An Algorithm for Distributed On-line, On-board Evolutionary Robotics

Giorgos Karafotias
giorgos00k@gmail.com

Evert Haasdijk
e.haasdijk@vu.nl

Agoston Endre Eiben
a.e.eiben@vu.nl

Vrije Universiteit Amsterdam
Dept. of Computer Science
Amsterdam, The Netherlands

ABSTRACT

This paper presents part of an endeavour towards robots and robot collectives that can adapt their controllers autonomously and self-sufficiently and so independently learn to cope with situations unforeseen by their designers.

We introduce the Embodied Distributed Evolutionary Algorithm (EDEA) for on-line, on-board adaptation of robot controllers. We experimentally evaluate EDEA using a number of well-known tasks in the evolutionary robotics field to determine whether it is a viable implementation of on-line, on-board evolution. We compare it to the encapsulated $(\mu+1)$ ON-LINE algorithm in terms of (the stability of) task performance and the sensitivity to parameter settings.

Experiments show that EDEA provides an effective method for on-line, on-board adaptation of robot controllers. Compared to $(\mu + 1)$ ON-LINE, in terms of performance there is no clear winner, but in terms of sensitivity to parameter settings and stability of performance EDEA is significantly better than $(\mu + 1)$ ON-LINE.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*; I.2.9 [**Artificial Intelligence**]: Robotics—*Autonomous vehicles*; I.2.6 [**Artificial Intelligence**]: Learning —*Parameter learning*

General Terms

algorithms, performance, reliability

Keywords

evolution, robotics, on-line evolution, adaptive behaviour, controllers

1. INTRODUCTION

Evolutionary computing techniques for optimisation and design have been used in robotics for well over a decade[24]. An overwhelming majority of the work in this field has focussed primarily on *off-line* evolution of robot controllers, where the evolutionary process takes place as a separate development phase before proper deployment of the robots and there is no subsequent adaptation –at least by evolution– of the controllers. Evolution is orchestrated by an overseer –an external computer– outside the robots themselves: the population of controllers undergoes selection and variation inside this computer. Fitness can either be evaluated in simulation (again inside this computer), or *in vivo* by uploading the controller onto a real robot that uses it for a while to collect information on controller quality. While the latter is often referred to as "embodied evolution", strictly speaking it only amounts to embodied fitness calculations; the evolutionary operators for selection and variation are not embodied in the robots.

In this paper we advocate a radically different approach to evolutionary robotics with genuine embodiment and on-line evolution. The essence of this approach is to implement evolutionary operators (selection, mutation, crossover) on-board and to evolve designs on the fly, as the robots go about their tasks [6]. As explained in [13], this approach offers the necessary adaptivity in collective robotic systems to cope with a number of fundamental challenges:

- Unforeseen environments that are not fully known during the design period.
- Changing environments where the extent and/or type of the change make the pre-designed solutions inadequate.
- Reality gap, that is, the phenomenon that off-line design is based on approximations and simulations, necessitating that robots be fine-tuned to the real operational conditions after deployment.

The essence of the problem we address here is producing/adapting robot controllers on-the-fly, without humans in the loop. This problem is highly relevant in the light of the global trend of increasing adaptivity and autonomy of computer systems, including those running on mobile hardware.

Considering a swarm of robots, we can distinguish two approaches to on-line, on-board evolution [6]:

Encapsulated evolution Each robot autonomously runs an independent evolutionary algorithm: each robot implements a centralised evolutionary algorithm and maintains a population of genomes using some time-sharing scheme to evaluate each controller;

Distributed evolution Each robot carries a single genome and uses that as its controller. The population comprises of the collection of the controllers of all robots and evolutionary operations take place in an autonomous and distributed manner by the robots interacting to exchange and recombine genetic material.

Obviously, these approaches can be combined to yield a hybrid approach where each robot runs an autonomous evolutionary algorithm and individual controllers are transferred between robots, similar to the island model in parallel evolutionary algorithms.

In this paper we introduce the Embodied Distributed Evolutionary Algorithm (EDEA), a new algorithm that adopts the distributed approach, and experimentally compare it with the $(\mu + 1)$ ON-LINE algorithm as an exemplar of the encapsulated approach, which was described in detail in [6, 7, 12]. The first assessment of the new algorithm is based on task performance, using a number of well-known tasks in the evolutionary robotics field: phototaxis, obstacle avoidance, and collective patrolling.

Furthermore, we note that the robustness required for the robots' controllers is also required for the evolutionary algorithm that adapts the controllers: it, too, has to operate reliably under unforeseen and possibly very different conditions. Unfortunately, the performance of evolutionary algorithms is, in general, quite dependent on their settings [5]. Hence, on-line evolutionary robotics requires an evolutionary algorithm with robust parameter settings that perform well over a wide range of problems, or an evolutionary algorithm that is capable of calibrating itself on-the-fly. Therefore, we evaluate the two algorithms not only in terms of performance, but also consider the number of parameters they have and the sensitivity to settings for these parameters (tuneability).

A third consideration is the stability of performance: reliable robot control requires consistently good or at least acceptable performance: a large variance in performance implies that an algorithm may perform well in one instance only to fail in another without any apparent difference in circumstances.

Summarising, the main objectives of this paper are: 1) to introduce the EDEA algorithm and to determine whether it is a viable implementation of on-line, on-board evolution for producing robot controllers without humans in the loop, 2) to compare the task performance of the distributed approach with the encapsulated one (exemplified by EDEA and $(\mu + 1)$ ON-LINE, respectively), 3) to compare the robustness (i.e., parameter sensitivity) of the distributed and the encapsulated evolutionary algorithms.

2. RELATED WORK

In this paper we examine on-line on-board evolution [6] in a bio-inspired manner motivated by the vision of self-adaptive, reliable, self-organising and self-developing swarms of robots and artificial multi-robot organisms [16]. Other work on on-line evolution of robot colonies is presented in [8] that describes the evolution of controllers for activating hard-coded behaviours for feeding and mating. In [2], Bianco and Nolfi experiment with open-ended evolution for robot swarms with self-assembling capabilities and report results indicating successful evolution of survival methods and the emergence multi-robot individuals with co-ordinated movement and co-adapted body shapes.

Watson et al. describe the notion of embodied evolution as the evolution taking place within a population of real robots in a distributed and asynchronous manner and report results on a resource gathering experiment [32]. Although their definition of embodied evolution is similar to our concept of on-line distributed evolution, EDEA's implementation is quite different from their probabilistic gene transfer algorithm, in which single genes are broadcast by every robot at a rate proportionate to its fitness. Experiments with on-line distributed evolution also appear in [2]. The $(\mu+1)$ ON-LINE algorithm –this paper's exemplar for the encapsulated approach– was reported on in [12] and [7]. Floreano et al use encapsulated evolution in [11] to evolve spiking circuits for a fast forward task. Encapsulated on-line evolution as a means for continuous adaptation by using genetic programming is suggested in [25]. An island model evolutionary algorithm is used in [31] to evolve a fast forward behaviour. Hybrid approaches are also taken in [8] (island model) and [21] (hall-of-fame approach) though in both cases evolved controllers merely activate hard-coded behaviours.

The majority of the experimental work in the field of evolutionary robotics has concentrated on the off-line evolution of robot controllers, e.g. [14], [9], [23]. In many of these cases incremental evolution is used to tackle complicated problems while co-evolution has also been examined as a way to address complex tasks [10]. Collective robotics settings have been addressed with off-line evolution as well: an extensive framework is presented in [19] while application examples can be found in [1], [27] and [18].

A recent extensive review of the literature in the evolutionary robotics field can be found in [22].

3. ON-LINE, ON-BOARD EVOLUTION

Any algorithm that implements on-line, on-board evolution has to take some uncommon considerations into account:

- On-board evolution implies (possibly very) limited processing power and memory, so the evolutionary algorithm must show restraint concerning computations, evaluations and population sizes;
- The best performing individual is not as important as in off-line evolution: because controllers evolve as the robots go about their tasks, if a robot continually evaluates poor controllers, that robot's actual performance will be inadequate, no matter how good the best individuals in the population. Therefore, the evolutionary algorithm must converge rapidly to a good solution and display a more or less stable level of performance throughout the continuing search;
- On-line evolution requires that the robots autonomously load and evaluate controllers without human intervention or any other preparation: the evaluation of a controller simply picks up the task where the previous evaluation left off. This introduces significant noise in fitness evaluations because the starting conditions of an evaluation obviously can have great impact on a controller's performance;
- Because the evolutionary algorithm has to be able to contend with unforeseen circumstances, it must either be able to (self-) adapt its parameter values as it operates or its parameters must be set to robust values that produce good performance under various conditions.

Subsection 3.1 lists design choices specific to distributed evolutionary algorithms for on-line, on-board evolution and introduces the EDEA as a implementation of a distributed evolutionary algorithm that takes all pertinent considerations in its stride. Subsection 3.2 provides some details on the $(\mu + 1)$ ON-LINE algortihm that was designed to address these considerations with the encapsulated approach.

3.1 A Distributed Algorithm

In distributed implementations of on-line, on-board controller evolution, each robot contains a single genotype that it decodes into its controller and evaluates during regular operation. The population of the evolutionary process is the aggregate of genotypes held by all the robots together; selection and variation occur through robot interactions. Distributed on-line evolution renders many standard centralised evolutionary algorithm concepts inapplicable, specifically requiring a different approach to selection and reproduction.

A centrally orchestrated algorithm would be possible, but it would limit the robots' autonomy, lead to scalability issues for large populations and introduce a single point of failure into the system. Thus, the crucial distinction of our envisioned distributed evolutionary algorithm is the lack of a central authority to guide selection or recombination and the ability of the robots to decide autonomously with whom and when to exchange genetic material, to generate new individuals from it and to deploy them.

To mate –to exchange genetic controller encodings– autonomously, the robots must identify potential partners, select one (disregarding the possibility of multi-parent recombination) and, once offspring genetic material has been constructed, embody it: actually run/evaluate the resulting controller on a robot, replacing that robot's current controller.

Partner identification For small populations –as in the experiments described below– where all robots are constantly in communication range, the robots can have a global view of all the group and contact random robots when interested in mating. This does, however, not scale to large numbers of robots or to environments where only some of the other robots are within communication range. Alternatively, the robots could engage in some hard-coded mate locating behaviour every so often as presented in [8]. Obviously, this detracts from the time robots actually spend tackling their proper tasks. Another approach relies on incidental physical colocation with information being transmitted through some communication channel with limited range as in the Probabilistic Gene Transfer Algorithm (PGTA) [32]. While an elegant and scalable approach, it does assume a group of robots that is densely deployed in space.

Although not used in the experiments here, EDEAcan maintain a peer-to-peer network (using wireless communication) where each individual has a small number of contacts and this overlay network is preserved through gossiping protocols that maintain connectedness even in the face of massive node failures [15]. Similar networks for structuring the population of an evolutionary algorithm have been successfully employed in experiments presented in [17]. Note the similarity of this set-up to cellular evolutionary algorithms [29].

Partner selection Once potential partners have been contacted, the robot has to select one to mate with. Selection strategies need not be uniform: one role may display eager behaviour (willing to mate with anyone) while another may hold a 'picky' stance (subjecting mating candidates to stricter criteria) – not unlike male and female behaviour in nature. It has been speculated that such a split between male and female behaviour is beneficial because males are forced to explore the genotype space as a result of the females' selectiveness [4, 20]. In a distributed evolutionary algorithm, there is no need to fix an individual's role one way or another: in PGTA, for instance, every robot plays both roles by constantly broadcasting its genes and at the same time evaluating received genes before incorporating them into its own genome [32].

In EDEA, robots play both an eager and a selective role, on the one hand selectively initiating the mating process using binary tournament while on the other hand eagerly responding to any mate proposal. Once a candidate has been selected, the initiating robot (which plays the selective role) compares its own fitness with that of the prospective partner to weight a probabilistic decision whether or not to press on with mating.

Embodiment Once a partner has been selected and a new genome created using standard recombination and mutation operators, the resulting genome must be deployed in a robot to be evaluated, replacing the current controller. Since there is no global view of the population in EDEA, the new controller must replace one in the direct neighbourhood (in terms of the overlay network) if it is to be deployed immediately. In fact, EDEA replaces the genome (only one offspring is created per mating interaction) of the initiating robot, justifying its fastidiousness during mate selection.

The Embodied Distributed Evolutionary Algorithm.

We introduce EDEA as an implementation of the distributed approach that follows from these considerations.

```
genome ← CreateRandomGenome;          // Initialisation
initiating ← false;
myFitness ← 0;
for ever do                           // continuous adaptation
    act();
    age++;
    fitness ← updateFitness();
    if age > $\alpha$ then
        offers ← P2PGetOffers();       // eagerly accept
        for $o \in$ offers do
            P2PSend(o.sender, genome, myFitness);
        end
        if initiating then             // selectively initiate
            candidates ← P2PGetCandidates();
            partner ← BinaryTournament(candidates);
            if random() < $\frac{candidate.Fitness}{s_c \cdot myFitness}$ then
                genome ← Crossover(candidate, genome);
                Mutate(genome);        // Gaussian $N(0, \sigma)$
                age ← 0;
            end
            initiating ← false;
        else
            initiating ← (random() < $p_c$);
        end
    end
end
```

Algorithm 1: The EDEA evolutionary algorithm.

Algorithm 1 provides pseudo-code for EDEA, which has the following parameters:

Maturation age α Before a genome can be considered in the mating process, it must have been evaluated for at least some time α to make its fitness measure reliable. The maturation age does not define a standard duration of evaluation but rather a lower bound, as a controller may continue to be active after it reaches age α. Adjusting the value of α affects speed of convergence because α implements a trade-off between the reliability of fitness evaluations (long evaluation times increase reliability) and the number of generations that can be achieved in a fixed amount of time (short evaluation time increase the number of evaluations).

Selection coefficient $\mathbf{s_c}$ Once a potential partner has emerged as the winner of a binary tournament from the neighbours of the initiator, it is selected based on its fitness in comparison to the fitness of the initiator. This confirmation is probabilistic and the selection coefficient s_c defines how fastidious the receiver is: the probability of mating is calculated as:

$$\frac{fitness_{candidate}}{fitness_{receiver} \cdot s_c}$$

Thus, larger values for s_c increase the selective pressure.

Preliminary experiments showed that the probability of initiating mating does not have any appreciable impact on the evolutionary process, so it has been set to a fixed value of 0.2. In large groups, however, it may perhaps be used to regulate network load.

From [12], we expect the mutation step size σ to have considerable impact on the algorithm's performance. We employ the derandomised self-adaptive strategy [26] to control this parameter during

EDEA runs: this has been shown to work well in similar settings with $(\mu + 1)$ ON-LINE [7].

3.2 An Encapsulated Algorithm

As benchmark, we consider an example of the encapsulated case: the $(\mu+1)$ ON-LINE algorithm as studied in [6, 7, 12]. In an encapsulated scheme, each robot contains an evolutionary algorithm to adapt its controller without reference to other robots or any central authority, therefore, it is not limited to situations involving groups of robots: it can be applied to a single robot as well. Such an application of encapsulated on-line evolution to a single robot realises the basic notion of on-line on-board evolution: an evolutionary process that continuously runs during deployment and task execution in order to provide constant adaptation in a changing and unpredictable environment. The $(\mu + 1)$ ON-LINE algorithm is an adaptation of the classic evolution strategy [28] with a fairly small population generating only $\lambda = 1$ child per cycle[1] to save on fitness evaluations. The $(\mu + 1)$ ON-LINE algorithm employs standard evolutionary algorithm operators (selection, variation and recombination) on a population of size μ to develop a new individual. That new individual –the challenger– is then evaluated by letting it take control of the robot for τ time steps and measuring the robot's task performance over that period. If the challenger's performance proves better than that of the worst in the population, the challenger replaces the current worst and the next iteration commences. To cope with noisy fitness evaluations, $(\mu + 1)$ ON-LINE re-evaluates genomes in the population with a given probability. This means that at every evolutionary cycle, either a new individual is generated and evaluated (with probability $1 - \rho$), or an existing individual is re-evaluated (with probability ρ). The fitness values from subsequent (re-)evaluations of any given individual are combined using an exponential moving average; this emphasises newer performance measurements and so is expected to promote adaptivity in changing environments;

For a detailed description of and discussion on $(\mu+1)$ ON-LINE, refer to [6, 7, 12].

4. EXPERIMENTAL ASSESSMENT

To assess EDEA, we compare it to $(\mu + 1)$ ON-LINE in a number of well-established settings as described below. While having multiple robots around is not a requirement for the encapsulated algorithm, it is obviously essential in the distributed case and to ensure equal circumstances, we have a group of 10 robots simultaneously tackling the problem in each instance. For the distributed approach, this means that there is a single evolutionary process using 10 robots with a population of 10 (one genotype per robot). In the encapsulated runs, there are 10 evolutionary processes (one in every robot), each with a separate population of μ individuals.

To ensure a fair comparison, we perform a modest parameter sweep for each algorithm in each task: multiple values are chosen for the most influential parameters and the algorithm is run several times for all parameter value vectors per task. We perform 20 repeats with different random seeds for each combination of task, algorithm and parameter vector. With 10 robots in each run, this yields 200 observations of robot performance for each combination. The values in the parameter sweep are chosen based on previous experience with the $(\mu + 1)$ ON-LINE [12] [7] and preliminary experiments with EDEA.

For each task, only the best parameter vector for each algorithm was used in the final comparison of the performance of encapsulated and distributed evolution. The variety of performance across parameter vectors can be seen as an indication of each algorithm's tuneability –the sensitivity to the parameter settings– the lower the tuneability, the less effort one needs to spend to get the parameter values just right.

In all experiments, the robots –simulated e-pucks in the Webots simulator[2]– are controlled by simple perceptron neural networks and the evolutionary algorithms determine the weights of the connections between the neurons. The fitness functions are obviously task dependent and are described with each task, below. The perceptrons use a $tanh$ activation function and receive inputs from light, distance, pheromone and food sensors and camera (depending on the task) and have two or three (depending on the task) output neurons that drive the wheels and LEDs. All inputs are normalised in the $[0, 1]$ interval before fed to the neurons. Equally, the outputs are normalised to $[0, 1]$ and interpreted as fraction of the full speed for the motors and as an on/off value for the LEDs (values less than 0.5 turn the LEDs off while larger values turn them on).

The experiments run for 10,000 seconds of simulated time; we use time rather than number of evaluations or generations because we are primarily interested in the performance of the robots in real time, regardless of how that is achieved by the evolutionary algorithm.

As stated in Section 3, we are primarily interested in the actual performance of robots, not in the performance of the best individual in the population at any given time. Actual performance is measured as the average performance during a time-span, irrespective of how many controllers may have been activate during that time.

Because of the limited group size of the experiments, we do not use the gossiping maintenance scheme described in Section 3.1, but randomly select potential partners from the whole population.

The settings for the experiments are summarised in Table 1.[3]

Phototaxis.

Phototaxis –seeking out or tracking a light source– is a very straightforward task that has been addressed by many ER researchers. The task is frequently combined with other tasks such as goal homing [30] and flocking [1]. In our comparison, we use the simplest version of phototaxis: robots only have to move towards a stationary light source and then remain as close to it as possible. In the phototaxis task, the robots use eight light sensors to detect light intensity and base their behaviour on that. The fitness function simply rewards intensity of received light:

$$f = \sum_{t=0}^{\tau} \max_{i=1}^{8}(lightSensor_i) \qquad (1)$$

where $lightSensor_i$ is the normalised input from a light sensor between 0 (no light) and 1 (brightest light).

The arena is an empty (apart from the ten robots) square with a light source in the middle: we ignore collisions between robots in these experiments, so we can do without the robot's distance sensors. For this simple experiment, we also compare the performance of both algorithms against a Braitenberg [3] controller as a baseline.

Fast Forward.

Fast forward –moving in as straight line as possible as fast as possible while avoiding obstacles– is maybe the most common task

[1]A value that would be considered extremely small by evolution strategy standards

[2]`http://www.cyberbotics.com/`

[3]Source code for the algorithm as well as the experiments described here is available at `http://www.few.vu.nl/~ehaasdi/papers/GECCO-EncapsvsDistr`

Experiment details	
Task	phototaxis, fast forward, collective patrolling
Robot group size	10
Simulation length	10,000 seconds (simulation time)
Number of repeats	20
Controller details	
NN type	Perceptron
Input nodes	*Phototaxis:* 8 light sensors + bias; *Fast forward:* 8 distance sensors + bias; *Collective patrolling:* 8 distance sensors + 4 pheromone sensors + bias;
Output nodes	2 (left and right motor values)
Evolution details	
Representation	real valued vectors with $-4 \leq x_i \leq 4$
Chromosome length	*Phototaxis, Fast forward:* 18; *Collective patrolling:* 26
Fitness	See task descriptions
Mutation	Gaussian $N(0, \sigma)$
Mutation step-size	Derandomised self-adaptive
Crossover	averaging
EDEA settings	
Maturation time α	300, 600, 1200, 1800 time steps
Selection coefficient s_c	0.5, 0.75, 1.0
Partner location	peer-to-peer network
Partner selection	binary tournament and fitness-based probabilistic
Embodiment	replace initiating parent
$(\mu + 1)$ ON-LINE settings	
Evaluation time τ	300, 600, 1200 time steps
Re-evaluation rate ρ	0.2, 0.4, 0.6
Re-evaluation strategy	exponential moving average
Population size μ	6, 10, 14
Parent selection	binary tournament
Crossover rate	1.0
Survivor selection	replace worst in population if challenger is better

Table 1: Experimental set-up

in evolutionary robotics research. In a confined environment with obstacles this task implies a trade-off between avoiding obstacles and maintaining speed and forward movement. The fitness function we use has been adapted from [24]; it favours robots that are fast and go straight ahead. Equation 2 describes the fitness calculation:

$$f = \sum_{t=0}^{\tau} (v_{trans} \cdot (1 - v_{rot}) \cdot (1 - d)) \tag{2}$$

where v_{trans} and v_{rot} are the translational and the rotational speed, respectively. v_{trans} is normalised between -1 (full speed reverse) and 1 (full speed forward), v_{rot} between 0 (movement in a straight line) and 1 (maximum rotation); d indicates the distance to the nearest obstacle and is normalised between 0 (no obstacle in sight) and 1 (touching an obstacle). As for phototaxis, we also compare the performance against a Braitenberg controller as a baseline.

Although fast forward is considered a trivial task, here some extra difficulty is added by using a complicated maze-like arena (Figure 1(a)) with tight corners and narrow corridors that fit only a single robot and sometimes lead to dead ends. This arena structure, compounded by the fact that multiple robots will be simultaneously deployed, makes the task considerably harder than commonly seen instances. This additional complexity is confirmed by the results of the baseline trials: the Braitenberg controllers invariably get stuck after a while (Section 5).

Collective Patrolling.

An obvious real-world task for a group of autonomous mobile robots is that of a distributed sensory network, where the robots have to patrol an area as a group and detect events that occur periodically. It differs from the previous tasks since it requires some level of co-ordination: the success of the group depends not only on the efficient movement of the individual robots but also on the spread of the group across the arena to maximise the probability of detecting events. Somehow, robots need to liaise so as not to patrol the same areas. To this end, they are equipped with a pheromone system: robots continuously drop pheromones (this is a fixed behaviour and not controlled by the evolved controller) while sensors detect the local pheromone levels. The collective patrolling task is described in [19] where controllers evolve off-line, although in that work the approach to events is more complicated and the robots use other sensory inputs.

The experiments for this task take place in the arena shown in Figure 1(b).

Every $T_e = 50ms$ with probability $p_e = 0.0005$, an event occurs at a random location with a duration of $d_e = 500 + \mathcal{N}(0, 2)$ seconds. Thus, in one run $(10,000$ seconds) approximately 100 events occur and that at any time around 5 events are active in the whole arena.

A robot detects an event whenever it comes within $0.3m$ of the event, so a robot's sensory coverage is $0.283m^2$. Since the arena is $25m^2$, a group of 10 robots can at any moment cover at most 11% of the whole arena; conversely, a group of stationary robots should detect around 11% of the events.

Pheromones are simulated as follows: the $5m \times 5m$ arena is divided into 500×500 cells, each with a pheromone level between $[0, 2]$. Every second, each robots drops 1 unit of pheromones at the cell the robot is currently in, and a linearly decreasing amount in nearby cells up to a range of $R_p = 0.07m$. Pheromone levels decay over time at a rate of $R_c = -0.024/s$ in each cell.

As sensory input to the controller, 4 pheromone sensors are placed at the periphery of the circular body at $\frac{\pi}{4}$, $\frac{3\pi}{4}$, $\frac{5\pi}{4}$ and $\frac{7\pi}{4}$. Each sensor detects the accumulated pheromone levels of all cells in a range of $0.05m$ (with detected levels decreasing linearly with distance from the sensor). For fitness calculation only, a similar sensor is positioned on the centre of the robot.

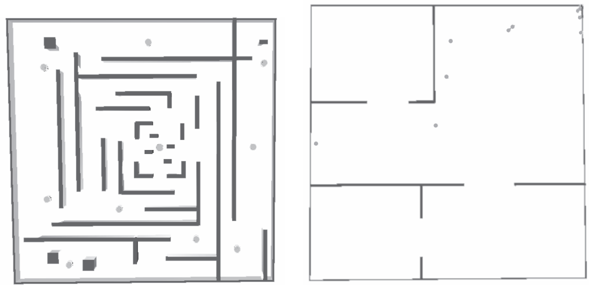
The fitness function penalises pheromones presence (detected by the central pheromone sensor) and proximity to obstacles:

$$f = \sum_{t=0}^{\tau} ((1 - p) \cdot (1 - d)) \tag{3}$$

where p indicates pheromone presence between 0 (no pheromones) and 1 (strongest pheromone level) at the current location and d indicates the distance to the nearest obstacle and is normalised between 0 (no obstacle in sight) and 1 (touching an obstacle).

This fitness function rewards covering behaviour and does not include the number of events detected, but we report the percentage of detected events as performance (e.g., in Fig. 4) rather than on the fitness itself.

Although movement is not included explicitly, it should emerge due to the continous dropping of pheromones and the deleterious effect of staying in a place where the robot just dropped them.



(a) Fast forward arena (b) Patrolling arena

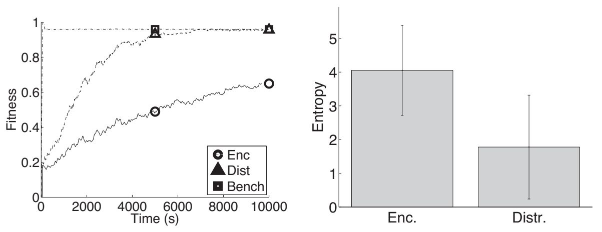
Figure 1: Arenas for two tasks; the circles represent the robots to scale.

5. RESULTS

Figures 2 to 4 compare the performance of the encapsulated $(\mu + 1)$ ON-LINE and distributed EDEA algorithms on the various tasks, using the best performing parameter settings for that task. The graphs on the left present performance –averaged over all repeats and robots– versus time. In these plots, the performance is scaled so that the theoretical optimum is 1. For the phototaxis and fast forward tasks, we additionally plot performance for the Braitenberg vehicles as a baseline.

To assess the volatility of the robot's actual performance over the course of the experiments, we calculate the differential entropy of actual performance over the last 20% of each run: lower entropy indicates a lower level of volatility. The right-hand plots show average entropy (grey bars) with standard deviation (black whisker lines) for the 20 runs with the best performing parameter vector over all robots for both algorithms.

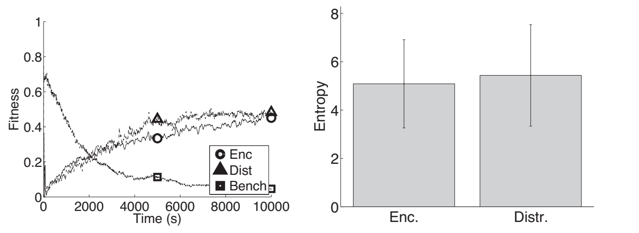
Far all comparisons, we test the significance of difference with t-tests at 95% confidence.



(a) Performance vs. time (b) Entropy (lower means more stable performance)

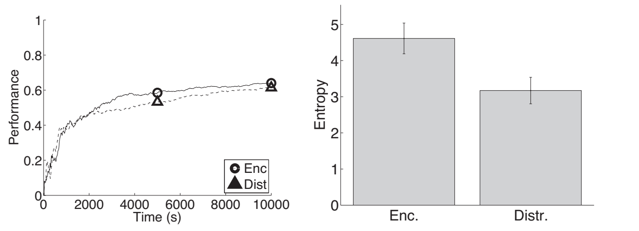
Figure 2: Phototaxis results; EDEA is significantly better in terms of both performance and stability.

Summarising these comparisons, EDEA performs significantly better in the phototaxis and fast forward tasks and is more stable in the phototaxis task (while there is no significant difference in stability for the fast forward task). However, $(\mu+1)$ ON-LINE has significantly better performance in the patrolling task, although EDEA is more stable there, as well.



(a) Performance vs. time (b) Entropy (lower means more stable performance)

Figure 3: Fast forward results; EDEA performs better at this task, the difference in stability cannot be shown to be significant.



(a) Performance vs. time (b) Entropy (lower means more stable performance)

Figure 4: Collective patrolling results; $(\mu + 1)$ ON-LINE performs better, EDEA is significantly more stable.

For the phototaxis and fast forward tasks, the evolved controllers compare very well to the Braitenberg baseline. Evolved phototaxis controllers match the behaviour of the benchmark while the evolved fast forward controllers outperform the benchmark controller –which gets stuck in cul-de-sacs where left and right sensory input are equal– by a considerable amount.

	EDEA		$(\mu+1)$ ON-LINE		
	α	s_c	μ	ρ	τ
Phototaxis	***1800***	***0.5***	**10**	**0.4**	**300**
	1800	1	6	0.2	600
	1800	0.75	6	0.4	300
			6	0.6	1200
			10	*0.6*	*300*
			10	0.4	300
			10	0.4	600
Fast forward	**1200**	**0.75**	**6**	**0.6**	**300**
	1800	*0.5*	*10*	*0.6*	*300*
	600	1	6	0.4	300
	600	0.75			
Patrolling	**1200**	**0.5**	**10**	**0.6**	**300**
	1800	0.75	14	0.6	300
	1800	1	14	0.4	300
	1200	1	*10*	*0.6*	*300*
	1800	*0.5*	10	0.4	300
	1200	0.75	6	0.6	600

Table 2: The best performing vectors are listed in bold; the remainder are vectors whose performance is not significantly worse (according to 95% t-tests) than the best. Settings in italics denote a parameter vector that is common for all tasks.

Table 2 shows the parameter vectors that lead to optimal or near optimal (not statistically different from optimal according to 95% t-test) performance. Interestingly, for all tasks there is at least one (near) optimal vector with every value of s_c tested. This seems to indicate that s_c may be kept constant as there will always be a value for α that will work well with it.

	EDEA	$(\mu + 1)$ ON-LINE
Phototaxis	25% (3/12)	25% (7/27)
Fast forward	33% (4/12)	11% (3/27)
Patrolling	50% (6/12)	22% (6/27)

Table 3: Tuneability comparison of EDEA and $(\mu+1)$ ON-LINE. Shows, for each task, the percentage of parameter vectors that yield (near) optimal performance.

Table 3 compares the tuneability –the dependence on parameter settings to achieve good performance– of the two algorithms by comparing the ratio of the parameter vectors that result in near optimal performance out of all parameter vectors that were considered. This shows that EDEA is more resilient when it comes to parameter tuning: a substantially higher ratio of vectors leads to near optimal performance for two tasks, while for phototaxis the same ratio of vectors is near optimal.

6. DISCUSSION AND CONCLUSION

On the whole, on-line, on-board evolution of robot controllers is a success: with only very small populations and within short times, both EDEA and $(\mu+1)$ ON-LINE are able to evolve good controllers for all three tasks, matching or even outperforming benchmark performance. Although the tasks used are somewhat straightforward, these same tasks have been used for off-line evolutionary robotics experiments that had superior resources available in terms of population size, generations and time. In that light, the success of on-line evolution as shown here is noteworthy and we can conclude that EDEA does provide a successful implementation of the general idea. It is a matter of further investigation whether the on-line algorithms tested here will scale up to more complex tasks and environments.

In terms of performance, the comparison of these implementations of the encapsulated and distributed approaches shows no clear winner: EDEA outperforms $(\mu+1)$ ON-LINE for the (simpler) phototaxis and fast forward tasks, but $(\mu + 1)$ ON-LINE is better when it comes to patrolling. Of course, this comparison relies on the parameter values of the evolutionary algorithms, which are determined by our tuning process. Forced by computational limitations we only tested a small number of different parameter values. Nevertheless, it seems safe to say that the relative performance of the algorithms depends on the nature of the task, and the results simply beg for a combined island model algorithm that may have the best of both worlds.

In terms of sensitivity to parameter settings, EDEA does seem to be the clear winner: even in the patrolling task, where $(\mu + 1)$ ON-LINE performs better, EDEA has the advantage that half of the tested parameter vectors were optimal while the performance is not much (although significantly) worse than that of $(\mu + 1)$ ON-LINE. Moreover, substantially more effort was put into tuning $(\mu + 1)$ ON-LINE than EDEA (2 parameters and 12 vectors for EDEA versus 3 parameters and 27 vectors for $(\mu + 1)$ ON-LINE). Table 2 indicates that EDEA's two parameters may be reduced to one: for all tasks there is at least one (near) optimal vector with every value of s_c tested, so it seems that it is merely a matter of finding the appropriate setting for α for some constant s_c (possibly through on-line parameter control (e.g. based on racing), making EDEA completely parameter-free).

One reason for EDEA's success may lie in the fact that a distributed evolutionary algorithm exploits the presence of multiple robots by effectively implementing concurrent evaluation of the population, allowing evolution to progress rapidly in real time. Meanwhile, the encapsulated approach can only evaluate its populations sequentially, falling rapidly behind in terms of evolutionary steps taken.

On the other hand, the multiple instance nature of $(\mu + 1)$ ON-LINE's encapsulated approach can offer an advantage when dealing with tasks or environments that involve competition or require various skills: here, the separate evolutionary algorithms of robots can promote co-evolution, speciation and/or specialisation.

The advantage of the concurrent evaluation can be easily understood for the tasks where EDEA shines, but the connection between the patrolling task, where the encapsulated $(\mu + 1)$ ON-LINE algorithm performs better, and the advantage of co-evolution and/or speciation is not so straightforward. Here, performance is based on the presence of pheromones –that all robots emit continuously– and a robot must learn to move around efficiently while avoiding its own as well as others' pheromones to claim fresh locations. Robots that have the same strategy for moving around are hamstrung in such a scenario: they always trip over each other's pheromones, exactly because they follow similar paths. Obviously, in the distributed case, robots are more likely to have similar controllers (as they are from the same population) than in the encapsulated case (where each controller stems from a different, independently evolving population), and so are more likely to follow a trodden path. In fact, the set-up here introduces a subtle form of competitive co-evolution: although the task is a collective one, the robots actually compete for sites in the arena to claim.

An obvious next step from this research is to try and have the best of both worlds by merging the encapsulated and distributed approach into an island-like model of autonomously evolving populations in each robot with individuals migrating from one to another. Research in this direction is currently underway.

Our conclusions need to be confirmed with more extensive experiments of increased complexity –both in terms of task and controller structure. Work to this end is underway.

Acknowledgements

This work was made possible by the European Union FET Proactive Intiative: Pervasive Adaptation funding the SYMBRION project under grant agreement 216342. We also thank Selmar Smit, Nicolas Bredeche and other consortium members for their enthusiastic and insightful contribution to our discussions.

7. REFERENCES

[1] G. Baldassarre, S. Nolfi, and D. Parisi. Evolving mobile robots able to display collective behaviours. *Artificial Life*, 9:255–267, 2002.

[2] R. Bianco and S. Nolfi. Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce. *Connection Science*, 4:227–248, 2004.

[3] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, 1984.

[4] C. Darwin. *The Descent of Man*. John Murray, London, 1871.

[5] A. Eiben, R. Hinterding, and Z. Michalewicz. Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[6] A. E. Eiben, E. Haasdijk, and N. Bredeche. Embodied, on-line, on-board evolution for autonomous robotics. In P. Levi and S. Kernbach, editors, *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, chapter 5.2, pages 361–382. Springer, May 2010.

[7] A. E. Eiben, G. Karafotias, and E. Haasdijk. Self-adaptive mutation in on-line, on-board evolutionary robotics. In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW 2010)*, pages 147–152. IEEE Press, Piscataway, NJ, September 2010.

[8] S. Elfwing, E. Uchibe, K. Doya, and H. Christensen. Biologically inspired embodied evolution of survival. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation IEEE Congress on Evolutionary Computation*, volume 3, pages 2210–2216, Edinburgh, UK, 2-5 Sept. 2005. IEEE Press.

[9] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:396–407, 1996.

[10] D. Floreano, S. Nolfi, and F. Mondada. Co-Evolution and Ontogenetic Change in Competing Robots. In *Advances in the Evolutionary Synthesis of Intelligent Agents*. MIT Press, 2001.

[11] D. Floreano, N. Schoeni, G. Caprari, and J. Blynel. Evolutionary bits'n'spikes. In R. K. Standish, M. A. Bedau, and H. A. Abbass, editors, *Artificial Life VIII : Proceedings of the eighth International Conference on Artificial Life*, pages 335–344, Cambridge, MA, USA, 2002. MIT Press.

[12] E. Haasdijk, A. E. Eiben, and G. Karafotias. On-line evolution of robot controllers by an encapsulated evolution strategy. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, Barcelona, Spain, 2010. IEEE Computational Intelligence Society, IEEE Press.

[13] E. Haasdijk, A. E. Eiben, and A. F. T. Winfield. Individual, social and evolutionary adaptation in collective systems. In S. Kernbach, editor, *Handbook of Collective Robotics*, pages 295–336. Springer, Berlin, Heidelberg, New York, 2011.

[14] I. Harvey, P. Husbands, D. Cliff, A. Thompson, and N. Jakobi. Evolutionary robotics: the sussex approach. *Robotics and Autonomous Systems*, 20:205–224, 1996.

[15] M. Jelasity and M. van Steen. Large-scale newscast computing on the internet. Technical report, Vrije Universiteit Amsterdam, 2003.

[16] S. Kernbach, O. Scholz, K. Harada, S. Popesku, J. Liedke, H. Raja, W. Liu, F. Caparrelli, J. Jemai, J. Havlik, E. Meister, and P. Levi. Multi-robot organisms: State of the art. In *2010 IEEE International Conference on Robotics and Automation workshop*, pages 1 – 10. IEEE Press, 2010.

[17] J. L. Laredo, A. E. Eiben, M. Steen, and J. J. Merelo. Evag: a scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11(2):227–246, 2010.

[18] D. Marocco and S. Nolfi. Origins of communication in evolving robots. In *From Animals to Animats 9: Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior*, Lecture Notes in Computer Science, pages 789–803. Springer, Berlin, 2006.

[19] A. Martinoli. *Swarm Intelligence in Autonomous Collective Robotics from Tools to the Analysis and Synthesis of Distributed Control Strategies*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1999.

[20] G. F. Miller and P. M. Todd. The role of mate choice in biocomputation: Sexual selection as a process of search, optimization and diversification. In *Evolution and Biocomputation, Computational Models of Evolution*, pages 169–204. Springer-Verlag, 1995.

[21] U. Nehmzow. Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. In C. Prince, Y. Demiris, Y. Marom, H. Kozima, and C. Balkenius, editors, *Proceedings of The Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, number 94 in Lund University Cognitive Studies, Edinburgh, UK, August 2002. LUCS.

[22] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345 – 370, 2009.

[23] S. Nolfi. Evolving non-trivial behaviors on real robots: A garbage collecting robot. *Robotics and Autonomous Systems*, 22(3-4):187–198, 1997.

[24] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA, 2000.

[25] P. Nordin and W. Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behavior*, 5:107–140, 1997.

[26] A. Ostermeier, A. Gawelczyk, and N. Hansen. A derandomized approach to self adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380, 1995.

[27] M. A. Potter, L. A. Meeden, and A. C. Schultz. Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1337–1343. Morgan Kaufmann, 2001.

[28] H.-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[29] M. Tomassini. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[30] E. Tuci, M. Quinn, and I. Harvey. Evolving fixed-weight networks for learning robots. In *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*, pages 1970–1975. IEEE Computer Society, 2002.

[31] Y. Usui and T. Arita. Situated and embodied evolution in collective evolutionary robotics. In *Proceedings of the 8th International Symposium on Artificial Life and Robotics*, pages 212–215, 2003.

[32] R. A. Watson, S. G. Ficici, and J. B. Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, April 2002.