

**Evolving Drivers for TORCS  
using On-Line Neuroevolution**

**Luigi Cardamone, Daniele Loiacono,  
and Pier Luca Lanzi  
IlliGAL Report No. 2009008  
November, 2009**

Illinois Genetic Algorithms Laboratory  
University of Illinois at Urbana-Champaign  
117 Transportation Building  
104 S. Mathews Avenue Urbana, IL 61801  
Office: (217) 333-2346  
Fax: (217) 244-5705

# Evolving Drivers for TORCS using On-Line Neuroevolution

Luigi Cardamone<sup>†</sup>, Daniele Loiacono<sup>†</sup>, Pier Luca Lanzi<sup>†\*</sup>

<sup>†</sup>Artificial Intelligence and Robotics Lab.  
Dip. di Elettronica e Informazione  
Politecnico di Milano  
Milano 20133, Italy  
{cardamone,loiacono,lanzi}@elet.polimi.it

\*Illinois Genetic Algorithms Lab.  
Dept. of General Engineering  
University of Illinois at Urbana-Champaign  
Urbana, Illinois, USA  
{lanzi}@illigal.ge.uiuc.edu

November 2, 2009

## Abstract

We applied on-line neuroevolution to evolve non-player characters for The Open Racing Car Simulator. While previous approaches allowed on-line learning with performance improvements during each generation, our approach enables a finer grained on-line learning with performance improvements within each lap. We tested our approach on three tracks using two methods of on-line neuroevolution (NEAT and rtNEAT) combined with four evaluation strategies ( $\varepsilon$ -Greedy,  $\varepsilon$ -Greedy-Improved, Softmax, and Interval-based) taken from the literature. We compared the eight resulting configurations on several driving tasks involving (i) the learning of a driving behavior for a specific track, (ii) its adaptation to a new track, and (iii) the generalization capability to unknown tracks. The results we present show that our approach can successfully evolve drivers from scratch and can also be used to transfer evolved knowledge to other tracks. Overall, our results suggest that the approach performs significantly better when coupled with on-line NEAT and also indicate that  $\varepsilon$ -Greedy-Improved, Softmax are generally better than the other evaluation strategies.

## 1 Introduction

The application of neuroevolution to modern computer games has been mainly restricted to off-line learning (e.g., [17, 5, 20, 14]). In the typical scenario, an evolutionary algorithm is applied to a population of networks [25, 17, 35, 5]; at each generation, the fitness of each network is evaluated by applying it to the game for a rather long timeslot (e.g., to play one or more matches [17], to drive one or more laps [8], etc.); then selection, recombination and mutation are applied as usual; these steps are repeated until a termination condition is met. At the end, the best individual is deployed to the actual game. Off-line neuroevolution can also be applied in *real-time* to train a team of agents during the game by replacing the rather expensive generational evolutionary process with a steady-state algorithm [25, 34].

In the field of computer games, there are several interesting applications that involve on-line learning scenarios [25] and thus may benefit from on-line neuroevolution. Recently, Whiteson and Stone [32] focused on neuroevolution for stochastic reinforcement learning problems and proposed an approach to enable on-line learning by modifying the fitness evaluation process. Neuroevolution typically deals with stochastic problems by computing the fitness of each individual as the average over a fixed number of repeated evaluations (e.g., as the average over 100 trials [32]). In contrast,

Whiteson and Stone [32] define a reservoir of evaluations that are allocated to individuals based on a mechanism that borrows from reinforcement learning [28]. As in off-line neuroevolution, each evaluation involves an entire problem instance (e.g., one match of an arcade game, one lap in a racing game, or an episode of a reinforcement learning task [32]). Opposite to what happens in typical off-line neuroevolution, each individual receives a variable number of evaluations that depends on its performance: most promising individuals receive an increasingly higher number of evaluations while less promising individuals receive fewer and fewer evaluations [32].

The approach of Whiteson and Stone [32] enables *on-line learning within the same generation* in that, more and more evaluations are allocated to the best performing individuals, so that the overall performance *improves during each generation*. However, it does not allow any learning or adaptation with a finer granularity since each evaluation exactly corresponds to one problem instance. This represents a significant limitation for the application of [32] to computer games which often require learning at a finer granularity.

In this work, we applied on-line neuroevolution to evolve drivers for The Open Racing Car Simulator (TORCS) [1], a state-of-the-art open source car racing simulator. Our approach, preliminarily introduced in [7, 6], extends the approach of [32, 23] to the realm of simulated car racing, allowing on-line learning at a finer granularity than [32]. In previous approaches [32, 35, 17, 5, 20], evaluations correspond to one or more problem instances. In contrast, our approach focuses on *very short timeslots* which cover only a small fraction of a problem instance. Accordingly, our evaluations involve only very small parts of the overall task. Consequently, while in [32, 35, 17, 5, 20], a problem instance is solved by one individual, in our approach several individuals might be used in sequence to solve one problem instance.

Our approach reduces the time spent for the evaluation of poor performing drivers. In fact, by using very short timeslots to evaluate individuals, it is possible to focus on the best drivers *within the same lap* (i.e., within the same problem instance). In [32], the decision of what individual should be evaluated affects one or more problem instances (one or more laps in our case), so that large amounts of time may be spent to evaluate poor drivers. In our case, the same decision usually affects only a very small portion of a problem instance (a small fraction of the overall lap), so that less time is wasted on less promising individuals. Thus, the use of very short timeslots leads to a finer grained on-line learning with performance improvements *during each lap*, and not just *during each generation* as in [32]. However, the use of short timeslots also poses major challenges. Firstly, since the individuals are tested only on a fraction of the overall task, their evaluation is extremely noisy as the same driver may be evaluated on very different parts of the track. Secondly, since learning happens on-line and involves one car, individuals have to be tested sequentially so that (i) subsequent evaluations are highly correlated; (ii) evaluations are performed starting from very different initial conditions, (as opposed to [32] where all the evaluations start from the same initial conditions); (iii) there are abrupt discontinuities in the learning process which must be adequately managed. Finally, since the individuals are tested only on a fraction of the overall track, in principle, there is no guarantee that at the end there will be an individual capable of driving on the whole track.

We tested our approach using two methods of on-line neuroevolution and four possible evaluation strategies for the allocation of timeslots to the individuals in the populations. We compared the performance of the eight resulting configurations, both in terms of learning capabilities and learning speed, on several driving task involving (i) the learning of a driving behavior for a specific track, (ii) its adaptation to a new track, and (iii) the generalization capability to unknown tracks. In particular, we considered on-line NEAT, introduced by Whiteson and Stone [32], and its steady-state version, on-line rtNEAT, introduced by Reeder et al. [23]. With respect to the evaluation selection strategies, we considered the three policies introduced in [32] (namely,  $\epsilon$ -Greedy, Softmax,

and Interval-based) and  $\varepsilon$ -Greedy-Improved we firstly introduced in [7, 6].

The results we present here show that our approach can successfully evolve drivers from scratch and can also be used to adapt networks evolved for a certain track to another one. Although, in our approach evaluations are more noisy than in [32] and performed only on a fraction of the target driving task, yet the best evolved networks are capable of driving on the entire track and generalizing to unknown and more difficult tracks. In particular, our results show that when learning to drive from scratch, on-line rtNEAT initially performs significantly better than NEAT but later on it is outperformed by on-line NEAT which, at the end, reaches a significantly better performance. Instead, when adapting a driver to another track, our results suggest that on-line NEAT is able to learn faster. However, at the end, both approaches reach a comparable performance with no statistically significant difference between their performances. The experiments on generalization show that on-line NEAT produces more drivers capable of driving on several difficult unknown tracks. Finally, with respect to the evaluation strategy, all our experiments show that  $\varepsilon$ -Greedy is significantly worse than the other strategies, which perform similarly, confirming the findings in [32]; they also suggest that Softmax and  $\varepsilon$ -Greedy-Improved perform better than the others, extending the findings in [32].

The paper is organized as follows. In Section 2, we briefly overview The Open Racing Car Simulator (TORCS) [1]. In Section 3, we present Neuroevolution with Augmenting Topology (NEAT), the off-line neuroevolution approach that has been extended for on-line learning in [32, 23]. In Section 4, we overview on-line NEAT, on-line rtNEAT and the four evaluation selection strategies we compared. In Section 5, we discuss the published work that relates to our study. In Section 6, we discuss our approach and in Section 7 we outline the experimental design we followed. In Section 8, we present the results of our experiments and the statistical analysis we performed. Finally, in Section 9, we draw some conclusions from our study and outline the possible future research directions.

## 2 TORCS

The Open Racing Car Simulator (TORCS) [1] is a state-of-the-art open source car racing simulator which provides a sophisticated physics engine, full 3D visualization, several tracks, several models of cars, and various game modes (e.g., practice, quick race, championship, etc.). The car dynamics is accurately simulated and the physics engine takes into account many aspects of racing cars such as traction, aerodynamics, fuel consumption, etc.

Each car is controlled by an automated driver or *bot*. At each control step (game tick), a bot can access the current game state, which includes several information about the car and the track, as well as the information about the other cars on the track; a bot can control the car using the gas/brake pedals, the gear stick, and steering wheel. The game distribution includes many programmed bots which can be easily customized or extended to build new bots.

All the experiments reported in this paper have been carried out with the version 1.3.1 of TORCS using the setup of the 2009 Simulated Car Racing Championship (see the competition pages for CEC-2009<sup>1</sup>, GECCO-2009<sup>2</sup>, CIG-2009<sup>3</sup>, and [16]). The car sensory inputs consist of two rangefinder sensors (one for sensing the track borders, one for sensing the opponents) and other sensors describing the car state such as, the car direction with respect to the track axis, the amount of damages, the current gear, etc. (see [16] for the complete list). The car is controlled by four

---

<sup>1</sup><http://www.cec-2009.org/>

<sup>2</sup><http://www.sigevo.org/gecco-2009>

<sup>3</sup><http://www.ieee-cig.org/cig-2009>

effectors: the steering wheel, the gas pedal, the brake pedal, and the gear change [16].

### 3 Neuroevolution with Augmenting Topology (NEAT)

In this paper, we focused on Neuroevolution with Augmenting Topology or NEAT [27], one of the most successful and widely applied neuroevolution approaches. NEAT [27] works as the typical population-based selecto-recombinative evolutionary algorithm: first the fitness of the individuals in the population is evaluated, then selection, recombination and mutation operators are applied, and this cycle is repeated until a termination condition is met. NEAT is based on rather simple but very effective principles. First, NEAT does not make any assumption about the optimal topology of the target neural network nor about the type of connections involved (e.g., they can be feed-forward or recursive). Accordingly, its evolutionary search starts from the simplest topology possible (i.e., a fully connected network containing only the input and the output layers) and complex structures emerge during the evolutionary process, surviving only when useful. Furthermore, NEAT deals with the problem of recombining networks with different structures through a *historical marking* mechanism that assigns a unique *innovation number* to genes originated from structural mutations. Recombination uses the innovation number to identify similarities between networks without the need of complex and expensive topological analyses. Finally, NEAT protects the structural innovations through the mechanism of *speciation* which restricts the selection process to niches containing networks with similar topology. To identify such niches, NEAT uses the innovation numbers of the genes to measure the similarities between topologies. To prevent a single species from taking over the population, the networks in the same niche share their fitness [13].

Later, an extension of NEAT for real-time learning (rtNEAT) was introduced to evolve populations of agents for the game of NERO [26], while the game is being played. While NEAT focuses on finding the very best individual to solve a certain problem, rtNEAT focuses on finding the best team of agents quickly. For this purpose, rtNEAT replaces the generational genetic algorithm of NEAT with a steady-state genetic algorithm, so that, at each generation, only one individual in the population is modified.

## 4 On-line Neuroevolution

Our work is based on two methods of on-line neuroevolution: (i) the generational approach, introduced by Whiteson and Stone [32], which extends NEAT for the on-line scenario, and (ii) its steady-state version [23], which extends the steady-state version of NEAT (rtNEAT) to an on-line scenario.

### 4.1 On-line Generational Neuroevolution with NEAT

In [32], Whiteson and Stone extended NEAT for on-line learning in stochastic problems. Instead of computing the fitness of each individual as an average over a fixed number of repeated evaluations (as usually done when off-line neuroevolution is applied to stochastic problems), a variable number of evaluations is allocated to each individual based on a mechanism that borrows from the action-selection strategies employed in reinforcement learning. For this purpose, Whiteson and Stone [32] introduce a budget of evaluations that the system can spend on the individuals in the populations. As in off-line neuroevolution, each evaluation involves one problem instance. Opposite to what happens in typical off-line neuroevolution, each individual receives a variable number of evaluations

that depends on its performance: most promising individuals receive an increasingly higher number of evaluations while less promising individuals receive fewer and fewer evaluations [32].

At each generation, the on-line version of NEAT selects which individuals in the population should be evaluated so that less promising individuals will receive fewer evaluation slots while most promising ones will receive more evaluation slots. For this purpose, the system needs to balance exploration and exploitation as reinforcement learning algorithms do. Accordingly, on-line neuroevolution applies an *evaluation selection strategy*, borrowed from reinforcement learning, to identify the most promising candidates in the population so as to allocate more resources (more evaluation slots) to them.

The on-line version of NEAT is reported as Algorithm 1. At first, the population is randomly initialized as in the most typical evolutionary algorithm (Algorithm 1, line 2). Then, the individuals in the population are evaluated (Algorithm 1 from line 4 to line 12) according to a certain action-selection strategy (Algorithm 1, line 9). For this purpose, for each individual  $p$  in the population  $P$ , the system maintains a vector  $f(p)$  estimating the individual fitness and a vector  $e(p)$  keeping the number of evaluations performed for  $p$ ; the vector  $e(\cdot)$  is initialized to zero while  $f(\cdot)$  is initialized with the minimum fitness value which depends on the problem. Every time an individual is selected for evaluation, its counter is increased (Algorithm 1, line 10) and its estimated fitness is updated (line 11) based on the result of the current evaluation and its previous value. The selection of an individual is performed by the function call  $\text{SELECT}(p)$  at line 9 in Algorithm 1 that implements one of the evaluation selection strategies proposed in the literature [32, 10] (see Section 4.3). When the criterion to stop the evaluation is met (line 12), the evolutionary process continues as usual with selection, recombination and mutation. These steps are repeated until a target number of generations have been completed.

## 4.2 On-line Steady-State Neuroevolution with rtNEAT

rtNEAT [26] is the steady-state version of NEAT in which the usual population-based selection, recombination and mutation operators are replaced with the following steps. First, the worst individual (according to its shared fitness) in the population is removed. Then, two parents are selected, recombined and mutated so as to generate a new individual which is added to the population. rtNEAT was extended for an on-line scenario in [19] following what already done by Whiteson and Stone for NEAT [32]. The on-line version of rtNEAT is reported as Algorithm 2. The only difference with respect to the generational version (Algorithm 1) are the lines from 13 to 18 that implements steady-state evolution and initialize the evaluation count  $e(p)$  and the fitness estimation  $f(p)$ .

## 4.3 Evaluation Selection Strategies

In our study, we considered all the four evaluation selection strategies introduced in the literature: three of them ( $\varepsilon$ -Greedy, Softmax and Interval-based) are the ones considered in [32];  $\varepsilon$ -Greedy-Improved is an extension of  $\varepsilon$ -Greedy we introduced [6, 7]. Each strategy has been used to implement the  $\text{SELECT}$  procedure for generational or steady-state on-line neuroevolution (see Algorithm 1 and Algorithm 2).

**$\varepsilon$ -Greedy** is probably the simplest action-selection strategy used in reinforcement learning [28] and its application to on-line evolution is rather straightforward. At each iteration, it selects with probability  $\varepsilon$  a random individual from the population while with probability  $1 - \varepsilon$  it selects the individual with the highest fitness (see Algorithm 3).

---

**Algorithm 1** Generational On-line Evolution.

---

```
1: procedure EVOLUTION( $P$ )
     $\triangleright P$  is the population,  $p$  is an individual of  $P$ 
     $\triangleright e(p)$  is the number of evaluations of  $p$ 
     $\triangleright f(p)$  is the fitness of  $p$ 
     $\triangleright$  Randomly initialize  $P$ 
2:   Init( $P$ );
3:   repeat
     $\triangleright$  Init fitness and evaluation count in  $P$ 
4:     for  $p \in P$  do
5:        $e(p) = 0$ 
6:        $f(p) = f_{min}$   $\triangleright$  Set to min fitness
7:     end for
8:     repeat
     $\triangleright$  Select the individual to evaluate
9:        $p \leftarrow \text{SELECT}(P)$ ;
     $\triangleright$  Update the evaluation count
10:       $e(p) \leftarrow e(p) + 1$ ;
     $\triangleright$  Update the fitness of based on
     $\triangleright$  its current evaluation EVAL( $p$ )
11:       $f(p) \leftarrow f(p) + \frac{1}{e(p)}(\text{EVAL}(p) - f(p))$ ;
12:     until Evaluation Termination Criteria Met;
     $\triangleright$  Usual NEAT operators
13:      $P_s \leftarrow \text{Selection}(P)$ ;
14:      $P_r \leftarrow \text{Recombination}(P_s)$ ;
15:      $P_m \leftarrow \text{Mutation}(P_r)$ ;
16:      $P \leftarrow P_m$ ;
17:   until Maximum Number of Generations Reached
18: end procedure
```

---

The evaluation process ends (Algorithm 1 line 12 and Algorithm 2 line 12) after  $k$  evaluations are completed. The parameter  $k$  must be of the same order of the number of new individuals added to the population through recombination and mutation. Accordingly, for on-line NEAT, a generational algorithm,  $k$  need to be around (possibly larger than) the population size [32], whereas for on-line rtNEAT, a steady-state algorithm,  $k$  will be small since only one new individual in the population is added during each generation. For instance, in the experiments reported in this paper  $k$  is 300 for on-line NEAT while it is set to 5 for on-line rtNEAT. Note that this termination criteria does not guarantee that all the individuals in the population will be evaluated at least once [32, 10].

$\varepsilon$ -**Greedy-Improved** is an extension of  $\varepsilon$ -Greedy which we devised to avoid some of the limitations of the previous strategy. Like  $\varepsilon$ -Greedy, this strategy selects the best individual with probability  $(1 - \varepsilon)$  while with probability  $\varepsilon$  it selects a random individual *among the ones that have not been evaluated yet*. However, in this case, the best individual is eligible for selection only if its fitness is higher than a threshold  $\theta_{best}$ . The individuals evaluation process ends when (i) *all the individuals have been evaluated at least once* and (ii) the best individual of the current population has been evaluated for at least  $\theta_{eval}$  or it is not good enough (i.e., its fitness is below a threshold  $\theta_{best}$ ). This termination condition guarantees that all the individuals of the population have been evaluated and that the fitness of the champion is very accurate.

The  $\varepsilon$ -Greedy-Improved strategy is reported as Algorithm 4. At first, line 3, the fitness of the

---

**Algorithm 2** Steady-State On-line Evolution.

---

```
1: procedure EVOLUTION( $P$ )
     $\triangleright P$  is the population,  $p$  is an individual of  $P$ 
     $\triangleright e(p)$  is the number of evaluations of  $p$ 
     $\triangleright f(p)$  is the fitness of  $p$ 
     $\triangleright$  Randomly initialize  $P$ 
     $\triangleright$  Init fitness and evaluation count in  $P$ 
2:   Init( $P$ );
3:   for  $p \in P$  do
4:      $e(p) = 0$ 
5:      $f(p) = f_{min}$   $\triangleright$  Set to min fitness
6:   end for
7:   repeat
8:     repeat  $\triangleright$  Select the individual to evaluate
9:        $p \leftarrow \text{SELECT}(P)$ ;  $\triangleright$  Update the evaluation count
10:       $e(p) \leftarrow e(p) + 1$ ;  $\triangleright$  Update the fitness of based on
 $\triangleright$  its current evaluation EVAL( $p$ )
11:       $f(p) \leftarrow f(p) + \frac{1}{e(p)}(\text{EVAL}(p) - f(p))$ ;
12:    until Evaluation Termination Criteria Met;  $\triangleright$  Usual rtNEAT operators
13:    Remove the Worst Individual from  $P$ 
14:    Select two parents
15:    Apply recombination and mutation to generate  $p$   $\triangleright$  Init the count and the fitness estimate
16:     $e(p) \leftarrow 0$ 
17:     $f(p) \leftarrow f_{min}$ 
18:    Add  $p$  to  $P$ 
19:  until Maximum Number of Generations Reached
20: end procedure
```

---

best individual in the population is compared to the target threshold. If its fitness is too low, an individual from the population is randomly selected (note that, such an individual exists since otherwise, the termination condition would have been met at the previous iteration so that the evaluation process would have been terminated). Then, if all the individuals have been evaluated but the best has not been evaluated enough, the best is reevaluated (the process continues in its exploitation of the current result). Otherwise, if the best individual is eligible (its fitness is high enough) and other candidates have not been evaluated, the usual  $\varepsilon$ -Greedy selection is applied (line 9).

**Softmax** is a probabilistic action-selection strategy widely used in reinforcement learning. When applied to on-line evolution [32], it computes the selection probability of an individual  $p$  by using the Boltzmann distribution [28, 32] as follows,

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

---

**Algorithm 3**  $\varepsilon$ -Greedy selection strategy.

---

```
1: procedure SELECT( $P$ ) ▷  $P$  is the population
2:   if ( $rand() < \varepsilon$ ) then
3:     return  $p = \text{random}(P)$ ; ▷ Select a random individual from  $P$ 
4:   else
5:     return  $\text{argmax}_{p \in P} f(p)$ ; ▷ Get the fittest individual in  $P$ 
6:   end if
7:   return  $p$ ;
8: end procedure
```

---

where  $f(p)$  is the fitness of  $p$ ,  $P$  is the population and  $\tau$  is a parameter to control the balance between exploration and exploitation which is set empirically [32]. A low value of  $\tau$  favors exploitation, since small differences in the fitness of individuals will correspond to large differences in their selection probabilities; a high value of  $\tau$  favors exploration, since even large differences in the individual fitnesses will correspond to small differences in their selection probabilities. The strategy is reported as Algorithm 5. Initially, all the individuals in the population are evaluated once (Algorithm 5, line 2). Then, the fitness of individuals is used to compute their selection probability according to the Boltzmann distribution (Algorithm 5, line 5). At the end, an individual is selected based on these probabilities (Algorithm 5, line 7). The evaluation process terminates after  $k$  evaluations are completed [32]. The parameter  $k$  is set using the same criterion used for  $\varepsilon$ -Greedy.

**Interval-based.** None of the previous strategies takes into account the uncertainty affecting the current estimation of the fitness of the individuals. Interval-based selection addresses this issue by introducing confidence intervals (see Algorithm 6). At first, all the newly created individuals are evaluated, then for the remaining iterations the most promising individual (the one with the highest upper bound of its confidence interval) is returned (Algorithm 6, line 5). As for  $\varepsilon$ -Greedy and Softmax the evaluation process stops after  $k$  iterations. Also in this case, the parameter  $k$  is set using the same criterion used for  $\varepsilon$ -Greedy.

## 5 Related Work

In this section, we provide a brief overview of the most relevant published works related to this study.

### 5.1 On-line Neuroevolution for TORCS

We firstly presented our approach in [7, 6] where we also reported some initial results for the learning performance of on-line NEAT [32] on two tracks (Aalborg and Wheel 1 [1]) using  $\varepsilon$ -Greedy-Improved and Softmax. In this paper, we extend our previous analysis and consider also (i) on-line rtNEAT [23], which was separately introduced at the same time our approach was initially developed [7, 6], and (ii) other two evaluation selection strategies,  $\varepsilon$ -Greedy and Interval-based, which we did not consider before. In our initial analysis [7, 6], we focused only on the overall learning performance and the results were not statistically analyzed. Here, we compare all the approaches and statistically analyze the results. In particular, we take into account both the learning speed, the learning performance and we also test the performance of the best evolved individuals to check how many of them are actually *complete drivers* capable of driving the whole

---

**Algorithm 4**  $\varepsilon$ -Greedy-Improved selection strategy.

---

```
1: procedure SELECT( $P$ )                                ▷  $P$  is the population,  $p$  is an individual of  $P$ 
                                                         ▷  $e(p)$  is the number of evaluations of  $p$ 
                                                         ▷  $f(p)$  is the fitness of  $p$ 
2:    $best = \operatorname{argmax}_{p \in P} f(p)$ ;
                                                         ▷ If the best individual is not good enough
                                                         ▷ return an individual never
                                                         ▷ evaluated (explore)
3:   if ( $f(best) \leq \theta_{best}$ ) then
4:     return  $p | e(p) = 0$ ;
5:   end if
                                                         ▷ If all the individuals have been evaluated
                                                         ▷ return the best individual (exploit)
6:   if ( $\nexists p \in P | e(p) = 0$ )  $\wedge$  ( $e(best) < \theta_{eval}$ ) then
7:     return  $best$ ;
8:   end if
                                                         ▷ Otherwise apply the  $\varepsilon$ -greedy strategy
9:   if ( $\operatorname{rand}() < \varepsilon$ ) then
10:    return  $p | e(p) = 0$ ;
                                                         ▷ Return an individual never evaluated (explore)
11:  else
12:    return  $best$ ;
                                                         ▷ Return the best individual (exploit)
13:  end if
14: end procedure
```

---

tracks. In addition, we analyze adaptation of individuals and their generalization capabilities to unknown difficult tracks.

## 5.2 On-line Neuroevolution

Our approach shares some similarities with the work of Reeder et al. [23] who extended the approach of [32] to steady-state neuroevolution (i.e., rtNEAT [25]). However, there are some relevant differences between [23] and our approach. Firstly, [23] considers a continual learning problem in that there is no there is no concept of problem instance. Secondly, the problem in [23] is also simpler than simulated car racing since subsequent evaluations are totally independent whereas in our case, they are strongly correlated so that the outcome of previous individual can dramatically influence the current evaluation (e.g., if the previous driver crashed the car offroad, the next individual starts its evaluation from a very unfavorable position). Thirdly, in [23], the game has no exceptional events which can jeopardize the outcome of subsequent evaluations which are actually present in our case (e.g., a driver can get stuck in a corner). Moreover, although in [23] individuals are evaluated on fixed timeslots (as in our approach), the fitness is not computed by averaging the outcomes of repeated evaluations, but updates by adding positive and negative rewards achieved during each evaluation. In addition, Reeder et al. [23] report only the performance during learning, which measures the contribution of more individuals, while the performance of the best individuals alone is not tested. Accordingly, it is not clear whether/how many complete players are evolved in [23]. Furthermore, they only consider  $\varepsilon$ -Greedy (the simplest evaluation strategy),

---

**Algorithm 5** Softmax selection strategy.

---

```
1: procedure SELECT( $P$ )
     $\triangleright P$  is the population,  $p$  is an individual of  $P$ 
     $\triangleright e(p)$  is the number of evaluations of  $p$ 
     $\triangleright f(p)$  is the fitness of  $p$ 
     $\triangleright$  Evaluate all individuals at least once

2:   if  $\exists p \in P | e(p) = 0$  then
3:     return  $p$ ;
4:   else
5:      $total = \sum_{p \in P} e^{-\frac{f(p)}{\tau}}$ 
6:     for all  $p \in P$  do
7:       if  $rand() < \frac{e^{f(p)/\tau}}{total}$  then
8:         return  $p$ ;
9:       else
10:         $total = total - e^{-\frac{f(p)}{\tau}}$ ;
11:      end if
12:    end for
13:  end if
14: end procedure
```

---

while we consider and compare all the three strategies introduced in [32] (i.e.,  $\varepsilon$ -Greedy, Softmax, and Interval-based) and the  $\varepsilon$ -Greedy-Improved strategy introduced in [7, 6]. Reeder et al. [23] also do not deal with the issues of generalization and adaptation, considered in this paper. Finally, the work of Reeder et al., does not include any comparison with the original approach of Whiteson and Stone [32].

### 5.3 On-Line Learning and Evolutionary Computation

On-line neuroevolution [32, 33] is related to Learning Classifier Systems (LCSs) [12, 15], probably the most well-known evolutionary computation approach to on-line learning. These are on-line evolutionary rule-based systems which can solve both supervised (classification) problems and reinforcement learning problems. As any other reinforcement learning method, LCSs need to balance exploration and exploitation and they do it by using the typical action-selection mechanisms [28] — the same used in on-line neuroevolution to select individuals for reevaluation [32, 33].

On-line neuroevolution usually reevaluates an individual several times and combines all the evaluations in one fitness value. A similar issue arises when evolutionary methods face problems involving a noisy fitness function. Stagge [24] proposed a mechanism to select which individuals need more evaluations under the assumption that the fitness is affected by a Gaussian noise. More recently, Beielstein and Markon [4] exploited a similar mechanism to choose the individuals in the population to be replaced. Note however that, these works have a different focus in that they aim at solving off-line optimization problems while minimizing the number of evaluations. In contrast, on-line neuroevolution aims at solving on-line learning problems while maximizing the performance during learning. Finally, although the trade-off between exploration and exploitation is quite a novel problem in the evolutionary computation research, it has been extensively studied in the reinforcement learning literature (e.g., [31, 28]) and in the literature related to the multi-armed bandit problems (e.g., [2, 18]).

---

**Algorithm 6** Interval-Based selection strategy.

---

1: **procedure** SELECT( $P$ )

- ▷  $P$  is the population,  $p$  is an individual of  $P$
- ▷  $e(p)$  is the number of evaluations of  $p$
- ▷  $f(p)$  is the fitness of  $p$
- ▷ Initially, all the individuals are evaluated

2:   **if**  $\exists p \in P | e(p) = 0$  **then**

3:     **return**  $p$ ;

4:   **else**

- ▷ Adjust the fitness according to
- ▷ an  $\alpha$  confidence interval

5:     **return**  $\operatorname{argmax}_{p \in P} \left[ f(p) + z_{(\frac{1+\alpha}{2})} \cdot \frac{\sigma(p)}{\sqrt{e(p)}} \right]$ ;

6:   **end if**

7: **end procedure**

---

## 5.4 On-Line Learning and Games

In the recent years, several researchers applied on-line learning either (i) to develop innovative games [25] or (ii) to improve the players' game experience [32, 33, 21]. Among the others, the NERO project [25] is probably the most relevant example of on-line learning applied to games in which the goal is to train a team of agents to solve navigation and combat tasks through on-line evolution. In the development of NERO, Stanley et al. introduced rtNEAT [25], a *real-time neuroevolution* approach, and applied it to train a team of agent *during* the game on the tasks defined by the players. Recently, rtNEAT has been applied also to a real-time strategy game [19] and to evolve a team of ghosts in Pac-Man [34]. Both these works exploited rtNEAT to adapt on-line the difficulty level of the game to the skill of the player.

Although on-line neuroevolution and real-time neuroevolution (rtNEAT) are strictly related, they differ in several respects. Firstly, real-time neuroevolution deals with a population of agents, evaluated in parallel, while on-line neuroevolution is devised for a single agent. Secondly, in real-time neuroevolution all the individuals are evaluated only once using the same procedure (e.g., one timeslot); in contrast, in on-line neuroevolution most promising individuals tend to be evaluated more often. Finally, real-time neuroevolution and on-line neuroevolution have different goals. The former aims at converging to a good team behavior quickly (*in real-time*) whereas the latter aims at finding the best performing individual during the whole learning process.

Beside neuroevolution, Bakkes et al. [3] applied an evolutionary algorithm to learn on-line a team-strategy for the Capture of the Flag (CTF), a team-oriented game type of Quake III (a popular 3D First Person Shooter). In the racing game domain, Tan et al. [29] applied evolutionary strategies to adapt on-line the opponent's skills in the 2007 IEEE CEC Simulated Car Racing Competition [30]. Finally, Priesterjahn et al. [22] first learned the behavior of a bot in Quake III by imitation, then applied an evolutionary rule-based system to optimize it during the game.

## 6 Learning to Drive in TORCS using On-Line Neuroevolution

The application of off-line neuroevolution to TORCS is straightforward [9]: the population consists of candidate drivers (i.e., networks) whose fitness is computed as their performance on one problem instance which corresponds to driving for one or more laps; most important, all the evaluations are independent and they potentially can be carried out in parallel.

In contrast, the use of on-line neuroevolution in TORCS poses several challenges since there is just one car racing on a track and a population of networks competing to gain control of that one car. This scenario demands for reasonable performance in a limited amount of time (i.e., game ticks) and makes the evaluation of each network using several laps infeasible. In fact, it would be unacceptable to allow networks with poor driving capabilities to control the only car for too long. As a consequence, the evaluations of candidate drivers have to be carried out using rather short time slots. This approach however opens up several issues since (i) each candidate network is tested on rather brief sections of the track, (ii) the evaluation of an individual intrinsically depends on the results of the evaluation of the previous individuals; (iii) there might be several abrupt and unsafe changes in the car driving behavior when a controller replaces the previous one in any position of the track (in fact, different controllers might have totally different ways to deal with the same situation, leading to chopped driving behaviors); finally, (iv) extremely poor controllers might stop the whole evaluation process (for instance, if the car get stuck somewhere, all the subsequent controllers will perform poorly and possibly leading to a premature convergence).

Since evaluations cover only brief portions of the track (because of the short time slots), they might easily lead to either underestimate or overestimate the performance of candidate drivers. For instance, a network might have a high fitness, just because it has been evaluated only on favorable parts of the track (e.g., straight stretches), while performing poorly on the entire track. The on-line neuroevolution approaches applied in this work (see Section 4) aim at reducing the noise and the uncertainty due to the evaluation process (i) by reevaluating the most promising candidates more than once and (ii) by averaging the results of more evaluations into a unique fitness value.

Most important, in the on-line scenario, the evaluation of a candidate driver begins where the evaluation of the previous one ended. Thus, an evaluation intrinsically depends on the results of the previous ones; for instance, a candidate network might be evaluated starting in a unfavorable position of the track just because the previous drivers performed poorly. Accordingly, its evaluation might be worse than it should be just because of what happened during the earlier evaluations. To limit the dependency from previous evaluations, we introduced a sort of *warm up* procedure to separate subsequent evaluations. The warm up lasts for about the 10% of the time slot allowed for the evaluation of an individual and allow for a smooth transition between the previous evaluated network and the network currently evaluated. During warm up the behavior of the car is determined by the output of a weighted sum of both the previously evaluated network and the next network to be evaluated. At the beginning of warm up, the previous network weights more (i.e., the car behavior is more similar to the one just tested). As the warm up time passes, the weight of the evaluated network decreases and the contribution of the network to be evaluated increases. At the end of the warm up period the car is in full control of the new network and the actual evaluation can start. Overall, the warm up procedure reduces the dependency between subsequent evaluations and, by averaging the behaviors of subsequent controllers, it also reduces the abrupt and unsafe changes in the driving policy introduced by the subsequent use of different networks leading to nice and fluid driving behaviors.

Finally, an evaluation of a poor controller might jeopardize the entire learning process. For instance, the car might get stuck somewhere offtrack so that the subsequent candidate drivers would receive low fitnesses possibly leading to a premature convergence. To avoid this issue, we introduced a recovery procedure that checks, at the end of each evaluation, whether the car is lying outside the track. If this is the case, a programmed recovery policy (similar to the one available in the drivers distributed with TORCS) is activated and the car is brought back inside the track limits. Then, the evaluation of the next individual can start.

## 7 Design of Experiments

All the experiments presented in this paper have been carried out with TORCS 1.3.1, using the setup of the 2009 Simulated Car Racing Championship [16], and the version 1.0 of the NEAT/rtNEAT package distributed by the NEAT user group<sup>4</sup>, modified to compile with gcc-4.1.2.

**Sensors and Actuators.** We used the sensors provided by the competition software to build a sensor model consisting of (i) six range finder sensors to perceive the track edges along several directions (i.e.,  $-90^\circ$ ,  $-60^\circ$ ,  $-30^\circ$ ,  $+30^\circ$ ,  $+60^\circ$ ,  $+90^\circ$ ); (ii) an aggregate sensor to perceive the track edges *in front of the car*, which combines the readings of the three range finders along the direction  $-10^\circ$ ,  $0^\circ$  and  $+10^\circ$ ; and (iii) the current speed of the car. The effectors provided in the competition software were mapped into two real-valued effectors that control the steering wheel and the gas/brake pedals. In addition, when the car frontal sensor does not perceive the track edge within 100 meters (i.e., when the car is probably on a straight stretch), the gas pedal is set to the maximum value by default. Therefore, drivers need to control the gas/brake pedals only when facing a turn. This design forces controllers to drive as fast as possible from the very early generations and prevents the evolutionary search from wasting resources on reliable but slow controllers. Note that, drivers do not control the gear shift which, in this work, is controlled by a programmed policy taken from one of the drivers distributed with TORCS.

**Experimental Setup.** An experiment consists of ten runs. In each run, a driver is evolved using either the on-line version of NEAT or the on-line version of rtNEAT and one of the four evaluation-selection policies available (i.e.,  $\epsilon$ -Greedy,  $\epsilon$ -Greedy-Improved, Softmax, and Interval-based).

Each run lasts for 2000 laps (i.e., the car must complete 2000 laps before the evolutionary process stops) and involves a population of 100 controllers. An evaluation slot consists of 1000 game ticks (approximately 20 seconds of actual play time) while the warm up period lasts for 100 game ticks (around two seconds of actual play time). Accordingly, a network is evaluated for 20 seconds and then after a two seconds transition the control is passed to the next network to be evaluated.

Note that, our timeslot is very short, actually shorter than the time needed to complete one lap on the simplest track (A-Speedway). In fact, the fastest driver we evolved in A-Speedway requires around 30 seconds (Section 8) to complete a lap so that our evaluation slot is only the 66% of the fastest lap. In a more difficult track like Ruudskogen this difference is more dramatic where, in the initial learning stage, the evaluation timeslot roughly corresponds to the 5% of the average lap time (Table 1a).

The fitness of a controller during an evaluation slot is computed as,

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

where  $T_{out}$  is the number of game ticks the car is outside the track;  $\bar{s}$  is the average speed during the evaluation, computed as meters for game tick;  $d$  is the distance raced by the car during the evaluation, computed as meters run;  $\eta$  and  $\beta$  are two constants introduced to guarantee that the fitness is positive and as a scaling factor for the average speed term (both  $\eta$  and  $\beta$  have been empirically set to 1000 in all the experiment reported here). All the NEAT/rtNEAT parameters were set to their default values, taken in the configuration files distributed with the software package, except for a few ones.<sup>5</sup> For  $\epsilon$ -Greedy, Softmax, and Interval-based,  $k$  is set to 300 for on-line NEAT

---

<sup>4</sup><http://www.cs.ucf.edu/~kstanley/neat.html#group>

<sup>5</sup>With respect to the default settings we modified the following parameters: `weigh_mut_power=0.1`; `recur_prob=0.05` (i.e., we allowed recurrent connections); `mutdiff_coeff=1.0`; `compat_thresh=1.0`; `mutate_add_node_prob=0.005`; `mutate_add_link_prob=0.005`; and `interspecies_mate_rate=0.01`.

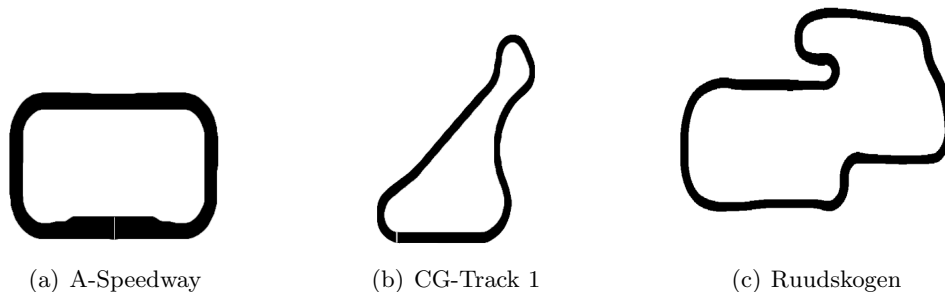


Figure 1: The three tracks used for the experiments reported in this paper.

and to 5 for on-line rtNEAT following the indications given in [32]. The other parameters were set to values which we empirically determined. In particular, for  $\varepsilon$ -Greedy and  $\varepsilon$ -Greedy-Improved,  $\varepsilon$  is set to 0.25; for  $\varepsilon$ -Greedy-Improved,  $\theta_{eval}$  is 5 (the best individual should have been evaluated at least five times) and  $\theta_{best}$  is 2100 (i.e., only individuals with a fitness higher than 2100 can be considered a champion); for Softmax,  $\tau$  is 50; for Interval-based, the confidence level  $\alpha$  has been set to 0.8. All the statistics reported in this paper are averages over ten runs.

**Statistical Analysis.** To analyze the results produced during each experiment, we performed a two-ways analysis of variance (ANOVA) followed by the typical post-hoc procedures (SNK, Tukey, Scheffe, and Bonferroni) [11]. The two-ways ANOVA has been applied to check whether there was some significant difference in the collected results with respect to (i) the type of neuroevolution applied (NEAT or rtNEAT), and to (ii) the evaluation strategy used ( $\varepsilon$ -Greedy,  $\varepsilon$ -Greedy-Improved, Softmax, or Interval-based). The post-hoc procedures have been applied to identify groups of configurations with similar performance.

## 8 Experimental Results

We applied on-line neuroevolution using NEAT and rtNEAT to evolve a driver for each one of the three tracks depicted in Figure 1. The tracks have increasing difficulty and they are all available in the TORCS distribution. A-Speedway is a simple oval with four *identical* left turns; a driver for this track, to be competitive, just needs to learn how to approach that one turn. CG-Track 1 mainly contains high-speed turns; it is not very difficult, but it requires complex trajectories to achieve low lap times. Ruudskogen is a rather complex and narrow track which contains both fast and slow turns, both on the left and on the right side; moreover the height of the track changes significantly making the control of the car more difficult when approaching some of the turns; accordingly, in third track, drivers must learn sophisticated behaviors for each part of the track to be competitive.

### 8.1 Learning from Scratch.

In the first set of experiments, we applied the on-line versions of NEAT and rtNEAT to learn a driver starting from a population initialized as usually done in NEAT and rtNEAT, i.e., using the simplest network topology possible (Section 3). Table 1 compares the performance of on-line NEAT/rtNEAT using one of the four selection strategies on the three tracks; statistics are averages over ten runs. Table 1a reports the average lap time over *the first 100 laps*; Table 1b reports the average lap time over *the first 500 laps*; Table 1c reports the average lap time over *the last 100*

laps; and Table 1d reports the average lap time of *the best individuals evolved* for each configuration during each run.

**Statistical Analysis.** For each track, we applied a *two-ways analysis of variance* (a two-ways ANOVA [11]) to test whether the differences in Table 1 are statistically significant *with respect to two factors*, the type of neuroevolution approach used (NEAT or rtNEAT) and the type of evaluation selection used ( $\epsilon$ -Greedy,  $\epsilon$ -Greedy-Improved, Softmax, and Interval-based). When the tests returned a statistical significance we also applied the typical post-hoc procedures (SNK, Tukey, Scheffe, and Bonferroni) to analyze the differences among the four selection strategies.

**Initial 100 Laps.** As can be noted from Table 1a, in the initial 100 laps, rtNEAT appears to learn faster than NEAT: in fact, the average lap times obtained by rtNEAT are usually lower than the corresponding values obtained by NEAT (apart from very few exceptions). The two-ways analysis of variance [11] of the data in Table 1a, for the averages over the first 100 laps, shows that the reported differences between NEAT and rtNEAT are statistically significant with a 100% confidence level in A-Speedway and with a 95% confidence level in CG-Track 1. However, the differences are not statistically significant in Ruudskogen. The same analysis shows that, with respect to the selection algorithms, there is no statistically significant difference between  $\epsilon$ -Greedy-Improved, Softmax, and Interval-based. Finally, the post-hoc procedures applied reports a clear distinction in A-Speedway between  $\epsilon$ -Greedy (which performs worse) and all the other three selection policies ( $\epsilon$ -Greedy-Improved, Softmax, and Interval-based).

**Initial 500 Laps.** As the learning proceeds, NEAT improves much more than rtNEAT so that the difference between their performances becomes blurred. The statistical analysis of the average lap times over the first 500 laps (Table 1b) shows that the difference between NEAT and rtNEAT is now statistically significant *only* in the most difficult track, Ruudskogen, with a confidence level of the 95%. This result might appear counterintuitive since, in A-Speedway and CG-Track 1, rtNEAT still performs better than NEAT for three out of the four selection algorithms. Note however that, in the same tracks, when  $\epsilon$ -Greedy is applied, rtNEAT performs much worse than NEAT so that, on the average, the differences between NEAT and rtNEAT are not statistically significant. With respect to the selection strategies, the data in Table 1b confirm and extend what previously found: there is a significant difference between  $\epsilon$ -Greedy and the other three strategies in A-Speedway and now also in CG-Track 1.

**Last 100 Laps.** Table 1c reports the average lap times over the last 100 (learning) laps. The reported results show that, at the end, NEAT reaches a better performance than rtNEAT. The statistical analysis of the data in Table 1c shows that, on the average, the difference between NEAT and rtNEAT is statistically significant with a 99.9% confidence level. The difference between NEAT and rtNEAT is more evident in Ruudskogen, the most difficult track, where NEAT always performs significantly better. In contrast, in the easier tracks, A-Speedway and CG-Track 1, there is no clear winner: NEAT performs better than rtNEAT, when  $\epsilon$ -Greedy and Interval-based are applied; whereas rtNEAT performs better than NEAT, when  $\epsilon$ -Greedy-Improved and Softmax are applied. With respect to the selection strategies, the statistical analysis confirms the previous findings showing a statistically significant difference between  $\epsilon$ -Greedy and the other three strategies in *all the tracks*.

**Testing.** During learning, the whole population controls the only car available. Therefore, several networks selected from the population may be used within the same lap so that there is no guarantee that a complete driver has been evolved. Accordingly, at the end of the learning, we tested the best individuals, the champions, evolved during each run for each configuration to assess its final performance on the whole track. For this purpose, each champion drove two laps and we measured

Selection Strategy	Laps	A-Speedway		CG-Track 1		Ruudskogen	
		NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )
$\varepsilon$ -Greedy	100	98.8 $\pm$ 23.9	118.1 $\pm$ 36.7	234.5 $\pm$ 36.2	232.3 $\pm$ 38.4	467.9 $\pm$ 67.1	498.2 $\pm$ 87.1
$\varepsilon$ -Greedy-Improved	100	84.4 $\pm$ 9.0	67.4 $\pm$ 11.1	261.6 $\pm$ 25.9	228.9 $\pm$ 42.3	523.4 $\pm$ 58.7	478.4 $\pm$ 66.1
Softmax	100	84.6 $\pm$ 22.4	73.9 $\pm$ 8.3	216.5 $\pm$ 30.4	195.1 $\pm$ 46.9	426.4 $\pm$ 81.1	429.2 $\pm$ 65.4
Interval-based	100	89.2 $\pm$ 22.1	75.2 $\pm$ 11.7	227.9 $\pm$ 37.9	210.6 $\pm$ 23.8	479.3 $\pm$ 66.1	432.1 $\pm$ 73.7

(a)

Selection Strategy	Laps	A-Speedway		CG-Track 1		Ruudskogen	
		NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )
$\varepsilon$ -Greedy	500	68.0 $\pm$ 6.5	97.9 $\pm$ 32.1	161.4 $\pm$ 26.8	215.4 $\pm$ 39.6	318.0 $\pm$ 55.3	425.4 $\pm$ 116.8
$\varepsilon$ -Greedy-Improved	500	55.4 $\pm$ 3.4	47.2 $\pm$ 2.7	151.6 $\pm$ 13.8	126.6 $\pm$ 27.0	302.4 $\pm$ 51.3	365.1 $\pm$ 97.8
Softmax	500	56.7 $\pm$ 6.4	49.2 $\pm$ 3.8	127.4 $\pm$ 14.1	119.2 $\pm$ 12.8	263.5 $\pm$ 44.9	276.1 $\pm$ 68.5
Interval-based	500	58.7 $\pm$ 4.7	53.4 $\pm$ 8.5	141.9 $\pm$ 14.9	141.7 $\pm$ 27.7	306.2 $\pm$ 86.0	299.3 $\pm$ 56.0

(b)

Selection Strategy	A-Speedway		CG-Track 1		Ruudskogen	
	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )
$\varepsilon$ -Greedy	39.3 $\pm$ 4.7	72.3 $\pm$ 19.8	91.0 $\pm$ 14.7	175.5 $\pm$ 50.5	178.5 $\pm$ 34.4	284.2 $\pm$ 69.5
$\varepsilon$ -Greedy-Improved	37.2 $\pm$ 1.7	34.7 $\pm$ 1.1	75.1 $\pm$ 12.5	69.4 $\pm$ 5.0	117.1 $\pm$ 6.4	175.1 $\pm$ 79.8
Softmax	36.8 $\pm$ 5.0	35.3 $\pm$ 0.7	76.6 $\pm$ 10.4	75.6 $\pm$ 8.0	131.9 $\pm$ 8.9	156.6 $\pm$ 28.4
Interval-based	34.5 $\pm$ 4.7	35.0 $\pm$ 1.0	71.1 $\pm$ 7.5	73.5 $\pm$ 4.3	147.5 $\pm$ 22.8	163.2 $\pm$ 16.6

(c)

Selection Strategy	A-Speedway		CG-Track 1		Ruudskogen	
	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )
$\varepsilon$ -Greedy	31.9 $\pm$ 1.2	(*) 61.7 $\pm$ 22.3	61.9 $\pm$ 9.5	(*) 132.4 $\pm$ 42.4	(*) 131.1 $\pm$ 18.5	(*) 248.3 $\pm$ 96.1
$\varepsilon$ -Greedy-Improved	31.1 $\pm$ 0.4	30.5 $\pm$ 1.1	55.4 $\pm$ 7.9	56.4 $\pm$ 8.7	93.2 $\pm$ 10.6	(*) 103.7 $\pm$ 8.1
Softmax	30.6 $\pm$ 0.6	31.1 $\pm$ 0.6	56.6 $\pm$ 5.9	57.6 $\pm$ 9.5	96.8 $\pm$ 6.9	113.8 $\pm$ 16.0
Interval-based	30.9 $\pm$ 0.7	31.4 $\pm$ 1.2	59.8 $\pm$ 18.4	65.6 $\pm$ 18.2	(*) 125.9 $\pm$ 37.7	(*) 159.1 $\pm$ 84.7

(d)

Table 1: On-line NEAT and rtNEAT applied to TORCS: (a) average lap time after 100 and (b) after 500 laps; (c) final performance computed as the average lap time during the last 100 laps; (d) average performance of the best individuals in the final populations; statistics are averages over ten runs.

its performance as the time taken to complete the second lap. Table 1d reports the average lap times obtained by the best ten individuals, the ten champions, evolved for each configuration (one for each one of the ten runs); an “\*” indicates that one of the champions could not complete the two test laps, i.e., the champion was not able to drive on the entire track but only part of it. The results show that, although the drivers are evaluated only on small parts of the track, the vast majority of the best individuals can drive on the whole track reliably. This is not surprising as the best drivers are actually reevaluated several times and thus, overall, are very likely to be tested on the entire track. The results in Table 1d confirm and extend what found for the final 100 laps of the learning phase (Table 1c). The differences between NEAT and rtNEAT are still statistically significant and the  $\varepsilon$ -Greedy still performs significantly worse than the other three evaluation strategies. In this case however, NEAT turns out to be the best algorithm since its average lap time is always lower than the one achieved by rtNEAT. The only exception being the performance of  $\varepsilon$ -Greedy-Improved on A-Speedway which however per-se is not statistically significant.

## 8.2 Seeded Evolution

We repeated the same set of experiments starting from populations seeded with the best individuals evolved, for each track, in the previous set of experiments. The goal was to test whether on-line neuroevolution could exploit previous existing knowledge and possibly improve it so as to evolve a

better driver.

For each configuration, we applied on-line neuroevolution starting from a seeded population and let the car drive for 2000 laps. Table 2 reports, for each one of the three tracks, the average lap times of on-line NEAT and rtNEAT using one of the four selection strategies on the three tracks; statistics are averages over ten runs. More precisely, Table 2a reports the average lap time over *the first 100 laps*; Table 2b reports the average lap time over *the last 100 laps*; and Table 2c reports the average lap time of *the best individuals* evolved in each run.

All the tables evidence the same pattern. When starting from a population seeded with a champion evolved using a track that is simpler than the target track (e.g., using the A-Speedway champion to seed the evolution of a driver for Ruudskogen), the results are similar to those obtained in the previous set of experiments: (i) rtNEAT is initially better than NEAT, but then, at the end, NEAT becomes significantly better than rtNEAT; (ii)  $\varepsilon$ -Greedy is significantly worse than the other three evaluation strategies.

In contrast, when seeding the initial population with a champion of a track that is more challenging than the target track the results are different. Initially, opposite to what happened when learning from scratch, in several cases, rtNEAT performs slightly worse than NEAT (Table 2a) but then, as the learning proceeds, rtNEAT improves and reaching a slightly better performance than NEAT (Table 2b).

Interestingly however, no matter what the original seed was, the statistical analysis of the evolved solutions (see Table 2c) shows that there is no significant difference between NEAT and rtNEAT when the evolved champions are tested.

With respect to the evaluation strategies, in all the cases, at the beginning,  $\varepsilon$ -Greedy and  $\varepsilon$ -Greedy-Improved strategies perform significantly better than Softmax and Interval-based (Table 2a); while, at the end (Table 2b) and during testing (Table 2c), there is no overall statistically significant difference.

### 8.3 Generalization

At the end, we tested the champions evolved for each one of the three tracks (i.e., one champion for each one of the ten learning runs executed for each configuration on each track) on a large set of the most difficult tracks available in the TORCS distribution. For each track, we let each champion drive two laps and recorded its performance as the time to complete the second lap. Table 3 reports the average lap time for each champion on each track with the standard deviations and, between parentheses, the number of evolved champions that were able to complete the two laps.

As can be noted, NEAT always performs better than rtNEAT in that the average lap times of the champions evolved using NEAT are always lower than the corresponding average lap times of the champions evolved using rtNEAT (there are only very few exceptions which correspond to configuration where very few drivers were able to complete two laps). We performed a two-ways analysis of variance (ANOVA) on these data to test whether there was a statistically significant difference in the performance of the champions with respect to (i) the type of neuroevolution applied (NEAT or rtNEAT), and (ii) the evaluation strategy used ( $\varepsilon$ -Greedy,  $\varepsilon$ -Greedy-Improved, Softmax, or Interval-based). The results show that the reported difference is not significant for the champions evolved in the simplest track (A-Speedway) but it becomes significant with a confidence level of 100% for the champions evolved in CG-Track 1 and Ruudskogen. With respect to the evaluation strategies, the two-ways ANOVA confirms what found in the previous two sets of experiments showing that  $\varepsilon$ -Greedy performs significantly worse than the other three strategies. In addition, the data collected for the champions evolved in Ruudskogen show a statistically significant difference *among all the four selection strategies* suggesting that, in this case, the best driver is the one evolved

Training Track	Selection Strategy	Target Track for Adaptation					
		A-Speedway		CG-Track 1		Ruudskogen	
		NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )
A-Speedway	$\epsilon$ -Greedy			132.2 $\pm$ 12.3	134.6 $\pm$ 14.4	392.1 $\pm$ 86.5	460.8 $\pm$ 83.2
	$\epsilon$ -Greedy-Improved			163.3 $\pm$ 32.8	139.3 $\pm$ 28.7	432.9 $\pm$ 31.3	328.6 $\pm$ 52.9
	Softmax			133.1 $\pm$ 15.4	121.6 $\pm$ 13.9	356.9 $\pm$ 68.1	289.9 $\pm$ 44.1
	Interval-based			125.4 $\pm$ 11.2	122.1 $\pm$ 11.1	392.4 $\pm$ 64.0	298.8 $\pm$ 76.5
CG-Track 1	$\epsilon$ -Greedy	33.1 $\pm$ 0.7	32.9 $\pm$ 0.7			211.3 $\pm$ 18.8	198.4 $\pm$ 13.5
	$\epsilon$ -Greedy-Improved	33.1 $\pm$ 0.7	33.1 $\pm$ 0.9			276.4 $\pm$ 22.0	279.7 $\pm$ 31.7
	Softmax	35.9 $\pm$ 1.0	37.0 $\pm$ 0.9			215.3 $\pm$ 12.4	206.9 $\pm$ 18.4
	Interval-based	35.0 $\pm$ 0.6	35.6 $\pm$ 1.1			212.1 $\pm$ 6.7	201.6 $\pm$ 12.8
Ruudskogen	$\epsilon$ -Greedy	33.8 $\pm$ 1.1	33.3 $\pm$ 0.6	53.1 $\pm$ 2.0	54.4 $\pm$ 1.7		
	$\epsilon$ -Greedy-Improved	33.0 $\pm$ 0.7	32.6 $\pm$ 0.5	53.2 $\pm$ 1.5	53.2 $\pm$ 1.5		
	Softmax	35.9 $\pm$ 1.0	36.6 $\pm$ 0.9	57.5 $\pm$ 1.4	63.5 $\pm$ 1.2		
	Interval-based	35.8 $\pm$ 0.5	36.2 $\pm$ 0.8	56.9 $\pm$ 1.8	61.9 $\pm$ 2.0		

(a)

Training Track	Selection Strategy	Target Track for Adaptation					
		A-Speedway		CG-Track 1		Ruudskogen	
		NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )
A-Speedway	$\epsilon$ -Greedy			77.5 $\pm$ 10.0	105.7 $\pm$ 21.1	175.7 $\pm$ 24.5	253.4 $\pm$ 52.4
	$\epsilon$ -Greedy-Improved			72.6 $\pm$ 13.2	62.7 $\pm$ 2.6	113.6 $\pm$ 7.7	125.6 $\pm$ 10.2
	Softmax			69.5 $\pm$ 3.6	65.2 $\pm$ 5.6	124.5 $\pm$ 9.3	131.7 $\pm$ 9.3
	Interval-based			70.1 $\pm$ 7.3	69.0 $\pm$ 8.3	132.4 $\pm$ 15.9	132.0 $\pm$ 10.5
CG-Track 1	$\epsilon$ -Greedy	35.3 $\pm$ 1.7	33.8 $\pm$ 1.2			190.5 $\pm$ 61.8	169.8 $\pm$ 8.8
	$\epsilon$ -Greedy-Improved	35.0 $\pm$ 1.8	33.7 $\pm$ 1.0			112.8 $\pm$ 5.0	146.1 $\pm$ 37.5
	Softmax	38.8 $\pm$ 1.8	34.2 $\pm$ 1.0			139.5 $\pm$ 18.0	142.8 $\pm$ 9.6
	Interval-based	33.2 $\pm$ 3.1	34.2 $\pm$ 1.2			134.1 $\pm$ 11.8	136.5 $\pm$ 15.5
Ruudskogen	$\epsilon$ -Greedy	33.9 $\pm$ 1.2	32.8 $\pm$ 0.9	56.7 $\pm$ 3.3	53.1 $\pm$ 2.1		
	$\epsilon$ -Greedy-Improved	33.1 $\pm$ 1.3	33.2 $\pm$ 0.6	53.9 $\pm$ 1.8	53.3 $\pm$ 1.7		
	Softmax	31.7 $\pm$ 0.4	33.0 $\pm$ 1.1	60.1 $\pm$ 1.2	54.6 $\pm$ 1.3		
	Interval-based	35.0 $\pm$ 0.8	32.7 $\pm$ 0.5	58.4 $\pm$ 0.9	54.4 $\pm$ 1.4		

(b)

Training Track	Selection Strategy	Target Track for Adaptation					
		A-Speedway		CG-Track 1		Ruudskogen	
		NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )
A-Speedway	$\epsilon$ -Greedy			57.0 $\pm$ 8.1	78.7 $\pm$ 26.7	122.1 $\pm$ 24.7	188.0 $\pm$ 36.8
	$\epsilon$ -Greedy-Improved			51.4 $\pm$ 2.1	50.5 $\pm$ 2.3	88.1 $\pm$ 3.1	97.7 $\pm$ 8.0
	Softmax			53.2 $\pm$ 4.2	49.2 $\pm$ 1.7	91.3 $\pm$ 4.2	99.4 $\pm$ 4.8
	Interval-based			55.7 $\pm$ 11.2	85.6 $\pm$ 33.3	103.0 $\pm$ 12.0	128.4 $\pm$ 34.8
CG-Track 1	$\epsilon$ -Greedy	30.8 $\pm$ 0.7	31.4 $\pm$ 0.7			147.0 $\pm$ 72.3	139.5 $\pm$ 18.6
	$\epsilon$ -Greedy-Improved	30.9 $\pm$ 0.6	30.8 $\pm$ 0.5			92.7 $\pm$ 5.8	104.9 $\pm$ 11.0
	Softmax	30.4 $\pm$ 0.5	30.9 $\pm$ 0.6			102.1 $\pm$ 13.1	108.7 $\pm$ 7.7
	Interval-based	30.2 $\pm$ 0.4	31.3 $\pm$ 1.1			108.3 $\pm$ 21.8	134.6 $\pm$ 57.0
Ruudskogen	$\epsilon$ -Greedy	(*) 31.8 $\pm$ 1.5	(*) 30.9 $\pm$ 0.7	(*) 47.6 $\pm$ 1.1	(*) 47.5 $\pm$ 1.0		
	$\epsilon$ -Greedy-Improved	30.4 $\pm$ 0.3	30.6 $\pm$ 0.4	46.7 $\pm$ 0.7	46.2 $\pm$ 0.7		
	Softmax	(*) 30.6 $\pm$ 0.3	30.2 $\pm$ 0.5	46.2 $\pm$ 0.7	47.2 $\pm$ 1.4		
	Interval-based	30.0 $\pm$ 0.2	31.9 $\pm$ 3.8	(*) 51.2 $\pm$ 9.8	51.7 $\pm$ 13.5		

(c)

Table 2: On-line NEAT and rtNEAT applied from a seeded population: (a) average performance after 100 laps; (b) average performance of the last 100 laps; (c) average performance of the best individuals.

using  $\varepsilon$ -Greedy-Improved, followed by Softmax, Interval-based, and  $\varepsilon$ -Greedy.

We also analyzed the data for the number of champions that were able to complete the test process (reported between parentheses in Table 3) using a two-ways ANOVA. The analysis shows that the number of champions evolved using NEAT, that are capable of completing the test, is significantly higher (with a confidence level of 99%) than the number of champions evolved using rtNEAT, i.e., NEAT evolves more champions which can complete new unseen tracks. With respect to the evaluation strategies, the two-ways ANOVA again shows that  $\varepsilon$ -Greedy performs worse than the other three strategies. In addition, all the four post-hoc tests cluster the four evaluation strategies in three groups: one consisting of  $\varepsilon$ -Greedy (the worse one), one consisting of  $\varepsilon$ -Greedy-Improved and Softmax (the best ones), and one containing Interval-based (the middle one).

## 8.4 Discussion

Our experimental analysis suggests that, *overall*, on-line NEAT may be the best approach to evolve a driver for TORCS; it also suggests that  $\varepsilon$ -Greedy-Improved and Softmax may be the best evaluation strategies to be used. Indeed, when learning from scratch, rtNEAT *at the beginning* (Table 1a) performs significantly better than NEAT: since it uses steady-state evolution, rtNEAT explores the solution space less than NEAT (which is generational); accordingly, at the beginning, rtNEAT reaches a better performance. However, in the long run, NEAT catches up and at the end it performs significantly better than rtNEAT.

The experiments with seeded populations (Section 8.2) show that is possible to transfer previous knowledge (previous evolved drivers) to seed the evolution of drivers for different tracks. In particular, when using a driver evolved on a simple track as the seed for a more challenging track, the results are similar to the ones obtained when learning from scratch (rtNEAT is initially better, but then NEAT catches up). In contrast, when using a driver evolved on a challenging track as the seed for a simple track, the results show the opposite behavior: initially, rtNEAT performs worse and then both models reach a similar performance. The analysis of the populations suggests that, when starting from a more experienced driver, capable of driving in many different situations (e.g., on the Ruudskogen track), *at the beginning* (Table 2a), steady-state evolution tends to exploit the current knowledge rather than trying to recombine good building-blocks of the experienced behaviors already present in the initial population. Accordingly, NEAT tends to perform better since, by using a generational evolutionary algorithm, it tends to recombine the existing good behaviors more effectively, so as to adapt faster on the new track. In contrast, in rtNEAT this process is slower so that at the beginning the performance of rtNEAT is worse. However, at the end, both methods reach a comparable performance (in fact, there is no statistically significant difference at the end).

The final testing performed on many unknown tracks suggest that NEAT can generalize more, producing (significantly more) controllers capable of driving on difficult unknown tracks.

With respect to the evaluation strategies, all the experiments seem to agree.  $\varepsilon$ -Greedy is the worse evaluation strategy often leading to a significantly lower performance while the other three evaluation strategies usually perform similarly. Only in some cases, the statistical analysis reported a significant difference in the performance of  $\varepsilon$ -Greedy-Improved, Softmax and Interval-based, which resulted in a clustering between what appear to be the best techniques ( $\varepsilon$ -Greedy-Improved and Softmax) and Interval-based.

Test Track	Selection Strategy	Training Track					
		A-Speedway		CG-Track 1		Ruudskogen	
		NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )	NEAT ( $\mu \pm \sigma$ )	rtNEAT ( $\mu \pm \sigma$ )
alpine-1	$\epsilon$ -Greedy	195.7 $\pm$ 42.5 (4)	265.4 $\pm$ 93.9 (3)	202.0 $\pm$ 22.6 (7)	— (0)	312.8 $\pm$ 112.1 (6)	205.1 $\pm$ 16.4 (2)
	$\epsilon$ -Greedy-Improved	178.0 $\pm$ 13.9 (6)	204.2 $\pm$ 20.8 (3)	263.4 $\pm$ 104.7 (7)	243.2 $\pm$ 40.5 (4)	199.3 $\pm$ 16.7 (9)	243.1 $\pm$ 16.0 (9)
	Softmax	174.4 $\pm$ 17.4 (6)	178.7 $\pm$ 31.8 (4)	295.1 $\pm$ 103.4 (7)	282.1 $\pm$ 140.3 (8)	219.6 $\pm$ 17.6 (8)	270.9 $\pm$ 35.4 (6)
	Interval-based	251.4 $\pm$ 87.2 (5)	185.9 $\pm$ 43.4 (4)	261.4 $\pm$ 70.1 (7)	268.4 $\pm$ 36.7 (7)	248.8 $\pm$ 37.8 (7)	280.2 $\pm$ 64.0 (6)
alpine-2	$\epsilon$ -Greedy	245.3 $\pm$ 112.3 (3)	193.9 $\pm$ 0.0 (1)	174.2 $\pm$ 67.8 (7)	152.6 $\pm$ 17.6 (3)	138.7 $\pm$ 11.7 (9)	205.9 $\pm$ 102.6 (4)
	$\epsilon$ -Greedy-Improved	213.3 $\pm$ 110.4 (8)	184.9 $\pm$ 27.5 (3)	143.5 $\pm$ 17.5 (8)	150.4 $\pm$ 23.7 (7)	130.1 $\pm$ 17.5 (9)	142.2 $\pm$ 8.8 (7)
	Softmax	187.6 $\pm$ 62.2 (4)	149.4 $\pm$ 28.8 (3)	136.1 $\pm$ 6.1 (6)	128.3 $\pm$ 7.4 (6)	134.1 $\pm$ 9.0 (10)	159.4 $\pm$ 23.6 (7)
	Interval-based	177.8 $\pm$ 51.3 (5)	136.5 $\pm$ 15.8 (4)	136.3 $\pm$ 15.6 (6)	196.1 $\pm$ 46.1 (5)	137.9 $\pm$ 12.7 (5)	171.1 $\pm$ 57.1 (6)
e-track-1	$\epsilon$ -Greedy	209.1 $\pm$ 55.6 (4)	236.2 $\pm$ 57.4 (4)	213.8 $\pm$ 40.8 (6)	280.2 $\pm$ 75.0 (6)	195.0 $\pm$ 39.6 (9)	208.8 $\pm$ 37.2 (5)
	$\epsilon$ -Greedy-Improved	229.8 $\pm$ 35.9 (7)	263.8 $\pm$ 41.5 (4)	162.6 $\pm$ 32.1 (8)	186.2 $\pm$ 64.9 (6)	131.1 $\pm$ 53.6 (10)	125.1 $\pm$ 11.4 (6)
	Softmax	238.7 $\pm$ 60.4 (6)	251.2 $\pm$ 50.8 (5)	152.5 $\pm$ 28.8 (8)	160.1 $\pm$ 22.4 (8)	151.6 $\pm$ 36.0 (10)	192.0 $\pm$ 80.5 (8)
	Interval-based	206.2 $\pm$ 31.7 (5)	254.1 $\pm$ 76.2 (6)	158.8 $\pm$ 40.7 (8)	238.0 $\pm$ 75.6 (6)	203.6 $\pm$ 18.8 (4)	151.6 $\pm$ 30.6 (5)
e-track-2	$\epsilon$ -Greedy	266.3 $\pm$ 30.4 (2)	348.8 $\pm$ 19.7 (2)	242.0 $\pm$ 46.6 (6)	371.5 $\pm$ 0.0 (1)	207.6 $\pm$ 38.5 (9)	282.7 $\pm$ 27.1 (3)
	$\epsilon$ -Greedy-Improved	— (0)	284.6 $\pm$ 0.0 (1)	227.7 $\pm$ 46.6 (6)	239.3 $\pm$ 38.4 (7)	151.1 $\pm$ 60.1 (9)	198.0 $\pm$ 68.6 (9)
	Softmax	— (0)	324.1 $\pm$ 0.0 (1)	220.6 $\pm$ 21.3 (7)	254.9 $\pm$ 65.6 (5)	179.2 $\pm$ 56.0 (10)	209.0 $\pm$ 45.6 (7)
	Interval-based	328.8 $\pm$ 0.0 (1)	279.7 $\pm$ 50.7 (4)	179.4 $\pm$ 26.4 (7)	208.3 $\pm$ 25.0 (4)	155.7 $\pm$ 40.8 (7)	233.5 $\pm$ 59.8 (6)
e-track-3	$\epsilon$ -Greedy	222.6 $\pm$ 64.1 (5)	391.3 $\pm$ 129.4 (6)	210.9 $\pm$ 59.0 (8)	320.3 $\pm$ 153.3 (4)	202.0 $\pm$ 101.8 (10)	236.6 $\pm$ 88.3 (5)
	$\epsilon$ -Greedy-Improved	272.3 $\pm$ 85.0 (8)	273.2 $\pm$ 78.2 (6)	161.2 $\pm$ 22.3 (8)	155.9 $\pm$ 22.9 (9)	123.0 $\pm$ 16.7 (10)	138.7 $\pm$ 33.3 (6)
	Softmax	241.1 $\pm$ 47.5 (5)	219.7 $\pm$ 63.1 (6)	172.0 $\pm$ 30.0 (9)	185.1 $\pm$ 31.8 (10)	140.5 $\pm$ 18.3 (9)	175.0 $\pm$ 79.5 (9)
	Interval-based	223.6 $\pm$ 46.6 (6)	244.9 $\pm$ 102.4 (6)	183.4 $\pm$ 25.5 (7)	194.0 $\pm$ 47.4 (7)	151.4 $\pm$ 35.6 (8)	204.8 $\pm$ 100.3 (8)
e-track-4	$\epsilon$ -Greedy	331.7 $\pm$ 67.1 (6)	297.3 $\pm$ 42.8 (2)	276.9 $\pm$ 58.9 (10)	405.8 $\pm$ 126.5 (4)	247.9 $\pm$ 83.4 (10)	350.9 $\pm$ 132.4 (8)
	$\epsilon$ -Greedy-Improved	360.1 $\pm$ 50.4 (4)	320.6 $\pm$ 54.1 (4)	217.9 $\pm$ 79.3 (9)	199.3 $\pm$ 40.3 (10)	142.8 $\pm$ 17.7 (10)	184.9 $\pm$ 32.0 (9)
	Softmax	373.4 $\pm$ 101.7 (6)	334.2 $\pm$ 54.8 (6)	204.0 $\pm$ 46.8 (9)	227.2 $\pm$ 71.2 (10)	155.4 $\pm$ 16.5 (10)	191.6 $\pm$ 49.8 (10)
	Interval-based	319.1 $\pm$ 53.5 (3)	347.4 $\pm$ 83.8 (7)	259.0 $\pm$ 77.9 (10)	259.4 $\pm$ 96.9 (9)	188.9 $\pm$ 49.3 (8)	277.3 $\pm$ 166.8 (8)
e-track-6	$\epsilon$ -Greedy	262.6 $\pm$ 12.4 (2)	515.9 $\pm$ 0.0 (1)	243.9 $\pm$ 43.3 (5)	353.1 $\pm$ 120.5 (4)	206.6 $\pm$ 47.2 (9)	284.2 $\pm$ 56.1 (2)
	$\epsilon$ -Greedy-Improved	290.7 $\pm$ 0.0 (1)	225.2 $\pm$ 0.0 (1)	180.3 $\pm$ 26.1 (7)	240.8 $\pm$ 73.7 (10)	139.9 $\pm$ 23.9 (10)	179.7 $\pm$ 38.8 (9)
	Softmax	— (0)	320.5 $\pm$ 0.0 (1)	209.2 $\pm$ 89.8 (7)	290.1 $\pm$ 127.1 (8)	165.6 $\pm$ 25.6 (10)	183.7 $\pm$ 55.2 (8)
	Interval-based	— (0)	289.8 $\pm$ 114.3 (4)	213.0 $\pm$ 40.2 (7)	229.8 $\pm$ 62.6 (7)	198.9 $\pm$ 93.2 (7)	211.8 $\pm$ 63.2 (6)
forza	$\epsilon$ -Greedy	300.5 $\pm$ 102.7 (4)	423.7 $\pm$ 166.8 (4)	222.5 $\pm$ 22.8 (9)	335.7 $\pm$ 114.0 (8)	222.8 $\pm$ 88.4 (10)	348.1 $\pm$ 158.4 (5)
	$\epsilon$ -Greedy-Improved	374.0 $\pm$ 94.3 (6)	367.3 $\pm$ 95.3 (5)	231.0 $\pm$ 59.8 (9)	259.7 $\pm$ 134.6 (8)	141.3 $\pm$ 40.2 (10)	192.5 $\pm$ 62.9 (9)
	Softmax	305.8 $\pm$ 85.8 (8)	220.4 $\pm$ 43.3 (4)	185.4 $\pm$ 38.6 (9)	247.6 $\pm$ 81.8 (9)	158.0 $\pm$ 49.5 (10)	191.8 $\pm$ 45.0 (10)
	Interval-based	298.6 $\pm$ 46.5 (6)	243.8 $\pm$ 61.1 (5)	197.9 $\pm$ 52.6 (8)	285.7 $\pm$ 70.0 (6)	173.8 $\pm$ 36.9 (8)	279.9 $\pm$ 125.7 (9)
g-track-1	$\epsilon$ -Greedy	103.4 $\pm$ 37.8 (7)	134.7 $\pm$ 41.8 (5)	61.9 $\pm$ 9.5 (10)	132.4 $\pm$ 42.4 (8)	70.9 $\pm$ 18.2 (9)	103.0 $\pm$ 32.3 (7)
	$\epsilon$ -Greedy-Improved	115.5 $\pm$ 20.4 (9)	152.5 $\pm$ 54.7 (9)	55.4 $\pm$ 7.9 (10)	56.4 $\pm$ 8.7 (10)	50.8 $\pm$ 3.6 (10)	63.1 $\pm$ 9.5 (9)
	Softmax	112.7 $\pm$ 32.9 (9)	133.2 $\pm$ 41.9 (6)	56.6 $\pm$ 5.9 (10)	57.6 $\pm$ 9.5 (10)	57.8 $\pm$ 7.8 (10)	62.1 $\pm$ 5.4 (9)
	Interval-based	147.9 $\pm$ 45.5 (10)	135.9 $\pm$ 40.6 (7)	59.8 $\pm$ 18.4 (10)	65.6 $\pm$ 18.2 (10)	65.0 $\pm$ 10.9 (8)	72.0 $\pm$ 27.8 (7)
g-track-2	$\epsilon$ -Greedy	190.3 $\pm$ 101.7 (7)	262.8 $\pm$ 56.0 (4)	107.8 $\pm$ 19.2 (10)	243.3 $\pm$ 89.6 (8)	105.4 $\pm$ 26.5 (10)	131.9 $\pm$ 33.8 (8)
	$\epsilon$ -Greedy-Improved	135.5 $\pm$ 41.8 (7)	236.7 $\pm$ 66.1 (9)	88.6 $\pm$ 16.5 (10)	89.8 $\pm$ 11.5 (10)	72.3 $\pm$ 4.3 (10)	85.9 $\pm$ 8.6 (9)
	Softmax	147.6 $\pm$ 39.1 (9)	171.1 $\pm$ 92.0 (5)	91.5 $\pm$ 17.6 (10)	103.7 $\pm$ 27.7 (10)	85.7 $\pm$ 15.0 (10)	100.2 $\pm$ 29.4 (10)
	Interval-based	251.9 $\pm$ 57.7 (6)	175.1 $\pm$ 55.9 (6)	101.3 $\pm$ 23.6 (10)	118.3 $\pm$ 53.2 (10)	92.3 $\pm$ 15.0 (9)	99.8 $\pm$ 19.1 (7)
g-track-3	$\epsilon$ -Greedy	261.8 $\pm$ 42.8 (4)	233.2 $\pm$ 54.0 (4)	138.8 $\pm$ 40.7 (7)	242.7 $\pm$ 74.4 (6)	140.6 $\pm$ 24.2 (9)	198.5 $\pm$ 22.7 (4)
	$\epsilon$ -Greedy-Improved	196.0 $\pm$ 49.0 (7)	248.9 $\pm$ 33.4 (4)	146.6 $\pm$ 44.6 (9)	144.5 $\pm$ 18.9 (8)	95.8 $\pm$ 6.1 (10)	116.8 $\pm$ 9.3 (8)
	Softmax	258.8 $\pm$ 53.9 (5)	197.2 $\pm$ 31.0 (4)	143.6 $\pm$ 49.4 (8)	169.6 $\pm$ 42.4 (8)	118.1 $\pm$ 25.0 (10)	127.4 $\pm$ 17.2 (9)
	Interval-based	250.4 $\pm$ 49.2 (5)	215.2 $\pm$ 81.4 (6)	140.8 $\pm$ 20.4 (7)	179.2 $\pm$ 40.8 (6)	146.8 $\pm$ 39.4 (8)	149.0 $\pm$ 23.3 (6)
ole-road-1	$\epsilon$ -Greedy	490.7 $\pm$ 151.5 (5)	448.7 $\pm$ 138.0 (4)	277.1 $\pm$ 60.1 (9)	378.8 $\pm$ 112.1 (5)	252.1 $\pm$ 51.5 (9)	422.7 $\pm$ 144.0 (6)
	$\epsilon$ -Greedy-Improved	380.4 $\pm$ 58.1 (7)	427.9 $\pm$ 85.7 (6)	240.9 $\pm$ 50.3 (8)	248.8 $\pm$ 19.0 (8)	172.3 $\pm$ 14.8 (10)	200.2 $\pm$ 21.6 (8)
	Softmax	360.7 $\pm$ 102.0 (8)	363.3 $\pm$ 92.0 (4)	229.3 $\pm$ 30.4 (8)	280.3 $\pm$ 60.0 (8)	189.9 $\pm$ 23.2 (10)	220.4 $\pm$ 31.1 (10)
	Interval-based	377.7 $\pm$ 66.2 (6)	329.5 $\pm$ 144.0 (5)	229.7 $\pm$ 25.6 (7)	330.8 $\pm$ 125.6 (6)	282.5 $\pm$ 92.4 (9)	297.3 $\pm$ 117.4 (7)
ruudskogen	$\epsilon$ -Greedy	236.2 $\pm$ 76.8 (6)	278.3 $\pm$ 107.7 (2)	156.2 $\pm$ 24.0 (9)	227.2 $\pm$ 46.4 (5)	131.1 $\pm$ 18.5 (9)	248.3 $\pm$ 96.1 (8)
	$\epsilon$ -Greedy-Improved	272.5 $\pm$ 70.0 (8)	239.9 $\pm$ 0.0 (1)	149.4 $\pm$ 29.2 (10)	178.8 $\pm$ 81.7 (10)	93.2 $\pm$ 10.6 (10)	103.7 $\pm$ 8.1 (9)
	Softmax	261.1 $\pm$ 53.2 (6)	214.4 $\pm$ 48.1 (5)	128.9 $\pm$ 14.3 (8)	143.3 $\pm$ 32.0 (7)	96.8 $\pm$ 6.9 (10)	113.8 $\pm$ 16.0 (10)
	Interval-based	259.4 $\pm$ 49.0 (6)	207.3 $\pm$ 81.2 (4)	131.3 $\pm$ 15.9 (6)	215.6 $\pm$ 82.9 (7)	125.9 $\pm$ 37.7 (9)	159.1 $\pm$ 84.7 (8)
wheel-1	$\epsilon$ -Greedy	209.7 $\pm$ 71.9 (7)	311.6 $\pm$ 114.2 (6)	185.5 $\pm$ 38.9 (10)	318.0 $\pm$ 115.4 (7)	163.9 $\pm$ 47.8 (10)	199.7 $\pm$ 102.6 (7)
	$\epsilon$ -Greedy-Improved	208.7 $\pm$ 49.7 (8)	274.6 $\pm$ 61.8 (8)	147.8 $\pm$ 33.3 (9)	160.1 $\pm$ 16.5 (9)	109.4 $\pm$ 7.9 (9)	136.1 $\pm$ 9.5 (8)
	Softmax	208.6 $\pm$ 41.2 (9)	231.1 $\pm$ 92.1 (5)	157.0 $\pm$ 27.6 (9)	147.9 $\pm$ 12.3 (9)	127.5 $\pm$ 17.0 (10)	153.6 $\pm$ 28.8 (9)
	Interval-based	207.0 $\pm$ 61.5 (8)	213.2 $\pm$ 74.7 (7)	147.2 $\pm$ 11.8 (8)	184.8 $\pm$ 87.5 (8)	145.1 $\pm$ 28.8 (8)	156.7 $\pm$ 52.9 (6)
wheel-2	$\epsilon$ -Greedy	409.0 $\pm$ 71.8 (6)	514.5 $\pm$ 147.8 (3)	273.2 $\pm$ 43.3 (9)	364.9 $\pm$ 111.5 (4)	277.0 $\pm$ 95.3 (8)	355.4 $\pm$ 144.3 (6)
	$\epsilon$ -Greedy-Improved	404.7 $\pm$ 79.2 (6)	477.4 $\pm$ 113.7 (4)	225.2 $\pm$ 41.2 (9)	251.2 $\pm$ 57.0 (9)	166.0 $\pm$ 18.1 (10)	270.0 $\pm$ 143.2 (9)
	Softmax	431.9 $\pm$ 103.1 (8)	329.5 $\pm$ 80.9 (4)	235.8 $\pm$ 36.1 (9)	293.6 $\pm$ 111.1 (9)	197.6 $\pm$ 28.3 (9)	285.6 $\pm$ 120.2 (8)
	Interval-based	320.5 $\pm$ 51.0 (5)	388.8 $\pm$ 169.7 (5)	272.8 $\pm$ 28.8 (8)	355.7 $\pm$ 128.7 (7)	228.0 $\pm$ 52.7 (9)	276.6 $\pm$ 132.0 (6)

Table 3: Testing the champions of each track on other unseen tracks.

## 9 Conclusions

We applied on-line neuroevolution to learn drivers for The Open Racing Car Simulator (TORCS) [1]. We focused on the two major on-line neuroevolution approaches (on-line NEAT [32] and on-line rtNEAT [23]) and on the four evaluation selection strategies introduced in the literature ( $\varepsilon$ -Greedy, Softmax, Interval-based [32] and  $\varepsilon$ -Greedy-Improved [7]). We considered eight methods (each one combining one neuroevolution approach with an evaluation strategy) and performed three sets of experiments to compare the methods in terms of learning capabilities, adaptation, and generalization.

In the first set of experiments, we applied the eight methods to evolve a driver for three different tracks of increasing difficulty and compared their performance at the beginning of the learning phase and at the end of it. When learning to drive from scratch, rtNEAT initially performs significantly better than NEAT; as the learning proceeds, NEAT starts to outperform rtNEAT and at the end it reaches a significantly better performance.

The second set of experiments focused on knowledge transfer. The champion evolved for each track were used to seed the evolution of drivers for different tracks. The results suggest that is convenient to exploit previous knowledge: the new seeds boost the learning improving the final performance effectively. When using drivers evolved on simple tracks as the seed for more challenging track, the overall behavior is similar to what previously found in that rtNEAT is initially better, but then NEAT catches up. However, when using drivers evolved on more challenging tracks, the behavior is opposite: initially, rtNEAT performs worse and then both models reach a similar performance. Our analysis suggests that, when starting from a more experienced driver, steady-state evolution (rtNEAT) tends to exploit the current knowledge rather than recombining good building-blocks of the experienced drivers in the initial population, resulting in a lower performance. In contrast, generational evolution (NEAT) recombines the existing good behaviors more rapidly, resulting in a faster adaptation to the new track. However, at the end, NEAT and rtNEAT reach a comparable performance (in fact, there is no statistically significant difference at the end).

The last set of experiments was aimed at testing the generalization capabilities of the eight approaches. The results suggest that NEAT can generalize more, producing (significantly more) drivers capable of completing two laps on difficult unknown tracks.

With respect to the evaluation strategy, all the experiments performed basically agree:  $\varepsilon$ -Greedy is significantly worse than  $\varepsilon$ -Greedy-Improved, Softmax, and Interval-based, which perform similarly. This results confirm the findings of Whiteson and Stone [32], who also noted that, on the typical reinforcement learning benchmarks,  $\varepsilon$ -Greedy performed worse. In addition, as reported in [32], also our analysis shows no statistically significant difference between Softmax and Interval-based. In particular, with respect to the generalization capabilities, the results also shows a difference in the performance of  $\varepsilon$ -Greedy-Improved, Softmax and Interval-based, with a cluster containing what appear to be the best techniques ( $\varepsilon$ -Greedy-Improved and Softmax) and a cluster containing only Interval-based.

When compared to off-line neuroevolution (e.g., [8]), our results suggest that on-line neuroevolution can evolve faster drivers while being computationally less expensive. Although, a fair comparison of the two approaches is infeasible (as they differ both for their goals and the experimental setup), a quick comparison we performed show that with the same number of evaluations, the best individuals evolved using off-line neuroevolution perform slightly worse than the best ones evolved using on-line neuroevolution. The average lap times achieved using off-line neuroevolution for A-Speedway, CG-Track 1, and Ruudskogen are respectively  $30.11 \pm 0.4$ ,  $62.00 \pm 4.8$ , and  $106.21 \pm 8.5$ ; our approach performs statistically significantly better on the most difficult tracks, CG-Track 1 and Ruudskogen (Table 1). As expected [32], a qualitative analysis shows that off-line

neuroevolution spends most of its computation resources on the evaluation of poor drivers. Overall, off-line neuroevolution turns out to be computationally more expensive since it also introduces huge overheads to reset the state of the simulators between evaluations and crashes. In fact, we noted that, off-line neuroevolution only spends between the 12% and the 22% of the overall simulation time for the actual evaluation of individuals.

There are still several issues worth investigating. Our future research directions include the use of adaptive evaluation selection strategy (for instance, by adapting the  $\varepsilon$  in  $\varepsilon$ -Greedy as done in reinforcement learning [28]) to improve the performance during the last stage of the evolution. We also plan to investigate how the length of the evaluation timeslot and the number of reevaluations (the parameter  $k$ ) influence the performance with respect to the characteristics of the track and the neuroevolution approach used.

## References

- [1] The Open Racing Car Simulator Website, 2008. <http://torcs.sourceforge.net/>.
- [2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, 2002.
- [3] Sander Bakkes, Pieter Spronck, and Eric O. Postma. Team: The team-oriented evolutionary adaptability mechanism. In Matthias Rauterberg, editor, *ICEC*, volume 3166 of *Lecture Notes in Computer Science*, pages 273–282. Springer, 2004.
- [4] 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.
- [5] Bobby D. Bryant. *Evolving Visibly Intelligent Behavior for Embedded Game Agents*. PhD thesis, Department of Computer Sciences, University of Texas, Austin, TX, 2006.
- [6] L. Cardamone, D. Loiacono, and P.L. Lanzi. On-line neuroevolution applied to the open racing car simulator. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 2622–2629, May 2009.
- [7] Luigi Cardamone. On-line and Off-line Learning of Driving Tasks for The Open Racing Car Simulator (TORCS) Using Neuroevolution. Master’s thesis, Politecnico di Milano — Dipartimento di Elettronica e Informazione.
- [8] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1179–1186, New York, NY, USA, 2009. ACM.
- [9] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1179–1186, New York, NY, USA, 2009. ACM.
- [10] Luigi Cardamone, Daniele Loiacono, and Pier-Luca Lanzi. On-line neuroevolution applied to the open racing car simulator. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 2622–2629, May 2009.

- [11] S. A. Glantz and B. K. Slinker. *Primer of Applied Regression & Analysis of Variance*. McGraw Hill, 2001. Second edition.
- [12] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.
- [13] 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.
- [14] I.V. Karpov, T. D’Silva, C. Varrichio, K.O. Stanley, and R. Miikkulainen. Integration and evaluation of exploration-based learning in games. In *Proc. IEEE Symposium on Computational Intelligence and Games*, pages 39–44, 2006.
- [15] 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.
- [16] Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Simulated car racing championship 2009: Competition software manual. Technical report, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2009.
- [17] Simon Lucas. Evolving a neural network location evaluator to play ms. pac-man. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 203–210, 2005.
- [18] W.G. Macready and II Wolpert, D.H. Bandit problems and the exploration/exploitation tradeoff. *Evolutionary Computation, IEEE Transactions on*, 2(1):2–22, Apr 1998.
- [19] Jacob Kaae Olesen, Georgios Yannakakis, and John Hallam. Real-time challenge balance in an RTS game using rtNEAT. In *Computational Intelligence and Games, 2008. CIG ’08. IEEE Symposium On*, pages 87–94, December 2008.
- [20] M. Parker and G.B. Parker. The evolution of multi-layer neural networks for the control of xpilot agents. In *Proc. IEEE Symposium on Computational Intelligence and Games CIG 2007*, pages 232–237, 2007.
- [21] Steffen Priesterjahn. *Online Imitation and Adaptation in Modern Computer Games*. PhD thesis, University of Paderborn, Paderborn, Germany, 2008.
- [22] Steffen Priesterjahn, Alexander Weimer, and Markus Eberling. Real-time imitation-based adaptation of gaming behaviour in modern computer games. In *GECCO ’08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1431–1432, New York, NY, USA, 2008. ACM.
- [23] John Reeder, Roberto Miguez, Jessica Sparks, Michael Georgiopoulos, Georgios Anagnostopoulos, J. Reeder, R. Miguez, J. Sparks, M. Georgiopoulos, and G. Anagnostopoulos. Interactively evolved modular neural networks for game agent control. In *Computational Intelligence and Games, 2008. CIG ’08. IEEE Symposium On*, pages 167–174, Dec. 2008.
- [24] 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.

- [25] 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.
- [26] Kenneth O. Stanley, Ryan Cornelius, and Risto Miikkulainen. Real-time learning in the nero video game. In R. Michael Young and John E. Laird, editors, *AIIDE*, pages 159–160. AAAI Press, 2005.
- [27] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.
- [28] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
- [29] C.H. Tan, J.H. Ang, K.C. Tan, and A. Tay. Online adaptive controller for simulated car racing. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*. *IEEE Congress on*, pages 2239–2245, June 2008.
- [30] Julian Togelius, Simon M. Lucas, Ho Duc Thang, Jonathan M. Garibaldi, Tomoharu Nakashima, Chin Hiong Tan, Itamar Elhanany, Shay Berant, Philip Hingston, Robert M. MacCallum, Thomas Haferlach, Aravind Gowrisankar, and Peter Burrow. The 2007 ieeec simulated car racing competition. *Genetic Programming and Evolvable Machines*, 9(4):295–329, 2008.
- [31] C. Watkins. *Learning from delayed rewards*. PhD thesis, King’s College, Cambridge, UK, 1989.
- [32] 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.
- [33] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Empirical studies in action selection with reinforcement learning. *Adaptive Behavior*, 15(1):33–50, 2007.
- [34] Mark Wittkamp, Luigi Barone, and Philip Hingston. Using neat for continuous adaptation and teamwork formation in pacman. In *Computational Intelligence and Games, 2008. CIG ’08. IEEE Symposium On*, pages 234–242, Dec. 2008.
- [35] Georgios N. Yannakakis and John Hallam. A generic approach for generating interesting interactive pac-man opponents. In *CIG. IEEE*, 2005.