motion. Then, it computes the distance between the last states of each motion $m \in \mathcal{M}^i$ to the m_r as $d(m.\mathbf{x}_f, m_\mathrm{r}.\mathbf{x}_f)$. An intermediate cluster $\mathcal{K}_{int}$ is generated by including all motions that have a distance lower than the threshold τ. The value of τ is a hyperparameter. After, within each cluster, trajectories are sorted by h-value, and element selection can be performed using the same strategies (*(i-iii)*) used for GOC. Selected n motions are then reordered in an inside-out sequence, starting from the middle element and alternating outward toward the ends: $\langle m_{\lceil n/2 \rceil},\ m_{\lceil n/2 \rceil - 1},\ m_{\lceil n/2 \rceil + 1},\ \ldots \rangle$. Generated motion clusters for GOC (left) and SC-GOC (right) are shown in Fig. 3. Analysis of db-LaCAM's performance using different clustering methods is presented in Sec. 5.3.

4.5 Livelock Behavior

PIBT suffers from deadlock/livelock due to its short-horizon nature. One example is illustrated in Fig. 3 (bottom row). In MAPF settings, the livelock problem can be resolved by introducing special techniques like *swap* (Okumura 2023a), or by backtracking recent agent-wise state histories (Jain et al. 2025). Using the same techniques in the continuous domain is not trivial. For instance, a unicycle robot might return to the same position with a different orientation, which is treated as a new state distinct from previously visited states.

Detection and Recovery We resolve livelock as follows: *(i)* identifying robots that have oscillations in their motions; *(ii)* using state-aware motion clustering to provide a better space cover. For *(i)*, each time a new motion is generated in Alg. 2, each robot's heuristic history is backtracked to check for oscillations. A robot is considered to be in a livelock if the change Δh between consecutive high-level nodes alternates in sign. Once robots are identified, we break the livelock behavior by providing more diverse motions to the search with SC-GOC. The intuition is that with a diverse set of motions—including different orientations—robots are more likely to take sideway motions, effectively breaking potential livelocks. As discussed in Sec. 4.4, SC-GOC provides a good cover of the space by clustering motions based on spatial proximity.

A visual representation of the final clustering results using GOC and SC-GOC is given in Fig. 3 (top row). Here, although m_1 is the best option to reach the goal, it triggers livelock unless one of the robots decides to move sideways. However, sideway motions (m_2, m_3) have a higher h-value, so can never be taken by db-PIBT. Moreover, with the GOC clustering technique, motions m_2, m_3 can be grouped in one cluster due to their similar distances to the goal state $\mathbf{x}_g$. In Sec. 5.3, we analyze motion clustering methods.

4.6 Properties

The planner db-LaCAM is probabilistic resolution-complete (PRC; for any fixed resolution, the probability of finding a solution, if one exists at that resolution, approaches 1).

Theorem 1. *The db-LaCAM motion planner in Alg. 2 is probabilistically resolution-complete when elements in each cluster are selected probabilistically based on their weights.*

Proof Sketch: Db-LaCAM is probabilistically complete up to the resolution of the implicit graph defined by the chosen motion primitives. With an exhaustive search over the finite search space, it eventually finds a solution if one exists. Because motion clustering involves stochastic sampling, there exists a non-zero probability that db-LaCAM selects the correct motion. □

We consider the time complexity of db-PIBT in Alg. 1.

Proposition 1. *The time complexity of db-PIBT for a single fixed-length horizon planning is $\mathcal{O}(N^2 M^2)$, where N is the number of robots, M is the maximum number of motion primitives.*

Details of Theorem 1 and the proof of Proposition 1 are provided in the appendix.

5 Experimental Evaluation

For benchmarking we consider robot dynamics like unicycle ($1^{(\mathrm{st})}$-order), double integrator 3D, see (Hönig, Ortiz-Haro, and Toussaint 2022) for dynamics and bounds.

In each environment, we test db-LaCAM with db-PIBT, Discontinuity-bounded CBS (db-CBS) (Moldagalieva et al. 2024), its suboptimal variant db-ECBS (Moldagalieva, Ortiz-Haro, and Hönig 2025). We analyze the success rate, computational time until the first solution is found, and solution cost. The cost is a time, which is equal to the sum of control duration over all single-robot paths (Eq. (2)).

Our planner db-LaCAM is implemented in C++. For db-CBS and db-ECBS, we use the respective publicly available implementations from the authors. All planners rely on FCL (Flexible Collision Library) (Pan, Chitta, and Manocha 2012) for collision checking. The experiments are run on a workstation (AMD Ryzen Threadripper PRO 5975WX @ 3.6 GHz, 64 GB RAM, Ubuntu 22.04).

5.1 Benchmarking

Motion primitives are pre-computed offline by solving two-point boundary value problems with random start and goal configurations with nonlinear optimization (Ortiz-Haro et al. 2024). Motion clustering uses deterministic selection to choose elements per cluster. The used hyperparameter values are listed in the appendix. An instance is not solved successfully if no solution is found after the timelimit.

Environments Problem instance visualizations are given in Fig. 1. The timelimit for all instances is $60\,\mathrm{s}$. We consider five types of scenarios as follows:

- **Canonical 2D** examples include problems like *alcove, atgoal*, where one of the robots is forced to move to the alcove or aside to let the other robot reach its goal (Fig. 1a–b).
- **Circle 2D** example has from $N = 2$ up to $N = 10$ unicycle robots operating in 11×11 environment, where robots need to swap to reach their goals (Fig. 1c).
- **Random 2D** instances feature randomly placed obstacles and randomly assigned start and goal states for $N = 8$ unicycle robots operating in a 10×10 environment. Of these, 10 instances use box-shaped robots (Fig. 1d), and the other 10 use spherical robots with a radius of $20\,\mathrm{cm}$ (Fig. 1e).
- **Problems 3D** comprise compact environments with moderate obstacle density, including narrow corridors and cluttered spaces (Fig. 1f–h). The environments measure up to approximately $4 \times 6 \times 1.5$, and the number of robots ranges from four to ten.
- **Heterogeneous Robots** problems involve teams of ten robots (e.g., unicycle, 3D double integrator) in environments with and without obstacles. Some instances require robot swapping (Fig. 1i), while others feature random start and goal states (Fig. 1j)

Results Overall, db-CBS performs poorly in *2D* environments with frequent close-proximity interactions, such as those involving large or spherical robots, where numerous inter-robot conflicts lead to extended runtimes and low success rates (around 30%). In contrast, db-ECBS handles these scenarios more effectively, achieving success rates up to 90% by leveraging its conflict-guided heuristic to focus the low-level search. The planner db-LaCAM consistently outperforms all other planners, solving every instance rapidly (below $3\,\mathrm{s}$ even in the most cluttered setups) and maintaining high success across all environment types.

In *3D* environments, db-LaCAM achieves the lowest runtime consistently across all instances. Planners db-CBS and db-ECBS perform reasonably well in compact environments such as *forest*, *door*, but their efficiency degrades as the environment size and search space grow. In larger 3D setups like *passage*, both planners often fail to compute solutions within the timelimit due to the high computational cost of the reverse-search heuristic.

With *heterogeneous robots* instances, db-LaCAM consistently solves all problems with the lowest runtime. Planners db-CBS and db-ECBS achieve only around 60% success, as the optimization component often fails to converge, though the discrete search produces a solution.

Physical Robots We demonstrate the safe execution of db-LaCAM trajectories on real robot platforms. The real-world experiments are conducted inside a $7 \times 4 \times 2.5\,\mathrm{m}^3$ room equipped with a motion capture system with twelve Optitrack cameras. We use Sanity custom drones for flying robots (Woo et al. 2025), and Polulu 3pi+ 2040 differential-drive robots for ground robots. Pictures of the deployed platforms are illustrated in Fig. 1 (bottom row). In the first scenario, we test the *forest* example with ten flying robots modeled as 3D double integrator dynamics. In the second scenario, four ground robots modeled as cars with trailers are tasked to swap their positions. Experiments with physical robots are available in the supplemental video.

5.2 Scalability Test

We consider teams of $N = 10, 20, 30, 40, 50$ unicycle robots operating in 20×20 size environment (Fig. 4). Start, goal states for each robot are generated randomly in each instance. We set the timelimit to $5\,\mathrm{min}$, since baselines like db-CBS, db-ECBS require a longer time to find a solution for cases with $N > 10$ robots.

Results Scalability test results with increasing number of robots are shown in Fig. 4. The planner db-LaCAM consistently solves all instances with low runtime across different team sizes. For large teams ($N \geq 40$), both db-CBS and db-ECBS frequently fail, highlighting db-LaCAM's robustness in dense and challenging environments.

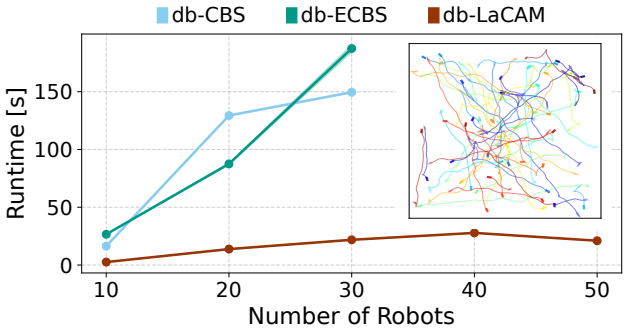

Figure 4: Runtime required to find a feasible solution over different numbers of robots.

5.3 Ablation Study

Computation time analysis We evaluate how much computational time is spent on some key components of db-LaCAM. and discuss how it varies with the increasing number of robots. Time statistics for key components of db-LaCAM while solving the *circle 2D* example are given in Fig. 5. The total time is mainly occupied by the heuristics h estimation, which will be discussed in detail in the following paragraph. The second most expensive operation is

the collision checking against the potential motions of low-priority neighboring robots that have no plans yet. Motion clustering is another time-consuming component, though it is relatively insensitive to the number of robots. Finally, collision with high-priority robots and motion rollout operations exhibit similar runtime across all instances.

Analysis of Heuristic Look-up Table Estimation We discuss two methods for heuristic value h computation from Sec. 4.3: *(i)* a single-robot kinodynamic planner db-A* (Discontinuity-Bounded A*) (Hönig, Ortiz-Haro, and Toussaint 2022) run in a reverse manner from $\mathbf{x}_g$ to $\mathbf{x}_s$; *(ii) HEST*. We analyze how the choice of the heuristic estimation affects the runtime of the planner. The summary of results is given in Fig. 5. *HEST*-based heuristic estimation method runs significantly faster compared to db-A*-based reverse search. This is due to the db-A*-based method exploring unnecessary parts of the environment as shown in Fig. 7 in the appendix. Whereas, *HEST*-based method explores only the relevant part of the environment. This happens because *HEST* explores nodes on demand, when the lookup table does not contain a nearby state and fails to return a h-value.

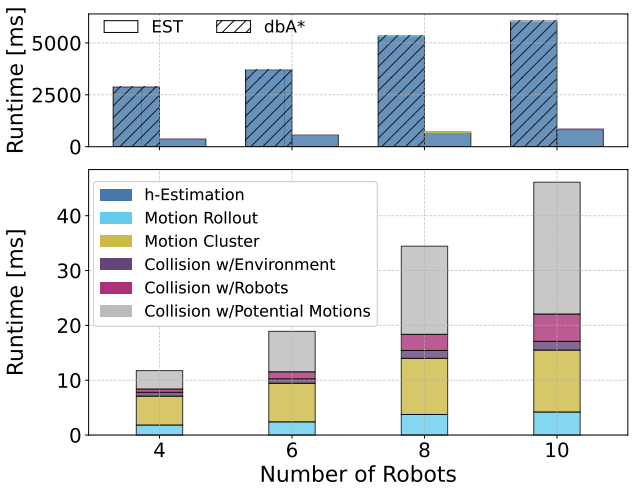

Figure 5: Computation time analysis for db-LaCAM. Upper row: two different methods for the heuristics h computation. Bottom row: analysis of time spent on some key components of db-LaCAM. Experiments are conducted using *circle 2D* example with varying numbers of robots as labeled.

Analysis of Motion Primitive Clustering We evaluate two methods for clustering applicable motion primitives discussed in Sec. 4.4: *(i)* motions are grouped based on their heuristic value h, thus goal reaching is faster (GOC); *(ii)* motions are grouped based on relative distance and heuristic value h enabling a better cover of the space (SC-GOC). Both methods employ weighted selection to choose elements from each cluster. We compare the impact of the two methods on the solution cost of db-LaCAM. Results are summarized in Fig. 6.

Instances are designed to have close-proximity interaction between robots to produce livelock cases. In these cases, db-LaCAM with GOC performs poorly because it always prioritizes goal-oriented motions. In contrast, SC-GOC resolves these instances faster by favoring sideway motions, which break livelock and improve solution quality (Fig. 6). For these instances, db-LaCAM with vanilla selection of elements per cluster fails to find a solution within the time-limit. This is because the motion set contains many similar motions as described in Sec. 4.4; since db-LaCAM performs an exhaustive search, it explores all possible motion combinations, resulting in longer runtimes.

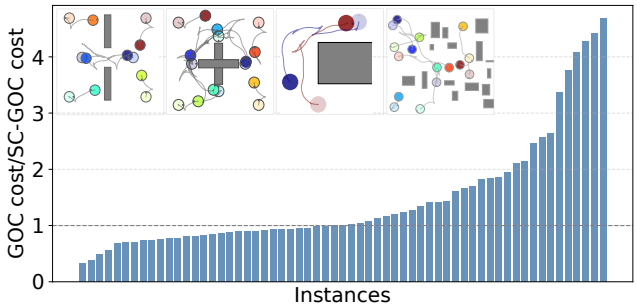

Figure 6: Analysis of two different methods for motion clustering: GOC and SC-GOC. Evaluation is performed with a *random* problem instance with eight unicycle robots of spherical shape with radius 20 cm over 70 instances with varying settings.

6 Conclusion

We introduce db-LaCAM, a planner that combines the scalability of discrete MAPF techniques with the dynamic feasibility of kinodynamic planning. By allowing bounded discontinuities between motion primitives, db-LaCAM enables efficient and flexible trajectory generation for multi-robot systems of arbitrary dynamics. Built upon the coordination mechanism of db-PIBT, it combines local dynamic feasibility with long-horizon search, achieving order-of-magnitude speedups over existing kinodynamic planners while maintaining comparable solution quality. The successful deployment of flying and car-with-trailer robots highlights the approach's potential for real-world multi-robot coordination.

Limitations and Future Work Our planner db-LaCAM is an informed search-based planner guided by a heuristic h. If the heuristic is imprecise, the planner can get stuck in local minima. While using the hierarchical EST to estimate h works well for the dynamics considered here, it struggles with more complex systems such as cars with trailers—for example, computing h for the *swap* problem with four car-with-trailer robots takes 6.40 s of a total 6.47 s runtime, as HEST must run for nearly all expanded nodes. Moreover, the current approach is centralized and not designed for real-time execution. Future work will explore learning-based heuristics for better efficiency, as in MAPF (Jain et al. 2025) and enabling real-time, decentralized execution.

Acknowledgments

This research was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 448549715, JST ACT-X (JPMJAX22A1), and JST PRESTO (JPMJPR2513).

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 Edelkamp, S.; and Barták, R., eds., *International Symposium on Combinatorial Search (SoCS)*, 19–27.

Chen, J.; Li, J.; Fan, C.; and Williams, B. C. 2021. Scalable and Safe Multi-Agent Motion Planning with Nonlinear Dynamics and Bounded Disturbances. In *AAAI Conference on Artificial Intelligence (AAAI)*, 11237–11245.

Guo, T.; and Yu, J. 2024. Decentralized Lifelong Path Planning for Multiple Ackerman Car-Like Robots. arXiv:2402.11767.

Hou, J.; Zhou, X.; Pan, N.; Li, A.; Guan, Y.; Xu, C.; Gan, Z.; and Gao, F. 2025. Primitive-Swarm: An Ultra-Lightweight and Scalable Planner for Large-Scale Aerial Swarms. *IEEE Transactions on Robotics (T-RO)*, 41: 3629–3648.

Hönig, W.; Ortiz-Haro, J.; and Toussaint, M. 2022. db-A*: Discontinuity-bounded Search for Kinodynamic Mobile Robot Motion Planning. In *International Conference on Intelligent Robots and Systems (IROS)*.

Jain, R.; Okumura, K.; Amir, M.; and Prorok, A. 2025. Graph Attention-Guided Search for Dense Multi-Agent Pathfinding. *arXiv preprint arXiv:2510.17382*.

Kottinger, J.; Almagor, S.; and Lahijanian, M. 2022. Conflict-Based Search for Multi-Robot Motion Planning with Kinodynamic Constraints. In *International Conference on Intelligent Robots and Systems (IROS)*, 13494–13499.

Li, Y.; Littlefield, Z.; and Bekris, K. E. 2016. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research (IJRR)*, 35(5): 528–564.

Moldagalieva, A.; Ortiz-Haro, J.; and Hönig, W. 2025. db-ECBS: Interaction-Aware Multi-Robot Kinodynamic Motion Planning. *IEEE Transactions on Robotics (T-RO)*.

Moldagalieva, A.; Ortiz-Haro, J.; Toussaint, M.; and Hönig, W. 2024. db-CBS: Discontinuity-Bounded Conflict-Based Search for Multi-Robot Kinodynamic Motion Planning. In *International Conference on Robotics and Automation (ICRA)*, 14569–14575.

Okumura, K. 2023a. Improving LaCAM for Scalable Eventually Optimal Multi-Agent Pathfinding. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 243–251.

Okumura, K. 2023b. LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding. In Williams, B.; Chen, Y.; and Neville, J., eds., *AAAI Conference on Artificial Intelligence (AAAI)*, 11655–11662.

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

Ortiz-Haro, J.; Hönig, W.; Hartmann, V. N.; and Toussaint, M. 2024. iDb-A*: Iterative Search and Optimization for Optimal Kinodynamic Motion Planning. *IEEE Transactions on Robotics (T-RO)*, 1–19.

Pan, J.; Chitta, S.; and Manocha, D. 2012. FCL: A General Purpose Library for Collision and Proximity Queries. In *International Conference on Robotics and Automation (ICRA)*, 3859–3866.

Phillips, J.; Bedrossian, N.; and Kavraki, L. 2004. Guided Expansive Spaces Trees: a search strategy for motion- and cost-constrained state spaces. In *International Conference on Robotics and Automation (ICRA)*, volume 4, 3968–3973 Vol.4.

Shankar, A.; Okumura, K.; and Prorok, A. 2025. LF: Online Multi-Robot Path Planning Meets Optimal Trajectory Control. arXiv:2507.11464.

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

Tajbakhsh, A.; Biegler, L. T.; and Johnson, A. M. 2024. Conflict-Based Model Predictive Control for Scalable Multi-Robot Motion Planning. In *International Conference on Robotics and Automation (ICRA)*, 14562–14568.

Woo, H.; Sanim, K. R. I.; Okumura, K.; Yang, G.; Shankar, A.; and Prorok, A. 2025. Sanity: An Agile Brushless Quadrotor for Multi-Agent Experiments.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *AAAI Conference on Artificial Intelligence (AAAI)*, 1443–1449.

Appendix

A Remarks for Thrm. 1

LaCAM is a complete algorithm for MAPF, i.e., if a solution exists, it will find it (Okumura 2023b). LaCAM relies on the argument that, for completeness, the search explores all states that could contain a solution; no potential solution paths are pruned. For kinodynamic motion planning problems, continuous time and space render full enumeration proofs infeasible, as the set of possible states is infinite. Instead, we consider resolution-completeness (RC; if a solution exists at the chosen resolution, the algorithm is guaranteed to find it) and probabilistic-completeness(PC; the probability of finding a solution if one exists is 1).

Table 1 summarizes properties for db-LaCAM based on the motion clustering technique and set of motion primitives.

Remark 1. *By relaxing clustering conditions, such as disabling clustering in Alg. 2, db-LaCAM can achieve RC with respect to motion primitives.*

Remark 2. *With an iterative run of db-LaCAM, where with each iteration more motion primitives are added to the set $\mathcal{M}$, db-LaCAM can achieve PC. A growing set of motion primitives $\mathcal{M}$ results in a larger search graph, enabling exploration of all reachable states within the closed search space.*

Motion Clustering Method	Fixed Motion Set	Incremental Motion Set
No Cluster	RC	PC
Deterministic	Incomplete	Incomplete
Probabilistic	PRC	PC

Table 1: Summary of properties of db-LaCAM.

B Proof of Proposition 1

Proof Sketch: For each fixed-horizon planning, the procedure db-PIBT is called N times, once for each robot. This is because, db-PIBT(i, ..) is called if and only if the robot i has no reserved motion for the next horizon (Line 3). The collision checking between considered motion and high-priority robot motions is $\mathcal{O}(N)$ in the worst case (Line 7), resulting in $\mathcal{O}(MN)$ for the loop (Line 6-Line 14). The collision checking against potential motions of low-priority robots can be $\mathcal{O}(MNM_{max})$ (Line 11), where M_{max} is the maximum number of motions among all robots. Combining these two operations gives $\mathcal{O}(NM^2)$, assuming all robots have the same number of motions. □

C Procedure `Set_Constraint_Tree` in Alg. 2

Unlike LaCAM, where constraints correspond to occupied vertices, db-LaCAM defines constraints as valid motions. Alg. 4 illustrates how the constraint tree of a high-level node Q is updated given a set of valid motions $\boldsymbol{M}$. Constraints are assigned to robots according to their priority order (Line 6). Once a robot is identified, the algorithm loops over its valid motions (Line 8), creating a new constraint tree C_{new} for each motion (Line 10). Each newly created constraint tree is then added to the constraint trees of Q. If this high-level node Q is picked in further iterations of db-LaCAM (Alg. 2, Line 9), these constraints are respected during motion generation with db-PIBT (Alg. 2, Line 17).

Algorithm 4: Set_Constraint_Tree

1: **input:** motions $\boldsymbol{M}$, high-level node Q, constraint tree C
2: **output:** updated constraint tree for Q
3: $N = |Q.state|$ ▷ Number of robots
4: **if** $\texttt{depth}(C) \leq N$ **then**
5: $\quad Q.order = \texttt{Get_Order}(Q)$ ▷ assign robot priority order
6: $\quad i \leftarrow Q.order[\texttt{depth}(C)]$
7: $\quad \mathcal{M}^i = \boldsymbol{M}[\boldsymbol{i}]$ ▷ get valid motions
8: $\quad$ **for** $m \in \mathcal{M}^i$ **do** ▷ each valid motion as constraint
9: $\quad\quad \mathbf{x} = m.\mathbf{x}_f$
10: $\quad\quad C_{new} \leftarrow \langle\, parent : C, who : i, where : \mathbf{x}, motion : m \,\rangle$
11: $\quad\quad Q.tree.push(C_{new})$

D Details of Hierarchical EST from Sec. 4.3

.

As can be seen in Fig. 7(left), reverse search with db-A* can lead to exhaustive exploration of states that are irrelevant for finding a solution. In contrast, *HEST* explores states that can be used to find a solution from the start to the goal state efficiently.

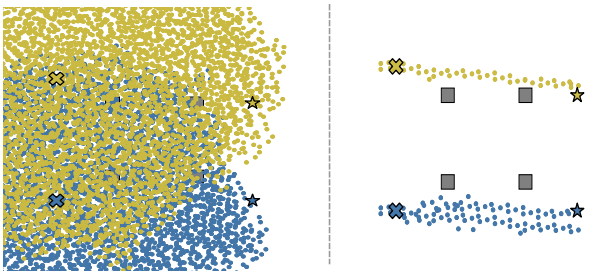

Figure 7: Visualization of state expansions for heuristic h computation: reverse search from goals to starts using db-A*, and forward search with EST. Each robot's start and goal are marked by crosses and stars; explored states are shown as colored dots per robot.

E Hyperparameter values for Benchmarking (Sec. 5)

The number of motion primitives used for the search is identical for all problem instances and equal to 300. The value of the discontinuity bound δ is 0.5 for all instances. The value of the distance threshold Δ for the lookup table is 1.0 for all dynamics. The range value for the GOC clustering method is 0.05 for unicycle dynamics, 1.0 for 3D double integrator, and the car with trailer. The distance threshold τ for the SC-GOC clustering method is set to 1.0 for all dynamics.