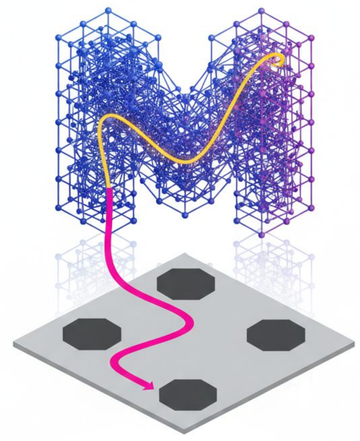
MeshA*: Efficient Path Planning With Motion Primitives

Marat Agranovskiy[1], Konstantin Yakovlev[1,2]

[1]Saint-Petersburg University
[2]CogAI Lab

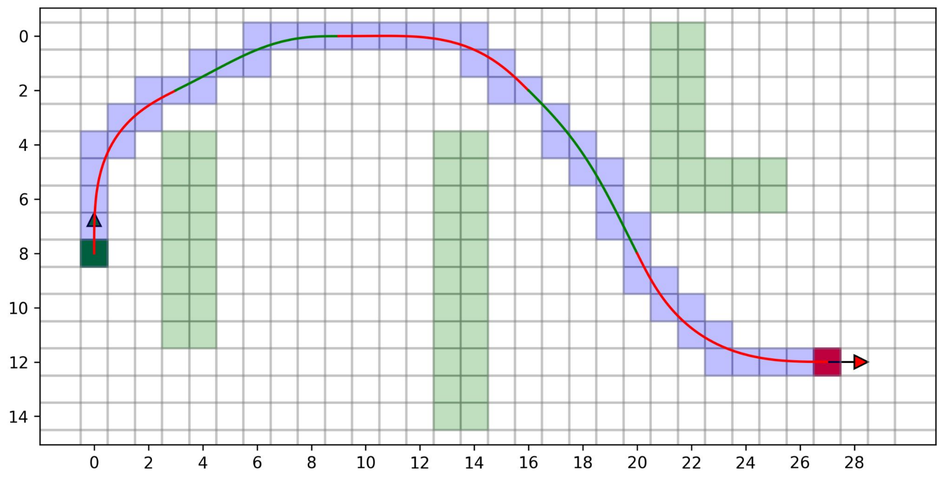
Corresponding author: ✉agrinscience@gmail.com



Problem Statement

Lattice-based Planning. We address the problem of constructing a kinodynamically feasible trajectory by assembling motion primitives (templates).

- **State:** A discrete tuple $s = (i, j, \theta)$, where (i, j) are grid coordinates and $\theta \in \Theta$ is a discrete heading.
- **Motion Primitives:** Transitions $s \to s'$ are defined by a finite, translation-invariant *Control Set*. Each primitive has a **cost** and a **collision trace** (sequence of swept grid cells, depicted in blue at Fig.).
- **Objective:** Find a path π (sequence of primitives) from s_{start} to s_{goal} that is **collision-free** (trace ∩ obstacles = ∅) and **optimal** (minimizes total cost).



Mesh Graph: New Search Space

We define a new search space where the nodes are **Extended Cells** $u = (i, j, \Psi)$. Here, (i, j) corresponds to a grid cell, and $\Psi = \{(prim_1, k), (prim_2, k), \ldots, (prim_n, k)\}$ is a **Configuration of primitives** – a set of active primitives traversing this cell at step k of their collision traces.

Algorithm 1: Building the Initial Configuration

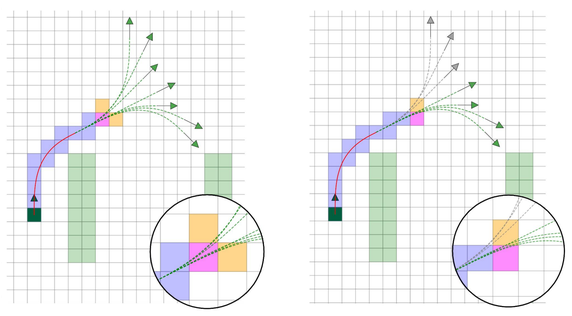
Input: Discrete angle $\theta \in \Theta$
Function name: InitConf(θ)

1: $\Psi \leftarrow \emptyset$
2: **for all** $prim \in$ `ControlSet` **do**
3: **if** $prim$ emerges in θ **then**
4: Ψ.add$\{(prim, 1)\}$
5: **return** Ψ

Algorithm 2: Generating Successors of an Extended Cell

Input: Extended cell u
Output: Set of pairs of successors and transition costs
Function name: GetSuccessors(u)

1: $i, j, \Psi \leftarrow u$
2: `Confs` $\leftarrow \{\}$ ▷ Dictionary
3: `Successors` $\leftarrow \emptyset$ ▷ Set of successors and costs
4: **for all** $(prim, k) \in \Psi$ **do**
5: $(a, b) \leftarrow \Delta_k^{prim}$
6: **if** $k = U_{prim} - 1$ **then** ▷ Case 1
7: $\theta \leftarrow$ angle at which $prim$ ends
8: $\Psi_1 \leftarrow$ InitConf(θ)
9: $v_1 \leftarrow (i + a, j + b, \Psi_1)$
10: `Successors`.add$\{(v_1, c_{prim})\}$
11: **else if** $(a, b) \notin$ `Confs` **then** ▷ Case 2
12: $conf_{new} \leftarrow \{(prim, k + 1)\}$
13: `Confs`$[(a, b)] \leftarrow conf_{new}$
14: **else**
15: `Confs`$[(a, b)]$.add$\{(prim, k + 1)\}$
16: **for all** $(a, b) \in$ `Confs` **do**
17: $\Psi_2 \leftarrow$ `Confs`$[(a, b)]$
18: $v_2 \leftarrow (i + a, j + b, \Psi_2)$
19: `Successors`.add$\{(v_2, 0)\}$
20: **return** `Successors`



The edges in the Mesh Graph are defined by advancing the primitives in Ψ by one step along their collision traces (successors depicted in orange on the Figure). For a cell u, successors are determined by grouping primitives based on their next occupied cell.

1. Continuation (Regular Successor). Primitives that have not finished their motion continue to the next cell in their trace. Crucially, all primitives from Ψ moving to the same neighbor (i', j') are **merged** into a single successor confi-guration Ψ' (Lines 11-15). The transition cost is 0 because the motion is ongoing.

2. Termination (Initial Successor). If a primitive finishes (line 6), the agent reaches a discrete state (i, j, θ). Here, we pay the primitive's cost (c_{prim}) and spawn an **Initial Configuration**, a new bundle containing all possible primitives starting from heading θ.

MeshA*

MeshA* Algorithm. We define MeshA* as the standard A* search applied to the Mesh Graph.

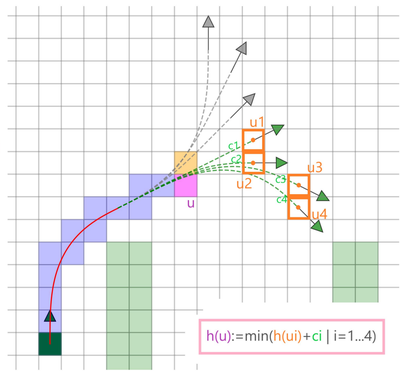
Efficiency & Pruning. The structural advantage of the Mesh Graph enables unique pruning techniques. To formalize them, let `Finals`(u) be the set of endpoints (discrete states) of all primitives passing through an extended cell u.

- **Cell-Level Heuristic.** Unlike standard lattice-based A*, which evaluates the heuristic only at primitive endpoints, MeshA* re-evaluates it at *every grid cell* using the information from `Finals`(u):

$$h(u) = \min_{(s_{end}, c) \in \texttt{Finals}(u)} \{h_{LBA*}(s_{end}) + c\}$$

  This allows the search to detect unpromising directions early and prune branches *"mid-motion"*, long before the primitive is fully traversed.

- **Terminal Pruning.** An extended cell u is immediately pruned (expansion skipped) if all states in `Finals`(u) have already been expanded.



Theoretical Analysis

Theorem 4 (Optimality of the path found by MeshA*)

The search for a least-cost, collision-free trajectory between discrete states $s_a = (i_a, j_a, \theta_a)$ and $s_b = (i_b, j_b, \theta_b)$ is equivalent to performing two steps:

1. **Find a least-cost path** on the mesh graph between the corresponding initial extended cells $u_a = (i_a, j_a, \Psi_{\theta_a})$ and $u_b = (i_b, j_b, \Psi_{\theta_b})$. This path must be collision-free, meaning the projection of every vertex on the path is a free grid cell.
2. **Recover the trajectory** from this path using the reconstruction algorithm.

Conclusion: The resulting trajectory is guaranteed to be **optimal** and **collision-free**.

Experimental Evaluation

Setup & Benchmark:

- **Maps:** 4 maps from MovingAI benchmark (varying topology).
- **Instances:** >25,000 start-goal pairs (from MovingAI scenarios) with random headings.
- **Control set:** 16 discrete headings, 24 primitives per heading.
- **Primitive cost:** length

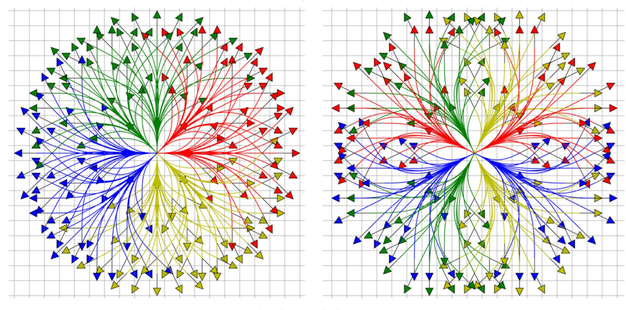
Baselines:

- **LBA*** (Standard Lattice-based A*)
- **LazyLBA*** (lazy collision checking)

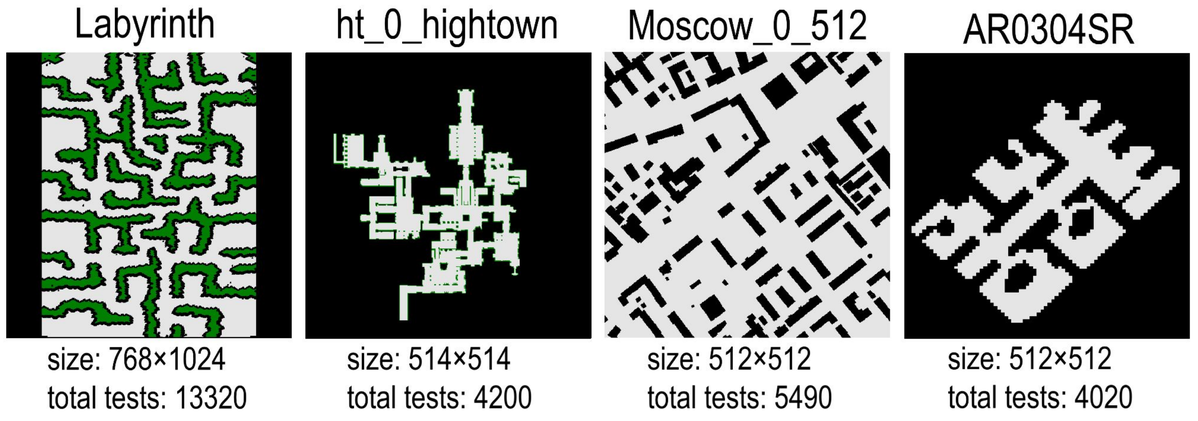
Heuristics:

- **LBA*:** Euclidean distance
- **MeshA*:** a derivative of Euclidean one
- **Weights:** 1, 1.1, 2, 5, 10 (factors for heuristics of both MeshA* and LBA*)

Motion primitives:

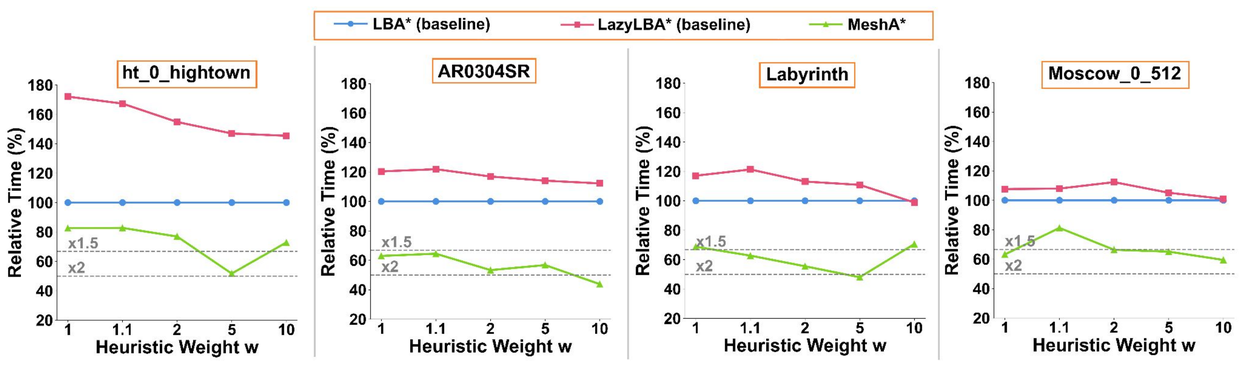


total: 24×16 =384



size: 768×1024 total tests: 13320 | size: 514×514 total tests: 4200 | size: 512×512 total tests: 5490 | size: 512×512 total tests: 4020

Median Runtime Analysis: MeshA*consistently outperforms lattice-based baselines. It achieves **1.5x speedup for optimal search** (w=1) and up to **2x speedup with weighted heuristics** (w>1). Notably, the lazy variant (LazyLBA*) performs worse than the standard LBA* in this setup. The overhead incurred by frequent priority queue operations (pushing and popping invalid states) outweighs the benefits of deferred collision checking, particularly because the collision checks themselves are computationally inexpensive in this domain.



Efficiency & Solution Quality: MeshA* **examines ≈50% fewer grid cells** than LazyLBA*, proving that cell-by-cell search effectively prunes invalid branches early, unlike lattice-based methods. While guaranteed optimal for w=1, MeshA* yields slightly higher costs than LBA* with weighted heuristics (w>1) due to more frequent heuristic evaluation and aggressive pruning.

Algorithms	ht_0_hightown			
	$w = 1$	$w = 2$	$w = 5$	$w = 10$
LBA*	100.0	105.7	110.0	113.2
LazyLBA*	100.0	105.7	110.0	113.2
MeshA*	100.0	109.2	117.6	122.3

pinely
Marat's travel was supported by Pinely

Code:

ArXiv: