Two Foraging Algorithms for Robot Swarms Using Only Local Communication

Nicholas R. Hoff III, Amelia Sagoff, Robert J. Wood, Radhika Nagpal

***Abstract*— Large collections of robots have the potential to perform tasks collectively using distributed control algorithms. These algorithms require communication between robots to allow the robots to coordinate their behavior and act as a collective. In this paper we describe two algorithms which allow coordination between robots, but do not require chemical pheromones or physical environment marking. Instead, these algorithms rely on simple, local, low–bandwidth, direct communication between robots. We describe the algorithms and measure their performance in worlds with and without obstacles.**

I. Introduction

As the cost of robotic hardware has come down and availability has gone up, there has been growing interest in robotic systems which are composed of multiple simple robots rather than one highly-capable robot. This tradeoff reduces the design and hardware complexity of the robots and removes single point failures, but adds complexity in algorithm design. The challenge is to program a swarm of simple robots, with minimal communication and individual capability, to perform a useful task as a collective.

Nature has solved this problem many times over. Schools of fish swim in unison, and are able to execute large scale collective maneuvers to avoid a predator. Termite colonies build large and very complex nests (complete with thermal regulation). Ants collectively search a very large area and are capable of returning food to the nest. In each of these examples, there was no central leader with all the information making decisions for each individual. Leaderlessness is a central aspect of distributed swarm algorithms.

In this work, we take inspiration from natural swarm behaviors to build algorithms for robot swarms. Ant colony foraging is our primary example. Ants use pheromones to mark trails in the environment, which allows them to efficiently communicate the location of food sources and collectively bring food back to the nest [1]. This work focuses on finding a way for simple robots with simple sensing capabilities to achieve the same task, without having to implement physical pheromones or physical environment marking.

In our application, we assume there is a swarm of robots which has simple sensing capability: each robot can communicate directionally with other robots within a short-range. The task is for the group of robots to search the environment for an object of interest ("food") and then return the food to the base ("nest"). The robots do not know the location of the food apriori, nor do they have GPS/odometry capabilities, but the efficiency of the group can dramatically be improved through coordination. We present two algorithms, inspired by ant colony behavior, that use a simple concept: each robot can dynamically take on one of two roles, being a stationary environment beacon (like a pheromone) or being a wandering robot. The two algorithms (termed "virtual pheromone" and "cardinality") differ in when the beacon role is chosen and what information the beacon emits.

We show that both algorithms are able to effectively construct paths between food and nest, and adapt to environments with static obstacles. We compare the performance of the algorithms to the performance of robots without coordination and robots with explicit global position and knowledge of the food; these comparisons represent two extremes in hardware complexity of the robots. We show that the virtual pheromone and cardinality algorithms dramatically increase efficiency over no coordination, and performance scales directly as the number of robots increases until congestion impacts all algorithms. Our work shows that for the task of foraging, we can take advantage of the collective efficiency that ant colonies achieve, while relying on hardware implementations that are feasible in simple and small robots.

In the future, when these algorithms can be implemented on large collections of small robots, they will enable applications such as distributed cleanup of environmental toxins and automated repair. For example, a swarm of small robots could search a building for defects, then recruit other robots to bring tools and materials to the site of the damage.

A. Related Work

Multiple researchers have studied foraging and other pheromone-mediated coordination, and have reduced this behavior to algorithms [2], [3], [4], [5]. Pheromone-trail based algorithms sometimes have the ability to dynamically improve their path [6] and can adapt to changing terrain [7].

Ant-inspired foraging has been implemented in robots by various groups. One of the chief difficulties is in implementing the pheromone itself, or some way for the robots to interact. There have been many approaches to this problem:

- Physical marks. Robots can physically mark their trails in a variety of ways, such as leaving alcohol [3], heat [8], odor [9], visual marks [10], or RFID tags [11].
- Use existing communication channels. In the work of Vaughan et al., robots maintain an internal pheromone model with trails of waypoints, and share it with other robots over a wireless network [12], [13], [14].
- Virtual pheromones. In [15], authors use direct infrared-based communication between robots to transmit a kind

of virtual pheromone. They study the use of these signals to create world–embedded computation and world–embeded displays. It is assumed that the robots that receive the pheromone can measure the intensity of the IR reception to estimate their distance from the transmitter.

- Pre-deployed sensor network. The GNATS project [16] uses robots which operate in an environment which is pre-populated with a regular grid of beacons on which information can be stored. They have had success in finding close-to-optimal paths from one place in the environment to another, even in the presence of obstacles.
- Deployable beacons. Some groups have explored the idea of having each robot be able to deploy beacons as it moves through the environment. The beacons can be movable or non-movable [17], and contain pheromone-like information.
- Robot chains. In [18], [19], the robots form chains and attempt to remain in close proximity with each other, through communication or even by physically gripping the next robot in the chain.

There are several shortcomings of these approaches which we aim to address. Making permanent physical marks in the environment is generally not acceptable, and temporary or decayable marks are difficult to physically implement. Relying on predeployed sensor networks is highly restrictive and prevents operation in a new or unexplored environment. Deploying beacons is a good method but requires building robots which can carry many beacons and can also intelligently recover previously laid beacons. Instead, we use the robots themselves as beacons.

Of the work cited above, our work is most similar to the virtual pheromone approach, except that our robots do not have different behaviors depending on a direct measurement of their distance from each other. This simplifies the communication hardware because distance measurement is not required. Secondly, we show that multiple algorithms can be used with this communication model to achieve coordination and approximate the collective benefit of pheromones.

B. Robot Model and Simulator

Robot Model: We assume a simple non–holonomic robot that moves and turns in continuous space. Each robot has sensors for nest, food, and obstacle detection in direct proximity to the robot. Each robot can also communicate with nearby robots using a simple IR ring architecture. The robots have omnidirectional transmission, and directional reception. This means that when a robot receives a transmission, it knows roughly which direction the transmission came from (see Fig. 1).

Simulator: In order to test the algorithms, we developed a simulator based on Microsoft(R) Robotics Studio (MRDS). The simulator models a continuous world in which the robots, food, nest, and obstacles exist, and each object occupies some physical extent. Realistic physics and visualization is provided by MRDS. A snapshot from the simulator is shown in Fig. 2. We chose this simulation environment over a gridded world environment so that we would face the real–world problems of collisions and congestion. Our simulator closely matches a model of the Khperea II robots which we plan to use in the future for hardware verification of the algorithm.

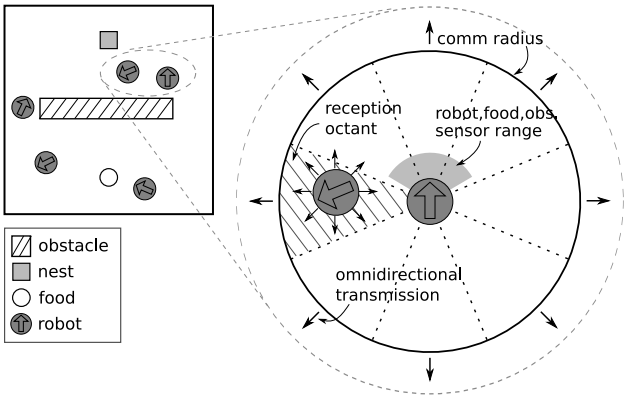

Fig. 1. This figure shows the communication and sensing structure of the robots. The figure in the upper left shows an example world setup. The blowup details a hypothetical communication situation. When the robot on the left transmits, the robot in the center can receive the transmission, and also knows that the transmission came from the left-facing octant. The light gray range in front of the central robot indicates the approximate area in which the nest, food, and obstacle sensors are sensitive.

In the remaining sections of this paper, we will describe two ant-inspired algorithms capable of foraging without physical pheromones, then present performance results measured in simulated worlds with and without obstacles, and finally conclude with a discussion of future algorithm work and application to small autonomous robots.

II. Algorithm Description

In this section, we will describe two algorithms, called the virtual pheromone (`VP`) algorithm and the `cardinality` algorithm.

Ant colony foraging: Both algorithms are based on the foraging behavior of ants. Ants mark trails leading from the nest to food and back by depositing a chemical pheromone on the ground. Each ant can both deposit and detect this chemical, and each ant uses the distribution of pheromone in its immediate vicinity to decide where to move. When an ant is carrying food, it lays pheromone as it searches for the nest, and other ants follow this trail outbound to the food. This distributed leaderless pheromone algorithm is the starting point for the `VP` algorithm, but several important changes have been made.

Virtual Pheromone Algorithm: The first change has to do with the way the robots return to the nest once they have found the food. The pheromone is laid by individuals who are carrying food, so this trail leads to the food, but the individual still needs a way to return to the nest. Ants have various methods to do this, including visual landmarks and odometry (remembering how far and in what direction the nest is). Because of our focus on miniaturizability and hardware simplicity, neither of these approaches is attractive

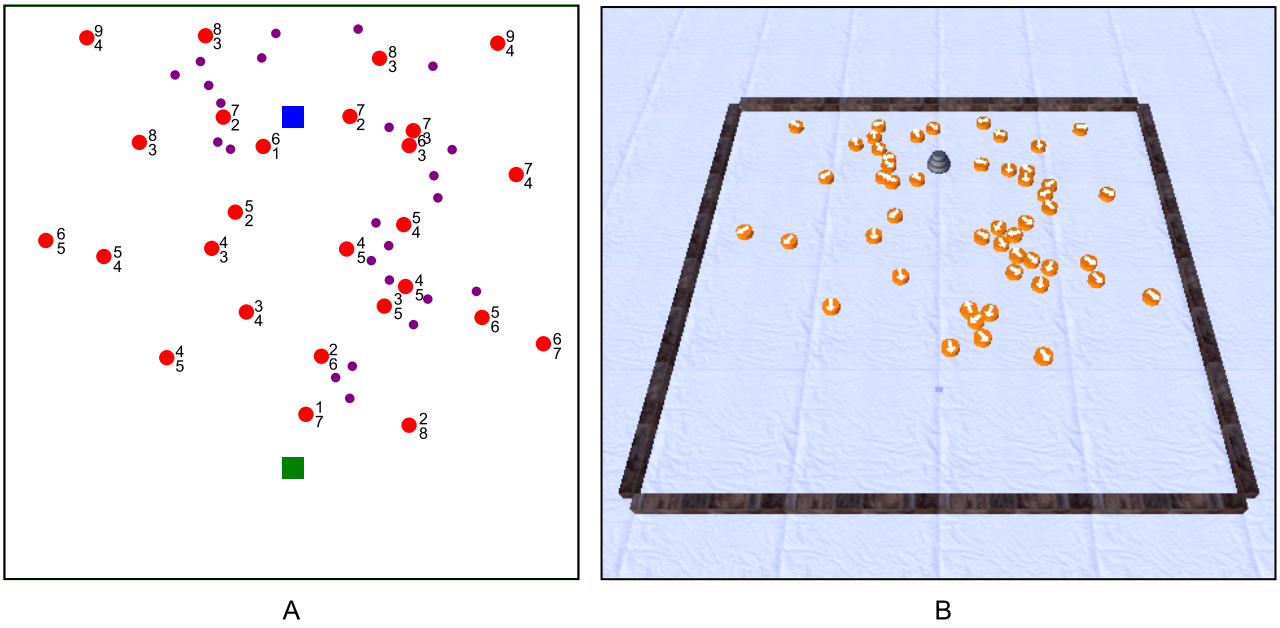

Fig. 2. This shows a screenshot of the simulator. Window A shows a dot for each robot, with beacons shown as larger dots and cardinalities drawn to the right of beacons. The square near the top is the nest and the square near the bottom is the food. Window B shows the actual MRDS physics simulation and visualization.

– vision requires significant computation and memory to interpret the image, and odometry requires high accuracy encoders. As an alternative method to navigate back to the nest, two distinguishable pheromones are used instead of one. One pheromone leads to the food and a second leads to the nest.

A second change from the biological algorithm is that virtual pheromones are substituted for real ones. Real chemical pheromones are difficult to implement in hardware because a deposition system and a detection system must both be built, and these systems must be simple, inexpensive, robust, and light. Instead of implementing chemical pheromones, simple local direct communication between robots is used to transmit a virtual pheromone value.

During the execution of the algorithm (see pseudocode in Fig. 4), some robots will decide to stop their normal search behavior and become 'pheromone robots', also known as beacons, which means they stop moving and act as locations on which virtual pheromone can be stored. Walker robots can read the pheromone level by receiving a transmission from the pheromone robot, and they can lay virtual pheromone by transmitting to the pheromone robot. So, if there were a network of pheromone robots, the walker robots could use the distribution of virtual pheromone they can sense to decide how to move. A diagram of an example situation of the `VP` algorithm is shown in Fig. 3a.

Cardinality Algorithm: The `cardinality` algorithm is similar to the `VP` algorithm in that robots can decide to act as either beacons or walkers — beacons transmit values, and walkers use those values to decide where to move. The main difference is that instead of the values that the beacons store and transmit being real-valued floating-point numbers, they are integers (called "cardinalities"). The first beacon standing

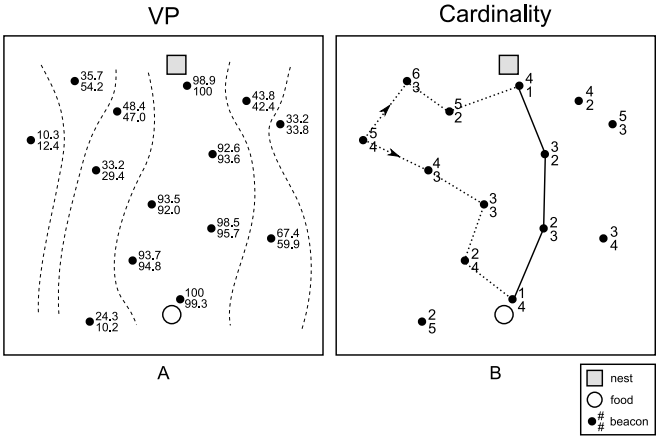

Fig. 3. This diagram shows example situations in both the `VP` and `cardinality` algorithms. Each black dot represents a robot which has decided to become a beacon (walkers are not shown). The numbers to the right of each beacon indicate the virtual pheromone levels (fig a) or cardinalities (fig b) – food pheromone/cardinality on the top and nest pheromone/cardinality on the bottom. In fig a, the dotted lines indicate approximately level curves in both pheromone values. In fig b, the solid line indicates the shortest path between the nest and the food. As an example, the dotted lines indicate paths that a robot near the $\frac{5}{4}$ beacon on the left could take to get either to the nest or to the food.

next to the nest transmits a 1, then the next beacon a little further out would be able to hear the 1 and so would transmit a 2. In general, each beacon transmits the minimum of all the other beacons it can hear, plus one. In this way, the cardinality of each beacon can be interpreted as the number of beacons between that beacon and the nest. Furthermore, a walker robot can use these cardinalities to find a path to the nest by always moving to the lowest cardinality it can hear. In a similar manner to the `VP` algorithm, each beacon actually

```
start:
    if beacon, goto beacon
    if walker, goto walker

beacon (VP alg):
hear 3 or more beacons?
    with prob p, become walker
    END
else
    both pheromones < threshold?
        become walker
        END
    else
        decay pheromones
        END

beacon (cardinality alg):
hear 3 or more beacons?
    with prob p, become walker
    END
else
    both pheromones < threshold?
        become walker
        END
    else
        decay pheromones
        END

walker (either alg):
hear 2 or more beacons?
    carrying food?
        goto NEST SEARCH
    else
        goto FOOD SEARCH
else
    become beacon
    END
```

```
FOOD SEARCH:
try to pick up food
am I doing AVOID?
    sense obstacle?
        turn t degrees
        END
    else
        move forward
        lay inPheromone (VP only)
        END
am I doing EXPLORE?
    stepCounter >n?
        stepCounter = 0
        end behavior
        END
    else
        try to pick up food
        sense obstacle?
            turn t degrees
            END
        else
            move forward
            lay inPheromone (VP only)
            stepCounter ++
            END
neither AVOID nor EXPLORE:
    should I explore?
        set stepCounter = 0
        execute EXPLORE
    else
        turn to strongest outPheromone
        or lowest foodCardinality
        sense obstacle?
            execute AVOID
        else
            move forward
            lay inPheromone (VP only)
            END
```

Fig. 4. This figure shows combined pseudocode for both the `VP` and `cardinality` algorithms. Robots begin execution at "start" at the top and then proceed according to which algorithm is being run and the role (walker or beacon) of the robot. The normal food search part of the algorithm is shown; the nest search part is nearly identical, the biggest difference being that robots lay *outPheromone* instead of *inPheromone*. Some of the parameters of the algorithm are shown in this pseudocode. Probability `p` is currently set to 30%, angle `t` is 45°, and step counter `n` is 4.

transmits two cardinalities — one indicating how many beacons away from the nest it is, and the other indicating the number of beacons away from the food. Psudeocode for the `cardinality` algorithm is shown in Fig. 4.

An snapshot of the `cardinality` algorithm is shown in Fig. 3b. The beacon standing directly next to the nest is transmitting a 1 for its nest cardinality, and the beacon standing next to the food is transmitting a 1 for its food cardinality. All other beacons are updating their cardinalities accordingly. Now, by listening to the cardinalities and always moving to the smallest one, the walker robots can walk to the nest or to the food from wherever they are.

One significant difference to notice between the `VP` and `cardinality` algorithms is that `VP` requires the walkers to transmit to the beacons every time they want to lay a virtual pheromone. The `cardinality` algorithm, however, only requires beacons to transmit – the walkers only need to receive.

III. Test Parameters and Metrics

In this section, we will discuss the performance of each algorithm in an obstacle–free world.

A. World and Test Parameters

The world setup, including positions of nest and food, is shown in Fig. 8a. (See section V for results in worlds B, C, and D.)

There are several test-related parameters which must be chosen, such as world size, food placement, run time, and communication radius.

- **world size —** World size can be non–dimensionalized by dividing by robot body length. The robots we will ultimately use to test this algorithm have a body length of about 8 cm. A square world in which each side is 50 times the body length would be 4m×4m. Real ants have a body length around 8 mm, but operate in roughly the same size areas, so they would have a world size to body length ratio of around 500. In this study, we use square worlds for their symmetry and simplicity. In the future, we may experiment with non–square worlds, or worlds with no boundaries at all. (The presence of boundaries makes the problem simpler by focusing the robots on the area of interest; in a world with no boundaries, the robots could get permanently lost.)
- **robot density —** The number of robots in the swarm divided by the total world size gives the average robot density. In this paper, we will primarily explore the effect of this parameter, presenting results as a function of both absolute number of robots and robot density.
- **nest–food separation —** This can be non–dimensionalized by dividing by world size. In this paper, we separate the nest and the food by a distance equal to 60% of the world size. This gives a large enough separation that the problem is hard, but not so large that both the food and nest are next to the world boundaries. In these experiments, the food piles never run out of food.
- **run time —** To non–dimensionalize the run time, we will use the number of direct nest–food traversals that are theoretically possible during the run (assuming the robots could travel straight to the food and back). We will run each simulation long enough for a robot to traverse the nest-food distance 25 times. Assuming the robots can move approximately one body length per time step, that would yield a run time of 750 time steps.
- **communication radius —** The non–dimensional parameter here is communication radius divided by body length. With the hardware we intend to use to implement the local communication, a ratio of 10 seems conservative. So in this work, we assume a communication radius equal to 10 times the body length, or 80 cm.

In this paper, we will focus on the effect of world size and robot density. A study of the sensitivity of the results to variations in the other parameters is too large to fit here, and will be published separately.

B. Metrics

To measure the performance of the VP and cardinality algorithms, three metrics will be used:

- **total food returned —** The total amount of food that has been returned to the nest by the entire swarm after 750 time steps.
- **rate of food return —** The average rate at which the swarm returns food to the nest, measured in food units per time step.
- **cost —** The total energy consumed by the swarm. For the purposes of calculating this cost metric, every communication action will incur a cost of 1 and every movement action will incur a cost of 100. In this way, cost is roughly analogous to battery usage. The total cost is the sum of all the costs of all the robots for the duration of the test.

Both algorithms will be measured using each of these metrics.

The performance of the algorithms, as measured by these metrics, will be compared against two comparison algorithms. The first is a randomWalk algorithm, in which each robot decides randomly whether to turn a random amount, or to move forward. The second comparison algorithm is called the foodGPS algorithm. In this algorithm, each robot has perfect knowledge of the locations of the food and the nest, so it is capable of turning directly to the food source and moving in that direction, then turning directly to the nest and returning. The foodGPS algorithm will not always yield perfect or optimal performance because the robots may still be blocked by obstacles or other robots (of which they have no knowledge). The robots do not take optimal paths around obstacles, they just know the final goal location and attempt to go straight there, avoiding obstacles when they encounter them.

randomWalk was chosen as a lower comparison algorithm because it has the smallest requirements for communication and sensor hardware. It represents a minimum use of coordination among robots – none. foodGPS, on the other hand, has very large hardware requirements. Every robot must know the position of the food and the nest regardless of other robots, obstacles, or distance. These two algorithms represent extremes of hardware requirements. Our aim is to develop algorithms with significantly increased performance with only moderately increase hardware requirements.

IV. Results in an Uncluttered World

Here we show results of the VP and cardinality algorithms over the course of a single run and compared across multiple runs, operating in obstacle–free worlds.

A. Performance of Algorithms over Time

At each time step during each run, we measured the total food that had been returned to the nest, the fraction of non-beacon robots that were carrying food, and the total number of beacons. Data from example runs of the VP and cardinality algorithms is shown in Fig. 5. These examples

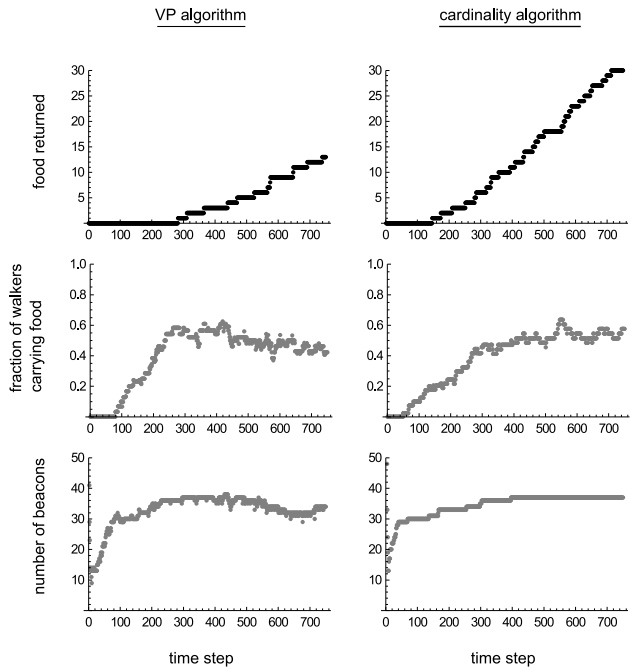

Fig. 5. This shows example runs of each algorithm in the world configuration of Fig. 8A. Each run was performed with 70 robots.

were run in a 4m×4m world in configuration A (see Fig. 8) with 70 robots. In the first 50-100 time steps, the number of beacons rapidly increases to a steady value as the ad-hoc network of beacons is deployed. Next, the fraction of food-carrying ants becomes non-zero, as the first robots are finding and picking up the food. Some time after that (about 100 time steps in these examples), the first robots are able to return food to the nest. For the rest of the run, the amount of food returned increases and the number of walker robots carrying food approaches roughly 50%. It makes sense that half of the walkers would be carrying food in steady state, because approximately half would be on their way to the food and half would be returning.

The major difference between the two algorithms is that cardinality returns food faster that VP, and returns more. VP requires time for the trail between the food and the nest to be built and reinforced after it is found, whereas no such time is required in cardinality. In cardinality, once any robot has found the food, all robots know a path to get there, and then once they pick up food, they have a path to get back to the nest. No reinforcement is required.

B. Performance and Comparison of Algorithms

Each of these four algorithms (randomWalk, VP, cardinality and foodGPS) were run for 750 simulation time steps, and the total amount of food collected at the end of the run was recorded. The results are shown in Fig. 6a.

randomWalk never returns any food. Evidently in worlds this big, the probability of a robot randomly finding the food and then randomly finding the nest is so small that it never happened in any of these tests (a brief analysis shows that this is expected, but there is insufficient space to include it

here). With this set of test parameters, foodGPS returns a large amount of food. cardinality and VP are in the middle, and cardinality outperforms VP. In these tests, the food pile can never run out of food, so there is food available for each algorithm for the whole duration of each run. The algorithms can never be 'finished', so foodGPS is used as an optimal performance.

From this data, the effect of congestion can also be seen in VP and cardinality. As the number of robots increases, the performance also increases because there are more robots available to transport food. However, when there are more than about 100 robots, it is so crowded that they have trouble moving and the performance again decreases. See section IV-C for a more detailed discussion of congestion.

The fact that VP and cardinality perform better than randomWalk at all shows that some coordination is being achieved between the robots using the virtual pheromones and cardinalities. These algorithms sacrifice some robots to be immobile beacons, and these beacons are no longer directly picking up and dropping of food. It could be the case that this sacrifice outweighs the benefit derived from the virtual pheromone or cardinality. If that were so, using the VP or cardinality algorithms at all would be harmful and randomWalk would be better. In fact, they outperform randomWalk, which shows that the sacrifice of some robots to be beacons confers a net benefit on the algorithm. Furthermore, although each algorithm uses robots as beacons to achieve a coordination benefit, cardinality derives more benefit from these beacons. This benefit is reduced and eventually eliminated in both cases, however, at small numbers of robots because in that case, the robots are mostly used as beacons, so there are few walkers left to search for food.

The second metric is rate of food return, and the results for each algorithm are shown in Fig. 6b. The same trends can be seen here in return rate as could be seen in total returned – cardinality outperforms VP and there is a congestion effect in which too many robots harms the return rate.

Finally, the third metric is cost. We assume that each movement action incurs a cost of 100 and each transmission incurs a cost of 1. In this way, cost is similar to battery usage in the real robots, but with arbitrary units. When using this metric, we divide it by the total food returned by the swarm to arrive at a cost-per-food metric. The results are shown in Fig. 6c. For each number of robots, cardinality is capable of returning food at a lower cost than VP.

There also appears to be a trend in each algorithm in which smaller swarms are more efficient than larger swarms. This is likely related to congestion. Usually, when a robot encounters another robot in its path, it must do an avoidance maneuver, which requires several movement actions (a sequence of turns and forward movements). If it had not encountered the obstructing robot, it would have been able to simply move forward along its desired path with a single movement action. Because movement is expensive, and avoidance maneuvers require more movement than simple path following, larger swarms are less efficient because of the increased frequency of avoidance.

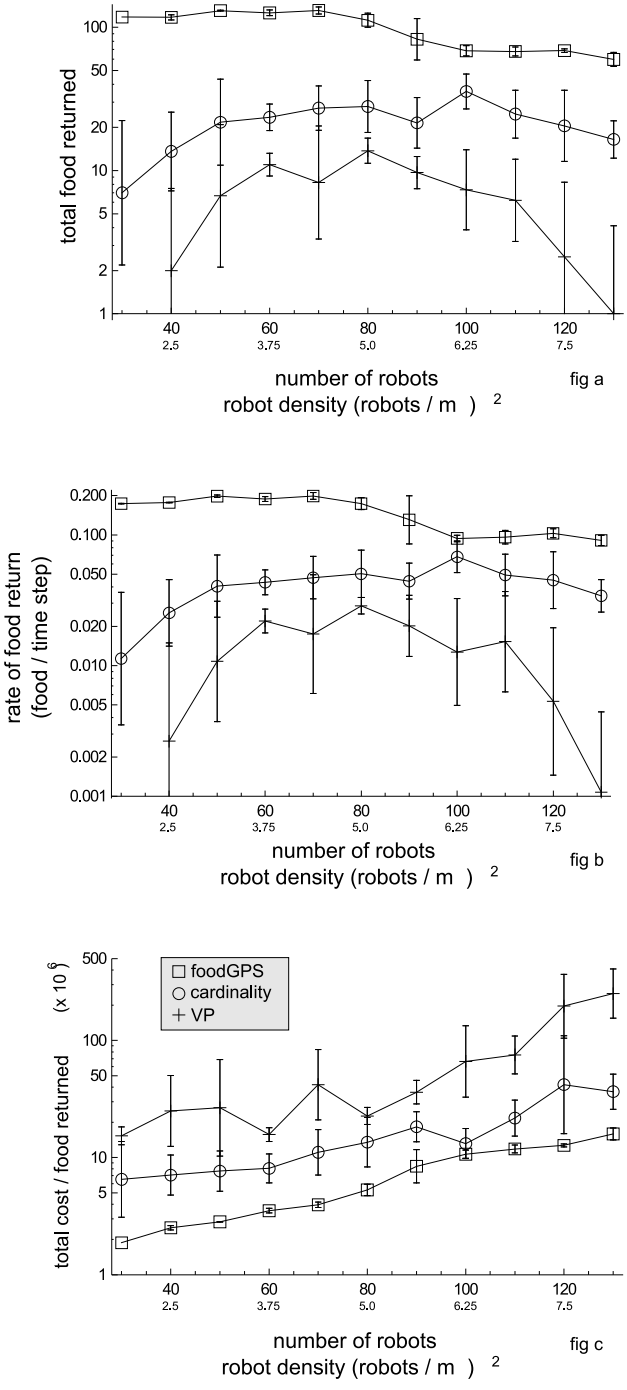

Fig. 6. These plots show the performance of foodGPS, cardinality, and VP as measured by the three metrics. The horizontal axes are marked both in absolute number of robots and in robot density. randomWalk never returns food, so it is not shown. Each point is an average of approximately 15 runs, with standard deviation error bars. Note the log scale on the vertical axes.

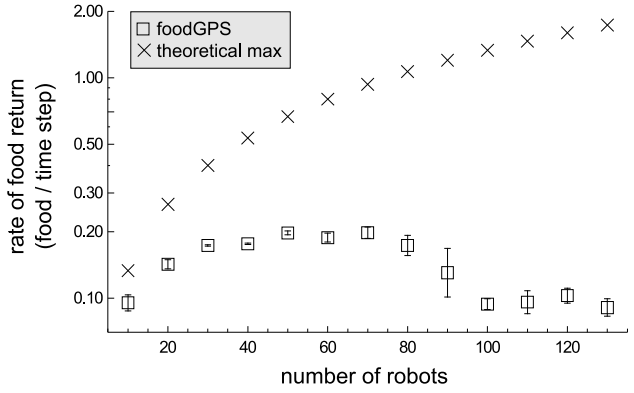

Fig. 7. Effect of congestion. When the number of robots is small it is not a problem, when the number of robots is large, it is the dominant effect, and there is an optimum in the middle. Each `foodGPS` point on this plot is an average of approximately 15 trial runs, with standard deviation error bars.

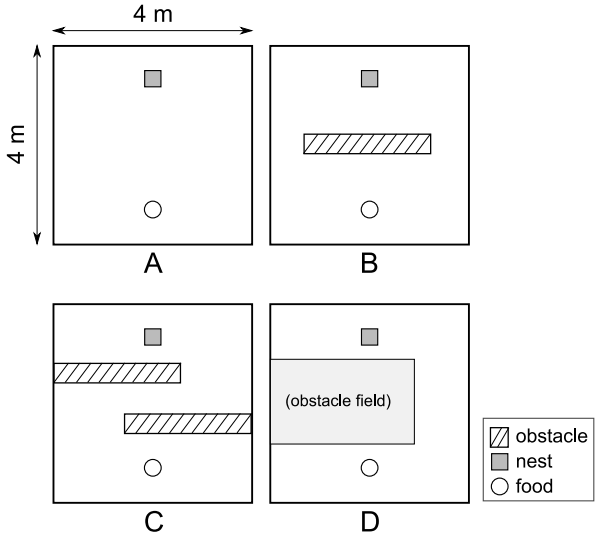

Fig. 8. There are four world configurations used in this paper. Results from configuration A are discussed in section IV, and results from configurations B, C, and D are discussed in section V. The obstacle field of configuration D is an area with many small obstacles obstructing simple straight line paths.

C. Congestion

Congestion occurs when many robots all want to occupy the same physical space, usually when they are crowding around the nest or the food, or on a heavily trafficked path. This can be disruptive, because even if the robots knew where to go, they could not go there and would have to take an alternate rout. A crowd of robots surrounding the nest, some of whom want to enter and some of whom want to exit, could be less efficient than a smaller number of robots without the congestion problem.

To measure the effect of congestion, a comparison is made between the performance of an algorithm under normal conditions, and its theoretical maximum performance if robots were able to occupy the same physical space. The algorithm is the same in both cases, and there is no sensor noise, so the difference in performance can be attributed to congestion. The algorithm used for this test is the `foodGPS` algorithm. All robots know the exact location of the food and nest, but not other robots. The robots will attempt to move in a straight–line path toward the food or nest, and must avoid collisions with each other.

In the absence of physical collisions, the maximum rate at which a swarm of robots could return food to the nest would be $\frac{fns}{2d}$, where f is the amount of food each robot can carry, n is the number of robots in the swarm, s is the distance each robot can travel in one time step, and d is the distance between the food and the nest. (d is doubled because the robots must travel to the food and back.) This rate is measured in food units per time step.

To test the effect of congestion, the `foodGPS` algorithm was run several times with different numbers of robots, and the rate at which food was returned to the nest was measured. The measured rate is then compared against the theoretical maximum, and the result is plotted in Fig. 7. At small numbers of robots, congestion seems to have little effect as the measured performance and the congestion-free maximum are close. This makes sense because there are so few robots in the world that they seldom run into each other and have to spend little time walking around each other. As the number of robots increases, congestion becomes more and more of a problem, and eventually, there are so many robots that congestion is the dominant effect and robots have so much trouble moving around that they have little ability to actually do the task.

V. Tests and Results in Worlds with Obstacles

A. Description of Obstacle Configurations

In this study, we used three different obstacle configurations, in addition to an obstacle-free world. All configurations are shown in Fig. 8. In configuration B, a simple obstacle blocks the most direct path from the nest to the food, such that the swarm must find a path around the obstacle. Configuration C has two obstacles that require the robots to take a more complex curving path to get to the food and back. Configuration D gives the swarm a choice between two paths: a short path that goes through an obstacle field, or a long but clear path.

`VP`, `cardinality`, and `foodGPS` do obstacle avoidance in nearly the same manner. Each walker robot has a direction it wants to go, determined either by virtual pheromones, cardinalities, or just knowing the correct direction (in the case of `foodGPS`). For each of these three algorithms, when a robot encounters an obstacle, it attempts to avoid it, usually by simply turning left and moving forward.

The performance of `foodGPS` can be used as a measure of the difficulty of each obstacle field. Taking the `foodGPS` result as a type of performance standard for each obstacle field, we can use it to normalize the performance of the other algorithms. So, for each algorithm and obstacle field, we will normalize the performance to the performance of `foodGPS` on that obstacle field. Results of these normalized performances are shown in Fig. 9. These algorithms do quite well compared to `foodGPS` considering their hardware

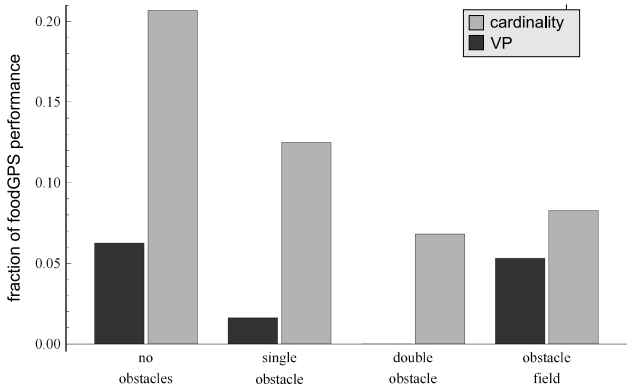

Fig. 9. This figure shows the amount of food returned by each algorithm in each obstacle field, normalized to the amount of food returned by `foodGPS` on that obstacle field. For example, in the single obstacle environment, `cardinality` returns about 12% of the food that `foodGPS` does.

requirements. `foodGPS` requires all robots to have global knowledge, and `cardinality`can achieve a fraction of that performance (about 5% - 20%) with no global knowledge or communication at all and only simple local communication.

`cardinality` has been observed to behave well in the case of changing obstacle configurations. When an obstacle is removed, for example, `cardinality` robots find a new shorter path. Space limitations prevent inclusion of results; they will be published in the future.

VI. Conclusion

This paper is concerned with leaderless distributed algorithms for swarms of robots. We have presented two algorithms, the Virtual Pheromone (`VP`) algorithm and the `cardinality` algorithm, which allow swarms of robots to forage for food in an unknown environment without central leadership. These algorithms do not require depositing real pheromones or beacons in the environment, rather the pheromone is an analogy for the way information is transmitted between robots.

`VP` and `cardinality` both performed better than `randomWalk`, which indicates that they were able to achieve some coordination benefit from the beacons. These algorithms were also generally returned food in the presence of obstacles. Congestion was observed to be a significant effect, in which increasing the number of robots in the increases improves the performance up to a point, but then decreases the performance as the robots have to spend so much time avoiding each other.

There are several additions to these algorithms that will be studied in future work. One obvious way that the cardinality algorithm could be improved is to reclaim useless beacons for use as walkers. After the swarm has found the food and robots are busy returning it, the beacons on the periphery of the field could abandon their role without negatively affecting the swarm. If there are only beacons along the food–nest path, however, then a robot who gets lost will have no way to return to the swarm. In the future, we will study ways to reclaim useless beacons without impairing the functionality of the swarm.

A paralel line of future development will focus on the hardware required to implement the communication model described in this paper. A preliminary prototype has been designed and built, and testing is under way. The ultimate goal is to interface the communication structure with the Khepera hadware. This project also fits into a larger micorobotics research effort. These algorithms are designed with a focus on minimal hardware partially because we envision them running on microrobotic insects. Such microrobotic hardware platforms are currently under development at the Harvard Microrobotics Lab [20].

References

[1] B. Hölldobler and E.O. Wilson. *The Ants*. Springer-Verlag, 1990.
[2] F. Adler and D. Gordon. Information collection and spread by networks of patrolling ants. *The American Naturalist*, 140(3):373–400, 1992.
[3] R. Sharpe and B. Webb. Simulated and situated models of chemical trail following in ants. *In Proceedings of the International Conference on Simulation of Adaptive Behavior*, pages 195–204, 1998.
[4] D. Lambrinos, R. Möller, R. Labhart, R. Pfeifer, and R. Wehner. A mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems*, 30:39–64, 2000.
[5] M. Nakamura and K. Kurumatani. Formation mechanism of pheromone pattern and control of foraging behavior in an ant colony model. *In C. G. Langton and K. Shimohara, editors, Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, pages 67–76, 1997.
[6] J. Deneubourg, S. Aron, S. Goss, and J. Pasteels. The self-organising exploratory pattern of the argentine ant. *Journal of Insect Behaviour*, 3(2):159–168, 1990.
[7] M. Resnick. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. MIT Press, 1994.
[8] R. Russell. Heat trails as short-lived navigational markers for mobile robots. *In Proceedings of the International Conference on Robotics and Automation*, pages 3534–3539, 1997.
[9] R. Russell. *Odour Sensing for Mobile Robots*. World Scientific, 1999.
[10] J. Svennebring and S. Koenig. Building Terrain-Covering Ant Robots. *Autonomous Robots*, 16(3):313–332, 2004.
[11] Mamei et al. Spreading Pheromones in Everyday Environments via RFID Technologies. *2nd IEEE Symposium on Swarm Intelligence*, 2005.
[12] R. T. Vaughan, K. Stoy, G. S. Sukhatme, and M. J. Mataric. Blazing a trail: insect-inspired resource transportation by a robot team. *Proceedings of the International Symposium on Distributed Autonomous Robot Systems*, 2000.
[13] R. T. Vaughan, K. Stoy, G. S. Sukhatme, and M. J. Mataric. Whistling in the dark: Cooperative trail following in uncertain localization space. *In C. Sierra, M. Gini, and J. S. Rosenschein, editors, Proceedings of the Fourth International Conference on Autonomous Agents*, pages 187–194, 2000.
[14] R. Vaughan, K. Stoy, G. Sukhatme, and M. Mataric. LOST: Localization-space trails for robot teams. *IEEE Transactions on Robotics and Automation*, 18(5):796–812, 2002.
[15] D. Payton, M. Daily, R. Estkowski, M. Howard, and C. Lee. Pheromone Robotics. *Autonomous Robots*, 11(3):319–324, 2001.
[16] K. OHara, D. Walker, and T. Balch. The GNATs Low-cost Embedded Networks for Supporting Mobile Robots. pages 277–282, 2005.
[17] 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.
[18] S. Goss and J. L. Deneubourg. Harvesting by a group of robots. In *First European Conference on Artificial Life*, pages 195–204. MIT Press, 1992.
[19] B. Werger and M. J. Mataric. Robotic food chains: Externalization of state and program for minimal-agent foraging. *P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, and S. W. Wilson, editors, Proceedings of the International Conference on Simulation of Adaptive Behavior*, pages 625–634, 1996.
[20] A. Baisch, R. Wood. Design and Fabrication of the Harvard Ambulatory Micro-Robot. *International Symposium on Robotics Research*, 2009.