# Self-adapting Fitness Evaluation Times for On-line Evolution of Simulated Robots

C.M. Dinu
VU University Amsterdam
c.dinu@student.vu.nl

P. Dimitrov
VU University Amsterdam
p.dimitrov@student.vu.nl

Berend Weel
VU University Amsterdam
b.weel@vu.nl

A.E. Eiben
VU University Amsterdam
a.e.eiben@vu.nl

## ABSTRACT

This paper is concerned with *on-line* evolutionary robotics, where robot controllers are being evolved during a robots' operative time. This approach offers the ability to cope with environmental changes without human intervention, but to be effective it needs an automatic parameter control mechanism to adjust the evolutionary algorithm (EA) appropriately. In particular, mutation step sizes ($\sigma$) and the time spent on fitness evaluation ($\tau$) have a strong influence on the performance of an EA. In this paper, we introduce and experimentally validate a novel method for self-adapting $\tau$ during runtime. The results show that this mechanism is viable: the EA using this self-adaptative control scheme consistently shows decent performance without a priori tuning or human intervention during a run.

## Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

## Keywords

Evolutionary Robotics, Swarm Robotics, On-line Evolution, Embodied Evolution, Dynamic Environments

## Track

Artificial Life, Robotics, Evolvable Hardware

## 1. INTRODUCTION AND OBJECTIVES

Evolutionary algorithms (EAs) are popular heuristic optimisers, often used in robotics to develop robot controllers in an *off-line fashion* [10, **?**]. This means that an EA is employed to find a very good (albeit not necessarily optimal) controller before the real operational period of the robots in question, then this controller is deployed in the robots and remains unchanged while the robots are going about their tasks. The alternative modus operandi –that is used much less frequently– is to apply evolution in an *on-line fashion*, not before but during the operational period of the robots. In principle, this approach offers important advantages as it allows the adaptation of robot controllers on the fly, hence enabling robots to adjust to changing circumstances.

However, on-line evolution comes at a cost. In particular, by the very nature of this approach the user / experimenter is not able to configure the parameter values of the evolutionary mechanism on-the-spot, whereas it is widely known fact that the performance of evolutionary algorithms severely depends on the values of their parameters [3, 4]. This leaves the user with two options in general: 1) use robust parameter values that work well over a wide range of circumstances, 2) apply a parameter control mechanism that adjusts the values of the given parameter(s) during the run.

The goal of this paper is to investigate the second option. Previous studies have given strong indications that the most influential EA parameter in on-line EAs for the evolution of robot controllers is $\tau$, the time spent on a fitness evaluation [6]. This is certainly not surprising in light of the classic exploration-exploitation dilemma. Larger values of $\tau$ allow for longer testing of controllers, thus increasing the reliability of fitness estimates. However, this EA is running in real time, spending more time on one fitness evaluation implies that fewer evaluations can be performed in any given time interval. In other words, high / low values for $\tau$ imply less / more exploration, while the quality of information (that is important for good exploitation) shows the opposite trend.

Our investigations address the (im)possibilities of controlling the values of $\tau$ during the run of an on-line EA applied to evolve controllers for robots in a swarm. To be specific, we introduce a new mechanism for self-adapting the values of $\tau$ and perform experiments to answer the following questions:

1. How does a self-adaptive $\tau$ affect the performance of the robot swarm in a static environment?
2. How does a self-adaptive $\tau$ affect the performance of the robot swarm in a changing environment?

In both cases we use fixed values for $\tau$ as benchmark and compare the performance achieved by various fixed $\tau$ values to the co-evolving $\tau$ algorithm for the same scenario. It should be noted however, that we are not trying to find an 'ideal' setting for the given scenario's. Thus, we are not trying to show that some setting (a specific value of $\tau$ or the self-adaptive scheme) is better. Rather, we are seeking


*GECCO'13,* July 6-10, 2013, Amsterdam, The Netherlands.

empirical evidence to assess the practicability of the self-adaptive $\tau$ mechanism, keeping the relevant application scenarios in mind, where problem tailored tuning by the user is not feasible. The main research question is thus the following.

> Is the use of a self-adaptive $\tau$ practicable? That is, does it enable the system to perform reasonably and to adjust to changes without any human intervention?

This motivates our basic methodological attitude (that is somewhat different from the mainstream Evolutionary Computing approach): the fixed values of $\tau$ are not a set of benchmarks to beat, but are necessary to provide a good indication about 'reasonable' performance.

The rest of this paper is structured as follows. In Section 2 we briefly discuss related work with a special focus on on-line evolutionary algorithms in (swarm) robotics. The system setup, including the definition of the new mechanism to self-adapt $\tau$, is presented in Section 3. After this, we exhibit and discuss the experimental results in Section 4 and we wind up the paper with the usual Conclusions and Future Work in Section 5.

## 2. RELATED WORK

Our work is related to both swarm and evolutionary robotics. Swarm robotics is an area closely related to Swarm Intelligence, where a swarm of robots can collaborate in solving complex tasks in a scalable and robust manner. While swarm-robots often have the ability for physical self-assembly, the robots in our study only collaborate in exchanging their active solutions during parent selection (in a panmictic fashion).

The area of evolutionary robotics is in general described by Nolfi and Floreano in [10]. To distinguish various options regarding the evolutionary system Eiben et al. proposed a naming scheme based on when, where and how this evolution occurs [2]. The resulting taxonomy distinguishes between design time and run-time evolution (off-line vs. on-line) as well as between evolution inside or outside the robots themselves (on-board vs. off-board).

Traditionally, robot controllers are evolved in an off-line fashion, using an evolutionary algorithm searching through the space of controllers and only calling on the actual robots when a fitness evaluation is required. Evolving the robot controllers on-line and on-board is a fairly new approach in evolutionary robotics, initiated by the seminal paper of Watson et al.[12]. There a system was presented where a population of physical robots (i.e. their controllers) autonomously evolved while situated in their task environment. Since then the area of on-line on-board evolution in swarm-robotics, as classified in [2], has gained a lot of momentum.

While several studies [9, **?**] explore the concept of evolving robot controllers on-board and online, in both static and dynamic environments, the evaluation time ($\tau$) is often arbitrary or highly tuned before the run. Traditionally, each controller is evaluated for a fixed period of time before its fitness is finally computed.

On-line evolutionary robotics has been discussed in studies such as [12], where several robots evolve at the same time, exchanging parts of their genotypes when within communication range. [7] compares an encapsulated and a distributed version. In the latter, each robot has one active solution (genotype) and a cache of genotypes that are active in neighbouring robots. Parent selection is achieved through a binary tournament in each robot's cache. If the new solution (candidate) is better than the active, it replaces the active solution. The work compares this algorithm with a panmictic version, where parents are selected (again using binary tournament) from the combined active solutions of all robots. This relates to our algorithm in that parents are selected over the entire set of active solutions in all robots, whereas survivor selection happens locally and individually for each robot.

A bit more remotely related area of existing work is that of evolutionary optimisation in dynamic environments [1, **?**]. This is similar to our on-line on-board evolutionary algorithm mainly because the actual (on-line) performance is more important than the end result (off-line performance).

Regarding the specifics of the given evolutionary algorithms at work, the main issue here is the adequacy of parameter values. In particular, parameter tuning and parameter control are the relevant areas and our work falls clearly in the latter category. In terms of the widely used taxonomy, cf. [3], we introduce and investigate a self-adaptive scheme by adding the length of the evaluation time ($\tau$) as a parameter to the genome and thus making it co-evolve with the robot controllers.

## 3. SYSTEM SETUP

### 3.1 Scenario and Simulated Environment

Testing the viability of the self-adaptive $\tau$ mechanism requires a scenario that challenges an evolutionary algorithm sufficiently so that it exhibits significantly different behaviour for short vs. long evaluation times. To this end, we employ a variant of the classic foraging scenario. The main premise is the existence of a resource that 'grows' in atomic units in a finite arena. Using an agricultural metaphor, we can call them plants and postulate that a swarm of robots is tasked with collecting as many as possible. Unfortunately, the distribution of plants is not known, and is likely to change with time as different plant species come in and out of season. The robots therefore have to be able to both find the optimal foraging behaviour for an unforeseen plant distribution, as well as adapt to potentially radical changes over time. We also assume that foraging / collecting operations must start as soon as possible, without time to do trial runs and tune the parameters of the swarm's evolution.

In order to simulate the behaviour of the evolving swarm of robot foragers, we used `roborobo`[**?**], a fast and highly configurable evolutionary robotics simulator, which we subsequently customised for our scenario.

The foraging ground surrounding the swarm is modelled as a rectangular arena featuring a number of obstacles that may represent buildings or rock formations (see Figure 1). Robots cannot pass through walls or other robots, nor can they leave the arena - this latter programming constraint is modelled as a virtual wall surrounding the foraging field.

The arena is also littered with a number of pellets representing the plants that the robots need to collect. The plants are placed randomly, according to a uniform or non-uniform distribution, and their number is kept constant. As soon as a plant is consumed by a robot, another one will be placed in random location according to the distribution.

Each possible distribution of the plants challenges the ro-

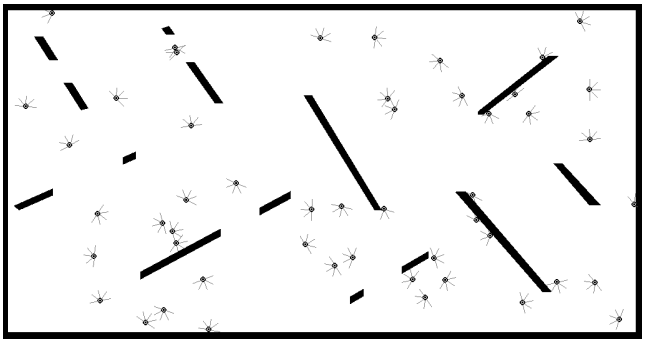


**Figure 1: The environment as simulated in roborobo. The black lines represent obstacles in the environment, while the grey circles are individual robots with their sensor rays visible.**

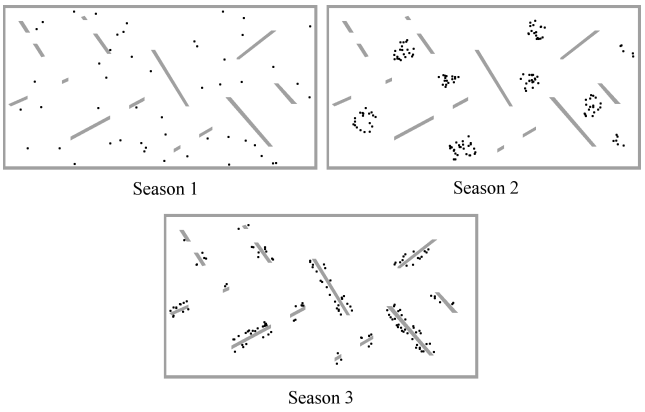



**Figure 2: Plant distributions for the three seasons. Season 1 is a scarce environment with few plants, uniformly distributed. Season 2 is a dense environment with the plants distributed in patches. Season 3 is a dense environment with the plants distributed close to obstacles in the environment.**

bots differently and encourages different kinds of behaviours for optimal foraging. For our experiments, we have devised three different plant distributions, which we have dubbed 'seasons' (see Figure 2):

- *Season 1 (Uniform, sparse)* features 50 plants in a *uniform* distribution. The sparse layout means that robots are unlikely to run into plants by blind exploration using their short-range sensors. Instead, they are encouraged to develop the use of the long-range 'smell' sensor so that they can find plants effectively. (See technical details later on.)

- *Season 2 (Patches, dense)* has 150 plants laid out in dense 'patches' all over the arena, a *non-uniform* distribution. This time, robots do not need their smell sensor as much; rather, it is simpler to develop a method of 'lingering' around a patch once the robot has run into one accidentally.

- *Season 3 (Close to walls, dense)* features another interesting non-uniform distribution of 150 plants. In effect, plants will now only appear near physical walls - this could represent mushrooms and other species that thrive in the shade. The obvious incentive for the robots is to develop wall-following behaviour, but this must be tempered with a touch of randomness, as it is maladaptive to get stuck following the virtual wall around the arena, which harbours no plants whatsoever.

**Note**: The number of plants differs between seasons, but this does not pose a problem when comparing scenarios. As shown in the Methods section, our performance indicators account for this parameter variation.

## 3.2 Robot Architecture

The robots in our experiment have a cylindrical ground-based morphology and are fitted with the following equipment:

- *5 proximity sensors*, placed at $0°$, $\pm 45°$ and $\pm 90°$ with respect to the direction the robot is travelling. Each sensor reports the distance to the closest robot or wall in the corresponding direction, up to a maximum of 1.5 times the robots' diameter, beyond which they saturate. These sensors do not 'see' plants.

- *5 plant sensors*, identical in placement, functionality, and range to the proximity sensors, except they detect plants instead of solid obstacles.

- *1 non-directional plant proximity sensor*, dubbed the 'smell' sensor. It reports the distance to the nearest plant in any direction. This sensor has a much larger range than the previous sensors (14 robot diameters), but, being non-directional, requires the robot to evolve an intelligent strategy of finding its way to the nearest plant by distance alone.

- *Two independently spinning wheels.* The differential drive design allows for a wide variety of movements to be performed by controlling only two parameters, namely the rotation speed for each wheel: the robot can drive forward or backward at different speeds, perform sharp or gentle turns, and even spin in place.

- *A radio*, used by the evolutionary algorithm for exchanging solutions. The robots do not otherwise cooperate or communicate.

- *A plant collection and storage mechanism* that automatically collects the plants a robot runs over. The storage bin is large enough to never saturate over the evaluation time of a robot.

The link between the sensors and the effectors is provided by a three-layer *neural network controller*, featuring 11 input neurons, a bias node, 3 neurons in the hidden layer, and 2 output neurons. Each input neuron is connected to a sensor and experiences activation between 0.0 (object detected at minimum distance) and 1.0 (object at maximum distance or beyond). The output neurons are connected directly to the wheel motors in a common-differential pattern, i.e.:

$$wheelSpeed_L = maxSpeed \cdot (output_1 + output_2)$$
$$wheelSpeed_R = maxSpeed \cdot (output_1 - output_2)$$

As the controller drives the robot directly instead of selecting from a set of predefined strategies, the robots can evolve relatively complex behaviours with little in the way of biases or constraints arising from their design.

## 3.3 Evolutionary Algorithm

The experiments are carried out in simulations in a swarm of 50 robots that use a so-called hybrid on-line on-board evolutionary algorithm specified as follows.

- *Representation:* A genome is a vector of $2 \cdot n + 1$ genes composed as follows. Variables $x_1, x_2, ..., x_n$ directly encode the $n$ weights of the neural network, that is, the robot controller. In the present experiments $n = 42$. Furthermore, the genes denoted by $\sigma_1, \sigma_2, ..., \sigma_n$ stand for the corresponding mutation step sizes and an additional gene $x_\tau$ is used for encoding $\tau$, the time allowed for fitness evaluation (the mapping between $x_\tau$ and $\tau$ is described in section 3.4).
- *Evaluation function:* The fitness is defined as the *amount of plant matter collected per unit of time*, i.e.:

  $$F = \frac{100 \cdot N_{plantsCollected}}{\tau}$$

  where 100 is the 'score' per plant collected. Note how this performance measure is entirely objective (free of any behavioural suggestions for the robots) and fair with respect to the evaluation time, i.e. similarly performing robots will achieve similar fitness regardless of their $\tau$ value.
- *Population:* Each robot evolves its own local population of $\mu = 3$ genomes individually. Parent selection is, however, performed on the entire set of genomes in all robots. Therefore a distinction has to be made between local and global population. Robots use a variant of the newscast [8] algorithm to exchange genomes.
- *Parent selection mechanism:* Parents are selected using a binary tournament over all genomes in all robots.
- *Variation operators, recombination and mutation:* Mutation is a Gaussian perturbation with noise drawn from $N(0, \sigma_i)$ for $x_i$. The step sizes $\sigma_i$ are self-adapted using the standard formulas as in [5, 11]. Recombination is achieved through averaging crossover with probability 1.0.
- *Survivor selection mechanism:* At the beginning of each evaluation, a random choice (with a chance $\rho = 0.5$) is made between creating a new controller by mutation or re-evaluating an existing one. Once the evaluation is complete, the evaluated controller is compared to the local population of $\mu$ other controllers at that robot with respect to their fitness. If it is better than the worst controller, it replaces it.
- *Initialisation:* All genes are initialised by drawing a real number from the [0,1] interval with uniform probability.
- *Termination condition:* The simulation ends after $N = 15,000,000$ iterations.

Following the taxonomy specified in [2], the present evolutionary algorithm falls under the **hybrid** category because it is encapsulated –each robot evolves its own inner population of $\mu$ genomes– and also distributed as there is an exchange of genotypes between the robots. A specific feature of our hybrid scheme is the combination of global parent selection (considering the entire set of genomes in all robots), and local survivor selection (restricted to the local population in each individual robot). This scheme proved to be very powerful in [7].

## 3.4 Self-adapting the Evaluation Time

A self-adaptive mechanism that co-evolves the values of $\tau$ with the robot controllers poses a particularly difficult problem due to the unique effects of $\tau$ on the evolutionary process itself. Let us assume we treated $x_\tau$ as the other genes $x_i$ and computed the corresponding evaluation time in a simple manner like:

$$\tau = \tau_{min} + x_\tau \cdot (\tau_{max} - \tau_{min})$$

where $\tau_{min}$ and $\tau_{max}$ are loaded with reasonable defaults. Given that $x_\tau$ varies uniformly between 0.0 and 1.0, it may seem that this would provide a fair test of each possible evaluation time value between $\tau_{min}$ and $\tau_{max}$. However, upon closer inspection, two serious problems become apparent:

- Larger $\tau$ values are less 'distinct' than their peers in the lower part of the spectrum. For instance, a system will behave very differently for $\tau = 50$ vs. $\tau = 100$, but almost the same for $\tau = 2000$ vs. $\tau = 2050$, even though the spacing is the same (50). Therefore, probabilities should be skewed so that there is less investment in a specific $\tau$ value as we move higher up the spectrum.
- Larger $\tau$ values will be changed less often than their smaller peers, as the robots spend proportionally more time until the next reproduction. Thus, given equal selection probabilities, a larger amount of time will be allocated to evaluating larger $\tau$'s, which is temporally unfair. More precisely, if $p(t)$ is the probability for selecting $\tau = t$, then the fraction of the total time dedicated to evaluating that particular $\tau$ value is:

  $$f(t) = \frac{p(t) \cdot t}{\sum_{\tau=\tau_{min}}^{\tau_{max}} p(\tau) \cdot \tau}$$

  It is easy to see that if all $p(t)$ are equal, $f(t) \sim t \ \forall t$. What we would like instead is for all $f(t)$ to be equal. A quick calculation shows us that for this to occur, the condition:

  $$p(t) \sim \frac{1}{t} \tag{1}$$

  must hold for all $t$.

Our solution for addressing these biases is to make the relation between the $\tau$ gene and the evaluation time *exponential* instead of *linear*:

$$\tau = e^{\alpha \cdot x_\tau + \beta}$$

where $\alpha = \ln \tau_{max} - \ln \tau_{min}$ and $\beta = \ln \tau_{min}$. It can be seen that $\tau$ still varies monotonically from $\tau_{min}$ to $\tau_{max}$ as $x_\tau$ goes from 0.0 to 1.0. However, assuming $x_\tau$ is picked uniformly from the [0,1] interval, we have that:

$$p(t) = \frac{d}{dt}\mathrm{CDF}(t) = \frac{d}{dt}\frac{\ln t - \beta}{\alpha} = \frac{1}{\alpha t}$$

Note that equation (1) applies, therefore this method is temporally fair.

Furthermore, let us consider any arbitrary value $t$ for $\tau$. What other values of $\tau$ are nearly equivalent to this setting?

A reasonable idea is to say that $t'$ is nearly equivalent to $t$ if $t < t' < \gamma t$, where $\gamma$ is an arbitrarily chosen factor. Thus, if $t = 1.1$ is similar to $t = 1$, but $t = 1.4$ is considered different, then likewise $t = 1100$ is equivalent to $t = 1000$ and $t = 1400$ is different. Thus, the 'equivalence class' of any value $t$ is the interval $[t, \gamma t]$.

The probability of picking any of the values in an arbitrary $t$'s equivalence class is:

$$P(t) = \int_t^{\gamma t} p(\tau)d\tau = \int_t^{\gamma t} \frac{1}{\alpha\tau} d\tau \ = \frac{1}{\alpha} \ln \gamma$$

Note that the $P(t)$ is constant for all $t$. Thus, our method is also fair in the sense that intervals of nearly equivalent $t$ values are assigned the same selection probability.

# 4. EXPERIMENTS

The set of fixed $\tau$ values we have chosen for our experiments is $\{64, 128, 256, 512, 1024, 2048\}$. Note that they are in a geometric progression, in accordance to our 'equivalence class' principle described in section 3.4 - in effect, each value $t$ is a representative for all $t' \in [t, \gamma t)$, where $\gamma = 2$. A finer scale may have yielded yet more detailed data, but this would have hampered visualisation and used inordinate amounts of computing time.

In order to evaluate performance in the *static* case, we use scenarios featuring a single season all throughout (one experiment for each season, for a total of 3 experiments). For the *dynamic* case, we use scenarios featuring three seasons in succession for an equal length of time and we execute one experiment for each of the 6 possible permutations 1-2-3 through 3-2-1. Each such scenario contains two transitions that provide an opportunity to test an algorithm's capacity for recovery in the face of changing conditions.

To obtain statistically significant results, we performed 30 trial runs for each experimental configuration[1]. Unless otherwise stated, all statements referring to the performance or behaviour of the EA refer to those measures averaged over all of the trials.

### *Measuring Performance*

The typical definition of global/average fitness per generation is found inadequate for the current situation; first of all, for different values of $\tau$ in a fixed-$\tau$ algorithm, generations advance at a different pace. As a result, a comparison between generations belonging to different $\tau$ is useless, as they represent potentially different points in time. Moreover, in case of self-adaptive $\tau$ values the very idea of a generation is no longer applicable, as each robot evolves independently and may undergo a fitness evaluation at any given time. Therefore, the only applicable coordinate is *time*, the only measure that flows equally for all of the experiments.

In order to have a fitness value for every time instance, and not just at the moments when evaluations are completed, we choose to define the *the fitness of a robot at time T, f(T)* as the fitness that was assigned to it at the most recently completed evaluation. This definition can be further extended to the collective level: *the fitness $F(T)$ of a swarm at time $T$* is the mean of the individual fitness values of the robots in the swarm:

$$F(T) = \frac{1}{N} \sum_{i=1}^{N} f_i(T)$$

Although this definition is sufficient for evaluating the evolving $\tau$ algorithm, we found that the function is very noisy on a small time scale and produces nigh-unreadable graphs. To account for this, when graphing the fitness we applied a smoothing method using the formula:

$$F_W(T) = \frac{1}{W} \int_{T-W}^{T} F(t)dt$$

where $W$ is the *smoothing window size* parameter. For our experiments, $W = 50000$.

Using this measure of global fitness, we note two essential evolutionary algorithm performance measures that can be derived from it:

- *The plateau fitness*: The maximum value achieved for the global fitness during the simulation.
- *The convergence time*: The time needed to reach 90% of the plateau fitness.

Note that we expect seasons to have different plateau fitness values. Thus, when comparing behaviour across seasons, we will apply a normalisation procedure.

## 4.1 Results and Discussion

We have performed experiments in nine different scenarios, three static scenarios and six dynamically changing ones.

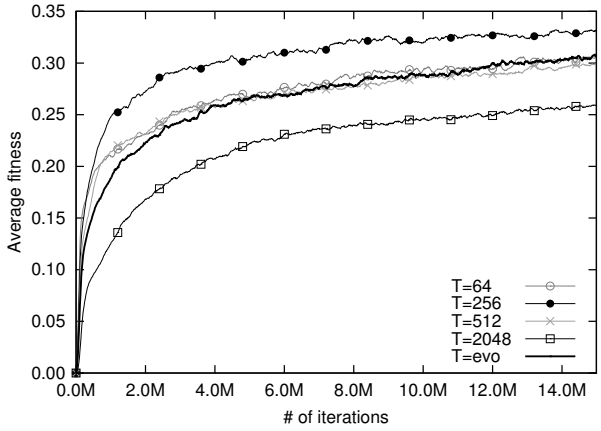
### *Static Scenarios.*

The results for the three static scenarios are shown in Figure 3. Note that, for the sake of readability, we omitted the traces for $\tau = 128$ and $\tau = 1024$, as these closely follow other traces in the graph and exhibit no notable characteristics. An unabridged, full colour version of the graphs can be found on the authors' website.

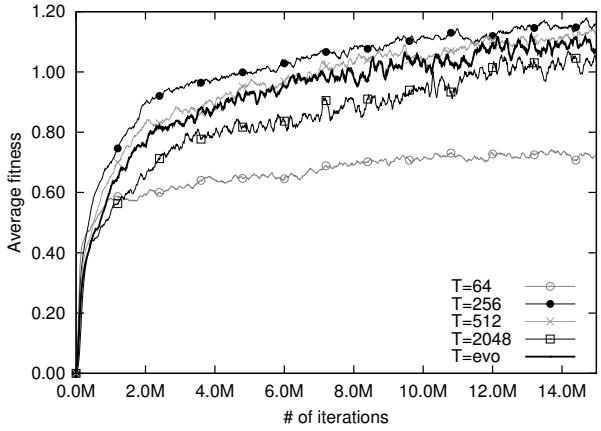
For all three seasons, we can see that the traces are more or less monotonic and exhibit fast growth in the beginning, followed by settling into either a plateau or a slowly increasing linear evolution. This indicates that the EA is indeed successful, with robots quickly developing the behaviour required for foraging. We confirmed this by observing the simulated robots in `roborobo` and noting that they exhibit reasonable behaviour consistent with the season's specific challenge (finding plants by smell, patch exploitation, wall following), as well as more general adaptive behaviours such as always driving forwards to make better use of the sensors, avoiding collisions with other robots, etc.

We note that the traces for season #2 exhibit considerably more variance than the other seasons, especially for higher $\tau$ values. This occurs due to competition effects between the robots as they become highly evolved: a successful robot will find a patch and exploit it efficiently, quickly devouring all of the plants within. Although plants will reappear in all patches, the selection effects of being so quickly consumed in one patch will cause them to effectively be redistributed among the other patches within a very short time. This temporarily 'exhausts' the patch and causes a momentary dip in fitness as many of the robots scramble to find a new patch until the current one can regenerate. This phenomenon repeats over and over again, leading to the jagged appearance of the fitness traces in this scenario.

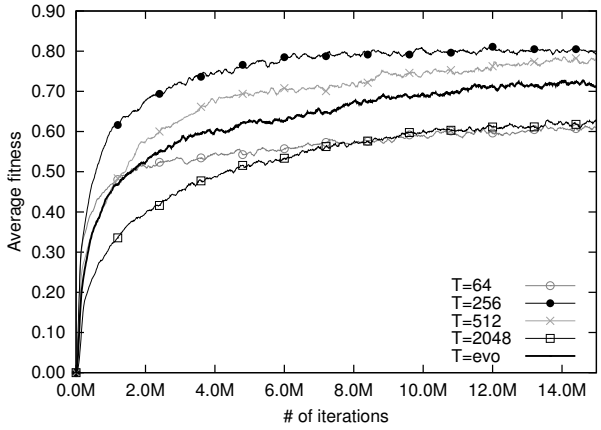
[1]For consistency and reproducibility, the random number generator's seed was controlled. The sources, configurations and data for our experiments can be found at the authors' web site at `http://www.few.vu.nl/~bwl400`




**(a) Season 1**




**(b) Season 2**




**(c) Season 3**

**Figure 3: Fitness in single season scenarios. The x-axis shows the number of iterations, the y-axis the average fitness of the controllers at a certain iteration averaged over all runs. In all three seasons the evolved $\tau$ shows a reasonable performance.**
**Note that uninteresting traces for $\tau = 128$ and $\tau = 1024$ have been omitted for readability.**

The value of $\tau$ has a marked effect on the performance of the algorithm. The pattern we can observe is the EA behaving poorly for low $\tau$'s (64, 128), then improving in performance as the $\tau$ increases, ultimately reaching an optimum ($\tau = 256$ for all three seasons), and then declining again as the $\tau$ is increased further (512, 1024, 2048). This is consistent with our expectations of lower $\tau$'s causing poor performance due to the reduced solution quality, as well as higher $\tau$'s being handicapped due to getting less exploration done per unit of time. The value $\tau = 256$ is a clear winner in terms of both plateau fitness and convergence time, as it appears to strike a good balance between these two factors. The presence of a clear optimum is also a good omen for the self-adaptive $\tau$ strategy, as it suggests that the performance vs. $\tau$ landscape is well behaved and roughly unimodal, allowing for efficient evolutionary search of a good $\tau$ value.

The performance of the self-adaptive $\tau$ algorithm is found to consistently lie between the best and the worst traces, being average for seasons #1 and #3 and above average for season #2. This confirms that our method is a viable alternative for situations where no effort can be expended for tuning the system in search of the optimal $\tau$ value: its performance may not be optimal, but it is acceptable and consistent. It is not clear why the performance of the self-adaptive $\tau$ method is so much better for the second season, but we suspect it might be due to the performance vs. $\tau$ landscape being flatter for this scenario, with success having less to do with strategy development and more to do with keeping a consistent controller while a patch is being exploited.

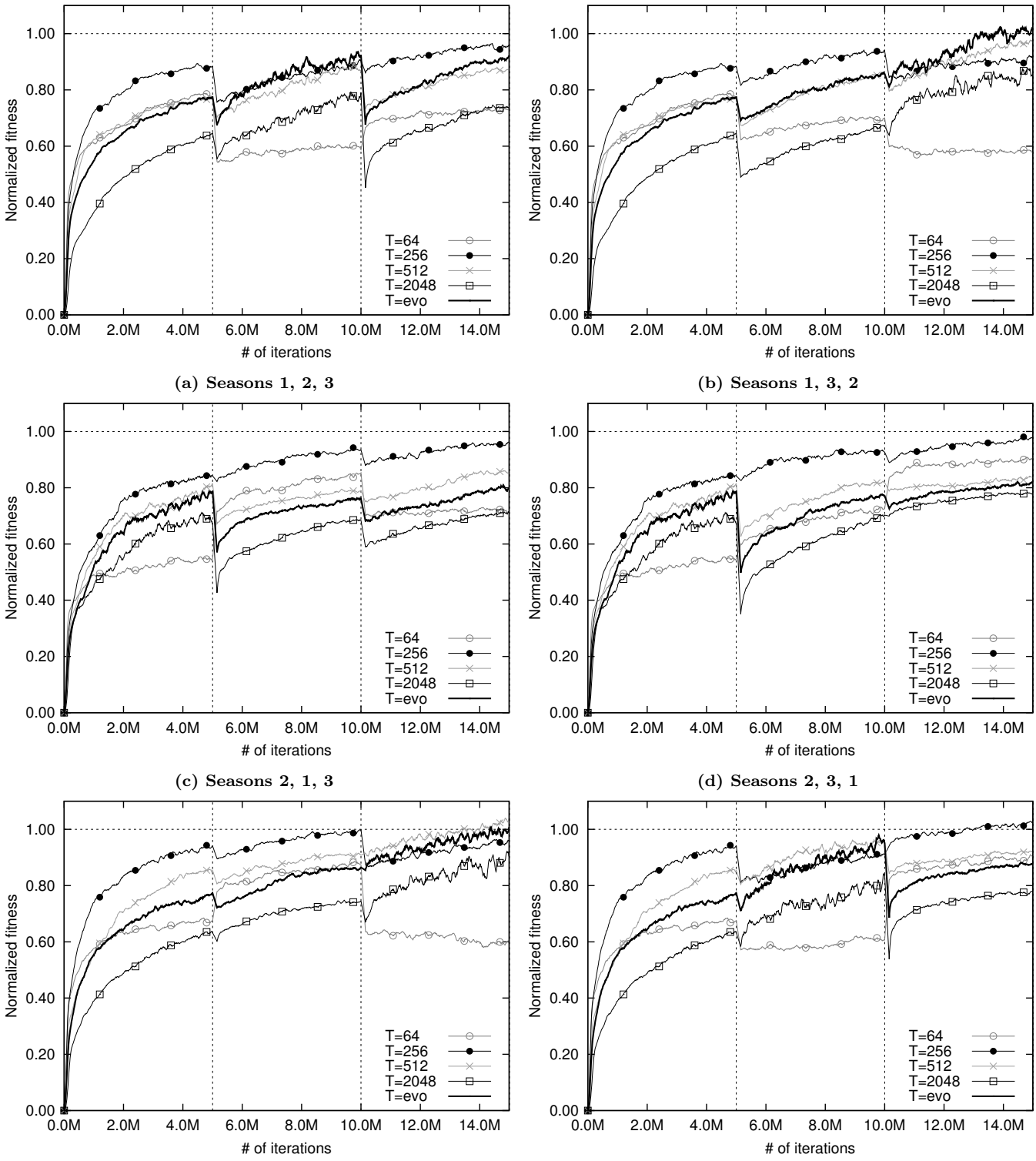
### *Dynamic Scenarios.*

The results for the six dynamic scenarios are shown in Figure 4. Note that in order for the data in the multi-season graph to be easily understandable, the data for each season has been normalised with respect to the maximum fitness achieved for that season in the static experiments.

As expected, for the first third of the simulation timespan, the performance traces behave the same way as for the static case of the first scenario in the sequence. As we reach the transition points one and two thirds into the simulation time, the fitness values show a dip as the robots' evolved behaviour within the previous season becomes ill-adapted for the new season. This dip seems to be most dramatic for the large $\tau$ values when switching to and from the second scenario. This is probably due to the 'feeding frenzies' that evolved robots can engage in during the second season, which keep the overall fitness up due to so many plants being accumulated by some of the robots. This is no longer possible in seasons #1 and #3, or even in season #2 if a low enough $\tau$ is used so that a feeding frenzy is interrupted by switching to another controller.

As the robot swarm recovers and starts evolving towards the new optimal behaviour, we may expect its evolution to be markedly different from the static case as this time the robots already have a 'cache' of potentially reusable evolutionary work as opposed to starting 'cold' with a completely random genome. For instance, behavioural elements such as avoiding collisions or not driving backwards are general enough to provide a benefit within any season, and even season-specific adaptations such as using the smell sensor can be repurposed for a new situation. However, the overall behaviour and ranking of the graphs for any given season seems to stay much the same regardless of their position in the sequence.

Interestingly, the best performance of the static case is exceeded in 3 of the 6 dynamic scenarios. This suggests




**(a) Seasons 1, 2, 3**

**(b) Seasons 1, 3, 2**

**(c) Seasons 2, 1, 3**

**(d) Seasons 2, 3, 1**

**(e) Seasons 3, 1, 2**

**(f) Seasons 3, 2, 1**

**Figure 4: Fitness in multi-season scenarios. The x-axis shows time in iterations, the y-axis the normalised fitness of a controller. Normalisation is done by dividing the fitness by the maximum fitness obtained in the single season scenario for each respective season. The self-adaptive $\tau$ shows it can cope with the changes in environment quite decently, in some cases even out-performing any of the pre-defined $\tau$ values.**

that switching between different challenges at runtime, while usually considered harmful, can also have a positive effect. Probably we are witnessing a 'stepping stone' phenomenon, where a solution adapted for problem X can feature elements that help with problem Y, while these features are hard to reach by evolving towards Y directly.

The self-adaptive $\tau$ scheme manages to hold its own for the adaptive case as well. Performance is rarely worse than average, and, in fact, the method even manages to achieve the best performance overall during most runs of season #2 that do not occur first in the scenario. Season #1 appears to be pose the greatest trouble to the self-adaptive method, with its ranking going down from being tied with $\tau = 64$ to being second to last. For season #3, the self-adaptive approach ranks either third, as in the static case, or second, beating $\tau = 512$. Overall, the plateau fitness of the self-adaptive $\tau$ approach achieves between 75% and 100% of the maximum achieved fitness for the static case.

# 5. CONCLUSIONS AND FUTURE WORK

In this paper we have explored the possibility of automatic run-time adaptation of the fitness evaluation time $\tau$ in on-line evolutionary robotics. In particular, we investigated how this problem can be solved by a self-adaptive mechanism for $\tau$, such that its values co-evolve with the robot controllers in a swarm. Our contributions include an analysis of the difficulties involved in self-adapting (co-evolving) a gene that interacts so intimately with the EA, as well as a novel, mathematically sound method for overcoming these issues.

This method was validated by a series of experiments with a robot simulator in nine different scenarios, including static as well as dynamically changing environments. Set against fixed values of $\tau$, our self-adaptive $\tau$ method proved to be viable, delivering decent performance and being capable of adaptation. In other words, the performance loss w.r.t. a hand-tuned $\tau$ is by all means acceptable making this a practicable solution is cases where a priori tuning of EA parameters is not feasible.

Future work should expand the comparison to yet more and more complex scenarios, featuring explicit cooperation or competition interactions between the robots and perhaps even organism formation. Given that our solution does not always converge on the 'optimal' $\tau$ value, it seems that we need to investigate more deeply into the subtle interactions between the self-adaptive $\tau$ mechanism and the evolutionary process as a whole. Preliminary work suggests that $\tau$ values should be kept more stable for their benefits to become apparent, therefore we want to explore strategies for slowing down $\tau$ variation.

## Acknowledgments

This work was made possible by the European Union FET project SYMBRION. under grant agreement 216342.

# 6. REFERENCES

[1] J. Branke and S. Kirby. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Boston, 2001.

[2] 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.

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

[4] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[5] A. E. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin Heidelberg, 2003.

[6] E. Haasdijk, S. Smit, and A. Eiben. Exploratory analysis of an on-line evolutionary algorithm in simulated robots. *Evolutionary Intelligence*, 5:213–230, 2012.

[7] R.-J. Huijsman, E. Haasdijk, and A. E. Eiben. An on-line on-board distributed algorithm for evolutionary robotics. In *Proceedings of Artificial Evolution, 10th International Conference, Evolution Artificielle (EA 2011)*, pages 119–131, Angers, France, 24-26 October 2011.

[8] M. Jelasity and M. V. Steen. Large-scale newscast computing on the internet. Technical report, 2002.

[9] S. Kernbach, E. Meister, O. Scholz, R. Humza, J. Liedke, L. Rico, J. Jemai, J. Havlik, and W. Liu. Evolutionary robotics: The next-generation-platform for on-line and on-board artificial evolution. *2009 IEEE Congress on Evolutionary Computation*, pages 1079–1086, 2009.

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

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

[12] 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.