A Myopic Monte Carlo Strategy for the Partially Observable Travelling Salesman Problem

Andrew R. Buck and James M. Keller

Department of Electrical and Computer Engineering
University of Missouri
Columbia, Missouri, USA

WCCI 2016 July 25, 2016 Vancouver, Canada

Outline

- Motivation and Background
- The Partially Observable Travelling Salesman Problem (PO-TSP)
  - Generating Problem Instances
  - Client/Server Architecture
- Agent Policies for the PO-TSP
  - Greedy Policy
  - Myopic Monte Carlo Policy
- Experiments and Results
- Conclusions and Future Work

An Example Problem

- Suppose you are tasked with finding the shortest path that visits a set of flags within a cave-like environment.
- You can only see your immediate surroundings.
- More of the cave is revealed as you explore.
- How should you navigate the cave in order to minimize the total distance traveled?

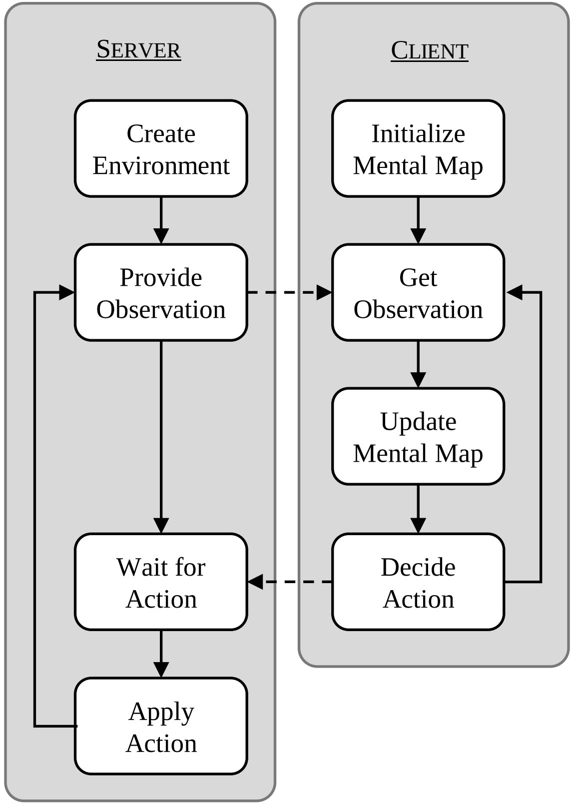
Motivation

- Agent-based models for navigation
  - This project grew out of a need to create models for agent behavior in uncertain environments.
  - Agents may not have a complete map of the environment and must navigate with only partial information.
    - Walls or obstacles may limit visibility.
    - Agents use a *mental map* to represent the environment and plan actions.
- Applications
  - Robotic mapping and navigation
  - Search and rescue
  - Developing intelligent agents
  - Modeling human decision-making behavior under uncertainty

Developing a Benchmark Problem

- The objective of the travelling salesman problem (TSP) is to find the shortest route for an agent that visits a given set of waypoint locations.
- The TSP provides a generic problem for the agent to solve that can be adapted to many different problem domains.
- The partially observable TSP (PO-TSP) restricts the visibility of the environment to what the agent can see locally.
  - Agents must decide how to acquire new information and act on existing knowledge.
- For this work, we focus on developing agent strategies for PO-TSP problems in discrete grid-world domains.

Overview of the PO-TSP

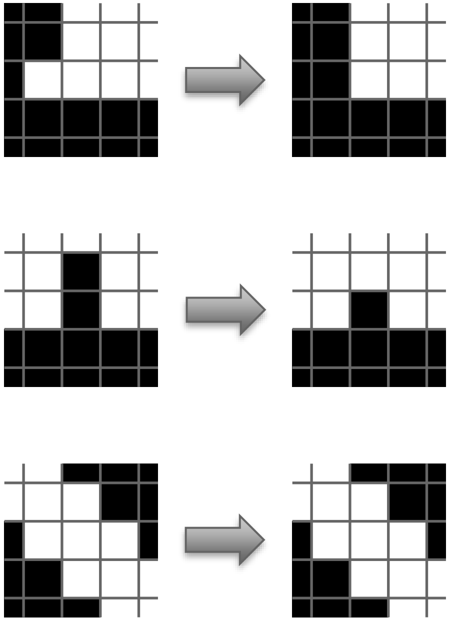


- The PO-TSP is implemented using a client/server architecture.
- The server maintains all environment variables and provides observations to the agent.
- The agent uses observations to update its mental map and decide actions.
- The server applies the agent's actions and determines if the goal conditions have been met.

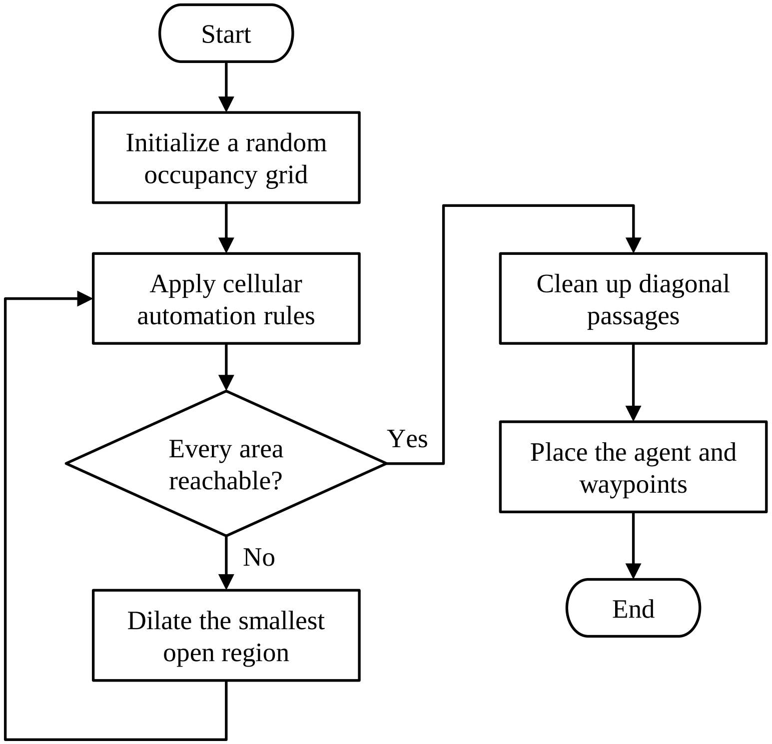
Cellular Automata

- The grid-world environments are generated using cellular automation rules similar to Conway’s Game of Life.

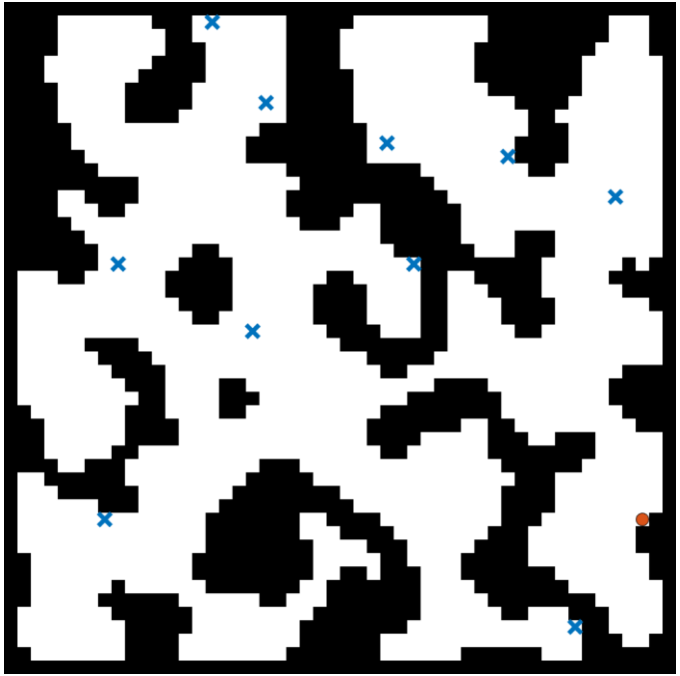
- Our implementation uses the following rules based on a cell’s 8 neighbors:
  – An open cell becomes occupied if it has fewer than 3 open neighbors.
  – An occupied cell becomes open if it has more than 4 open neighbors.



Environment Generation

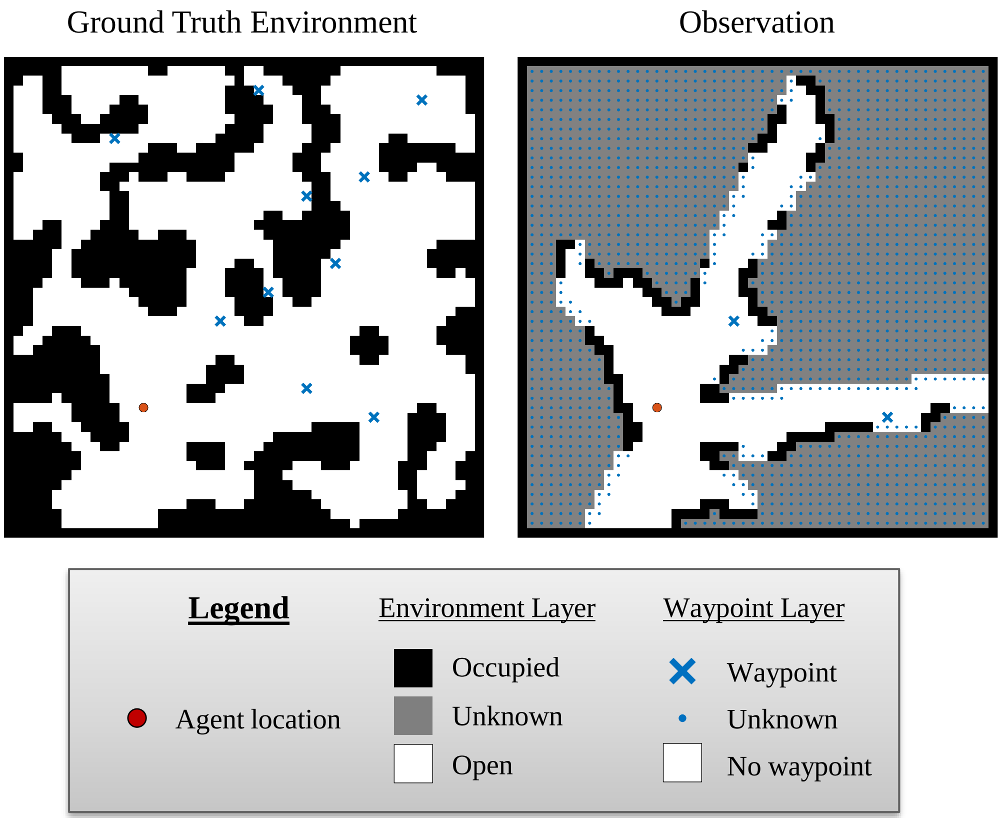


Place the agent and waypoints

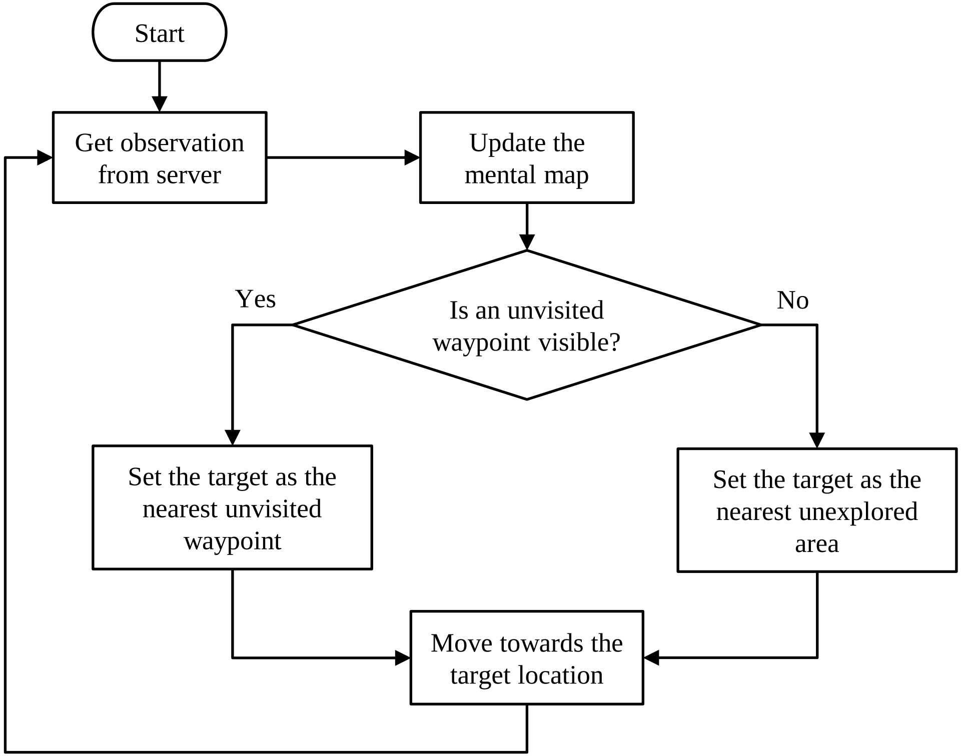


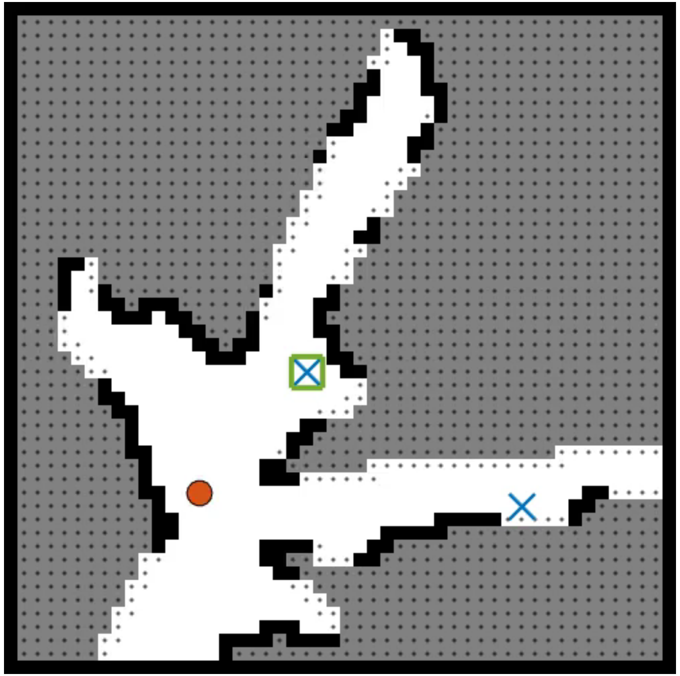
Observations

- The server provides the agent with observations containing
  - An environment grid layer where each cell is either *Open*, *Occupied*, or *Unknown*
  - A waypoint grid layer where each cell is either *Waypoint*, *No Waypoint*, or *Unknown*
  - The current agent location
- Line-of-sight visibility is computed using Bresenham's line algorithm.
- The agent maintains the history of observations as its *mental map*.



A Greedy Policy

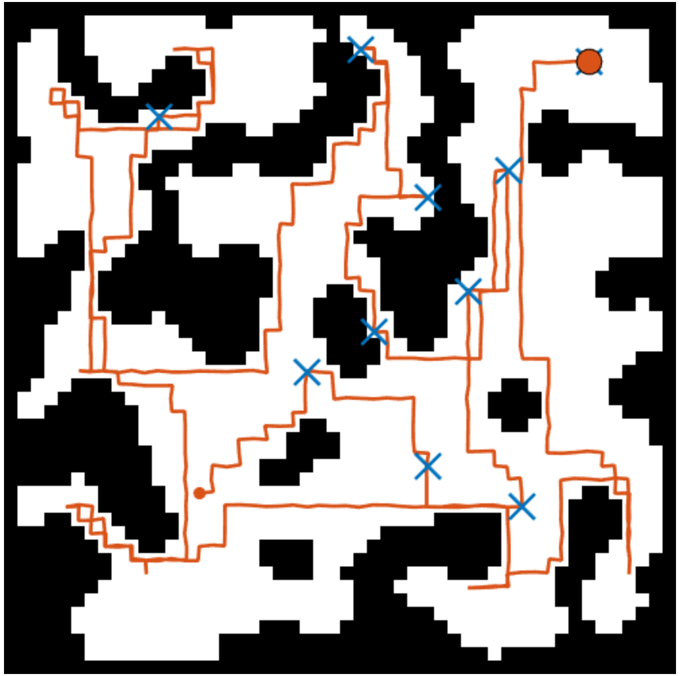




Improvements to the Greedy Policy

- Greedy policy
  - Strengths:
    - Simple to implement
    - Low computation cost
  - Weaknesses:
    - Prioritizes visible waypoints over unexplored areas
    - Leaves areas only partially explored
    - High amount of backtracking

- Improvements:
  - Finish exploring a region before moving on to the next waypoint
  - Use random sampling to compute the value of exploring each new area

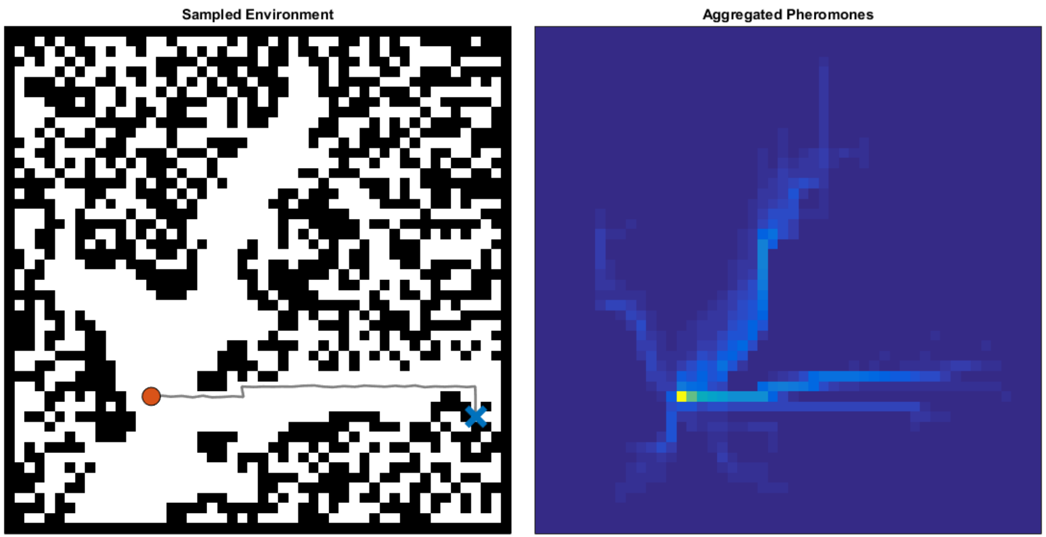


Monte Carlo Sampling

Use a persistent pheromone map to aggregate the shortest paths to many possible waypoints.

For many iterations:

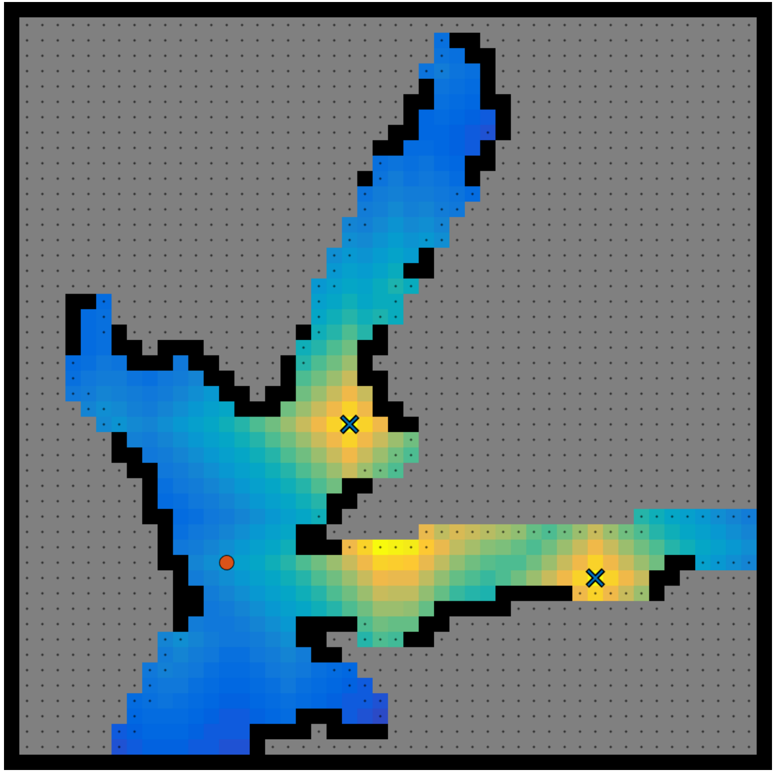
1. Sample an environment from the mental map
2. Sample a single waypoint
3. Get the shortest path from the agent to the waypoint if a path exists
4. Deposit pheromone on the shortest path



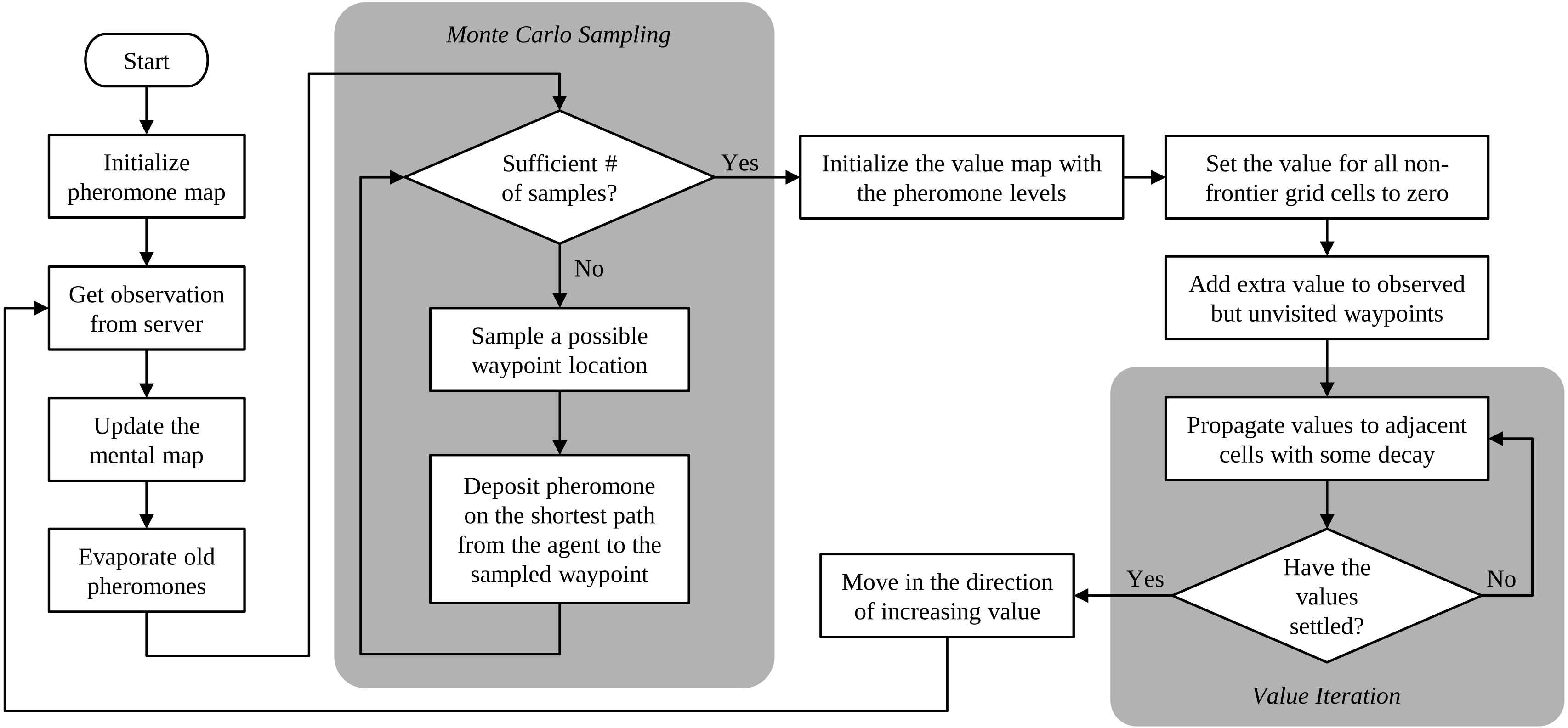
Value Iteration

Define a value map over the environment representing the attractiveness of each grid cell.

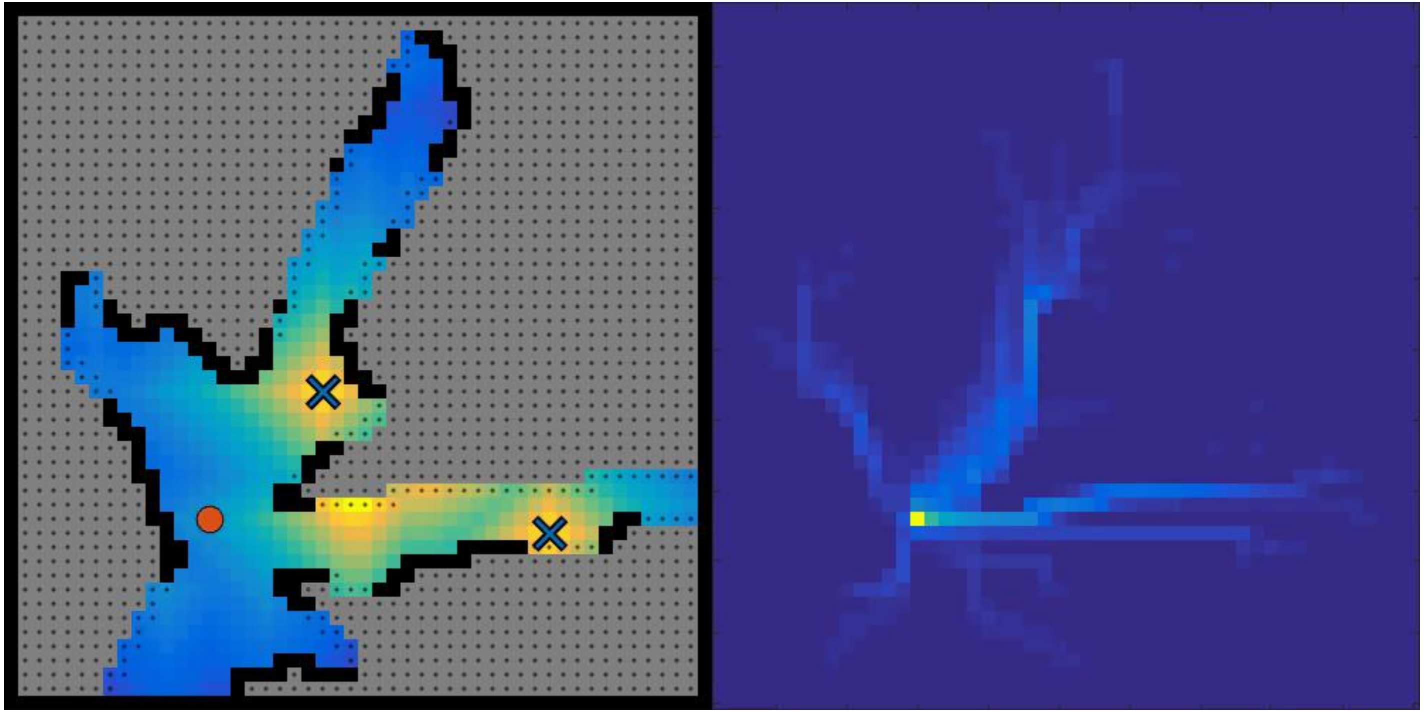
1. Copy values from the pheromone map in observed regions
2. Set the value for all non-frontier grid cells to zero
3. Add extra value to observed but unvisited waypoints
4. Propagate values to adjacent cells with some decay until the values settle
5. Move in the direction of increasing value



A Myopic Monte Carlo Policy



A Myopic Monte Carlo Policy



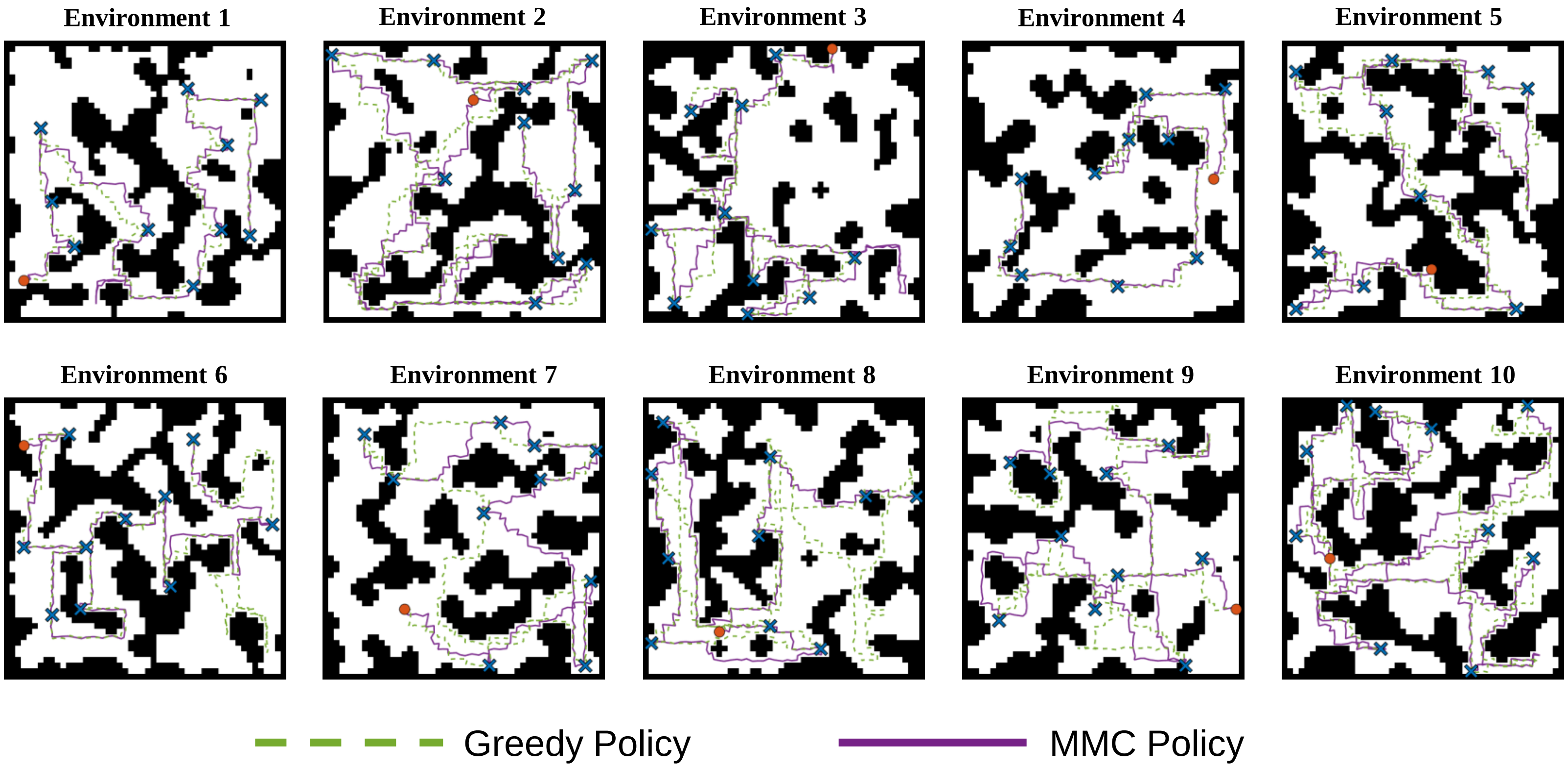
MMC Policy Parameters

- The Myopic Monte Carlo (MMC) policy has many more adjustable parameters then the greedy policy.
  - Number of samples (n)
    - We use 1000 samples in our experiments
  - Evaporation rate (γ)
    - Percentage of the pheromone that is maintained between movement actions
    - We use $\gamma = \{0.9, 0.95, 0.99\}$
  - Discount factor (λ)
    - Percentage of the value that is spread to adjacent grid cells
    - We use $\lambda = \{0.9, 0.95, 0.99\}$
  - Waypoint weight (η)
    - Amount of extra value given to observed but unvisited waypoints
    - We use $\eta = \{1, 10\} \times maxPheromone$

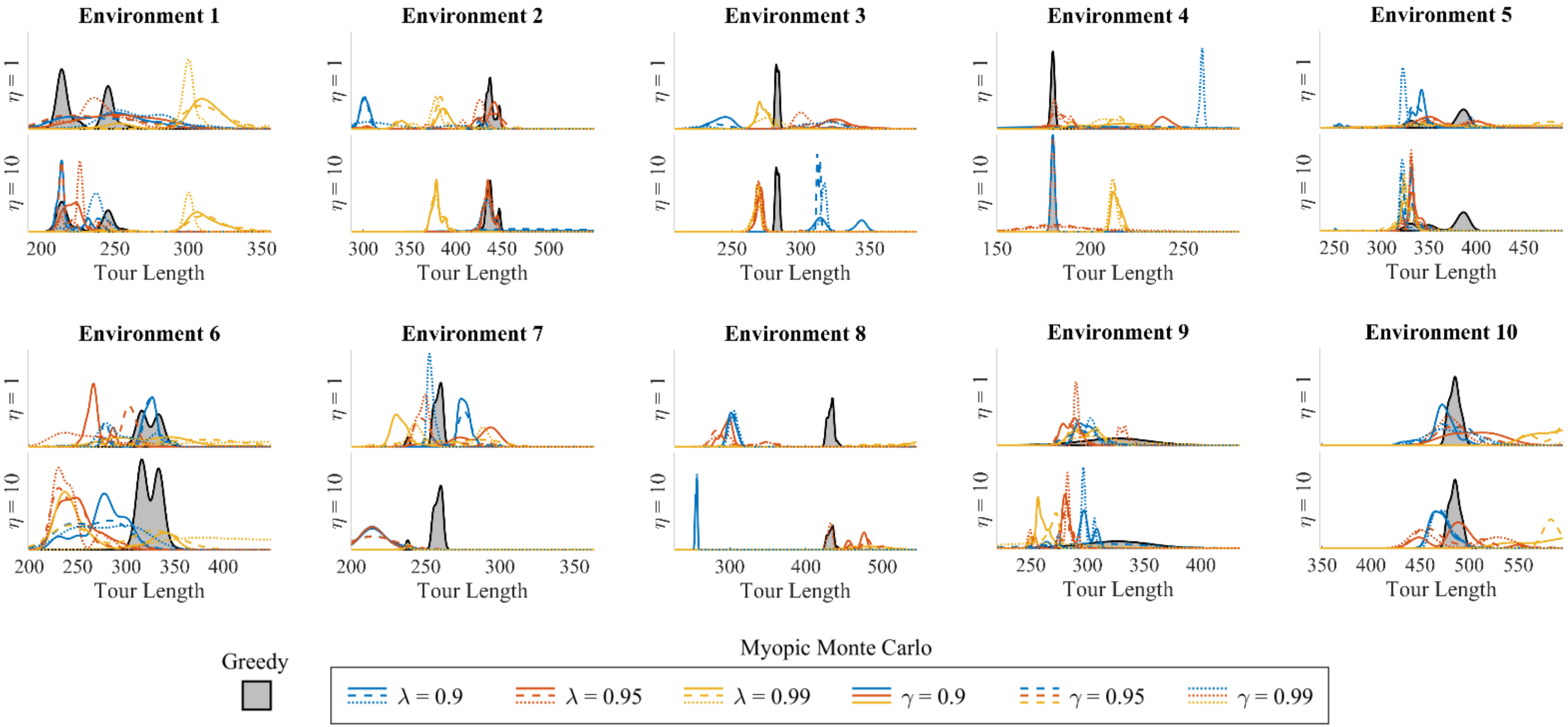
Experiments

- To compare the greedy and MMC policies, we create 10 benchmark problem sets.
  - 50×50 grids
  - 10 waypoints with a minimum separation of 10 grid cells
- We then run 100 trials each of the greedy policy and 18 different parameter configurations of the MMC policy.
  - Varying $\gamma = \{0.9, 0.95, 0.99\}$, $\lambda = \{0.9, 0.95, 0.99\}$, and $\eta = \{1, 10\}$
- We report
  - An example solution plotted for each problem set for both the greedy and MMC policies
  - The distribution of solution path lengths for each method
  - Average improvement of the MMC policy over the greedy policy

Example Solutions



Distribution of Solution Lengths



γ = Evaporation Rate λ = Discount Factor η = Waypoint Weight

Average Improvement

- To draw some general conclusions, we compute the difference between the average path lengths of the various MMC policy parameterizations and the greedy policy.
- The values in the table show the average difference over all 10 environments.
  - Negative values indicate better performance by the MMC policy.

Evaporation Rate (γ)	Waypoint Weight (η) = 1			Waypoint Weight (η) = 10		
	Discount Factor (λ)			Discount Factor (λ)		
	0.9	*0.95*	*0.99*	*0.9*	*0.95*	*0.99*
0.9	-30.1	-8.4	27.5	4.7	-18.4	27.7
0.95	-23.4	-14.9	33.7	-22.3	-19.4	51.5
0.99	-18.5	21.0	68.7	-30.7	-21.1	64.2

Conclusions and Future Work

- For certain parameter settings, the MMC policy can outperform the greedy policy.
  - None of the tested parameterizations was the best in every environment.

- Next steps:
  - Identify environment features that impact performance and cluster similar environments
  - Use methods such as Ant Colony Optimization and Monte Carlo Tree Search to plan farther into the future
  - Develop a more scalable mental map representation