1

Bio-inspired search strategies for robot swarms

James M. Hereford and Michael A. Siebold
Murray State University, Murray, KY
USA

1. Introduction

Our goal is as follows: build a suite/swarm of (very) small robots that can search a room for a "target". We envision that the robots will be about the size of a quarter dollar, or smaller, and have a sensor or sensors that "sniff" out the desired target. For example, the target could be a bomb and the robot sensors would be a chemical detectors that can distinguish the bomb from its surroundings. Or the target could be a radiation leak and the sensors would be radiation detectors. In each search scenario, we assume that the target gives off a diffuse residue that can be detected with a sensor.

It is not very efficient to have the suite of robots looking randomly around the room hoping to "get lucky" and find the target. There needs to be some way to coordinate the movements of the many robots. There needs to be an algorithm that can guide the robots toward promising regions to search while not getting distracted by local variations. The search algorithm must have the following constraints:

- The search algorithm should be distributed among the many robots. If the algorithm is located in one robot, then the system will fail if that robot fails.
- The search algorithm should be computationally simple. The processor on each bot is small, has limited memory, and there is a limited power source (a battery) so the processor needs to be power efficient. Therefore, the processor will be a simple processor.
- The algorithm needs to be scalable from one robot up to 10's, 100's, even 1000's of robots. The upper limit on the number of robots will be set by the communication links among the robots; there needs to be a way to share information among the robots without requiring lots of communication traffic.
- The search algorithm must allow for contiguous movement of the robots.

This chapter will describe two search strategies for robot swarms that are based on biological systems. The first search strategy is based on the flocking behavior of birds and fishes. This flocking behavior is the inspiration behind the Particle Swarm Optimization (PSO) algorithm that has been used in software for many types of optimization problems. In the PSO algorithm the potential solutions, called particles, "fly" through the problem space by following some simple rules. All of the particles have a fitness value based on the value or measurement at the particle's position and have velocities which direct the flight of the particles. The velocity of each particle is updated based on the particle's current velocity as well as the best fitness of any particle in the group.

We describe how the PSO algorithm can be embedded into a robot swarm by letting each bot's behavior be like a particle in the PSO. We call the algorithm the physically embedded PSO (pePSO). The bots swarm throughout the search space and take measurements. Over time, they cluster near the peak(s) or targets. We show through both 2D simulation results and robot hardware results that the pePSO effectively finds the targets with a minimum number of bot-bot communications.

The second search strategy is based on the trophallactic behavior of social insects. Trophallaxis is the exchange of fluid by direct mouth-to-mouth contact. This phenomenon is observed in ants, bees, wasps and even dogs and birds. In our trophallaxis-based algorithm, the bots do not actually exchange information but instead make sensor measurements when two or more bots/particles are "in contact". The bots remain stationary for a certain time that is proportional to the measurement value. Bots thus cluster in areas of the search space that have high fitness/measurement values.

This new trophallaxis-based search algorithm has several advantages over other swarm-based search techniques. First, no bot-bot communication is required. Thus, there is no concern with communication radius, protocol, or bandwidth. Second, the bots do not have to know their position. During the search, the bot/particle moves randomly except when it collides and stops, takes a measurement, and waits. At the end of the search, the cluster locations can be determined from a remote camera, special-purpose robot, or human canvassing.

This paper is organized as follows: section 2 gives background on the Particle Swarm Optimization algorithm and its use in robot swarms and section 3 gives results from simulations and hardware results of embedding the PSO into a robot swarm. Section 4 discusses the trophallaxis- based search algorithm and section 5 gives simulation results using the trophallaxis algorithm. In section 6 we give our conclusions.

2. Particle Swarm Optimization and robots

In Particle Swarm Optimization (PSO) (Eberhart & Kennedy, 1995; Clerc &Kennedy, 2002; Eberhart & Shi, 2004), the potential solutions, called particles, "fly" through the problem space by following some simple rules. All of the particles have fitness values based on their position and have velocities which direct the flight of the particles. PSO is initialized with a group of random particles (solutions), and then searches for optima by updating generations. In every iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) the particle has achieved so far. This value is called pbest. Another "best" value that is tracked by the particle swarm optimizer is the best value obtained so far by any particle within the neighborhood. This best value is a neighborhood or local best and called lbest.

After finding the two best values, the particle updates its velocity and positions with following equations:

$$v_{n+1} = w_i v_n + c1 * r1 * \left(pbest_n - p_n\right) + c2 * r2 * \left(lbest_n - p_n\right) \quad (1)$$

$$p_{n+1} = p_n + v_{n+1} \quad (2)$$

w_i is the inertia coefficient which slows velocity over time; v_n is the current particle velocity;

p_n is the current particle position in the search space; r1 and r2 are random numbers between (0,1); c1 and c2 are learning factors. The stop condition is usually the maximum number of allowed iterations for PSO to execute or the minimum error requirement.

Because of the required search algorithm characteristics listed in section 1, we chose the PSO as the starting point for the search control algorithm. The PSO is computationally simple. It requires only four multiplies and four add/subtracts to update the velocity and then one add to update the position. The PSO can also be a distributed algorithm. Each agent/particle/bot can update its own velocity and position. The only external information is the local best – the best value by any particle within the neighborhood. The calculation of the local best can be done with a simple comparative statement. Thus, each bot does not need to know the results from each member of the population as in many traditional schemes. (Though in some versions of the PSO, the lbest is replaced by gbest, the global best – the best value of any particle within the population.)

The PSO also allows the contiguous movement of the bots. The updated position is relative to the current position so there are no jump changes in position or random movements. If there are constraints on the movement of the bot during each iteration, then limitations can be placed on the maximum and minimum velocity that is allowed for each particle/bot.

We propose embedding the PSO into a swarm of robots. In our approach, each bot is behaves as one particle in the PSO. Thus, each bot moves based on the PSO update equations (eqs. 1 and 2), makes a measurement and then broadcasts to the other bots in the swarm if it finds a new lbest measurement. We make some slight changes to the classic PSO algorithm to make it work with a robot swarm, so we call our algorithm the physically-embedded PSO (pePSO).

Other authors have investigated using biological principles (Zarzhitsky & Spears, 2005; Valdastri et al., 2006; Schmickl & Crailsheim, 2006; Triannni et al., 2006) and PSO-type algorithms (Hayes et al., 2000; Hayes et al., 2002, Doctor et al., 2004; Pugh & Martinoli, 2006; Pugh & Martinoli, 2007; Jatmiko et al., 2006; Jatmiko et al., 2007) with multiple (simple) robots for search applications. Specifically, Hayes et al. report using autonomous mobile robots for beacon localization (Hayes et al., 2000) and plume tracking/odor source localization (Hayes et al., 2002); they base their search techniques on biological principles (surge "upwind") but do not use the PSO algorithm directly. Doctor et al. (Doctor et al., 2004) discuss using the PSO for multiple robot searches. Their focus is on optimizing the parameters of the search PSO and do not consider the scalability of the standard PSO to large numbers of robots.

There are at least three other research groups that have investigated using mobile robots to do searches under the control of a PSO algorithm. Pugh et al. (Pugh & Martinoli, 2006; Pugh & Martinoli, 2007), explored using PSO on problems in noisy environments, focusing on unsupervised robotic learning. They used the PSO to evolve a robot controller that avoids obstacles and finds the source. They also investigated the possibility of using PSO without global position information. Jatmiko et al. (Jatmiko et al., 2006; Jatmiko et al., 2007) used mobile robots for plume detection and traversal. They utilized a modified form of the PSO to control the robots and consider how the robots respond to search space changes such as turbulence and wind changes. Akat and Gazi (Akat and Gazi, 2008) propose a version of the PSO for robot swarms that uses dynamic neighborhoods and asynchronous updates.

3. pePSO results

3.1 Simulation Conditions

We simulated the pePSO using three different test functions and five different target points with each function. The five different target points are shown in Figure 1. Only one target point was active during each simulation run; that is, there is only one target value in the search space at a time.

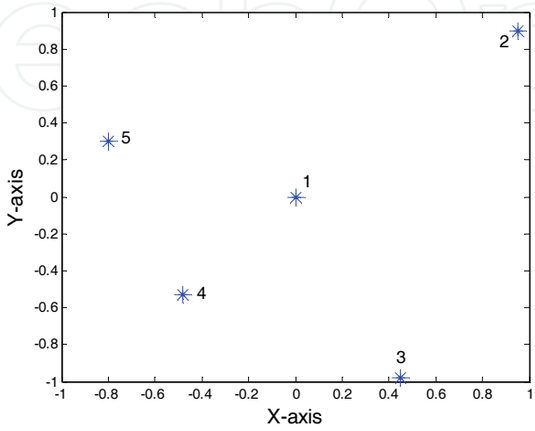


Fig. 1. Map of simulation search space scaled to cover -1 to +1 and showing the 5 different target point locations.

The three functions used in the simulation were the parabolic/spherical function, Rastrigin function, and Rosenbrock function. The three test functions are given by:

parabolic/spherical:

$$f(x,y) = (x - x_{target})^2 + (y - y_{target})^2 \tag{3}$$

Rastrigin:

$$f(x,y) = (x - x_{target})^2 - 10\cos(2\pi(x - x_{target})) + 10) + (y - y_{target})^2 - 10\cos(2\pi(y - y_{target})) + 10) \tag{4}$$

Rosenbrock:

$$f(x,y) = (1 - (x - x_{target} + 1))^2 + 100((y - y_{target} + 1) - (x - x_{target} + 1)^2)^2 \tag{5}$$

where (x,y) is the position of the bot/particle and (x_{target}, y_{target}) is the position of the target point. The spherical function was chosen because it approximates the expected dissipation pattern of chemicals, heat, etc that would be emitted by real-world targets. The Rastrigin and Rosenbrock functions were chosen because they are commonly used functions in PSO testing and they approximate a search space with obstacles and undulations. The Rosenbrock function used in our simulations is slightly different than the one reported in other simulations. We have shifted the minimum value of the function to the target location

instead of offset by (1,1) in x and y. This ensures that the target point is within the search space.

Plots of the three test functions are shown in Figure 2. In each case, we are trying to find the location of the global minimum. The size of the search spaces were -100 to 100 for the parabolic function and -5.1 to 5.1 for the Rastrigin and Rosenbrock functions. (The dimensions correspond roughly to meters as we set the velocity on the bot based on meters per second.) Unlike most PSO simulations, the search space boundary for our simulations represents a "hard" border – the particles/bots can not go outside the search space. We use a hard border because we want to approximate the conditions of actual robots searching in a room or some other confined space. The target point locations shown in Figure 1 were scaled for each of the search spaces. The (x,y) target locations are given in Table 1.

Target point	Parabolic	Rastrigin/ Rosenbrock
1	(0,0)	(0,0)
2	(94.7, 90.0)	(4.86, 4.61)
3	(45.1, -98.5)	(2.3, -5.02)
4	(-48.0, -52.7)	(-2.46, -2.71)
5	(-79.6, 29.9)	(-4.1, 1.54)

Table 1. Target Locations for each of the three Test Functions

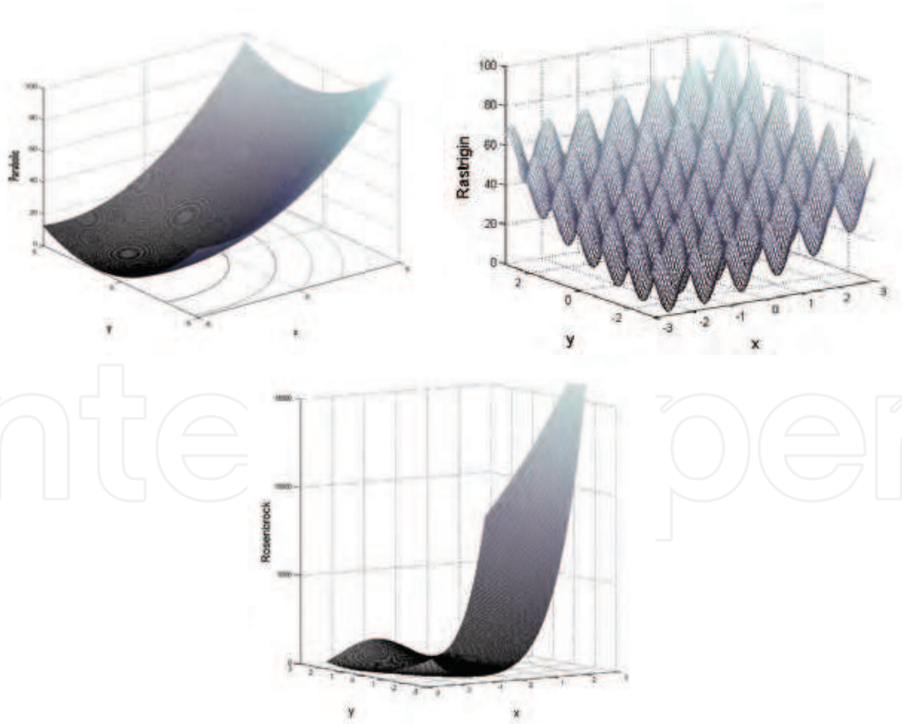


Fig. 2. Plots of three test functions. (a) Parabolic function with contours. Min value (the target value) is at [-4.1 1.54]; (b) Rastrigin function with min (target) point at -2.46 -2.71; (b) Rosenbrock function with target point = [0 0]. Min value is at a saddle point.

The simulation incorporated the mobility limitations of our small robots. The max turning radius allowed was 36 degrees and the maximum velocity was 0.9 m/s. The bots were positioned randomly around the search space at t = 0. In our expected deployment scenario the bots will most likely be "dropped off" at a particular point (e.g., near a door or window) and not dispersed randomly about the search space. However, we can easily incorporate a dispersion algorithm (Hsiang et al. 2002; Morlok & Gini, 2004) to spread the bots throughout the search space before beginning the search phase using pePSO.

For the simulation results, we used the following parameter values. Inertia coefficient, w_i, was set to 0.9. The max velocity of the bots was set to 0.9 m/s. Note that this is different than the old Vmax parameter in the original version of the PSO. We set an initial v for each bot/particle to simulate the behavior of the physical robot. The c1 and c2 coefficients were both set to 2.1. Since only three bots were used, the lbest topology was the same as gbest. That is, all of the bots communicated gbest and gbest location with all of the other bots.

One major difference between our simulations and the PSO results reported in the literature is how we calculate a successful search, and, relatedly, the stop condition. A successful search occurred when a bot/particle got within a tolerance of 0.2 in the x and y dimensions of the target location. Most PSO researchers track function value but we used location because of our search application. The simulation stopped when a bot got near the target or when 5400 function evaluations (about 15 minuntes) elapsed. Compared to other results, 5400 is a relatively small number of function evaluations but we want the bots to find the target within a reasonable time period.

3.2 Simulation results

To evaluate the effectiveness of the pePSO, we made several simulation runs. An effective "hit" occurred when one of the bots found the global peak within 15 minutes of simulation time (5400 function iterations). We used the 5 different target points and did 200 test cases for each target point for a total of 1000 runs for each test function. In each test case, the initial start locations of the bots were different. We evaluated the overall effectiveness of the algorithm (i.e., how many times it found the "target") and compared the pePSO to the standard PSO. In tables 2, 3 and 4 we compare the pePSO to the standard PSO with the number of particles set to 3. We modified the standard PSO to calculate the percent found (number of hits) based on proximity to the target location rather than on the function value so that it was consistent with the pePSO. The tables report the percentage of targets found. The results are shown for each of the three test functions as well as the total for all five target points.

For all three test functions, the pePSO performed much better than the PSO using same number of particles.

	Target point 1	Target point 2	Target point 3	Target point 4	Target point 5	Tot al
PSO	100	38.5	51	98	90.5	75.6
pePSO	99.5	99.5	99.5	99.5	100	**99.6**

Table 2. Parabolic Function, Results show the percent of targets found (out of 200 searches) for each of the five target points and also the overall results.

	Target point 1	Target point 2	Target point 3	Target point 4	Target point 5	Total
PSO	31	33.5	48	26	19	31.5
pePSO	97.5	96.5	65.5	99.5	92	**90.2**

Table 3. Rastrigin Function. Results show the percent of targets foune (out of 200 searches) for each of the five target points and also the overall results.

	Target point 1	Target point 2	Target point 3	Target point 4	Target point 5	Total
PSO	75	53	29.5	51.5	45	50.8
pePSO	86.5	92	55.5	88.5	92	**82.9**

Table 4. Rosenbrock Function. Results show the percent of targets foune (out of 200 searches) for each of the five target points and also the overall results.

The results in Tables 2-4 show that none of the algorithms found the target location 100% of the time, even with a relatively easy problem like the parabolic function. There are a number of reasons for the missed detections. First, we declared a failed search after relatively few function evaluations (relative compared to other researchers' results); we limited both the standard PSO and the pePSO to 5400 function evaluations. Second, we are not using very many particles/bots. Third, we have placed some of the target locations at positions near the edge of the search space which makes it somewhat more difficult for the PSO to find them.

3.3 Hardware test conditions

To investigate the effectiveness of the pePSO algorithm, we did several hardware experiments. In the experiments, a diffuse light source was placed near the ceiling of a dark room and pointed downward. Bots with light sensors were placed at various starting positions about 2 m apart to see (a) how often and (b) how quickly they could find the brightest spot of light in the room.

The layout of the test area is shown in Figure 3. The hatched boxes in the middle represent the obstacles that we placed in the search space for the second half of the testing. The highest concentration of light is immediately to the left of the vertical obstacle. Since we are using a diffuse light source, the global best is actually a rectangle approximately .25 m by .3 m.

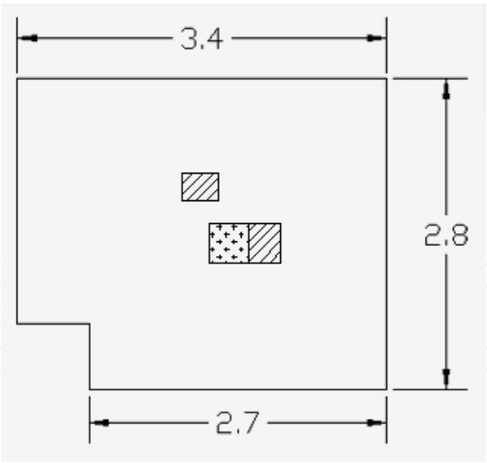


Fig. 3. Graphical representation of test area. Dimensions shown are in meters. Figure shows two obstacles and starred area is location of peak light intensity.

To make the pePSO work in a robotic swarm, we had to make several adjustments. First, the bots determine their position by triangulating from three cricket motes set up as beacons. To correct for any missed packets, the bots are programmed to move to the next position and then wait for two consecutive "clean" measurements (distance measurements within 2 cm of each other) from all three beacons. This wait leads to relatively long search times.

A second adjustment had to be made because of mechanical limitations in each bot. Since the bots steer with the front two wheels, the bots move in arcs rather than straight lines (see Figure 4). Each bot moves toward the desired position in the search space but upon arrival at the destination point, the orientation of the bot, as given by the direction of the wheels, is usually skewed relative to the movement. To compensate, we updated the bot's orientation angle at each iteration based on the bot's current position and its previous position. This eliminates the buildup of orientation errors that occur when a strict dead reckoning system for orientation is used.

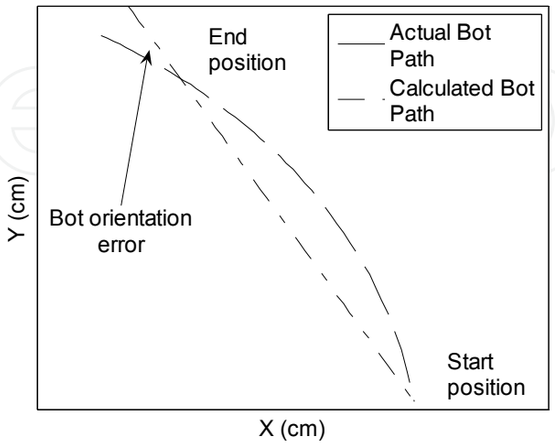


Fig. 4. Illustration of bot orientation mismatch.

A third adjustment was required because of the size of the bots. Unlike a simulation-only PSO, the hardware bots can get "stuck" at an obstacle or collide with another bot (particle). Once a bot got stuck or collided, we programmed the bot to back up and turn right. This allows the bot to move around long obstacles, such as a wall, even though it may require more than one cycle of backing up and turning to avoid.

3.4 Hardware results

During the hardware experiments, we programmed each bot in the swarm with identical programs. (The only difference is that each bot has a different identification number.) The PSO parameters used were c1 = 2, c2 = 2, and w_i = 1.0. We tried using w_i = 0.8 but it slowed the bots down considerably and led to many failed searches. Each bot was programmed to move in the desired direction for approximately 0.5 sec. The bot would then make a measurement, determine its new position, calculate its desired movement direction based on the PSO update, orient its wheels to that direction, if possible, and then move for 0.5 sec. The search was ended when there had been 20 iterations of the algorithm with no new local best discovered. We made some test runs using a stop condition with 10 iterations with no new lbest but we determined that 10 was insufficient.

Figure 5 shows the path traced out by 1 bot during a search sequence. The x and y axes are to scale and the axes are in cm. The asterisks (*) marks in the figures represent positions where a new local best was found. The rectangular box is the peak area of the search space. The bot starts in lower right corner. It moves up (north), finds new lbests and continues upward. When it starts to move away from the peak, it circles clockwise and then begins moving to the left (westward). When the light intensity measurements begin to taper off again, the bot circles counterclockwise back toward its previous best. The circle behavior is because the bot is limited in its turning radius – it can not make a sharp turn. Thus, it must move toward the global best in a roundabout fashion. Eventually, it settles on to the peak light value.

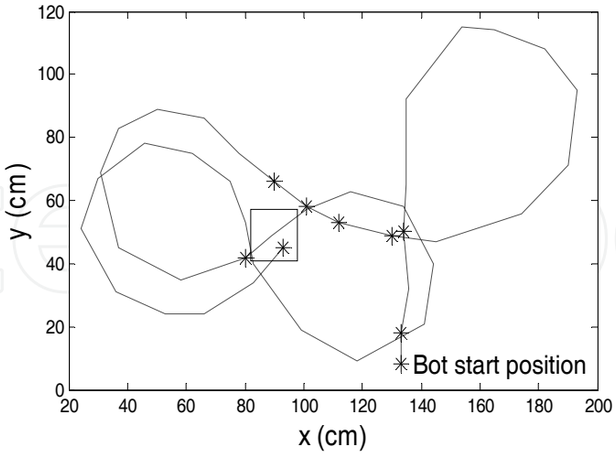


Fig. 5. Path in search space with no obstacles for 1 bot. Asterisks show positions where lbests were recorded.

The PSO algorithm was tested using swarm sizes of 1, 2 and 3 bots. (The 1-bot swarm was merely for baseline comparison.) Ten runs were made for each of the three cases. We tracked how many times the bots found the brightest spot and how long it took for the bots

to locate the peak. If twenty iterations of the algorithm passed with no new global best, then we declare the search is over.
The quantitative results are shown in Tables 5 and 6. The results are for two different search spaces: one with no obstacles and one with obstacles. The table shows the number of bots in the swarm, the number of successful (completed) runs, the median search time for all searches, the average search time for just the successful searches, the standard deviation of the successful search times and the 90% confidence interval for the average time. The confidence interval is based on t-statistics using the successful searches as the degrees of freedom.

Runs without obstacles.

Test condition	Complete runs (out of 10)	Median time (sec)	Average (sec)	Standard deviation (sec)	Confidence Interval (90%)
pePSO (1 bot)	6	118.5	180.2	101	± 83.1
pePSO (2 bots)	10	177	176.1	68.9	± 39.9
pePSO (3 bots)	10	133.5	109.6	55.0	± 31.9
Phototaxis (1 bot)	9	280	273.8	45.1	± 28.0
3 bot (no commun)	10	351.5	362.7	98.7	± 57.2

Table 5. Results from using PSO to program a suite of bots with sensor and position corrections. No obstacles in the search space

Test condition	Complete runs (out of 10)	Median time (sec)	Average (sec)	Standard deviation (sec)	Confidence Interval (90%)
pePSO (1 bot)	8	268	250.2	108.6	± 71.4
pePSO (2 bots)	10	178	181.7	76.1	± 44.1
pePSO (3 bots)	10	108.5	125.8	65.2	± 37.8
3 bot (no commun)	9	290	294.1	111.3	± 69.0

Table 6. Results from pePSO in a search space with obstacles

Table 5 also shows the results from searches using a phototaxis approach and using three bots without any communication among the bots. These experiments provide a comparison for the effectiveness and search times for the pePSO. For the phototaxis experiments, we used a cylinder to make a directional light sensor on one of the bots. The bot then took light intensity measurements as it made a 360 degree loop. (The loop had a radius of 0.25 m.) The bot then moved in the direction of the greatest light reading. Our test results show that the multi-bot pePSO is faster and more effective than the phototaxis approach.

To compare the effectiveness of the multi-bot pePSO, we did a 3 bot search with no communication among the bots. Essentially, each bot moved toward bright regions of the room based on the PSO update equation without the lbest term. If a bot received a higher light reading than ever before, then it continued moving in that direction; thus, it is a pseudo-gradient method. The results in Table 6 show that the 3-bot pseudo-gradient method was effective (19 out of 20 successful runs) but it was much slower than the 3-bot pePSO. The 3-bot pePSO search is over 55% faster than the pseudo-gradient based 3-bot search both with obstacles and without obstacles.

Increasing the number of bots does two things. It leads to more successful searches and reduces the time to find the peak/best value. We see that even with the obstacles in the search space, the swarm is still able to find the "target" or peak light intensity every time for a multi-bot search. In general, the searches take longer when there are obstacles in the search space but the search is still successful. The times should be used for comparison purposes and not necessarily as an absolute reference. As mentioned earlier, sketchy communications with the beacons forced the bots to wait for two consecutive good data points at each iteration. This waiting time greatly increased the search time.

To overcome the weak signal from one of the beacons, we modified the position algorithm to allow the bot to calculate its position with distance data from only two beacons. The bot initially tried to get two consecutive good measurements from all three beacons. If after seven seconds there was still not accurate distance information, then the bot would use the distance measurements from two beacons to calculate its (x,y) position.

During our initial experiments, we noticed that the light sensors on the bots were mismatched. That is, different bots would read different values for the same light level. This imbalance led to some erroneous values for the search times. Specifically, if the bot with the lowest light readings found the peak value first, then other bots would circle toward that point. When another bot (with a light sensor that recorded higher light values) moved to the same location, it would record a higher light value and the algorithm would think a "new" target had been found. To correct for the different sensor readings, we used linear splines for each individual sensor to adjust and match the light sensor outputs of the three bots.

4. Trophallaxis and robots

4.1 Background

In our trophallaxis-based algorithm, the bots (or, more generally, particles) do not actually exchange information but instead make sensor measurements when two or more bots/particles are "in contact". The bots remain stationary for a certain time that is proportional to the measurement value. Bots thus cluster in areas of the search space that have high fitness/measurement values. Each bot is independent (i.e., not reliant on neighbor bot measurements) but must have contact with the other bots to find the peaks.

This new trophallaxis-based search algorithm has several advantages over other swarm-based search techniques. First, no bot-bot (particle-particle) communication is required. Thus, there is no concern with communication radius, protocol, or bandwidth. Unlike classic PSO-based techniques, the bots do not have to be arranged in topologies or communicate personal or global best information to any neighbors. The only communication that may be

required is signaling between bots to differentiate collisions between bots and collisions with obstacles.

Second, the bots do not have to know their position. If position information is available, from beacons or some other source, then position information can be communicated at the end of the search. But during the search, the bot/particle moves randomly except when it stops, takes a measurement, and waits. At the end of the search, the cluster locations can be determined from a remote camera, special-purpose robot, or human canvassing.

Third, no on-board processing or memory is required – the bot does not even have to do the relatively simple PSO update equations. The bot/particle moves at random, takes a measurement and does a multiplication. It is so simple that a microcontroller may not be required, only some simple digital logic hardware.

The Trophallactic Cluster Algorithm (TCA) has four basic steps:

Step 1: Bots start randomly throughout the search space and then move at random through the search space.

Step 2: If a bot intersects or collides with another bot, then it stops.

Step 3: After stopping, the bot measures the "fitness" or function value at that point in space. It then waits at that point for a prescribed time based on the measurement. The higher the measurement value, then the longer the wait time.

Step 4: When done, determine the locations of the clusters of bots. (We assume that this step is performed by an agent or agents that are separate from the swarm.)

Step 1 is similar to the first step in the standard Particle Swarm Optimization (PSO) algorithm. For a software only optimization scheme, it is straightforward to randomly initialize the particles within the search boundaries. For a hardware scheme, a dispersion algorithm (Siebold & Hereford 2008; Spears et al., 2006) can be used to randomly place the bots.

For random movement, we pick a direction and then have the bots move in a straight line in that direction until they encounter an obstacle, boundary or other bot. Thus, there is no path adjustment or Brownian motion type movement once the (random) initial direction is set. There is a maximum velocity at which the bots move throughout the search space. We experimented briefly with different maximum velocities to see its affect on overall results but we usually set it based on the expected maximum velocity of our hardware bots.

In step 2, we detect collision by determining whether bots are within a certain distance of each other. In software, this is done after each time step. In hardware, it can be done using infrared sensors on each bot.

In a hardware implementation of the TCA, we would need to distinguish between collisions with obstacles and collisions with other bots. Obstacles and walls just cause the bot/particle to reorient and move in a new direction. They do not lead to a stop/measure/wait sequence. So once a collision is detected, the bot would have to determine if the collision is with another bot or with an obstacle. One way to do this is to have the bots signal with LEDS (ala trophallaxis where only neighbors next to each other can exchange info). In this way, each bot will know it has encountered another bot.

Once a bot is stopped (as a result of a collision with another bot), then it measures the value of the function at that location. (In hardware, the bot would take a sensor reading.) Since the bots are nearly co-located, the function value will be nearly the same for both bots. The wait time is a multiple of the function value so the bot(s) will wait longer in areas of high fitness relative to areas with low function values. Other bots may also "collide" with the stopped and waiting bots but that will not reset the wait times for the stopped bots. For the results in

this paper the wait time was exponentially related to the measurement value; we experimented with linear wait times as well.
We did step 4 (determine the clusters) when the search is "done". In general, the bots begin to collide/stop/wait at the beginning. Thus, the bots tend to cluster soon after the search begins so the search can be stopped at any time to observe the location(s) of the clusters. In 2D, the clusters tend to become more pronounced as the iterations increase so waiting a longer time can make the position(s) of the peak(s) more obvious.

4.2 Related work

The TCA is based on the work of Thomas Schmickl and Karl Crailsheim (Schmickl & Crailsheim 2006; Schmickl & Crailsheim 2008) who developed the concept based on the trophallactic behavior of honey bees. Schmickl and Crailsheim use the trophallactic concept to have a swarm of bots move (simulated) dirt from a source point to a dump point. The bots can upload "nectar" from the source point, where the amount of nectar for each bot is stored in an internal variable. As the robots move, the amount of stored nectar decreases, so the higher the nectar level, then the closer to the source. Each robot also queries the nectar level of the robots in the local neighborhood and can use this information to navigate uphill in the gradient. There is a also a dump area where the loaded robots aggregate and drop the "dirt" particles. The swarm had to navigate between the source and the dump and achieved this by establishing two distinct gradients in parallel.
Their preliminary results showed a problem where the bots tended to aggregate near the dump and the source. When that happened, the bridge or gradient information between the source and the dump was lost. To prevent the aggregation, they prevented a percentage of their robots from moving uphill and just performed a random walk with obstacle avoidance. Even though the work of Schmickl and Crailsheim is significant, they show no published results where they apply the trophallactic concept to strictly search/optimization type problems. Nor do they show results when there is more than one peak (or source point) in the search space. They also require bot-bot communications to form the pseudo-gradient that the loaded (or empty) bots follow, while our TCA approach does not require adjacent particles/bots to exchange nectar levels (or any other measured values).
In (Ngo & Schioler, 2008), Ngo and Schioler model a group of autonomous mobile robots with the possibility of self-refueling. Inspired by the natural concept of trophallaxis, they investigate a system of multiple robots that is capable of energy sharing to sustain robot life. Energy (via the exchange of rechargeable batteries) is transferred by direct contact with other robots and fixed docking stations.
In this research, we are applying the concept of trophallaxis to solve a completely different type of problem than Ngo and Schioler, though some of their results may be applicable if we expand our research to include energy use by the robots.

5. Trophallaxis search results

5.1 Test conditions

We tested the TCA algorithm on three functions: two 1D and one 2D. The 1D functions are denoted F3 and F4 and were used by Parrott and Li (Parrott and Li, 2006) for testing PSO-based algorithms that find multiple peaks in the search space. The equations for F3 and F4 are given by

$$F3(x) = \sin^6(5\pi(x^{3/4} - 0.05)) \tag{6}$$

$$F4(x) = \exp(-2\log(2)(\frac{x - 0.08}{.854})^2)\sin^6(5\pi(x^{3/4} - 0.05)) \tag{7}$$

Plots for the function F3 and F4 are shown in figure 6. Each 1D test function is defined over the scale of 0 ≤ x ≤ 1. F3 has five equal-height peaks (global optima) that are unevenly spaced in the search space. F4 also has five unevenly spaced peaks but only one is a global optimum while the other four peaks are local optima. Our goal is to find all five peaks; that is, the global optimum plus the local optima. The peak locations are given in Table 7.

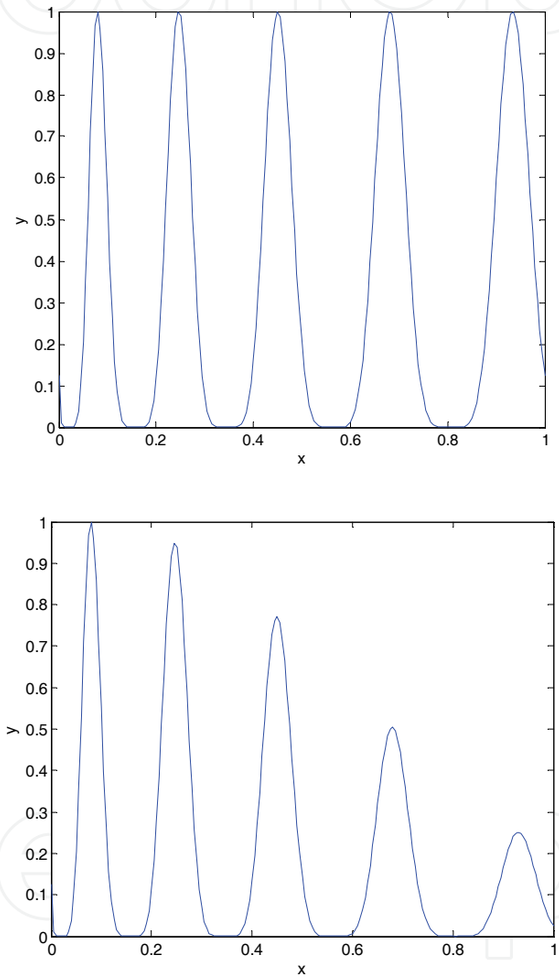


Fig. 6. 1D test function with equal peaks, F3, and with unequal peaks, F4

Peak #	1	2	3	4	5
X locations	.0797	.2465	.4505	.6815	.9340

Table 7. Peak locations for test functions F3 and F4

The 2D function is a slight variation of the standard Rastrigin function. The equation for the Rastrigin is given in equation 4 with the range on x and y is -5.12 to 5.12. The Rastrigin is

highly multimodal (see figure 2) and has one global minimum. For the TCA simulations, we modified the Rastrigin so that it has a peak of 1 at (.35, .35) instead of a minimum at the origin. We also scaled the function slightly so that there are nine peaks (1 global, 8 local) within the [-5.12, 5.12] range. The modified Rastrigin function is shown in figure 7.

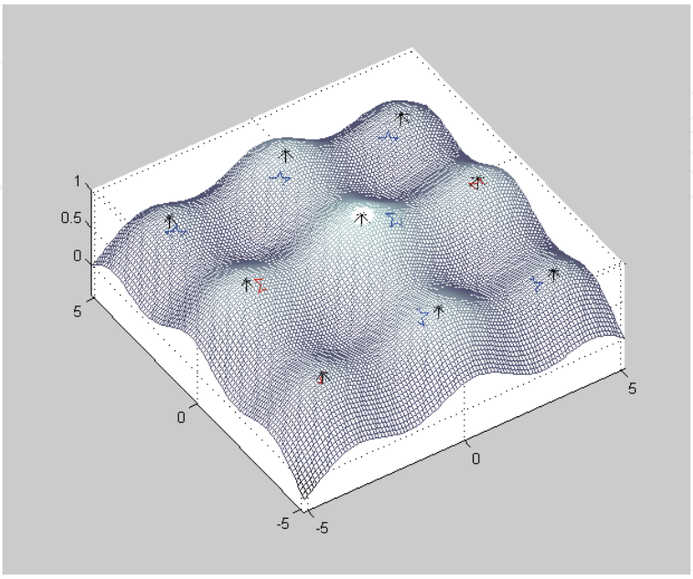


Fig. 7. Plot of modified Rastrigin function scaled so that it has 9 peaks in [-5.12,5.12] range and peak value is equal to 1.

We evaluated the effectiveness of the TCA algorithm using two different metrics. The first metric is the total percentage of peaks found (*found rate*). Since each function has multiple peaks (both global and local) we totaled the number of actual peaks that the swarm found divided by the total number of peaks (see equation 8). Note that "peaks found" refers to only those bot clusters that are within ± .04 (1D) or a radius of .4 (2D) of the actual peak location.

Found rate = (peaks found)/(total number of peaks in search space) (8)

The second metric is related to the success rate from Parrott and Li:

Success rate = (Runs where more than half of total peaks found)/(total number of runs) (9)

The *success rate* gives an indication of how many of the runs were successful. We designate a successful run as one where a majority of the peaks (more than half) in the search space are found. This definition is based on the robot search idea where the goal is for the robots to quickly cluster near the target points. We note that this is a slightly different definition for success rate than Parrott and Li; their success rate is based on the closest particle and finding all of the peaks in the search space.

The locations of the bot clusters were determined using the K-means clustering algorithm. The K-means algorithm minimizes the sum, over all clusters, of the point-to-cluster centroid distances. We then compared the cluster centroid to the actual peak location to determine if the peak was found or not. For the 1D functions, we used a tolerance of ± 0.04 between the cluster centroid and the actual peak and for the 2D functions we used a radius of 0.4.

We considered two clustering approaches; the first one uses all of the bots when determining the cluster centroids. The second approach uses only the final position of the

bots that are stopped (that is, in a collision) when determining clusters. We refer to the second approach as "cluster reduction", since it reduces the number of bots that are considered.

5.2 Trophallactic Cluster Algorithm 1D results

Qualitative results from computer simulations of the TCA for the two 1D functions F3 and F4 are shown in Figure 8. The top plot in the figure shows the original function; the middle plot shows the final bot positions (after 400 iterations) with each bot position represented by a star (*). The bottom plot is a normalized histogram; the histogram is made by tracking the position of each bot after each time interval. The figure reveals that the bots do cluster around the peaks in the function and thus give evidence that the TCA will reliably find multiple peaks.

The histogram plots reveal some interesting information. For both F3 and F4, there are significant peaks in the histogram at the same locations as the function peaks, providing evidence that the bots spend a majority of time near the peaks and it is not just at the end of the simulation that the bots cluster. Also, for F3 the histogram peaks have approximately the same amplitude (peak amplitudes range from 0.7 to 1.0). For F4, however, the histogram peaks diminish in amplitude in almost direct proportion to the function peaks (peak amplitudes diminish from 1.0 down to 0.25). This implies that the bots are spending more time in the vicinity of the larger peaks. The bots are thus "attracted" to stronger signals, as expected.

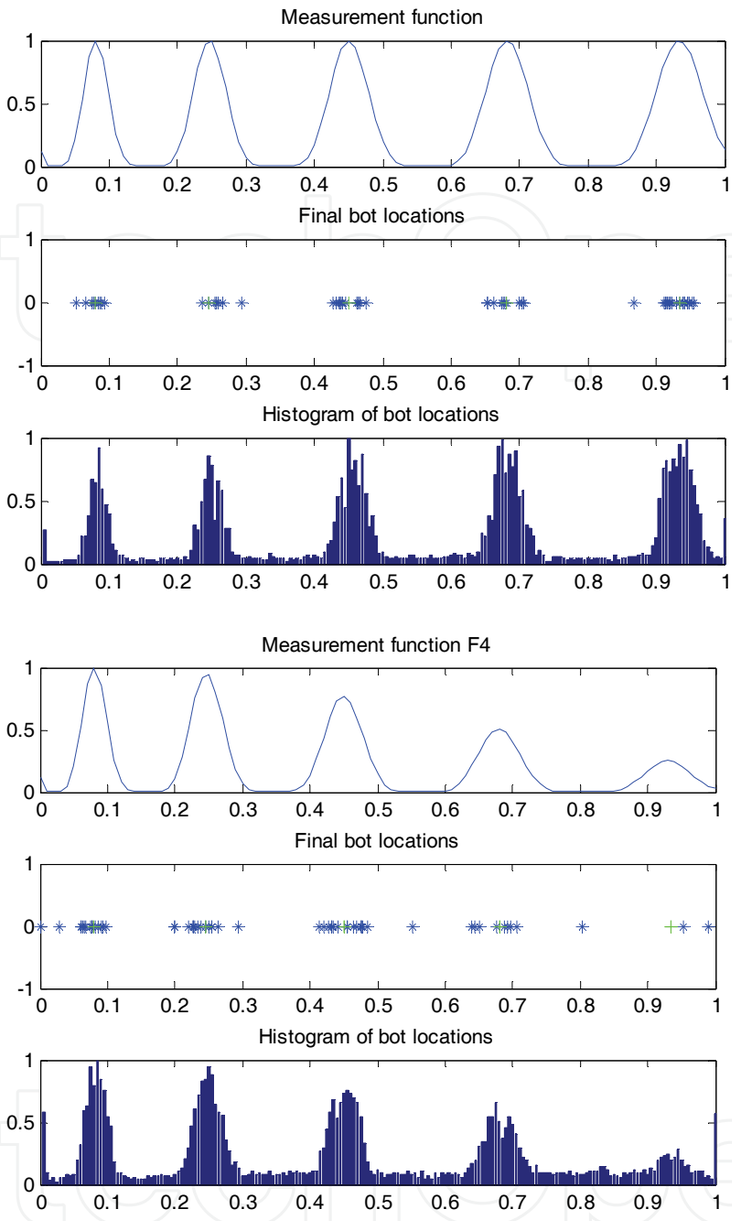


Fig. 8. Qualitative results for function F3 (left) and function F4 (right). Final bot locations are shown in the middle plot and histograms of bot positions are shown in the bottom plot

We performed computer simulations to tailor three of the parameters of TCA algorithm for 1D functions. The three parameters were *tmax*, the maximum number of iterations for the simulation, *nbots*, the number of bots to use, and *waitfactor*. The *waitfactor* sets how long each bot waits based on the measured value after a collision. We tried linear wait functions (wait time increases linearly with measurement value) but had more success with exponential wait functions give by

$$\text{Wait time} = waitfactor * (e^{(\text{measurement})}-1) \quad (11)$$

For the parameter selection, we varied one parameter at a time and repeated the simulation 100 times. Plots of the average found rate for parameters *nbots* and *waitfactor* with no cluster reduction are shown in Figure 9. The first plot shows the found rate as *nbots* is varied from 10 to 200 with *tmax* set to 500 and *waitfactor* set to 5. The second plot shows the found rate as *waitfactor* is varied from 1 to 10 with *tmax* = 500 and *nbots* = 80. Similar tests were done with cluster reduction.

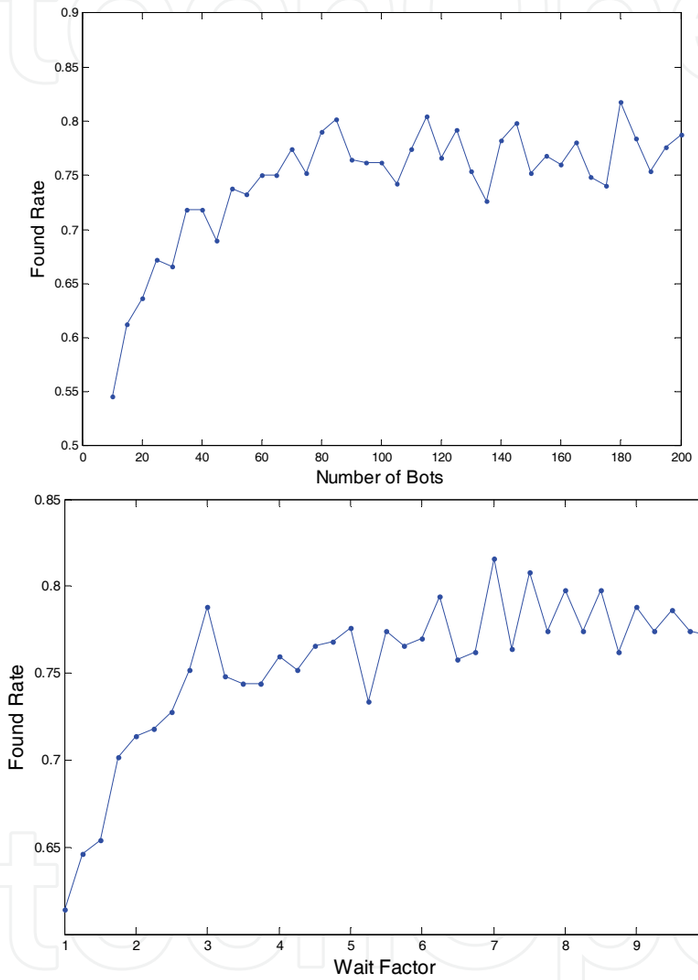


Fig. 9. Results showing found rate vs *nbots* (left figure) and *waitfactor* (right figure) for F3 function

When the peaks were found with no cluster reduction, the found rate versus the parameter value curve resembled (1-e-^{x}) shape and asymptotically approached a found rate of about 78%. Thus, there was not one precise parameter value but a range of parameter values that led to the best performance: *nbots* greater than 80 and *waitfactor* greater than 3. The *tmax* curve was flat – it appears that the bots quickly cluster near the peaks and there is little change in performance as the *tmax* is increased. A summary for the parameter selection process is shown in Table 7.

Parameter	w/out cluster reduction	w/ cluster reduction
nbots	≥ 80	< 20
tmax	≥ 300	≥ 100
waitfactor	≥ 3	≥ 4

Table 7. Best parameter ranges for TCA for 1D functions

When the bot cluster centroids were found with cluster reduction, the response curves for *tmax* (flat) and *waitfactor* (exponential) were similar in shape as without cluster reduction, though the curve asymptotically approached a found rate of 86%. The response curve for *nbots* was different, however. The found rate went **up** as *nbots* decreased so fewer bots was better than more bots, assuming that there at least 5 stopped bots in the search space. It appears that for a small number of bots, that there was a smaller percentage of bots in the clusters (say 13 out of 20 instead of 85 out of 90) and those off-clusters bots moved the cluster centers away from the peaks and led to missed detections.
For the final results we used the parameter values *tmax* = 500, *waitfactor* = 4, and *nbots* = 60. We used the same parameter values for all test cases: two different functions (F3 and F4) and with and without cluster reduction. There was a slight dilemma on the choice for *nbots* since more bots did better without cluster reduction and fewer bots did better without cluster reduction so we compromised on 60. We ran the 1D simulations 500 times and the results are shown in Table 8.

	F3				F4			
	Avg # peaks found	Std dev	Found rate	Success rate	Avg # peaks found	Std dev	Found rate	Success rate
w/out cluster reduction	4.052	.9311	81.04 %	97.2%	4.016	.9195	80.32%	97.6%
w/ cluster reduction	4.130	.7761	82.6%	98.2%	4.098	.7782	81.96%	99.0%

Table 8. Final results showing average number of peaks found (out of 5 peaks), found rate and success rate for 500 iterations

The 1D results show that the TCA was very effective at finding a majority of the peaks in the 2 different functions. The success rate was above 97% and the found rate was above 80%. These are good results for an algorithm where the individual particles/bots do not have position information and no bot-bot communication is required.
The results shown in Table 8 are very consistent. The TCA algorithm finds 4 out of the 5 peaks and there is little difference in the results between F3 (peaks of equal height) and F4 (peaks have different heights). There is a slight improvement with cluster reduction, that is, when only the stopped bots are used to determine the peak locations.

5.3 Trophallactic Cluster Algorithm 2D results

The results of a typical two dimensional search using the TCA are shown in Figure 10. The first figure shows a plot of the Rastrigin function with the final bot positions superimposed

on top of it. The second figure shows the final bot positions and the centroids of nine clusters found by the K-means clustering algorithm (black stars). Note that six of the cluster centroids are close to actual peaks in the Rastrigin function, but only one of the centroids was within the required tolerance and was thus declared a peak (red diamond).

The initial 2D results, like those shown in Figure 10, illustrate the usefulness of cluster reduction. Figure 11 shows the same final bot positions as in Figure 10, except only the clusters with three of more bots are kept. That is, small clusters of two bots and any bots not in a cluster are eliminated. The K-means clustering is performed with this smaller set of bots and the cluster centroids compared to the peak locations.

After cluster reduction, there are four cluster centroids that are within the tolerance radius of the peak instead on only one centroid. Thus, ignoring the still-moving bots after the conclusion of the search clarifies the definition of the clusters of bots. This in turn leads to more accurate identification of the peaks in the function.

As with the 1D test functions, computer simulations were conducted to refine the three parameters of *tmax*, *nbots*, and *waitfactor* for the 2D case. The results from these simulations for *nbots* and *tmax* are shown in Figure 12. Each graph shows the found rate as the parameter was varied; they are the result of averaging 100 simulations for each set of parameters. For found rate vs *nbots* graph, *tmax* was set to 1600 and *waitfactor* was set to 4. For the found rate vs *tmax* graph, *nbots*= 300 and *waitfactor* = 4.

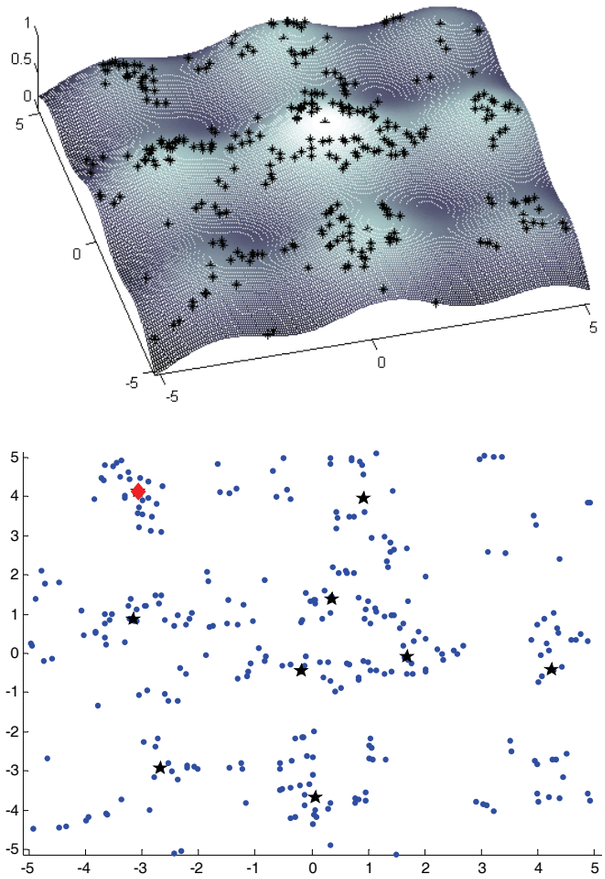


Fig. 10. Typical TCA search results for the Rastrigin 2D function. Left figure: Rastrigin function showing final bot position. Right figure: final bot position with cluster centroids - red diamond denotes found peak and black star denotes cluster centroid.

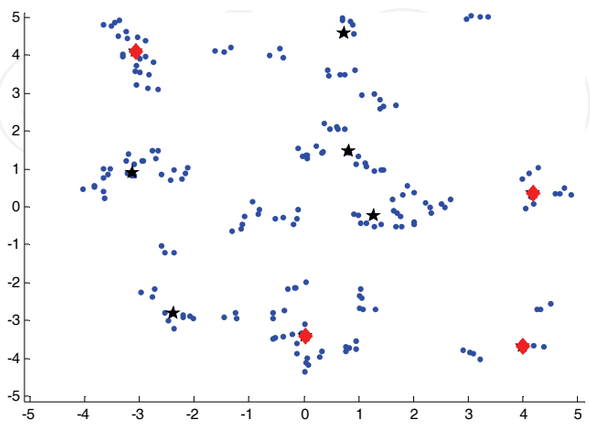


Fig. 11. Analysis of typical TCA search with cluster reduction. Red diamond denotes found peak. Black star denotes inaccurate peak.

The results and interpretation of these two dimensional results are similar to the one dimensional case. The results roughly follow a $(1\text{-}e^{x})$ form. Therefore, the appropriate parameter values are again ranges rather than precise values. The values are given in Table 9.

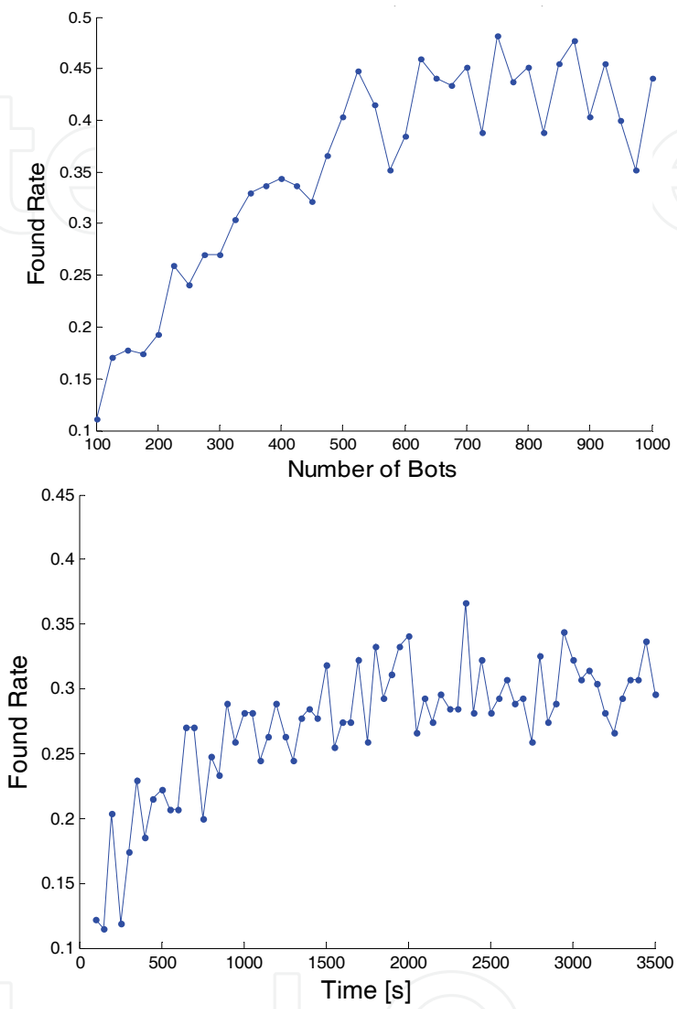


Fig. 12. Results showing found rate vs *nbots* (left figure) and *tmax* (right figure) for Rastrigin function

Parameter	**w/out cluster reduction**	**w/ cluster reduction**
nbots	≥ 500	< 300
tmax	≥ 1600	≥ 1700
waitfactor	≥ 4	≥ 4

Table 9. Best parameter ranges for 2D Rastrigin function

The final two dimensional results were obtained using parameter values *tmax = 1600, nbots = 600,* and*waitfactor = 4.* The same parameters were used both with and without cluster reduction. We averaged the results from 500 simulations and the results are shown in Table 10.

	Avg # peaks found	Std deviation	Found rate	Success rate
w/out cluster reduction	3.7360	1.4375	41.5%	29.2%
w/ cluster reduction	3.3820	1.4888	37.6%	21.8%

Table 10. Final results showing average number of peaks found (out of 9 peaks), found rate and success rate for 500 iterations for the Rastrigin 2D function

The results from the 2D Rastrigin function are not as good as the results from the 1D functions. The lower found rate is due primarily to the fact that the Rastrigin function is a hard function – the peaks do not stand out as prominently as the F3 or even the F4 peaks. In addition, the 2D search space is much larger; for the Rastrigin function, we used a scale of -5.1 to +5.12 for both x and y, while the 1D functions are only defined between $0 \leq x \leq 1$. We increased the tolerance for the 2D results to 0.4 and it appeared that many cluster centroids were close to the actual peaks but, unfortunately, not within the tolerance radius.

6. Conclusions

We developed and tested two biologically inspired search strategies for robot swarms. The first search technique, which we call the physically embedded Particle Swarm Optimization (pePSO) algorithm, is based on bird flocking and the PSO. The pePSO is able to find single peaks even in a complex search space such as the Rastrigin function and the Rosenbrock function. We were also the first research team to show that the pePSO could be implemented in an actual suite of robots.

Our experiments with the pePSO led to the development of a robot swarm search strategy that did not require each bot to know its physical location. We based the second search strategy on the biological principle of trophallaxis and called the algorithm Trophallactic Cluster Algorithm (TCA). We have simulated the TCA and gotten good results with multi-peak 1D functions but only fair results with multi-peak 2D functions. The next step to improve TCA performance is to evaluate the clustering algorithm. It appears that many times there is a cluster of bots near a peak but the clustering algorithm does not place the cluster centroid within the tolerance range of the actual peak. A realistic extension is to find the cluster locations via the K-means algorithm and then see if the actual peak falls within the bounds of the entire cluster.

7. References

Akat S., Gazi V., "Particle swarm optimization with dynamic neighborhood topology: three neighborhood strategies and preliminary results," IEEE Swarm Intelligence Symposium, St. Louis, MO, September 2008.

Chang J., Chu S., Roddick J., Pan J., "A parallel particle swarm optimization algorithm with communication strategies", Journal of Information Science and Engineering, vol. 21, pp. 809-818, 2005.

Clerc M., Kennedy J., "The particle swarm – explosion, stability, and convergence in a multi-dimensional complex space", IEEE Transactions on Evolutionary Computation, vol. 6, pp. 58-73, 2002.

Doctor S., Venayagamoorthy G., Gudise V., "Optimal PSO for collective robotic search applications", IEEE Congress on Evolutionary Computation, Portland, OR, pp. 1390 – 1395, June 2004.

Eberhart R., Kennedy J., "A new optimizer using particle swarm theory", Proceedings of the sixth international symposium on micro machine and human science, Japan, pp. 39-43, 1995.

Eberhart R., Shi Y., Special issue on Particle Swarm Optimization, IEEE Transactions on Evolutionary Computation, pp. 201 – 301, June 2004.

Hayes A., Martinoli A., Goodman R., "Comparing distributed exploration strategies with simulated and real autonomous robots", Proc of the 5th International Symposium on Distributed Autonomous Robotic Systems, Knoxville, TN, pp. 261-270, October 2000.

Hayes A., Martinoli A., Goodman R., "Distributed Odor Source Localization", IEEE Sensors, pp. 260-271, June 2002.

Hereford J., "A distributed Particle Swarm Optimization algorithm for swarm robotic applications", 2006 Congress on Evolutionary Computation, Vancouver, BC, pp. 6143 – 6149, July 2006.

Hereford J., Siebold M., Nichols S., "Using the Particle Swarm Optimization algorithm for robotic search applications", Proceedings of the 2007 IEEE Swarm Intelligence Symposium, Honolulu, HI, pp. 53-59, April 2007.

Hereford J., "A distributed Particle Swarm Optimization algorithm for swarm robotic applications", 2006 Congress on Evolutionary Computation, Vancouver, BC, pp. 6143 – 6149, July 2006.

Hereford J., Siebold M., Nichols S., "Using the Particle Swarm Optimization algorithm for robotic search applications", Proceedings of the 2007 IEEE Swarm Intelligence Symposium, Honolulu, HI, pp. 53-59, April 2007.

Hereford J., Siebold M., "Multi-robot search using a physically-embedded Particle Swarm Optimization", *International Journal of Computational Intelligence Research*, March 2008.

Hsiang T-R, Arkin E. M., Bender M. A., Fekete S. P., Mitchell J. S. B., "Algorithms for rapidly dispersing robot swarms in unknown environments", Fifth International Workshop on Algorithmic Foundation of Robotics, December 2002.

Jatmiko W., Sekiyama K., Fukuda T., "A PSO-based mobile sensor network for odor source localization in dynamic environment: theory, simulation and measurement", 2006 Congress on Evolutionary Computation, Vancouver, BC, pp. 3781 – 3788, July 2006.

Jatmiko W., Sekiyama K., Fukuda T., "A PSO-based mobile robot for odor source localization in dynamic advection-diffusion with obstacles environment: theory, simulation and measurement", IEEE Computational Intelligence Magazine, vol. 2, num. 2, pp. 37 – 51, May 2007.

Kennedy J., "Some issues and practices for particle swarms", Proceedings of the 2007 IEEE Swarm Intelligence Symposium, Honolulu, HI, pp. 162 – 169, April 2007.

Morlok R., Gini M., "Dispersing robots in an unknown environment", Distributed Autonomous Robotic Systems 2004, Toulouse, France, June 2004.

Ngo T. D., Schioler H., "Randomized robot trophollaxis", in *Recent Advances in Robot Systems*, A. Lazinica ed., I-Tech Publishing: Austria, 2008.

Parrott D., Li X., "Locating and tracking multiple dynamic optima by a particle swarm model using speciation", *IEEE Transactions on Evolutionary Computation*, vol. 10, pp. 440-458, August 2006.

Pugh J., Martinoli A., "Multi-robot learning with Particle Swarm Optimization", Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, May 2006.

Pugh J., Martinoli A., "Inspiring and modeling multi-robot search with Particle Swarm Optimization", Proceedings of the 2007 IEEE Swarm Intelligence Symposium, Honolulu, HI, pp. 332 – 339, April 2007.

Reynolds C. W., "Flocks, Herds, and Schools: A Distributed Behavioral Model", ACM SIGGRAPH '87 Conference Proceedings, Anaheim, CA, pp. 25-34, July 1987.

Schmickl T., Crailsheim K., "Trophallaxis among swarm-robots: A biologically inspired strategy for swarm robots", BioRob 2006: Biomedical Robotics and Biomechatronics, Pisa, Italy, February 2006.

Schmickl T., Crailsheim K., "Trophallaxis within a robotic swarm: bio-inspired communication among robots in a swarm", *Autonomous Robot*, vol. 25, pp. 171-188, August 2008.

Siebold M., Hereford J., "Easily scalable algorithms for dispersing autonomous robots", 2008 IEEE SoutheastCon, Huntsville, AL, April 2008.

Spears D., Kerr W., and Spears W., "Physics-based robot swarms for coverage problems", *The International Journal of Intelligent Control and Systems*, September 2006, pp. 124-140.

Spears W., Hamann J., Maxim P., Kunkel P., Zarzhitsky D., Spears, C. D. and Karlsson, "Where are you?", Proceedings of the SAB Swarm Robotics Workshop, September 2006, Rome, Italy.

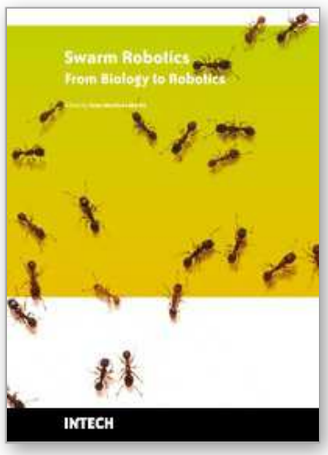
Teller S., Chen K., Balakrishnan H., "Pervasive Pose-Aware Applications and Infrastructure", IEEE Computer Graphics and Applications, July/August 2003.

Triannni V., Nolfi S., Dorigo M., "Cooperative hole avoidance in a swarm-bot", Robotics and Autonomous Systems, vol. 54, num. 2, pp. 97-103, 2006.

Valdastri P., Corradi P., Menciassi A., Schmickl T., Crailsheim K., Seyfried J., Dario P., "Micromanipulation, communication and swarm intelligence issues in a swarm microbotic platform", Robotics and Autonomous Systems, vol. 54, pp. 789-804, 2006.

Zarzhitsky D., Spears D., Spears W., "Distributed robotics approach to chemical plume tracing," Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2974-2979, August 2005.

IntechOpen



Swarm Robotics from Biology to Robotics

Edited by Ester Martinez Martin

ISBN 978-953-307-075-9
Hard cover, 102 pages
Publisher InTech
Published online 01, March, 2010
Published in print edition March, 2010

In nature, it is possible to observe a cooperative behaviour in all animals, since, according to Charles Darwin's theory, every being, from ants to human beings, form groups in which most individuals work for the common good. However, although study of dozens of social species has been done for a century, details of how and why cooperation evolved remain to be worked out. Actually, cooperative behaviour has been studied from different points of view. Swarm robotics is a new approach that emerged on the field of artificial swarm intelligence, as well as the biological studies of insects (i.e. ants and other fields in nature) which coordinate their actions to accomplish tasks that are beyond the capabilities of a single individual. In particular, swarm robotics is focused on the coordination of decentralised, self-organised multi-robot systems in order to describe such a collective behaviour as a consequence of local interactions with one another and with their environment. This book has only provided a partial picture of the field of swarm robotics by focusing on practical applications. The global assessment of the contributions contained in this book is reasonably positive since they highlighted that it is necessary to adapt and remodel biological strategies to cope with the added complexity and problems that arise when robot individuals are considered.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

James M. Hereford and Michael A. Siebold (2010). Bio-Inspired Search Strategies for Robot Swarms, Swarm Robotics from Biology to Robotics, Ester Martinez Martin (Ed.), ISBN: 978-953-307-075-9, InTech, Available from: http://www.intechopen.com/books/swarm-robotics-from-biology-to-robotics/bio-inspired-search-strategies-for-robot-swarms

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen