Evolving Systems manuscript

Controlling maximum evaluation duration in on-line and on-board evolutionary robotics

On-the-fly control of maximum evaluation duration in on-line and on-board evolutionary robotics in highly dynamic and uncertain environments

Atta-ul-Qayyum Arif · D.G. Nedev · Evert Haasdijk

Abstract On-line evolution of robot controllers allows robots to adapt while they perform their proper tasks. In our investigations, robots contain their own self-sufficient evolutionary algorithm (known as the encapsulated approach) where individual solutions are evaluated by means of a time sharing scheme: an individual controller is given the run of the robot for some amount of time and fitness corresponds to the robot's task performance during that period.

In this paper, we propose and provide a detailed analysis of two *on-the-fly* control schemes to set the *evaluation time* in highly dynamic scenarios with completely different tasks. One scheme, called the roulette-wheel selection scheme, stochastically selects evaluation time from promising intervals similar to multi-armed bandit schemes. The other scheme, named H-Rule, tweaks the evaluation time using specific heuristics. Our experiments show that H-Rule gives stable performance in different scenarios and can serve as a viable alternative to pre-selected optimal evaluation time.

Keywords on-line evolution · parameter control · evolutionary robotics · racing · on-board evolution · heuristics

1 Introduction

Evolutionary robotics incorporates the principles of evolution to develop controllers for autonomous robots.

Atta-ul-Qayyum Arif
Vrije Universiteit Amsterdam
E-mail: naarads@gmail.com

D.G. Nedev
MonetDB Solutions*
E-mail: dimitar.nedev@monetdbsolutions.com

Evert Haasdijk
Vrije Universiteit Amsterdam
E-mail: e.haasdijk@vu.nl

Conventional evolutionary robotics focuses on *off-line* evolution, where controllers are optimized in a separate development stage *before* proper development of the robots and there is no subsequent adaptation – at least not through evolution of those controllers [Haasdijk et al., 2011]. In *off-line* training includes selection, variation and evaluation of genomes, which is usually done through computer simulations, results in a nearly optimized controller that is ultimately deployed on robots to tackle their actual task. Alternatively, evaluations could be done by uploading the controller onto a real robot and fitness is calculated from the deployed controller's actual performance. This latter form of evolution is commonly termed as 'embodied evolution'. Strictly speaking, a complete evolutionary process is not truly embodied in this form of evolution since both selection and variation do not occur on-board, i.e. inside robot [Eiben et al., 2010].

On-line evolution, on the other hand, provides continuous adaptation as the robots perform their tasks after deployment. The major difference with *off-line* evolution is that in this case controllers are evaluated, selected, and modified as they perform their tasks and not in a separate training phase. When dealing with the implementation of on-line evolution in robotics, one finds three different flavors, namely: encapsulated, distributed or hybrid [Eiben et al., 2010]. Some work on *on-line* evolution in robotics include the works by [Nordin and Banzhaf, 1997], [Floreano and Mondada, 1998] and [Watson et al., 1999]. The authors describe physical robots with controllers are designed on-the-fly through an evolutionary process driven by feedback from the on-board sensors.

The on-line flavor of evolution poses a number of challenges. One of these challenges includes the designers' inability to foresee all the operational complexities of unknown and possibly highly dynamic environments, in which robots must adapt autonomously. Be-

cause the performance of evolutionary algorithms depends on proper parameter settings [Hart and Belew, 1991], on-line evolution requires either very robust parameter settings that perform well over a wide range of problems or some way to control parameter settings on-the-fly, as evolution progresses.

An important parameter in on-line evolution of robot controllers is the *evaluation time* of robot controllers. A robot's evolving population contains multiple controllers, but at any given time only one of them can actually control the robot. Therefore, a time-sharing mechanism is commonly used to activate and evaluate controllers one by one. A selected controller runs for a certain amount of time and the robot's task performance over that period determines the controller's fitness. Subsequently, another controller is activated and evaluated, and so on. The duration of these evaluations has a profound effect on the quality of the evolving solutions [Haasdijk et al., 2012]. Too short evaluation times would not give controllers enough time to prove their mettle and even good candidates may be ignored frequently. Overly lengthy evaluations could cause the algorithm to spend a lot of time on bad genotypes; good genotypes would get less time to evaluate and develop further. Finding good values for evaluation time on-the-fly is of vital importance for on-line evolution, particularly in dynamic scenarios, where no single parameter value may be appropriate. Even with improving hardware, the performance of an agent remains highly dependent on the time for which each controller is evaluated or used. For example, if bad controllers are evaluated 40% of the overall time, an agent will not appear to perform well, as only in 60% of the time it does its job well, even on fast hardware. This is because evaluation time is independent from how fast any computation is actually done, rather it depends on how often and how long controllers are evaluated.

In this paper we propose and assess two schemes to set evaluation time on-the-fly. We use simulated robot experiments to investigate whether these provide a viable alternative to pre-set static evaluation times in dynamic environments. We extend the (μ+1) ON-LINE algorithm as presented in [Haasdijk et al., 2011] to include these schemes and run a series of experiments where agents adapt to changing tasks. The real-world scenarios present us with various challenges, since a dynamic and non-stationary system exhibit the randomness of real-world phenomena. This make it nearly impossible to estimate what might happen in the future during the task execution nor can this be interpolated from a model based on historical data [Sayed-Mouchaweh and Lughofer, 2012]. Modeling a real-world system to unpredicted chaotic events, especially relevant if the task changes over time [Kasabov, 2007].

Dynamic scenarios, where an agent is faced with multiple tasks and a changing environment is closer to real-life situations a robot will face. The research in the paper is based on the premise that dynamic adaptation of evaluation time will be beneficial to a wide range of tasks that a robot needs to resolve. This is in contrast to a single, static parameter, as that setup is optimal for a selected subset of objectives. As it can be seen from the results, different *evaluation time* values best suited for different tasks, while a single dynamic scheme (H-Rule) allows the agent to adapt to the changing goals with outside interaction.

2 Related work

Evolutionary robotics is a novel technique for the automatic creation of autonomous robots. Inspired by the Darwinian principle of selective reproduction of the fittest, it views robots as autonomous artificial organisms that develop their own skills in close interaction with the environment and without human intervention. Each robot is free to act according to its genetically specified controller. At the same time, its performance on various tasks is evaluated. The best performing robots then reproduces, creating a process "death" and 'birth" cycle [Nolfi and Floreano, 2000]. Algorithms such as MONEE allow the robots to learn adapt to their dynamic environment, as well as to drive task-driven adaptation. In such approaches, the robots become increasingly proficient at their assigned tasks [Noskov et al., 2013].

Early example of Evolutionary Computing applications in robotics include the work of [Nordin and Banzhaf, 1997] on Genetic Programming for real-time robot control. The idea behind the developed method is to use evolutionary techniques to generate the actual code for the robot controller. The system evolves over time driven by feedback from the environment. Later development is the work of [Watson et al., 1999], where the authors describe a new methodology for evolutionary robotics called Embodied Evolution. It makes use of a population of robots, which reproduce autonomously, while executing predefined tasks. The described methodology is a form of a fully distributed evolutionary algorithm and it was implemented on physical robots [Watson et al., 2002]. Similar work by [Floreano and Mondada, 1998] describe a methodology for evolving an Artificial Neural Network-based controller for mobile robots.

2.1 Parameter Control

Parameter optimization for evolutionary algorithms is done either through parameter tuning or through parameter control. Parameter tuning means that parameter values are static and set beforehand, based on a –possibly extensive– set of preliminary experiments [Eiben and Smit, 2011]. Parameter Control is a process where parameter values are set during the course of the evolutionary process: the algorithm's settings are modified during the run [Eiben et al., 2007].

In on-line evolutionary robotics, researchers cannot always pre-determine appropriate parameter values for the algorithm because the circumstances under which the algorithm must perform may be incompletely known or changing. Therefore, parameter control seems a suitable technique.

Eiben et al. [2007] identify three approaches to Parameter Control:

- **Deterministic parameter control** This technique involves using a predetermined set of static rules to modify the parameter. The actual performance of the algorithm is not taken into account.
- **Adaptive parameter control** The changes to the parameters are based on feedback delivered by the running Evolutionary Algorithm. It is important to note that this technique involves using a heuristic mechanism to adapt the parameter values based on the performance of the Evolutionary Algorithm.
- **Self-adaptive parameter control** This approach eliminates the need of an external heuristics for the parameter selection. Instead, the parameters themselves are part of the genotype of the Evolutionary Algorithm.

There is a multitude of techniques for parameter tuning of evolutionary algorithms, while on-the-fly parameter control is a relatively sparse field. Most of the existing parameter control techniques focus on common evolutionary algorithm parameters such as crossover rate, mutation rate and mutations step-size [Eiben and Smit, 2011]. Davis' adaptive operator, for instance, modifies the crossover rate by rewarding operations that create better offspring [Davis, 1989]. There are some computational intelligence systems that tackle dynamic world problems, such as fuzzy systems, artificial neural networks, evolutionary computation (the latter two are also used in the design proposed in this paper), yet they all face challenges in complex evolving processes. These include: difficulty in preselecting the systems architecture, catastrophic forgetting, excessive training time required and lack of knowledge representation facilities [Kasabov, 2007]. A more practical solution is to rely on *online learning*, where learning and prediction processes alternate. This is contrary to an *off-line* system, where the learning processes on sample training data precedes the evaluation [Kasabov, 2007][Angelov et al., 2010].

Karafotias et al. [2012] propose a generic parameter control process that integrates both on-line parameter adjustments and off-line tuning. This scheme aims at repetitive applications and can be applied to any numeric parameter. It has been tailored to specific classes of problems through an off-line calibration process. Because evaluation time is such a specific parameter for on-line evolution with a time-sharing approach, it is uncertain that this technique is suitable. Haasdijk et al. [2011] introduced a racing scheme that goes some way towards a control method for evaluation time. Racing aborts the evaluation of apparently poor genomes by estimating their potential survival probability. While racing triggers an early abort of poor individuals and accelerates convergence, it does not remove the need to pre-select a value for the standard evaluation time (τ_{max}). Haasdijk et al. [2012] showed that even with the racing scheme, algorithm performance is very sensitive to τ_{max} settings.

The Multi-armed bandit problem, first introduced by Robbins [1952], can be seen as a form of the the exploration versus exploitation dilemma - trying to find optimal balance between exploring the environment and taking the so-far empirically best action [Auer et al., 2002]. The multi-armed bandit problem is possibly the most generic setting in which this trade-off can be modelled [Berry and Fristedt, 1985]. Similar to the problem a gambler faces when playing on slot machines ("one-armed bandits") when deciding which machines to play, depending on the choices made, one receives a stochastic reward [Gittins, 1989]. The goal is to maximize the sum, or some other function, of the series of rewards. One of the control schemes we propose in this paper (the roulette-wheel selection scheme) is modelled after the Multi-armed bandit problem.

3 Algorithms

The evaluation time selection schemes we propose in this paper are generic: they are not particular to any online evolutionary algorithm. Our experiments employ the (μ+1) ON-LINE algorithm. This algorithm employs standard evolutionary operators on a population of size μ to develop new individuals. That new individual, the challenger, is then evaluated by letting it take the control of the robot for a period of (τ_{max}) time steps, measuring the robot's task performance over that particular period. If the challenger proves to perform better than

the current worst of the population, the challenger replaces the current worst. As described above, the evaluation may be aborted ahead of time through a racing procedure. The (μ+1) ON-LINE algorithm re-evaluates genomes in the population to combat noisy fitness evaluations and changing circumstances with a probability ρ: it chooses to either re-evaluate an already existing genome or it generates a new genome by using variation mechanisms and evaluates that [Haasdijk et al., 2012].

In this paper we extend and analyse in detail the Roulette-wheel and H-Rule τ_{max} control schemes Arif et al. [2013] propose.

```
for j ← 1 to r do
    //initialize τ intervals
    τ_interval[j] ← 1;
end
for ever do
    τ_interval[current] ←
    RouletteWheelSelection(τ_interval[r]);
    τ_max ← random(τ_interval[current]);
    //Do (re-)evaluation with current τ_max
    Fitness.current ← RunAndEvaluate(τ_max);
    // performing heuristics
    if Fitness.current ≥ Fitness.worst and
    Fitness.current > Fitness.previous then
        τ_interval[current] ← τ_interval[current]+1;
    else
        τ_interval[current] ← τ_interval[current]-1;
    end
end
```

Algorithm 1: Pseudocode for Roulette-wheel τ_{max} selection

3.1 Roulette-wheel τ_{max} selection scheme

This evaluation time selection scheme is inspired by the Roulette-wheel selection mechanism that selects parent individuals proportionate to their performance. Roulette-wheel selection assigns sections of a virtual wheel to each individual and the area of these sections reflects their performance. Better individuals obtain a greater area and thus have a higher chance of selection. However, there is always some probability to select any individual: none of the portions has the probability equal to zero.

We take 50 and 1000 as theoretical limits for τ_{max}, where value 1000 is considered to be a large enough maximum evaluation time for the considered tasks. We divide this limit into r equal-sized τ_{max} intervals, where r determines the granularity of the intervals. For instance $r = 4$ implies four equal-sized intervals. This τ_{max}-selection scheme initializes the weights of all intervals with a constant value 1 so that each interval has an equal probability of being selected. To set a τ_{max} value at the outset of an evaluation, standard roulette wheel selection is used: the likelihood of an interval being selected is proportionate to its weight, i.e. an interval with a higher weight has higher probability of being selected. When an interval has been selected, a value is drawn from a uniform random distribution across the interval. Upon the end of the evaluation, the fitness of that particular interval is updated by a simple rule. If the resulting performance turns out to be better than or equal to the worst in the population and better than the previously reported fitness, the weight of the selected interval is increased by a value of 1. Otherwise it is decreased by a value of -1. The weight of an interval cannot decrease below 1, and consequently, at any point of evolutionary process, no interval is completely disregarded. Unlike racing in (μ+1) ON-LINE algorithm, which is applied only when a new genome is generated and evaluated, this control scheme is applied also when re-evaluating a genome in the population. The pseudocode of this technique is provided in Algorithm 1.

3.2 H-Rule for τ_{max} selection

The H-Rule (Heuristic-Rule) scheme relies on monitoring the actual fitness of the genomes after (re-)evaluation. It considers three consecutive (re-)evaluations: if *at least one* evaluation turned out to be better than or equal to the best in the population, then it is a fair indication that the currently considered τ_{max} should remain unchanged. If that was not the case then the current τ_{max} value is changed by randomly adding or subtracting a value within the range of 50. At any point of evolution, the actual τ_{max} does not go beyond its logical lower and upper bounds i.e. 50 and 1000 respectively. The pseudocode of this technique is provided in Algorithm 2.

Since the environment and tasks are dynamic, they could change at any moment, different evaluation time might be needed. As such, the H-Rule scheme tries selecting a best overall (for the complete run) τ_{max} value. The probability for a particular τ is increased only when the measured fitness is better than the worst and the previous. This interval's performance is also compared against the previous best to avoid getting stuck in a local maximum. In addition to (more or less) elitist approach, the performance of this interval is compared against the worst, to avoid increasing its probability even though it's performing worse than the worst.

```
//initialize τ intervals
τ_max ← random(τ);
for ever do
    improvement ← false;
    for i ← 1 to 3 do
        //Do (re-)evaluation with current τ_max
        Fitness.current ← RunAndEvaluate(τ_max);
        if Fitness.current ≥ Fitness.best then
            improvement ← true;
        end
    end
    if !improvement then
        τ_max ← τ_max ± random (50);
    end
    improvement ← false;
end
```

Algorithm 2: Pseudocode for H-Rule τ_{max} selection

4 Experimental setup

In order to test the validity and efficiency of the proposed schemes, different experiments were carried out in a simulated robot environment using Roborobo software [Bredeche et al., 2013]. While no physical robots are used, the target of this research is on-board evolution. For this purpose the used software package provides support for independent evolution control over each of the robotic agents. Thus it is effectively simulating an on-board controller, albeit without the limited hardware resources associated with the use of a physical embedded system.

We use the (μ+1) ON-LINE algorithm as described in [Haasdijk et al., 2011] with slight modifications (the details of the experiments are listed in Table 1) as a test bed for the τ_{max} control schemes. Such schemes are integrated in the algorithm for three completely different tasks and environments. For experimental purposes, we considered three tasks: Fast forward, Phototaxis, and Predator-Prey.

The tasks are switched on and off multiple times to simulate a dynamic environment in which the task objective and arena complexities are changed on-the-fly. Robots need to adapt to the tasks on-line. Keeping in view the complexity of the Predator-Prey task, we let it run for the first 300,000 time steps followed by Fast forward and Phototaxis tasks that run for 100,000 time steps each. Subsequently, we switched back to Predator-Prey task again and let it go on for the next 300,000 time steps. However, we have considered two different scenarios when we switch the Predator-Prey task for the second time. The *first* scenario (scenario 1) was that the Predator and Prey agents keep their same roles when task-switch happens. While in the *second* scenario (scenario 2), the roles of Predator and Prey are also switched along-with the task i.e. Predator becomes Prey and vice versa. After that, we changed to Phototaxis and Fast forward respectively for 100,000 time steps each (in reversed order). A single experiment, consisting of the sequence of these tasks, therefore runs for one million time steps. Each experiment was repeated 52 times, with different random seeds.

Table 1 Experiment settings

Experiment details	
Number of robot agents	2
Number of repeats	52
Simulation Time	
Total simulation time	1 million time steps
Predator-Prey	600,000 time steps
Fast forward	200,000 time steps
Phototaxis	200,000 time steps
Evolution Details	
Representation	Real-valued vector with $-4 \leq x_i \leq 4$
Mutation	Gaussian (N, σ)
Mutation Step size	De-randomized self-adaptive
Parent Selection	Binary Tournament
Crossover Rate	0.0
Survivor Selection	Replaces worst in population if challenger is better
Algorithm Details	
μ	6
ρ	0.3
chromosomes length	34
Controller Details	
Neural Network type	Simple perceptron
Input nodes	8 obstacle sensors + 8 light sensors + bias;
Output nodes	2 (left and right motor values)

Since all the three tasks, considered in our experiments, were completely different in their objectives, we used specialized sensors and certain logic to tackle these differences. Next to 8 obstacle sensors, commonly used in Fast forward experiments [Arif et al., 2013], there are also 8 *multi-purpose* sensors that could sense both light and the absence or presence of another robot in their range. These additional sensors could detect light and consequently determine the distance from the light source. Moreover, the same sensors, in predator-prey scenario, could detect the presence of another robot in its range and calculate the distance. The sensor ranges for all 16 sensors were the same. Signals from each sensor are sent directly to the actuators and the weights of the simple perceptron neural network controller are adjusted by the evolutionary algorithm. The main reason we have opted to use a perceptron neural network is simplicity, as more complex controller will require a more complex learning function.

Task switching involve changes in fitness function, due to the different objective for each task, as well as neural network controller changes, since different sen-

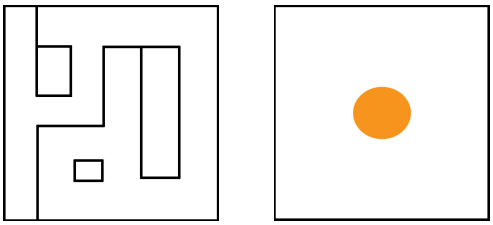

Fig. 1 Arenas used for experiments: Fast forward arena(left), phototaxis arena(right)

sors with different interpretations are used. In addition the arena used in the Fast forward task is very different from the one used in the other two. As a result the tasks should be starkly different from one other and this helps reinforcing the original goal, that agent should be able to perform well, by adapting to wide selection of environments and objectives. Our expectation is that any adaptation from, for example, the Fast forward task will not help an agent executing the Phototaxis one, as both use very different sensors, fitness functions and arenas.

4.1 Fast forward task

In this common task, the objective of the robot is to move as fast as it can, while avoiding obstacles in an arena. In an obstacle-restrained environment the task implies a trade-off between avoiding obstacles and maintaining speed and forward movement. The fitness function is listed in Equation 1.

$$f = \sum_{i=1}^{\tau_{max}} (v_t.(1 - v_r)) \tag{1}$$

where v_t and v_r are the translational and rotational velocities, respectively. v_t is normalized between -1 (full speed reverse) and 1 (full speed forward). And v_r is normalized between 0 (movement in a straight line) and 1 (maximum rotation). In our simulations, whenever a robot touches an obstacle, v_t is set to 0, so the fitness increment for the time while the it is in collision is 0. A good controller will turn only when necessary to avoid collisions and try to find paths that allow it to run in a straight line for as long as possible. To add sufficient complexity to this task, we have selected an arena with dead ends and narrow corridors. For each run of experiments the robot's initial position is set to a random location in arena. While performing this task, the 8 additional light sensors are fed with a constant low value of 0.001 and their signals are still sent directly to the actuators

4.2 Phototaxis task

The objective of robots performing Phototaxis task is mainly to stay as close as possible to a stationary light source. The inputs to the artificial neural network are a set of 8 light (multi-purpose) sensors. The arena is an empty square with a light source in the middle. The values of the 8 obstacle sensors, used in the Fast forward task, were set to a constant low value of 0.001 and their signals still sent directly to the actuators. The fitness function for this task is defined in Equation 2.

$$f = \sum_{i=1}^{\tau_{max}} maxSensorValue; \tag{2}$$

4.3 Predator-Prey task

Predator-Prey is a task in which robots are assigned specific roles and based on their roles they have different objectives. A robot assigned a role of predator aims at following and chasing a robot labeled as prey. The prey robot, on the other hand, needs to stay away from the predator. It is evident from this explanation that their objectives are conflicting since good performance for one is bad for the other. The competitive nature of this task is evident when we visualize its evolutionary development over time (see figure 2). Since the fitness for both is measured independently, the graph should be an indication that the task remains competitive and eventually the performance of both agents levels off. Initially the prey has an advantage initially since good performance is measured when there is no predator in range. Over time the predator evolves and learns how to find and follow the prey. The gradual slope is an indication of learning.

Equipped with 8 multi-purpose sensors, both predator and prey agents could sense each other's presence using these sensors. The other 8 obstacle-avoidance sensors were again supplied with an invariable low value of 0.001 and their signals still sent directly to the actuators. The fitness functions for both agents are listed in Equation 3 and Equation 4.

$$f_{predator} = \sum_{i=1}^{\tau_{max}} maxSensorValue; \tag{3}$$

$$f_{prey} = \sum_{i=1}^{\tau_{max}} minSensorValue; \tag{4}$$

The prey robot has an inherent advantage in our experimental setup as its performance is maximum if the predator is not in range. The predator needs to learn to stay close to the prey agent since its fitness becomes zero as soon as the predator is out of its sensor range. The arena used in this task is an empty square.

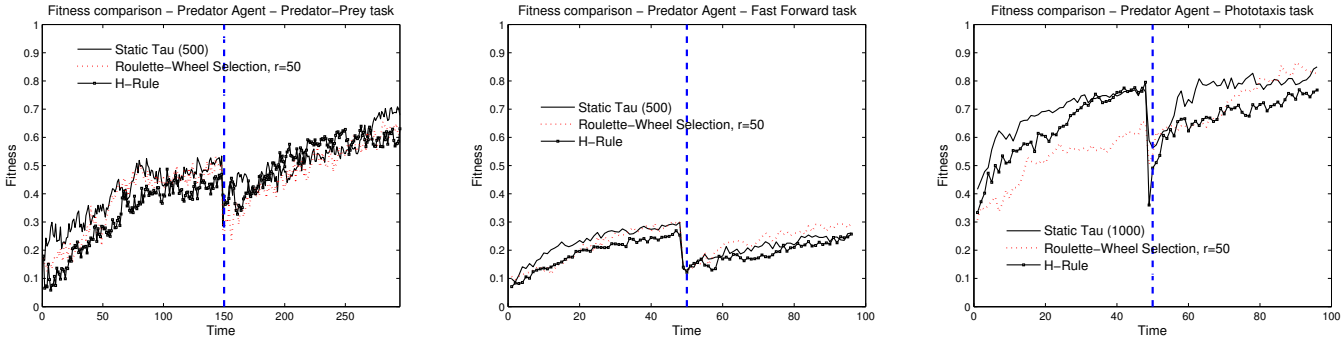

Fig. 3 Scenario 1 (Predator agent only): Fitness Comparison plots for the best configurations of static τ_{max}, Roulette-wheel selection and H-Rule; *[Predator Prey (left), Fast forward (middle), and Phototaxis (right)]*. Each data-point on x-axis represents 2000 time steps.

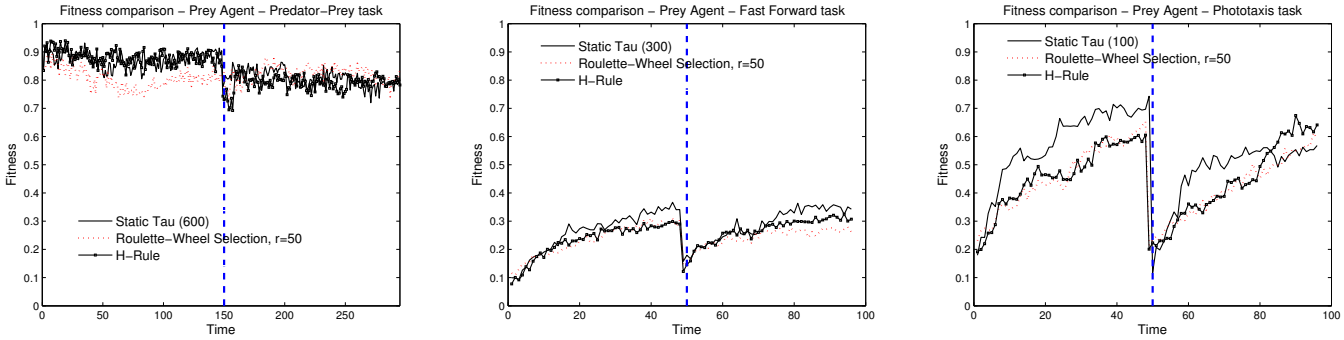

Fig. 4 Scenario 1 (Prey agent only): Fitness Comparison plots for the best configurations of static τ_{max}, Roulette-wheel selection and H-Rule;*[Predator Prey (left), Fast Forward (middle), and Phototaxis (right)]*. Each data-point on x-axis represents 2000 time steps.

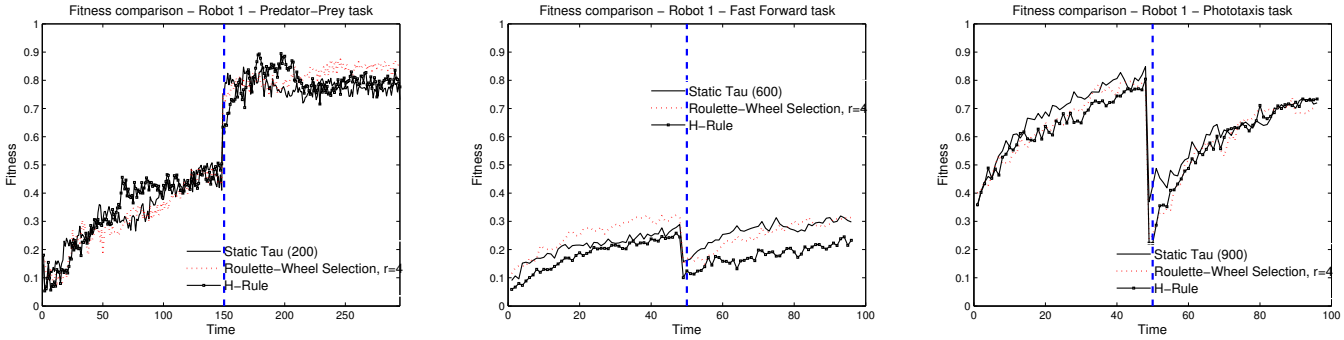

Fig. 5 Scenario 2 (Robot 1 only): Fitness Comparison plots for Roulette-wheel selection, H-Rule, and the best configurations of static τ_{max} for *[Predator Prey (left), Fast Forward (middle), and Phototaxis (right)] subsets*. Each data-point on x-axis represents 2000 time steps.

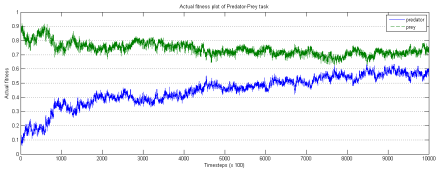

Fig. 2 Evolutionary development of predator-prey task using (μ+1) ON-LINE algorithm. The plot shows average actual performance (the higher the better), represented on y-axis, of 52 runs over time, represented on x-axis.

5 Results and discussion

On figures 3, 4, 5 and 6 we have plotted the comparison of the change of fitness (the higher the better) over time for all scenarios and tasks. To have a cleaner visual comparison we have only included the best performing schemes - in each comparison we have picked the best *static* τ_{max} scheme, for a specific task and scenario, and plotted it against the two *dynamic* schemes with best *overall* performance. In this regard, static schemes

were hand-picked with τ_{max} ensuring maximum fitness for each task separately, rather than such suited for best overall performance for all tasks throughout evolution. This comparison serves to demonstrate if dynamic schemes are applicable in wider number of tasks and scenarios.

During the whole evolutionary cycle we toggle each task *two* times at different moments and in different orders. These transitions of the tasks are depicted by drawing a blue dashed line in the middle of the plots (see figures 3, 4, 5 and 6). Left and right sides of this line show the robots' actual average performance for certain tasks during their first and second period respectively. Similarly, horizontal axes represent the time steps during the first and second period of the same tasks in that order. Moreover, in our experiments, we let two robots perform different tasks over the time. Specifically during Predator-Prey task, both robots had different (competitive) tasks to perform. Therefore, graphs for both Predator-agent and Prey-agent have been presented separately as aggregated performance plots were not possible.

As seen in most cases the results are quite similar for the measured fitness through the runs of the best static and the selected dynamic schemes. In some occasions the pre-selected static schemes have measurable better performance than both dynamic schemes. For the Predator-Prey task in all variations, the results are too close to point out any clear winner. This holds for both the setup when the robot is only a predator or prey, and when the role changes after the tasks switch. For Fast forward, in all selected Scenario 2 comparisons, static evaluation time scheme performed better (Figures 5 middle and 6 middle). For the Phototaxis the results are mostly similar, with only in Scenario 2, Robot 2 comparisons (Figure 6 right) we can observe that selected static evaluation time has better performance than the dynamic schemes.

Our results showed that Roulette-wheel selection scheme had comparable performance to the best static ones in many cases. However, we see a great decrease in its performance even in (simple) Phototaxis task in both scenarios. Similarly, in scenario 2, when the roles were switched for predator and prey, this scheme did not show any clear performance improvement (leftmost picture in figure 6). Moreover, this scheme introduced a new parameter r, denoting the number of intervals to use for selection. A one-way analysis of variance between *r=4* and *r=50*, for example in scenario 1 for predator robot, illustrated that both settings of r could significantly affect the performance of the algorithm ($F(1,19998) = 60.41$, p-value $= 8.1\times10^{-15}$) and more investigation is required to find an optimum value of r. For both scenarios of experiments, we considered *r=4,12,20,50* for Roulette-wheel selection. *r=50* and *r=4* turned out to be best in the first and second scenarios respectively. The evaluation of all r paper was done "off-screen". It was not included in the paper since it would only provide a distraction. In addition one-way analysis of variance between the r values showed that they could be quite influential. As such, the scheme is not as stable as we hoped. In the end we put more influence on H-rule due to the lack of such parameters.

Besides the influence of the parameter r on the performance of the algorithm in both scenarios, we also see an odd behavior of the actual development of τ_{max} over time. Our analysis shows that, even with totally different r values, the actual τ_{max} stays within a certain range always and we do not see any visible trend in the change of actual τ_{max} values (see figure 7). At the same time we also cannot imply that the scheme is selecting τ_{max} completely randomly since the actual τ_{max} values are mostly in the range of 600 and 800 and not all over the total τ_{max} considered domain. This unusual behavior of τ_{max} selection and the sensitivity of the newly introduced parameter do not give a promising evidence to consider this scheme for highly dynamic and uncertain scenarios.

It is evident from the performance plots that, in nearly all tasks, the best values of τ_{max} were found to be different (see figures 3, 4, 5 and 6). That re-confirms the importance of proper selection of τ_{max} in different tasks and scenarios. Obviously, it is hard to find any constant value which could serve as a global optimum value for all the tasks throughout the evolutionary process; especially when differences between the best settings of τ_{max} for several tasks are reasonably big.

Figures 8 and 9 present the average mean performance boxplots of the corresponding best settings of τ_{max} (as depicted in figures 5 and 6 for individual subsets tasks) and H-Rule. This comparison is performed for the *whole evolutionary run* and not only for certain tasks' subsets. Keeping in view the inconsistency and sensitivity of Roulette-wheel τ_{max} selection scheme (as mentioned earlier in our discussion), we did not include Roulette-wheel τ_{max} selection in this comparison. Instead, we considered random τ_{max} selection in our comparison to show if our H-Rule scheme is better than random τ_{max} selection. It is clear from the plots that although in individual subsets of tasks, the best static settings for τ_{max} worked marginally better than both dynamic schemes (as visible in figures 5 and 6), but their overall average performance throughout the evolution with same settings of τ_{max} were found to be statistically less than or comparable to H-Rule (see figures 8,9). This observation is promising since it makes

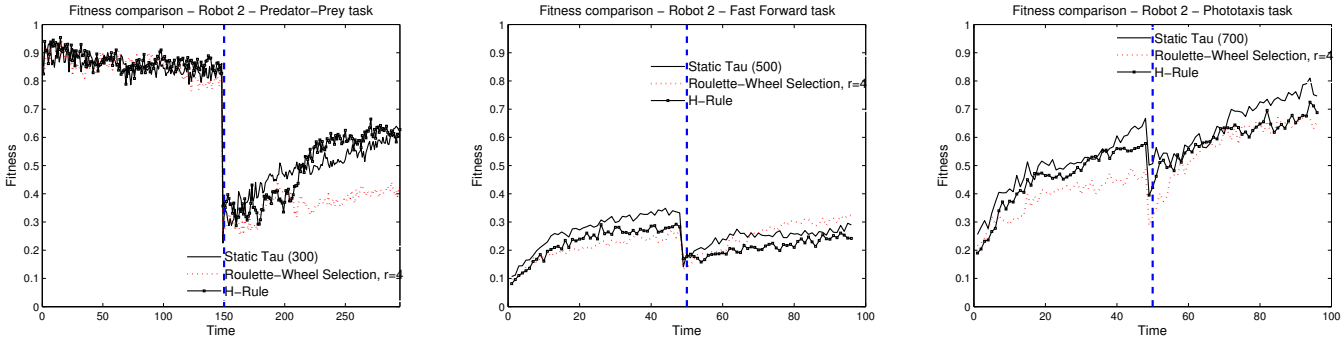

Fig. 6 Scenario 2 (Robot 2 only): Fitness Comparison plots for Roulette-wheel selection, H-Rule, and the best configurations of static τ_{max} for *[Predator Prey (left), Fast Forward (middle), and Phototaxis (right)] subsets*. Each data-point on x-axis represents 2000 time steps.

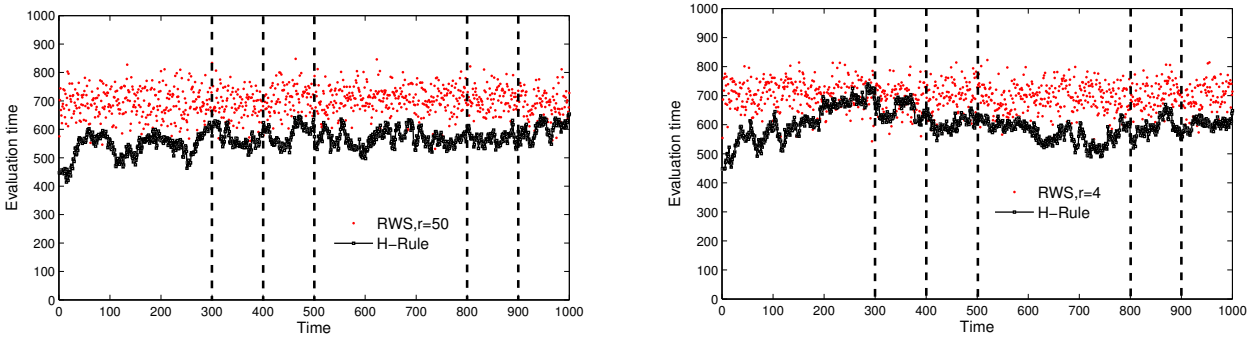

Fig. 7 τ_{max} development over time, Roulette-wheel selection vs. H-Rule; Dashed vertical lines indicate task switching. τ_{max} development over time for predator agent in scenario 1 (left), τ_{max} development over time for robot 2 in scenario 2 (right). Each data-point on x-axis represents 1000 time steps.

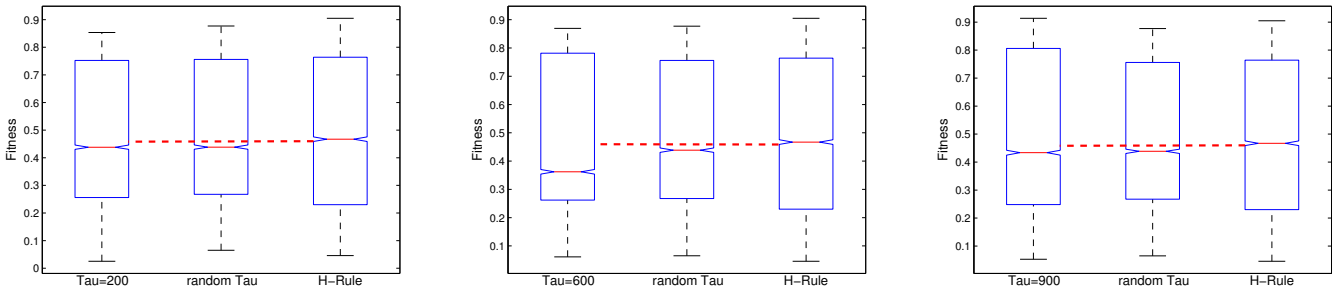

Fig. 8 Scenario 2 (Robot 1 only): Fitness Comparison boxplots for the respective τ_{max} values against random τ_{max} and H-Rule in Predator-Prey, Fast forward and Phototaxis respectively (for complete evolution). Y-axis represents fitness. F(2,29997)=20.53, p-value=1.23×10^{-009} ***(left)***, F(2,29997)=20.92, p-value=8.31×10^{-10} ***(middle)***, and F(2,29997)=18.21, p-value=1.25×10^{-008} ***(right)***

us to believe that H-Rule could serve as a viable and robust scheme to control the evaluation time in highly dynamic and uncertain scenarios. Partially in different phases or tasks during the complete evolutionary cycle, its performance could be less or comparable to one of the best static ones but, in general, it maintains a reasonably good and stable performance. The lower extreme notches of H-Rule boxplots seem to always stand above the upper notches of different static τ_{max} and random τ_{max} settings. Therefore, we can say by 95% certainty that the median performance of H-Rule is not only better than random τ_{max} selection but also from all the considered best static settings for τ_{max} for specific tasks. The plots also list the ANOVA statistics respectively for different variants showing that differences between the obtained results were statistically significant. We have presented the comparison box-plots of

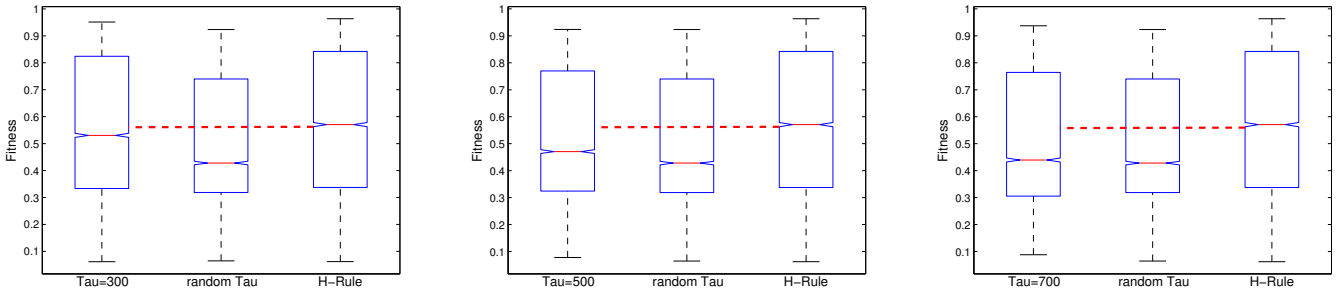

Fig. 9 Scenario 2 (Robot 2 only): Fitness Comparison boxplots for the respective τ_{max} values against random τ_{max} and H-Rule in Predator-Prey, Fast forward and Phototaxis respectively (for complete evolution). Y-axis represents fitness. F(2,29997)=194.76, p-value=0.000***(left)***, F(2,29997)=178.53, p-value=0.000***(middle)***, and F(2,29997)=163.84, p-value=0.000***(right)***

scenario 2 only because this scenario is more difficult and dynamic in nature.

It is worth noting that H-Rule relies on evaluating last *three* consecutive performance evaluations and this might be considered as some parameter. Evaluating last three evaluations seems a reasonable number since relying on too many evaluations may well result in slower convergence. Nevertheless, a better exploration about the number of evaluations to be considered during H-Rule could be investigated further. Our current analysis does not cover this aspect and keeping in view different tasks and scenarios we examined, and number of repeats we performed for experiments, it is safe to assume that H-Rule is more generic and has the ability to give more stable performance in highly dynamic and uncertain scenarios.

Figure 7 shows the distribution of τ_{max} for Roulette-wheel selection and H-Rule. In Roulette-wheel selection scheme, it is evident that most of the actual τ_{max} selections are done from a specific range and this behavior does not change even upon multiple task switch occurrences. For the sake of clarity in the plots, we plotted dotted graph for the τ_{max} selections. The peaks in changes in selected τ_{max} for H-Rule are generally more smooth and relatively more visible changes in τ_{max} over time are observable. Nevertheless, in both dynamic schemes, a vigilant and rapid change in the actual τ_{max} upon task switch for multiple times is not noticeable, which was initially expected. The optimum τ_{max} value is different for each task, as tested with the *static* values are runs. While it is possible that the selected *dynamic* τ_{max} values (plotted on figure 7) to converge to the best previously evaluated τ_{max}, it is not guaranteed. In addition the probabilities for τ_{max} selected also have an effect on the next task. Our assumption is that the algorithm does not try to select either very smaller or larger evaluation times at task switches. It rather relies on the so-far best performing value, instead of swiftly exploring new ones. This behavior could be an obvious outcome of incorporating *racing*, since it generally does not favor higher values of evaluation time.

As shown on the comparison figures, a different static τ_{max} value gives best overall fitness for each of the task. At the same time, the same H-Rule dynamic scheme shows comparable performance to the static one in each task. The advantage of H-Rule in this case is that it allows the agent to adapt to the objectives and environment changes without the need of additional tuning or outside assistance, unlike when a static scheme is used.

6 Conclusion and Future Research

In this paper we set as a goal to demonstrate that presented heuristic techniques are viable control schemes, better than pre-selecting influential parameters for online on-board evolutionary robotics. Both evaluation time selection schemes presented in this paper were evaluated against multiple tasks with diverse objectives and required sensors. Our analysis demonstrates that, in general, the H-Rule managed to give stable performance even during task switching and had performance comparable to that of the best static evaluation time for a particular task. This heuristic scheme also allowed for a reasonable fitness recovery upon task switching. Not only it showed comparable performance during particular tasks but it also maintained a reasonably good overall performance during the whole evolutionary run. Moreover, H-Rule in particular has demonstrated being better than a random evaluation time selection.

At the same time Roulette-wheel selection performed less consistently during task switching and did introduce a new parameter. Using H-Rule instead of Roulette-wheel selection also avoids the use of parameter r, de-

noting the number of intervals that divide the evaluation time value space. This indicated that the new higher-level heuristic can be used as true dynamic evaluation time selection scheme and does not introduce new parameter(s). These results also indicate that dynamic schemes, in general, can be applied to a wider selection of task with less effort. Unlike static schemes, which are best suited to single static task experiments, the proposed H-Rule scheme does not require tuning at task switching, as it allows the agent to adapt to the new objective. That reduced the need of outside interaction and would give an advantage to robots operating independent from humans, while using a dynamic scheme.

Another observed behavior was that both dynamic schemes, though performed reasonably well, did not manage to *visibly* control τ_{max}, and contrary to expected, they did not opt for exploring reasonably higher or lower τ_{max} values after task switching. This behavior was more obviously evident in Roulette-wheel τ_{max} selection, which continued to select τ_{max} values mostly from a specific domain only. Both schemes frequently relied on exploiting the values or domains for τ_{max} that gave fair performance and did not swiftly explore new ones after task switching. Our theory is that the generic dynamic τ_{max} selection scheme clashes with the use together with *Racing*, as both try to control or operate on the same parameter. In other words any higher values for evaluation time are cut short when Racing aborts the run and we are more likely to get a wrong indication of was it the selected τ_{max} value that was not good or the genome. This indication may mislead our search process for a proper τ_{max} and it seems very hard to achieve good results with the simple heuristics used. Keeping in view the profound effect on accelerating the convergence rate of algorithm, switching Racing off and only relying on the currently examined τ_{max} control schemes might not be a good idea either. A detailed study on their collaborated effort might open a new vista of robust and reliable parameter control mechanisms for on-line and on-board evolution.

Despite this observed behavior for the examined dynamic τ_{max} selection schemes, the results are promising and one of the presented schemes, i.e. H-Rule, was found to perform consistently across multiple scenarios and tasks. The robustness of this scheme make it a viable alternative to setting τ_{max} separately for each task and scenario. However, further tests are required to verify the theory whether using two evaluation time control schemes are causing some of the observed issues or not. Future research involving H-Rule should also evaluate if dynamic schemes have advantage over static schemes, especially when it comes to even more advanced experiments. The assumption this further research will try to prove is that dynamic τ_{max} schemes allow an agent to adapt to a wider range of task. For example, a very different class of tasks are such involving Physical Robotics [Pollack et al., 2000]. One physical task could be Control of Walking Robots, similar to the experiment presented in [Mitobe et al., 2000]. A Physical Robotics task would be a significantly more complex than the three executed in the context of this paper, as it requires the development of a more advanced controller with numerous additional sensors and actuators. At the same time another software suite will most likely be used for the task, working further to test the application of the H-Rule dynamic scheme in a wider range of tasks. A step further can be poring the H-Rule scheme to a physical robot, verifying the advantages of the dynamic τ_{max} control scheme in a real embedded system.

Alternatively more complex scenarios could be used for the experiments with a higher number of tasks executed in various order. Such experiments can test if the order of task execution influences the average performance.

Another potential point of further research is working on studying alternative and relatively complex heuristic techniques that have robust performance and can be generalized to work with a multitude of scenarios. Moreover, further development of the technique would be to use alternative algorithms – replacing (μ+1) ON-LINE as evolutionary algorithm. One area of interest could be to use a distributed algorithm that allows to parallelize the process. This would mean increasing the number of robot agents in the arena and accounting for the interaction between them.

References

Plamen Angelov, Dimitar P. Filev, and Nik Kasabov. *Evolving Intelligent Systems: Methodology and Applications*. Wiley-IEEE Press, 2010. ISBN 0470287195, 9780470287194.

Atta-ul-Qayyum Arif, Dimitar Nedev, and Evert Haasdijk. Controlling evaluation duration in on-line and on-board evolutionary robotics. In *Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence*. IEEE Press, 2013.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Donald A. Berry and Bert Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Springer, October 1985. ISBN 0412248107.

Nicolas Bredeche, Jean-Marc Montanier, Berend Weel, and Evert Haasdijk. Roborobo! a fast robot simulator for swarm and collective robotics. *CoRR*, abs/1304.2888, 2013.

L. Davis. Adapting operator probabilities in genetic algorithms. In *Third International Conference on Genetic Algorithms,*, pages 61–69, 1989.

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

A. E. Eiben, Zbigniew Michalewicz, Marc Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. In Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer, 2007. ISBN 978-3-540-69431-1.

A.E. Eiben, Evert Haasdijk, and Nicolas Bredeche. Embodied, on-line, on-board evolution for autonomous robotics. In P. Levi and S. Kernbach, editors, *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, pages 361–382. Springer Verlag, 2010.

D. Floreano and F. Mondada. Evolutionary neurocontrollers for autonomous mobile robots. *Neural Networks*, 11(78):1461 – 1478, 1998. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/S0893-6080(98)00082-3.

J. C. Gittins. *Multi-armed Bandit Allocation Indices*. Wiley-Interscience series in systems and optimization. Wiley, Chichester, NY, 1989.

Evert Haasdijk, Atta-ul-Qayyum Arif, and A.E. Eiben. Racing to improve on-line, on-board evolutionary robotics. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2011)*. ACM, 2011.

Evert Haasdijk, Selmar K. Smit, and A. E. Eiben. Exploratory analysis of an on-line evolutionary algorithm in simulated robots. *Evolutionary Intelligence*, 5(4):213–230, 2012.

William E. Hart and Richard K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 190–195. Morgan Kaufmann, 1991.

Giorgos Karafotias, Selmar K. Smit, and A. E. Eiben. A generic approach to parameter control. In Cecilia Di Chio, Alexandros Agapitos, Stefano Cagnoni, Carlos Cotta, Francisco Fernández de Vega, Gianni A. Di Caro, Rolf Drechsler, Anikó Ekárt, Anna Isabel Esparcia-Alcázar, Muddassar Farooq, William B. Langdon, Juan J. Merelo Guervós, Mike Preuss, Hendrik Richter, Sara Silva, Anabela Simões, Giovanni Squillero, Ernesto Tarantino, Andrea Tettamanzi, Julian Togelius, Neil Urquhart, Sima Uyar, and Georgios N. Yannakakis, editors, *EvoApplications*, volume 7248 of *Lecture Notes in Computer Science*, pages 366–375. Springer, 2012. ISBN 978-3-642-29177-7.

N Kasabov. Evolving connectionist systems. Springer London, London, 2007.

Kazuhisa Mitobe, Genci Capi, and Yasuo Nasu. Control of walking robots based on manipulation of the zero moment point. *Robotica*, 18(6):651–657, 2000.

Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology,Intelligence,and Technology*. MIT Press, Cambridge, MA, USA, 2000. ISBN 0262140705.

Peter Nordin and Wolfgang Banzhaf. Real time control of a khepera robot using genetic programming. *Cybernetics and Control*, 26(3):533–561, 1997.

Nikita Noskov, Evert Haasdijk, Berend Weel, and A. E. Eiben. Monee: Using parental investment to combine open-ended and task-driven evolution. In Anna Isabel Esparcia-Alcázar, editor, *EvoApplications*, volume 7835 of *Lecture Notes in Computer Science*, pages 569–578. Springer, 2013. ISBN 978-3-642-37191-2.

Jordan B. Pollack, Hod Lipson, Sevan Ficici, Pablo Funes, Greg Hornby, and Richard A. Watson. Evolutionary techniques in physical robotics. In *Proceedings of the Third International Conference (ICES2000*, pages 175–186. Springer-Verlag, 2000.

Herbert Robbins. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.*, 58(5):527–535, 1952.

M Sayed-Mouchaweh and E Lughofer. Learning in non-stationary environments. Springer New York, New York, NY, 2012.

Richard A. Watson, Sevan G. Ficici, and Jordan B. Pollack. Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In *CONGRESS ON EVOLUTIONARY COMPUTATION*, pages 335–342. IEEE Press, 1999.

Richard A. Watson, Sevan G. Ficici, and Jordan B. Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1 – 18, 2002. ISSN 0921-8890. doi: http://dx.doi.org/10.1016/S0921-8890(02)00170-7.