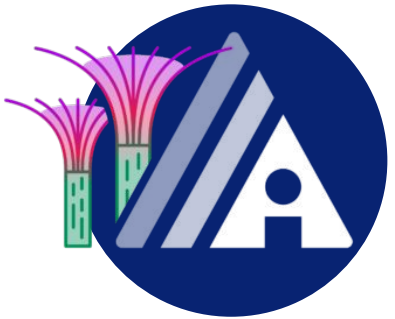


MeshA*: Efficient Path Planning With Motion Primitives

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

[1]Saint-Petersburg University
[2]AIRI

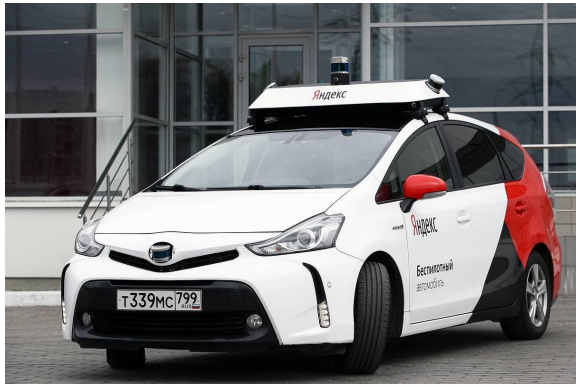
23 January 2026, Singapore

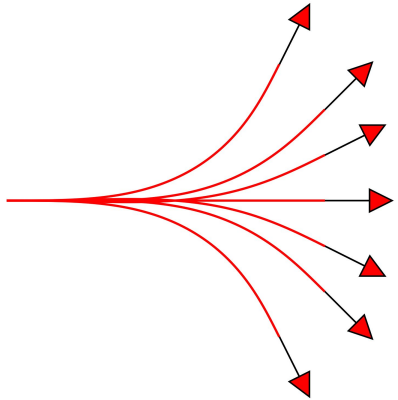
pinely
Marat's travel was supported by Pinely

Motivation: Lattice-Based Path Planning

Goal: Planning smooth, feasible trajectories for non-holonomic agents (e.g., self-driving vehicles).

Approaches: Sampling-based (RRT), Optimization-based, and Lattice-based (graph search).





Our focus: Lattice-Based Planning:

- Discretizes state space s = (x, y, θ).
- Motion Primitives: Short, pre-computed, feasible motion segments.
- Algorithm: A* searches for a sequence of primitives that smoothly connect.

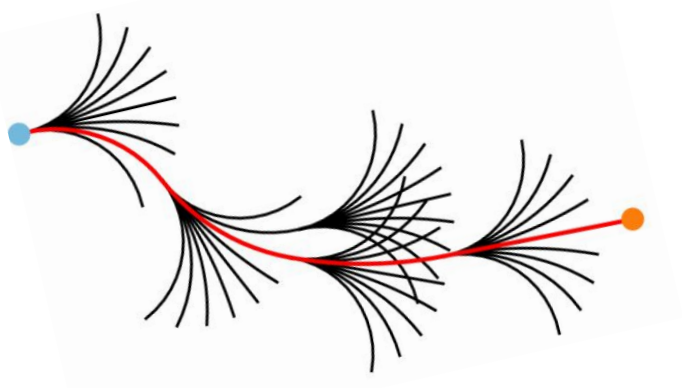


Image from: Thoresen, Marius & Nielsen, Niels & Mathiassen, Kim & Pettersen, K.Y.. (2021). Path Planning for UGVs based on Traversability Hybrid A*. IEEE Robotics and Automation Letters. PP. 1-1. 10.1109/LRA.2021.3056028.

Problem Statement

Workspace & State Representation

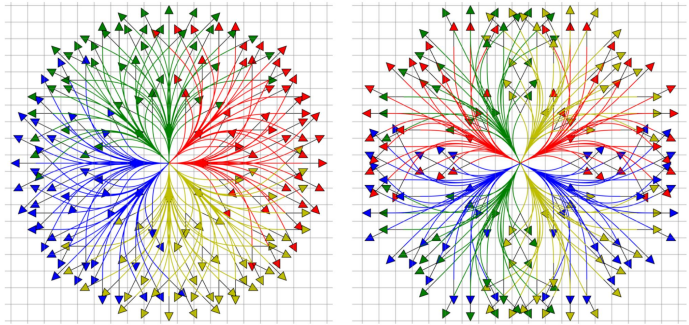
- Workspace: A 2D grid of free/blocked cells.
- State s = (i, j, θ): Agent is located at the center of grid cell (i, j) with a discrete heading θ.

Motion Primitives

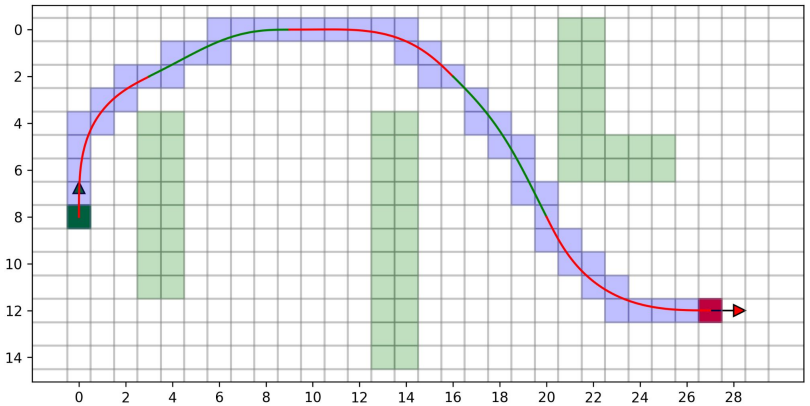
- Short, kinodynamically feasible motions connecting two discrete states: (i, j, θ) → (i', j', θ').
- Associated with a collision trace (sequence of swept cells) and cost.
- **Control Set:** A canonical finite set of motion templates, encapsulates all kinematic constraints.

Objective: Find a collision-free path (sequence of primitives from the control set) with minimal cost.

Motion primitives:



total: 24×16 =384

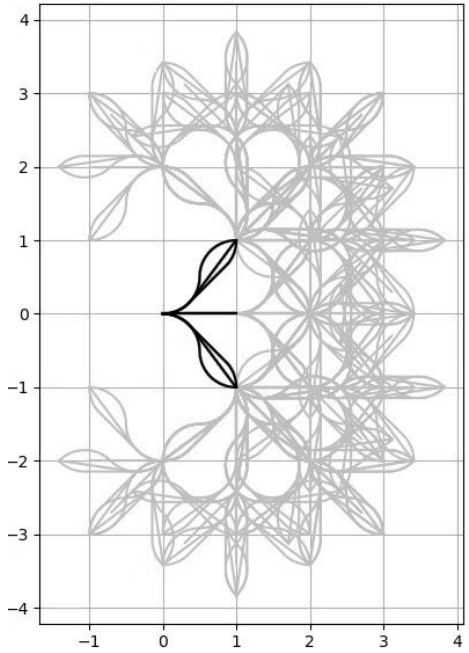


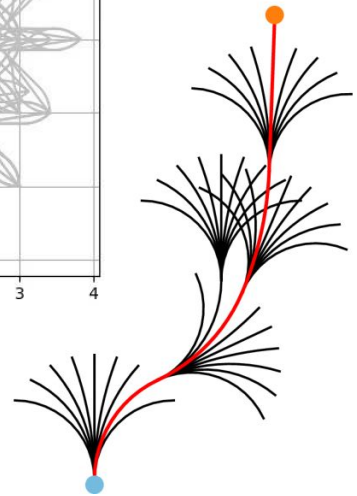
Green/red primitives and blue collision trace

The Challenges

- To achieve high-quality motion planning, we need a **rich set of motion primitives**.
- This inevitably leads to a **high branching factor** in the search graph.
- **Lattice-based A*** (LBA*) must generate and collision-check all applicable primitives for every state (i, j, θ).
- Lazy approaches (**LazyLBA***): postpone collision checks but frequently expand invalid states in cluttered environments.

The search tree is flooded with unpromising (LBA*) or invalid (LazyLBA*) partial paths.





Images from:
[1] A. Botros and S. L. Smith, "Computing a Minimal Set of t-Spanning Motion Primitives for Lattice Planners," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China, 2019, pp. 2328-2335, doi: 10.1109/IROS40897.2019.8968150.
[2] Thoresen, Marius & Nielsen, Niels & Mathiassen, Kim & Pettersen, K.Y.. (2021). Path Planning for UGVs based on Traversability Hybrid A*. IEEE Robotics and Automation Letters. PP. 1-1. 10.1109/LRA.2021.3056028.

MeshA*: A New Perspective

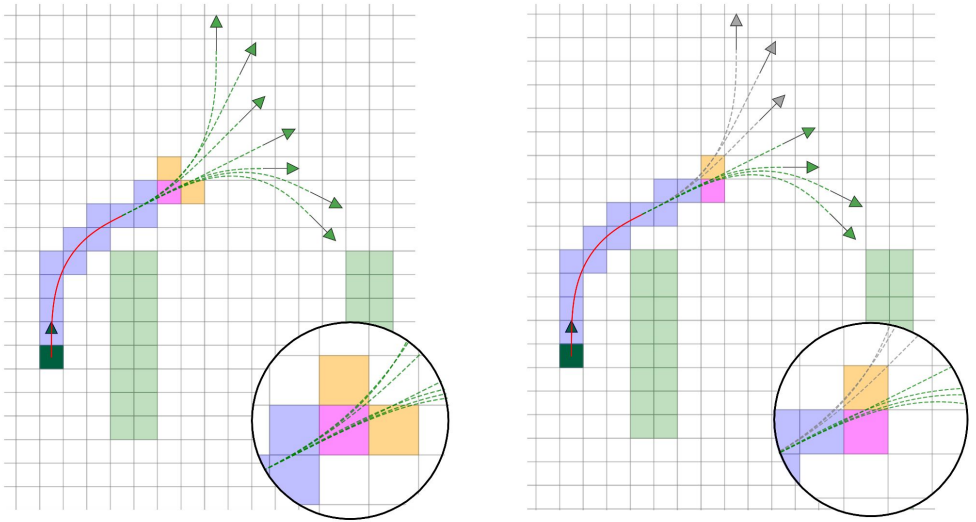
State Lattice (graph of primitives) → Mesh Graph (graph of cells).

Key Insight: Cell-Centric Expansion

- Do not jump from state to state via full primitives.
- Instead, step through the environment cell-by-cell.
- Reason simultaneously about the grid cell and the set of primitives passing through it.

The Benefit: Early Pruning

- We can detect collisions or high costs immediately as we enter a cell.
- Allows pruning unpromising branches before a primitive is fully traversed/expanded.



*Intuition behind MeshA**: The current cell (magenta) stores a set of active primitives (green). Moving to a specific neighbor (orange) effectively filters this set, keeping only the primitives that actually traverse that neighbor and discarding the rest (gray).

The Search Space: Mesh Graph

We define our search space as the Mesh Graph, its nodes are Extended Cells.

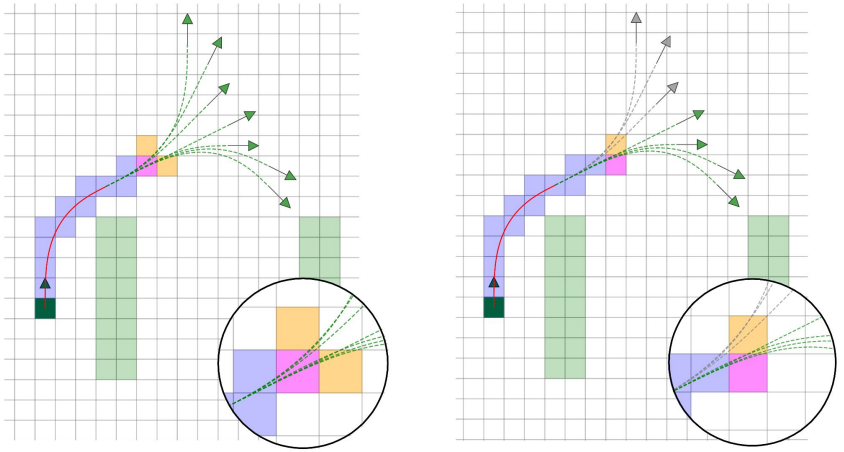
Definition: Configuration of Primitives Ψ

A set of pairs $\Psi = \{(prim_1, k), \ldots, (prim_n, k)\}$, where k is the index of the current cell in the primitive's collision trace.

Definition: Extended Cell u

$$u = (i, j, \Psi)$$

Encapsulates the grid location (i, j) and the state of all motion primitives currently traversing this cell.



Example of Extended Cell u = (i, j, Ψ):

- The magenta grid cell corresponds to the location (i, j).
- The green lines represent the Configuration Ψ: the set of primitives passing through this cell (i, j) at a specific step k.

MeshA* Efficiency: Heuristics & Pruning

Primitive Endpoints: For extended cell u=(i, j, Ψ), Finals(u) is the set of all possible states where primitives from Ψ terminate.

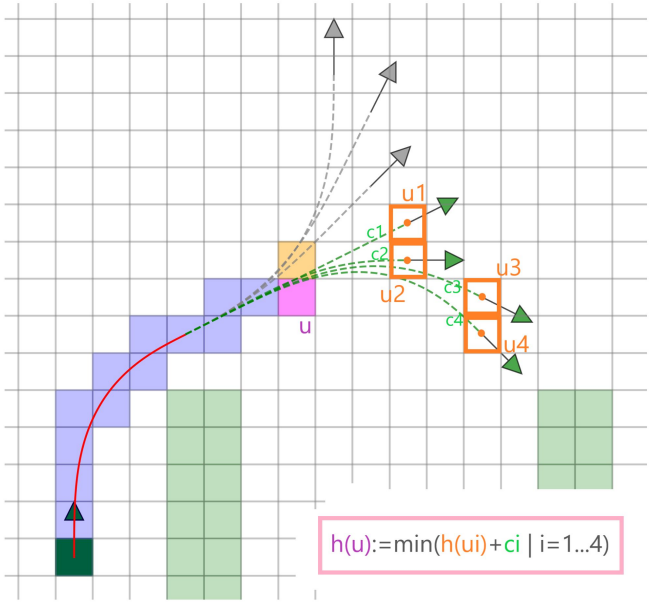
Cell-Level Heuristic:

- MeshA* re-evaluates heuristic at *every grid cell*:

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

- This allows to detect unpromising directions early and prune branches without traversing the full primitive.

Terminal Pruning: If all states in Finals(u) have already been expanded (closed), we stop expanding u immediately.



Visualization of Finals(u): orange cells.

Theoretical Properties: Equivalence to State Lattice

Theorem (Equivalence of Optimal Pathfinding)

Searching for a least-cost trajectory between $s_a = (i_a, j_a, \theta_a)$ and $s_b = (i_b, j_b, \theta_b)$ is equivalent to:

1. *Finding a least-cost path on the* **Mesh Graph** *between $u_a = (i_a, j_a, \Psi_a)$ and $u_b = (i_b, j_b, \Psi_b)$.*
2. *Recovering the trajectory using a constructive algorithm.*

Key Implications:

• **Optimality**: MeshA* guarantees finding the path with minimal cost (identical to LBA*).

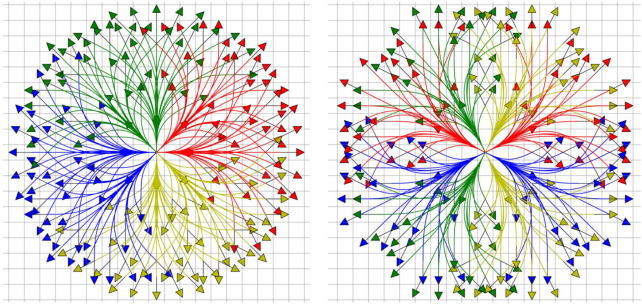
• **Completeness**: If a solution exists in the State Lattice, MeshA* will find it.

Experimental Setup

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

Motion primitives:



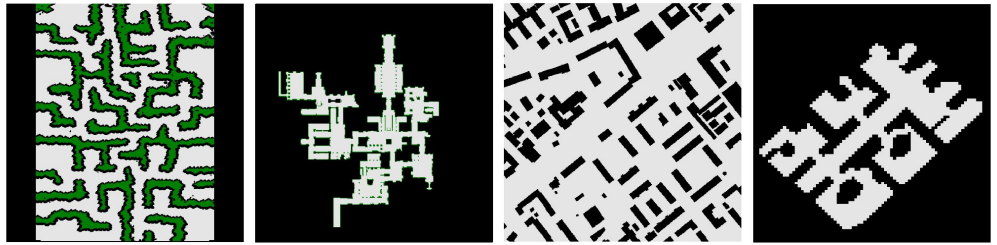
total: 24×16 =384

Baselines:

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

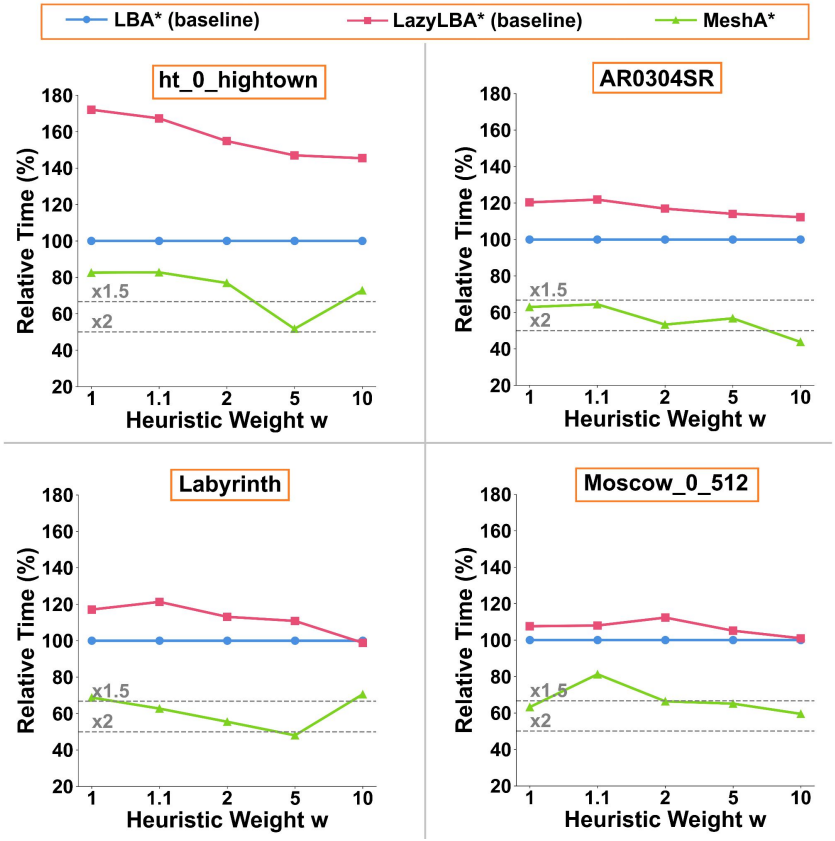
Heuristics:

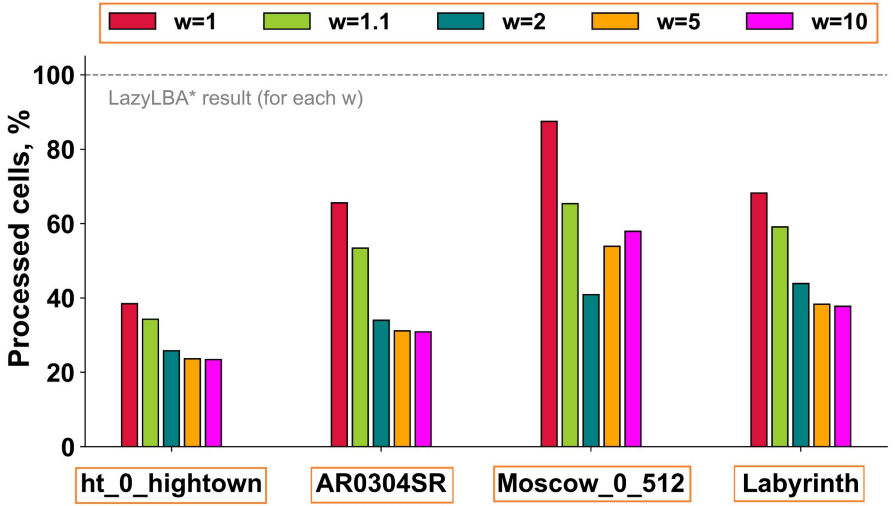
- **LBA*:** Euclidean distance
- **MeshA*:** a derivative of Euclidean one
- **Weights:** w ∈ {1, 1.1, 2, 5, 10}
  (factors for heuristics of both MeshA* and LBA*)



Maps from https://movingai.com/benchmarks/grids.html

Empirical Results: Runtime & Efficiency

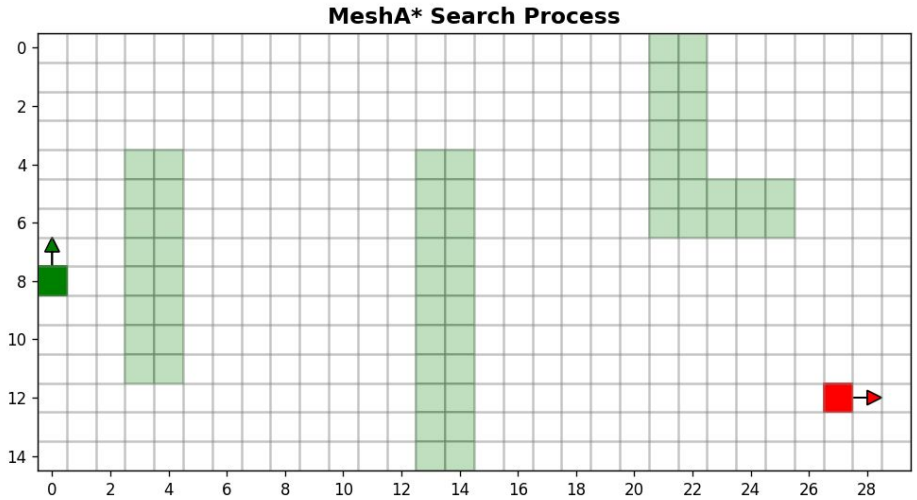




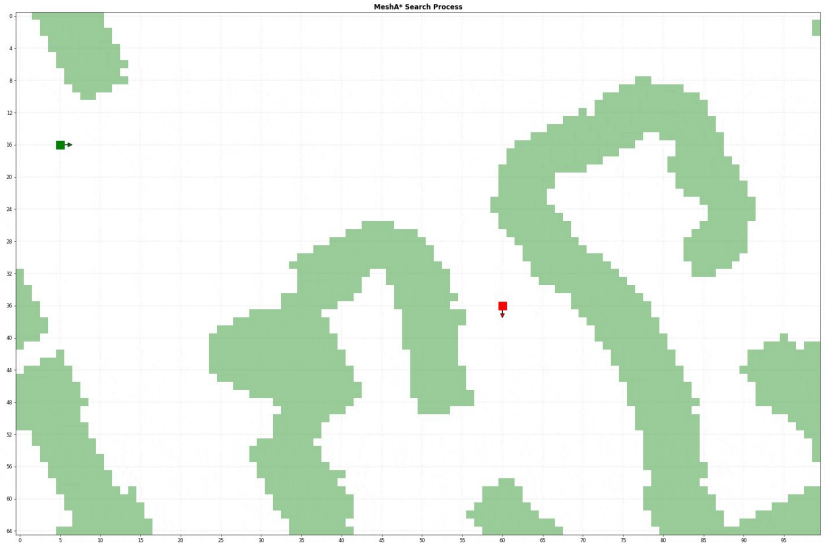
Key Results for MeshA*:

- 1.5x - 2x faster than LBA* and LazyLBA*.
- Processes ≈ 50% fewer grid cells than LazyLBA*.

MeshA* in Action



Cell-by-Cell Propagation The search expands grid cells incrementally, strictly guided by the geometry of primitive bundles.



Effective Branching Factor. In the end the visualization retains only one **representative** primitive per indistinguishable bundle to illustrate the actual search complexity.

Thank you for your attention!

For further questions about MeshA*:

agrinscience@gmail.com

Code

ArXiv