Improving the Efficiency and Scalability of Multi-Drone Coverage Systems with Decentralized Control

Adit Shah,[1†] Bryce L. Ferguson,[2] Jason R. Marden[2]
[1]*The Athenian School, 2100 Mt. Diablo Scenic Boulevard, Danville, CA 94506*
[2]*Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106*
[†]*Email address: aditshah00@gmail.com*

Multi-agent systems are effective for numerous applications because they complete tasks more resiliently and efficiently than monolithic systems. This paper focuses on a sensor-coverage problem in which a limited fleet of sensor-equipped drones must survey an area and extract maximum information. Applications include environmental monitoring, disaster relief, and surveillance systems. Traditionally, these systems are implemented through a centralized approach, which can run into obstacles, including communication and computational constraints. These limitations can be mitigated through decentralized control algorithms, where the decision-making and navigation processes are localized to individual robots. In this paper, we look to quantitatively compare algorithms for such decentralized systems through simulation. We establish a baseline with a greedy algorithm, and we then compare its performance to that of a log-linear learning approach, which adds stochasticity. Finally, we propose a method to automatically generate a constant for this algorithm, and verify its performance.

Keywords: distributed algorithms, multi-robot systems, multi-agent systems, greedy algorithms, log-linear learning

1. INTRODUCTION

In the last decade, multi-agent systems have gained popularity for applications which range from coordinating autonomous cars to ease traffic and improve road safety, to synchronizing manufacturing robots to automate an assembly line efficiently. Drones (also known as unmanned aerial vehicles, or UAVs) have been a major technology focus in this area of research, as they can be applied to delivery, emergency response, mapping, inspections, and much more [1]. A subset of these problems involves a fleet of drones which need to spread out across a large area [2]. For instance, a multi-drone surveillance system could be used as either a substitute or addition to ground personnel for the security of sensitive government and military sites. Other examples of applications include drone networks that map the course of disasters such as oil spills and wildfires [1].

In this paper, we focus on a type of sensor coverage problem where a network of drones need to survey a large area. When the drones cannot cover the entire region, their positions must be set to extract as much information as possible. This involves minimizing overlapping coverage and prioritizing areas of most importance. To do so, each region in the area to be covered would be assigned different weights. For instance, a surveillance system may prioritize the boundaries of a compound, which is where intruder would likely enter from. A system has a given number of agents, each of which have a sensing radius. An agent is covering a given location if it is within the range of its sensor. The ultimate goal is to maximize the number of regions covered, with priority given to those with higher weight.

There are two main types of control approaches for multi-agent systems: centralized and decentralized. Centralized control, which is the more easily regulated approach, is where a central node manages the location and navigation of every drone. However, the coverage problem is known to be NP-complete [3, 6], which means computing the optimal locations for each agent is very slow. A number of approximations have been developed with varying results to improve runtimes, both for drone swarms [4] and sensor coverage applications in general [5]. However, centralized systems have limits in efficiency and scalability, because one node needs to maintain contact with every drone, and is responsible for each of its movements.

For this reason, decentralized control approaches are a viable alternative for coverage applications. Instead of a central controlling node, each drone has its own decision-making capabilities to determine its optimal location and how to navigate there. Recent research has focused on the efficacy of this decentralized approach [6-8]. One method of doing so prioritizes drones navigating to specific "destination points" and navigating around obstacles to do so [6]. Two other methods, a single robot coverage algorithm applied on regions created through cellular decomposition and a greedy approach applied to an area divided into equal regions, have also been shown to be effective through simulation [7]. In [8], this greedy algorithm is taken further by applying a game-theoretic approach to multi-robot systems. It utilizes log-linear learning to add stochasticity (randomness) into the system, where there is a certain likelihood that each agent will execute a suboptimal action. This research also discusses the effects of limitations on the information provided to each agent. Finally, [9] develops navigation algorithms for decentralized drone systems with limited information and sensing capabilities.

Our main contribution in this paper is demonstrating the effectiveness of log-linear learning for multi-agent coverage systems. We redefine the problem setting to combine the navigation and optimal location settings into one problem by limiting the movements an agent can make at each time-step. Additionally, unlike previous research, we develop a simulation of a multi-agent system, and compare the greedy algorithm quantitatively through this simulation. Based on the results of these experiments, we present a method to automatically select a temperature constant for the log-linear learning algorithm.

In Section II, we formulate the problem mathematically, discuss metrics for evaluating algorithms in this setting, and introduce the greedy and log-linear learning algorithms.

II. MODEL AND ALGORITHMS

A. Problem Formulation

In this paper, we focus on improving decentralized control approaches for a weighted coverage problem where there is a set of decision-making agents $N = \{1,2,3,...,n\}$, each of which have a sensing radius R and a movement radius r. We consider a discretized map M of a region that needs to be covered. This map is an p by q matrix of cells. The value of any cell $M_{x,y}$ represents the utility, or importance, of covering cell $(x, y) \in X \times Y$ where $X = \{1,\dots, p\}$ and $Y = \{1,\dots, q\}$.

Let $a_i(t) = (x_i, y_i)$ represent the position of agent $i \in N$ at time-step t, and $a_{-i}(t)$ represent the positions of every agent except for i. Each agent is assigned a starting position $a_i(0) = (x_i, y_i)$, which is either chosen randomly or pre-determined. At each time-step t, an agent is chosen uniformly at random and allowed to update its position. If $d(a_1, a_2)$ represents the Euclidean distance between two points, each agent i has an action set $A_i(t)$ with agents $a_i \in A_i(t)$ such that:

$$A_i(t) = \{(x, y) \in X \times Y \mid d((x, y), a_i(t-1)) \le r, (x, y) \notin a_{-i}(t-1)\} \quad (1)$$

i.e. an agent can move to all cells within their movement radius if another agent is not occupying that position. This differs from previous literature [8, 9], but simplifies the problem because it combines the navigation and optimization into a single problem: determining to which cell on the grid an agent should move at each time-step.

For each action $a_i \in A_i(t)$, the coverage set $C(a_i)$ consists of each cell $\{(x, y) \in X \times Y \mid d(a_i, (x, y)) \le R\}$. The utility function U_i of taking a particular action is defined as follows:

$$U_i(a_i, a_{-i}) = \sum_{(x,y) \in C(a_i) \setminus \bigcup_{j \neq i} C(a_j)} M_{x,y} \quad (2)$$

In words, the utility of any possible action is the sum of the utilities of every cell that is within the agent's sensing radius and is not already covered by another agent. The ultimate goal is to maximize the utility of the total system, which is the sum of the values of every cell in M covered by an agent:

$$\mathcal{U}_t(a) = \sum_{(x,y) \in C(a(t))} M_{x,y} \quad (3)$$

Because the utility values are arbitrary and vary for each map, the algorithms will be compared with a normalized system utility $\tilde{\mathcal{U}}_t(a)$, which scales the utilities of the map such that total coverage yields $\tilde{\mathcal{U}}_t(a) = 1$:

$$\tilde{\mathcal{U}}_t(a) = \frac{\mathcal{U}_t(a)}{\sum_{x,y \in X \times Y} M_{x,y}} \quad (4)$$

This value $\tilde{\mathcal{U}}(t)$ can be thought of as a proportion of the total utility that is covered, e.g. $\tilde{\mathcal{U}}(t)$=0.4653 would mean that the system achieved 46.53% of the maximum system utility (which would occur if every cell were covered).

The overall objective is to maximize the total value of the covered cells, utility $\mathcal{U}_t(a)$, while minimizing the time it takes for this value to converge. We now look at decentralized algorithms for this objective.

B. The Greedy Algorithm

Greedy algorithms have been widely studied in the context of such set-cover problems for multi-agent systems [7,10]. In this algorithm, at each time-step t, each agent i choses its action $a_i(t+1) \in A_i(t)$ as follows:

$$a_i(t+1) = \arg \max_{a_i \in A_i(t)} U_i(a_i, a_{-i}) \quad (5)$$

Essentially, the greedy algorithm will always choose to make a move that maximizes its utility, and terminates when $a_i(t) = a_i(t+1)$. Because of these characteristics, the greedy algorithm will get stuck in local optima, because it only takes an action that immediately benefits the utility function. For this reason, we only utilize the greedy algorithm as a baseline with which to compare other algorithms.

C. Log-Linear Learning

The log-linear learning algorithm takes a game theoretic approach to maximizing the utility of a multi-agent system, as described in [8]. Rather than always picking the immediately optimal move, it choses an action $a_i(t+1) \in A_i(t)$ through a probability distribution that is based on the utility of each possible action. The probability of agent i taking each action $a_i \in A_i(t)$ with temperature $\tau > 0$ is defined as follows:

$$p_i^{a_i}(t) = \frac{e^{\frac{1}{\tau} U_i(a_i, a_{-i}(t-1))}}{\sum_{\overline{a}_i \in A_i} e^{\frac{1}{\tau} U_i(\overline{a}_i, a_{-i}(t-1))}} \quad (6)$$

The temperature τ determines the likelihood that an agent will make a suboptimal action, i.e. one that does not maximize the utility function. As $\tau \to 0$, the action with the highest utility will always be selected, so the algorithm will act as a greedy algorithm. As $\tau \to \infty$, the actions will be chosen uniformly at random. Intuitively, τ can be thought of as an "exploration" constant, where higher values of τ encourage agents to search for global optima that may be beyond the local optima near its initial position. In our definition of the log-linear learning algorithm, τ decays at a constant rate, in this case 0.997, such that the algorithm will eventually approach the behavior of a greedy algorithm, and U_i will stabilize.

Our definition of the log-linear learning algorithm differs from [8] because in our approach, agents make their moves one at a time rather than simultaneously, the action set $A_i(t)$ is limited to cells within a movement radius r away from $a_i(t-1)$, and τ decays rather than staying constant.

In Section III, we propose a method for improving the results of log-linear learning, and compare the performance of the aforementioned algorithms.

III. RESULTS AND DISCUSSION

A. Automatic Tau Generation

The log-linear learning algorithm as established in [8] specified that there is a temperature value τ as explained above. However, it does not discuss how to choose the initial value, which means that manual tuning is necessary to determine the ideal starting value. Through testing of the impact of various variables on the ideal τ value, the factors that matter the most were found to be the sensing radius of each agent, the average utility value of the map, and the starting positions of the agents. Thus, the formula for generating the τ value if the starting positions of each agent, $A_i(0)$, are decided randomly, is as follows, where $\overline{M}$ is the average value of M:

$$\tau = \frac{1}{2}\pi R^2 \cdot \overline{M} \tag{7}$$

Essentially, the automatic tau generation is proportional to the average utility for a particular agent if placed on the map at random, which is the area an agent can cover times the average value of each cell.

However, when $A_i(0)$ is not randomly determined, but the agents instead begin in consecutive cells closest to a corner of the map, the τ value is scaled:

$$\tau = \pi R^2 \cdot \overline{M} \tag{8}$$

This accounts for the fact that if the agents all begin in one corner, they have a higher probability of encountering additional local optima. Because they are not spread out, they will need more time-steps to discover the global optimum. A limitation of this approach is that it requires access to the average value of the map, although this only needs to be computed once before the agents are deployed.

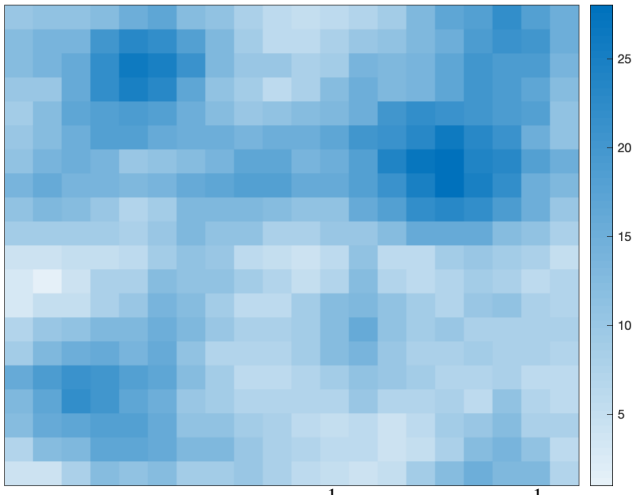

FIG. 1. A visualization of map M^1, a 20x20 matrix. $M^1_{1,1}$ is at the top left corner. The color of each square represents the utility of that cell $M_{x,y}$, as illustrated in the scale shown on the right.

B. Algorithm Comparison

In order to quantitatively compare the greedy, log-linear learning (LLL), and LLL with automatic τ generation, three maps M^k were generated such that they would contain multiple local optima in addition to a global optimum. Figure 1 shows a visualization of one of these maps, M^1, where each cell is a different color based on its utility value. Each algorithm was run on the three maps with the greedy algorithm (equivalent to $\tau = 0$), the automatically generated τ value, and two manually τ values: one above and one below the generated value. All of these were simulated for two staring configurations: one where the locations of the agents are chosen randomly at every trial, and one where all of the agents begin either at position (1,1) or adjacent cells closest to this corner. The model of starting every agent at adjacent locations more accurately represents how a fleet of drones would be deployed in practical applications.

For concision, the trends in the algorithms' performances will mainly be discussed with examples from M^1 (Table 1). The final resting positions of each drone in M^1 are visualized in Appendix A, and data from other maps is in Appendix B, which display similar trends to M^1.

The greedy algorithm consistently has the worst performance regardless of starting configuration, because of its inability to overcome local optima. For instance, in Table 1a, $\tilde{\mathcal{U}}_t(a)$ converges to 0.4257, which is lower than any of the results for log-linear learning. However, the gap between the greedy algorithm and log-linear learning is much more pronounced when every agent begins in one corner of the map. This occurs because the agents are much more likely to encounter local optima that by definition they are unable to cross, because the greedy algorithm only makes moves that provide immediate benefit; this tendency

Table 1a: Random starting positions

		t	$\mathcal{U}_t(a)$	$\tilde{\mathcal{U}}_t(a)$
Greedy	$\tau = 0$	54.7	2082.3	0.4257
LLL	$\tau = 40$	570.0	2110.3	0.4315
	$\tau = 96$ (auto)	675.3	2177.0	0.4451
	$\tau = 130$	700.0	2131.7	0.4358

Table 1b: Starting position in one corner

		t	$\mathcal{U}_t(a)$	$\tilde{\mathcal{U}}_t(a)$
Greedy	$\tau = 0$	110.7	1907.7	0.3900
LLL	$\tau = 153$	771.0	2174.7	0.4446
	$\tau = 192$ (auto)	753.3	2215.0	0.4529
	$\tau = 250$	1125.3	2141.0	0.4377

Table 1. Performance metrics for the greedy and log-linear learning algorithms (with varying τ values) on M^1, including number of time-steps to stabilization, the final utility value, and the normalized final utility value. (These simulations were run with $n = 6$ agents, sensing radius $R = \sqrt{5}$ and moving radius $r = 1$. Three trials per setting were conducted and the average values of t, $\mathcal{U}_t(a)$, and $\tilde{\mathcal{U}}_t(a)$ are displayed. For (a), a random starting position was used, and for (b) the agents began in one corner.

is visualized in Appendix A. For this reason, the greedy approach only reaches $\tilde{\mathcal{U}}_t(a) = 0.3900$, which is much lower than the result for the greedy algorithm when placed at random. However, the greedy algorithm has one consistent advantage: the number of time-steps t that it takes for $\mathcal{U}_t(a)$ to stabilize, which is defined as the point after which $\mathcal{U}_t(a)$ stays within 10% of the final utility. On average, it took 54.7 and 110.7 time-steps to stabilize for the random and fixed starting positions respectively, whereas the log-linear algorithm takes 5 to 15 times as long to do so.

Although the log-linear algorithm algorithm takes much longer to stabilize than the greedy algorithm, it can reach significantly higher utility values. In Table 1a, the highest log-linear learning result performs 4.5% better than the greedy algorithm, and in Table 1b, this improvement grows to 16%. However, there is significant variance in results based on the τ value. In both starting configurations across all maps (Table 1, Appendix A) the automatically-generated value reaches a higher utility value than the lowest value, with a comparable or higher number of time-steps. This is an expected result, because a system with a higher τ value has a longer "exploration phase" where agents are very likely to make suboptimal moves. This makes such a system more likely to reach a global optimum while also increasing the time to stabilization. Similarly, systems with τ values higher than the automatically-generated value take more time to stabilize because of their longer exploration phase.

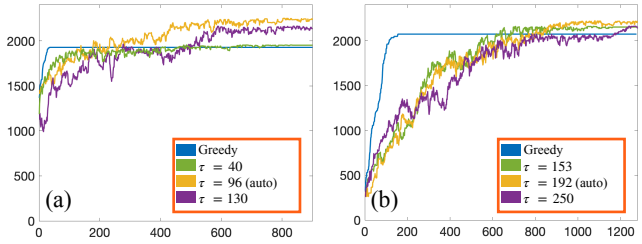

FIG. 2. Sample graphs of $\mathcal{U}_t(a)$ (vertical axis) over t (horizontal axis) for M^1 for each algorithm. (a) shows the random starting position, (b) shows the corner starting position.

However, the final $\tilde{\mathcal{U}}_t(a)$ does not improve with τ values higher than the automatically-generated value. This trend is not only seen in Tables 1 and 2, but throughout the data collected (Appendix B). This seems to contradict an intuitive understanding of log-linear learning which suggests that higher τ will help a system discover the global optimum, albeit with more time-steps. This behavior could be occurring because at a τ value above a certain threshold, the moves of the agents are almost completely at random, which means that on average, agents will remain at their starting locations. Additionally, even if agents do find optimal positions, they are unlikely to remain there because the algorithm is not considering the utilities at all at high temperature values. Both of these patterns were empirically observed; however further study is necessary to definitively determine the cause of this unexpected outcome.

IV. CONCLUSION AND FUTURE WORK

In this paper, we formulate a modified version of a decentralized multi-agent coverage problem, where the agents do not have complete knowledge of the map, and they are restricted to movements in their immediate vicinity; this system model more closely mirrors the real-life applications of such multi-drone coverage systems. Multiple control approaches for this problem settings are discussed, including a greedy approach and a log-linear approach. By simulating this multi-robot system, we demonstrate that the log-linear learning approach has superior performance than the greedy algorithm. We automate the generation of a temperature constant for this algorithm, which achieves optimal performance and eliminates the need for manual parameter tuning, allowing the algorithm to be deployed in different environments without needing to validate it for each one.

In the future, the algorithms could be tested on more maps with different characteristics, additional starting configurations, and the impacts of sensing radius, movement radius, number of agents, distance from other agents, and various other variables could be evaluated. Obstacles that the agents must navigate around could be added to the maps. The formula for generating the τ values could take additional variables into account, including those listed above. Finally, these algorithms could be verified on physical multi-robot systems.

V. ACKNOWLEDGEMENTS

The first author would like to thank lab peers Siddharth Ganesan and Nischal Sinha for their support throughout the program. We would like to acknowledge Center for Computation and Dynamical Control for access to their resources and support during this research. Finally, we thank Dr. Lina Kim, Dr. Michael Hughes, and A S M Iftekhar from the Research Mentorship Program, and the University of California, Santa Barbara, for the research opportunity.

REFERENCES

[1] G. Chmaj and H. Selvaraj, "Distributed Processing Applications for UAV/drones: A Survey," *Progress in Systems Engineering. Advances in Intelligent Systems and Computing*, vol. 366, pp. 449–454, 2015.

[2] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, Dec. 2013.

[3] S. Khuller, A. Moss, and J. (S. Naor, "The budgeted maximum coverage problem," *Information Processing Letters*, vol. 70, no. 1, pp. 39–45, Apr. 1999.

[4] K. Loayza, P. Lucas, and E. Pelaez, "A centralized control of movements using a collision avoidance algorithm for a swarm of autonomous agents," 2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM), 2017.

[5] W. Wu, Z. Zhang, W. Lee, and D.-Z. Du, "Optimal Sensor Coverage," *University of Texas, Dallas,* https://www.utdallas.edu/~dzdu/cs7301c/main.pdf.

[6] C. Zhai and Y. Hong, "Distributed coverage control of multi-agent systems using navigation functions," *2013 10th IEEE International Conference on Control and Automation (ICCA)*, Jun. 2013.

[7] N. Karapetyan, K. Benson, C. Mckinney, P. Taslakian, and I. Rekleitis, "Efficient multi-robot coverage of a known environment," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Dec. 2017.

[8] J. R. Marden and J. S. Shamma, "Revisiting log-linear learning: Asynchrony, completeness and payoff-based implementation," *Games and Economic Behavior*, vol. 75, no. 2, pp. 788–808, Jul. 2012.

[9] R. Maeda, T. Endo, and F. Matsuno, "Decentralized Navigation for Heterogeneous Swarm Robots With Limited Field of View," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 904–911, Jan. 2017.

[10] D. Grimsman, M. S. Ali, J. P. Hespanha, and J. R. Marden, "The Impact of Information in Greedy Submodular Maximization," *IEEE Transactions on Control of Network Systems*, Dec. 2018.

APPENDIX

A. Drone Position Visualizations on M^1

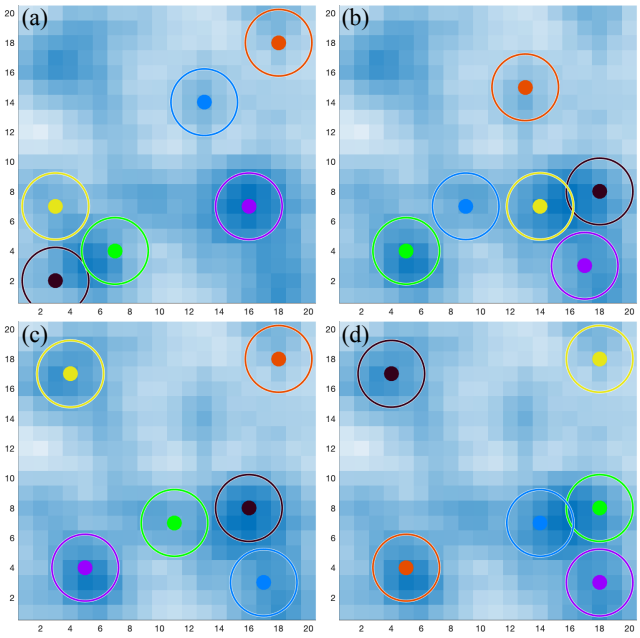

FIG. A1. Visualization of drone resting positions for each algorithm in the random starting configuration. Darker regions represent higher utility, agents are represented by colored dots, and sensing radii are represented by a circle around each agent. (a) shows the result of the greedy algorithm, (b) shows $\tau = 40$, (c) shows $\tau = 96$ (automatically generated) and (d) shows $\tau = 130$.

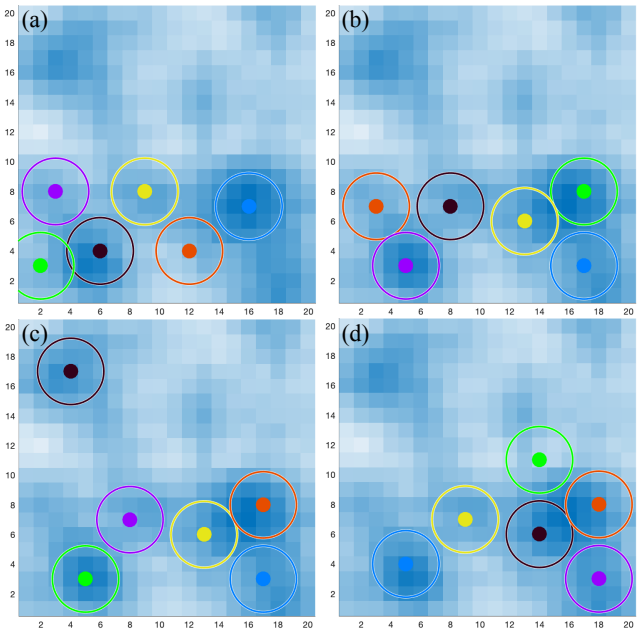

FIG. A2. Visualization of drone resting positions for each algorithm in the random starting configuration. Darker regions represent higher utility, agents are represented by colored dots, and sensing radii are represented by a circle around each agent. (a) shows the result of the greedy algorithm, (b) shows $\tau = 153$, (c) shows $\tau = 192$ (automatically generated) and (d) shows $\tau = 250$.

B. Additional Maps: Visualization and Data

1. Map M^2

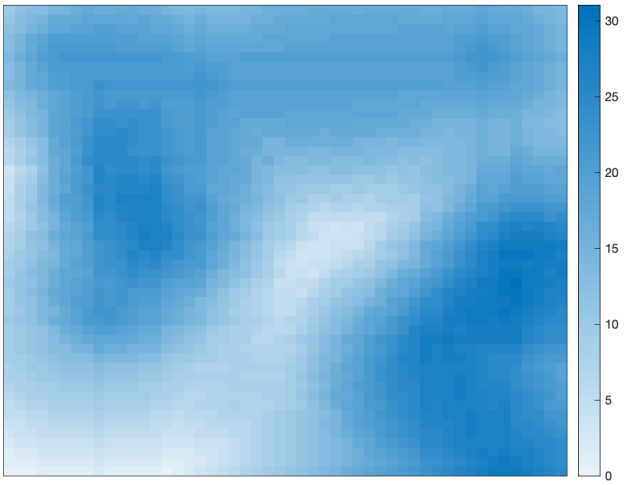

FIG. B1. A visualization of map M^2, a 50x50 matrix.

Table B1a: Random starting positions

		t	$\mathscr{U}(t)$	$\tilde{\mathscr{U}}(t)$
Greedy	$\tau = 0$	40.7	15195.7	0.3680
LLL	$\tau = 350$	673.7	15404.0	0.3730
	$\tau = 933$ (auto)	672.3	15658.7	0.3792
	$\tau = 1700$	971.0	15523.0	0.3759

Table B1b: Starting position in one corner

		t	$\mathscr{U}(t)$	$\tilde{\mathscr{U}}(t)$
Greedy	$\tau = 0$	70.7	13325.3	0.3227
LLL	$\tau = 700$	779.0	13931.0	0.3373
	$\tau = 1867$ (auto)	953.7	15340.0	0.3715
	$\tau = 3000$	1130.0	15286.0	0.3702

Table B1. Performance metrics for the greedy and log-linear learning algorithms (with varying τ values) on M^2, including number of time-steps to stabilization, the final utility value, and the normalized final utility value. These simulations were run with $n = 6$ agents, sensing radius $R = 6$ and moving radius $r = 5$. Three trials for each setting were conducted, and the average values of t, $\mathscr{U}_t(a)$, and $\tilde{\mathscr{U}}_t(a)$ are displayed. For (a), a random starting position was used, and for (b) the agents began in one corner.

2. Map M^3

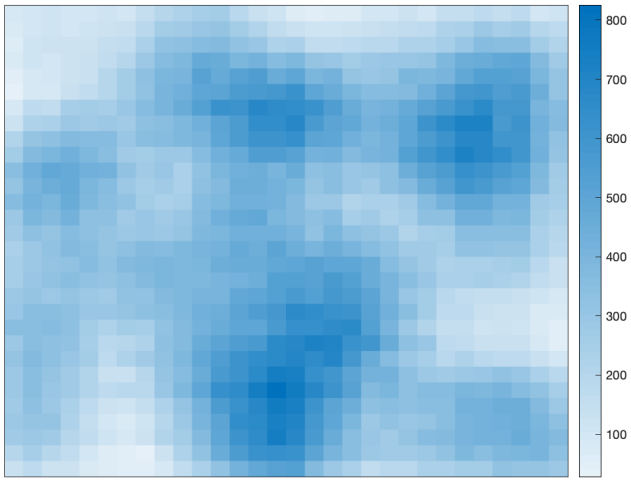

FIG. B2. A visualization of map M^3, a 30x30 matrix.

Table B2a: Random starting positions

		t	$\mathscr{U}_t(a)$	$\tilde{\mathscr{U}}_t(a)$
Greedy	$\tau = 0$	23.0	104335.7	0.3302
LLL	$\tau = 3500$	624.3	107473.0	0.3401
	$\tau = 8819$ (auto)	672.3	110314.0	0.3491
	$\tau = 15000$	971.0	110151.3	0.3486

Table B2b: Starting position in one corner

		t	$\mathscr{U}_t(a)$	$\tilde{\mathscr{U}}_t(a)$
Greedy	$\tau = 0$	61.7	101197.7	0.3203
LLL	$\tau = 4500$	564.7	110268.3	0.3490
	$\tau = 17368$ (auto)	700.0	110334.0	0.3492
	$\tau = 25000$	1034.7	106702.3	0.3486

Table B2. Performance metrics for the greedy and log-linear learning algorithms (with varying τ values) on M^3, including number of time-steps to stabilization, the final utility value, and the normalized final utility value. These simulations were run with $n = 4$ agents, sensing radius $R = 4$ and moving radius $r = 3$. Three trials for each setting were conducted, and the average values of t, $U_t(a)$, and $\tilde{\mathscr{U}}_t(a)$ are displayed. For (a), a random starting position was used, and for (b) the agents began in one corner.