
An Exploration of Online Parallel Learning in Heterogeneous Multi-Robot Swarms

Jim Pugh and Alcherio Martinoli

Swarm-Intelligent Systems Group
École Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
{jim.pugh,alcherio.martinoli}@epfl.ch

Summary. Designing effective behavioral controllers for mobile robots can be difficult and tedious; this process can be circumvented by using unsupervised learning techniques which allow robots to evolve their own controllers online in an automated fashion. In multi-robot systems, robots learning in parallel can share information to dramatically increase the evolutionary rate. However, manufacturing variations in robotic sensors may result in perceptual differences between robots, which could impact the learning process. In this paper, we explore how varying sensor offsets and scaling factors affects parallel swarm-robotic learning of obstacle avoidance behavior using both Genetic Algorithms and Particle Swarm Optimization. We also observe the diversity of robotic controllers throughout the learning process using two different metrics in an attempt to better understand the evolutionary process.

1 Introduction

Designing even simple behaviors for robots that are efficient and robust can be very difficult for humans; it is often not hard to implement a rudimentary controller that accomplishes the task, but achieving near-optimal performance can be very challenging, especially for miniature robotic platforms with severe hardware and computational limitations. Unsupervised robotic learning allows for automated design of efficient, robust controllers, which saves much design time and effort. Unsupervised learning is also useful for allowing robots to adapt to situations where the task/environment is unknown beforehand or is constantly changing.

Genetic Algorithms (GAs) are a very common method of accomplishing machine learning and optimization. Candidate solutions to a problem are modeled as members of a population, and breeding (selection and crossover) and mutation are applied to “parents” (high performing solutions) in the population to generate “children” (new candidate solutions). GA can be used to shape an Artificial Neural Network (ANN) controller by using the parameter

set as the weights, and the evaluative function as a measure of the performance of a desired robot behavior.

Particle Swarm Optimization (PSO) is a promising new optimization technique which models a set of potential problem solutions as a swarm of particles moving about in a virtual search space. The method was inspired by the movement of flocking birds and their interactions with their neighbors in the group. PSO achieves optimization using three primary principles: evaluation, where quantitative fitness can be determined for some particle location; comparison, where the best performer out of multiple particles can be selected; and imitation, where the qualities of better particles are mimicked by others. The algorithm can also be used to evolve ANN robotic controllers.

In robotic learning, in order to evaluate a candidate controller solution, a robot must run the controller for some period of time (typically from several seconds to several minutes) and observe how well it performs over that duration according to a pre-established metric (fitness function). The computational and temporal resources needed for this evaluation are drastically higher than those needed for the processing of the learning algorithm itself in almost all cases (less than 1% for the case study presented here). Therefore, robotic learning can be considered to be a very expensive optimization problem, and speed-up efforts should be focused on decreasing the number/length of evaluations rather than the internal workings of the learning technique.

Both GA and PSO use groups of interacting virtual agents in order to achieve their optimization. In collective robotics, groups of robots interact to accomplish their goals. It is therefore possible to implement these algorithms in a parallel distributed fashion for learning in multi-robot systems. Each robot is responsible for several virtual agents, which it evaluates at each iteration. After each set of evaluations, the robots communicate to share the fitness information needed to progress to the next iteration of the algorithm. By running the algorithms in this fashion, we need no external supervisor to oversee the learning process, and the speed of learning is significantly improved, as many robots evaluating in parallel increase the rate of candidate solution evaluation and therefore decrease the total learning time.

On real robots, sensors and actuators may have slightly different performances due to variations in manufacturing. As a result, multiple robots of the same model may actually perceive and interact with their shared environment differently, creating a physically heterogeneous swarm. While robotic controller evaluations are inherently stochastic due to partial, noisy environmental perception by individual robots, this heterogeneity may add a systematic bias between agents. This is a crucial difference; partial perception and noise can be overcome by information exchange between robots (i.e. shared evaluation measures of a controller between multiple robots will give a better estimate of its true performance), while sharing information between robots with different systematic biases may actually hinder the learning process, since biased results from other robots could cause a robot to incorrectly adapt its behavior. The result may be increased difficulty in evolving effective be-

havioral controllers. In order to model swarm heterogeneity, simulations of swarm-robotic learning can include fixed random variations in, for example, the sensitivity and offsets of on-board sensors. This should reflect a more realistic learning scenario, particularly for miniature robots where low-cost sensors often exhibit higher levels of manufacturer variation (often on the order of 10% or more).

In this paper, we explore the efficacy of parallel robotic learning using GA or PSO on swarms of heterogeneous robots with sensor variations. We observe the diversity of the GA and PSO populations in different instances to attempt to gain insight into the evolutionary process. Section 2 provides some background on GA, PSO, unsupervised robotic learning, and multi-robot learning. Section 3 details our robotic learning case study and gives results in the case of a homogenous robot swarm. Section 4 explores the impact of sensor offsets on the learning process, while section 5 studies the impact of different sensor scaling factors. In section 6, we analyze the diversity of the GA and PSO populations throughout the learning process in order to determine the cause of difference between the algorithms with and without sensor variations. In section 7, we discuss the implications of our results, and section 8 concludes and provides outlook on future work.

2 Background

Genetic algorithms were originally developed in the 1960s by John Holland. The algorithms are inspired by evolution, where the fittest members of a population tend to reproduce more often than the less fit members. Candidate solutions are modeled as a population of “chromosomes”. At each iteration of the algorithm, a new population is generated from the previous one. Selection of the parents of the new generation is implemented using one or more of several schemes, such as elitism (using only the top performing members of the population), Roulette Wheel sampling (stochastically choosing parents with weight proportional to performance), and rank selection (ranking chromosomes from best to worst and stochastically choosing parents with weight proportional to their rank). After parents have been chosen, crossover between the parents can occur with some probability (each chromosome is split into multiple pieces, and children use some parts from one parent and some parts from the other). This allows positive aspects from different chromosomes to be merged into a single chromosome. Last, mutation is applied, where each element of the chromosome may have its value randomly changed with some probability. This provides a random local search, which allows solutions to continue to improve beyond the genetic diversity that was available in the original population ([8], [17]).

The original PSO method was developed by James Kennedy and Russel Eberhart [5], [10]. Every particle in the population begins with a randomized position ($x_{i,j}$) and randomized velocity ($v_{i,j}$) in the n-dimensional search

space, where i represents the particle index and j represents the dimension in the search space. Candidate solutions are optimized by flying the particles through the virtual space, with attraction to positions in the space that yielded the best results. Each particle remembers the position at which it achieved its highest performance ($x_{i,j}^*$). Each particle is also a member of some neighborhood of particles, and remembers which particle achieved the best overall position in that neighborhood (given by the index i'). This neighborhood can either be a subset of the particles (local neighborhood), or all the particles (global neighborhood). For local neighborhoods, the standard method is to set neighbors in a pre-defined way (such as using particles with the closest array indices modulo the size of the population as neighbors, henceforth known as a “ring topology”) regardless of the particles’ positions in the search space. The equations executed by PSO at each step of the algorithm are

$$\begin{aligned} v_{i,j} &= w \cdot v_{i,j} + pw \cdot rand() \cdot (x_{i,j}^* - x_{i,j}) + nw \cdot rand() \cdot (x_{i',j}^* - x_{i,j}) \\ x_{i,j} &= x_{i,j} + v_{i,j} \end{aligned}$$

where w is the inertia coefficient which slows velocity over time, pw is the weight given to the attraction to the previous best location of the current particle and nw is the weight given to the attraction to the previous best location of the particle neighborhood. $rand()$ is a uniformly-distributed random number in $[0, 1]$.

PSO has been shown to perform as well as or better than GA in several instances. Eberhart and Kennedy found PSO performs on par with GA on the Schaffer f6 function [5, 10]. In work by Kennedy and Spears [11], a version of PSO outperforms GA in a factorial time-series experiment. Fourie showed that PSO appears to outperform GA in optimizing several standard size and shape design problems [7].

Unsupervised learning describes learning scenarios where there is no external entity which decides upon the training set inputs for the learning agent(s). Rather, inputs are generated dynamically as the agents interact with their environment. This is as opposed to supervised learning, where the inputs are generated/collected first and then used repeatedly. In supervised learning, the accuracy of the system at each iteration is usually decided by an external “teacher” evaluating the system output. The pre-defined inputs are split into two separate sets, one for training the system and the other for testing the performance. Supervised learning tends to be easier than unsupervised, as the data does not change between iterations of the algorithm and can be preselected to avoid using unusual or particularly noisy data points. However, supervised learning is not possible in situations where the input data to the system depends on the current state of the learning agent; this is the case for online robotic learning, since the robot’s movements affect what its sensors will perceive.

Evolutionary algorithms have been used extensively for unsupervised learning of robotic behavior. A good survey of the work is given in [14] and

more recently in [21]. More specifically, standard GA has been shown to be effective in evolving simple robotic controllers [6], and modified noise-resistant versions of both GA and PSO were shown to achieve very good performance on simulated unsupervised robotic learning, outperforming the standard versions of the algorithms [22].

Multi-robot learning has been used and explored in various ways; a survey of work (including learning on other multi-agent systems) can be found in [26]. Mataric studied mechanisms to encourage individual agents in a group to act in ways to help the group performance [13]. Multi-robot learning using several methods in a wide variety of scenarios has been explored [2], [25]. Specialization in multi-agent systems using reinforcement learning was studied in [19]. Techniques for increasing individual learning speed via multi-robot learning were studied in [9] and [15]. A modified version of a genetic algorithm has been embedded onto a 2-robot system to allow for distributed parallel learning [20]. Pugh and Martinoli found that both GA and PSO could be used for effective distributed parallel multi-robot learning in [23] and applied to heterogeneous robot groups in [24].

3 Parallel Swarm-Robotic Learning Case Study: Obstacle Avoidance

Following on the work presented in [23] and [24], we use the case study of obstacle avoidance for our swarm-robotic learning task.

3.1 Experimental Setup

For the learning techniques, we use the noise-resistant GA and PSO algorithms from [23]. GA uses elitism to select the best half of the population as the parent set, and then applies Roulette Wheel sampling to replenish the missing chromosomes. PSO uses a local neighborhood in a ring topology with one neighbor on each side. At every iteration, these algorithms reevaluate their previous best locations and parent sets for PSO and GA, respectively, combining the new fitness value with previous ones to get a more accurate measure of the actual fitness. Although this requires twice as many fitness evaluations at each iteration as their standard counterparts, this technique prevents noisy fitness evaluations from severely disrupting the learning process and gives much better results given the same number of evaluations of candidate solutions. Flowcharts of both algorithms are shown in Fig. 1.

For both PSO and GA, initial population member elements are randomly generated in the range $[-20, 20]$ but allowed to change to any value during evolution. Velocity in PSO is also randomly initialized in the range $[-20, 20]$ but prevented from ever going outside this range.

The parameters for the algorithms are given in Table 1.

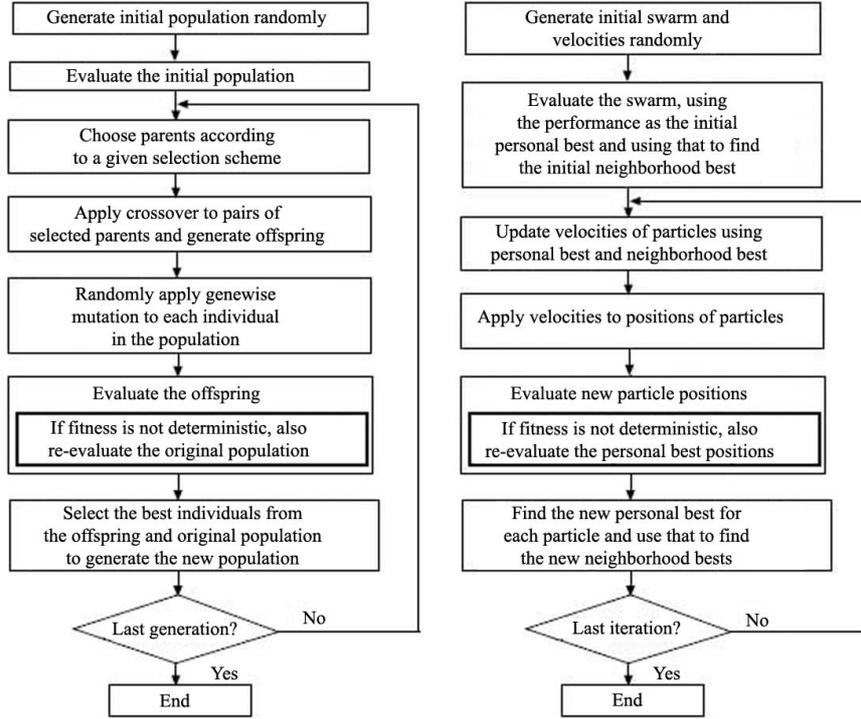


Fig. 1. Flowcharts for noise-resistant GA (*left*) and PSO (*right*).

Table 1. GA and PSO Parameters for Unsupervised Learning

| GA | | PSO | |
|-----------------------|-----------------|-----------------|-----|
| Population Size | 20 | Population Size | 20 |
| Mutation Probability | 0.15 | pw | 2.0 |
| Crossover Probability | 0.2 | nw | 2.0 |
| Mutation Range | $[-20.0, 20.0]$ | w | 0.6 |

We use Webots, a realistic simulator, for our robotic simulations [16], using 20 e-puck¹ robots [4] (this is as opposed to the Khepera robot [18] used in some previous experiments). The robot(s) operate in a 2.0 m x 2.0 m square arena with no additional obstacles (see Fig. 2 *left*). The robotic controller is a single-layer discrete-time artificial neural network of two neurons, one for each wheel speed, with sigmoidal output functions. The inputs are the

¹ <http://www.e-puck.org>

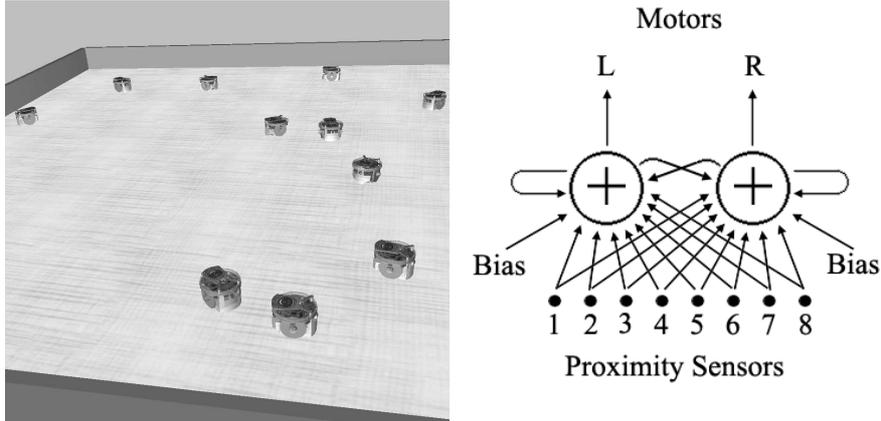


Fig. 2. *Left:* Robot arena with e-puck robots. *Right:* Depiction of the artificial neural network used for the robot controller. Curved arrows are recurrent connections and lateral inhibitions.

eight infrared proximity sensors, as well as a recursive connection from the previous output of the neuron, lateral inhibitions and bias values (see Fig. 2 *right*), giving us 22 weights total. Sensors have a maximum range of 5.0 cm, and sensor output varies linearly from 0 at maximum range to 1 at minimum range (0.0 cm) with 3% noise. Slip noise of 10% is applied to the wheel speed. The time step for neural updates is 128 ms. We use the fitness function used originally in [6] and again in [23]. The fitness function is given by:

$$F = V \cdot (1 - \sqrt{\Delta v}) \cdot (1 - i)$$

$$0 \leq V \leq 1, \quad 0 \leq \Delta v \leq 1, \quad 0 \leq i \leq 1$$

where V is the average absolute wheel speed of both wheels, Δv is the average of the difference between the wheel speeds, and i is the average activation value of the most active proximity sensor over the evaluation period. These factors reward robots that move quickly, turn as little as possible, and spend little time near obstacles, respectively. The terms are normalized to give a maximum fitness of 1. The evaluation period of the fitness tests for these experiments is 240 steps, or approximately 30 seconds. Between each fitness test, the position and bearing of the robots are randomly set by the simulator to ensure the randomness of the next evaluation.

During evolution, each of the 20 robots is responsible for a single member of the GA/PSO population (i.e. they must evaluate that member at each iteration and communicate the resulting fitness measure). With 100 iterations of the learning algorithms, this results in a total simulated learning time of approximately 1 hour 40 minutes (100 iterations comprised of 2 evaluations each lasting 30 seconds).

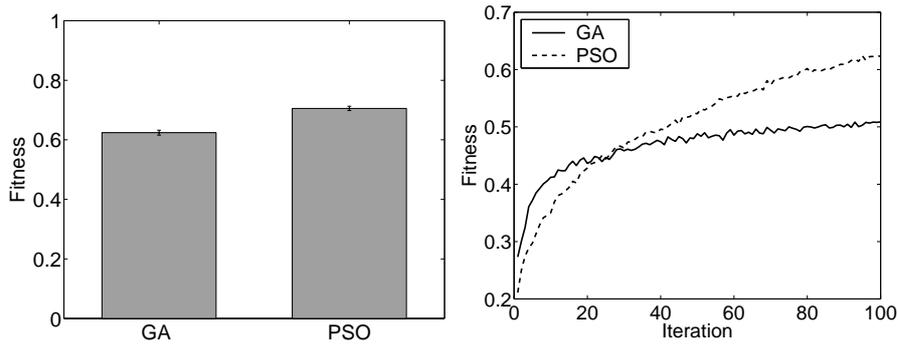


Fig. 3. *Left:* Fitness of best evolved controllers in population after evolution averaged over 100 runs for GA and PSO. Error bars represent standard error. *Right:* Average fitness of population members throughout evolution averaged over 100 runs for GA and PSO.

3.2 Results

The average performance over 100 runs of the best evolved controllers in the population can be seen in Fig. 3 *left*. As previously found, PSO is able to achieve superior performance, as GA occasionally produces poorer solutions. The progress of evolution can be seen in Fig. 3 *right*. While GA improves faster in the first few iterations, it quickly changes to a gradually increasing plateau, while PSO continues improving throughout the entire process.

Two primary types of obstacle avoidance controllers were observed in successful evolutionary runs of both GA and PSO. In the absence of obstacles, both types caused the robot to move ahead at full speed. The first controller consistently made the robot perform either a right turn or a left turn at the activation of any of its front sensors, thereby avoiding the obstacle in its path. The second controller used the recurrent neural connection to achieve a multi-state behavior; the robot would run forward until the front sensors detected an obstacle, at which point it would reverse its wheel speed and run backwards at full speed. When the rear sensors were activated, the robot would again resume forward movement, resulting in a back-and-forth behavior which somewhat resembles bouncing. Depending upon the environment, either of these two types of behavior might be preferable to the other.

4 Heterogeneity from Sensor Offsets

One potential variation between sensors is different offset values (i.e. if some sensor A detects value x , some other sensor B would detect $x + a$, where a is the offset value). We can incorporate this variation in our simulation using sensor values of:

$$v'_i = v_i + a_i$$

where v_i is the original sensor value, v'_i is the offset sensor value used by the robot controller, and a_i is a Gaussian offset value randomly generated at the start of the simulation with mean of 0 and standard deviation of s .

4.1 Experimental Setup

We repeat the experiments from section 3 using sensor offsets with standard deviation of 0, 0.05, 0.1, 0.2, and 0.5 (corresponding to 0%, 5%, 10%, 20%, and 50% of the maximum sensor value, respectively). In order to accurately evaluate the performance of the controller, the original sensor values are used for the proximity term of the fitness calculation. The same parameters are used for both GA and PSO.

4.2 Results

The average performance over 100 runs of the best evolved controllers in the population for different levels of sensor offset variation can be seen in Fig. 4 *left*. While the performance is not highly impacted for a standard deviation of up to 0.1, a significant decrease can be seen at 0.2, and a standard deviation of 0.5 yields quite poor performance. This suggests that neither GA nor PSO are affected by minor variations, but major variations could cause some sensors to be always/never perceived as active, which would cause very poor performance. The same trend can be seen in both GA and PSO, though it appears as though GA performs relatively better with large offsets.

While the previous results indicate how well evolved controllers perform on their heterogeneous robot group, we can evaluate the general performance of the controllers evolved on a heterogeneous swarm by testing them on a homogenous set of robots without sensor variations (see Fig. 4 *right*). We observe that while performance still decreases as sensor offsets increase, the effect is noticeably less. This indicates that the previously observed performance drop is due to a combination of suboptimal controllers from impaired learning and degraded performance from operating on a heterogeneous swarm. The relative performance of GA versus PSO is even higher for large sensor offsets, suggesting that learning is less impacted when using GA than when using PSO.

The performance of GA throughout learning with different offset levels can be seen in Fig. 5 *left*. We notice that the trend of rapid initial improvement followed by a gradually increasing plateau is not affected, but rather performance seems to be scaled down as the standard deviation of the offset grows. A similar trend can be observed for the PSO results in Fig. 5 *right*.

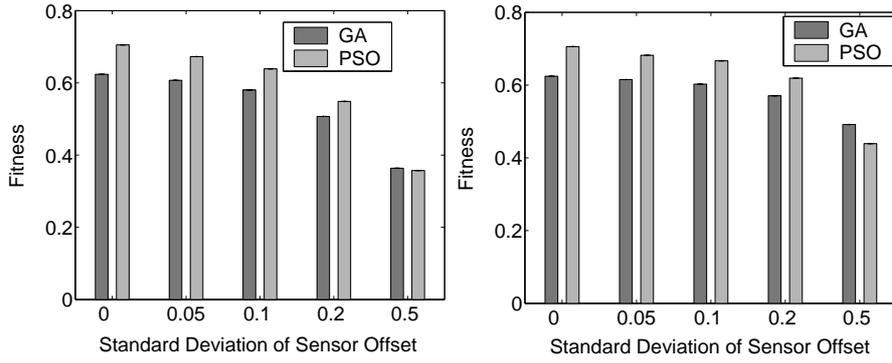


Fig. 4. Fitness of best evolved controllers in population after evolution averaged over 100 runs for GA and PSO with different standard deviations (s) of sensor offset. Results shown for final evaluation with both heterogeneous (*left*) and homogenous (*right*) robots. Error bars represent standard error.

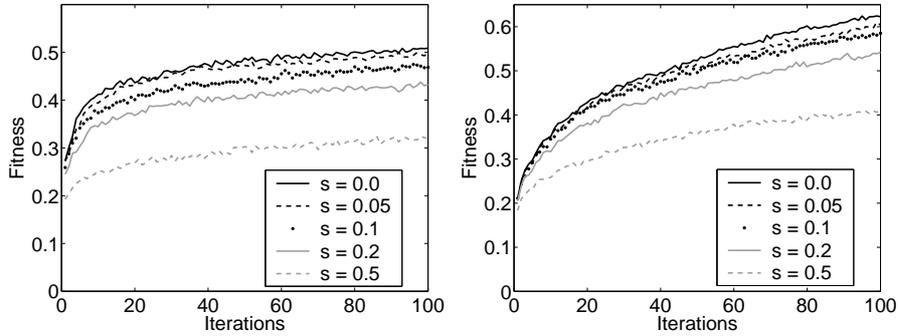


Fig. 5. Average fitness of population members throughout evolution averaged over 100 runs for GA (*left*) and PSO (*right*) with different standard deviations (s) of sensor offset.

5 Heterogeneity from Sensor Scaling

Another variation that can occur between sensors is sensitivity. One sensor may be more sensitive to input, which causes its output to vary more quickly than others. We can model this in our simulation using the equation:

$$v'_i = m_i v_i$$

where v_i is the original sensor value, v'_i is the offset sensor value used by the robot controller, and m_i is a Gaussian scaling factor randomly generated at the start of the simulation with mean of 1 and standard deviation of s .

5.1 Experimental Setup

We repeat the experiments from section 3 using sensor scaling factors with standard deviation of 0, 0.05, 0.1, 0.2, and 0.5 (and mean of 1). In order to accurately evaluate the performance of the controller, the original sensor values are used for the proximity term of the fitness calculation. The same parameters are used for both GA and PSO.

5.2 Results

The average performance over 100 runs of the best evolved controllers in the population for different levels of sensor scaling variation can be seen in Fig. 6 *left*. The performance here is impacted less than for the case of sensor offsets; good performance is maintained even for a scaling factor standard deviation of 0.5, though we do see an observable decrease here. The cause of this may be that, although the sensitivity is changed, sensors still react in the same way (i.e. higher for close proximity, lower for far proximity), and therefore the performance of the controller is not significantly impacted. We again see the same trend in both GA and PSO, and GA again performs relatively better with large scaling variations.

Observing the performance on a homogenous set of robots (see Fig. 6 *right*), we see that the actual quality of the evolved controller has not been at all affected by sensor scaling variation in almost all cases. The one minor exception is for controllers evolved using GA with a scaling factor standard deviation of 0.5, where we see a very slight performance decrease. Following the theory suggested above, it could be that because sensors respond in a similar way in spite of scaling, the same obstacle avoidance approaches are selected by the learning algorithms, which results in unaffected performance when using a homogenous set of robots.

The performance of GA throughout learning with different scaling levels can be seen in Fig. 7 *left*. As a result of the better performance, there is not such an obvious decrease as the standard deviation of the scaling grows, with the exception of a standard deviation of 0.5 where there is a marked performance drop. There is only very minor differences between the curves for PSO for different sensor scaling variation levels (see Fig. 7 *right*).

6 Diversity in Evolving Robot Swarms

Although PSO was able to achieve superior performance to GA in evolving robotic controllers in [22] and [23], the reasons for this were never apparent. A major drawback to using stochastic multi-agent algorithms such as GA and PSO is that they lack transparency about what exactly is happening during the evolutionary process and why they achieve the results that they do. In an attempt to shed some light on this issue, we analyze the diversity of the GA and PSO populations through their evolution using two different metrics.

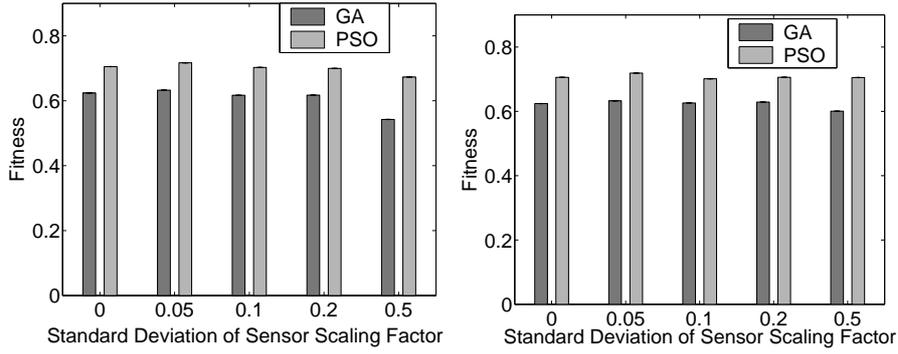


Fig. 6. Fitness of best evolved controllers in population after evolution averaged over 100 runs for GA and PSO with different standard deviations (s) of sensor scaling factors. Results shown for final evaluation with both heterogeneous (*left*) and homogenous (*right*) robots. Error bars represent standard error.

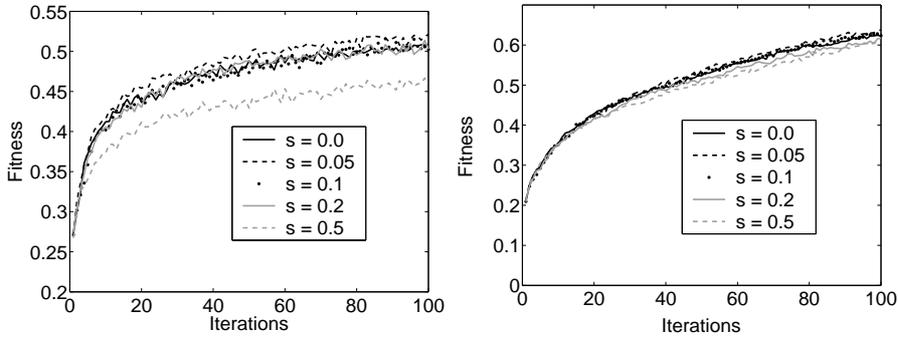


Fig. 7. Average fitness of population members throughout evolution averaged over 100 runs for GA (*left*) and PSO (*right*) with different standard deviations (s) of sensor scaling factors.

6.1 Euclidean Diversity

For our first diversity metric, we choose one of the most simple: Euclidean distance. We consider each candidate solution as a location in a virtual search space. The pairwise diversity between two members of a population is therefore given by:

$$d(a, b) = \sqrt{\sum_i (a_i - b_i)^2}$$

where $d(a, b)$ is the pairwise diversity between member a and member b , and x_i is element i of member x . The diversity of the entire population is given by the average pairwise diversity of all pairs of members in the population, or:

$$D_{eu} = \frac{1}{N(N-1)} \sum_a \left[\sum_{b \neq a} d(a, b) \right]$$

where a and b are members of the population, and N is the total size of the population.

6.2 Entropic Diversity

For our second diversity metric, we make use of a diversity measure dubbed the “hierarchic social entropy”, first proposed in [3] and later used in [12]. For this metric, population members are divided into clusters, where clusters are defined as maximal sets of “connected” population members. Two population members are considered directly connected if the measure of some distance metric between them (Euclidean distance in this case) is less than some threshold value h , and two population members are considered connected if there exists a set of direct connections linking them. Each cluster i can then be given a “probability” p_i which is calculated as follows:

$$p_i = \frac{n_i}{N}$$

where n_i is the number of members in cluster i and N is the total population size. Because clusters are maximal, an individual population member will appear in only a single cluster, and therefore:

$$\sum_i p_i = 1$$

We can therefore calculate the entropy of these probabilities as:

$$H = - \sum_i p_i \log p_i$$

resulting in an entropy of zero for a single cluster and increasing entropy as the number of clusters increases. To calculate the hierarchic social entropy (or Entropic diversity), we integrate entropy over all values of the threshold distance h :

$$D_{en} = \int_0^\infty H(h) dh$$

where $H(h)$ is the entropy of cluster probabilities as a function of the threshold distance h . While both Euclidean and Entropic diversities provide a measure of the distance between population members (high distance implies more clusters for the same value of h which yields higher entropy), only Entropic diversities directly reflects whether or not there exist compact groups of members in the virtual search space (which would suggest similar behaviors among these members).

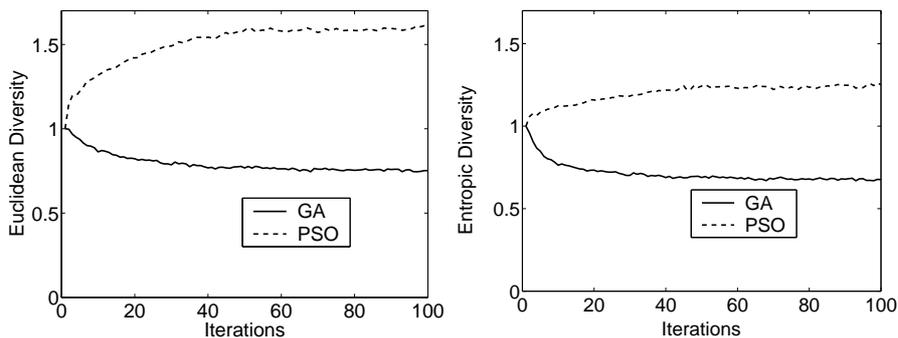


Fig. 8. Diversity of population members throughout evolution averaged over 100 runs for GA and PSO for a homogenous robot swarm for both Euclidean (*left*) and Entropic (*right*) diversity measures.

We measure diversity using both metrics throughout evolution for parallel learning using GA and PSO for homogenous robotic swarms and for heterogeneous robotic swarms with sensor offsets. We keep the same scenario and parameters as used previously. In order to make valid comparison, we normalize diversity measures to the diversity of the population at the first iteration (i.e. diversity always begins with value 1).

6.3 Results

The progress of population diversity throughout evolution for GA and PSO on a homogenous swarm can be seen in Fig. 8. For both metrics, the diversity in GA drops quickly in the initial iterations and maintains a stable lower value for the rest of the evolution. In contrast, the diversity of PSO actually *increases* initially and maintains a higher value. This is evidence for the hypothesis given in [23] that by very quickly converging to a good candidate solution, GA sacrifices the ability to further explore for better possible solutions, which causes it to occasionally converge to a mediocre final solution. On the other hand, by maintaining high diversity throughout evolution, PSO is able to continuously discover new and better solutions and continue improving throughout the entire evolution. This also demonstrates why PSO is better able to maintain good performance with a smaller population size, while GA does very poorly with small populations since it requires enough genetic diversity to already include a good solution.

Comparing Euclidean versus Entropic diversity, we see that while the trends are the same, the diversity increase in PSO is consistently less than for Entropic diversity (the difference between diversity measures, $D_{eu} - D_{en}$, can be seen in Fig. 9). This suggests that while PSO particles do indeed spread out in the virtual search space, they simultaneously form clusters, with some particles maintaining close proximity to some others. This follows as a result

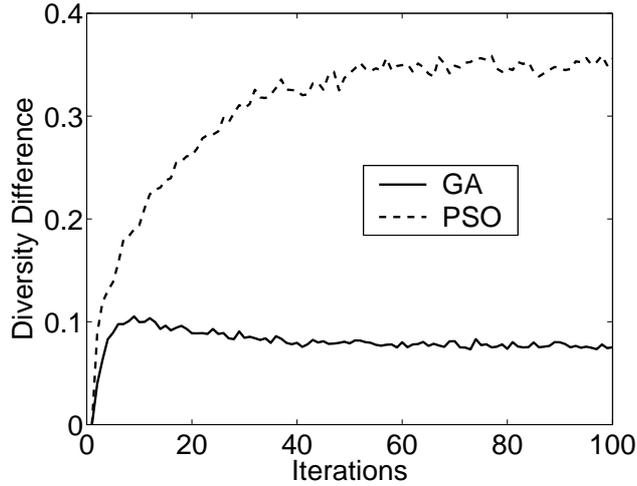


Fig. 9. Euclidean Diversity minus Entropic Diversity of population members throughout evolution averaged over 100 runs for GA and PSO for a homogenous robot swarm.

of the local neighborhood topology used by PSO, as neighboring particles should often assume similar values and therefore have a small Euclidean distance between them. For GA, the initial diversity decrease is greater for Entropic diversity than Euclidean, indicating that individuals are simultaneously converging and forming clusters. However, the difference in diversity plateaus and actually decreases slightly in the latter portion of evolution, suggesting that all further diversity decreases are the result of convergence rather than clustering.

We can also compare the progression of diversity for evolution of heterogeneous robotic swarms with sensor offsets using GA (see Fig. 10) and PSO (see Fig. 11). Both diversity metrics yield very similar results here. In the case of GA, sensor offset variation seems to have little impact on diversity, with much overlap between diversity measures for different standard deviations throughout the learning process. For PSO, there is no significant change in diversity for low sensor offset variations; for higher variations, the diversity grows at the same pace initially but achieves higher values towards the end of the evolution. This may be caused by less useful information sharing between robots due to major sensorial differences, promoting individual reliance. This might prevent convergence of particles, which would increase the diversity in the latter evolutionary stages.

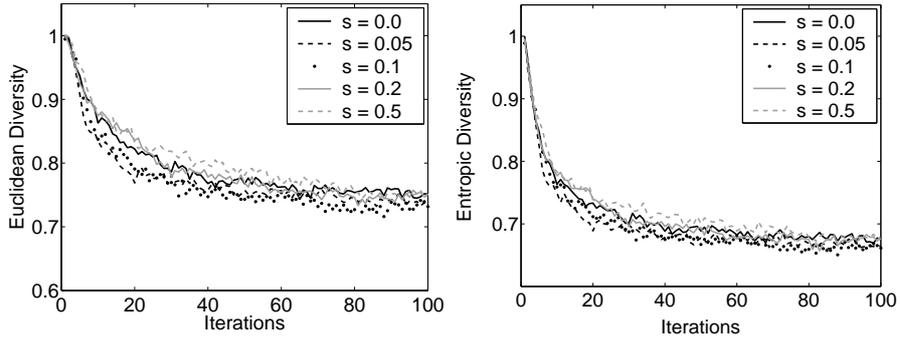


Fig. 10. Diversity of population members throughout evolution averaged over 100 runs for GA with different standard deviations (s) of sensor offset for both Euclidean (*left*) and Entropic (*right*) diversity measures.

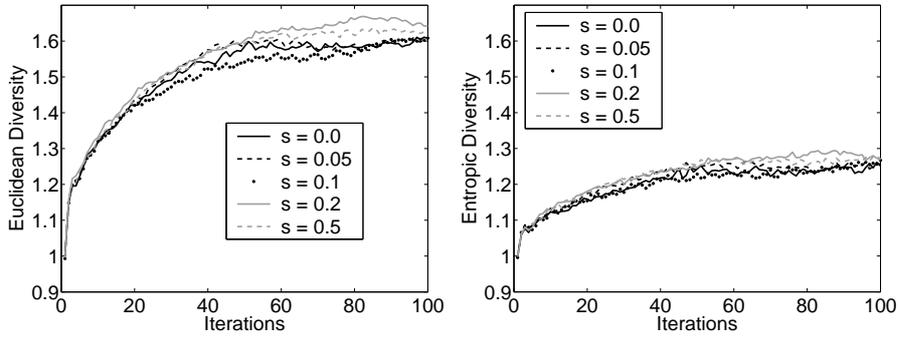


Fig. 11. Diversity of population members throughout evolution averaged over 100 runs for PSO with different standard deviations (s) of sensor offset for both Euclidean (*left*) and Entropic (*right*) diversity measures.

7 Discussion

For parallel swarm-robotic learning of obstacle avoidance behavior, offset variation seemed to have a much higher impact on performance than scaling variation. However, it is highly likely that this result is specific to our case study. As mentioned previously, because of the linear manner in which proximity sensors are used (i.e. higher - there is an obstacle, lower - there is no obstacle), scaling does not have much effect, while offset variation could potentially confuse the sensor as to its current state. One can envision other scenarios where this is not the case: if a sensor has some intermediate optimum value (such as would be the case in wall-following or flocking behavior), both offset and scaling would disrupt its reading; comparing subsequent sensor readings would remove any offset bias but still be affected by scaling. Therefore, parallel learning in heterogeneous robotic swarms needs to be tested on many other case studies before any such generalizations can be made.

Although not shown here, we also performed several parallel swarm-robotic learning tests using PSO without any velocity limit (w was kept at 0.6). While the performance of the resulting controllers was not significantly different, the diversity of the population exploded, growing exponentially until the end of the evolution (it was not uncommon to have a final Euclidean diversity on the order of 100,000). The plateau previously observed is therefore only caused by a hard velocity bound. We believe the reason for this explosive behavior is the same as the reason for the low impact of variations in sensor scaling; ANNs continue to function nearly the same if every weight is multiplied by a constant factor. The only change will be the sensitivity of the sigmoid output function, which will approach a step function as the neural weights increase.

8 Conclusion and Outlook

We have explored some of the effects of robot heterogeneity on parallel swarm-robotic learning. In the case of evolving obstacle avoidance, both Genetic Algorithms and Particle Swarm Optimization were able to withstand small variations in sensor offsets and large variations in sensor scaling factors, while showing poor performance with high offset variations. By observing population diversity throughout evolution, we discovered that PSO maintains much higher diversity, which could explain its superior ability to GA in avoiding local optima.

Though not explored here, there could be ways in which heterogeneous robotic swarms could identify differences between robots by sharing and comparing the observed fitness values of common controllers. This information could be used to help avoid the fitness degradation previously observed and might even be useful for deliberately pursuing specialization within the swarm in scenarios not limited to evolving one single controller.

We have shown that tracking population diversity can yield some insight into the progress of the evolving swarm. One could therefore imagine that these learning algorithms could be improved if they were to track their own diversity. A simple example would be allowing GA to scale the values of some of its population members if the diversity were too low. While calculating the diversity of the entire population requires a global manager, it might be feasible for individual robots to acquire a good estimation using local sensing of nearby teammates, which could allow this change to be implemented in a distributed fashion.

9 Acknowledgements

Jim Pugh and Alcherio Martinoli are currently sponsored by a Swiss NSF grant (contract Nr. PP002-116913).

References

1. Antonsson E K, Zhang Y, Martinoli A (2003) Evolving Engineering Design Trade-Offs. Proc. of the ASME Fifteenth Int. Conf. on Design Theory and Methodology, September, Chicago, IL, USA, paper No. DETC2003/DTM-48676.
2. Balch T (1998) Behavioral diversity in learning robot teams. PhD Thesis, College of Computing, Georgia Institute of Technology.
3. Balch T (2002) Measuring robot group diversity. In T. Balch & L. E. Parker (Eds.), Robot teams: From diversity to polymorphism, pp. 93-135.
4. Cianci C, Raemy X, Pugh J, Martinoli A (2007) Communication in a swarm of miniature robots: The e-puck as an educational tool for swarm robotics. Swarm-Robotics Workshop, Springer Lecture Notes in Computer Science, Vol. 4433, pp. 103-115.
5. Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. Proc. of the Sixth Int. Symposium on Micro Machine and Human Science, MHS '95, 4-6 Oct, pp. 39-43.
6. Floreano D, Mondada F (1996) Evolution of Homing Navigation in a Real Mobile Robot. Systems, Man and Cybernetics, Part B, IEEE Transactions on, Vol. 26, No. 3, June, pp. 396-407.
7. Fourie P C, Groenwold A A (2002) The particle swarm optimization algorithm in size and shape optimization. Struct. Multidisc. Optim., Vol. 23, pp. 259-267.
8. Goldberg D E (1989) Genetic Algorithms in Search, Optimization & Machine Learning. Addison-Wesley, Reading, MA.
9. Kelly I D, Keating D A (1998) Faster learning of control parameters through sharing experiences of autonomous mobile robots. Int. Journal of System Science, Vol. 29, No. 7, pp. 783-793.
10. Kennedy J, Eberhart R (1995) Particle swarm optimization. Proc. of the IEEE Int. Conf. on Neural Networks, Vol.4, Nov/Dec, pp. 1942-1948.
11. Kennedy J, Spears W M (1998) Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. Proc. of IEEE Int. Conf. on Evolutionary Computation, Anchorage, May, pp. 78-83.
12. Li L, Martinoli A, Abu-Mostafa Y (2004) Learning and Measuring Specialization in Collaborative Swarm Systems. Adaptive Behavior, Vol. 12, No. 3-4, pp. 199-212.
13. Mataric M J (1994) Learning to Behave Socially. Proc. of the 3rd Int. Conf. on Simulation and Adaptive Behaviors - From animals to animats 3, pp. 453-462.
14. Mataric M J, Cliff D (1996) Challenges in evolving controllers for physical robots. Robot. and Autonomous Sys., Vol. 19, No. 1, pp. 67-83.
15. Mataric M J (2001) Learning in behavior-based multi-robot systems: Policies, models, and other agents. Special Issue on Multi-disciplinary studies of multi-agent learning, Ron Sun, editor, Cognitive Systems Research, Vol. 2, No. 1, pp. 81-93.
16. Michel O (2004) Webots: Professional Mobile Robot Simulation. Int. J. of Advanced Robotic Systems, Vo. 1, pp. 39-42.
17. Mitchell M (1996) An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA.

18. Mondada F, Franzi E, Inne P (1993) Mobile robot miniaturisation: A tool for investigation in control algorithms. Proc. of the Third Int. Symp. on Experimental Robotics, Kyoto, Japan, October, pp. 501-513.
19. Murciano A, Millán J R, Zamora J (1997) Specialization in multi-agent systems through learning. Behavioral Cybernetics, Vol. 76, pp. 375-382.
20. Nehmzow U (2002) Learning in multi-robot scenarios through physically embedded genetic algorithms. Proc. of the 7th Int. Conf. on the Simulation of Adaptive Behavior: From animals to animats, pp. 391-392.
21. Nolfi S, Floreano D (2000) Evolutionary Robotics: The Biology, Intelligence, and Technology. MIT Press, Cambridge, MA, USA.
22. Pugh J, Zhang Y, Martinoli A (2005) Particle swarm optimization for unsupervised robotic learning. IEEE Swarm Intelligence Symposium, Pasadena, CA, June, pp. 92-99.
23. Pugh J, Martinoli A (2006) Multi-Robot Learning with Particle Swarm Optimization. Int. Conf. on Autonomous Agents and Multiagent Systems, Hakodate, Japan, May 8-12, pp. 441-448.
24. Pugh J, Martinoli A (2007) Parallel Learning in Heterogeneous Multi-Robot Swarms. IEEE Congress on Evolutionary Computation, Singapore, Sept 25-28. To appear.
25. Stone P (1998) Layered Learning in Multi-Agent Systems. PhD Thesis, School of Computer Science, Carnegie Mellon University.
26. Stone P, Veloso M (2000) Multiagent Systems: A Survey from a Machine Learning Perspective. Autonomous Robots, Vol. 8, No. 3, pp. 345-383.