Distributed Colony-Level Algorithm Switching for Robot Swarm Foraging

Nicholas Hoff, Robert Wood, Radhika Nagpal

Abstract

Swarm robotics utilizes a large number of simple robots to accomplish a task, instead of a single complex robot. Communications constraints often force these systems to be distributed and leaderless, placing restrictions on the types of algorithms which can be executed by the swarm. The performance of a swarm algorithm is affected by the environment in which the swarm operates. Different environments may call for different algorithms to be chosen, but often no single robot has enough information to make this decision. In this paper, we focus on foraging as a multi-robot task and present two distributed foraging algorithms, each of which performs best for different food locations. We then present a third adaptive algorithm in which the swarm as a whole is able to choose the best algorithm for the given situation by combining individual-level and distributed colony-level algorithm switching. We show that this adaptive method combines the benefits of the other methods, and yields the best overall performance.

1 Introduction

The performance of a robot swarm algorithm is affected by the environment in which it operates. In a search task for example, the location and number of search targets and presence or absence of obstacles could affect the efficiency of the swarm. One algorithm may be fast but fail in the presence of obstacles, and a slower one may be more resilient. Because the specifics of the environment are generally not known before hand, we would like the swarm itself to be able to intelligently change its own algorithm based on the environment.

Harvard University
`{nhoff, rjwood, rad}@eecs.harvard.edu`

An interesting challenge is whether a robot swarm, as a whole, can assess the success or failure of an algorithm and, as a whole, switch algorithms to increase its success.

Colony-level algorithm switching is difficult for two reasons. First, the information on which the colony will base its decision is distributed throughout the environment and not detectable by any one robot. Therefore, the environment detection must be distributed. Second, if a swarm algorithm relies on strong coordination, then switches must be nearly unanimous and synchronized. Changing algorithms will not help unless all individuals change to the same algorithm at the same time. Both of these actions—environment sensing and algorithm switching—must truly take place on the colony level, as opposed to the individual level.

In this work, we focus on the problem of foraging for robot swarms. Foraging algorithms enable a collection of robots to search a space for a goal (the 'food'), then return it incrementally to the nest. We assume robots with simple sensing and communication capabilities that can exchange simple messages (a few bits) directionally with other robots within a short range. Since the robots do not have global localization or odometry, coordination within the swarm is essential for performing the task efficiently.

We present three distributed foraging algorithms. All of the algorithms are based on a few core "sub-algorithms" which the swarm intelligently switches between in various combinations and at various times. The first is a gradient–based method in which robots form a stationary beacon field around the nest to create gradients to the nest and food. The second is an area–sweeping algorithm (called "sweeper") in which robots use virtual forces to coordinate and form a line which sweeps the world. Finally, in the third algorithm ("adaptive"), the colony cooperates to detect when one algorithms is failing and switches to another, thus increasing performance.

To evaluate the algorithms, we place food at varying distances from the nest. We find that the gradient algorithm works fast but only in a short range, and the sweeper algorithms is slower but has a larger range. We show that the adaptive algorithm is capable of high-level switching, and that the swarm is able to choose the algorithm best suited to the current food location.

1.1 Related Work

Gradient-based Foraging Gradient-based foraging methods create a gradient leading to the goal, using the sensing capabilities of the robot, such as chemical sensing or communication. Algorithms using many different types of sensing capabilities have been studied. Algorithms exist for robots with global positioning and global communication [18], robots that use physical marks to leave a trail [9, 11], robots that use a pre-deployed sensor network [10], and robots that use deployable beacons [6]. Our work focuses on robots with directional communication. Payton et.al. have developed an algorithm in which each robot can receive messages in a small radius, and use this to create a virtual pheromone. Our gradient algorithm [8] is similar to [5], [19], and [21] in that directional communication is used to transmit relative position information to establish the gradient. Networking researchers would also recognize the gradient algorithm as being very similar to "hop-count" routing [22].

Area-Sweeping Foraging In this algorithm, inter-robot virtual forces are used to make movement decisions which cause the swarm to adapt a line shape. Spears studied physics-based control of vehicle swarms, with attractive and repulsive forces forming a lattice of vehicles [20]. In the field of sensor networks, Howard et.al. have used potential fields to achieve dispersion of nodes [7]. The closest work to our application is Balch's notion of social potentials [3]. Social potentials involve robots navigating to a goal while remaining in a formation, feeling virtual forces based on the position of the goal and the relative positions of other robots. These algorithms are focused on maintaining a formation. We use a similar concept in which robots feel virtual forces, and we show how to use the formation to search the region. We use this to develop a virtual forces-based foraging method.

Algorithm Switching Multiple behaviors within a single algorithm are well-known inside the swarm algorithm community [16]. Matarić and Arkin have worked extensively in behavior-based robotics [2, 4]. It is common for individual robots to switch between the behaviors of food collecting, obstacle avoidance, and resting, for example. Parker has studied distributed consensus in a swarm setting, using it to enable the swarm to move between subtasks in an overall task [17]. We focus on foraging, and do not use formal distributed consensus (or assume that our robots are '*well-stirred*'). McLurkin has developed a large range of robot swarm behaviors[14] as well as dynamic task assignment methods for individual robots within a swarm [13]. These methods focus on individuals, whereas we need a method for the swarm as a whole to switch algorithms.

The central contribution of this work is an adaptive foraging method in which the swarm makes colony-level decisions based on distributed information, choosing the algorithm best suited to the given food location.

2 Robot and Task Model

For our robots, we use a simple model inspired by recent swarm robot hardware, such as the E-Puck [15] (shown in figure 1b), the RBZ communication board (an E-Puck extension described in [1]), McLurkin's SwarmBots [12], and Payton's pherobots [5]. We assume a simple non-holonomic robot that moves and turns in continuous space. Each robot has sensors for nest, food, and obstacles in direct proximity to the robot. The sweeper algorithm also requires two of the robots to have compasses. Each robot can communi-

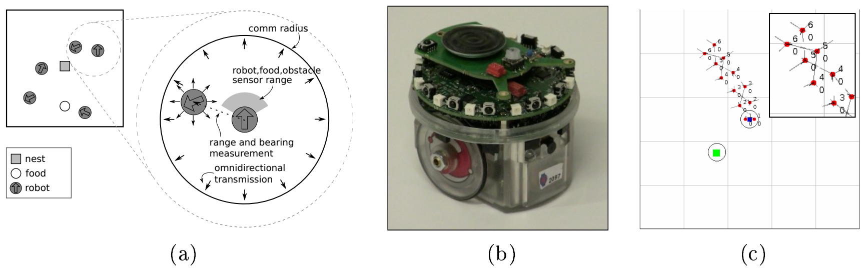

comm radius
robot,food,obstacle sensor range
range and bearing measurement
omnidirectional transmission
nest
food
robot
(a)
(b)
(c)

Fig. 1: The communication and sensing capabilities of the robots (1a), the E-Puck robot with communication ring (1b), and a simulation snapshot with a blowup for detail (1c).

cate with nearby robots and measure the range and bearing from which each transmission came. Robots do not have global position measurement or global communication. See figure 1a.

For the foraging task, we assume a world with a nest in the middle and one unlimited food source placed randomly. The swarm must find the food, then begin returning food units to the nest. Robots can pick up / drop food units when in direct proximity to the food / nest. Because the robots have no direct position measurement system, they must coordinate in order to maintain and share information about their own position and the food position.

To test the algorithms, we developed a continuous-world multi-robot simulator. The simulator models robots, food, and the nest, along with proper movement and interactions (collisions, communication). We chose a continuous world model over a gridded world environment so that the algorithms would face real problems such as collisions and congestion. A snapshot of the simulator is shown in figure 1c.

3 Algorithm Description

In this section, we will describe the three algorithms and the means by which algorithm switches are made at the individual- and colony-level. Figure 2 diagrams the relationship between the algorithms and their parts.

3.1 Gradient Algorithm

Robots need a way to navigate to the food and the nest. The gradient algorithm provides this information using two gradients—one leading to the nest, and a second leading to the food once it is found. To implement these gradients, some robots decide to stop their normal food searching and become fixed

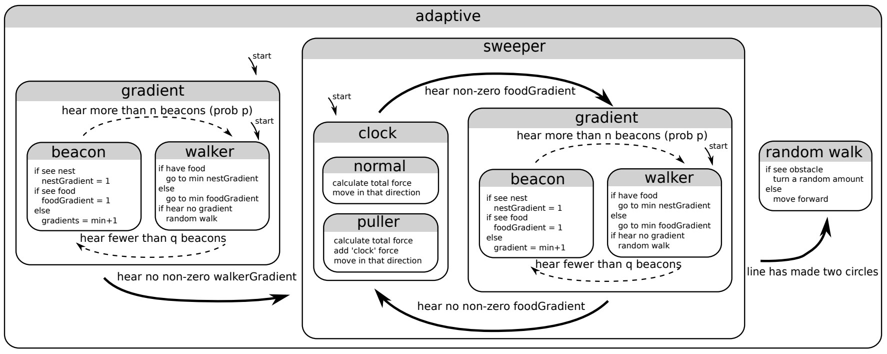


Fig. 2: This figure describes the relationships between the gradient, sweeper, and adaptive algorithms, and their switches. The "start" arrows indicate how each algorithm begins. Bold arrows indicate colony-level switches and dotted arrows indicate individual-level switches.

beacons. These beacons transmit two numbers (one for each gradient), which the remaining robots can use to navigate. As the swarm expands outward from the nest, the beacon network expands and creates the nest gradient. Once the food is found, the food gradient is developed. At that point, robots can navigate to either location to efficiently return food. (See also [8].)

3.1.1 Local Description

Beacon: Robots acting as beacons are stationary, and broadcast two numbers, called `nestGradient` and `foodGradient`. They listen for all other beacons in their communication range and record the `nestGradient` and `foodGradient` of each one. Beacons find the minimum of all `nestGradient` values they have received, increment that by one, and take that as their own `nestGradient`. An analogous procedure is used to calculate `foodGradient`. These new values are then broadcast by the beacon. Any beacon directly next to the nest/food broadcasts a 1 for its `nestGradient`/`foodGradient`.

If a beacon has no information about its distance from the food (as happens early in the run, before the food has been found), it broadcasts 0. The value of 0 is treated specially—when a beacon hears a 0, it does not include it in its normal 'minimum plus one' calculation.

Walker: Walker robots always attempt to navigate either to the food or the nest, depending on whether they have food. In either case, a walker measures the bearing to the minimum gradient value toward the target of interest, and moves in that direction. If a walker has no information about where it should go (it can only hear 0), it does a random walk.

Beacon to Walker transition: If a beacon robot can detect more than 4 other beacons, it will become a walker robot with a 20% chance. This

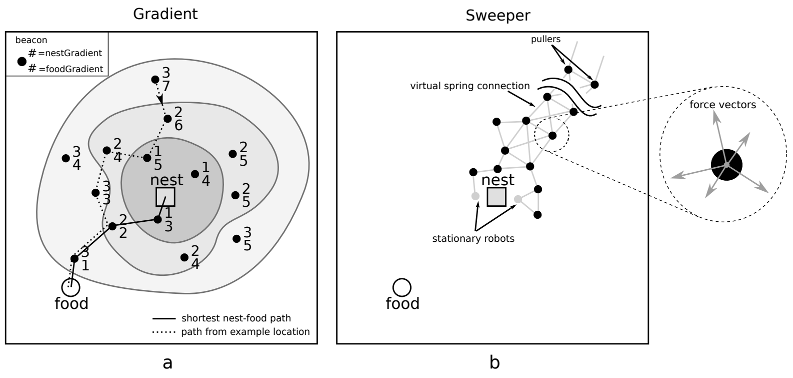


Fig. 3: Example situations in the gradient algorithm (a) and the sweeper algorithm (b). Walkers in (a) are not shown to reduce clutter. The gray contours roughly depict the gradient to the nest.

probabilistic effect is required to prevent several beacons, all of whom can collectively hear each other, from becoming walkers at exactly the same time and leaving a hole in the beacon field.

Walker to beacon transition: A walker robot will decide to become a beacon if it can only detect 1 or 2 other beacons.

3.1.2 Global Behavior

All robots start as walkers clustered around the nest. Some of them will decide to become beacons immediately because initially, there are no beacons. The remaining walkers will begin searching for the food. There will be no `foodGradient` (it will all be 0), so they will random walk. As they wander away from the nest, some will decide to become beacons, and the beacon field will expand away from the nest. Eventually one of the robots could stumble across the food, and would then begin transmitting 1 for its `foodGradient`, causing the food gradient to form. At this point, any walker can listen to the beacons near it and know how far it is from the food and how to get there. All the walkers immediately start moving directly toward the food. As walkers pick up food, they use the gradient field to bring it to the nest. Figure 3a illustrates an example snapshot of the gradient algorithm.

3.2 Sweeper Algorithm

A different strategy for search or foraging involves individuals forming a "search front" and systematically sweeping an area to find an object. Here we describe an algorithm that uses virtual forces to form a line of robots extending from the nest that sweeps the world like the hand of a clock. When the line finds food, some fraction of the robots remain as beacons while others act as walkers to return the food.

Fundamentally, this strategy creates a 1D structure of robots (roughly a line), as opposed to the gradient strategy which creates a 2D structure of robots (roughly a circle). The 1D structure is expected to be able to sweep a larger area than could be 'filled in' with the same number of robots

3.2.1 Local Description

Normal: In the sweeper algorithm, all robots are always transmitting. Each robot measures the range and bearing to all the other robots in its communication range. Based on the position of each other robot, it calculates a virtual force on itself. For each robot detected at relative position $\overrightarrow{r}$, this force is

$$\overrightarrow{F} = a\frac{\overrightarrow{r}}{r_c} - b\hat{r}$$

where r_c is the communication range of the robot and a and b are empirically chosen constants. In other words, the force is similar to what would be experienced if there were a virtual spring between the robots. The robot sums the virtual forces from each other robot in its communication range, and moves in that direction by an amount proportional to the magnitude of the force. There is one special case: two robots directly next to the nest never move regardless of the virtual forces on them.

While the robots are calculating forces and moving, they are simultaneously using communication to establish a gradient field similar to the one described in the gradient algorithm. This does not require extra communication. The data content of the signal encodes the two gradient values, and range and bearing to the transmitter are used to calculate the virtual forces.

The robot treats the `nestGradient` exactly as before, updating it using the $min + 1$ algorithm. `foodGradient` is treated slightly differently. Any time a robot sees a non-zero `foodGradient`, it temporarily stops executing the sweeper algorithm and switches to gradient. If the `foodGradient` returns to zero, the robot returns to executing the sweeper algorithm.

Puller: Two robots are pre-determined to be "puller" robots. These participate in the virtual forces system described above, but they also feel one additional force. These two robots must use their compasses to measure the relative bearing to north (the unit vector $\hat{N}$), then put a virtual 'clock' force on themselves equal to

$$\overrightarrow{F_c} = c\hat{N}R\left(\frac{2\pi t}{T}\right)$$

where $R(\theta)$ is simply the 2D rotation matrix,

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$\overrightarrow{F_c}$ is a force which simply rotates around like the hand of a clock as time t increases. The parameter c is an empirically chosen magnitude and T sets the period of the rotation. T is determined by the puller, and can be changed at any time based on the puller's `nestGradient` value.

3.2.2 Global Behavior

When the sweeper algorithm begins, all robots calculate forces and begin moving appropriately. Initially, repulsive forces cause the swarm to expand into a tight clump around the nest. The pullers will be forced to the edge of the pack and a line of robots will form extending from the nest to the pullers. This line of robots will rotate as the pullers pull it around, sweeping around the world like the hand of a clock. When the line encounters food, it stops moving and the swarm returns the food using the gradient algorithm, with walkers moving along the line of robots already established. When the food source is exhausted, the forces resume and the line keeps sweeping.

One can imagine that longer lines (with more robots) would need to rotate slower than shorter lines. Hard-coding the period T would require knowing the number of robots in the swarm, which is not a scalable solution. Instead, the pullers set T based on their `nestGradient`. A higher value for `nest-Gradient` indicates a long line, so the puller will choose a larger T.

Figure 3b illustrates an example snapshot of the sweeper algorithm.

3.3 Adaptive Algorithm

The first two algorithms, gradient and sweeper, each have strengths and weaknesses. Gradient operates in a short range but is fast, while sweeper has a longer range but is much slower. (This is quantified in section 4.) The adaptive algorithm combines the benefits of these two algorithms, along with random walk, by trying each one in sequence and choosing the best one for a given situation. It first tries gradient, which would work well if the food is near enough to use it. If not, it switches to sweeper to get food further away. If it still doesn't find the food, it switches to the last resort – random walk. Switches between these three algorithms are made in a distributed manner at the colony level. To accomplish this, a third gradient is included.

3.3.1 Local Description

Robots begin with an algorithm very similar to the gradient algorithm above. They are split between walker and beacon robots as before, but they maintain three gradients as opposed to two. The third one measures how far each beacon is from any walker robot. This requires all walker robots to transmit a single bit of information indicating their presence and identity as a walker. The beacons then transmit a 1 for the `walkerGradient` if they can see a walker, and $min + 1$ if they can not see a walker. Other than this, they execute the gradient algorithm exactly as described above.

To implement adaptive foraging, robots need to explicitly detect when to change algorithms. If a robot does not see any (non-zero) `walkerGradient` for several time steps in a row, it will completely switch algorithms from gradient to sweeper. Thereafter, if the line of robots has swept the world twice and still has not found food (as evaluated by the pullers), the pullers send a signal through the beacon network causing every robot to again switch algorithms to random walk.

3.3.2 Global Behavior

The swarm will begin executing the gradient algorithm. There are many walkers at the beginning of the execution, so the values for `walkerGradient` are all fairly low. If the swarm finds the food, it returns it as usual, and the `walkerGradient` remains irrelevant. If, however, the swarm expands to the point that all robots have become beacons and the swarm has still not found the food, then there will be no walkers left. When the last walker becomes a beacon, suddenly no beacon anywhere in the swarm has information on which to broadcast a `walkerGradient`, so all `walkerGradient` values revert to 0. A short time later, they all decide nearly-simultaneously to switch algorithms and begin the sweeper algorithm. From this point on, the swarm proceeds as normal as if it had just begun the sweeper algorithm, pulling out a line of robots and sweeping the world. Once this line has swept around the world twice without finding food, the swarm switches to random walk, which is the only option left. Random walk does not involve coordination, which relieves the robots of the requirement of staying near each other. This is the only way to get food so far away.

4 Performance

These algorithms were tested in a continuous-world multi-agent simulator (screenshot in figure 1c). An unlimited food source was placed at varying distances from the nest, with a swarm of 20 robots trying to find and retrieve it (as diagrammed in figure 4a). Since we are focusing on algorithm switching and the environmental impact, we chose to experiment with a fixed number of robots. In the future, we will study scalability of the individual algorithms more closely.

We assessed the performance of the algorithms using three simple metrics: (1) whether or not the swarm found the food, (2) how quickly it found the

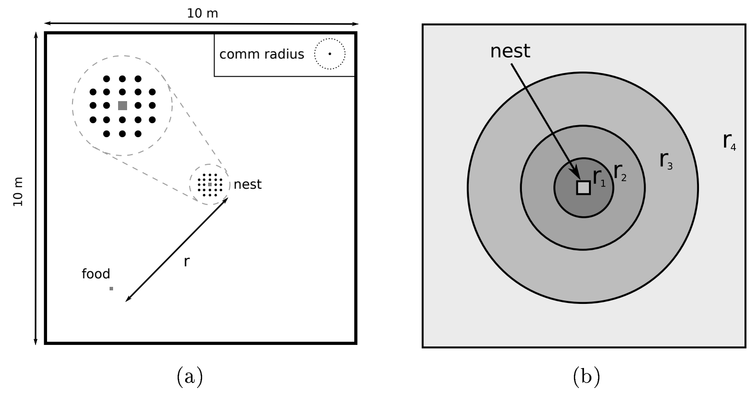


Fig. 4: Test setup is shown in figure 4a, drawn to scale. The parameter r ranges from 1m to 4m. Figure 4b shows the regions described in section 4.

food, and (3) the rate at which it returned the food to the nest. Each data point represents an average of 100 runs.

Region-Based Analysis Based on performance (figure 5), we can see that the world can be divided into four distinct regions, diagrammed in figure 4b.

region	description
r_1	any algorithm works
r_2	coordination needed, gradient works well
r_3	too far for gradient, sweeper works well
r_4	too far for sweeper, only random walk works

If the food is inside r_1, it is so close to the nest that any algorithm will find it (figure 5a), find it quickly (figure 5b), and return it quickly (figure 5c).

r_2 is the boundary inside which the gradient method works well. It finds the food, finds it quickly, and returns it quickly. Outside of r_2, gradient works poorly. As the robots expand and form a beacon field, eventually the swarm will expand to its maximum size and there will be no walkers left to continue the expansion. If the food is beyond this critical radius (r_2), there is no way for the gradient method to get it. The sweeper algorithm is also capable of finding the food inside r_2, but as seen in figure 5b, takes much more time to do it. The adaptive algorithm is able to choose the gradient method in this region, finding food quickly with a high success rate.

In r_3, the gradient algorithm is useless, and the sweeper algorithm performs well, forming a line and sweeping the world out to approximately r_3, although this boundary is less well defined. It finds the food but takes a long time to do it. In this region, the adaptive algorithm correctly selects sweeper.

Outside r_3, even the sweeper algorithm fails because the line of robots can not reach that far. In r_4, the adaptive algorithm switches to random walk. This works poorly ($\sim 20\%$ success rate, slow to locate and return food), but beyond about 3m, it is the only method capable of finding any food at all.

In every region, the adaptive algorithm is able to choose the most appropriate foraging method. In r_2 it runs the gradient method, in r_3 it runs the sweeper method, and in r_4 it runs random walk.

Overall Assessment As an overall assessment of each algorithm, we can place food in an unknown random location, and measure the performance. In the table below, "r_1, r_2, or r_3" indicates that the food is placed randomly anywhere in those three regions, and "whole world" indicates that the food is randomly placed anywhere. The numbers are averages over all placements. For example, if the food is randomly placed anywhere and the sweeper algorithm is running, the swarm can be expected to find it 32% of the time, after an average of 7300 time steps with a standard deviation of 3500 time steps. (One time step roughly corresponds to one second.)

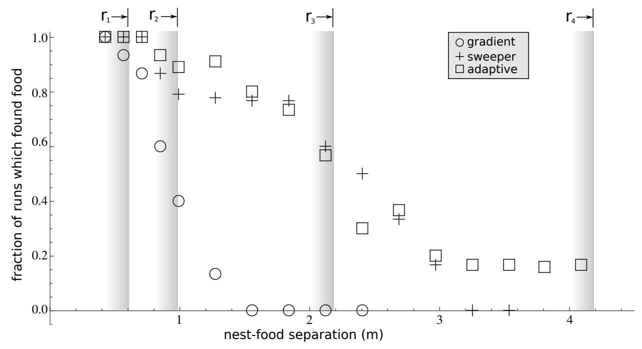
	r_1, r_2, or r_3		whole world	
algorithm	success rate	time food found	success rate	time food found
gradient	31%	69 ± 34	20%	69 ± 34
sweeper	82%	6500 ± 3900	32%	7300 ± 3500
adaptive	86%	5500 ± 3700	46%	9200 ± 6000

In r_1, r_2, or r_3, the adaptive algorithm finds the food almost as often as the sweeper algorithm, but does so faster. This is because it is able to take advantage of the speed of the gradient method when the food is nearby. When very distant food locations are included (whole world), all algorithms suffer lower success, but the adaptive algorithm is able to use random walk to at least achieve some success in a very long time.

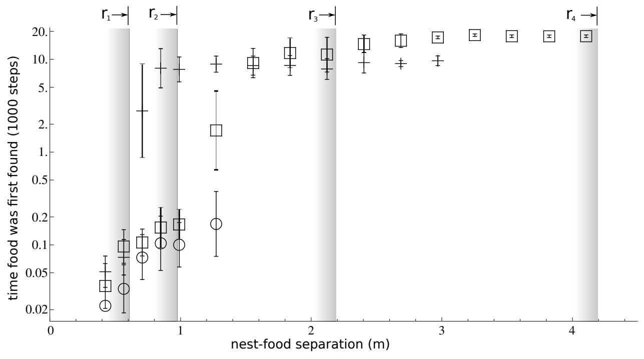
5 Algorithm Switching Generalizations

Algorithms other than those presented in this paper could potentially be combined into a single adaptive algorithm using the same method. The critical requirement is the connectedness of the communication network. This section will discuss several generalizations we can draw about colony-level algorithm switching, beyond the specific cases discussed in this paper.

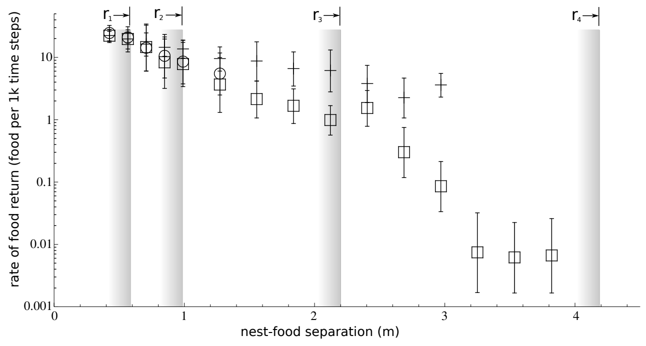
There are two kinds of algorithm switches: individual switches and colony switches (see figure 2), with two main differences between the types. First, individual switches are made by a particular robot based on the information it can perceive, whereas colony-level switches require information which is distributed throughout the environment and not directly perceivable by every robot making the decision. Second, colony-level switches must be nearly synchronized, whereas individual switches need not be explicitly coordinated.



(a)



(b)



(c)

Fig. 5: Figure 5a shows the success rate of each algorithm. Figure 5b shows the time at which food was first found. Figure 5c shows the rate at which food is returned to the nest once it is found. Each point represents an average of 100 runs, and the error bars indicate one standard deviation. Note the log scales in the second two plots.

To achieve a colony-level algorithm switch, information must be shared throughout the swarm, because each robot requires global-level knowledge in order to decide to switch algorithms. For example, when the adaptive algorithm switches from gradient to sweeper, each robot must be aware that no walkers are left, but no single robot is capable of perceiving this. This global information is detected and shared through the beacon network. Because global information is required for colony-level switches, maintaining the connectedness of the beacon network is critical for these switches.

We can distinguish three types of information:

information type	example from this paper
directly perceivable	sense a beacon ahead
directly perceivable by another robot	someone found food
only perceivable by swarm as a whole	swarm has expanded to max size

Individual switches can be made solely based on information of the first type. Colony switches require the second and third types, which requires a beacon network. There is no robot in the swarm with a sensor capable of detecting the third type of information; it can only be detected by the swarm as a whole through cooperation. A connected network is critical for detecting and transmitting this information.

6 Conclusion

We have presented two distributed foraging algorithms which perform best under different food locations, and a third method in which the swarm as a whole can choose the best algorithm for the given situation. The gradient algorithm can return nearby food quickly, and the sweeper algorithm can find food further away but is much slower. The adaptive algorithm uses the gradient, sweeper, and random walk methods, detecting in a distributed manner if one has failed and switching to the next. For food in any region of the world, the adaptive method is able to choose the most appropriate algorithm. Colony–level algorithm switching requires communication, but can combine benefits of multiple algorithms and improve overall performance.

There are several possible improvements and expansions planed for the future, both in hardware and software. Although these algorithms are designed to be scalable, scalability will be tested experimentally. Second, we will consider methods by which swarms could switch algorithms depending on dynamic environments (which could require them to switch back to previously tried algorithms). Finally, we will conduct a hardware study on the effect of sensor / communication capability on algorithm possibilities and performance. This study will use the E-Puck robots and IR communication rings described earlier.

References

1. A. Gutierrez et. al. Open E-Puck Range & Bearing Miniaturized Board for Local Communication in Swarm Robotics. *ICRA*, 2009.
2. Ronald Arkin. *Behavior-Based Robotics.* MIT Press, 1998.
3. Tucker Balch and Maria Hybinette. Social potentials for scalable multi-robot formations. pages 73–80, 2000.
4. Chris Jones and Maja Mataric. Behavior-Based Coordination in Multi-Robot Systems. *Autonomous Mobile Robots: Sensing, Control, Decision-Making, Applications.*
5. D. Payton, M. Daily, R. Estkowski, M. Howard, and C. Lee. Pheromone Robotics. *Autonomous Robots*, 11(3):319–324, 2001.
6. E. Barth. A dynamic programming approach to robotic swarm navigation using relay markers. *In Proceedings of the 2003 American Control Conference*, 6:5264–5269, 2003.
7. A. Howard, M. Mataric, and G. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. *Sixth Int. Symposium on Distributed Autonomous Robotics Systems*, pages 299–308, 2002.
8. N. Hoff III, A. Sagoff, R. Wood, and R. Nagpal. Two foraging algorithms for robot swarms using only local communication, 2010. `ftp://ftp.deas.harvard.edu/techreports/tr-2010.html`.
9. J. Svennebring and S. Koenig. Building Terrain-Covering Ant Robots. *Autonomous Robots*, 16(3):313–332, 2004.
10. K. O'Hara, D. Walker, and T. Balch. The GNATs Low-cost Embedded Networks for Supporting Mobile Robots. pages 277–282, 2005.
11. Mamei et al. Spreading Pheromones in Everyday Environments via RFID Technologies. *2nd IEEE Symposium on Swarm Intelligence*, 2005.
12. J. McLurkin. Measuring the accuracy of distributed algorithms on Multi-Robot systems with dynamic network topologies. *9th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2008.
13. J. McLurkin and D. Yamins. Dynamic task assignment in robot swarms. *Proceedings of Robotics: Science and Systems, June*, 8, 2005.
14. James McLurkin. *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots.* S.M. thesis, MIT, 2004.
15. F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, Floreano D., and A. Martinoli. The e-puck, a robot designed for education in engineering. *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 1(1):59–65, 2009.
16. S. Nouyan, R. Groß, M. Bonani, F. Mondada, and M. Dorigo. Teamwork in self–organized robot colonies. *IEEE Transactions on Evolutionary Computation*, 13(4):695–711, 2009.
17. C.A.C. Parker and H. Zhang. Collective unary decision-making by decentralized multiple-robot systems applied to the task-sequencing problem. *Swarm Intelligence*, 2009.
18. R. Vaughan, K. Stoy, G. Sukhatme, and M. Mataric. LOST: Localization-space trails for robot teams. *IEEE Trans. on Robotics and Automation*, 18(5):796–812, 2002.
19. T. Schmickl and K. Crailsheim. Trophallaxis within a robot swarm: Bio-inspired communication among robots in a swarm. *Autonomous Robots*, 25:171–188, 2008.
20. W. Spears, D. Spears, J. Hamann, and R. Heil. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, 17(2–3):137–162, 2004.
21. K. Sugawara, T. Kazama, and T. Watanabe. Foraging behavior of interacting robots with virtual pheromone. *IROS 2004*, 3:3074–3079, 2004.
22. S. Yoon, O. Soysal, M. Demirbas, and C. Qiao. Coordinated locomotion of mobile sensor networks. *5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 126–134, 2008.