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

Andrew R. Buck, *Student Member, IEEE* and James M. Keller, *Life Fellow IEEE*
Department of Electrical and Computer Engineering
University of Missouri
Columbia, Missouri, USA

***Abstract*—In this paper, we present two greedy, myopic algorithms for solving the partially observable travelling salesman problem. Although not optimal from a decision-theoretic viewpoint, these strategies are shown to perform reasonably well under the uncertain conditions of the environment. The first algorithm is a strictly greedy algorithm and has no tunable parameters, whereas the second algorithm uses Monte Carlo sampling to determine likely configurations of the environment and applies value iteration to pick an action. We present both approaches with illustrative examples and empirically demonstrate their relative strengths and weaknesses.**

***Keywords*—Uncertain environments; agent planning; travelling salesman problem; decision-making; procedural content generation**

I. Introduction

Navigation in an unfamiliar environment can be complicated by a lack of information. In many cases, a decision-making agent may have limited prior knowledge and must balance the exploration of new areas with the exploitation of known objectives. Walls or other obstacles may restrict the agent's view and limit the information available for planning. In these partially observable environments, agents can use a *mental map* to represent the information that has been collected thus far and plan a future course of action. These types of problems can arise in many fields, including robotic mapping [1], search and rescue operations [2], and human plan recognition [3]. Some of our previous work has focused on the correspondence problem of matching an approximate spatial configuration of landmarks to known ground truth [4], [5], and modeling the behavior of moving agents [6], [7]. Our goal in this paper is to extend these ideas by laying the groundwork for a benchmark problem that can be used to test models of agent behavior in unknown environments. Ultimately, we plan to use problems like these to study the effects of uncertainty on human judgement and spatial awareness and develop realistic agent models that can anticipate human behavior.

The travelling salesman problem (TSP) has been used as a benchmark for combinatorial optimization algorithms for decades. It comes from the class of NP-hard problems that make it interesting to study and challenging to solve. Several variations on the original definition have emerged over the years, including the probabilistic TSP [8], in which tours must generalize over multiple problem instances, and the physical TSP [9], in which agents must provide actions in real-time in order to navigate a vehicle that obeys the laws of kinematics. Our definition of the partially observable TSP, which we describe below, provides a simulation environment similar to the physical TSP, but defined on a discrete domain in which the environment is not completely known a priori. We use a grid to represent the world out of convenience, but the problem could conceivably be extended to continuous domains. In [9], the authors use Monte Carlo Tree Search (MCTS) [10] to design a control strategy for steering the physical vehicle. While MCTS methods have previously been adapted for partially observable domains [11], [12], we focus here on a simpler myopic approach that is shown to achieve promising results on our example problems.

The remainder of this paper is organized as follows. Section II presents the details of the partially observable travelling salesman problem, including the creation of our procedurally generated environments. Section III describes the algorithms we used for this problem. Section IV covers the experiments we ran, and Section V gives our conclusions and ideas for future work.

II. The Partially Observable Travelling Salesman Problem

The travelling salesman problem is a combinatorial optimization problem that takes a set of waypoints and seeks to find the shortest path that visits each one. Given a set of waypoints $\{w_1, \dots, w_N\}$ and a distance matrix D, where d_{ij} is the distance between waypoints w_i and w_j, the task is to find an ordering of the waypoints π that minimizes the tour length,

$$\sum_{i=1}^{N-1} d_{\pi(i)\pi(i+1)} + d_{\pi(N)\pi(1)}. \tag{1}$$

We implement the TSP on a grid-based map in which each grid cell is either open or blocked. Waypoints are assigned to open grid cells and the acting agent is placed in one of the open cells. Each time step, the agent moves into an adjacent open cell in one of the four cardinal directions. The agent's goal is to move in such a way as to visit each of the waypoints with the fewest steps possible. In our implementation, the episode is complete when the agent visits the last waypoint; we do not require the agent to return to its starting location.

A. Partial Observability

In the fully observable case, the grid-based TSP can be solved directly by constructing the distance matrix according to the obstacles in the environment and using any number of existing TSP algorithms. One popular approach that provided

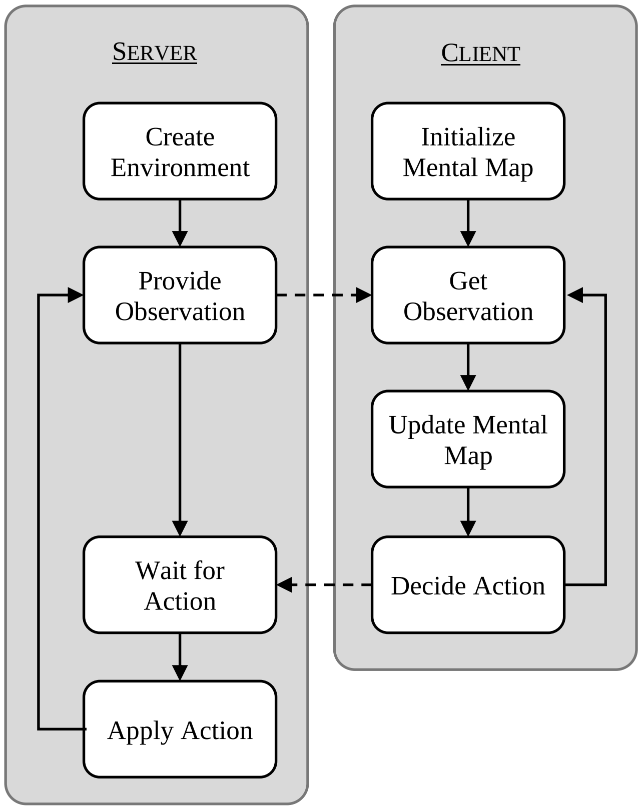

Fig. 1. Block diagram of the server/client architecture of the PO-TSP.

inspiration for our methods is ant colony optimization (ACO) [13], in which the environment is modified with pheromones to indicate favorable paths. Our main interest, however, is in understanding the effects of partial observability in this domain. We therefore proceed to define the partially observable traveling salesman problem (PO-TSP) for grid-based maps.

The PO-TSP is implemented using a client/server architecture (Fig. 1) in which a server program operates as the referee, which validates actions given by the client and provides an observation that the client can use to update its own mental map. The client defines the behavior of the agent in the environment and works with the limited information provided by the server in the form of observations. The server computes the visible region of the map based on the agent's current location and returns the information about the environment within this region. For our grid-based worlds, this consists of

- An environment grid layer, where each grid cell comes from the three element set {*True*, *False*, *Unknown*} indicating whether or not the cell is traversable;
- A waypoint grid layer, where each grid cell comes from the three element set {*True*, *False*, *Unknown*} indicating the presence of a waypoint;
- The current agent location.

We maintain separate layers for the environment occupancy and the waypoints because depending on the visibility method used, it may be possible to know that a grid cell is open, but not know

Algorithm 1 Environment Generation

```
1:  function GENERATE_ENVIRONMENT($m$, $n$, $p_0$, $r_b$, $r_d$, $k$)

2:      $E \leftarrow m{\times}n$ grid, where $prob(E_i = True) = p_0$
3:      while (# of connected components in $E$) > 1 do
4:          for $k$ iterations do
5:              $E \leftarrow$ CELLULAR_AUTOMATA($E$, $r_b$, $r_d$)
6:              $E \leftarrow False$ for all border grid cells
7:          end for
8:          if (# of conn. comp. in $E$) > 1 then
9:              $Z \leftarrow$ the smallest connected components in $E$
10:             $E \leftarrow Z \oplus [0\ 1\ 0;\ 1\ 1\ 1;\ 0\ 1\ 0]$        // Dilation
11:         else if (# of conn. comp. in $E$) < 1 then
12:             $E \leftarrow m{\times}n$ grid, where $prob(E_i = True) = p_0$
13:         end if
14:     end while
15:     $E \leftarrow$ REMOVE_DIAGONAL_PASSAGES($E$)

16:     return $E$
```

if it contains a waypoint. For example, when looking around a corner, one might be able to infer that a cell is open but not have a clear enough view to determine if the cell contains a waypoint. The details of the visibility computation are omitted for brevity, but the general approach used in our experiments is to repeatedly dilate the visible region around the agent and validate visible cells that have a line of sight back to the agent using Bresenham's line algorithm [14]. For a more in-depth overview of this problem, see [15]. Note that unlike many autonomous mapping algorithms, the agent's location within the environment is known.

Upon receiving an observation from the server, the client updates its mental map of the environment, which includes the agent's location, layers matching those provided by the observation, and an additional layer indicating which grid cells have been visited. Any environment layer cells that are observed to be open or blocked are set as such in the mental map and likewise, any cells that are observed to either have or not have a waypoint are set as such. If the client has some prior knowledge about the environment, it can use this as a heuristic to make changes to the mental map that might not be observed directly. For example, in the cave-like environments generated in the next section, we always apply a one-cell border to the edge of the environment to keep the agent from leaving the environment. Additionally, because we ensure that all open locations are reachable in our environments, any regions that become completely closed off in the mental map can have their interiors set to *False*.

It is the client's responsibility to define a policy for a given mental map that dictates the next action for the agent. We present two methods in Section III, although any policy that produces an action for any given observation can be used.

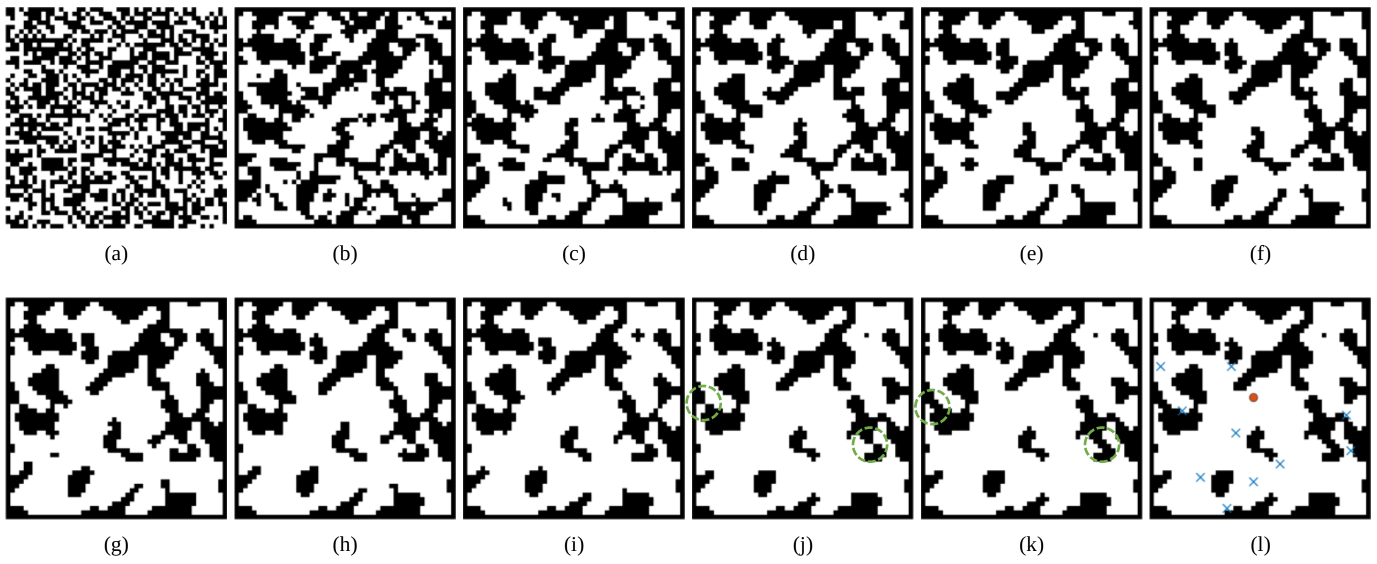
Fig. 2. Generation of a random environment for the PO-TSP using a cellular automation. (a) The grid starts with each cell randomly initialized according to the start probability. (b)-(j) Each iteration, the cellular automation rules are applied and the smallest continuous region is dilated. (k) A cleanup process removes passageways that are only connected diagonally to improve the reliability of the visibility algorithm (circled). (l) The agent is placed in a random initial location, shown as a red circle and waypoints are placed with a minimum spacing, shown as blue crosses, enlarged for clarity.

B. *Environment Generation*

To generate a problem instance for the PO-TSP, we use an iterative method that combines cellular automation with image dilation. The complete pseudocode is given in Algorithm 1 and an example of the process is shown in Fig. 2.

To start, a random $m \times n$ grid is generated where each cell is assigned *True* with probability p_0 and *False* with probability $1-p_0$. In our experiments, we use 50×50 grids with $p_0 = 0.45$. The algorithm then iterates through k steps of cellular automation rules, as in Conway's Game of Life [16]. (All of our environments are generated with a single step using $k = 1$.) Many different rules can be applied to two-dimensional cellular automata [17]. Our specific implementation updates each grid cell by looking at the 8 neighboring cells in the Moore neighborhood and changes a *False* cell to *True* if there are more than 4 *True* neighbors (birth rate, r_b) and changes a *True* cell to *False* if there are less than 3 *True* neighbors (death rate, r_d). This is represented by the CELLULAR_AUTOMATA function in line 5 of the pseudocode.

We then determine the smallest connected components, consisting of a contiguous set of 4-connected open cells, and dilate these sets by assigning any 4-connected neighbor of the set to *True* (lines 9-10). There may be multiple sets with the same number of open cells, in which case all are considered. This serves to expand the open area and eventually create a fully-connected environment with no unreachable areas. If after a single dilation there are still multiple independent connected components, an additional round of cellular automation and dilation occurs. This continues until there is only a single connected component. If all open areas are removed (as can happen for certain values of r_b and r_d), the algorithm restarts from the beginning.

As a final post-processing step, we remove any passageways that are only connected diagonally (line 15 of the pseudocode and Figs. 2j and 2k). This improves the reliability of the visibility computation, which is used to get the agent's observation at a given location. The end result is an environment that has an organic, cave-like structure and provides many concealed areas that are not visible without exploration. We randomly place the waypoints and the agent into the open cells of this environment with an optional minimum separation.

III. AGENT POLICIES FOR THE PO-TSP

After receiving an observation from the server, the client program updates its internal mental map representation of the environment and picks an action for the agent. We define a mental map structure M which is updated after each observation and contains the following attributes.

- *M.env* = an $m \times n$ grid, where each grid cell comes from the three element set {*True*, *False*, *Unknown*} indicating whether or not the cell is traversable.
- *M.wpt* = an $m \times n$ grid, where each grid cell comes from the three element set {*True*, *False*, *Unknown*} indicating the presence of a waypoint.
- *M.visited* = an $m \times n$ grid, where each grid cell is either *True* or *False* indicating if the cell has been visited.
- *M.pos* = (a_x, a_y) = the current position of the agent.

Fig. 3a shows what an agent's mental map might look like at the start of an episode. We next describe two policies that take the current mental map as input and return an action.

A. *Greedy Policy*

One of the simplest policies for the PO-TSP is a strictly greedy policy that first looks to see if there are any unvisited waypoints visible in the mental map. If so, the agent moves in the direction of the closest waypoint. If no waypoints are visible, the agent moves toward the nearest unexplored area. Provided that all areas in the environment are reachable, this policy will

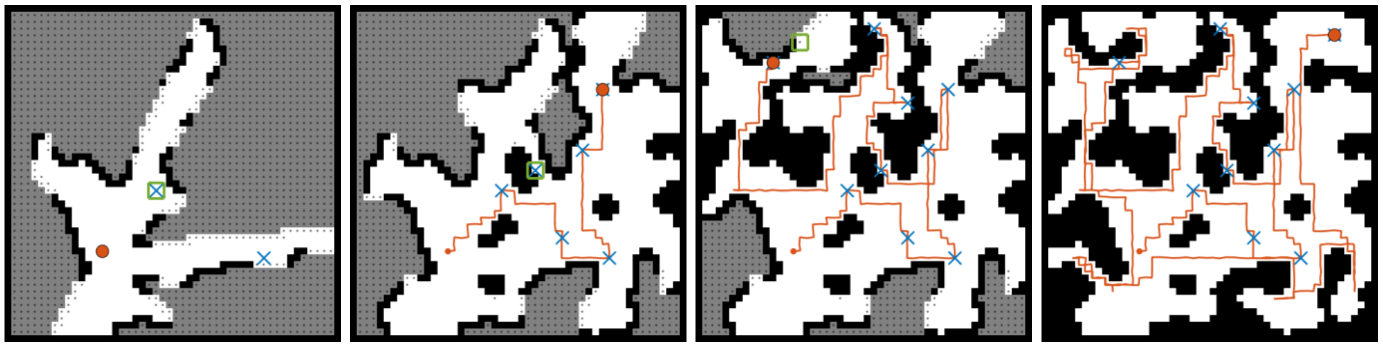
(a) (b) (c) (d)

Fig. 3. Some selected moments from the greedy policy's solution for the PO-TSP. Symbols are enlarged for clarity. (a) The initial mental map shows only what is visible from the agent's starting location (red circle), which includes two waypoints (blue crosses). The closest of these is chosen as the target objective (green square). Grey areas indicate unknown areas of the environment and dots signify the possibility of a waypoint. (b) The first five waypoints are acquired greedily and the sixth target is chosen as a waypoint that was discovered along the route, but requires the agent to backtrack. (c) Nine targets are acquired by always moving toward the nearest unvisited waypoint if one is visible, or the nearest unexplored area otherwise. (d) The final target is hidden behind a corner that was not fully explored on the first pass and is not discovered until the entire environment has been explored.

Algorithm 2 Greedy Policy for the PO-TSP

```
1:  function GREEDY_POLICY(M)

        // Get target locations
2:      $T \leftarrow \{\forall(x, y)$ s.t. $M.wpt_{(x,y)} \wedge \neg M.visited_{(x,y)}\}$
3:      if $T = \varnothing$ then
4:          $T \leftarrow \{\forall(x, y)$ s.t. $M.wpt_{(x,y)} = Unknown\}$
5:      end if

        // Compute the shortest path from the current agent
        position to one of the target locations
6:      $D \leftarrow$ SHORTEST_PATH($T$, $M$)

        // Return the next location from $D$
7:      $A \leftarrow D.next$

8:      return $A$
```

eventually pass through all waypoints. Algorithm 2 gives the pseudocode for this policy. We assume the availability of a routine, SHORTEST_PATH, for computing shortest paths through grid-based worlds. This function returns a set of points, in order, that would lead the agent from its current position to the nearest target location. One approach for this is to implement Dijkstra's algorithm seeded at the target locations and stop when the agent is reached. An example of the greedy policy evaluating an environment is shown in Fig. 3. This example shows that while the greedy agent may ultimately visit all waypoints, the path taken may be far from optimal.

The greedy policy, while simple to implement, has several shortcomings. Because visible waypoints are strictly prioritized over exploring new areas, greedy agents will often leave an area only partly explored before moving on. This can cause the agent to perform a great deal of unnecessary backtracking in order to return to areas that were only partially explored. Furthermore, as a greedy algorithm, the agent does no forward planning to formulate routes beyond the next immediate waypoint. Despite these flaws, the greedy algorithm performs reasonably well on many example environments.

B. *Myopic Monte Carlo Policy*

To improve upon the greedy policy, we now introduce a myopic Monte Carlo (MMC) policy that has many similarities with swarm intelligence techniques. For the MMC policy, we define an additional persistent layer P, which is an $m \times n$ grid initialized with all zeros that serves to aggregate "pheromones" in promising areas of the environment. This layer is modeled after the pheromone trails deposited by ants in ACO. The pheromone layer is maintained between observations and the values decay each time step with an evaporation rate γ. The pheromones serve as a form of memory, allowing the agent to remember its previous goals and exhibit some momentum toward achieving those original goals, even if new information would suggest that better goals exist elsewhere. We will show that this allows agents to finish exploring some areas before moving on, ameliorating a key weakness of the greedy policy.

The MMC policy is divided into two phases. The complete pseudocode is given in Algorithm 3. In the first phase (lines 3-10), we perform Monte Carlo sampling on the mental map to construct a distribution of possible ground truth environments. For each sample S, we define the following attributes.

- *S.env* = an $m \times n$ grid, sampled from *M.env* where any cells marked as *True* or *False* are copied directly and cells that are *Unknown* are sampled as either *True* or *False* with equal likelihood.
- *S.wpt* = an $m \times n$ grid, sampled from *M.wpt* in which only a single grid cell is marked as *True* and all others are marked as *False*. The *True* cell is chosen uniformly from the cells of *M.wpt* that are either *True* or *Unknown*.
- *S.pos* = *M.pos* = the current position of the agent.

Note that we only sample a single waypoint location for each sample. The aggregation of these individual samples forms a uniform distribution over all possible waypoint locations.

Algorithm 3 Myopic Monte Carlo Policy for the PO-TSP

```
1:  function MMC_POLICY($M, P, N, \gamma, \eta, \lambda, \varepsilon$)

      // Evaporate old pheromones
2:    $p_{(x,y)} \leftarrow \gamma p_{(x,y)}\ \forall (x, y)$

      /*** Monte Carlo Sampling ***/
3:    for $N$ iterations do

        // Sample an environment
4:      $S.env \leftarrow$ randomly determinized from $M.env$
5:      $S.wpt \leftarrow$ randomly determinized from $M.wpt$
6:      $S.pos \leftarrow M.pos$

        // Compute the shortest path from the current
        agent position to the sampled waypoint location
7:      $T \leftarrow \{\forall (x, y)$ s.t. $S.wpt_{(x,y)} = True\}$
8:      $D \leftarrow$ SHORTEST_PATH($T, S$)

        // Add the shortest path to the aggregation map
9:      $p_{(x,y)} \leftarrow p_{(x,y)} + 1\ \forall (x, y) \in D$

10:   end for

      /*** Value Iteration ***/
11:   $v_{(x,y)} \leftarrow 0\ \forall (x, y)$
12:   $v_{(x,y)} \leftarrow p_{(x,y)}\ \forall (x, y)$ s.t. $M.wpt_{(x,y)} \neq False \wedge$
        $\neg M.visited_{(x,y)}$
13:   $v_{(x,y)} \leftarrow \eta(\max_i p_i)\ \forall (x, y)$ s.t. $M.wpt_{(x,y)} = True \wedge$
        $\neg M.visited_{(x,y)}$

14:   $keepGoing \leftarrow True$
15:   while $keepGoing$ do
16:     for all $(x, y)$ do
17:       $N \leftarrow \{(x-1, y), (x+1, y), (x, y-1), (x, y+1)\}$
18:       $v'_{(x,y)} \leftarrow \lambda \max_{i \in N} v_i$
19:     end for
20:     $v''_{(x,y)} \leftarrow \max(v_{(x,y)}, v'_{(x,y)})\ \forall (x, y)$
21:     if $\sum_{(x,y)} (v_{(x,y)} - v''_{(x,y)})^2 < \varepsilon$ then
22:       $keepGoing \leftarrow False$
23:     end if
24:     $v_{(x,y)} \leftarrow v''_{(x,y)}\ \forall (x, y)$
25:   end while

      // Get the neighbor cell with the greatest value
26:   $(a_x, a_y) \leftarrow M.pos$
27:   $N \leftarrow \{(a_x-1, a_y), (a_x+1, a_y), (a_x, a_y-1), (a_x, a_y+1)\}$
28:   $A \leftarrow \arg\max_{i \in N} v_i$          // Ties are broken randomly

29:   return ($A, P$)
```

For each sample, we compute the shortest path between the single waypoint and the agent, if it exists. Because unknown cells in the environment layer are sampled randomly, it is possible for the waypoint to be unreachable by the agent, in which case the sample is ignored. If there are multiple shortest paths (as is often the case in grid-based environments) one path is chosen randomly and all grid cells belonging to this path are incremented in the pheromone layer. In this way, the pheromone layer serves to aggregate the best paths that lead to potential waypoint locations.

The second phase of the MMC policy performs value iteration using the unknown areas of the pheromone layer to determine the best immediate course of action. First, a new value layer V is created as a copy of the pheromone layer, but with all observed areas (except unvisited waypoints, see below) set to zero. This is done to signify that there is no inherent value in visiting a grid cell that has already been observed. It also helps prevent oscillatory behavior stemming from the high density of pheromone in the immediate vicinity of the agent that can interfere with moving toward more distant goals (See Fig. 4b and 4c for examples). Waypoints that have been observed but not visited are given an extra initial value, equal to some multiple η of the current maximum value in the pheromone layer. This can be adjusted to produce greedy behavior when η is large or wandering behavior when η is small. The creation of the value layer is outlined in lines 11-13 of the pseudocode.

The values in the value layer are then propagated using a version of the value iteration algorithm commonly used in Markov decision processes [18]. Until there is no apparent change in the value layer, each cell is iteratively assigned the maximum value of either itself, or its 4-connected neighbors multiplied by a discount factor, $\lambda \in [0, 1)$. The discount factor effectively determines how far the value propagates through the environment. A large discount factor can cause distant goals to influence the next action, whereas a smaller discount factor leads to more local behavior. Once the algorithm has converged ($\varepsilon < 0.001$ for our experiments), the neighboring cell of the agent with the greatest value is chosen as the agent's next location.

Fig. 4 gives an example that shows how the pheromone aggregation map and value layer evolve over the course of implementing the MMC policy. In comparison with the greedy policy used on the same environment in Fig. 3, the MMC policy is able to achieve a much shorter route by exploiting local gradients to finish exploring an area before moving on to more distant goals.

IV. RESULTS

We designed a series of experiments to empirically evaluate the greedy and MMC policies for the PO-TSP. First, we generated 10 environments using Algorithm 1, shown in Fig. 5. Each is a 50×50 grid in which the agent and 10 waypoints are distributed randomly using a minimum separation of 10 steps. Then, for each environment, we performed 100 trials each of the greedy policy and 18 different parameter configurations of the MMC policy. For the MMC policy, we vary the evaporation rate as $\gamma = \{0.9, 0.95, 0.99\}$, the discount factor as $\lambda = \{0.9, 0.95, 0.99\}$, and the waypoint weight as $\eta = \{1, 10\}$. These values were chosen based on preliminary experiments and

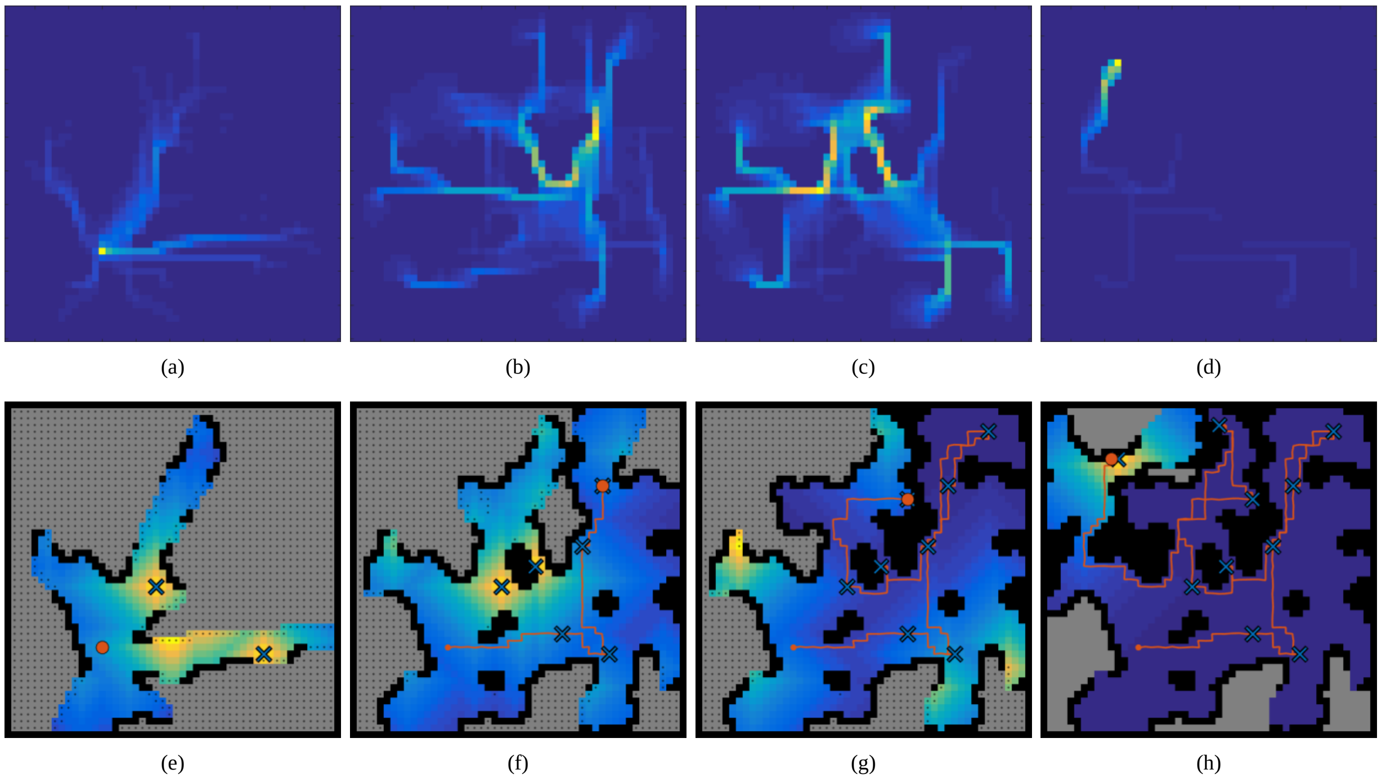

Fig. 4. Some selected moments from the MMC policy's solution to the same PO-TSP environment used in Fig. 3. The top row (a-d) shows the pheromone map, which aggregates the shortest paths from the agent to sampled waypoints. The bottom row (e-h) shows the value map, which defines a gradient that the agent follows. Symbols are enlarged for clarity. The initial observation (e) is identical to Fig. 3a, but instead of picking a target location, 1000 waypoint locations are sampled and the shortest paths back to the agent are aggregated in a persistent pheromone map (a). Performing value iteration on the unobserved regions of (a) gives the gradient map (e) that the agent follows. (b) and (f) show the maps after reaching four waypoints as the MMC agent recognizes the possibility of a waypoint in the top-right and discovers the waypoint that was missed by the greedy agent. (c) and (g) show the maps after visiting eight waypoints when the agent could proceed to the high value area in the left, but instead follows the local gradient toward the top and discovers the ninth waypoint. The final maps are shown in (d) and (h) where most of the pheromone has evaporated except for a single trail and only a single peak is left in the gradient map.

provide a reasonable range of results for the specified problem size. Traditionally, the evaporation rate and discount factors are set to be only slightly less than 1 to prevent the pheromone from evaporating too quickly and to ensure that the value does not dissipate too quickly as it propagates through the environment. We used 1000 samples for each iteration of the MMC methods. The results of these experiments are summarized in Fig. 6.

Each of the 10 environments is plotted separately in Fig. 6 since the optimal tour length is highly dependent on the specific problem instance. The upper and lower plots for each environment show the difference between the two waypoint weight values. Each plot shows the distribution of tour lengths for the 100 trials of each method or parameter setting. The grey filled plots represent the results of the greedy policy and the colored plots show the results of the various parameter settings of the MMC policy. The leftmost point of each horizontal axis represents the length of the "oracle" tour, that is, the shortest tour possible that visits all waypoints. The oracle has perfect knowledge of the environment a priori, and can therefore use standard offline TSP solvers. Note that the oracle tour is not the same as the decision-theoretic optimal tour. While it is possible (in rare circumstances) that an agent could just so happen to follow the oracle tour, in general, such an outcome is unlikely.

It is apparent from observing the plots that there is no universal best method that works well in all environments. Furthermore, a particular method may produce a wide range of tour lengths. The random sampling process of the MMC policy explains the wide distribution of results for these methods, and the random tie-breaking procedure explains why some distributions are multimodal. For example, if there are two equally short paths to a target, but one provides a vantage point that gives the location of another waypoint, it may ultimately lead to a shorter tour than the other path.

The maximum tour length shown on the horizontal axis occasionally cuts off some of the distributions to more clearly show the rest of the results. This occurs in environments 6 and 10 when the discount factor is set to the highest value of 0.99. With a high discount factor, the value iteration step is able to propagate high values farther through the environment, which can cause distant goals to become more attractive. This causes the agent to adopt a more global search strategy, which may force the agent to travel across the map repeatedly. A smaller discount factor results in more local search, but can lead to numerical underflow errors if the value iteration step terminates before influencing the area around the agent. For example, when the last corner of a cave branch is explored, the nearest unexplored area may be too far away for its value to reach the

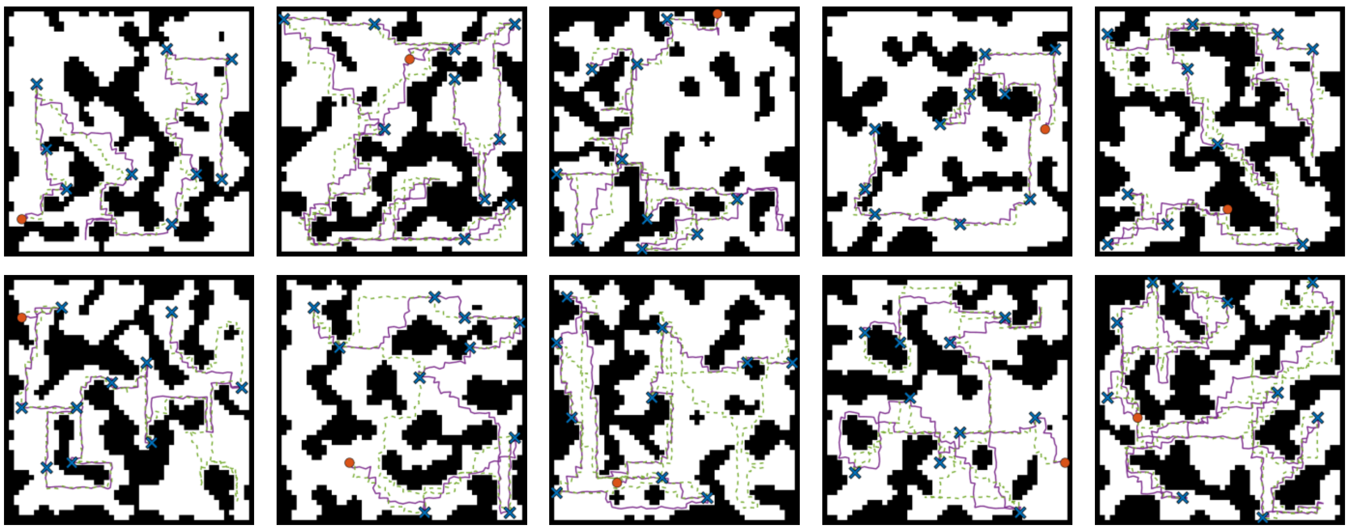

Fig. 5. The ten randomly generated environments used in the experiments. Each is a 50×50 grid with 10 waypoints (blue crosses) randomly placed with a minimum spacing of 10 steps. The initial agent location is shown as a red circle and example solutions from the greedy algorithm and the MMC algorithm are shown. The path of the greedy algorithm is shown as a green dotted line and the path of the MMC algorithm is shown as a solid purple line. For the MMC algorithm, the discount factor was set to λ=0.9, the evaporation rate was set to γ=0.99, and the waypoint weight was set to η=10. Symbols are enlarged for clarity.

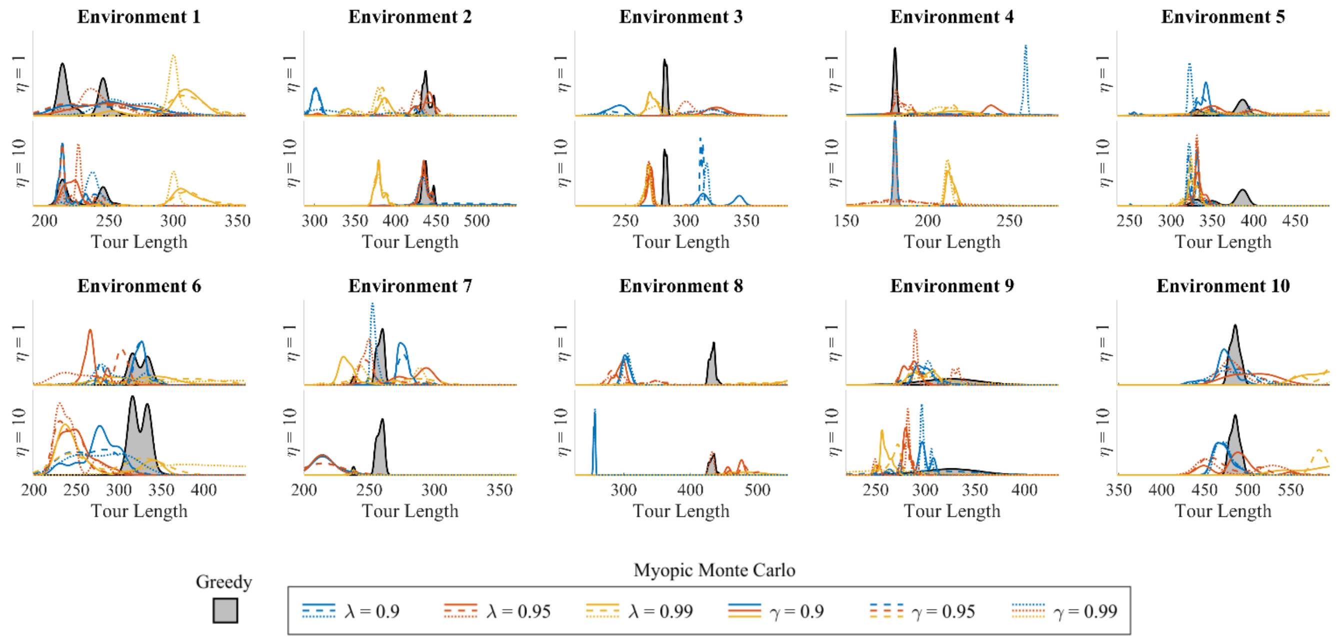

Fig. 6. Distributions of tour lengths for various methods on each of the 10 environments considered. Each plotted line is a probability density estimate of the tour length for a given method, computed using a kernel density function of 100 trials for each method. Each environment shows two plots, separated by waypoint weight η. The horizontal axis gives the number of steps in the tour, with the lower limit set to the length of the oracle tour (the best tour possible if the environment were known a priori). The grey filled plots show the distribution of the greedy policy tours. The myopic Monte Carlo policy tours are shown in various colors and line styles. Color indicates the discount factor λ and line style indicates the evaporation rate γ.

agent, resulting in the agent performing a random walk until it can "pick up the scent."

Table I draws some conclusions about our experiments by averaging the difference between the mean distribution values of the MMC policies and greedy policy in each environment. The values indicate the average number of additional steps required by the MMC policy as compared to the greedy policy. Values less than zero show that, averaged over all environments, the MMC policy performed better than the greedy policy, whereas values greater than zero show that the greedy policy was better. The only obvious trend is that smaller discount factors appear to perform better, which echoes our discussion above. It is difficult to draw any more general conclusions from these results, as the best method appears to be highly dependent on the problem environment. Although we do not explicitly compare computation time between the algorithms, it should be noted that the MMC algorithm is more computationally

TABLE I. AVERAGE IMPROVEMENT OF MMC OVER GREEDY POLICY[a]

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

[a.] Defined as the mean MMC tour length minus the mean greedy tour length, averaged over all 10 environments.

intensive than the greedy approach, but can be designed to scale with the number of samples used.

Fig. 5 shows an example solution from both the greedy and MMC algorithms on each of the tested environments. The parameters for the MMC algorithm shown were chosen based on the best results from Table I. The paths from the two algorithms are often nearly identical, however there are a few instances where one algorithm explores an area that the other did not. These differences stem from the different strategies employed by the two algorithms, along with some degree of random chance. Multiple runs of the same algorithm can yield different paths. Our results indicate that by averaging over many trials in all environments, the MMC policy with an appropriate parameter configuration is capable of outperforming the greedy policy.

V. CONCLUSION AND FUTURE WORK

The PO-TSP is an interesting benchmark problem to study and leaves many opportunities for future enhancements. Our greedy and MMC algorithms can solve most problem instances intelligently but do not behave optimally under all circumstances. It would be helpful to identify the characteristics of the problems that impact the performance and group similar problems together for a more consistent analysis. For example, one algorithm might work better on problems that have many waypoints hidden at the ends of passages, while another algorithm might work better when the waypoints are distributed in large open spaces. Our results show that the PO-TSP is highly problem dependent, making it difficult to develop an algorithm that performs well in every situation. This makes it a challenging and perhaps unfair problem to use in a competition setting but approachable in terms of optimal decision-theory. Future enhancements may consider the use of MCTS techniques and partially observable Markov decision process solvers to improve general agent strategies.

The mental map representation used in our work is applicable to small problem domains, but it may not scale well to large problems. However, the ideas established by the MMC algorithm in particular could prove useful in developing robust policies that operate on more sophisticated mental map representations. For example, a complete set of waypoints could be sampled each iteration and used in conjunction with ACO to plan future actions that look ahead past the next immediate waypoint.

Lastly, it would be interesting to compare the algorithmic policies developed here to the solutions of actual human beings. This could give insight into models of human decision-making under uncertainty. By solving a set of benchmark problems, the PO-TSP could serve as a useful tool for studying anticipatory analysis.

REFERENCES

[1] S. Thrun, "Robotic mapping: A survey," in Exploring Artificial Intelligence in the new Millenium, G. Lakemeyer and B. Nebel, Eds., Morgan Kaufmann, 2002, pp. 1-36.

[2] M. Morin, "Multi-criteria path planning with terrain visibility constraints: The optimal searcher path problem with visibility," M.S. thesis, Laval Unv., Québec, 2010.

[3] C. L. Baker and J. B. Tenenbaum, "Modeling human plan recognition using bayesian theory of mind," in Plan, Activity, and Intent Recognition: Theory and Practice, G. Sukthankar, R. P. Goldman, C. Geib, D. Pynadath, and H. Bui, Eds., Morgan Kaufmann, 2014, pp. 177–204.

[4] A. R. Buck, J. M. Keller, and M. Skubic, "A memetic algorithm for matching spatial configurations with the histograms of forces," IEEE Trans. Evol. Comput., vol. 17, no. 4, pp. 588-604, 2013.

[5] A. R. Buck and J. M. Keller, "A graph-based memetic approach to sketch geolocation," in IEEE Workshop on Memetic Computing, Singapore, 2013, pp. 44-51.

[6] J. M. Keller, M. Popescu, and D. Gibeson, "An extension of a confined space evacuation model to human geography," in 2012 IEEE Int. Geosci. and Remote Sensing Symp., 2012, pp. 531–534.

[7] M. Popescu and J. M. Keller, "Implementing bounded rationality in disaster agent behavior using OGA operators," in 2012 IEEE Int. Geosci. and Remote Sensing Symp., 2012, pp. 5379–5381.

[8] P. Jaillet, "Probabilistic traveling salesman problems," Ph.D. dissertation, MIT, 1985.

[9] D. Perez, P. Rohlfshagen, and S. M. Lucas, "Monte-Carlo tree search for the physical travelling salesman problem," in Applications of Evolutionary Computation, Cecilia Di Chio et al. Eds., Springer, 2012, pp. 255–264.

[10] C. B. Browne et. al., "A survey of Monte Carlo tree search methods," IEEE Trans. Comput. Intell. AI Games, vol. 4, no. 1, pp. 1–43, 2012.

[11] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set Monte Carlo tree search," IEEE Trans. Comput. Intell. AI Games, vol. 4, no. 2, pp. 120–143, 2012.

[12] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in Advances in Neural Information Processing Systems 23 (NIPS 2010), 2010, pp. 2164–2172.

[13] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," IEEE Trans. Syst. Man. Cybern., vol. 26, no. 1, pp. 29–41, 1996.

[14] J. E. Bresenham, "Algorithm for computer control of a digital plotter," IBM Syst. J., vol. 4, no. 1, pp. 25–30, 1965.

[15] F. Durand, "A multidisciplinary survey of visibility," ACM SIGGRAPH course notes: Visibility, Problems, Techniques, and Applications, 2000.

[16] M. Gardner, "Mathematical games - The fantastic combinations of John Conway's new solitare game 'life,'" Scientific American, no. 223, pp. 120–123, 1970.

[17] N. H. Packard and S. Wolfram, "Two-dimensional cellular automata," Journal of Statistical Physics, vol. 38, pp. 901–948, 1985.

[18] R. Bellman, "A Markovian decision process," J. Math. Mech., vol. 6, pp. 679–684, 1957.