Autonomous Coverage Path Planning using Artificial Neural Tissue for Aerospace Applications

Byong Kwon
Space and Terrestrial Robotic Exploration Laboratory
University of Arizona
1130 N. Mountain Ave, Tucson, AZ 85721
bykwon@email.arizona.edu

Jekanthan Thangavelautham
Space and Terrestrial Robotic Exploration Laboratory
University of Arizona
1130 N. Mountain Ave, Tucson, AZ 85721
jekan@email.arizona.edu

***Abstract*— Although many algorithms exist for complete, coverage path planning (CPP) by robots, most algorithms are not practical for real-world use, as they rely on perfect, prior knowledge of a static target environment, hardwired path planning or substantive human interaction, among other things. Moreover, many algorithms do not consider the real-world constraints of limited on-board power, computing, memory or communications, especially for low cost, multi-agent swarms. For aerospace applications, power-constrained CPP algorithms are critical because they can impact the effectiveness of future applications, such as the development of autonomous multi-robot teams for lunar site-preparation, mining and construction, or the development of terrestrial multi-robot teams to conduct visual or x-ray inspections of aircraft bodies. In this paper, we apply the Artificial Neural Tissue (ANT) control algorithm [1][2] to solve simulated CPP tasks, where multiple agents cooperate and completely or almost completely, cover 2-dimensional, basic geometric, open grid areas in linear or quasilinear time, where time complexity is measured by the number of robot time steps and the open grid cells to cover. In these ANT simulations, there is no central controller and the agents are constrained by limited time steps, a priori knowledge of the target environment, on-board memory and sensors, and no communications among themselves. However, the ANT agents do rely on pheromones/markers to track whether a grid cell has been visited, and receive information from a central station concerning total area coverage, time and global reference directions. In these CPP tasks, the performance of ANT is comparable to the best known grid-based, heuristic coverage algorithm with a quasilinear upper bound cover time [3][4].**

TABLE OF CONTENTS

1. INTRODUCTION

The Moon is the first destination in setting up an off-world base and for extracting critical resources to power a space economy. Significant deposits of water exist on the Lunar South and North Pole, but these regions are extremely inhospitable. To access these sites, autonomous and robust robotic systems are needed. Tele-robotic systems are an option, but they require large-scale infrastructure and placement of astronauts at the moon or the Lagrange points to minimize latency.

In comparison, autonomous robotic systems are a simpler option, but must operate efficiently at first to be productive. Coverage path planning (CPP) algorithms are critical because they can impact the effectiveness of autonomous multirobot teams in lunar site-preparation, excavation, open-pit mining and construction.

On Earth, the external visual inspection and internal x-ray inspection of aircraft bodies are other demanding tasks that can benefit from CPP. Typically, humans conduct these tedious and sometimes dangerous inspections. Small, light-weight robots that can carry and operate dangerous x-ray scanners are well-suited for these tasks. Again, one or more robot teams need to master CPP.

Although many algorithms exist for complete CPP by robots, most algorithms are not practical for real-world use, as they rely on perfect, prior knowledge of a static target environment, hardwired path planning or substantive human interaction, among other things. Moreover, many algorithms do not consider the real-world constraints of limited on-board power, computing, memory or communications, especially for low cost, multi-agent swarms.

In this paper, we apply the Artificial Neural Tissue (ANT) control algorithm [1][2] to solve simulated CPP tasks, where multiple agents cooperate and completely or almost completely, cover 2-dimensional, basic geometric, open grid areas in linear or quasilinear time, where time complexity is measured by the number of robot time steps and the open grid cells to cover. Trained by a simple genetic algorithm, and inspired by neuromodulation and coarse coding in the brain, ANT is an artificial neural network controller with a sparse neural network architecture, with adaptive activation functions that consist of a linear combination of real-valued step functions, which can replicate the Boolean XOR function in a single artificial neuron.

In these ANT simulations, there is no central controller and the agents are constrained by limited time steps, a priori knowledge of the target environment, on-board memory and sensors, and no communications among themselves. However, the ANT agents do rely on pheromones/markers to track whether a grid cell has been visited, and receive information from a central station concerning total area coverage, time and global reference directions. In these CPP tasks, the performance of ANT is comparable to the best known grid-based, heuristic coverage algorithm with a quasilinear upper bound cover time [3][4].

Preprint 2020 IEEE Aerospace Conference

In the following sections, we discuss the theoretical bounds of well-known grid-based CPP algorithms, highlight the novel features of ANT, describe the ANT robot controller implementation, described the simulated CPP tasks and review the simulated test results.

2. GRID-BASED COVERAGE ALGORITHMS

Grid-based, complete CPP algorithms inspired by ant pheromones, where a chemical or physical marker indicates whether a spot has been traversed by an agent, is a well-studied topic, but few studies analytically prove bounds on cover times, i.e. the time complexity in the number of grid cells to visit all cells at least once [4]. Such algorithms are difficult to use in real-world environments because they assume static environments that can be discretized into geometric grid cells, the grid environment is strongly connected, i.e. there is path from each grid cell to any other grid cell [6] and predetermined or hardwired path planning. However, these algorithms are useful benchmarks to evaluate and develop potential real-world coverage algorithms. Two well-known, grid-based CPP algorithms that have proven bounds on their cover times are the Learning Real-Time A* (LRTA*) and Node Counting algorithms [5]–[9], which are limited, look-ahead algorithms [6], related to reinforcement learning [5].

In grid-based CPP algorithms, each grid cell is assigned a binary, heuristic, probabilistic or energy potential value $h(v)$ that helps determine which direction or grid cell a robot should next move [10]. To evaluate their cover times, these algorithms are converted into graph search problems, where the grid cells are analogous to graph vertices $v \in V$ and the boundaries between adjacent grid cells are analogous to graph edges $e \in E(v)$, where $E(v)$ are all the graph edges connected to vertex v [10][11].

In Node Counting, the value-update rule for each vertex is $h(v) \leftarrow h(v) + 1$, where initially all $h(v) = 0$, and $h(v)$ is incremented by the value one, every time an agent visits vertex v [6]. Then, an agent at vertex v moves along edge $e \in E(v)$ to an adjacent vertex $\hat{v}$ with the lowest value $h(\hat{v})$ [6]. In the event of tied lowest values $h(\hat{v})$, the agent selects a random adjacent vertex $\hat{v}$ among these lowest values [6]. The foregoing move operation is captured by the update assignment $v \leftarrow next(v, e)$ [6]. So, the value $h(v)$ in Node Counting is solely dependent on local information concerning vertex v [6].

In LTRA*, the value-update rule for each vertex is $h(v) \leftarrow h(next(v, e)) + 1$, which means that the value $h(v)$ is dependent on the value of an adjacent vertex $\hat{v}$ with the lowest value $h(\hat{v})$ [6]. Initially, all $h(v) = 0$ in LTRA*. Since the value-update rule in LTRA* is a nested function dependent on $next(v, e)$, the value $h(v)$ implicitly captures non-local information about the graph (or grid environment), away from vertex v. Specifically, if the graph has not been searched completely and unvisited vertices exist, the LTRA* value $h(v)$ is an estimate of the number of edges (or steps) from vertex v to the nearest unvisited vertex v^*, where $h(v^*) = 0$, and LTRA* causes an agent to move toward that nearest, unvisited vertex [6]. On the other hand, the Node Counting value $h(v)$ only provides local information about vertex v and provides no information beyond its adjacent vertices. The difference in value-update rules between Node Counting and LTRA* has a substantial impact on theoretical cover times, where the LTRA* upper bound cover time is $O\left(n^2 - n\right)$ and the Node Counting cover time can be at least $\Omega\left(n^{\sqrt{n}}\right)$, where n is the number of vertices [6]. For an intuitive sense on why Node Counting possibly can have exponential cover times, consider the case of an agent searching an open grid area and then figuratively painting itself into a corner of visited grid cells, where the nearest unvisited grid cells are far away. Since the Node Counting value $h(v)$ only captures local information adjacent to vertex v, a Node Counting agent can cycle aimlessly among visited vertices before randomly reaching an unvisited vertex.

Notwithstanding the substantial theoretical difference in cover times, the empirical performance of the LTRA* and Node Counting algorithms are comparable in many simulated scenarios, but it is difficult to predict in advance what scenarios will cause the time complexity of Node Counting time to become adverse [6]. Accordingly, when the loss of power can have high, detrimental impacts on a mission or task, Node Counting algorithms should be avoided for complete CPP applications.

To our knowledge, the best grid-based, heuristic coverage algorithm is SWEEP with an upper bound cover time of $O\left(n^{1.5}\right)$ [3] [4]. However, SWEEP consists of hardwired path planning and defines rigorous protocols to mange how a group of cooperative cleaning robots move, pivot, signal and synchronize to clean a dirty target region [4]. SWEEP only relies on local sensing, has no prior knowledge of the target region but guarantees a quasilinear cover time because its search starts from the exterior boundary of the region, moves clockwise along the boundary, spirals inwards and avoids cleaning places that will cause the remaining target region to become discontinuous [4]. Accordingly, the takeaway from grid-based CPP algorithms is that robots will need some non-local sensing capabilities, or need to maintain a continuous, remaining area to cover, to avoid the possibility of detrimental cover times, which may cause the loss of power for the robots in complete CPP applications.

3. ARTIFICIAL NEURAL TISSUE

As describe more fully in [1] [2], ANT is a neural network control algorithm with a sparse, variable topology neural network, and adaptive activation functions. In this section, we highlight the novel features of ANT that allow it to outperform robotic controllers employing classic artificial neural networks (ANNs) [1]. Without loss of generality, we assume that ANT data inputs $x_i \in [-1, 1]$, neural network conneciton weights $w_i \in [0, 1]$ and activation function outputs $\{-1, 1\}$. However, these values can be other discrete or continuous Real values. We co-opt from neuroscience, the terms presynaptic and postsynaptic to indicate which nodes, i.e. artificial neurons, precede in a chain of node activation function outputs.

Variable Topology Network

Unlike classic ANNs, which are generally fixed topology, fully connected, feedforward networks, ANT is a sparse, variable topology, coarse coding neural network [2]. To implement this architecture, ANT consists of two neural networks, the decision and motor networks, which are co-located in the same 3-dimensional (3D) lattice space. The role of the ANT decision network is to select motor nodes that will comprise the ANT feedforward motor network [2].

During instantiation, ANT places decision and motor nodes into lattice cells, which may be co-located in the same lattice cell, but this is not required, a decision or motor node may be in a lattice cell independently.

The ANT activation process consists of two parts [2]. First, all decision nodes, $k = 1 \dots p$, receive data inputs x_i and yield an activation function output $z_k \in \{-1, 1\}$. If a decision node yields an excitatory output (e.g. $z_k = -1$ is excitatory), then the decision node figuratively releases a (chemical) neuromodulator that diffuses equally in all directions about the decision node, forming a 3D diffusion zone about the decision node. If several decision nodes diffuse, likely their diffusion zones will overlap in one or more lattice cells. The ANT algorithm identifies the lattice cells with the highest diffusion concentrations (i.e. most diffusion zone overlaps), and if any motor nodes are located in those cells, these motor nodes will comprise the ANT feedforward motor network [2]. This diffusion selection process occurs with each set of new data inputs x_i. So, the number of motor nodes in, and the topology of the feedforward motor network can vary with each set of data instances.

A postsynaptic motor node only receives inputs from presynaptic motor nodes in a 3×3 lattice cell grid on the prior hidden layer, whose central node is adjacent to, and on the same feedforward axis as the postsynaptic motor node. In classic ANNs, the postsynaptic node usually is fully connected to all nodes on the prior hidden layer. Whereas, the ANT feedforward motor network is sparse because of the diffusion selection process and the limited number of presynaptic node outputs that feed into each postsynaptic node.

In the second part of the ANT activation process, the first layer of the ANT feedforward motor network receives data inputs x_i, the activation function outputs flow through the network and the last layer of the ANT feedforward motor network acts as the output layer [2], comparable to a classic feedforward ANN.

ANT Activation Function

Contrary to classic ANNs that predominately use fixed, continuous functions, the ANT activation function is a linear combination of different step functions with trainable parameters

$$\begin{aligned}\Phi(\sigma) =& (1 - K_1)\left[(1 - K_2)\Phi_{down} + K_2\Phi_{up}\right] + \cdots \\ & K_1\left[(1 - K_2)\Phi_{ditch} + K_2\Phi_{mound}\right]\end{aligned} \quad (1)$$

$$\Phi_{down}(\sigma) = \begin{cases} -1, \sigma \geq \theta_1 \\ 1, \text{otherwise} \end{cases}$$

$$\Phi_{up}(\sigma) = \begin{cases} -1, \sigma \leq \theta_2 \\ 1, \text{otherwise} \end{cases}$$

$$\Phi_{ditch}(\sigma) = \begin{cases} -1, \min(\theta_1, \theta_2) \leq \sigma < \max(\theta_1, \theta_2) \\ 1, \text{otherwise} \end{cases}$$

$$\Phi_{mound}(\sigma) = \begin{cases} -1, \sigma \leq \min(\theta_1, \theta_2) \text{ or} \\ \quad \max(\theta_1, \theta_2) < \sigma \\ 1, \text{otherwise} \end{cases}$$

$$\sigma = \frac{\sum_{i=0}^{n} x_i w_i}{\sum_{i=0}^{n} |x_i|}$$

where activation function output $\Phi \in \{-1, 1\}$, parameters $K_1, K_2 \in \{0, 1\}$ and $\theta_1, \theta_2 \in [0, 1]$, weighted input σ, n is the number of presynaptic nodes that feed into the postsynaptic node, node inputs $x_i \in \{-1, 1\}$, $x_0 = 1$, weights $w_i \in [0, 1]$ and bias weight $w_0 \in [0, 1]$ [2]. Given the dominant (binary) coefficients K_1, K_2 in (1), Table 1 shows how the ANT activation function output reduces to one of the component step function in Equation 1.

Table 1. Conditional cases of ANT activation function output, Eq. 1

K_1	K_2	$\Phi(\sigma)$
0	0	Φ_{down}
1	0	Φ_{ditch}
0	1	Φ_{up}
1	1	Φ_{mound}

4. ANT ROBOT CONTROLLER

The ANT robot controller (ARC) implemented and tested in this paper substantially followed [2]. The ARC motor network consisted of three hidden layers, where each layer can contain a maximum of 20×20 nodes. So, the maximum number of possible ARC decision and motor nodes is 1,200 nodes each. All decision nodes, and all motor nodes on the first hidden layer of the feedfoward motor network receive inputs from the 26 robot sensors in Figure 1.

All parameters constituting an ARC instantiation, or individual, are contained in a symbolic genome, which consists of one tissue gene for the individual, one decision gene per decision node and one motor gene per motor node. Each decision gene, and motor gene on the first hidden layer, has a total of 34 trainable parameters, in which 27 are weight parameters (26 sensor input weights plus one bias weight). All remaining motor genes each have 17 trainable parameters, in which 10 are weight parameters (9 motor node weights plus one bias weight). The tissue gene only contains parameters that affect the genome's evolution.

Sensor(s)	Description
1 — 8	Short-range obstacle detector {-1,1}, -1 means obstacle present
9 — 16	Short- range cell visit detector {-1,1}, -1 means cell visited previously
17 — 19	Long-range front obstacle radar {-1,1}, -1 means entire range is occupied
20 —22	Long-range front visit radar {-1,1}, -1 means entire range has been visited
23	Current global reference direction of robot [-1,1]
24	Current global area coverage [-1,1]
25	One-bit memory of obstacle encounter {-1,1}, -1 means prior move blocked due to obstacle presence
26	Current percentage of maximum allowable steps taken [-1,1]

Figure 1. Breakdown of ARC Sensors

Simple Genetic Algorithm

For the simple genetic algorithm (SGA) that optimizes the trainable ARC parameters, an initial population of 100 ARC individuals, or genomes, were instantiated randomly. Each ARC individual (robotic controller) was created by instantiating two seed motor nodes on each hidden layer, and then

instantiating adjacent motor nodes on the same hidden layer to create a 3×3 grid of motor nodes centered about a seed node. Corresponding to this 3×3 grid, motor nodes on other hidden layers are instantiated to create a 3×3×3 column of motor nodes through the feedforward motor network. The foregoing process is repeated for each seed node. After all motor nodes are instantiated, decision nodes are instantiated based on a ratio of the number of decision to motor genes, which are parameters in the tissue gene.

The SGA maximizes a simple global fitness function, Equation 3, to train the ARC parameters. The population fitness score is that of the best individual in the population. Mating occurs in the population on a rank-based, roulette selection method, where individuals in the top half of fitness scores crossover among themselves to create children, then these genomes under go possible mutation and possible insertion of new (motor) genes. If a crossover does not occur because the ANT compatibility criterion is not met, then parent genomes undergo a mutation rate of 10 to 20 percent to generate children. After crossover, the mutation rate for genomes is between one and three percent, and the probability of insertion of a new (daughter motor) gene is between 25 and 75 percent in each generation.

Simulated CPP Task

Computer simulations of the CPP task consisted of attempts by a mutli-robot team to completely cover the open grid areas in Figures 2–7, or Scenarios (a)–(f), respectively, where the robot time steps were limited. Scenarios (a)–(d) are fixed square, triangular and circular areas, where robots would start from fixed locations and directions on the area exteriors. Scenarios (e)–(f) are fixed square and circular areas, where robots would start from fixed locations and random directions, in the area interiors. In these figures, the yellow cells are obstacles or boundaries, blue cells are open cells unvisited by a robot, cyan cells are open cells previously visited by a robot and cyan cells with a white (or colored) line from the cell center to its exterior (a direction vector) are cells occupied by a robot. Any cyan cell with a robot is recognized as an obstacle by other robots, and more than one robot cannot occupy a cell at the same time. Each scenario was an independent simulation. The square and circular scenarios used a team of four robots, limited to 100 time steps per robot during training, and the triangular scenario used a team of three robots, limited to 70 time steps per robot during training.

The number of open grid cells were 400, 333 and 210 cells for the square, circular and triangular areas, respectively. For the simulations, area coverage and the global fitness function were defined as

$$\text{area coverage} = \frac{U_{\text{beg}} - U_{\text{end}}}{U_{\text{beg}}} \tag{2}$$

$$\text{global fitness} = \text{area coverage} \cdot \text{visit award} - \cdots \\ \frac{R}{U_{\text{beg}}} \cdot \text{revisit penalty} \tag{3}$$

where U is the total number of unvisited cells, U subscripts "beg" and "end" mean at the beginning and end of the simulation, respectively, R is the total number of revisits to cells, visit award equals 1 and revisit penalty equals 0.5. So, the maximum global fitness score was 1, which meant complete coverage of the search area and no cells were revisited by the robots. If no movement or area coverage occurred in the simulation, the default global fitness score was -0.5.

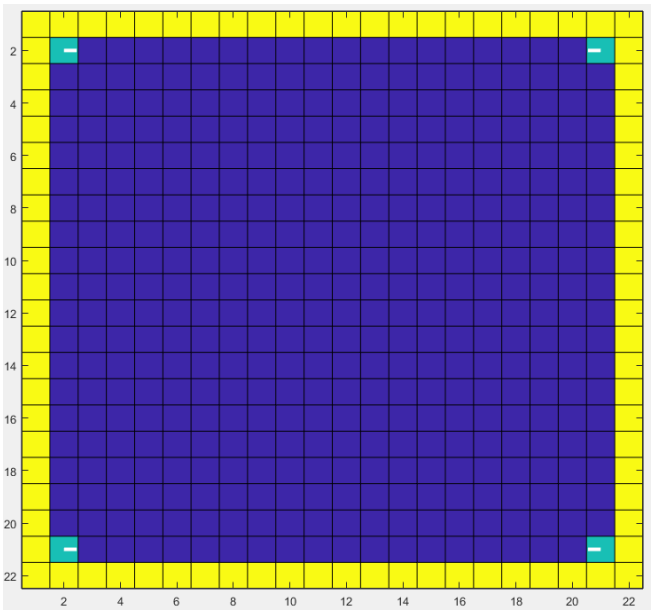

Figure 2. Scenario (a) - Exterior square corner

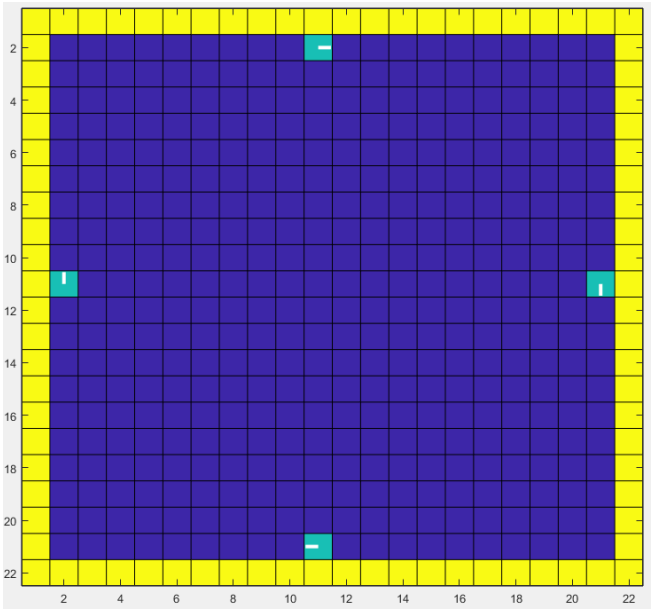

Figure 3. Scenario (b) - Exterior square side

Robot Platform

The ARC computer simulations used a team of identical, nonholonomic robots to search and train on scenarios (a)–(f). No central controller existed. All robot team members were independent agents endowed with the same control algorithm, i.e. ARC individual. An ARC individual represents a specific ARC instantiation, and not a robot in a multi-robot team. No communications between the robots existed and robots were not explicitly aware of the existence of other robots. Via their sensors, robots appeared to each other as obstacles. Except through training, robots had no prior knowledge of the search environment. A centralized base station existed and maintained a global map to provide global reference directions and area coverage information to the robots. However, the robots had no mapping algorithm, and except for global reference

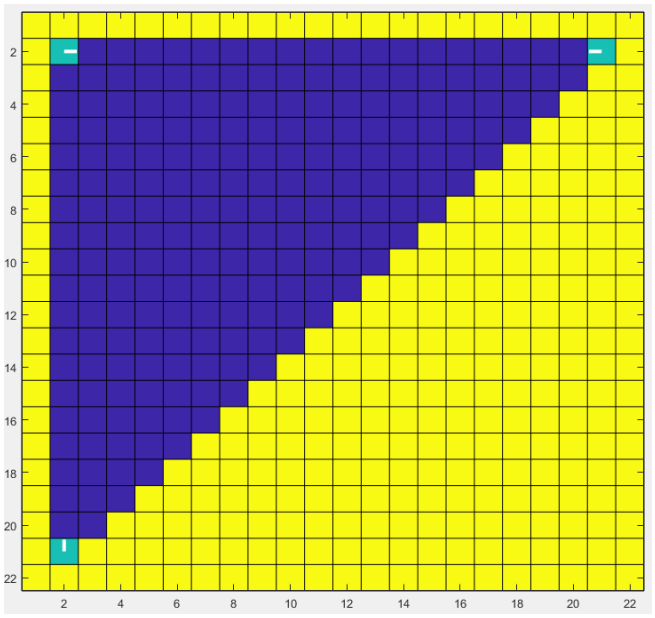

Figure 4. Scenario (c) - Exterior triangle

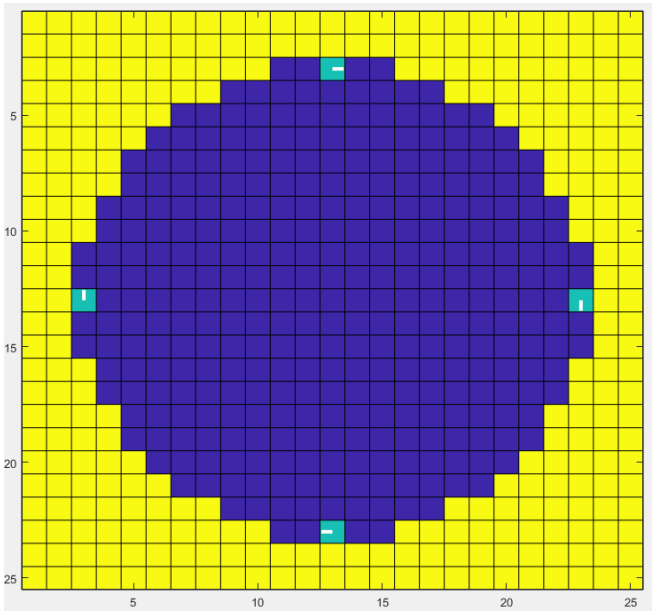

Figure 5. Scenario (d) - Exterior cirlce

directions, no access to a map.

During each time step, each robot received 26 robot sensor inputs, in which 23 were on-board sensors that yielded binary sensor data $\{-1, 1\}$, see Figure 1. The remaining three sensors provided global information from a central base station that yielded sensor data on the range $[-1, 1]$. Specifically, Sensor 23 provided each robot its current global reference direction; Sensor 24 provided the current global area coverage; and Sensor 26 was a countdown clock that provided each robot the current percentage of time steps taken out the maximum allowable steps for each robot.

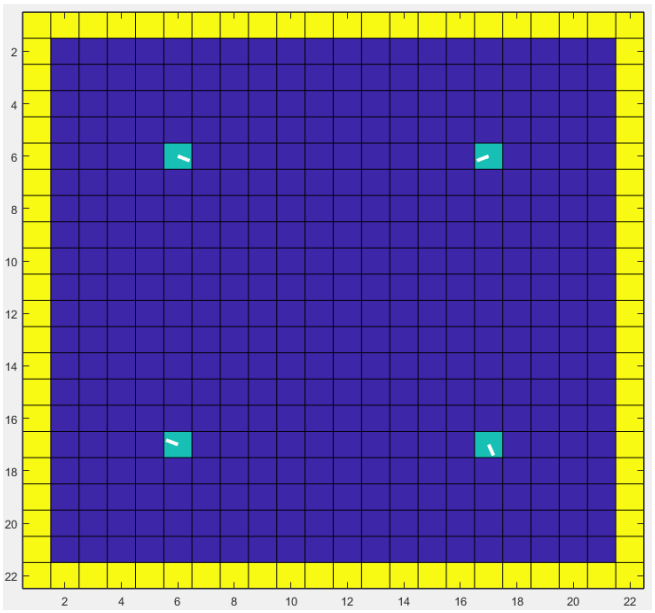

Figure 6. Scenario (e) - Interior square

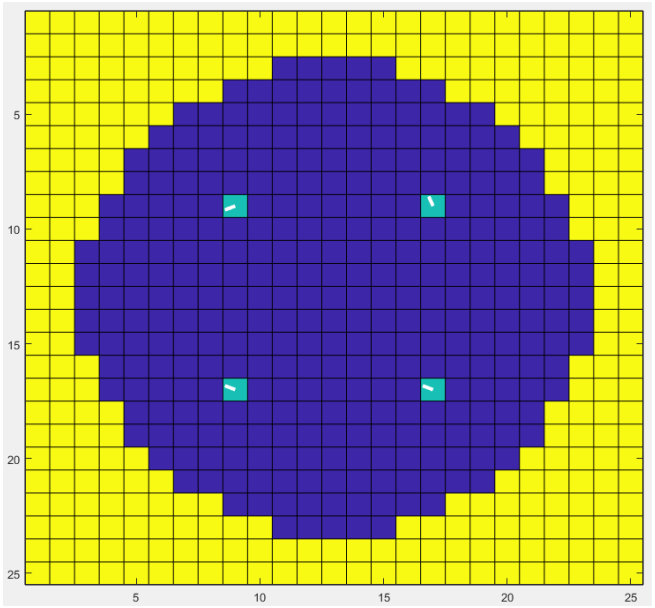

Figure 7. Scenario (f) - Interior circle

ARC Output Behaviors

Each ARC motor node is endowed with an output behavior $b = 1 \ldots 8$, listed in Table 2. To determine a robot's action, ARC calculated a weighted behavior score for each output behavior b, and the robot executed the behavior with the highest weighted behavior score greater than or equal to 0.5. In the event of a tie, ARC randomly selected among the tied behavior scores. For each output behavior b, its weighted behavior score was based on the activation function outputs y_i of the motor nodes $i = 1 \ldots m_b$ on the feedforward motor network output layer with output behavior b,

$$\frac{\text{number of nodes with } y_i = 1}{\text{total number of nodes with } y_i = \{-1, 1\}}. \quad (4)$$

If motor nodes with output behavior b did not exist or had

no activation function outputs on the output layer, then the weighted behavior score for b was null, and output behavior b was not included in the pool of possible robot actions. During each time step, ARC first evaluated and executed the turn in place output behaviors $\{1, 5, 6, 7, 8\}$, and then, evaluated and executed the cell move output behaviors $\{2, 3, 4\}$. Furthermore, the robots in a team moved asynchronously, where the order of which robot moved first, second, third, etc. were random during each time step. Robot is always pointed robot reference east, and motions are relative to this direction. Moves are one cell displacements, turns are in place, and "turn random" means a random pick from behaviors $\{1, 5, 6, 8\}$. In each time step, ARC evaluates and executes a turn behavior first, and then evaluates and executes a move behavior last.

Table 2. ARC Motor Node Output Behaviors

Behavior	Motion
1	turn north
2	move northeast
3	move east
4	move southeast
5	turn south
6	turn southeast
7	turn random
8	turn northeast

ARC Training

The ARC populations were trained using a Monte Carlo approach, where all individuals ran one or more complete training episode(s) to determine their fitness scores, and this score was used by a SGA to determine which individuals crossover and the number of training generations. A complete training episode is a simulated run to cover the entire search area in one attempt. Scenarios (a)–(d) used one training episode per individual per generation. Scenarios (e)–(f) used four training episodes per individual per generation, and averaged these training fitness scores to determine an individual's training fitness score because the random robot start directions in these scenarios affected fitness scores. For each scenario, 100 different populations, each consisting 100 different ARC individuals, were instantiated and trained on the scenario. Each trained population is called an EA run.

Each EA run was trained for a maximum of 500 SGA generations. If an ARC population fitness score met or exceeded the training fitness cutoff of 0.90 to 0.95 for scenarios (a)–(d) or 0.80 for scenarios (e)–(f), for ten generations consecutively, the EA training run was halted early. Extending the training past 500 generations did not yield a material difference in relative fitness scores among the scenarios, or qualitative difference in the robot paths. The ARC algorithm was written and tested in Matlab 2018b on a local Windows 10 (64-bit) workstation with an Intel Xeon (Broadwell) CPU, and trained on Intel Xeon (Broadwell) university clusters using Matlab C executables (.mex) generated with GCC 6.3.0 compilers and Matlab 2018b or 2019a on these clusters.

5. TEST SIMULATION RESULTS

After the ARC training, the final evolved ARC populations were tested on their scenarios to establish baseline test results. For these results, all ARC individuals in a population ran four test episodes in their scenario, these test fitness scores were averaged to determined an individual's fitness score and the population fitness score was that of the best individual in the population. The mentions of ARC individuals or solutions hereafter, refer to these best individuals.

Each scenario had 100 EA test runs and Table 3 summarizes the baseline test results. In this table, the column titled "Total steps" is the maximum allowable time steps per robot times the number of robots. The square and circular scenarios used a team of four robots, where each robot was limited to 100 time steps, and the triangular scenario used a team of three robots, where each robot was limited to 80 time steps.

Scenarios with Exterior Start Positions

For scenarios (a)–(d), where robots started from the area exteriors, some ARC solutions provided complete or nearly complete area coverage, with global fitness scores in the mid-0.90s or better. In scenario (a), where robots started from the corners of an open square area, ARC produced solutions with complete area coverage in linear time, where time complexity was measured by the total robot time steps needed to cover the open grid cells. In this scenario, robot start locations and directions were fixed, robots in the northwest and northeast corners pointed toward each other, and robots in the southwest and southeast corners also pointed toward each other. The ARC solutions with complete area coverage changed the robot directions to all move in the same direction and spiral inward to cover the entire square. In scenario (a), the asynchronous robot movements did not affect the solution paths or stability of the fitness scores among the test episodes.

In scenario (b), where robots started on the exterior sides of an open square, ARC produced a stable solution with complete area coverage and a fitness score of 0.99, in linear time. In this ARC solution, robots also spiraled inward to cover the entire square. However, other ARC solutions in this scenario were unstable and produced volatile fitness scores. For example, two test episodes of one ARC solution, had fitness scores of 0.94 and 0.74, respectively. In these cases, the robots first moved along the exterior boundary of the square, moved toward the center of the square and then attempted to partition the square area into quadrants. However, the asynchronous robot movements affected whether the robots successfully crossed paths in the center to establish the quadrants, and caused the fitness score difference above.

In scenario (d), where robots started from the exterior of an open circular area, some ARC solutions had nearly complete area coverage of 0.96 and fitness scores of 0.95, in quasi-linear time. In these solutions, the robots followed the exterior boundary and spiraled inward to fill the area. Another ARC solution had area coverage of 0.96 but lower fitness score of 0.89, in which the robots followed straight paths to fill the circular area and left some grid cells on the boundary unvisited.

In scenario (c), where robots started from the exterior of a triangular area, ARC yielded solutions with complete area coverage and fitness scores of 0.92 and 0.99, in quasi-linear time. The fitness score difference is due to the difference on how the ARC solutions maneuvered robots at the southern triangle corner.

Table 3. ARC Baseline Test Results

ANT Robotic Controller (ARC) Baseline	Number robots	Beginning unvisited	Total steps	Area coverage					
				Mean	Median	Std	Max	Min	>= 80%
(a) Exterior square corner	4	396	400	0.96	0.98	0.04	1.00	0.70	98
(b) Exterior square side	4	396	400	0.92	0.92	0.05	1.00	0.75	98
(c) Exterior triangle	3	207	240	0.78	0.90	0.23	1.00	0.27	64
(d) Exterior circle	4	329	400	0.88	0.89	0.06	0.97	0.67	91
(e) Interior square	4	396	400	0.76	0.76	0.05	0.89	0.59	19
(f) Interior circle	4	329	400	0.76	0.77	0.07	0.90	0.48	28

ANT Robotic Controller (ARC) Baseline	Number robots	Beginning unvisited	Total steps	Global Fitness					
				Mean	Median	Std	Max	Min	>= 80%
(a) Exterior square corner	4	396	400	0.95	0.97	0.05	1.00	0.65	97
(b) Exterior square side	4	396	400	0.88	0.88	0.07	0.99	0.64	92
(c) Exterior triangle	3	207	240	0.74	0.84	0.24	0.99	0.26	54
(d) Exterior circle	4	329	400	0.82	0.86	0.09	0.95	0.49	66
(e) Interior square	4	396	400	0.66	0.67	0.07	0.82	0.48	2
(f) Interior circle	4	329	400	0.62	0.62	0.08	0.78	0.46	0

Test results for 100 evolutionary algorithm (EA) runs for each scenario. The table column "Beginning Unvisited" is the number of open cells to cover, "Total steps" is the maximum allowable time steps for each robot times the number of robots, "Std" is standard deviation and ">=80%" means the number of EA runs with results greater than or equal to 0.80. Visit award equals 1 and revisit penalty equals 0.50 for the global fitness function.

Scenarios with Interior Start Positions

In scenarios (e)–(f), where robots started from the area interiors, the area coverage and global fitness scores were lower than the other scenarios. However, some ARC solutions in scenarios (e)–(f) yielded valuable initial paths to cover the areas. In these scenarios, the robot start locations were fixed but the start directions were random. In scenario (e), where robots started from the interior of an open square area, the best ARC solutions provided area coverage of about 0.90 and fitness scores in the mid-0.80s. Encouragingly, these ARC solutions caused robots to spiral outward and created rectangles of visited (cyan) cells in the center of the square area. When robot start directions were not perpendicular to the square boundaries, the robots attempted path adjustments to form such rectangles. When a robot did not make such adjustments, the robot created a diamond pattern with visitied (cyan) and unvisited (blue) cells, which negatively impact area coverage and fitness scores.

In scenario (f), where robots started from the interior of an open circle, the best ARC solutions did not yield paths similar to scenario (e). Instead, the ARC solutions caused the robots to move to the exterior boundary of the circular area, and spiral inward or criss-cross the circular area, which resulted in area coverage of 0.85 and 0.91, and fitness scores of 0.80 and 0.81, respectively.

6. Discussion

The test results align with the takeaways in Section 2 about the LTRA*, Node Counting and SWEEP algorithms. Comparable to SWEEP, ARC solutions that started from the exterior of a search area and spiraled inward, scenarios (a)–(d), performed better than those that started from area interiors, scenarios (e)–(f). When ARC solutions began from area interiors, chances increased that robot paths would create discontinuous search areas. Notwithstanding the foregoing, the ARC solution in scenario (e) may be valuable when combined with the ARC solutions in scenarios (a)–(d). For example, a multi-robot team could run a primary, complete CPP task by spiraling into the target area. After this is completed, the multi-robot team can run a secondary, complete CPP

task, with about 90% area coverage, by spacing themselves evenly in the target area, and spiraling outward. In this case, the secondary task should have a time step limit to prevent wasting power on inefficiently covering the last 10% of the area. This remaining 10% area can be covered by the same robots re-calibrated for this specific task, or under human control.

7. Conclusion

In summary, robotic controllers based on the ANT control algorithm provided in some cases, nearly complete area coverage in quasi-linear time, and other cases, complete area coverage in linear time, in computer simulations on open, basic geometric grid areas, where searches began from area exteriors. This performance was comparable to theoretical cover time bounds of the best grid-based, area coverage algorithm. Furthermore, ANT achieved these results with no central controller, no communications among the robots, no mapping algorithm, no hardwired path planning and limited on-board power, memory and sensors.

References

[1] Thangavelautham, J., DEleuterio, G. (2012, April). Tackling learning intractability through topological organization and regulation of cortical networks. IEEE Transactions on Neural Network and Learning Systems, 23(4), 552 - 564.

[2] Thangavelautham, J., Law, K., Fu, T., Samid, N., Smith, A., DEleuterio, G. (2017, January). Autonomous multirobot excavation for lunar applications. Robotica, 35 (12), 23302362. Retrieved from https://doi.org/10.1017/S0263574717000017.

[3] Altshuler, Y., Bruckstein, A. (2010). The complexity of grid coverage by swarm robotics. In M. Dorigo (Ed.), Swarm intelligence: 7th international conference, ants 2010 (p. 536543). Springer Berlin Heidelberg.

[4] Altshuler, Y., Pentland, A., Bruckstein, A. (2018). Swarms and network intelligence in search. In (Vol. 729, p. 15-49). Springer.

[5] Barto, A. G., Bradtke, S. J., Singh, S. P. (1995). Learning to act using real-time dynamic programming. Artificial Intelligence, 72(1), 81138.

[6] Koenig, S., Szymanski, B., Liu, Y. (2001, October). Efficient and inefficient ant coverage methods. Annals of Mathematics and Artificial Intelligence, 31(1), 4176.

[7] Korf, R. E. (1990, March). Real-time heuristic search. Artificial Intelligence, 42(2-3), 189211.

[8] Pirzadeh, A., Snyder, W. (1990). A unified solution to coverage and search in explored and unexplored terrains using indirect control. In IEEE International conference on robotics and automation proceedings (Vol. 3, p. 21132119).

[9] Szymanski, B., Koenig, S. (1998). The complexity of node counting in unidirected graphs (Tech. Rep. No. CS-98-02). Department of Computer Science, Rensselaer Polytechnic Institute.

[10] Galceran, E., Carreras, M. (2013, December). A survey on coverage path planning for robotics. Robotics and Autonomous Systems, 61, 12581276.

[11] Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., Thrun, S. (2005). Principles of robot motion. Cambridge, MA: MIT Press.

Biography

Byong Kwon *is a US Intelligence Community Postdoctoral Research Fellow working on autonomous multi-robot systems and applied artificial intelligence, with a focus on bio-inspired and evolutionary algorithms. Prior to SpaceTREx, BK was a USAID-Arizona State University (ASU) Global Development Research Scholar in Colombia (May 2017 April 2018) engaged in humanitarian demining research. In December 2019, BK will receive his PhD in Applied Mathematics for the Life and Social Sciences from ASU. Prior to ASU, BK obtained his Bachelor (2012) and Master (2015) of Sciences in Mathematics at George Mason University in Fairfax, Virginia, with a focus on nonlinear optimization and machine learning.*

Jekanthan Thangavelautham *has a background in aerospace engineering from the University of Toronto. He worked on Canadarm, Canadarm 2 and the DARPA Orbital Express missions at MDA Space Missions. Jekan obtained his Ph.D. in space robotics at the University of Toronto Institute for Aerospace Studies (UTIAS) and did his postdoctoral training at MIT's Field and Space Robotics Laboratory (FSRL). Jekan Thanga is an assistant professor and heads the Space and Terrestrial Robotic Exploration (SpaceTREx) Laboratory at the University of Arizona. He is the Engineering PI on the AOSAT I CubeSat Centrifuge mission and is a Co-Investigator on the CatSat 1 CubeSat mission.*