Turkish Journal of Electrical Engineering and Computer Sciences

Volume 15 | Number 2 Article 6

1-1-2007

# Aggregation in Swarm Robotic Systems: Evolution and Probabilistic Control

ONUR SOYSAL

ERKİN BAHÇECİ

EROL ŞAHİN

Turk J Elec Engin, VOL.15, NO.2 2007, © TÜBİTAK

# Aggregation in Swarm Robotic Systems: Evolution and Probabilistic Control

**Onur SOYSAL, Erkin BAHÇECİ and Erol ŞAHİN**
*KOVAN Research Lab., Department of Computer Engineering, Middle East Technical University, 06531, Ankara-TURKEY*
*e-mail: {soysal,bahceci,erol}@ceng.metu.edu.tr*

**Abstract**

*In this study we investigate two approachees for aggregation behavior in swarm robotics systems: Evolutionary methods and probabilistic control. In first part, aggregation behavior is chosen as a case, where performance and scalability of aggregation behaviors of perceptron controllers that are evolved for a simulated swarm robotic system are systematically studied with different parameter settings. Using a cluster of computers to run simulations in parallel, four experiments are conducted varying some of the parameters. Rules of thumb are derived, which can be of guidance to the use of evolutionary methods to generate other swarm robotic behaviors as well. In the second part a systematic analysis of probabilistic aggregation strategies in swarm robotic systems is presented. A generic aggregation behavior is proposed as a combination of four basic behaviors:* obstacle avoidance, approach, repel, *and* wait. *The latter three basic behaviors are combined using a three-state finite state machine with two probabilistic transitions among them. Two different metrics were used to compare performance of strategies. Through systematic experiments, how the aggregation performance, as measured by these two metrics, change 1) with transition probabilities, 2) with number of simulation steps, and 3) with arena size, is studied. We then discuss these two approaches for the aggregation problem.*

## 1. Introduction

Aggregation is one of the fundamental behaviors of swarms in nature and is observed in organisms ranging from bacteria to social insects and mammals [1]. Aggregation helps organisms to avoid predators, resist hostile environmental conditions and find mates. Some of the aggregation behaviors are known to be facilitated by environmental clues; flies use light and temperature, and sow bugs use humidity for aggregation. However, other aggregations are self-organized. Aggregation of cockroaches, young penguins and fish schools don't use such clues but are rather result of emergent cooperative decision.

This study focuses on the self-organized aggregation behaviors for swarm robotic systems [2, 3]. This behavior is required to form a robot cluster, where robots in the cluster is in close proximity, from any distribution of robots. Swarm robotics use simple and incapable robots and these robots must cooperate to accomplish tasks. Usually cooperation requires being in some proximity with other robots. Additionally, aggregation is a requisite for swarm robotic behaviors, such as self-assembly and pattern formation.

In this study, aggregation of a swarm of robots in a bounded arena is investigated. The robots in this study have limited perception range, through ambiguous and noisy sensors. We systematically evaluated two approaches for aggregation in swarm robotic systems. First method which extends [4] uses evolution to generate controllers whereas the second method [5], relies on a probabilistic controller. We investigate performance of genetic algorithms on aggregation problem with respect to optimization parameters. Similarly, the probabilistic controller is systematically analyzed with respect to probabilistic controller parameters. We then compare and discuss the results obtained from these two studies for a more general understanding of aggregation.

Aggregation studies in the literature come from different disciplines: biology, robotics and control theory. In [6], Deneubourg *et al.* investigate aggregation behavior in weaver ants and cockroaches. Weaver ants aggregate in chains to abridge gaps and cockroaches aggregate together in hiding sites. In these species, individuals rest in aggregations for varying time spans. Results indicate that the amount of time individuals spend in aggregations are modulated by environmental conditions and presence of other individuals. Individuals tend to spend more time in large aggregations, providing positive feedback for growth of aggregations. Individuals also spend more time on favorable sites, causing larger aggregates to form on such sites. Using this simple scheme individuals are able to make collective decisions.

Jeanson *et al.*, in [7], present a model of aggregation behavior of cockroach larvae in homogeneous conditions. The aggregation behavior observed in this species, include wall following, and two different resting behaviors. Individual behavior is modeled through systematic experiments. It is observed that the behavior of individual cockroaches depend on the number of cockroaches in close vicinity. A model of this behavior is used in robotic experiments to obtain similar behavior in a group of Alice robots (K-Team, Switzerland).

One of the early robotics applications of aggregation behavior is done by Melhuish *et al.* [8]. In this study, robots are required to form clusters of predetermined size around infrared beacons. Similar to the sounds produced by birds and frogs the method proposed uses chorus consisting of individuals where individuals try to produce sound simulatenously. However, individuals have small variations in elicitation of sound. Using these variations in sound elicitation, individuals can approximate the size of the clusters. This study has also been tested on systems without infrared beacons that trigger aggregation. Results indicate it is only possible to obtain this kind of self organized aggregation behavior, in virtually noiseless environment.

In control theory aggregation is often referred as gathering, agreement or rendezvous. These studies usually consider abstract models of robots with varying detail in modeling. Most of the studies [9, 10] in this approach ignore the dynamics of the robots, representing robots as points without orientation. Even though some studies use kinematic models of robots, use of rigid body physics is not common practice.

Another important assumption in these studies is the limit on perception range. Some studies in aggregation consider the infinite visibility case where robots are able to perceive all robots. Under the unlimited visibility assumption, strong results on aggregation such as explicit bounds on the swarm size and bounds on the time of convergence can be obtained [11, 12, 13]. This approach is quite powerful for theoretical analysis however, unlimited perception range is an unrealistic assumption for real robotic swarms.

Studies indicate, when perception range of individuals is limited, there are some required conditions for aggregation. One of these conditions is defined using a visibility graph, which is constructed by robots as nodes and by edges between robots that are able to perceive each other. In order for a deterministic control algorithm to allow aggregation, this graph needs to have at least one node which is accessible from all other nodes in the graph [14]. Flocchini *et. al* noted that, even convergence may not be enough since convergence may take infinitely long time [9]. They proposed a control algorithm that requires only limited visibility with distinguishable robots and it guarantees aggregation in finite time.

Another study by Gordon *et. al* investigate gathering of agents with asynchronous distributed control [10]. In this study, theoretical convergence is supported with simple kinematic simulation of robots modeled as point bodies. Asynchronous nature of the model introduces a randomness in the behavior of agent clusters. Agent clusters move randomly due to unordered motion of agents. These random motions allow, to a degree, the system to aggregate when necessity conditions are not satisfied.

Importance of strong theoretical support is emphasized in *Swarm engineering approach* by Sanza T. Kazadi [15]. This approach defines the global goals as mathematical constraints. Behaviors are then synthesized to satisfy these constraints. The behavior of system can be investigated using the goal constraints. Lee *et. al* apply this concept to robot aggregation in their recent work [16]. Their implementation of aggregation behavior allows robots to approximately estimate the size of the robot aggregation. Using this information and the analysis of puck clustering problem where similar properties exists, implemented aggregation behavior produces clusters with increasing size.

## 2. Evolutionary Robotics

### 2.1. Artificial Evolution

Evolutionary computational methods are inspired by the natural evolution. In nature, a population of animals struggle to survive and reproduce to produce the next generation. The principle of "survival of the fittest" applies: individuals that are fitter within their environments are more likely to survive and also more likely to produce offspring, transferring their genetic material onto the next generation. In this way, nature eliminates weak individuals and the population gets more adapted to the environmental conditions generation by generation.

John H. Holland began publishing on adaptive systems theory in 1962 [17] and wrote his book on *Genetic Algorithms* in 1975 [18], which mimics natural evolution and is basically adaptation of a *population* of candidate solutions for a problem with the use of genetic operators such as *selection*, *mutation* and *crossover*.

Genetic algorithms work with a set of candidate solutions rather than a single one as in other optimization methods. The encoded candidate solution is called a *chromosome* and the set of solutions is called a *population* analogous to a set of animals in a population, each of which are *encoded* in a single DNA molecule.

The genetic algorithm needs a way to evaluate the *goodness* (or *fitness*, as widely used) of a candidate solution. For the function minimization example, this would simply be evaluation of the function with the value of the variable that is the candidate solution. The smaller the result, the better the solution.

The whole population of candidate solutions is evaluated in this manner. Depending on their fitness values, the population is sorted and a subset of the population is *selected* among the better ranking individuals. This selected set is then used to produce the new population, i.e., the population of the next *generation*. It is this part of the Genetic Algorithm that implements the *survival of the fittest* principle. The new population, created through some genetic operators such as *crossover* and *mutation*, is expected to have higher fitness values.

*Crossover* or *recombination* is applied to the selected set of individuals, with a certain probability. Crossover swaps parts of two chromosomes, i.e., pairs from the selected set, where it can be applied at one point on the chromosomes or on multiple points. Its main purpose is to join two *useful* segments of two different chromosomes in one chromosome, where the resulting chromosome has more *useful* parts than the two initial chromosomes. This does not always happen, but when it happens often enough, it will result in a better performing population through improved individuals.

After crossover operations, individuals in the population are subjected to the *mutation* operator with a small probability. Mutation is used to alter a small portion of the chromosome at a random position. It helps in creating individuals that are randomly and slightly perturbed versions of the previous populations. In short, crossover combines solutions whereas mutation generates new ones.

Use of genetic operators such as crossover and mutation improves chances of introducing more fit individuals, however these operators may also destroy some highly fit ones. To overcome this disadvantage, a fraction of the top ranking individuals in the population may be transferred to the next generation without applying crossover or mutation. This helps preserving the best chromosomes, and usually accelerates evolution. This modification is called elitism and is commonly used in studies using genetic algorithms.

The encoding of a candidate solution and the fitness function is specific to the problem at hand. Furthermore, the crossover and mutation operations can be defined to suit the chromosome encoding. For example, random bits in the chromosome encoding can be altered or if the encoding consists of a set of numbers, the value of a randomly chosen one can be increased or decreased by a random amount.

## 2.2. Use of Artificial Evolution in Swarm Robotic Systems

Early studies on evolving behaviors for swarm robotic systems reported limited success and expressed pessimistic conclusions. In one of the earliest studies, Zaera *et al.* [19] used evolution to develop behaviors for dispersal, aggregation, and schooling in fish. Although they had evolved successful controllers for dispersal and aggregation; the performance of the evolved behaviors for schooling was considered disappointing, and they concluded that for complex actions like schooling, manual design of a controller would require less time and effort than evolving one, mainly due to the difficulty of determining a useful evaluation function for the specific task.

Mataric *et al.* [20] have made a comprehensive review of the studies until 1996 on evolving controllers to be used in physical robots and they have discussed the key challenges. They addressed approaches and problems such as evolving morphology and/or controller, evolving in simulation or with real robots, fitness function design, co-evolution, and genetic encodings. They emphasized that for an evolved controller to be beneficial, the effort to produce it in evolution should be less than the effort needed to manually design a controller for the same robotic task. They stated that it has not been the case, yet; but when the challenges and problems are handled, they may become a practical alternative to controllers designed by hand.

In [21], Lund *et al.* used evolution to develop minimal controllers for exploration and homing task. They evolved controllers for the Khepera robot (K-Team, Switzerland) for the task considered where the robot was desired to leave a light source, i.e., home, explore the surrounding for some time, and then return back home where it is virtually recharged. To obtain this periodic behavior, they used sampled sensory input and a minimal network architecture without recurrent connections, which can be used to obtain the notion of *return period*. Instead their evolution exploited the geometrical shape as perceived by robot and produced a suitable controller.

In contrast to some of these pessimistic conclusions, during the recent years optimistic results are being reported on the evolution of swarm behaviors. In the Swarm-bots Project [22], Baldassarre *et al.* [23] successfully evolved controllers for a swarm of robots to aggregate and move toward a light source in a clustered formation. Moreover, for this specific task, several distinct movement types emerged: constant formation, amoeba (extending and sliding), and rose (circling around each other). In [22], Trianni *et al.* also evolved successful controllers for a swarm of robots that can grip each other, called a swarm-bot, to fulfill tasks such as aggregation, coordinated motion in a common direction, cooperative transport of heavy loads (as in ants), and all-terrain navigation to avoid holes (connected in swarm-bot formation). Their evolved controllers made use of sound sensors, traction sensors, and flexible links. Trianni *et al.* [24] has also identified two types aggregation behaviors emerged from evolution: a dynamic and a static clustering behavior. In static clustering, robots move in circles until they are attracted to a sound source. Then they *bounce* against each other until an aggregate is formed. The clusters are tight and static with the robots involved turning on the spot, whereas in dynamic aggregation, the clusters are loose and they flock around. This study is a good example of evolution of different strategies, or behaviors, for a specific task. Furthermore, in [25] Dorigo *et al.* evolved aggregation behaviors for a swarm robotic system. They analyzed two of the evolved behaviors and showed that evolution was able to discover rather scalable behaviors.

Ward *et al.* [26] have evolved neural network controllers for such a survival scenario where there are two populations of animals, predators and preys that co-evolve to produce a schooling behavior. They have also studied on the connection of physiology with behavior and they claim that prey need a wide-range low-resolution visual sensors whereas predators are better off with visual input concentrated in the front.

Despite these studies, the use of artificial evolution to generate swarm robotic behaviors for a desired task is a rather unexplored field of study. The effort in using evolutionary methods can be reduced by suggestions on choosing parameters required in applying artificial evolution to swarm robotics. To the best of our knowledge, no systematic study has been made to investigate effects of parameters to help such choices. In this study, we addressed this lack of systematic analysis studies to deduce some rules of thumb on the choice of some parameters used in evolution of swarm robotic behaviors.

## 3. Simulation

In this study, a port[1] of MISS, a cut-down version of the Swarmbot3D simulator [27] is used. Swarmbot3D is a physics-based simulator developed within the Swarm-bots project that modeled the s-bots (mobile robots with the ability to connect to each other). Swarmbot3D simulator includes simulation models of the s-bot at different levels, all obtained from and verified against the actual s-bot. As Mataric *et al.* mentioned [20], evolving controllers for physical robots in simulation requires modeling of noise and error models to maximize transferability of controllers onto physical systems. This is ensured in this simulator with the sensor models implemented with sensory data coming from the physical s-bot. The minimal s-bot model of Swarmbot3D simulator is used here, with which evolution of aggregation behavior was first studied by Dorigo *et al.* in [25]. A snapshot of the simulator is shown in Figure 1.

### 3.1. Robot Architecture

A schematic view of the robot indicating the sensor and signal source configuration used in our experiments is shown in Figure 2. The robot is modeled as a differential drive robot with two wheels. The robot has 15 infrared range sensors around the robot, and one omni-directional speaker and 4 directional microphones placed at the center of the robot. Evolutionary model uses 8 of the range sensors to reduce complexity of neural controller and these sensors are indicated in Figure 2(a).

[1]In Kovan Research Lab, we ported MISS and Swarmbot3D simulators from Vortex, a commercial physics development platform, to the Open Dynamics Engine (ODE), a free physics-based simulation library. Extensions to ODE were done to add XML file loading capabilities, to improve rendering and camera handling, which were packaged under the name Kovan ODE eXtensions (KODEX). Using KODEX, converting Swarmbot3D from Vortex to ODE was possible with little effort.

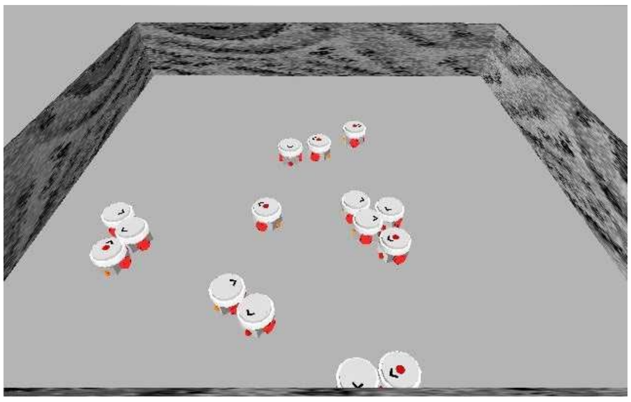


**Figure 1**. A screenshot of the simulator.

## 3.2. Robot Controller for Evolutionary Approach

In the experiments performed, robots act reactively depending only on their inputs. The controller is chosen to be a single-layer perceptron which has 12 input neurons (4 connected to microphones and 8 connected to infrareds), 3 output neurons (1 to control the omni-directional speaker and 2 to control the wheels) as seen in Figure 3.

## 3.3. Robot Controller for Probabilistic Approach

Probabilistic aggregation behavior used is implemented as a combination of three basic behaviors and obstacle avoidance. These behaviors are arranged in two layers according to the subsumption architecture as shown in Figure 4.

The higher-level behaviors are arbitrated using a finite state machine (FSM) with probabilistic transitions as shown in Figure 4 to implement a class of aggregation behaviors. At each state, the corresponding behavior becomes active. The robot initially starts in the *approach* state. In this state, the robot approaches the largest robot cluster in its view and switches to the *wait* state when it satisfies the “robot close” condition. The “robot close” condition is signaled when a robot can be perceived using infrared sensors. During the *wait* state, the robot picks a random number uniformly within the range $[0-1]$ at each time step. If this number is larger that $P_{leave}$, then the robot switches to the *repel* state. Otherwise, the robot remains in the *wait* state. Similarly, when the robot is in the *repel* state, with probability $P_{return}$ the robot switches back to the *approach* state.

In the lower level, an *obstacle avoidance* behavior is implemented, which becomes active when the values of infrared sensors become larger than a fixed threshold. This threshold is chosen as 10% of the range of IR sensor values, derived from the investigation of sample data used of IR modeling. Without this behavior robots get stuck on the walls and other aggregates.

We now introduce *soundThreshold* to allow robots to deal with noise inherent in sensors and to reduce jaggy behavior. This also allows robots to explore arena when they can not perceive any other robot. *avoidThreshold* on the other hand controls when the obstacle avoidance will suppress other behaviors. Here also note that, in *wait* state, obstacle avoidance is not activated, since without this modification, robots can not wait close to other robots.

```
ERROR: undefined
OFFENDING COMMAND:
STACK:
```

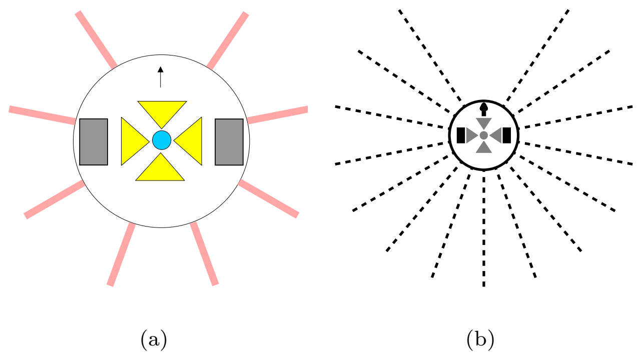


**Figure 2**. A schematic view of the robot model. The robot has a diameter of 5.8 cm. The dashed bars emanating from the body of the robot indicate the IR sensor direction and range. The 4 triangles are placed at the center represent microphones, 2 rectangles at the sides represent wheels, and the circle at the center represents an omni-directional speaker. Figure 2(a) shows the model for evolutionary runs and Figure 2(b) shows the model for probabilistic approach.

Robots use raw sensor values with their relative headings to approximate the relative direction of robot cluster. For this purpose, at each time step, a vector is constructed for each sound sensor, with relative direction same as the heading of the sound sensor and magnitude equal to the value of sensor at that time step. These four vectors are added to approximate direction of the cluster.

The robots try to minimize the angle between desired direction, which is described by either *Sound-Vector* or *obstacleVector*, and their heading. In order to reduce dead locks and allow smoother movement, robots turn in place until the angle is less than $\frac{\pi}{3}$. When the angle is less than $\frac{\pi}{3}$, robots both move forward and turn toward the desired vector.

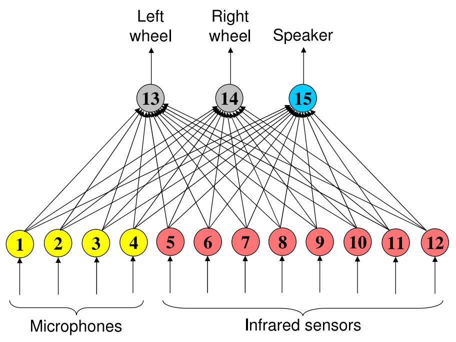


**Figure 3**. Neural network controller used as the controller for robots. Neurons match corresponding parts in Figure 2(a) as follows: 1-4: microphones, 5-12: IR sensors, 13-14: wheel actuators, and 15: speaker.

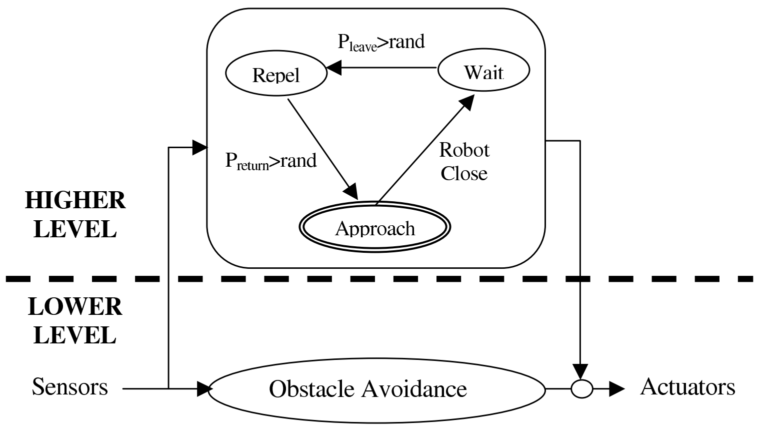


**Figure 4**. Minimal aggregation behavior. Ovals display the behaviors and arrows represent the behavior transitions.

### 3.4. Sensor Specifications

The details of the sensor models are described in detail in [27]. The infrared sensors are modeled using sampling data obtained from the real robot with the addition of white noise as described in [23] and [27]. As it is, the sound sensor model can be regarded as unrealistic due to its simplicity. However, using a proper placement of microphones robust sound source localization can be done as in [28], where Valin *et al.* has localized sound sources with a precision of 3 degrees in 3 meters range using an array of 8 sound sensors placed at the corners of a rectangular prism.

However, it should be noted that our simulator was neither verified against the original Swarmbot3D simulator, nor against the physical robots. Therefore, we make no claims about the portability of the evolved controllers onto the physical robots.

## 4. Evolution Experiments

Regarding evolutionary methods in developing controllers for swarm robotic studies, there are some parameters that should be considered, such as the number of generations, the number of simulation steps used for fitness evaluations, number of robots, and size of arena. We performed four experiments that altered some parameters and compared the results for different choices of parameter values.

An evolution experiment suite consists of an evolution and a scalability evaluation for each choice of parameters. The best evolved controller is analyzed for its performance in different sized set-ups to measure *scalability*. Since this study aims to derive rules of thumbs for evolutionary robotics, the credibility of results regarding the choice of parameter values should be as high as possible. To accomplish this, 4 evolution suites with different random seeds are carried out for each parameter value choice, extending [4], and the results were combined by averaging.

### 4.1. Genetic Algorithm and Evolution Scheme

The controller used in these experiments is a single layer perceptron with 12 inputs and three outputs. With additional bias terms, the perceptron is encoded with 39 floating point numbers corresponding to the weights in the perceptron. The genetic algorithm used runs with a population of size 50 chromosomes initialized randomly. Each population member or individual encodes a controller which is evaluated for its fitness. Then the controllers are sorted in descending order. The new population consists of an elite group from the current population, plus new controllers obtained by applying crossover (with 0.8 probability) and mutation to a group selected with tournament selection. Mutation is done by choosing one of 39 weights and adding a random value uniformly in $[-1.0, +1.0]$ range. Each chromosome is subjected to this type of mutation with a probability of 0.5, i.e., $\frac{0.5}{39}$ per weight.

For the genetic algorithm to function correctly, the chromosomes should be evaluated for their fitness, i.e., in our case, how good the encoded controller performs aggregation. Aggregation quality can be assessed in several ways. One way is to compute sum of the distances of each pair of robots. This measure gives smaller values as the robots get closer to each other. However, we chose another aggregation measure, which counts the robots in the formed clusters and computes the fitness as the size of the largest cluster with respect to the whole group, because this measure is a direct method of calculating what percent of the robots have clustered together.

In order to do the evaluation, the perceptron defined by that particular chromosome is replicated as the controller for all the robots in the swarm, and the swarm robotic system is simulated for a certain number of steps. At the end of a simulation run, sizes of clusters are computed. This is done as follows.

Robots $i$ and $j$ are referred to as neighbors if the $Neighbor(i,j)$ relation, defined in Figure 1, is true. Also, the two robots are in the same cluster, or aggregate or group, if the $Connected(i,j)$ relation, defined in Figure 2, is true. $Connected(i,j)$ is actually the *transitive closure* of the $Neighbor(i,j)$ relation. Transitive closure is computed using Warshall's algorithm, which has $O(n^3)$ complexity over the number of robots [29].

$$Neighbor(i,j) = \begin{cases} true & \text{if distance between} \\ & i \text{ and } j \leq 4 \text{ cm} \\ false & \text{otherwise} \end{cases} \tag{1}$$

$$Connected(i,j) = \begin{cases} true & \text{if there is a path from} \\ & i \text{ to } j \text{ over the} \\ & \text{relationship } Neighbor \\ false & \text{otherwise} \end{cases} \tag{2}$$

Using transitive closure, each robot is assigned to a cluster, while calculating the size of clusters. The primary purpose of this algorithm is to determine the largest cluster $l$. The aggregation performance, or $fitness$, of a single evaluation run is defined as $\frac{size(l)}{n_{robots}}$, i.e., ratio of size of the largest cluster to the number of all robots, where $size(c)$ is the number of robots in cluster $c$.

Different initial positions of robots in the arena lead to a significant bias for the resulting aggregation performance. Therefore, a fair evaluation of different controllers requires multiple performance evaluation simulations per controller, each starting with a different random initial placement. The number of simulations per controller will be called $n_{runs}$ from now on.

The fitness of a chromosome is defined as in Figure 3.

$$Fitness = F_{combine}(fitness_1, ..., fitness_{n_{runs}}) \tag{3}$$

where $F_{combine}$, *fitness combining function*, is used to join the fitness values of $n_{runs}$ simulation runs done for a single chromosome. These runs differ in their randomization seed, which determines the initial placement of robots. $fitness_i$ in this equation refers to the fitness value of a simulation run with the $i^{th}$ random seed. In this study, the $F_{combine}$ function is one of the parameters altered in experiments and is chosen among the following functions: *average*, *median*, *minimum*, and *maximum*.

## 4.2. Arena Set-ups for Evolution

Simulations involve robots that are initially randomly distributed in a closed square arena. Arena sizes that are used in the evolutions are $110 \times 110$ cm, $140 \times 140$ cm, $200 \times 200$ cm, and $282 \times 282$ cm. Initial positions and orientations of robots are random and are determined using the random seed coming from the genetic algorithm.

## 4.3. Scalability Evaluation

During scalability analysis, each evolved controller is tested with 50 different seeds on 5 different set-ups, which are all set-ups used for evolution plus a $400 \times 400$ cm arena, shown in Figure 1. In all these set-ups the robot density over the arena is kept the same. The number of simulation steps is increased in larger arenas to allow more time for aggregation.

**Table 1**. Set-ups used for scalability evaluation.

| set-up | # Robots | Arena Size | # Simulation Steps |
|---|---|---|---|
| **1** | 3 | $110 \times 110$ | 3000 |
| **2** | 5 | $140 \times 140$ | 6000 |
| **3** | 10 | $200 \times 200$ | 9000 |
| **4** | 20 | $282 \times 282$ | 12000 |
| **5** | 40 | $400 \times 400$ | 15000 |

The results, i.e., final largest cluster ratio of robots, obtained from the 50 runs are averaged and yield the result for a single controller and a single evaluation set-up. We had mentioned 4 different evolutions for each parameter value choice. Each one of the 4 controllers produced by these evolutions are evaluated for their scalability in the same manner, on the 5 different evaluation set-ups. The results are plotted for each evaluation set-up and each different parameter value choice averaged over the 4 evolution suites. The *scalability metric* we use is simply a vector of 5 numbers that are the average fitness values for 5 different scalability set-ups.

The numbers given above mean 50 *runs* $\times$ 5 *set-ups* $\times$ 4 *evolutions* = 1000 *evaluation simulation runs* for each parameter value choice and $1000 \times 4$ *parameter alternatives* = 4000 *total scalability evaluation runs* in a single experiment. As one can realize, this is a very large amount of computation for a single computer to overcome. Hence, just like we parallelized fitness evaluations of the genetic algorithm, we also distributed these simulation runs on the cluster of ULAKBİM.

## 4.4. Experiments and Results

Our main purpose in the experiments is to derive hints, in evolutionary swarm robotics, to how the available limited processing time should be utilized for evolving a desired robotic behavior. Should the available time be given to more generations, to more runs per controller, or to longer simulations for fitness evaluation? Choices of some parameters that do not affect total evolution time were also investigated.

With the experimental framework described in the previous section, we conducted four experiments to investigate the effect of different parameters on performance and scalability of evolved aggregation behaviors. Parameters altered in the experiments can be seen in Figure 2. Due to long computation times for simulations, for each parameter, a limited number of values could be investigated, which can be seen in Figure 3. Different ranges and more values for parameters could also be considered to extend the results obtained in this study. In the first three experiments total number of simulation steps used in evolution was kept constant. This roughly corresponds to keeping the total amount of processor time constant[2]. In this way we try to investigate the trade offs between length of runs and number repetitions in terms of performance.

**Table 2**. Parameters altered in evolution experiments.

| **Parameter Name** | **Parameter Description** |
|---|---|
| $F_{combine}(\cdot)$ | Fitness combining method |
| $n_{runs}$ | Number of simulation runs per controller |
| $n_{steps}$ | Number of simulation steps |
| $n_{gens}$ | Number of generations in evolution |
| Set-up Size | Set-up size in terms of number of robots, arena size, and number of simulation steps |

**Table 3**. Investigated parameters in evolution experiments.

| **Experiment** | **Investigated parameters** |
|---|---|
| 1 | $F_{combine}(\cdot)$ |
| 2 | $n_{steps}$ vs. $n_{runs}$ |
| 3 | $n_{gens}$ vs. $n_{runs}$ |
| 4 | Set-up Size |

The first experiment considered effects of changing a single parameter, while the second and third experiments specifically investigated trade-offs between the parameters shown in Figure 2 and the fourth experiment altered set-up size used in evolution. In the trade-off experiments, the aim was to answer the set of questions “If I have parameters that cause longer evolutions when increased, should I use the limited amount of CPU time for increasing parameter $A$ and decreasing parameter $B$, or vice versa?”

### 4.4.1. Experiment 1: Fitness Integration Method

Different initial positions of robots in the arena creates a large bias for the resulting performance and therefore a fair evaluation of different controllers requires multiple performance evaluations each starting from a different random initial placement. There are standard deviations that can be as high as 24% (Figure 5(a)) among fitness values obtained with the same controller. So the approach we choose to combine these values can influence the course of evolution and performance of resulting controller significantly.

[2]Roughly, because two simulations that run for $S$ simulation steps do not necessarily have equal execution durations due to longer execution when there are more collisions in simulation.

**Table 4**. Constant and variable parameters in the experiments are shown. Variable parameters are given as different values that the variable is assigned. Experiments 2, 3, and 4 used median as the $F_{combine}$ function, because initial trials had shown that median caused slightly better performance. However, this was observed to be incorrect when all results of experiment 1 were obtained as seen in Figure 4.4.1.

| **Exp.** | $n_{robots}$ | **Arena size** | $n_{steps}$ | $n_{runs}$ | $n_{gens}$ | $F_{combine}$ |
|---|---|---|---|---|---|---|
| 1 | 10 | 200×200 | 6000 | 5 | 50 | avg., median, min., max. |
| 2 | 10 | 200×200 | 18000, 6000, 3600, 1800 | 1, 3, 5, 10 | 50 | median |
| 3 | 10 | 200×200 | 6000 | 1, 3, 5, 10 | 150, 50, 30, 15 | median |
| 4 | 5, 10, 20 | 140×140, 200×200, 282×282 | 6000, 9000, 12000 | 3 | 50 | median |

We start with $n_{runs}$ different fitness evaluations for a controller, each obtained from different initial robot positions. The combination of the these evaluations for the controllers are accomplished through *fitness combining function*, $F_{combine}$.

The first experiment is motivated by the question of how should the $F_{combine}$ function be chosen to obtain the best results, among the *average*, *median*, *minimum*, and *maximum* functions.

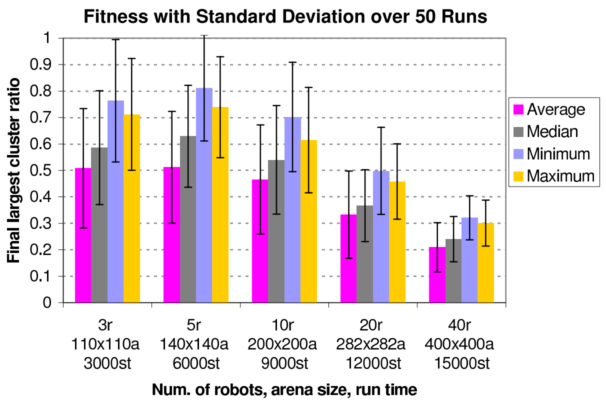
The results seen in Figure 5(a) indicate that the four functions are sorted according to their performance in the order *minimum*, *maximum*, *median*, and *average*. Each one dominates, i.e., is better in all evaluation set-ups than, the next one in this order. The *minimum* function, which corresponds to pessimistic evaluation, is clearly the best of the four.

It is also worth mentioning that, considering the standard deviations among the scalability performances of 4 distinct evolutions (Figure 5(b)), the *minimum* function performed close in each of the 4 evolutions, i.e., showed small variance, as well performing the best. On the other hand, the *maximum* function, which is the second best in performance, showed significant variance among evolutions compared to the other functions. This means that although both seem to perform well, the *maximum* function is riskier to use in evolution than the *minimum* function. Also, it should be noted that functions that consider only extrema in fitness, i.e., *minimum* and *maximum* functions, performed better than the functions emphasizing all fitness values, i.e., *median* and *average* functions.

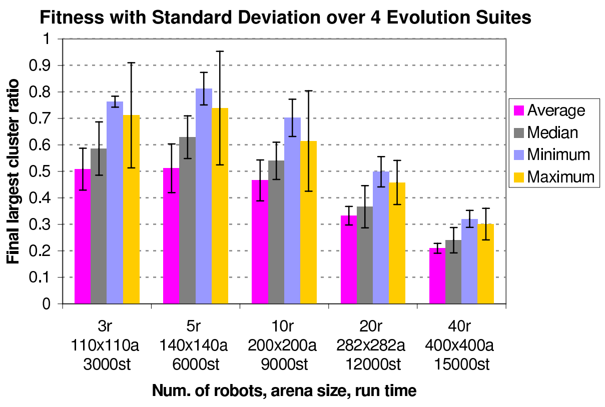
The behavior of one of the best controllers evolved in this experiment can be seen in Figure 6(a) and Figure 6(b). The evolved strategy can be described as "if no sound is heard then go straight; if there is wall then avoid it; if a sound is heard then approach the loudest sound source; but if the sound is very loud then turn on the spot". The emergent behavior of formed groups is to move slowly toward the loudest sound source. This can be seen in paths of groups in Figure 6(b), which are slowly going toward each other.

### 4.4.2. Experiment 2: Simulation Duration vs. Number of Runs per Controller

Choice of parameters that affect evolution time are hard to choose, since to alter one of them and to keep the total evolution duration constant, one needs to sacrifice another parameter. To shed light onto this situation, we considered a trade-off in the second experiment, between the number of runs per controller ($n_{runs}$) and the number of simulation steps in fitness evaluation ($n_{steps}$) while keeping the number of total simulation steps executed for a specific controller constant. Figure 7 shows a significant monotonous increase



(a)



(b)

**Figure 5**. Results of experiment 1. Different integrations of fitness values of the same controller are compared. The $y$-axis shows final largest cluster ratios, i.e., aggregation performance, whereas the $x$-axis designates 5 different set-ups used to evaluate scalability performance of produced controllers by the evolutions depicted on the legend. The evaluation set-ups increase in number of robots, size of arena, and number of simulation steps from left to right. For each of the 5 setups on x-axis, **(a)** shows $mean(mean$ over 50 runs) over 4 evolution suites with the error bars indicating $mean(std.dev.$ over 50 runs) over 4 evolution suites. **(b)** differs only in error bars, which indicate $std.dev.(mean$ over 50 runs) over 4 evolution suites.

in performance as $n_{runs}$ increases although while duration of simulation decreases. However, as $n_{runs}$ is increasing, performance increase is slowing down, which can be seen as decreasing gaps between the lines in the plot. This is probably due to decreased $n_{steps}$ in high $n_{runs}$ evolutions,

This implies that the number of runs for a controller is more important than the number of simulation steps up to a certain level, where the gain coming from high $n_{runs}$ is surpassed by the loss from low $n_{steps}$.

The degree of general drop in performance toward larger scalability evaluation arena set-ups in Figure 7 and the result plots of other experiments show the amount of deviation from perfect scalability, which would be shown as completely horizontal lines, i.e., no performance loss with bigger scales.

### 4.4.3. Experiment 3: Number of Generations vs. Number of Runs per Controller

Another trade-off we considered in total evolution time is between $n_{runs}$ and number of generations ($n_{gens}$) while keeping the number of total simulation steps in the whole evolution constant. Our third experiment investigates the choice of parameters in this trade-off. In the experimented values for the parameters, as $n_{runs}$ is increased $n_{gens}$ is decreased, so that the total number of simulation runs done in the evolution is constant.

The resulting performance and variance plots are shown in (Figure 8). The resulting scalability performances are rather close, which is pretty surprising because it shows that the decrease in performance by decreasing number of generations (as suggested by the fact that if elitism is used in a genetic algorithm, performance increases or stays the same at consequent generations) is compensated by the increase in performance caused by the increase in $n_{runs}$ (as shown in Experiment 2). Moreover, this balance of the two parameters seems to exist at the same parameter value ranges (that is $n_{gens} = [15, 150]$ and $n_{runs} = [1, 10]$) for all 5 set-ups of different size.

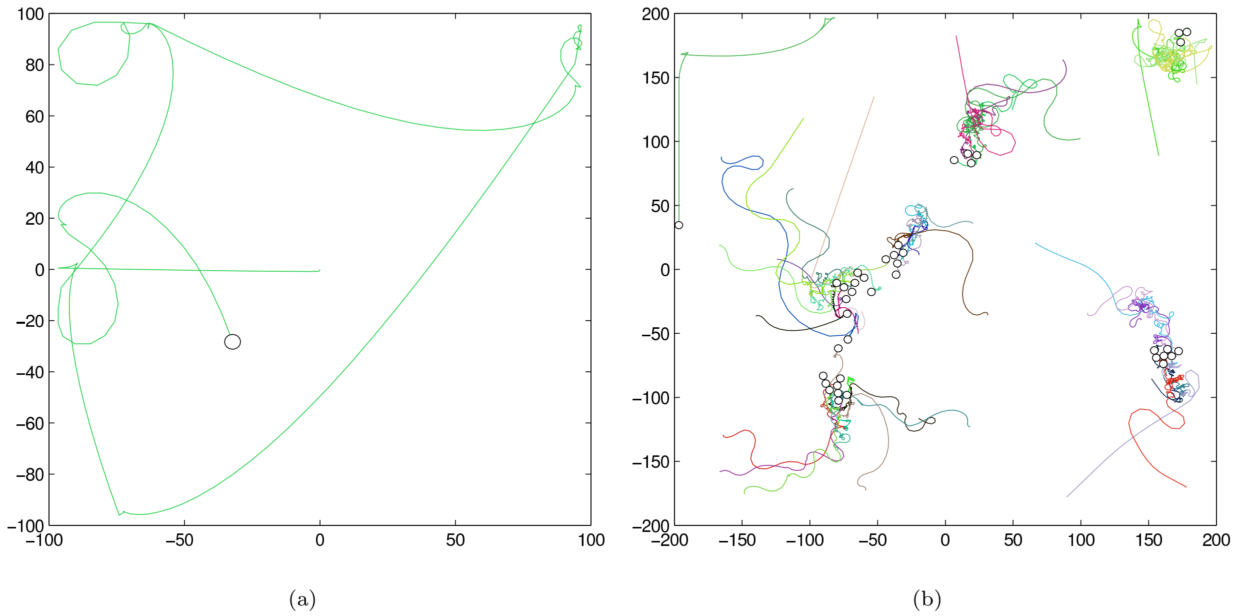


**Figure 6**. The behavior of an evolved controller with **(a)** a single robot in an arena of size $200 \times 200$ after 10000 time steps, and **(b)** 40 robots in an arena of size $400 \times 400$ after 15000 time steps. Final positions of robots are shown as circles together with the paths they followed during the whole simulation run.

### 4.4.4. Experiment 4: Set-up Size

Scalability is an important issue in swarm robotic systems, since it significantly affects the usefulness of a swarm robotic controller. Therefore, one might ask the question "how well does a controller, that is evolved using a set-up of certain size, perform on different sized set-ups?", or more generally "how does chosen set-up size in evolution affect the scalability performance of evolved controllers?". One would expect that each evolution produces a controller that runs best on its evolution set-up among controllers that are evolved on different sized set-ups. Is that really so?

As explained in earlier sections, in each evolution, controllers are evaluated for their fitness by running a simulation on a set-up of certain size (arena size, number of robots, and number of simulation steps combined). For evaluating the fitness of a controller, if we use multiple set-ups instead of a single one, would we be able to evolve a more *scalable* controller, i.e., a controller that performs higher in all different sized set-ups?

These questions were tackled in our fourth experiment. However, the type of scalability considered here does not involve solutions to *sparser* arenas, i.e., having lower *robot densities*, or shorter times to complete aggregation. If arena size and simulation duration were kept constant while number of robots was altered, the problem would change considerably. A sparser arena would mean that finding other robots would be a much more challenging task because of limited sensor ranges. The tactic required to aggregate in such an arena may be quite different from the one required in a less sparse one. It would also complicate the aggregation task if simulation duration was not increased together with number of robots. Aggregation in shorter times may require different strategies.

Therefore, the scalability we try to achieve is concerned with attempting the same task with different

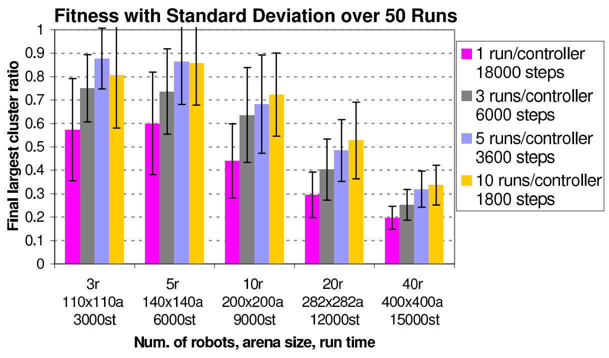


**Figure 7**. Results of experiment 2. The number of runs for the same controller and the number of simulation steps are varied while keeping total number of steps for a controller constant. Plot axes are the same as in Figure 5. For each of the 5 setups on x-axis, *mean*(*mean* over 50 runs) over 4 evolution suites is shown with the error bars indicating *mean*(*std.dev.* over 50 runs) over 4 evolution suites.

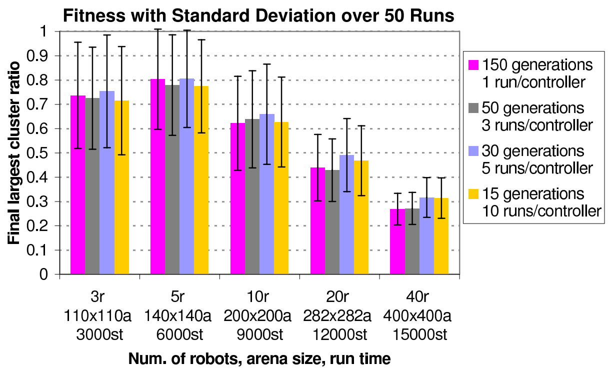
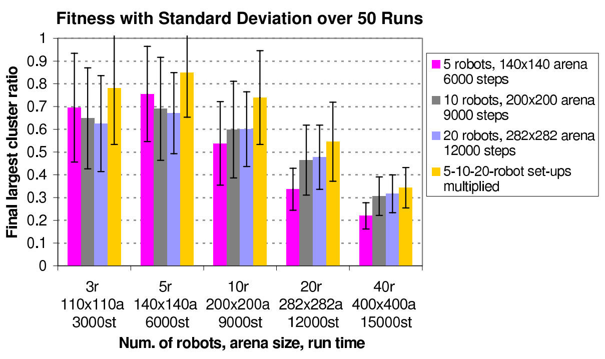


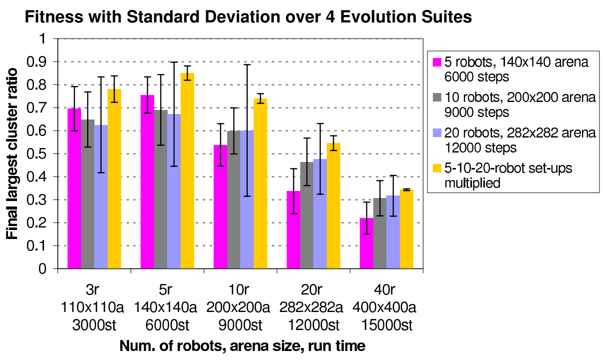
**Figure 8**. Results of experiment 3. The number of generations and the number of runs for the same controller are varied while keeping total number of steps constant. Plot axes are the same as in Figure 5. For each of the 5 setups on x-axis, *mean*(*mean* over 50 runs) over 4 evolution suites is shown with the error bars indicating *mean*(*std.dev.* over 50 runs) over 4 evolution suites.

number of robots, while also altering the arena size to keep the robot density constant, and also simulation duration to allow enough time for the robots to find each other and aggregate. Thus, this experiment investigates the effect of set-up size, i.e., number of robots, arena size, and number of simulation steps together, on performance and scalability. Unlike the first three experiments, which were conducted to find out how to use total processing time most effectively, this experiment does not keep the total number of simulation steps in the whole evolution constant. Instead it analyzes how good controllers evolved with different set-ups perform on smaller/larger set-ups. It investigates which set-up size leads to better scalability and also whether using all set-ups in one evolution (where calculating fitness a controller by multiplying simulation results on different set-ups) improves overall scalability.

The results in Figure 9(a) show that, as expected, the multiple set-up evolution performed the best among the five evolutions and displayed the highest scalability. Also, the evolution with the smaller set-up (5-robot set-up evolution) showed higher performance in smaller evaluation set-ups (3-robot and 5-robot set-ups), and lower performance in larger evaluation set-ups (10-robot, 20-robot, and 40-robot set-ups) than



(a)



(b)

**Figure 9**. Results of experiment 4, where evolution set-up size, i.e. number of robots, arena size, and number of simulation steps, is varied. Plot axes are the same as in Figure 5. For each of the 5 setups on x-axis, **(a)** shows *mean*(*mean* over 50 runs) over 4 evolution suites with the error bars indicating *mean*(*std.dev.* over 50 runs) over 4 evolution suites. **(b)** differs only in error bars, which indicate *std.dev.*(*mean* over 50 runs) over 4 evolution suites.

the evolutions with bigger set-ups (10-robot and 20-robot set-up evolutions).

When we look at Figure 9(b), we see that multiple set-up evolution is not only the best performing evolution, but also the one with the least "variance among 4 evolutions", hence the highest reliability among different evolutions. Furthermore, it is observed that 20-robot set-up evolution has the highest variance. This shows that if the 20-robot set-up is chosen for evolving controller, the resulting performance will be less predictable compared to evolutions on other set-ups. This may be caused by the size of the set-up, where even with a suitable controller for the task, 20 robots may end up at placements that have much more varying fitness the end of 12000 steps in a big arena with respect to smaller set-ups.

Among the three evolutions using only one set-up, each evolution produced a controller that performed best at its evolved set-up. The difference between the performance of a controller evaluated on the set-up that it is evolved on and performances of other two controllers becomes significant in the evaluation with the 20-robot set-up, where 20-robot set-up evolution scores much higher than 5-robot set-up evolution. However, the controller evolved in multiplied set-up evolution outperforms all others, even on their very own evolution set-ups. It is very interesting and can further be analyzed because the three evolutions get to have three evaluation simulation runs on their own set-up, while the multiplied set-up evolution gets to have only one on each of the three set-ups.

## 5. Probabilistic Experiments

The minimal aggregation behavior we used in this study has two probabilistic transitions, which govern the overall behavior of the robot. The first set of experiments reported in this section aims to understand the effect of these probabilities on aggregation.

Following the investigation of transition probabilities, progress of aggregation behavior is examined using four representative strategies, obtained from first set of experiments. This is followed by experiments to understand the effect of the arena size on aggregation performance, again on the four representative strategies. Increasing the arena size allows investigation of aggregation when visibility constraints mentioned in literature [14, 10, 9] are relaxed.

## 5.1. Aggregation Performance Metrics

Aggregation performance metrics are required to compare the performance of different aggregation behaviors. The difficulty of constructing aggregation metrics arises from the subjective and task depended nature of aggregation. Good aggregation for one task may not be as good for another task.

Considering the variety in requirements we introduce two different aggregation performance metrics. These metrics will be explained in detail in the following subsections.

### 5.1.1. Expected Cluster Size

*Expected Cluster Size* (ECS) metric aims to estimate the size of the cluster any robot belongs to after the aggregation algorithm is run on the swarm. We use the same neighbourhood and connectedness relations defined in Figure 1 and Figure 2 to calculate the cluster size for a robot, $Size(R_i)$, which is the number of robots in the cluster that robot belongs to. This metric calculates the average of cluster sizes for each robot in the swarm, where $n$ is the number of robots in the swarm.

$$ECS = \frac{1}{n}\sum_{i=1}^{n} Size(R_i) \tag{4}$$

This metric ignores spatial distribution of clusters, but gives a measure for size of cluster each robot belongs to. This approach is useful for applications where robots must maintain local links with other robots in a cluster.

### 5.1.2. Total Distance

The second metric, called *Total Distance* (TD), measures the total of distances between each robot pair. This metric gives more information about the spatial distribution of the swarm and clusters. This metric uses negative of distance to emphasize high metric value for better clustering. It is defined as follows:

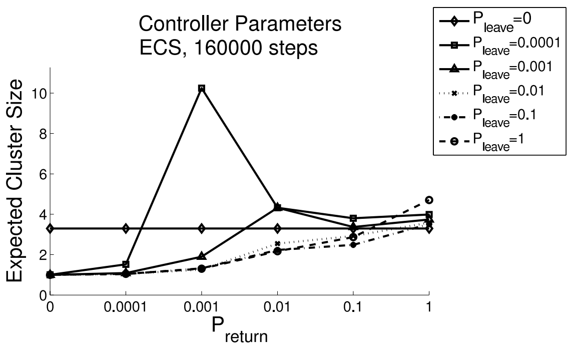
$$TD = -\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} dist(R_i, R_j) \tag{5}$$

This metric is useful when density of robots in an area is significant.

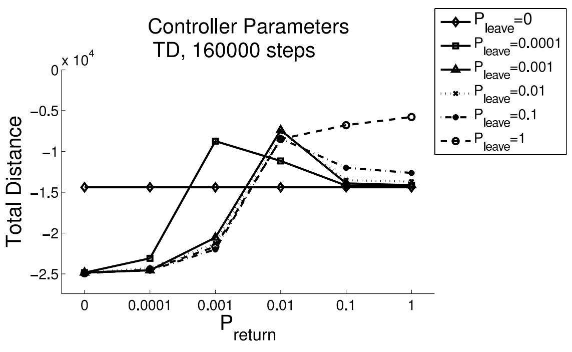
## 5.2. Experimental Setup

All experiments use a bounded square arena similar to the one shown in Figure 1. Bounded arena assumption is introduced to simplify the problem to a manageable size. The default size of this square arena is chosen to be large enough to allow visibility problems to arise, and small enough to be solvable by simple algorithms in some cases. Size of the standard arena is $200\ cm \times 200\ cm$.

Experiments are conducted in the simulator environment using 20 robots. Robots are able to move at a maximum speed of $0.3cm$ per simulation step. Fifty runs are made for each data point in the figures shown with different random initial placements. Performance metrics are evaluated for the positions of robots at the end of each run.



(a)



(b)

**Figure 10**. Effect of behavior transition probabilities. (a) On ECS metric values (b) On TD metric values.

## 5.3. Effect of Behavior Transition Probabilities

This part of the experiments aims to understand the effect of behavior transition probabilities $P_{leave}$ and $P_{return}$ to aggregation performance. These probabilities change the amount of time robots spend in behavior states. Figure 10 summarizes the effect of different $P_{leave}$ and $P_{return}$ probabilities in the standard arena. In this figure median performance for 50 runs of each pair of behavior transition probabilities is plotted. $P_{return}$ values change on $x - axis$ and different $P_{leave}$ values are on different series.

Figure 11 shows histogram of ECS metric values for aggregation behaviors. In this figure, each subplot is the histogram of ECS performance values for the 50 runs conducted for corresponding behavior transition probabilities. In this figure $P_{return}$ changes on horizontally and $P_{leave}$ changes vertically. In each histogram, ECS value increases from left to right and bars indicate the number of occurrences of the corresponding ECS value. These figures indicate a large variance in performance, esspecially for better performing parameter pairs.

In order to remedy this problem, the number of runs is increased and results have shown no significant improvement in variance. Median values in this case seems quite similar. This result points out that the number of runs is sufficient and the variance is inherent to the system.

Another important factor in the high variance, is the instantenous calculation of performance. Robots in this study are usually in motion, which introduces additional variance to the aggregation performance of group. By choosing different values for controller parameters, we constructed 4 strategies with different characteristics.

### 5.3.1. Strategy 1

When $P_{leave} \neq 0, P_{return} \neq 0$, the behavior has two competing dynamics determined by the transition probabilities. Increasing $P_{leave}$ increases the time spent by robots in repel state, thus reduces size of the robot clusters. But this also allows more robots to search for clusters, in turn increasing the chance of forming larger clusters.

Increasing $P_{return}$ on the other hand reduces the time robots spend in repel state. By this way, distance traveled by robot while changing clusters is also reduced. When robots can not move far away from robot clusters, more stable clusters are obtained.

To represent this strategy, $P_{leave}$ is chosen to be 0.0001 and $P_{return}$ is chosen to be 0.001. Aggrega-

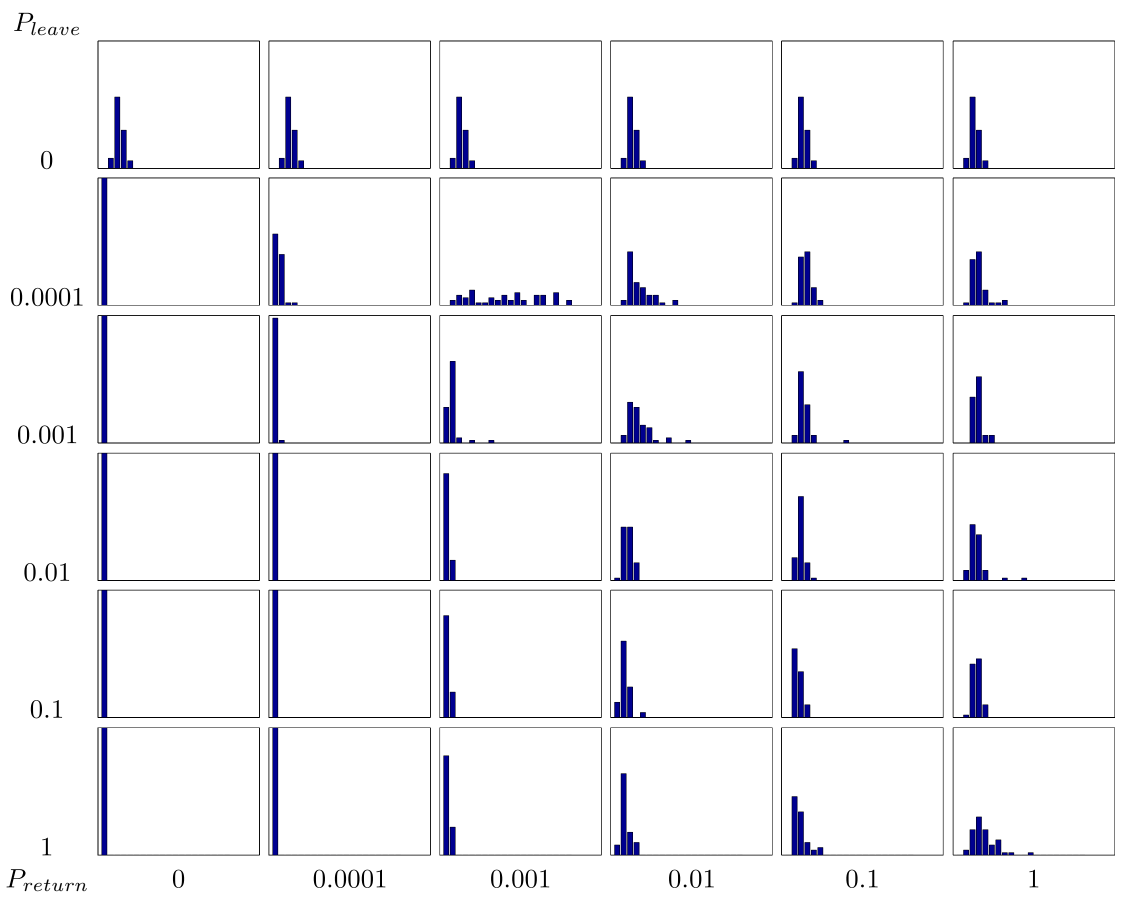


**Figure 11**. Histograms representing the distribution of ECS metric for different behavior transition probabilities. The $x - axis$ corresponds to ECS metric values and the bars represent the probability of obtaining corresponding ECS metric value for chosen controller parameters.

tion behavior using these probabilities is the highest performing value discovered in this study for the ECS metric and performs reasonably well with the TD metric. Figure 12(a) shows the FSM corresponding to this controller.

### 5.3.2. Strategy 2

When $P_{leave} = 0$, the FSM of the resulting behavior is reduced to Figure 12(b). In this case robots remain in *wait* state. This behavior can be summarized as: *Move toward sound sources and when close to a robot, stay there forever.*

In this strategy, when robot gets near another robot, the robot changes into *wait* state. Since the probability of switching to *repel* state is zero, the robot stays within the *wait* state forever. This strategy is deterministic, in the sense that robots never use probabilistic transitions of the behavior. This behavior leads to small "frozen" aggregations. Where robot clusters converge to locally formed clusters and stay in these clusters. It is also interesting to note that, the result of this strategy is only dependent on the initial distribution of the robots, since no random process is involved.

Due to the fact that robots being stuck in *wait* state, the expected cluster size for this strategy is

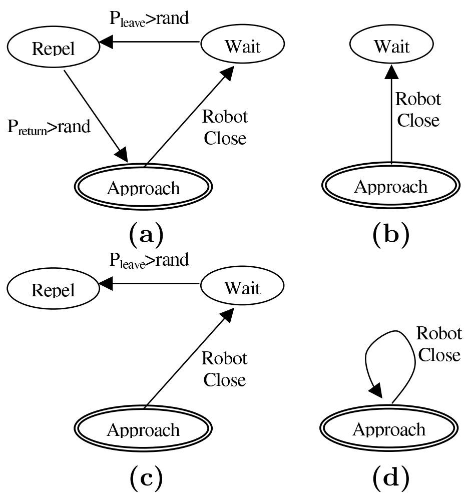


**Figure 12**. Finite State Machine representations of the representative strategies. (a) strategy 1, (b) strategy 2, (c) strategy 3, (d) strategy 4. See text for detailed description.

independent of $P_{return}$ and is considered as our baseline performance.

### 5.3.3. Strategy 3

When $P_{leave} \neq 0, P_{return} = 0$, the FSM of the resulting behavior is reduced to Figure 12(c). In this strategy, similar to Strategy 2, robots are stuck in a state: *repel.* Since this *repel* state causes robots to move away from sound sources, this behavior is really poor in aggregation performance. The behavior can be summarized as: *Move toward sound sources initially but then run away from sound sources forever.*

In this strategy, eventually, all robots are in the *repel* state, which means they are actively avoiding sound sources. This destroys all aggregations and leads to the worst aggregation performance.

An interesting point to note about this strategy is that, this behavior can accomplish another task that can be of interest to swarm robotic systems, *segregation.* In *segregation* task, robots are required to be as far as possible from each other, while maintaining visibility. Robots using this strategy can cover an area efficiently, similar to gas molecules, in a closed box.

### 5.3.4. Strategy 4

When $P_{leave} = P_{return} = 1$, the FSM of the resulting behavior is reduced to Figure 12(d). In this behavior, robots doesn't utilize the full potential of aggregation behavior. This time robots are stuck in *approach* behavior.

The strategy can thus be summarized as: *Move toward sound sources but don't stop; avoid robots like walls.* In this strategy robot never remains in states other than *approach*, since the probabilities of transition are one. This is equivalent to a fully reactive control with only *approach* and *obstacle avoidance.*

This behavior creates dynamic robot aggregations, exploiting the *obstacle avoidance* behavior. Robots approaching each other asynchronously and randomly cause a random drift in the robot clusters. When robots get too close, they try to avoid each other using *obstacle avoidance* behavior, moving in unpredictable directions. Robots are in constant motion in this strategy, as a result, the clusters they form are also in motion.

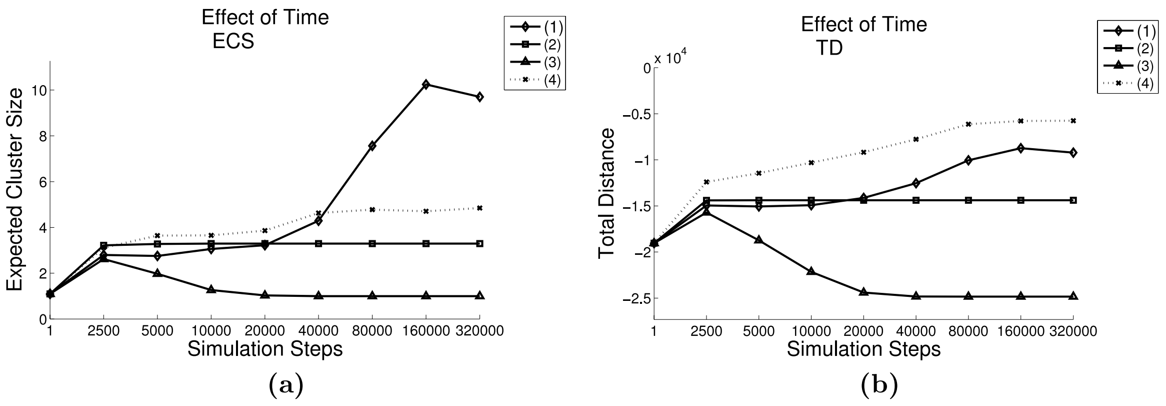


**Figure 13**. The effect of time on metric values for the four strategies. (a) shows median of metric values for ECS metric, (b) shows median of metric values for TD metric.

Although there is only a minor difference between this behavior and the behavior with $P_{leave} = 0.1$ and $P_{return} = 1$; the emergent behavior is quite different. In the latter case, robots use *wait* behavior and can't benefit from drift effect caused by obstacle avoidance in strategy 4.

## 5.4. Effect of Time

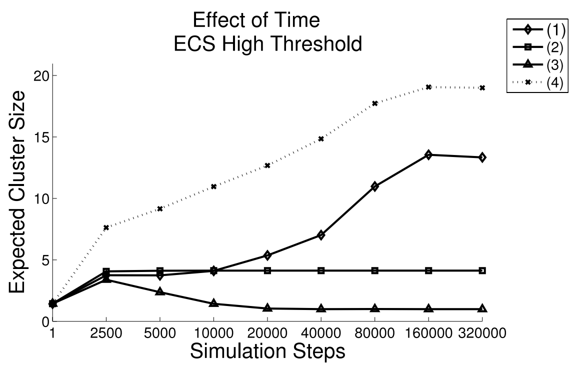
Robots must move to form clusters in aggregation behavior. This motion requires time. Moreover, the probabilistic controller used in this study includes *wait* behavior, which increases time required for aggregation. This part of experiments aims to observe phases of aggregation with respect to time.

Figure 13 gives the summary of change in performance of strategies with respect to number of simulation steps conducted for both, ECS and TD metrics. In this figures, number of simulation steps conducted change in $x - axis$ and performance metric values change on $y - axis$. This figure shows the median values of performance.

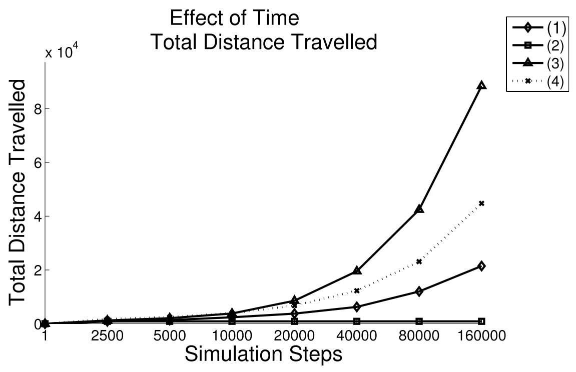
It can be seen that the ECS performance of strategy 1 is even lower than strategy 2 and 4 during the first 20000 steps. But after 40000 steps, performance of strategy 1 increases rapidly. Another interesting point is the initial increase of performance for strategy 3. All robots start with approach behavior thus they form initial clusters corresponding to increasing performance until 2500 simulation steps. After a random amount of time controlled with $P_{leave}$ robots start moving away from sound sources. In case of strategy 3, robots never change into the *approach* behavior. Which causes the performance of this strategy to fall afterward.

According to the results of the experiments with the TD metric, 4th strategy is superior. This effect can be explained by the definition of aggregation for the ECS metric. The threshold used for deciding on neighboring robots was 10 $cm$. This threshold is found to be high enough to detect clusters in strategy 1, but is not sufficient to label clusters in strategy 4 where clusters are more sparse internally. Figure 14(a) shows median cluster sizes for a larger threshold (maximum distance possible for IR detection range, 13 $cm$) for the four strategies. ECS metric values are calculated in the same way except the threshold is chosen to be higher.

Although the performance of strategy 4 is high, it is not very feasible for robotic systems. Apart from the risk of having large number of robots moving in close proximity, large energy consumption due to motion is problematic. In this strategy, robots never cease to move, therefore they use more energy.



(a) Effect of time on ECS for the four strategies with $T_{RobotClose}$ is equal to 13 $cm$.



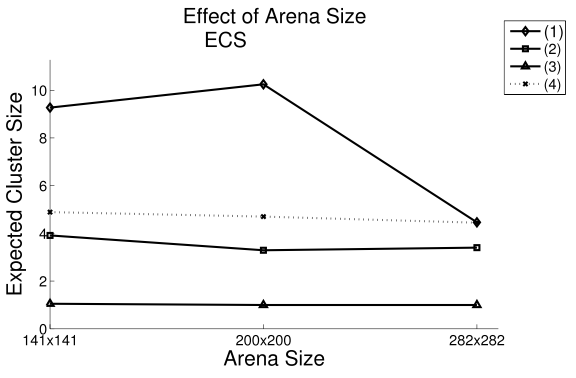
(b) Total distance traveled by robots with respect to time for the four strategies.

**Figure 14**. Additional experiments with 4 representative strategies

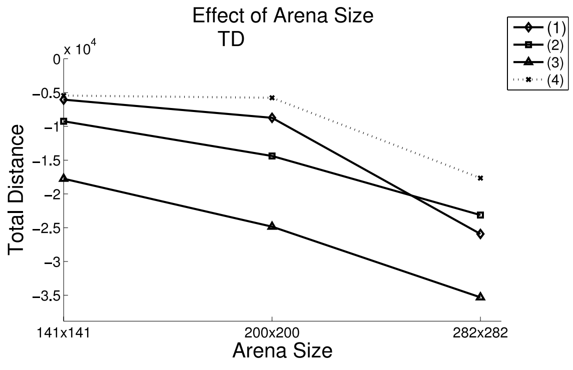
Figure 14(b) plots total distance traveled by all robots in the swarm for different strategies. In strategy 2, robots move only a little before coming to full stop and in strategy 3, robots move the largest distances among all strategies. There is also a significant difference between distances traveled by robots using strategy 1 and strategy 4.

## 5.5. Effect of Arena Size

Arena size in this study, controls the visibility of robots to each other. In standard arena size of 200 $cm \times 200\,cm$, a robot in the middle can perceive most of the arena. By changing the size of arena, relative size of the perceived area is changed. Arena size also effects the distance robots must traverse while forming robot clusters.



**(a)**



**(b)**

**Figure 15**. Effect of arena size on ECS for the four strategies. (a) shows median of metric values for ECS metric, (b) shows median of metric values for TD metric

Figures 15 display the performance of the four different strategies in different arena sizes. The results show that even when the transition probabilities are chosen for the arena with size 200 $cm \times 200\ cm$, the behavior can still outperform the other strategies in the smaller arena and perform reasonably well for the

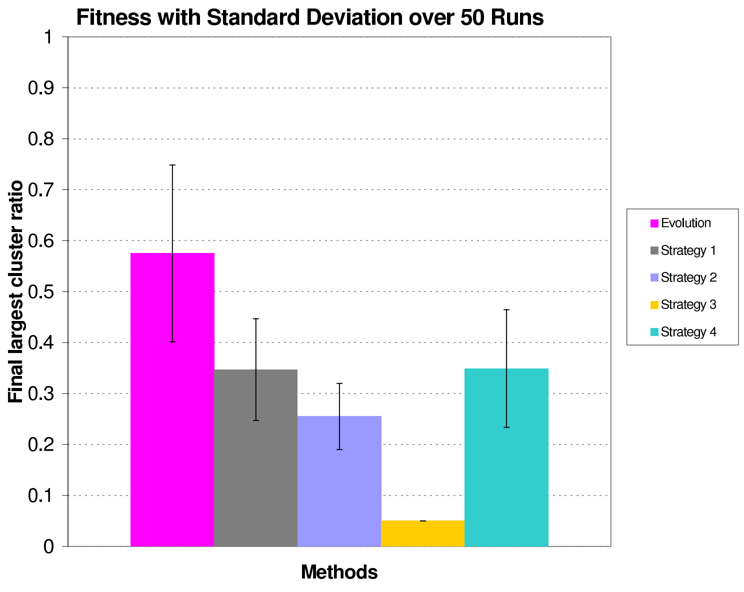


**Figure 16**. Comparison of 4 representative strategies and genetic algorithm controller in 282 × 282 arena with 20 robots.

larger arena.

With larger arena size the visibility graph of robots is usually disconnected. In this case, deterministic algorithms are bound to fail. Gordon *et. al* [10], note that additional random behavior can lead to better clustering when visibility graph is disconnected. Small improvement in performance supports this argument.

In small arena, without considering the perceptual aliasing effect, theoretical limit for performance is complete clustering. This is not observed in our experiments. It should be noted that, the sound sensor model used in this study is much simpler than the long range sensor models used in visibility graph discussions[9, 14, 10]. These discussions assume sensors to be able to distinguish sources or at least cluster directions.

In [30] a more powerful sensor is utilized for aggregation task. The sensor employed in that study is able to distinguish between different cluster sizes. Even with the relaxed sensor assumptions, the macroscopic model produced in that study predicts aggregation with constant probabilities is limited. Our results also agree with this macroscopic models predictions.

## 6. Comparison Experiments

We conducted additional experiments to compare performance of controllers. A random evolved controller is compared with 4 representative strategies identified in Figure 5. In Figure 16, metric used for evolution experiments is utilized in $282 \times 282$ sized arena with 20 robots. Probabilistic controllers are allowed to run 8 times more than the genetic controller and still genetic controller can significantly outperform probabilistic controllers.

## 7. Discussion and Conclusions

The probabilistic aggregation investigated is a minimal aggregation behavior composed of simple reactive behaviors. Parameters of this behavior and some parameters of the environment has been analyzed using two performance metrics.

Strategies obtained through variation of parameters provide segregation like behavior (strategy 3), static clustering (strategy 2), and dynamic clustering (strategies 1 and 4). Clustering metrics favor different dynamic clustering methods, indicating importance of an appropriate metric for the required task. Furthermore, threshold choice for the ECS metric is important since it can dramatically change the obtained results.

Although both strategies 1 and 4 form and break clusters during runs, their energy characteristics are different. Strategy 1 is more similar to cockroach behavior, where clustering is modulated by resting. This strategy allows robots to stay in closer proximity. On the other hand, strategy 4 looks more similar to schooling/flocking observed in fish and flies where agents are constantly in motion. In this strategy average distance between robots in the same cluster is relatively larger due to obstacle avoidance.

We wish to note that behaviors described in this study are not dependent on the specifics of the platform, thus are quite portable. Sound sensors can be replaced with any approximation of clusters around the robot and close range detection of other robots can be implemented with different methods like leds, bump sensors, etc.

Following this, we studied how several parameters involved in using evolutionary methods in swarm robotic systems affect the performance and the scalability of aggregation behavior. The experiments investigated trade-offs among number of runs per controller, number of generations in the genetic algorithm, and number of simulation steps to find out the most beneficial resource to dedicate processing time to. Furthermore, this study examined how to best merge fitness results obtained from simulation runs of the same controller with different seeds. Finally, one more experiment was done to better understand how evolution set-up size affects the scalability and performance on set-ups of different size.

We have made some assumptions regarding the problem domain in several aspects: the aggregation task, robot architecture, robot controller, and the genetic algorithm.

For the aggregation task, we considered clustering of robots in a closed arena where a robot can perceive a portion of the arena. Therefore, the robots had no means of broadcast communication. However, they were able to hear a group of robots better than a single one at the same distance. The controller of the robots was a single-layer perceptron, which means that the robots had reactive control, without any memory and acted without doing any planning. Furthermore, the robots were not capable of identifying each other. Also, the group of robots was homogeneous without a leader of any kind. This provides robustness for the system, which is one of the characteristics of swarm robotics.

There were also some assumptions on the genetic algorithm. The perceptron controller was encoded in the chromosome as a vector of floating point numbers with a mutation that adds or subtracts a random number with a probability of 1%. The size of the population used was 50, where the top 5 chromosomes were passed unaltered to the next generation. Tournament selection was used, where the selected chromosomes were subjected to cross-over with 80% probability.

Based on the results obtained from the experiments conducted, we conclude the following rules of thumb, which can be accepted as true in the assumed domain we described above:

1. To combine results from different fitness evaluations of the same controller (to overcome bias from random initial distributions), the use of *minimum* function should be preferred over the functions

*average*, *median*, and *maximum*.

2. When faced with the trade-off between the number of simulation steps for each run and the number of different runs per controller, one should choose the minimum sufficient number of simulation steps while maximizing the number of runs per controller. This will considerably eliminate negative effects of the high variance observed in robotics applications when initial positions are random.

3. The optimum value of the number of runs per controller and the number of generations (which is as important as number of runs) is not easy to obtain. Number of generations in evolution needed for emergence of a controller with acceptable performance, depends on architectural complexity of the controller and difficulty of the task. It is best to let the evolution run once initially for many generations to see about when the performance reaches a reasonable level.

4. In fitness evaluation, running simulations in multiple set-ups of different scale and multiplying the results improves scalability, since the controller is evaluated in multiple set-ups of different size, while being evolved. Also, evolving robot controllers on set-ups close in size to the actual set-ups that the robot is to be used will improve performance on that set-up. Therefore, using controllers evolved in set-ups too different than the application set-up is not recommended. Also evolving on a large set-up will increase variance of the evolved controller in performance among multiple evolutions.

The performance of aggregation behavior can be improved by more complex controllers. Using multi-layer perceptrons and other neural network structures can lead to better aggregation performance. Similarly, the probabilistic controller can benefit from more complex and numerous basic behaviors. Adding temporal, spatial and contextual components to transition probabilities are also viable alternatives. For instance, in [30], using cluster size dependent transition probabilities leads to better aggregation performance.

We believe that these results, obtained through the systematic experiments, have a high chance of being relevant both for evolving other swarm robotic behaviors in simulation, and for evolving behaviors for physical robotic systems. Some of these rules of thumb are already employed earlier in the studies that evolve self-organizing behaviors by Dorigo *et al.* [25]. They use many runs per controller to obtain more reliable fitness evaluations (as suggested by Figure 2 above) and for each of these runs they use a random number of robots to obtain a more scalable controller (which supports Figure 4).

## Acknowledgements

This work was partially funded by the "KARIYER: Kontrol Edilebilir Robot Oğulları" Career Project awarded to Erol Şahin (104E066) by TÜBİTAK (Turkish Scientific and Technical Council).

## References

[1] S. Camazine, N. R. Franks, J. Sneyd, E. Bonabeau, J.-L. Deneubourg, G. Theraulaz, *Self-Organization in Biological Systems.* Princeton, NJ, USA: Princeton University Press, 2001. ISBN: 0691012113.

[2] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, L. M. Gambardella, "Evolving self-organizing behaviors for a swarm-bot," *Autonomous Robots*, vol. 17, no. 2-3, pp. 223–245, 2004.

[3] E. Şahin, W. Spears, eds., *Swarm Robotics Workshop: State-of-the-art Survey*, vol. 3342 of *Lecture Notes in Computer Science*. Berlin Heidelberg: Springer-Verlag, 2005.

[4] E. Bahçeci, E. Şahin, "Evolving aggregation behaviors for swarm robotic systems: A systematic case study," in *IEEE Swarm Intelligence Symposium*, (Pasadena, CA, USA), 2005.

[5] O. Soysal, E. Şahin, "Probabilistic aggregation strategies in swarm robotic systems," in *Proc. of the IEEE Swarm Intelligence Symposium*, (Pasadena, California), pp. 325–332, June 2005.

[6] J. L. Deneubourg, A. Lioni, C. Detrain, "Dynamics of aggregation and emergence of cooperation," *Biological Bulletin*, vol. 202, pp. 262–267, June 2002.

[7] R. Jeanson, C. Rivault, J. Deneubourg, S. Blanco, R. Fournier, C. Jost, G. Theraulaz, "Self-organised aggregation in cockroaches," *Animal Behaviour*, vol. 69, pp. 169–180, 2005.

[8] C. Melhuish, O. Holland, S. Hoddell, "Convoying: using chorusing to form travelling groups of minimal agents," *Journal of Robotics and Autonomous Systems*, vol. 28, pp. 206–217, 1999.

[9] P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, "Gathering of asynchronous robots with limited visibility.," *Theoretical Computer Science*, vol. 337, no. 1-3, pp. 147–168, 2005.

[10] N. Gordon, I. A. Wagner, A. M. Bruckstein, "Gathering multiple robotic a(ge)nts with limited sensing capabilities," in *Lecture Notes in Computer Science*, vol. 3172, pp. 142–153, Jan 2004.

[11] V. Gazi, K. M. Passino, "Stability analysis of swarms," *IEEE Transactions on Automatic Control*, vol. 48, pp. 692 – 697, April 2003.

[12] V. Gazi, "Swarm aggregations using artificial potentials and sliding mode control," in *42nd IEEE Conference on Decision and Control*, vol. 2, pp. 2041– 2046, 2003.

[13] V. Gazi, K. M. Passino, "A class of attractions/repulsion functions for stable swarm aggregations," *International Journal of Control*, vol. 77, pp. 1567–1579, Dec 2004.

[14] Z. Lin, B. Francis, M. Ma, "Necessary and sufficient graphical conditions for formation control of unicycles," *IEEE Transactions on Automatic Control*, vol. 50, pp. 121–127, Jan 2005.

[15] S. T. Kazadi, *Swarm Engineering*. PhD thesis, Caltech, 2000.

[16] C. Lee, M. Kim, S. Kazadi, "Robot clustering," in *IEEE SMC [submitted]*, 2005.

[17] J. H. Holland, "Outline for a logical theory of adaptive systems," *J. ACM*, vol. 9, no. 3, pp. 297–314, 1962.

[18] J. H. Holland, *Adaptation in natural and artificial systems*. Ann Arbor, MI, USA: University of Michigan Press, 1992.

[19] N. Zaera, C. Cliff, J. Bruten, "(Not) evolving collective behaviours in synthetic fish," in *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behaviour*, (Cambridge, MA), pp. 635–6, MIT Press, 1996.

[20] M. J. Mataric, D. Cliff, "Challenges in evolving controllers for physical robots," *Journal of Robotics and Autonomous Systems*, vol. 19, pp. 67–83, Oct. 1996.

[21] H. H. Lund, J. Hallam, "Evolving sufficient robot controllers," in *Proceedings of the Fourth IEEE International Conference on Evolutionary Computation*, (Piscataway, NJ), pp. 495–499, IEEE Press, 1997.

[22] M. Dorigo, E. Tuci, R. G. V. Trianni, T. H. Labella, S. Nouyan, C. Ampatzis, "The SWARM-BOTS project," in *Swarm Robotics Workshop: State-of-the-art Survey* (E. Şahin and W. Spears, eds.), no. 3342 in Lecture Notes in Computer Science, (Berlin Heidelberg), pp. 31–44, Springer-Verlag, 2005.

[23] G. Baldassarre, S. Nolfi, D. Parisi, "Evolving mobile robots able to display collective behaviors.," *Artificial Life*, vol. 9, pp. 255–268, Aug. 2003.

[24] V. Trianni, R. Groß, T. Labella, E. Şahin, M. Dorigo, "Evolving Aggregation Behaviors in a Swarm of Robots," in *Advances in Artificial Life - Proceedings of the 7th European Conference on Artificial Life (ECAL)* (W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, J. Ziegler, eds.), vol. 2801 of *Lecture Notes in Artificial Intelligence*, pp. 865–874, Springer Verlag, Heidelberg, Germany, 2003.

[25] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano, L. M. Gambardella, "Evolving self-organizing behaviors for a *swarm-bot*.," *Autonomous Robots*, vol. 17, no. 2-3, pp. 223–245, 2004.

[26] C. R. Ward, F. Gobet, G. Kendall, "Evolving collective behavior in an artificial ecology.," *Artificial Life - Special issue on Evolution of Sensors in Nature, Hardware and Simulation*, vol. 7, no. 2, pp. 191–209, 2001.

[27] F. Mondada, G. C. Pettinaro, A. Guignard, I. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. Gambardella, M. Dorigo, "Swarm-bot: A new distributed robotic concept," *Autonomous Robots*, vol. 17, no. 2–3, pp. 193–221, 2004.

[28] J.-M. Valin, F. Michaud, J. Rouat, D. Letourneau, "Robust sound source localization using a microphone array on a mobile robot," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, pp. 1228–1233, 2003.

[29] R. Sedgewick, *Algorithms in C.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[30] O. Soysal, E. Şahin, "A macroscopic model for self-organized aggregation in swarm robotic systems," in *Second International Workshop on Swarm Robotics at SAB 2006* (E. Şahin, W. M. Spears, A. F. T. Winfield, eds.), vol. 4433, (Berlin, Germany), pp. 27–42, Springer Verlag, 2006.