On-line Neuroevolution Applied to The Open Racing Car Simulator

Luigi Cardamone, Daniele Loiacono and Pier Luca Lanzi

Abstract— **The application of on-line learning techniques to modern computer games is a promising research direction. In fact, they can be used to improve the game experience and to achieve a true adaptive game AI. So far, several works proved that neuroevolution techniques can be successfully applied to modern computer games but they are usually restricted to off-line learning scenarios. In on-line learning problems the main challenge is to find a good trade-off between the exploration, i.e., the search for better solutions, and the exploitation of the best solution discovered so far. In this paper we propose an on-line neuroevolution approach to evolve non-player characters in The Open Car Racing Simulator (TORCS), a state-of-the-art open source car racing simulator. We tested our approach on two on-line learning problems: (i) on-line evolution of a fast controller from scratch and (ii) optimization of an existing controller for a new track. Our results show that on-line neuroevolution can effectively improve the performance achieved during the learning process.**

I. Introduction

Computer games are becoming more and more sophisticated and realistic. In particular, many of the recent computer games have a very complex dynamics and sophisticated physics engines. As a result, it is getting increasingly difficult to design non-player characters (NPC) that exhibit effective behaviors using scripted AI. Under this perspective,

computational intelligence is a promising but not yet really exploited technology to support the development of better NPC and to improve the game experience. In particular, evolutionary computation and, more specifically, neuroevolution are among the most successfully computational intelligence techniques applied to modern computer games in recent years [20], [7], [15]. However, the application of these techniques is generally restricted to off-line learning scenarios, i.e., the focus is on the final performance achieved rather than of the performance achieved *during* the learning process. This is a serious concern as many interesting applications in the field of computer games involve on-line learning scenarios [11]. Recently, Whiteson and Stone [19] introduced an on-line neuroevolution approach inspired to the action-selection strategy used by temporal difference learning to deal with the exploration/exploitation dilemma. Their approach was originally devised for solving stochastic reinforcement learning problems. In this work we show that it can be successfully applied also in modern computer games

Luigi Cardamone (cardamone@elet.polimi.it), Daniele Loiacono (loiacono@elet.polimi.it) and Pier Luca Lanzi (lanzi@elet.polimi.it) are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy.

Pier Luca Lanzi (lanzi@illigal.ge.uiuc.edu) is also member of the Illinois Genetic Algorithm Laboratory (IlliGAL), University of Illinois at Urbana Champaign, Urbana, IL 61801, USA.

to solve on-line learning problems. In addition, we apply on-

line neuroevolution also to solve an on-line learning problem, assuming we are given a solution for a slightly different problem. This type of problem, not considered in [19], is interesting in the computer games domain as it is related to the more general issue of developing an adaptive game AI. In this paper we used The Open Racing Car Simulator (TORCS) [1], a state of the art open source racing simulator, as testbed for our empirical analysis. Our results, even if preliminary, show that on-line neuroevolution applied to computer games is effective to improve the performance of the system during the learning process. However, further improvements are necessary to develop an adaptive AI that could be applied to a real-time learning problem in a modern computer game. Nevertheless, the results obtained are promising and suggest that this is an interesting research direction.

II. Related Work

The on-line neuroevolution approach used in this paper has been introduced by Whiteson and Stone in [19] to solve stochastic problems taken from the reinforcement learning literature. In this paper, instead of considering some reinforcement learning testbeds, we focused on on-line learning problems in the computer games domain. In addition, here we extended the on-line neuroevolution approach also to deterministic problems by replacing the usual fitness evaluation on the considered task with many faster but less accurate partial evaluations.

The approach used in this work is also related to Learning Classifier Systems (LCS) [5], as they are evolutionary systems that can be applied to on-line reinforcement learning problem. Moreover, also in the classifier systems the action-selection mechanism is applied to deal with the exploration/exploitation dilemma. However, LCS solve problems by learning a value function and, therefore, they apply an action-selection mechanism to choose the action to perform as usual done in TD learning. Instead, in on-line neuroevolution the problem is solved by learning the whole policy and the action-selection mechanism is used to select which candidate solution to evaluate during learning.

Then, as we consider the problem of combining many evaluations to compute the fitness, this work is related also to the techniques for applying EC in presence of a noisy fitness function [10], [2]. Anyway, the focus in these works is still solving optimization problems in an off-line scenario, while here the focus is on the on-line learning performance.

Concerning the application domain considered in this paper, a lot of recent works investigated the application of computational intelligence techniques to car racing games. In an early work, Pyeatt and Howe [8] applied reinforcement learning to learn racing behaviors in RARS, an open source racing simulator. More recently, evolutionary computation techniques have been applied to improve the performance of a motocross game AI [3], to optimize the parameters in a sophisticated F1 racing simulator [20] and to evolve a neural

network to predict crash in car racing simulator [12]. Finally, several works on learning controllers for racing games have been done by Togelius and Lucas [18], [16], [17], [15]. However, few of these works apply evolutionary computation to on-line learning problems and they do not focus on the performance of the system *during* learning. Our work is also closely related to one of Spronck [9] that focused on the issue of adaptivity in the game AI. In particular, he applied an elitist evolutionary computation approach to solve an on-line learning problem in a 3D shooter game. In his work he proposed a fitness propagation mechanism to deal with the stochasticity of the fitness instead of averaging the results of several evaluations as done in [19] and in this paper.

III. On-line Neuroevolution

In this section we provide a brief description of NeuroEvolution of Augmenting Topologies (NEAT) [13], a widely used neuroevolution approach. Then, we introduce the on-line evolutionary computation framework proposed by Whiteson and Stone [19] that we applied in combination with NEAT in this work. However, a comprehensive introduction to these topics is behind the scope of this paper, thus we refer the interested reader to [13], [19].

A. NEAT

In this work, we focus on one of the most successful and widely applied neuroevoluton approach: NEAT [13]. It is specifically designed to evolve neural networks without

assuming any previous knowledge neither on the optimal topology nor on the type of connections, i.e. using or not recurrent connections. To deal effectively with these problems, avoiding the search in an huge space, NEAT combines the usual evolutionary search with the complexification of the network structure. NEAT is based on three main ideas. First, in NEAT the evolutionary search starts from a network topology as simple as possible, i.e. a fully connected network with only the input and the output layers. Therefore, more complex structures emerge naturally during the evolutionary process and survive only when useful. Second, NEAT deal with the problem of recombining networks with different structures through an *historical marking* mechanism. Whenever a structural mutation occurs, a unique *innovation number* is assigned to the gene representing this innovation. This mechanism is then used to perform the recombination and to identify similarities between the networks without the need of a complex and expensive topological analysis. Third, NEAT protect the structural innovations through the mechanism of *speciation*. The competition to survive does not happen in the whole population but it is restricted to niches where all the networks have a similar topology. Therefore, when a new

Algorithm 1 Fitness in on-line evolutionary computation.

1: **procedure** COMPUTE-FITNESS(P) ▷ P is the population
2: $n(p) = 0, f(p) = 0 \quad \forall p \in P$ ▷ Initialization of

the fitness and evaluation counter
3: **repeat**
4: $p \leftarrow$ SELECT(P); ▷ Selection of the individual to be evaluated
5: $n(p) \leftarrow n(p) + 1$; ▷ Update the evaluation counter
6: $f(p) \leftarrow f(p) + \frac{1}{n(p)}(EVAL(p) - f(p))$; ▷ Update the fitness with current evaluation
7: **until** termination criteria is met
8: **end procedure**

topology emerges, NEAT has enough time to optimize it. Again, the historical marking is used to identify the niches by using the innovation numbers of the genes for measuring the similarities between different topologies. In order to prevent a single species from taking over the population, the networks in the same niche share their fitness [4]. Then, in the following generation more resources are allocated to the species with a greater fitness.

B. On-line Evolutionary Computation

When evolutionary computation is applied to stochastic problems, several evaluations are necessary to assess the fitness of an individual. In this case, a fixed number of evaluations are usually performed for each candidate solution in the population. However this approach is not suitable

for on-line learning problems where the focus is on the performance achieved *during* the learning and not on finding the optimal solution. In [19] Whiteson and Stone proposed to borrow from the temporal difference (TD) learning the action-selection mechanism and to exploit it for on-line evolutionary computation. While in TD the action-selection mechanism is used to select which action the system has to perform on the basis of the current knowledge, in on-line neuroevolution it is used to select which candidate solution will be evaluated. Algorithm 1 illustrates the general schema used in on-line evolutionary computation for computing the fitness of individuals in each generation. First, an individual is selected according to some policy (Algorithm 1, line 4). Then, the individual is evaluated and, on the basis of the outcome, its fitness is updated (Algorithm 1, line 5 and line 6). In this paper to update the fitness of the individuals the outcome of evaluations are just averaged but more sophisticated approach can be used. In this work we focus on two different action-selection mechanism widely used in the TD literature and, following what done in [19], we show how they can be used as evaluation strategies in on-line evolutionary computation.

ε-Greedy. The ε-greedy is probably the simplest action-selection mechanism used in the TD literature [14] and applying it to evolutionary computation is straightforward.

Algorithm 2 ε-Greedy selection strategy.

```
1: procedure SELECT(P)                    ▷ P is the population,
2:     if (rand() < ε) then
3:         return random p ∈ P;           ▷ Pick randomly an
               individual in the population
4:     else
5:         return argmax_{p∈P} f(p);      ▷ Return the fittest
               individual in the population
6:     end if
7: end procedure
```

At each iteration the ε-greedy strategy selects the individual to be evaluated as follows. With probability ε a random individual in the population is chosen, i.e., the strategy *explores* the population searching for the optimal solution. With probability $1-\varepsilon$ the fittest individual in the population is selected, i.e., the strategy *exploits* the best solution discovered so far. Algorithm 2 describes in detail the `SELECT` procedure that implements the ε-greedy evaluation strategy. This procedure can be easily plugged into the general fitness computation procedure described above and reported as Algorithm 1. Concerning the termination criteria, Whiteson and Stone in [19] set a constant number of evaluations to be performed in each generation. However, such a criteria leads to poor exploration/exploitation balancing capabilities [19]. Therefore, we propose a different termination criteria: the

loop in Algorithm 1 is performed until all the individuals in the population have been evaluated at least once and the fittest individual has been evaluated at least θ_{best} times. Even if the proposed criteria involves a new parameter θ_{best} to set, our results suggest that, with this modification, the ε-greedy strategy is much more competitive with respect to more sophisticated ones.

Softmax. The softmax action-selection mechanism is also widely used in the TD literature. It is a probabilistic selection mechanism and when it is applied to evolutionary computation it works on the basis of the following idea. The higher is the current estimate of the fitness of an individual in the population, the higher is its probability of being selected. There are many ways to implement this idea, but one of the most popular relies on using a Boltzmann distribution [14]. When it comes to applying it to evolutionary computation, the probability of selecting an individual p in the population is computed as [19]

$$\frac{e^{\frac{f(p)}{\tau}}}{\sum_{p' \in P} e^{\frac{f(p')}{\tau}}}$$

where $f(p)$ is the current fitness of p, P is the population and τ is a parameter to control the exploration/exploitation balancing. The higher is the value of τ, the more even the selection probabilities are. Algorithm 3 describes in details the softmax evaluation strategy. First, all the individuals in

the population are evaluated once (Algorithm 3, line 2). Then, the current fitness of individuals in the population

Algorithm 3 Softmax selection strategy.

```
1:  procedure SELECT(P)                          ▷ P is the population
2:      if ∃p ∈ P|n(p) = 0 then                  ▷ First all individuals are
            evaluated
3:          return p;
4:      else
5:          total = Σ_{p∈P} e^{f(p)/τ}
6:          for all p ∈ P do
7:              if rand() < e^{f(p)/τ} / total then
8:                  return p;
9:              else
10:                 total = total − e^{f(p)/τ};
11:             end if
12:         end for
13:     end if
14: end procedure
```

is taken into account according to Boltzmann distribution (Algorithm 3, line 5) and it is used to select probabilistically an individual(Algorithm 3, line 7). As in the previous case, the `SELECT` procedure described in Algorithm 3 can be easily plugged into Algorithm 1 to use softmax evaluation

strategy. Concerning the termination criteria, we used the same one proposed in [19], that is a constant number of evaluation computed as

$$|P| \cdot \theta_{eval}$$

where $|P|$ is the size of the population and θ_{eval} is a control parameter.

IV. On-line Neuroevolution for TORCS

The application of on-line learning techniques to modern computer games is a promising but challenging research direction. In this work we focused on The Open Car Racing Simulator (TORCS) [1], a state-of-the-art open source car racing simulator. We propose, as testbed, the two following problems: (i) evolving a fast controller from scratch and (ii) optimizing an existing controller for a new track. Both these tasks are designed as on-line learning problem, i.e., the learning happens while the game is running and the focus is on maximizing the performance *during* the learning rather than on finding the optimal solution. In the first task the controller is evolved from scratch, i.e., we do not have any previous knowledge about how a fast controller looks like. In the second task we are given a fast controller evolved for a different track and we wish to optimize it on a new track. In order to apply effectively the on-line neuroevolution approach introduced before to the problems considered here, we designed a specific evaluation process

to compute the fitness of candidate solutions. The rest of this section is organized as follows. First of all we provide a brief description of TORCS. Then we describe the architecture of the controllers evolved to solve the task introduced above. Finally, the process used to evaluate the candidate solution and the fitness function are described in details.

Fig. 1. A screenshot from TORCS.

A. *TORCS*

TORCS provides a full 3D visualization, many tracks, a lot of types of car, and different game modes (Figure 1

shows a screenshot from the game). The car dynamics is accurately simulated with a sophisticated physics engine, that takes into account many aspects of the racing car (e.g. traction, aerodynamics, fuel consumption, etc.). Each vehicle is controlled by its own automated driver, also called *bot*. The game is provided with a lot of human programmed bots, that can be easily extended and customized to develop your own bot. The game provide a lot of information on the current state of the car and of the race, including the position on the track, the distance between the bot an other cars, the current speed, etc. However such information can be also easily preprocessed to provide the desired representation of the surrounding game environment. At each control step, each bot controls the gas/brake pedals, the gear stick, and steering wheel on the basis of sensory information it perceives. Although TORCS was not specifically designed to perform machine learning research, it is well suited to our purposes. In fact, it combines many features typical of commercial racing simulators with a modular and customizable software architecture. The experimental analysis performed in this paper has been carried out using the same version of TORCS used for the Simulated Car Racing Competition held in association with WCCI-2008 (we refer the interested reader to [6] for a detailed description of this event). In addition, we used a customized aerodynamic model that improves the grip, in order to slightly simplify the search for reliable controllers.

B. Controller Design

TORCS provides a lot of information about the surrounding game environment and the current race state. In this work, to define the input of the neural controller we focused on a subset of the sensory inputs available in the Simulated Car Racing Competition held in association with WCCI-2008 [6]. In particular we used as input (i) six range finder sensors to perceives the track edges along several directions: - 90°, - 60°, - 30°, + 30°, + 60°, + 90°; (ii) an aggregate sensor

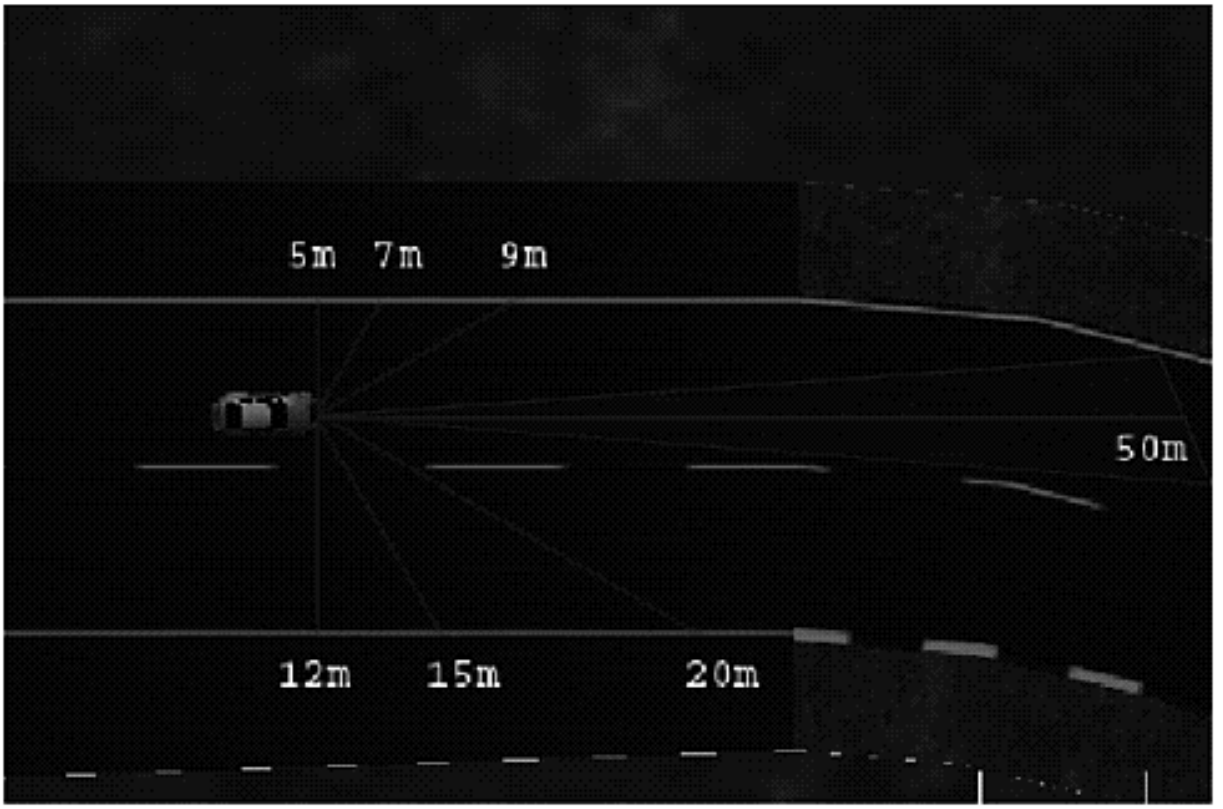


Fig. 2. An example of the six range finders and the aggregate frontal sensors used as input to neural controller.

that combines the readings of the three range finders along the direction - 10°, 0° and + 10° (see Figure 2); (iii) the current speed of the car. The controller provides two real-valued outputs to control both the steering wheel and the gas/brake pedals. In addition, when the car is in a straight segment of the track, i.e., when the frontal sensors of the car does not perceive the track edge within 100 meters, the gas pedals is set by default to the maximum value. Therefore, the controllers deal with the gas/brake pedals only when facing a turn. Such a design choice forces the controller to drive fast since the early generations and prevents the evolutionary search from wasting time with reliable but slow controllers. For the sake of simplicity, in this work we do not consider the control of the gear shifting policy and use a programmed policy to control this component of the car.

C. Evaluation Process Design

When it comes to evolve a controller for TORCS in an off-line learning scenario the design of the evaluation process is rather straightforward: each controller can be evaluated on the basis of its performance during a complete lap of the track. Instead, in the problem considered in this work, the learning happens on-line, i.e., while the game is running. Therefore, a controller at once has to be evaluated live in the game. As soon as an evaluation is finished, the following controller in the population or the first controller of the

next generation, takes the place of the one just evaluated. However, this evaluation process involves some technical issues: (i) the outcome of each evaluation may be affected by the previous one; (ii) a sudden change of the controller might not be safe; (iii) extremely poor controller might stop the evaluation process. To deal with the first issue, we introduced a warm-up interval before the evaluation lap actually begins. Such a mechanism makes two sequential evaluations almost independent because each controller has enough time to take the full control of the car before the evaluation starts. The second issue is a serious concern: different controllers might have a totally different way to deal with the same game situation. Thus, as soon as we change suddenly the controller policy of the car, the outcome might be unpredictable and it might even result in loosing the control of the car. To deal with this issue, we introduced a smooth transition from the old controller to the new one, i.e., the car controls are computed as a weighted average of the outputs of the two controllers. Finally, to deal with the third issue we used the following approach. As soon as a a controller goes outside of the track edges, the evaluation ends and a programmed recovery policy is used to bring the car in the correct position on the track, before the evaluation of the next controller will start.

Although the evaluation process just introduced (dubbed *off-line evaluation strategy* in the rest of the paper) can be applied to the on-line learning problems considered here, we

do not expect it would be able to solve them effectively. In fact, in the off-line evaluation strategy, *each* controller in the population is evaluated for a long interval (more than a lap) and therefore it affects heavily the average performance of the system *during* the game. Instead, to solve effectively the on-line learning problems considered, the evaluation process should focus on the most promising controllers rather than on the poor ones. To this purposes we used a new evaluation process that combines the on-line evolutionary computation approach described in Section III with a fast but approximated controllers evaluation. Each controller is evaluated only for T_{eval} game tics and not for a complete lap as in the off-line evaluation strategy. As a result the outcome of an evaluation heavily depends on which part of the track the controller faced. Accordingly, it becomes necessary to perform more evaluations of the same controller to assess its fitness. Therefore, we can apply the ε-greedy and softmax evaluation strategies described in the previous section to achieve a good trade-off between the search for new solutions (exploration) and the exploitation of the best solution discovered so far.

Concerning the fitness of the controllers, in the off-line evaluation strategy it is computed as the outcome of a single evaluation, while in the ε-greedy and softmax evaluation strategies the outcomes of several evaluations are averaged to compute the fitness. However, the outcome of the single evaluation is computed as follows, for all the three evaluation

strategies:

$$eval = \alpha - T_{out} + \beta \cdot \bar{s} + d,$$

where T_{out} is the number of game tics the car is outside the track; $\bar{s}$ is the average speed (meters for game tic) during the evaluation; d is the distance (meters) raced by the car during the evaluation; α and β are two constants introduced respectively to make sure that the fitness is positive and to weight to scale the average speed term (both α and β have been empirically set to 1000 in all the experiment reported here).

V. Experimental Results

We applied NEAT with different evaluation strategies (see Section III) to the tasks described in the previous section. This section is organized as follows. First, we present and discuss the results of the first experiment where we used NEAT to evolve from scratch a fast controller for a specific track in TORCS. Then we move to the second experiment, where NEAT is used to optimize a previously evolved controller for a new track,

A. Learning to drive from scratch

In the first experiment we applied NEAT to learn a fast controller for the `Wheel 1` track, depicted in Figure 3(b), available in the standard TORCS package. We assumed to

not have any previous knowledge of the structure a fast controller should have and thus we started the evolutionary process from a network with a minimal topology i.e., the input layer fully connected to the output layer without any hidden neurons. We compared three different evaluation strategies, off-line, ε-greedy and softmax, using a population of 100 individuals. The experiment consisted of 10 runs and each run involved a race of 2000 laps on the `Wheel 1` track[1]. For the ε-greedy strategy we set $\varepsilon = 0.25$ and $\theta_{best} = 5$; while for the softmax strategy we set $\tau = 50$ and $\theta_{eval} = 3$. Figure 4 compares the average lap-time achieved

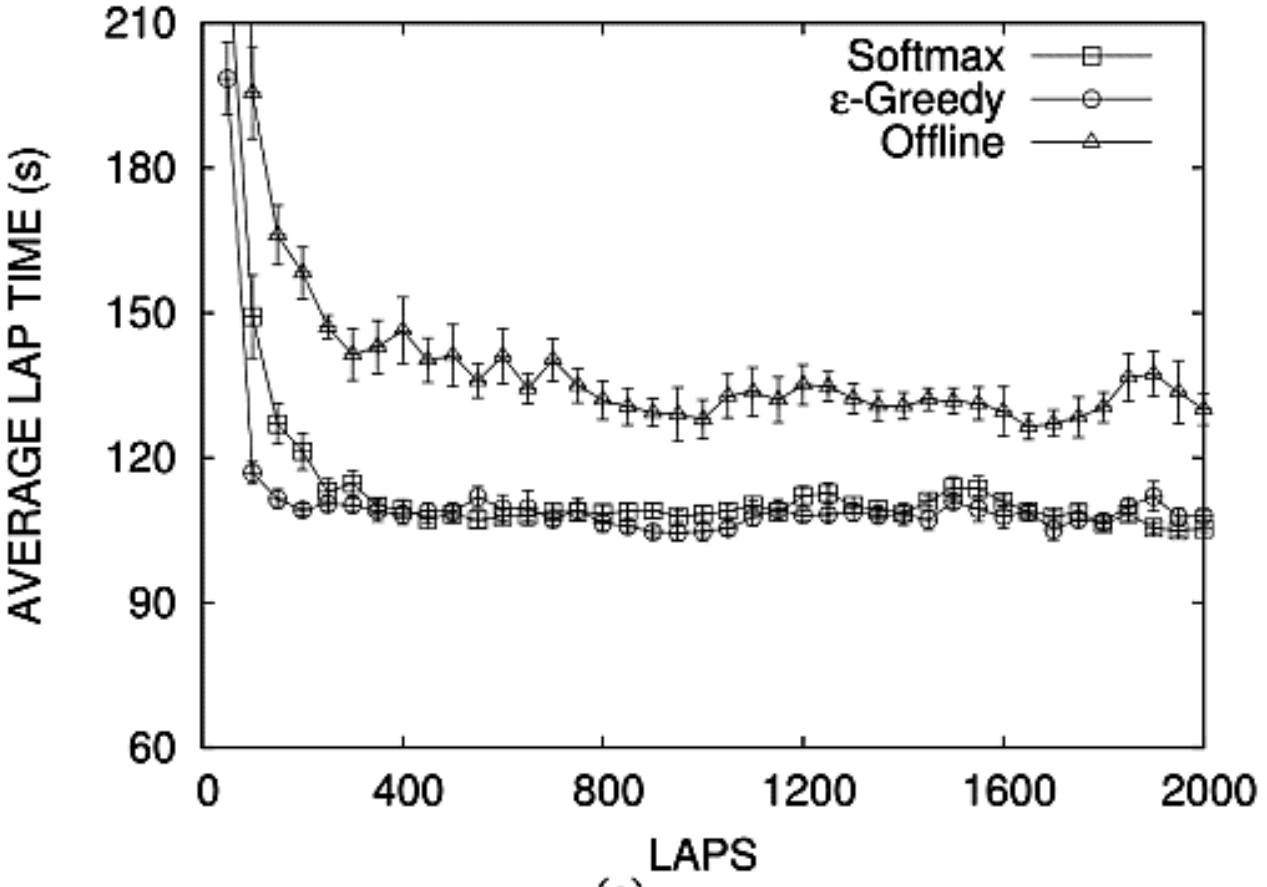
(a) Aalborg (b) Wheel 1

Fig. 3. Tracks used for the experiments reported in this paper.

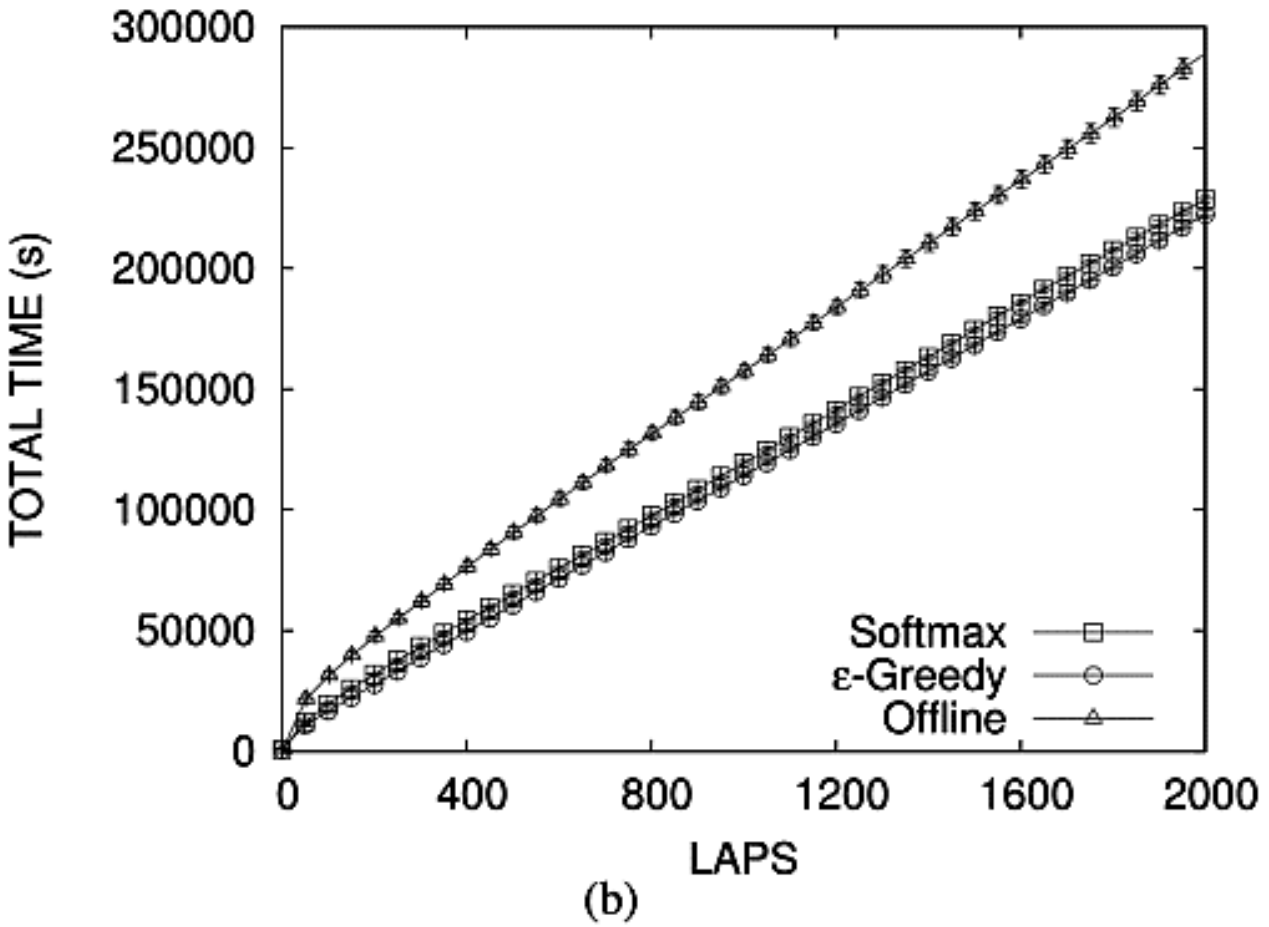
during the race by NEAT with off-line, ε-greedy and softmax evaluation strategies. Results show that both ε-greedy and softmax evaluation strategies outperform the off-line one in

terms of on-line performance. In particular, NEAT with ε-greedy and softmax reaches a good on-line performance after few hundreds of laps, while the performance of NEAT with the off-line strategy is poor even after 2000 laps (see Figure 4(a)). The differences is still more evident if we look at Figure 4(b) that shows the overall racing time elapsed during the experiment: the off-line evaluation strategy took approximately 60000 seconds of simulated racing time (i.e., 16 hours) more than the time taken by on-line evaluation strategies. Instead, the differences between ε-greedy and softmax strategies is very small even though the ε-greedy strategy seems to learn slightly faster (see Figure 4(a)). Figure 5 compares the best lap-time achieved by NEAT to the one achieved by the best programmed controller available in TORCS. Results show that NEAT outperforms the programmed controller after few hundreds of laps. In particular,

[1]In this work we use the number of laps to define the length of the experiments instead of the usual number of generations or the number of evaluations. We chose this measure because we wanted a measure of the learning speed that is easy to understand in term of game experience.


AVERAGE LAP TIME (s)
210
180
150
120
90
60
0
400
800
1200
1600
2000
LAPS
Softmax
ε-Greedy
Offline

(a)


TOTAL TIME (s)
300000
250000
200000
150000
100000
50000
0
0
400
800
1200
1600
2000
LAPS
Softmax
ε-Greedy
Offline

(b)

Fig. 4. NEAT with off-line, ε-greedy and softmax selection strategies evolving from scratch a controller for `Wheel 1` track: (a) average lap-time in the last 50 laps and (b) overall racing time Curves are averaged over 10 runs.

the off-line evaluation strategy is faster than the others at reaching a better performance than the best programmed controller available in TORCS. However, the performances of the three strategies at the end of the experiments are not significantly different. The results obtained suggest that, as already discussed in [19], the off-line evaluation strategy is outperformed by the on-line evaluation strategies during the learning process because it is not able to deal effectively with the exploration/exploitation trade-off. It is interesting to point out that our results do not suggest that the softmax is better than ε-greedy as found by Whiteson and Stone [19]. By the way, the ε-greedy evaluation strategy used in this work is substantially different from the one introduced in [19]: the number of evaluations is not fixed for each generation (as happens in the off-line and softmax strategies) but it is dynamically computed in order to guarantee that a fixed number of evaluations are assigned to the best solution found in each generation. With such a mechanism, the ε-greedy evaluation avoids wasting time evaluating poor solutions and thus it results competitive with softmax strategy. On the other hand, when we analyzed the best lap time achieved by NEAT, i.e., a typical off-line learning metrics, we found that the off-

line evaluation strategy learns faster than the other ones, even if all the three strategies reach almost the same performance

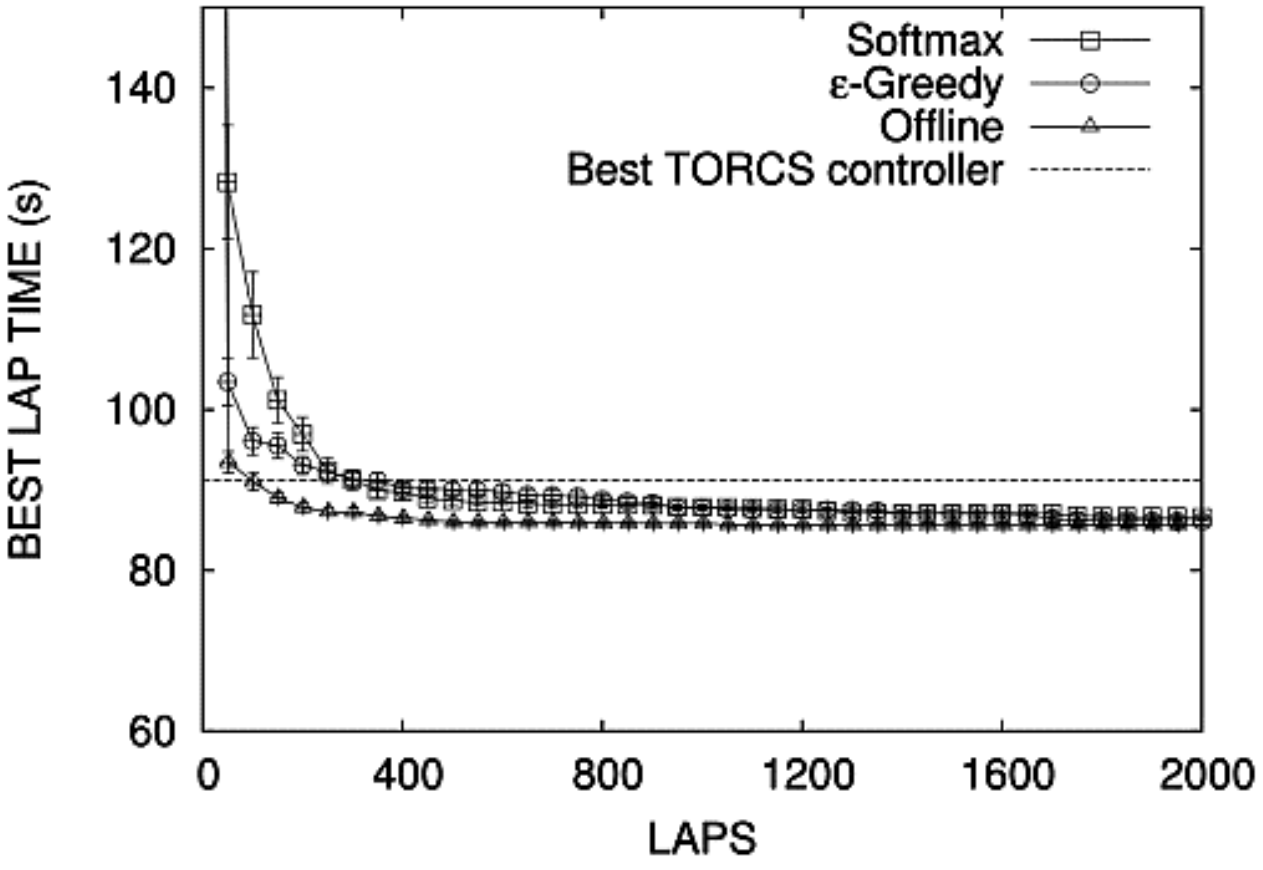


Fig. 5. Best lap times achieved by NEAT with off-line, ϵ-greedy and softmax selection strategies applied to the problem of evolving from scratch a controller for `Wheel 1` track. Curves are averaged over 10 runs.

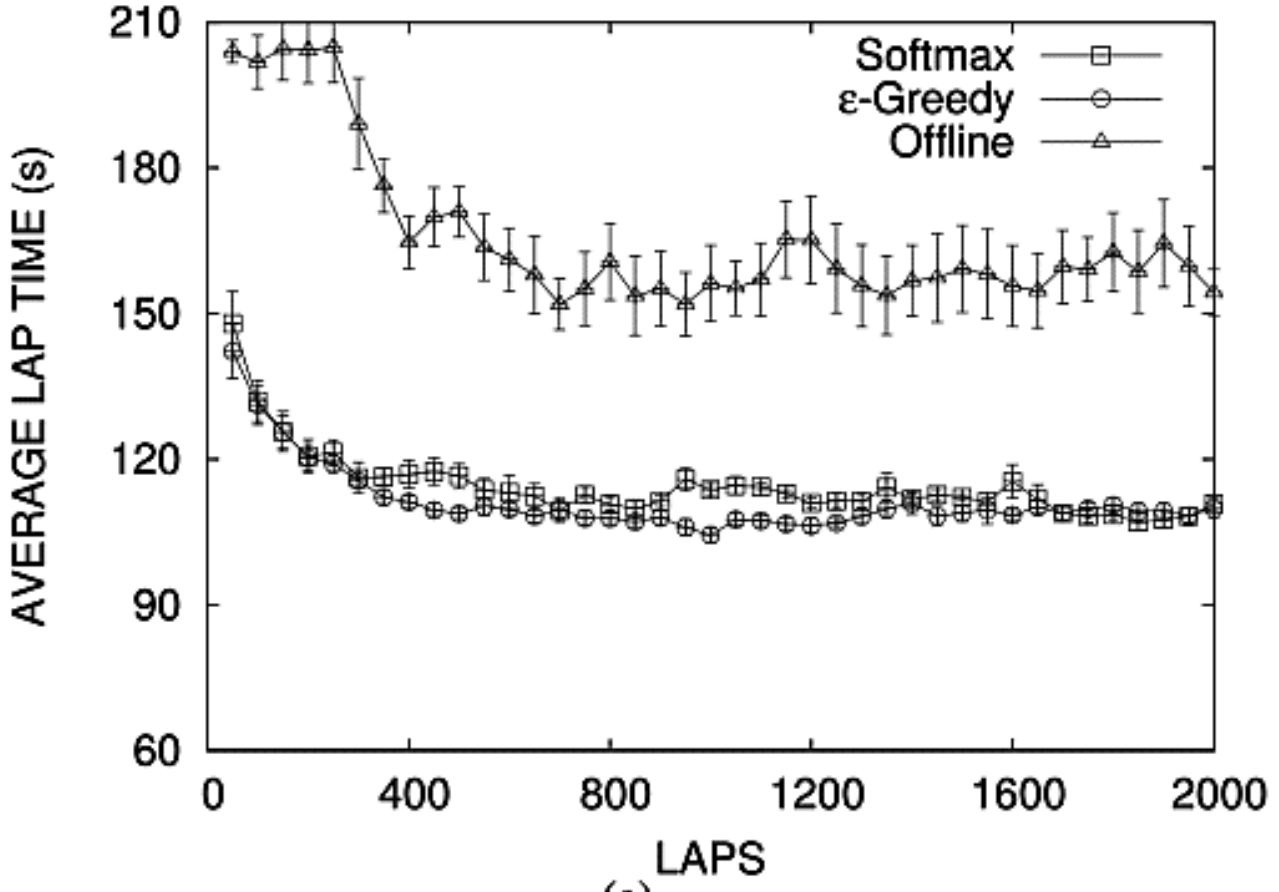
at the end of the experiment.

B. Learning to drive on a new track

In the second experiment we applied NEAT to evolve a fast controller for the `Wheel 1` track from a controller previously evolved for the `Aalborg` track, depicted in Figure 3(a). In this experiment we seeded the initial population

with a controller evolved off-line for the Aalborg track. As in the previous experiment, we compared three different evaluation strategies, off-line, ε-greedy and softmax, using a population of 100 individuals. The experiment consisted of 10 runs and each run involved a race of 2000 laps on the Wheel 1 track. For the ε-greedy strategy we set $\varepsilon = 0.25$ and $\theta_{best} = 5$; while for the softmax strategy we set $\tau = 50$ and $\theta_{eval} = 3$. Figure 6 compares the average lap-time achieved during the race by NEAT with off-line, ε-greedy and softmax evaluation strategies. Also in this second experiment, both ε-greedy and softmax evaluation strategies outperform the off-line one during the learning process. In this task the difference is still more evident as can be noticed when looking at the overall time that the three evaluation strategies need to complete 2000 laps (see Figure 6(b)): the off-line strategy took approximately 100000 seconds of simulated racing time (i.e., more than 27 hours) more than the time taken by the other strategies. At the opposite, the differences between softmax and ε-greedy is even smaller in this second experiment. Concerning the best lap-time achieved, Figure 7 compares NEAT to the the best programmed controller available in TORCS. Results show that NEAT outperforms the programmed controller at the end of experiment, even if much more laps are necessary with the respect to the first experiment. It can be also noticed, that performances achieved by the three evaluation strategies are not significantly different at the end of the

experiment although they are worse than the performances achieved in the first experiment. Finally, in this experiment, the off-line evaluation strategy learns slower than ε-greedy and softmax even when the best-lap time is used as metric. The results obtained show that seeding the population with a



(a)

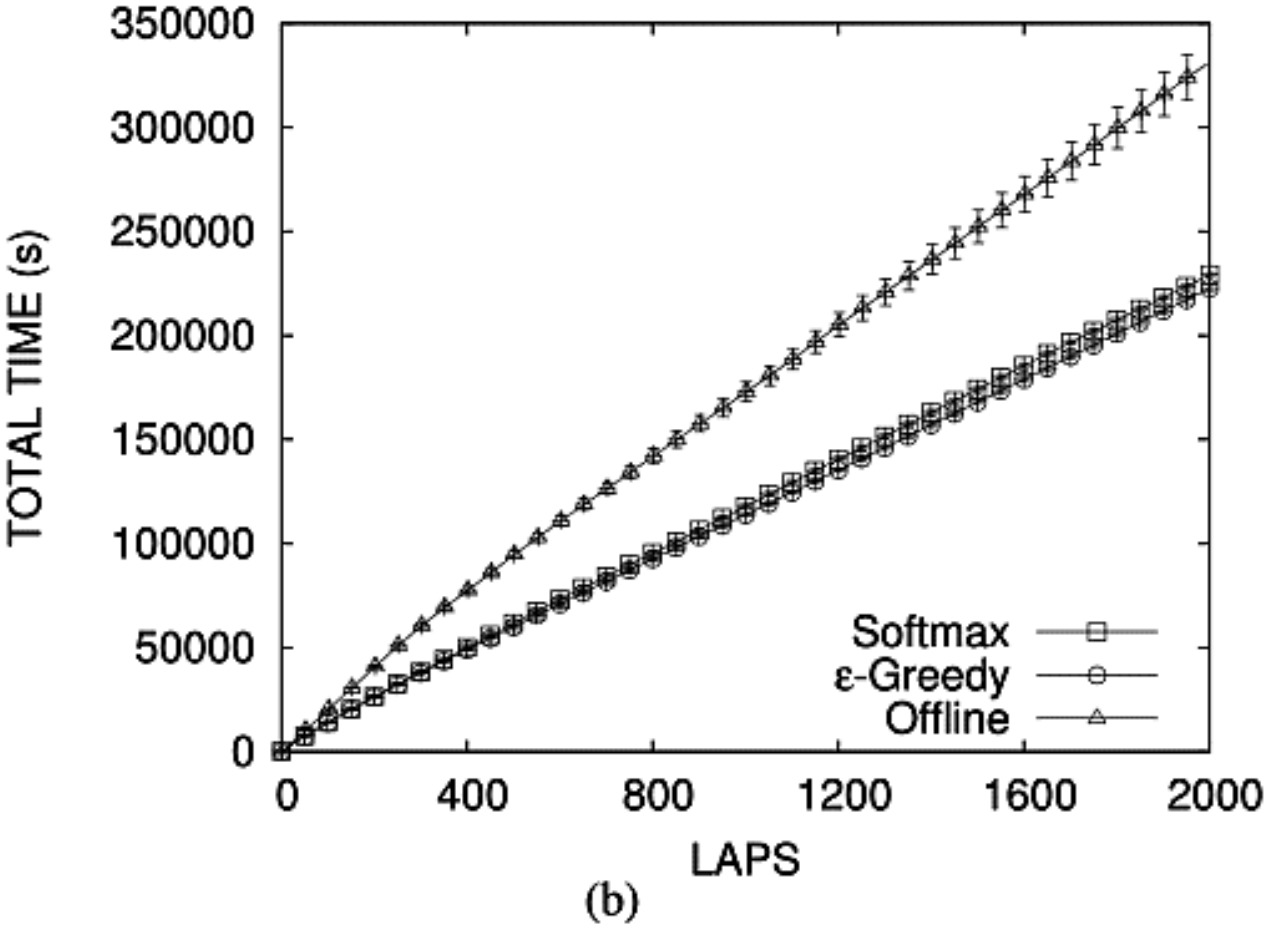


Fig. 6. NEAT with off-line, ε-greedy and softmax evaluation strategies applied to the problem of optimizing a previously evolved controller for a different track. (a) average lap-time in the last 50 laps and (b) overall racing time. Curves are averaged over 10 runs.

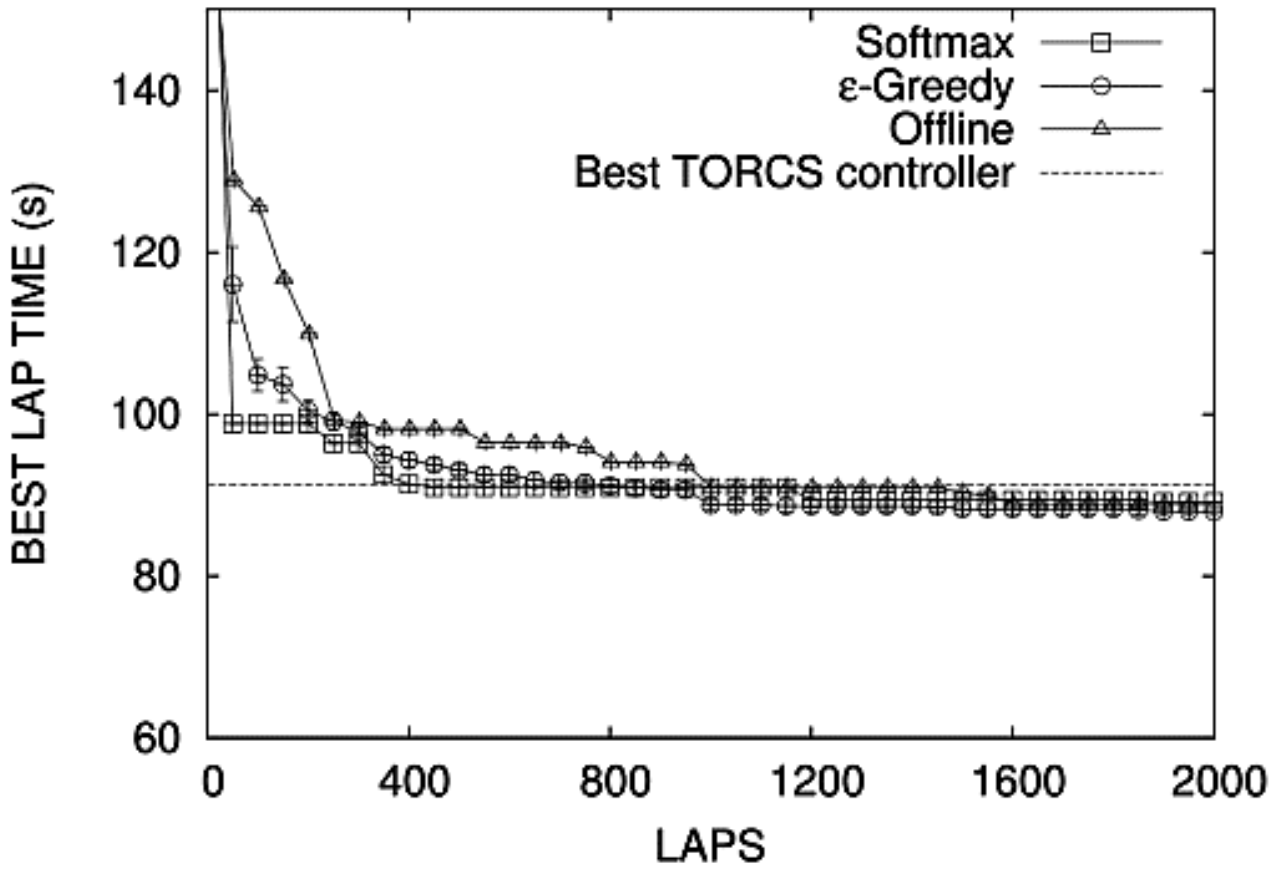


Fig. 7. Best lap times achieved by NEAT with off-line, ε-greedy and softmax selection strategies applied to the problem of optimizing a previously evolved controller for a different track. Curves are averaged over 10 runs.

previously evolved solution for a slightly different task might speedup the learning process in the early stages but might prevent NEAT from evolving an optimal solution. In fact as Figure 6(a) shows after an initial boost to the performances, the learning process get stuck and is not able to improve anymore. However, NEAT is still able to improve the best lap-time achieved by the TORCS programmed controller

(reported here as standard reference). It is important to stress that, in this second task, the off-line evaluation strategy leads to a slower learning even when we consider as performance the best lap-time (see Figure 7), a typical off-line learning metrics.

VI. Conclusions and Future Works

In this work we applied NEAT to TORCS, following the on-line neuroevolution approach introduced in [19]. We focused on two different tasks: (i) evolving a fast controller from scratch and (ii) optimizing an existing controller for a new track. Both these tasks have been designed as on-line learning problem, i.e., the learning happens running live the game and the focus is on maximizing the performance *during* the learning. In order to apply effectively on-line neuroevolution to such problems, we replaced the typical fitness evaluation performed in an off-line scenario with several faster but less accurate evaluations. Then, we compared different evaluation strategies, inspired to the action-selection mechanism used in temporal difference learning, to deal with the problem of finding a good trade-off between searching for new solutions in the population and exploiting the best one discovered so far. Our results show that, using the evaluation strategies proposed in [19], it is possible to improve effectively the on-line performance of NEAT. When it comes to optimize an existing solution, the results suggest that on-line neuroevolution might improve the performance of NEAT even considering an off-line performance measure.

The work presented here is still preliminary but the results obtained are promising and represent a good starting point for further analysis and improvements. In particular, a simple but interesting extension to this work would be using rtNEAT [11], a steady-state version of NEAT. Then, applying an action-selection mechanism with a decreasing exploration rate [14] might be more suitable for the problems considered here. Finally, we think that exploiting a given solution for a slightly different problem to speedup the neuroevolution is an important issue that needs a deeper analysis in a future work.

REFERENCES

[1] The open racing car simultaor website. http://torcs.sourceforge.net/.

[2] T. Beielstein and S. Markon. Threshold selection, hypothesis tests, and doe methods. *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, 1:777–782, May 2002.

[3] Benoit Chaperot and Colin Fyfe. Improving artificial intelligence in a motocross game. In *IEEE Symposium on Computational Intelligence and Games*, 2006.

[4] David E. Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 41–49, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[5] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems: From Foundations to Applications*, volume 1813 of *Lecture Notes in Computer Science*. Springer-Verlag, April 2000.

[6] Daniele Loiacono, Julian Togelius, Pier Luca Lanzi, Leonard Kinnaird-Heether, Simon M. Lucas, Matt Simmerson, Diego Perez, Robert G.

Reynolds, and Yago Saez. The wcci 2008 simulated car racing competition. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.

[7] Steffen Priesterjahn, Kramer, Alexander Weimer, and Andreas Goebels. Evolution of human-competitive agents in modern computer games. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, 2007.

[8] Larry D. Pyeatt and Adele E. Howe. Learning to race: Experiments with a simulated race car. In *Proceedings of the Eleventh International Florida Artificial Intelligence Research Society Conference*, pages 357–361. AAAI Press, 1998.

[9] Pieter Spronck. *Adaptive Game AI*. PhD thesis, University of Maastricht, 2005.

[10] Peter Stagge. Averaging efficiently in the presence of noise. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 188–200, London, UK, 1998. Springer-Verlag.

[11] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the nero video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, 2005.

[12] Kenneth O. Stanley, Nate Kohl, Rini Sherony, and Risto Miikkulainen. Neuroevolution of an automobile crash warning system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)*, 2005.

[13] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[14] R. Sutton and A. Barto. *Reinforcement Learning*. MIT Press, 1998.

[15] Julian Togelius. *Optimization, Imitation and Innovation: Computational Intelligence and Games*. PhD thesis, Department of Computing and Electronic Systems, University of Essex, Colchester, UK, 2007.

[16] Julian Togelius and Simon M. Lucas. Evolving controllers for simulated car racing. In *Proceedings of the Congress on Evolutionary Computation*, 2005.

[17] Julian Togelius and Simon M. Lucas. Arms races and car races. In *Proceedings of Parallel Problem Solving from Nature*. Springer, 2006.

[18] Julian Togelius and Simon M. Lucas. Evolving robust and specialized car racing skills. In *Proceedings of the IEEE Congress on Evolutionary*

Computation, 2006.

[19] Shimon Whiteson and Peter Stone. On-line evolutionary computation for reinforcement learning in stochastic domains. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1577–1584, New York, NY, USA, 2006. ACM.

[20] Krzysztof Wloch and Peter J. Bentley. Optimising the performance of a formula one car using a genetic algorithm. In *Proceedings of Eighth International Conference on Parallel Problem Solving From Nature*, pages 702–711, 2004.