Learning to Drive in the Open Racing Car Simulator Using Online Neuroevolution

Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi

***Abstract*—In this paper, we applied online neuroevolution to evolve nonplayer characters for *The Open Racing Car Simulator (TORCS)*. While previous approaches allowed online learning with performance improvements during each generation, our approach enables a finer grained online learning with performance improvements within each lap. We tested our approach on three tracks using two methods of online neuroevolution (NEAT and rtNEAT) combined with four evaluation strategies (ε-greedy, ε-greedy-improved, softmax, and interval-based) taken from the literature. We compared the eight resulting configurations on several driving tasks involving the learning of a driving behavior for a specific track, its adaptation to a new track, and the generalization capability to unknown tracks. The results we present show that, notwithstanding the several challenges that online learning poses, our approach 1) can successfully evolve drivers from scratch, 2) can also be used to transfer evolved knowledge to other tracks, and 3) can generalize effectively producing controllers that can drive on difficult unseen tracks. Our results also suggest that the approach performs better when coupled with online NEAT and also indicate that ε-greedy-improved and softmax are generally better than the other evaluation strategies. A comparison with typical offline neuroevolution suggests that online neuroevolution can be competitive and even outperform traditional offline approaches on more difficult tracks while providing all the interesting features of online learning. Overall, we believe that this study may represent an initial step toward the application of online neuroevolution in games.**

***Index Terms*—Machine learning, neuroevolution, online learning, online neuroevolution, simulated car racing.**

I. Introduction

In the field of computer games, offline learning has some limitations that online learning may be able to overcome. Recently, Spronck *et al.* [30] and Stanley *et al.* [32] suggested that a game intelligence learned offline may easily lead to predictable behaviors, whose weaknesses can be systematically exploited by the players to defeat the computer. Furthermore, the same approach might be unable to adapt to unexpected player's strategies or game situations (a typical example being a human player driving in the opposite direction in a car racing game) [30], [32]. Finally, they suggested that it might be difficult to adapt a game intelligence developed offline to match the skill level of a human player. These limitations are even more relevant in racing games where players demand opponents that are 1) believable, to preserve the suspension of disbelief; 2) adaptive, to match the players driving skills so as to offer a continually challenging game experience; and 3) nonpredicable, to avoid the repetitive patterns of the typical racing scenario. Accordingly, in this paper, we present an initial study on the application of online learning to simulated car racing. More precisely, we focus on existing methods of online neuroevolution and apply them to a state-of-the-art car racing simulator (*The Open Racing Car Simulator (TORCS)* [1]).

The application of neuroevolution to modern computer games has been mainly restricted to offline learning (e.g., [5], [14], [22], and [26]). In a typical scenario, an evolutionary algorithm is applied to a population of networks [5], [22], [32], [48]; 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 [22] or to drive one or more laps [7]); 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. Offline 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 [32], [47]. Recently, Whiteson and Stone [44] focused on neuroevolution for stochastic reinforcement learning problems and proposed an approach to enable online 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 [44]). In contrast, Whiteson and Stone [44] define a reservoir of evaluations that are allocated to individuals based on a mechanism that borrows from reinforcement learning [35]. As in offline 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 [44]). Opposite to what happens in typical offline neuroevolution, each individual receives a variable number of evaluations that depends on its performance: the most promising individuals receive an increasingly higher number of evaluations while less promising individuals receive fewer and fewer evaluations [44]. The approach of Whiteson and Stone [44] enables *online 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 [44] to computer games, which often require learning at a finer granularity.

Manuscript received November 02, 2009; revised December 29, 2009; accepted April 23, 2010. Date of publication June 07, 2010; date of current version September 15, 2010.

The authors are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano 20133, Italy (e-mail: cardamone@elet.polimi.it; loiacono@elet.polimi.it; lanzi@elet.polimi.it).

Digital Object Identifier 10.1109/TCIAIG.2010.2052102

In this work, we focus on existing methods for online neuroevolution, previously applied only to scientific testbeds [44] and a simple game [29], modify them for a realistic car racing simulator (*TORCS*) [1], and compare them in terms of learning capabilities, adaptivity, and generalization. Our study is based on the early works of Cardamone *et al.* [6], [9] and Reeder *et al.* [29] who, respectively, extended online neuroevolution [44] and real-time neuroevolution [32] to the finer granularity needed in the realm of computer games [6]. In previous approaches [5], [22], [26], [44], [48], evaluations correspond to one or more problem instances. In contrast, we focus on *very short timeslots* that cover only a small fraction of a problem instance. Accordingly, our evaluations involve only very small parts of the overall task. Consequently, while in [5], [22], [26], [44], and [48] 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 on 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 [44], the decision of which individual should be evaluated affects one or more problem instances (one or more laps in our case), so that longer amounts of time may be spent on evaluating 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 online learning with performance improvements *during each lap*, and not just *during each generation* as in [44]. However, the use of short timeslots also poses major challenges. First, 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. Second, since learning happens online and involves one car, individuals have to be tested sequentially so that 1) subsequent evaluations are correlated and performed starting from very different initial conditions; and 2) there may be abrupt discontinuities in the learning process (which can be adequately managed as in [9]). 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 online 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 tasks involving 1) the learning of a driving behavior for a specific track, 2) its adaptation to a new track, and 3) its generalization to unknown tracks. In particular, we considered online NEAT, introduced by Whiteson and Stone [44], and its steady-state version, online rtNEAT, introduced by Reeder *et al.* [29]. With respect to the evaluation selection strategies, we considered the three policies introduced in [44] (namely, ε-greedy, softmax, and interval-based) and ε-greedy-improved, which we first introduced in [6] and [9].

The results we present 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 [44] 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, rtNEAT initially performs significantly better than NEAT—as should be expected. However, later online NEAT quickly catches up and, at the end, it performs better than rtNEAT (although, the difference is not statistically significant). When adapting a driver to another track, our results show that both approaches reach a comparable performance with no statistically significant difference between them. The experiments on generalization show that online NEAT produces more drivers capable of driving on several difficult unknown tracks. With respect to the evaluation strategy, all our experiments show that ε-greedy is generally worse than the other strategies, which perform similarly, confirming the findings in [44]. We also performed a set of experiments to study the effect that the length of the evaluation timeslot has on the performance of online neuroevolution. Our results show that a too short evaluation slot boosts the learning at the beginning but usually leads to suboptimal drivers. A very long evaluation slot, comparable to the ones used in offline neuroevolution, dramatically slows down the learning and leads to poor drivers. Medium-sized evaluation slots provide reasonable learning speed and competitive drivers. Finally, we compared the online neuroevolution approaches with typical offline neuroevolution. Our results show that, notwithstanding the many more challenges that online neuroevolution has to deal with, it can provide competitive solutions and may even outperform offline neuroevolution on more difficult tracks.

Overall, our study highlights the many challenges that a realistic game domain poses to online learning (e.g., noisy and interdependent evaluations) and shows practical ways to tackle them using online neuroevolution. However, the methods we analyze here do not provide a principled solution to enable the application of online learning to commercial game platforms. In fact, the development of novel paradigms to solve such challenges is still an open issue, outside the goal of this study, and should be the topic of future research directions.

II. *TORCS*

TORCS [1] is a state-of-the-art open source car racing simulator that provides a sophisticated physics engine, full 3-D visualization, several tracks, several models of cars, and various game modes (e.g., practice, quick race, championship, etc.). The car dynamics are 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 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 that can be easily customized or extended to build new bots.

All the experiments reported in this paper have been carried out with version 1.3.1 of *TORCS* using the setup of the 2009 Simulated Car Racing Championship (see the competition pages for CEC-2009,[1] GECCO-2009,[2] CIG-2009,[3] and [18]). The car sensory inputs consist of two sets of rangefinders (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 damage, the current gear, etc. (see [18] for the complete list). The car is controlled by four effectors: the steering wheel, the gas pedal, the brake pedal, and the gear change [18].

III. Neuroevolution With Augmenting Topology

In this paper, we focused on neuroevolution with augmenting topology (NEAT [34]), one of the most successful and widely applied neuroevolution approaches. NEAT [34] 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 or about the type of connections involved (e.g., they can be feedforward or recursive). Accordingly, its evolutionary search starts from the simplest topology possible (i.e., a fully connected network containing only the input and 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 analysis. Finally, NEAT protects the structural innovations through the mechanism of *speciation*, which restricts the selection process to niches containing networks with a 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* [33], 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.

IV. Online Neuroevolution

Our work is based on two methods of online neuroevolution: 1) the generational approach, introduced by Whiteson and Stone [44], which extends NEAT for the online scenario, and 2) its steady-state version [29], which extends the steady-state version of NEAT (rtNEAT) to an online scenario.

A. Online Generational Neuroevolution With NEAT

In [44], Whiteson and Stone extended NEAT for online 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 offline 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 [44] introduce a budget of evaluations that the system can spend on the individuals in the populations. As in offline neuroevolution, each evaluation involves one problem instance. Opposite to what happens in typical offline neuroevolution, each individual receives a variable number of evaluations that depends on its performance: the most promising individuals receive an increasingly higher number of evaluations while less promising individuals receive fewer and fewer evaluations [44]. For this purpose, the system needs to balance exploration and exploitation as do reinforcement learning algorithms. Accordingly, online 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 online version of NEAT is reported in Algorithm 1.

Algorithm 1: Generational Online Evolution

procedure EVOLUTION(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
  Init(P); ▷ Randomly initialize P
  repeat
    ▷ Init fitness and evaluation count in P
    for $p \in P$ **do**
      $e(p) = 0$
      $f(p) = f_{\min}$ ▷ Set to min fitness
    end for
    repeat
      ▷ Select the individual to evaluate
      $p \leftarrow$ SELECT(P);
      ▷ Update the evaluation count
      $e(p) \leftarrow e(p) + 1$;
      ▷ Update the fitness of based on
      ▷ its current evaluation EVAL(p)
      $f(p) \leftarrow f(p) + (1/e(p))(\text{EVAL}(p) - f(p))$;
    until Evaluation Termination Criteria Met;
    ▷ Usual NEAT operators
    $P_s \leftarrow$ Selection(P);
    $P_r \leftarrow$ Recombination(P_s);
    $P_m \leftarrow$ Mutation(P_r);
    $P \leftarrow P_m$;
  until Maximum Number of Generations Reached
end procedure

[1]http://www.cec-2009.org/

[2]http://www.sigevo.org/gecco-2009

[3]http://www.ieee-cig.org/cig-2009

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, lines 4–12) 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 the evaluation, its counter is increased (Algorithm 1, line 10) and its estimated fitness is updated (Algorithm 1, 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 SELECT(p) at line 9 in Algorithm 1 that implements one of the evaluation selection strategies proposed in the literature [10], [44] (see Section IV-C). When the criterion to stop the evaluation is met (Algorithm 1, 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.

B. Online Steady-State Neuroevolution With rtNEAT

rtNEAT [33] 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 online scenario in [24] following what was already done by Whiteson and Stone for NEAT [44]. The online version of rtNEAT works basically the same as online NEAT; the only difference with respect to the generational version (Algorithm 1) are the lines from 12 to 16 that in rtNEAT are replaced with steady-state evolution (see [8] for an algorithmic description).

C. Evaluation Selection Strategies

In our study, we considered all the four evaluation selection strategies introduced in the literature: three of them (ε-greedy, softmax, and interval-based) are the ones considered in [44]; ε-greedy-improved is an extension of ε-greedy we introduced in [6] and [9]. Each strategy has been used to implement the SELECT procedure for generational or steady-state online neuroevolution (see Algorithm 1).

ε-**greedy** is probably the simplest action–selection strategy used in reinforcement learning [35] and its application to online evolution is rather straightforward. At each iteration, it selects with probability ε a random individual from the population while with probability $1-\varepsilon$ it selects the individual with the highest fitness (see [8] for an algorithmic description). The evaluation process ends (Algorithm 1, 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 online NEAT, a generational algorithm k has to be comparable to (possibly larger than) the population size [44], whereas for online 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 online NEAT while it is set to 5 for online rtNEAT. Note that this termination condition does not guarantee that all the individuals in the population will be evaluated at least once [10], [44].

ε-**greedy-improved** is an extension of ε-greedy, which we devised to avoid some of the limitations of the previous strategy. Like ε-greedy, this strategy selects the best individual with probability $(1-\varepsilon)$ while with probability ε 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 θ_{best}. The individuals evaluation process ends when 1) *all the individuals have been evaluated at least once* and 2) the best individual of the current population has been evaluated for at least θ_{eval} or it is not good enough (i.e., its fitness is below a threshold θ_{best}). These termination criteria guarantee that all the individuals of the population have been evaluated and that the fitness of the champion is very accurate. The parameter θ_{best} controls the exploration/exploitation tradeoff: the higher the value of θ_{best} is, the more the time spent exploring the search space for better networks will be. In practice, θ_{best} should be set to the lowest fitness value that a *good* network (i.e., a network with a reasonably high performance) should have. The parameter θ_{eval} is used to adjust the accuracy of the evaluation process: the noisier the single evaluation is, the higher the value of θ_{eval} should be.

The ε-greedy-improved strategy is reported in Algorithm 2.

Algorithm 2: ε-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: $\text{best} = \arg\max_{p \in P} f(p)$;
▷ If the best individual is not good enough
▷ return an individual never
▷ evaluated (explore)
3: **if** $(f(\text{best}) \leq \theta_{\text{best}})$ **then**
4: **return** $p | e(p) = 0$;
5: **end if**
▷ If all the individuals have been evaluated
▷ return the best individual (exploit)
6: **if** $(\nexists p \in P | e(p) = 0) \wedge (e(\text{best}) < \theta_{\text{eval}})$ **then**
7: **return** best;
8: **end if**
▷ Otherwise apply the ε-greedy strategy
9: **if** $(\text{rand}() < \varepsilon)$ **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**

At first, line 3, the fitness of the 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 ε-greedy selection is applied (Algorithm 2, line 9).

Softmax is a probabilistic action–selection strategy widely used in reinforcement learning. When applied to online evolution [44], it computes the selection probability of an individual p by using the Boltzmann distribution [35], [44] as follows:

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

where $f(p)$ is the fitness of p, P is the population, and τ is a parameter to control the balance between exploration and exploitation, which is set empirically [44]. A low value of τ favors exploitation, since small differences in the fitness of individuals will correspond to large differences in their selection probabilities; a high value of τ favors exploration, since even large differences in the individual fitnesses will correspond to small differences in their selection probabilities. The evaluation process terminates after k evaluations are completed [44]. The parameter k is set using the same criterion used for ε-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 3). 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 3, line 5). As for ε-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 ε-greedy.

Algorithm 3: 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 α confidence interval
5: **return** $\arg\max_{p \in P}[f(p) + z_{((1+\alpha)/2)} \cdot (\sigma(p)/\sqrt{e(p)})]$;
6: **end if**
7: **end procedure**

V. Related Work

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

A. Online Neuroevolution for TORCS

We first presented our approach in [6] and [9] where we also reported some initial results for the learning performance of online NEAT [44] on two tracks (Aalborg and Wheel 1 [1]) using ε-greedy-improved and softmax. In this paper, we extend our previous analysis and consider also 1) online rtNEAT [29], which was separately introduced at the same time our approach was initially developed [6], [9], and 2) another two evaluation selection strategies, ε-greedy and interval-based, which we did not consider before. In our initial analysis [6], [9], we focused only on the overall learning performance and the results were not statistically analyzed. In this work, we compare all the approaches and perform a statistical analysis of the results. In particular, we take into account both the learning speed and 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 tracks. In addition, we analyze adaptation of individuals and their generalization capabilities to unknown difficult tracks. We also study how the length of the evaluation slot influences the learning performance and, finally, we compare all the online approaches considered to offline neuroevolution.

B. Online Neuroevolution

Our approach shares some similarities with the work of Reeder *et al.* [29] who extended the approach of [44] to steady-state neuroevolution (i.e., rtNEAT [32]). However, there are some relevant differences between [29] and our approach. First, [29] considers a continual learning problem in that there is no concept of problem instance. Second, the problem in [29] is *simpler* than simulated car racing since subsequent evaluations are totally independent, whereas in our case, they are strongly correlated so that the outcome of a 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). Third, in [29], the game has no exceptional events that can jeopardize the outcome of subsequent evaluations that are actually present in our case (e.g., a driver can get stuck in a corner). Moreover, although in [29] individuals are evaluated on fixed timeslots (as in our approach), the fitness is not computed by averaging the outcomes of repeated evaluations, but updated by adding positive and negative rewards achieved during each evaluation. In addition, Reeder *et al.* [29] 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, in [29], it is not clear whether/how many complete players are evolved. Furthermore, Reeder *et al.* [29] only consider ε-greedy (the simplest evaluation strategy); in contrast, in this work, we consider and compare *all* the three strategies introduced in [44] (i.e., ε-greedy, softmax, and interval-based) and the ε-greedy-improved strategy introduced in [6] and [9]. Reeder *et al.* [29] 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 [44].

C. Online Learning and Evolutionary Computation

Learning classifier systems (LCSs) [12], [16] are probably the oldest and the most well-known evolutionary computation approaches to online learning. These are online evolutionary rule-based systems that 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 [35]—the same used in online neuroevolution to select individuals for reevaluation [44], [45].

Online 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 [31] 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 offline optimization problems while minimizing the number of evaluations. In contrast, online neuroevolution aims at solving online learning problems while maximizing the performance during learning. Finally, although the tradeoff 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., [35] and [43]) and in the literature related to the multiarmed bandit problems (e.g., [2] and [23]).

D. Online Learning and Games

In the recent years, several researchers applied online learning either to 1) develop innovative games [32] or 2) improve the players' game experience [3], [24], [27], [36], [47]. Among the others, the *NERO* project [32] is probably the most relevant example of online learning applied to games in which the goal is to train a team of agents to solve navigation and combat tasks through online evolution. In the development of *NERO*, Stanley *et al.* introduced rtNEAT [32], a *real-time neuroevolution* approach, and applied it to train a team of agents *during* the game on the tasks defined by the players. Recently, rtNEAT has been applied also to a real-time strategy game [24] and to evolve a team of ghosts in *Pac-Man* [47]. Both these works exploited rtNEAT to adapt the difficulty level of the game to the skill of the player.

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

Beside neuroevolution, Bakkes *et al.* [3] applied an evolutionary algorithm to learn online a team strategy for the Capture of the Flag (CTF), a team-oriented game type of *Quake III* (a popular 3-D first-person shooter). In the racing game domain, Tan *et al.* [36] applied evolutionary strategies to adapt online the opponent's skills in the 2007 IEEE Congress on Evolutionary Computation (CEC) Simulated Car Racing Competition [42]. Finally, Priesterjahn *et al.* [28] 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.

E. Transfer Learning

In this work, we also applied online neuroevolution to adapt previously evolved drivers to new, unseen, tracks. This type of problem is part of the area known as transfer learning (TL) [38], [41]. This area studies how the knowledge learned on a *source* task can be exploited to speed up the learning on a new *target* task. Several methods of transfer learning have been specifically devised to be applied in reinforcement learning (RL) or temporal difference (TD) learning [35]. Such methods either focus on the transfer of value functions (e.g., [37] and [39]) or experience instances (e.g., [17]). Only few works in the literature (see [38] for an overview) focused on transferring the learned policies and, thus, can be applied to the methods that perform a search in the policy space (like NEAT [34]). One of the earliest examples of transfer learning in online evolution is due to Lanzi [15] who used populations of condition–action–prediction rules (called classifiers) evolved for simple tasks to seed the evolution of solutions in similar problems involving more complex state–action spaces. More related to our study is the work of Taylor *et al.* [40] who showed that it is possible to transfer the knowledge evolved using NEAT, even when the target task has either a different state space or a different action space. In particular, they used the population evolved by NEAT in the source task to seed the initial population for the target task. Note that, in [40], there is no indication of how NEAT's historical marking and complexification mechanisms (which assume that the initial population is initialized with networks having the same topology) are managed. To avoid this type of issue, in this study, we followed a much simpler approach and used only one (the best) evolved individual to seed the population for the target task.

Our approach also shares some similarities with [21] who applied case-injected genetic algorithms [20] to first-person shooters and real-time strategy games. During the evolutionary process, the worst members of the population are periodically replaced with individuals previously evolved that 1) proved to be successful and 2) are similar to the current best members of the population. Finally, our approach is somehow related also to delta coding [46], in which individuals are represented as differences with respect to a solution previously discovered.

VI. Learning to Drive in *TORCS* Using Online Neuroevolution

The application of offline neuroevolution to *TORCS* is straightforward [7]: 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 importantly, all the evaluations are independent and they can be potentially carried out in parallel.

In contrast, the use of online 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 performances 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 1) each candidate network is tested on rather brief sections of the track; 2) the evaluation of an individual intrinsically depends on the results of the evaluation of the previous individuals; 3) there might be several abrupt and unsafe changes in the car driving behavior when a controller replaces the previous one (in fact, different controllers might have totally different ways to deal with the same situation, leading to chopped driving behaviors); finally, 4) extremely poor controllers might stop the entire evaluation process (for instance, if the car gets stuck somewhere, all the subsequent controllers will perform poorly and possibly lead 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). Most importantly, in the online 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 an unfavorable position of the track just because the previous drivers performed poorly. The online neuroevolution approaches applied in this work (see Section IV) aim at reducing the noise and the uncertainty due to the evaluation process 1) by reevaluating the most promising candidates more than once and 2) by averaging the results of more evaluations into a unique fitness value.

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 is activated when the car is outside the track (where rangefinder inputs are unavailable [19]). The recovery policy was implemented as a separate programmed module inspired by the one available in the drivers distributed with *TORCS* (similar to what was done in [25]).

VII. 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 [18], and the version 1.0 of the NEAT/rtNEAT package distributed by the NEAT user group,[4] modified to compile with gcc-4.1.2.

[4]http://www.cs.ucf.edu/kstanley/neat.html#group.

A. Sensors and Actuators

We used the sensors provided by the competition software to build a sensor model consisting of 1) six rangefinder sensors to perceive the track edges along several directions (i.e., -90°, -60°, -30°, $+30^\circ$, $+60^\circ$, $+90^\circ$); 2) an aggregate sensor to perceive the track edges *in front of the car*, which combines the readings of the three rangefinders along the direction -10°, 0°, and $+10^\circ$; and 3) 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 m (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*.

B. Experimental Setup

An experiment consists of ten runs. In each run, a driver is evolved using either the online version of NEAT or the online version of rtNEAT and one of the four evaluation–selection policies available (i.e., ε-greedy, ε-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. Thus, a network is evaluated for approximately 20 s of actual play time, then 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 s (Section VIII) to complete a lap so that our evaluation slot is only the 66% of the fastest lap. In a more difficult track such as 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 I(a)].

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

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

where T_{out} is the number of game ticks the car spent 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; η and β are two constants introduced to guarantee that the fitness is positive and as a scaling factor for the average speed term (both η and β have been empirically set to 1000 in all the experiments reported here). All the NEAT/rtNEAT parameters were set to their default values, taken from the configuration files distributed with the software

TABLE I
ONLINE NEAT AND RTNEAT APPLIED TO *TORCS*: AVERAGE LAP TIME AFTER (A) 100 LAPS AND (B) 500; (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

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 ($\mu \pm \sigma$)
ε-Greedy	142.74 ± 31.66	134.37 ± 55.73	281.22 ± 38.19	293.94 ± 69.44	448.81 ± 56.23	507.59 ± 104.83
ε-Greedy-Improved	110.14 ± 19.78	**82.65 ± 13.16**	273.87 ± 31.26	244.29 ± 37.27	460.66 ± 44.38	445.44 ± 84.33
Softmax	115.66 ± 14.52	106.11 ± 25.28	248.22 ± 37.11	224.77 ± 39.73	432.70 ± 46.08	356.01 ± 65.22
Interval-based	101.32 ± 24.55	106.81 ± 13.17	255.81 ± 42.70	**224.13 ± 42.90**	380.11 ± 49.89	**349.76 ± 71.88**

(a)

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 ($\mu \pm \sigma$)
ε-Greedy	76.41 ± 9.60	104.93 ± 49.58	154.95 ± 13.64	233.23 ± 85.26	264.02 ± 20.98	428.14 ± 121.07
ε-Greedy-Improved	63.01 ± 5.16	**51.90 ± 3.76**	134.38 ± 7.98	**118.41 ± 15.71**	239.36 ± 10.16	251.54 ± 42.58
Softmax	65.07 ± 5.88	60.75 ± 8.52	135.01 ± 10.61	121.79 ± 10.23	241.52 ± 15.86	**200.50 ± 21.77**
Interval-based	67.18 ± 8.31	60.86 ± 9.44	140.64 ± 15.14	133.38 ± 26.62	222.04 ± 15.49	207.95 ± 36.60

(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 ($\mu \pm \sigma$)
ε-Greedy	39.12 ± 3.03	81.94 ± 29.02	87.78 ± 11.04	160.39 ± 44.87	155.83 ± 27.33	314.40 ± 59.16
ε-Greedy-Improved	37.36 ± 2.32	37.09 ± 2.33	73.39 ± 15.28	**68.40 ± 3.41**	**125.67 ± 11.54**	139.19 ± 8.16
Softmax	37.96 ± 4.79	36.24 ± 1.29	76.59 ± 6.80	75.11 ± 6.89	140.75 ± 10.02	139.64 ± 6.82
Interval-based	41.83 ± 4.64	**35.79 ± 0.94**	83.60 ± 12.93	72.15 ± 5.99	144.00 ± 9.32	135.61 ± 3.76

(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 ($\mu \pm \sigma$)
ε-Greedy	31.86 ± 0.99	(*) 59.08 ± 29.27	73.09 ± 13.40	(*) 114.95 ± 44.68	151.34 ± 19.87	(*) 297.63 ± 67.61
ε-Greedy-Improved	31.33 ± 1.01	31.17 ± 0.97	60.34 ± 22.63	**52.05 ± 1.76**	**110.50 ± 19.57**	134.23 ± 20.28
Softmax	31.04 ± 0.46	**30.63 ± 0.56**	55.38 ± 9.08	59.86 ± 7.71	132.57 ± 27.56	136.18 ± 20.18
Interval-based	31.20 ± 0.67	30.92 ± 0.91	92.03 ± 39.83	(*) 73.20 ± 30.13	155.73 ± 29.57	136.06 ± 19.38

(d)

package, except for a few ones.[5] For ε-greedy, softmax, and interval-based, k is set to 300 for online NEAT and to 5 for online rtNEAT following the indications given in [44]. The other parameters were set to values that we empirically determined. In particular, for ε-greedy and ε-greedy-improved, ε is set to 0.25; for ε-greedy-improved, θ_{eval} is 5 (the best individual must be evaluated at least five times) and θ_{best} is 2100 (i.e., only individuals with a fitness higher than 2100 can be considered champions); for softmax, τ is 50; for interval-based, the confidence level α has been set to 0.8. All the statistics reported in this paper are averages over ten runs.

C. Statistical Analysis

To analyze the results produced during each experiment, we performed a two-way analysis of variance (ANOVA) followed by the typical post-hoc procedures (SNK, Tukey, Scheffé, and Bonferroni) [11]. The two-way ANOVA has been applied to check whether there are significant differences in the results with respect to 1) the type of neuroevolution applied (NEAT or rtNEAT), and to 2) the evaluation strategy used (ε-greedy, ε-greedy-improved, softmax, or interval-based). The post-hoc procedures have been applied to identify groups of configurations with similar performance.

[5]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; `interspecies_mate_rate` = 0.01.

VIII. EXPERIMENTAL RESULTS

We applied online neuroevolution using NEAT and rtNEAT to evolve a driver for each one of the three tracks depicted in Fig. 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 that contains both fast and slow turns, both on the left and on the

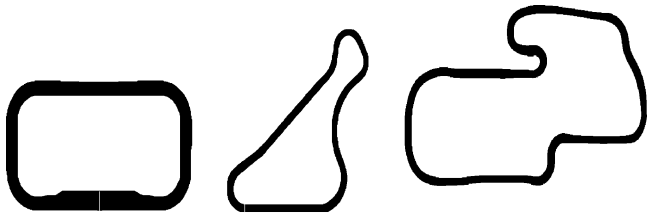

Fig. 1. Three tracks used for the experiments reported in this paper. (a) A-Speedway; (b) CG-Track 1; (c) Ruudskogen.

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.

A. Learning From Scratch

In the first set of experiments, we applied the online versions of NEAT and rtNEAT to evolve a driver starting from a population initialized as usually done in NEAT and rtNEAT, i.e., using the simplest network topology possible (Section III). Table I compares the performance of online NEAT/rtNEAT using one of the four selection strategies on the three tracks; statistics are averages over ten runs; best performances are highlighted in bold. Table I(a) reports the average lap time over *the first 100 laps*; Table I(b) reports the average lap time over *the first 500 laps*; Table I(c) reports the average lap time over *the last 100 laps*; and Table I(d) reports the average lap time of *the best individuals evolved* for each configuration during each run.

1) Statistical Analysis: For each track, we applied a two-way ANOVA [1] to test whether the differences in Table I are statistically significant *with respect to two factors*, the type of neuroevolution approach used (NEAT or rtNEAT), and the type of evaluation selection used (ε-greedy, ε-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.

2) Initial 100 Laps: As can be noted from Table I(a), 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 a very few exceptions). However, the two-way ANOVA [11] of the data in Table I(a), for the averages over the first 100 laps, shows that the reported differences are not statistically significant. The same analysis shows that, with respect to the selection algorithms, there is no statistically significant difference between ε-greedy-improved, softmax, and interval-based. Finally, the post-hoc procedures make evident a clear distinction in A-Speedway between ε-greedy (which is the worst one) and all the other three policies (ε-greedy-improved, softmax, and interval-based).

3) 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 I(b)] 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 ε-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 I(b) confirm and extend what was previously found: there is a significant difference between ε-greedy and the other three strategies in all the three tracks.

4) Last 100 Laps: Table I(c) reports the average lap times over the last 100 (learning) laps. The reported results show that, at the end, the difference between NEAT and rtNEAT is blurred: in eight out of the 12 cases, the average lap time obtained using NEAT is slightly higher than rtNEAT's one, but this difference is not statistically significant. In the remaining cases, NEAT performs significantly better than rtNEAT. With respect to the selection strategies, the statistical analysis confirms the previous findings showing a statistically significant difference between ε-greedy and the other three strategies in *all the tracks*.

5) 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 their final performance on the whole track. For this purpose, each champion drove two laps and we measured its performance as the time taken to complete the second lap. Table I(d) 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 on a 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 I(d) are similar to the ones found for the final 100 laps of the learning phase [Table I(c)] where the differences between NEAT and rtNEAT are blurred. At the end, when coupled with online rtNEAT, ε-greedy still performs significantly worse than the other three evaluation strategies. However, when online NEAT is used, interval-based performs worse in the most difficult tracks (although this difference is not statistically significant).

B. Seeded Evolution

We repeated the same set of experiments starting from populations seeded with the best individuals previously evolved for each track. The goal was to test whether, in *TORCS*, online neuroevolution could exploit previous existing knowledge and possibly improve it so as to evolve a better driver. Note that Taylor *et al.* [40] studied how to transfer previously evolved knowledge to more complex tasks involving different sensors/actuators models. In our case, the task is the same, to drive compet-

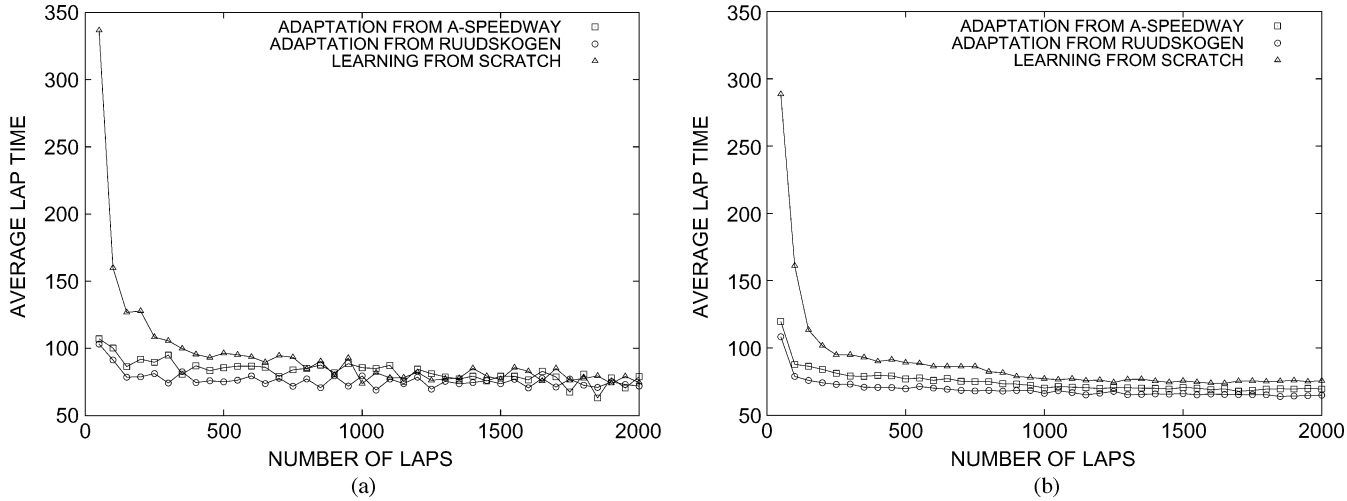

Fig. 2. Online (a) NEAT and (b) rtNEAT applied to the CG-Track 1 track with softmax starting from scratch (triangular dots), a population seeded with a champion from A-Speedway (squared dots), and a champion from Ruudskogen (circled dots). Curves are averages over ten runs.

TABLE II
ONLINE NEAT AND RTNEAT APPLIED FROM A SEEDED POPULATION: AVERAGE PERFORMANCE OF THE BEST INDIVIDUALS

Source Track	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 ($\mu \pm \sigma$)
A-Speedway	ε-Greedy			69.63 ± 20.52	83.61 ± 17.72	131.08 ± 17.67	164.62 ± 51.64
A-Speedway	ε-Greedy-Improved			**55.64 ± 10.32**	59.98 ± 21.48	121.76 ± 26.32	153.51 ± 21.27
A-Speedway	Softmax			60.20 ± 14.25	(*) 57.96 ± 15.28	**112.61 ± 10.58**	137.38 ± 23.76
A-Speedway	Interval-based			87.06 ± 39.79	(*) 86.98 ± 24.78	155.02 ± 46.57	154.40 ± 15.86
CG-Track 1	ε-Greedy	30.49 ± 0.68	31.43 ± 0.77			104.18 ± 21.95	86.99 ± 1.81
CG-Track 1	ε-Greedy-Improved	30.37 ± 0.66	29.96 ± 0.37			(*) 89.02 ± 3.93	90.42 ± 18.35
CG-Track 1	Softmax	**29.73 ± 0.18**	29.85 ± 0.38			**85.33 ± 3.63**	93.26 ± 20.83
CG-Track 1	Interval-based	29.80 ± 0.21	30.11 ± 0.41			102.69 ± 31.34	94.44 ± 20.54
Ruudskogen	ε-Greedy	30.94 ± 0.60	32.12 ± 0.27	61.97 ± 11.52	70.06 ± 9.03		
Ruudskogen	ε-Greedy-Improved	30.90 ± 0.63	**30.07 ± 0.97**	62.83 ± 15.39	**48.32 ± 1.41**		
Ruudskogen	Softmax	30.12 ± 0.14	30.24 ± 0.25	(*) 57.48 ± 11.67	64.45 ± 16.99		
Ruudskogen	Interval-based	30.31 ± 0.38	30.50 ± 0.50	61.50 ± 9.43	(*) 70.97 ± 22.01		

itively, as well as the sensors/actuators model; accordingly, we applied a much simpler approach than the one in [40].

For each configuration, we applied online neuroevolution starting from a seeded population and let the car drive for 2000 laps. All the experiments performed demonstrate a behavior similar to the one illustrated by Fig. 2 where we compare the performance of NEAT [Fig. 2(a)] and rtNEAT [Fig. 2(b)], in CG-Track 1 using softmax, when the evolutionary process 1) starts from scratch, 2) is seeded with a champion of A-Speedway, and 3) is seeded with a champion of Ruudskogen. As the curves show, seeding the population with a champion (significantly) boosts the learning at the beginning. At the end, after 2000 laps, the performance achieved from a seeded population is *at least* as good as the performance achieved from scratch.

Table II summarizes all the results reporting the average lap times of *the best individuals* evolved in each run with online NEAT and rtNEAT; statistics are averages over ten runs; best performances are highlighted in bold. In this case, the two-way ANOVA reports no statistically significant difference between the performance of NEAT and rtNEAT or among the four evaluation strategies. It is worth noting however that the best performance is obtained when a driver evolved on a complex track is used as a seed to evolve a driver for a simpler track. For instance, when the A-Speedway champion is used to seed the evolution of a driver for CG-Track 1, the final performances are worse than those obtained when using the Ruudskogen champion as a seed.

Note that our analysis does not take into account the time required to evolve the champions (and thus follows the *target task time scenario* of [38]). However, the goal of this set of experiments was rather limited and way behind the wider scope of transfer learning. In fact, our goal was just to investigate the possibility of reusing previously evolved drivers on new unseen track as typically happened during many recently organized scientific car racing competitions.

C. 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.

Our results (see Table IV in the Appendix) show that NEAT evolves drivers that, on the new unseen tracks, are generally faster than the ones evolved by rtNEAT. The usual two-way ANOVA shows that such difference is always statistically significant on all the three tracks with a confidence level of 99.9%. With respect to the evaluation strategies, the statistical analysis confirms what was previously found showing that ε-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 using ε-greedy-improved, followed by softmax, interval-based, and ε-greedy.

We also analyzed the data regarding the number of champions that were able to complete the test process (reported between parentheses in the Appendix as Table IV) using a two-way ANOVA. The analysis shows that the number of champions evolved using NEAT, which 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 that can race on new unseen tracks. With respect to the evaluation strategies, the two-way ANOVA again shows that ε-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 ε-greedy (the worse one), one consisting of interval-based and softmax (the middle ones), and one containing ε-greedy-improved (the best one).

Interestingly, our analysis also shows that the champions evolved in A-Speedway can generalize significantly less than the drivers evolved in CG-Track 1 and Ruudskogen. In fact, the number of A-Speedway champions that can complete two laps on an unknown track is much smaller than that of CG-Track 1 and Ruudskogen champions and this difference is statistically significant with a confidence level of 99%. This result confirms what was previously found in the experiments with seeded population: it is convenient to use drivers evolved on the most difficult tracks.

D. Discussion

Our experimental results show that online rtNEAT learns faster at the beginning but, somewhat surprisingly, the difference soon becomes blurred so that at the end there are no statistically significant differences. The two approaches perform similarly also when applied to adapt an old driver to a new track. However, NEAT appears to be more reliable when it comes to generalizing in that it evolves drivers that can complete significantly more laps on unknown tracks. Among the four selection strategies, the analysis suggests that ε-greedy-improved and softmax may be the best ones. Thus, *overall*, our results suggest that online NEAT may be the best approach to evolve a driver for *TORCS* and that NEAT should be coupled with ε-greedy-improved or softmax.

Indeed, when learning from scratch, rtNEAT *at the beginning* [Table I(a)] performs significantly better than NEAT: since it uses steady-state evolution, rtNEAT explores the solution space less than NEAT (which is generational); accordingly, rtNEAT initially reaches a better performance. However, in the long run, NEAT catches up and at the end NEAT and rtNEAT perform similarly.

The experiments with seeded populations (Section VIII-B) show that it is possible to transfer previous knowledge (previous evolved drivers) to seed the evolution of drivers for different tracks. More interestingly, they show that it is more convenient to transfer the knowledge evolved from difficult tracks to simpler ones (e.g., to use a driver evolved for Ruudskogen to seed a population for CG-Track 1) rather than the other way around. Overall, the experiments with seeded populations show no statistically significant difference between the performance of NEAT and rtNEAT or the four selection strategies. Moreover, 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 suggests that NEAT can generalize better than rtNEAT in that it generates significantly more controllers capable of driving on difficult unknown tracks.

With respect to the evaluation strategies, all the experiments seem to agree. ε-greedy is the worst 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 that the performance achieved with interval-based is significantly worse than the one achieved with ε-greedy-improved and softmax.

IX. Length of the Evaluation Slot

We repeated the experiments of Section VIII-A using short evaluation slots of 500 ticks, medium evaluation slots of 2000 ticks (twice the length used in Section VIII-A), and long evaluation slots of enough ticks to complete two laps in each track (i.e., 3000 ticks for A-Speedway, 6000 ticks for CG-Track 1, and 12000 ticks for Ruudskogen).

Overall, all the experiments show the same behavior exemplified in Fig. 3 where we report the learning curves for online NEAT [Fig. 3(a)] and online rtNEAT [Fig. 3(b)] applied to CG-Track 1 using softmax with different evaluation slot sizes. As can be noted, very short evaluation slots result in faster learning at the beginning, but lead to a final performance worse than that achieved with longer evaluation slots. However, with very long evaluation slots, the learning is initially very slow and, after 2000 laps, the performance is much worse than that achieved with shorter slots. Evaluation slots of 1000 or 2000 ticks (approximately 20 and 40 s of actual game time) result in the best performances and reasonable learning speeds at the beginning. In particular, our results show that, in general, there is no significant difference between the performance using 1000 or 2000 game ticks. Thus, an evaluation slot of 1000 ticks seems to be the best tradeoff since it allows for the best learning speed (shorter slots correspond to faster learning) without a statistically significant decrease in the final performance.

X. Comparison With Offline Neuroevolution

In the last set of experiments, we compared the online approaches considered in this paper with the typical offline neuroevolution [5], [14], [22], [26], previously applied to *TORCS* in [7]. In this case, the evaluations of individuals are performed 1) *independently*, in that, they potentially may be carried out

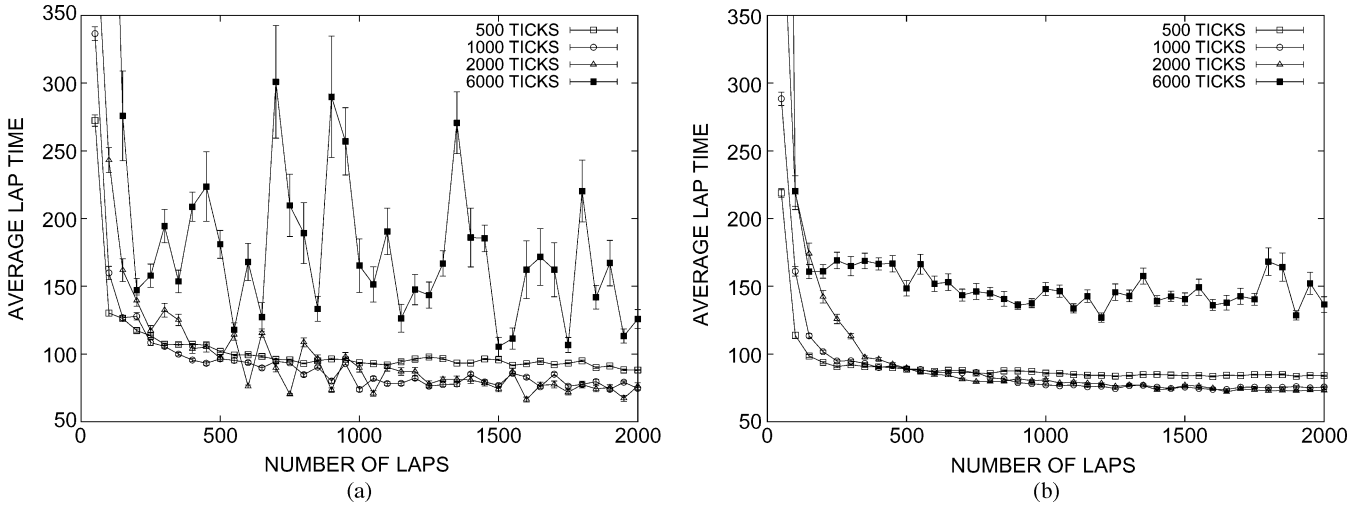

Fig. 3. Online neuroevolution applied to the CG-Track 1 track using softmax and different sizes of the evaluation slot: (a) NEAT; (b) rtNEAT. Curves are averages over ten runs; bars report the standard error.

TABLE III
OFFLINE AND ONLINE NEUROEVOLUTION APPLIED TO *TORCS*: AVERAGE PERFORMANCE OF THE BEST INDIVIDUALS IN THE FINAL POPULATIONS; STATISTICS ARE AVERAGES OVER TEN RUNS

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 ($\mu \pm \sigma$)
ε-Greedy	31.86 $\pm$ 0.99	(*) 59.08 $\pm$ 29.27	73.09 $\pm$ 13.40	(*) 114.95 $\pm$ 44.68	151.34 $\pm$ 19.87	(*) 297.63 $\pm$ 67.61
ε-Greedy-Improved	31.33 $\pm$ 1.01	31.17 $\pm$ 0.97	60.34 $\pm$ 22.63	**52.05 $\pm$ 1.76**	**110.50 $\pm$ 19.57**	134.23 $\pm$ 20.28
Softmax	31.04 $\pm$ 0.46	30.63 $\pm$ 0.56	55.38 $\pm$ 9.08	59.86 $\pm$ 7.71	132.57 $\pm$ 27.56	136.18 $\pm$ 20.18
Interval-based	31.20 $\pm$ 0.67	30.92 $\pm$ 0.91	92.03 $\pm$ 39.83	(*) 73.20 $\pm$ 30.13	155.73 $\pm$ 29.57	136.06 $\pm$ 19.38
off-line	30.21 $\pm$ 0.42	**29.73 $\pm$ 0.22**	55.10 $\pm$ 5.51	59.24 $\pm$ 8.01	(*) 145.45 $\pm$ 24.08	(*) 144.09 $\pm$ 31.59

in parallel, and 2) more accurately, in that, each evaluation involves a complete problem instance. We applied (offline) NEAT and rtNEAT to evolve drivers for each one of the tracks previously considered. Each individual was evaluated by applying it to drive two laps on the target track. All the evaluations started with the car placed in the same position and ended when either the car completed two laps or a large amount of game ticks had elapsed (to avoid spending to much time when a driver gets stuck). The fitness is the same used for online neuroevolution (see Section VI and [7]). Note that it is not possible to compare online and offline neuroevolution based on the number of laps completed (as we did for the online case) since offline evolution involves multiple evaluations and not just one car racing on the track. Accordingly, to provide a fair comparison, we computed the average number of game ticks used for each setting tested in Section VIII-A and stopped offline neuroevolution when a comparable number of actual game seconds had elapsed.[6]

Table III compares the average lap times of the best individuals evolved using 1) online neuroevolution [taken from Table I(d)], and 2) offline neuroevolution; the statistics are averages over ten runs. In the simplest track, A-Speedway, offline neuroevolution performs significantly better than online NEAT and offline rtNEAT at a 99% confidence level. As the track difficulty increases, in CG-Track 1, the performance of online approaches increases and the difference becomes more blurred: offline neuroevolution only outperforms online approaches using ε-greedy, which proved to be the worst performing online approach. In the most difficult track, Ruudskogen, online NEAT using ε-greedy-improved performs significantly better than offline neuroevolution (with a confidence of the 99%) while online rtNEAT using ε-greedy (the worst combination possible for online neuroevolution) performs significantly worse than offline NEAT and rtNEAT.

[6]In A-Speedway, offline neuroevolution was stopped after 110 000 actual game seconds were elapsed; in CG-Track 1 after 220 000 game seconds; in Ruudskogen after 405 000 game seconds.

XI. CONCLUSION

We applied online neuroevolution to evolve drivers for *TORCS* [1]. We focused on the two major online neuroevolution approaches (online NEAT [44] and online rtNEAT [29]) and on the four evaluation selection strategies introduced in the literature (ε-greedy, softmax, interval-based [44], and ε-greedy-improved [6]). We considered eight methods (each one combining one neuroevolution approach with an evaluation strategy). We performed three sets of experiments to compare the methods in terms of learning capabilities, adaptation, and generalization. We also performed an experiment to study the influence of the length of the evaluation timeslot on the overall performance. At the end, we compared the online approaches considered with the most typical offline neuroevolution.

In the first set of experiments, we applied the eight methods to evolve a driver for three different tracks of increasing difficulty

TABLE IV
TESTING THE CHAMPIONS OF EACH TRACK ON OTHER UNSEEN TRACKS. THE AVERAGE LAP TIME FOR EACH CHAMPION ON EACH TRACK WITH THE STANDARD DEVIATION AND, BETWEEN PARENTHESES, THE NUMBER OF EVOLVED CHAMPIONS THAT WERE ABLE TO COMPLETE THE TWO LAPS

Test Track	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 ($\mu \pm \sigma$)
alpine-1	ε-Greedy	260.54 ± 118.15 (5)	— (0)	299.20 ± 198.56 (7)	303.85 ± 55.28 (3)	184.73 ± 21.46 (7)	343.88 ± 135.86 (2)
	ε-Greedy-Improved	236.00 ± 51.13 (7)	276.81 ± 157.98 (7)	213.94 ± 36.55 (7)	219.50 ± 28.80 (9)	215.95 ± 14.57 (9)	180.64 ± 17.27 (9)
	Softmax	220.13 ± 95.06 (3)	163.58 ± 10.72 (5)	204.66 ± 20.04 (8)	204.10 ± 39.06 (7)	211.63 ± 35.64 (10)	211.29 ± 30.63 (8)
	Interval-based	206.31 ± 51.08 (5)	310.19 ± 111.59 (2)	229.33 ± 71.03 (7)	214.07 ± 10.31 (6)	248.12 ± 139.08 (7)	235.89 ± 82.29 (10)
alpine-2	ε-Greedy	168.65 ± 18.27 (3)	284.36 ± 0.00 (1)	148.83 ± 31.60 (7)	185.56 ± 74.33 (5)	135.29 ± 23.08 (7)	203.28 ± 94.59 (4)
	ε-Greedy-Improved	231.12 ± 112.84 (5)	141.50 ± 19.59 (5)	160.16 ± 43.37 (6)	174.31 ± 65.65 (8)	143.11 ± 18.03 (9)	124.16 ± 13.04 (9)
	Softmax	168.76 ± 71.42 (3)	163.19 ± 19.99 (2)	141.81 ± 13.43 (6)	151.51 ± 27.64 (6)	128.66 ± 9.83 (9)	156.24 ± 61.37 (8)
	Interval-based	162.18 ± 50.40 (5)	— (0)	186.05 ± 75.69 (7)	150.90 ± 12.88 (6)	136.22 ± 25.46 (9)	157.26 ± 63.23 (9)
e-track-1	ε-Greedy	251.97 ± 69.38 (9)	338.74 ± 39.44 (3)	168.17 ± 53.13 (9)	253.36 ± 79.85 (7)	202.77 ± 79.32 (8)	331.61 ± 10.18 (2)
	ε-Greedy-Improved	255.03 ± 66.41 (7)	214.63 ± 51.70 (7)	152.79 ± 25.49 (7)	169.75 ± 32.12 (8)	140.71 ± 30.07 (9)	162.59 ± 35.52 (8)
	Softmax	238.78 ± 42.87 (5)	256.87 ± 47.86 (7)	157.51 ± 21.27 (9)	193.63 ± 36.36 (7)	165.40 ± 35.97 (10)	183.93 ± 19.14 (8)
	Interval-based	224.05 ± 37.71 (8)	249.21 ± 44.19 (5)	206.51 ± 48.93 (6)	214.42 ± 50.17 (8)	160.22 ± 12.69 (5)	181.56 ± 20.40 (10)
e-track-2	ε-Greedy	280.19 ± 87.57 (2)	339.87 ± 0.00 (1)	217.22 ± 40.16 (6)	329.91 ± 47.39 (4)	261.40 ± 31.31 (4)	— (0)
	ε-Greedy-Improved	328.99 ± 0.00 (1)	271.21 ± 50.51 (4)	240.59 ± 60.98 (6)	197.62 ± 44.28 (8)	195.93 ± 62.66 (9)	205.93 ± 39.49 (6)
	Softmax	— (0)	279.32 ± 0.00 (1)	194.37 ± 33.05 (8)	235.38 ± 33.36 (7)	190.26 ± 51.08 (9)	213.82 ± 27.09 (5)
	Interval-based	— (0)	364.15 ± 0.00 (1)	265.67 ± 62.00 (4)	259.89 ± 57.26 (5)	209.85 ± 14.32 (5)	192.68 ± 33.42 (5)
e-track-3	ε-Greedy	287.92 ± 99.13 (7)	309.47 ± 68.56 (4)	197.03 ± 69.46 (9)	314.80 ± 111.38 (8)	180.81 ± 26.96 (10)	354.14 ± 124.12 (4)
	ε-Greedy-Improved	216.21 ± 36.78 (6)	201.03 ± 40.90 (7)	177.41 ± 23.94 (9)	169.85 ± 21.66 (9)	148.21 ± 33.67 (9)	178.45 ± 32.53 (7)
	Softmax	250.83 ± 31.63 (7)	263.42 ± 76.56 (6)	163.05 ± 22.30 (9)	183.22 ± 41.27 (10)	177.38 ± 34.19 (9)	186.09 ± 24.51 (10)
	Interval-based	176.81 ± 17.74 (6)	276.81 ± 115.09 (6)	254.12 ± 99.43 (10)	215.03 ± 71.87 (7)	192.29 ± 74.77 (9)	175.19 ± 23.79 (10)
e-track-4	ε-Greedy	403.87 ± 123.13 (7)	569.66 ± 142.25 (4)	287.51 ± 61.01 (9)	309.64 ± 80.42 (7)	285.93 ± 48.84 (8)	357.71 ± 116.45 (4)
	ε-Greedy-Improved	375.14 ± 93.63 (9)	292.69 ± 47.83 (7)	217.97 ± 60.05 (10)	198.69 ± 52.11 (10)	158.92 ± 21.91 (9)	237.09 ± 71.89 (9)
	Softmax	376.91 ± 101.42 (5)	382.36 ± 88.56 (6)	203.24 ± 58.27 (10)	207.36 ± 52.76 (10)	181.99 ± 37.77 (9)	233.52 ± 62.42 (8)
	Interval-based	322.25 ± 46.29 (8)	408.96 ± 107.28 (3)	229.67 ± 74.04 (10)	305.51 ± 111.50 (9)	245.00 ± 53.77 (8)	244.45 ± 63.17 (10)
e-track-6	ε-Greedy	375.74 ± 82.58 (2)	— (0)	212.10 ± 66.39 (3)	320.39 ± 75.96 (6)	192.88 ± 23.67 (2)	183.94 ± 0.00 (1)
	ε-Greedy-Improved	401.83 ± 0.00 (1)	345.25 ± 45.95 (3)	191.75 ± 58.01 (6)	225.96 ± 71.06 (7)	155.02 ± 27.01 (9)	201.02 ± 32.32 (6)
	Softmax	— (0)	— (0)	212.47 ± 45.40 (6)	220.04 ± 63.93 (8)	206.81 ± 65.15 (7)	207.68 ± 23.79 (6)
	Interval-based	296.12 ± 0.00 (1)	277.53 ± 0.00 (1)	227.87 ± 46.03 (7)	279.84 ± 120.50 (5)	225.98 ± 46.61 (4)	213.93 ± 33.15 (7)
forza	ε-Greedy	262.76 ± 89.89 (4)	458.56 ± 103.31 (7)	193.74 ± 11.48 (8)	311.57 ± 109.16 (7)	216.49 ± 50.41 (10)	357.84 ± 99.18 (8)
	ε-Greedy-Improved	274.26 ± 30.45 (7)	277.38 ± 48.07 (7)	192.76 ± 37.71 (8)	238.41 ± 67.02 (9)	186.87 ± 37.34 (10)	193.37 ± 28.93 (10)
	Softmax	328.81 ± 119.77 (7)	375.33 ± 98.19 (7)	229.11 ± 76.68 (7)	214.38 ± 36.23 (8)	194.83 ± 29.21 (10)	204.00 ± 28.69 (10)
	Interval-based	327.16 ± 141.17 (8)	306.18 ± 63.84 (4)	261.42 ± 55.61 (8)	293.07 ± 86.04 (6)	239.13 ± 69.77 (10)	221.12 ± 61.09 (10)
g-track-1	ε-Greedy	120.12 ± 52.80 (9)	200.69 ± 37.06 (3)	73.09 ± 13.40 (10)	114.95 ± 44.68 (9)	85.14 ± 18.00 (10)	137.41 ± 38.64 (6)
	ε-Greedy-Improved	120.27 ± 44.21 (10)	106.14 ± 22.54 (8)	60.34 ± 22.63 (10)	52.05 ± 1.76 (10)	61.98 ± 14.23 (10)	66.80 ± 7.91 (10)
	Softmax	135.37 ± 61.48 (10)	113.00 ± 49.52 (8)	55.38 ± 9.08 (10)	59.86 ± 7.71 (10)	66.61 ± 13.16 (10)	83.66 ± 26.65 (10)
	Interval-based	120.25 ± 31.30 (10)	117.00 ± 29.57 (6)	92.03 ± 39.83 (10)	73.20 ± 30.13 (8)	91.83 ± 19.01 (10)	75.73 ± 19.86 (10)
g-track-2	ε-Greedy	172.11 ± 61.49 (7)	259.56 ± 49.01 (5)	123.44 ± 44.69 (10)	168.56 ± 39.69 (7)	133.29 ± 37.71 (10)	220.30 ± 79.96 (6)
	ε-Greedy-Improved	169.58 ± 42.42 (8)	131.47 ± 20.17 (7)	100.45 ± 21.72 (9)	105.63 ± 26.52 (10)	92.47 ± 20.78 (10)	103.57 ± 18.85 (10)
	Softmax	150.08 ± 43.13 (7)	154.28 ± 44.60 (6)	105.96 ± 19.88 (10)	121.74 ± 28.48 (10)	111.70 ± 33.27 (9)	115.28 ± 34.49 (9)
	Interval-based	143.35 ± 39.10 (8)	191.50 ± 65.99 (6)	115.52 ± 36.10 (8)	130.68 ± 39.76 (9)	124.63 ± 21.16 (8)	111.60 ± 40.88 (8)
g-track-3	ε-Greedy	223.03 ± 70.42 (7)	273.48 ± 14.85 (2)	176.42 ± 32.21 (7)	224.03 ± 71.71 (7)	166.30 ± 22.56 (10)	230.16 ± 0.00 (1)
	ε-Greedy-Improved	225.26 ± 70.90 (6)	205.16 ± 55.28 (5)	170.74 ± 68.66 (8)	163.33 ± 40.07 (10)	125.70 ± 29.63 (10)	147.18 ± 35.02 (8)
	Softmax	173.09 ± 17.75 (5)	208.70 ± 49.98 (6)	163.97 ± 53.88 (8)	177.67 ± 63.11 (9)	125.06 ± 24.52 (10)	175.01 ± 54.81 (10)
	Interval-based	203.26 ± 69.48 (5)	230.83 ± 15.99 (2)	168.58 ± 48.68 (8)	174.49 ± 16.07 (6)	133.05 ± 28.40 (8)	157.10 ± 50.09 (10)
ole-road-1	ε-Greedy	360.64 ± 91.97 (7)	624.40 ± 44.80 (2)	309.83 ± 109.62 (9)	374.49 ± 138.31 (7)	279.90 ± 74.83 (10)	485.27 ± 120.26 (4)
	ε-Greedy-Improved	346.33 ± 74.46 (8)	363.50 ± 59.95 (8)	272.69 ± 76.28 (8)	268.03 ± 59.51 (9)	222.22 ± 34.62 (10)	228.36 ± 28.94 (9)
	Softmax	377.35 ± 127.89 (8)	350.89 ± 156.47 (6)	272.28 ± 68.90 (8)	351.67 ± 143.24 (8)	211.02 ± 28.98 (10)	269.58 ± 75.37 (10)
	Interval-based	357.63 ± 111.94 (6)	391.85 ± 129.64 (3)	331.53 ± 117.33 (9)	341.41 ± 118.52 (6)	280.09 ± 61.33 (10)	246.49 ± 33.37 (10)
ruudskogen	ε-Greedy	259.34 ± 79.69 (5)	149.69 ± 0.00 (1)	175.18 ± 41.47 (10)	196.41 ± 32.48 (6)	151.34 ± 19.87 (10)	297.63 ± 67.61 (7)
	ε-Greedy-Improved	267.07 ± 67.02 (7)	230.70 ± 81.21 (7)	147.00 ± 21.28 (9)	170.31 ± 42.19 (10)	110.50 ± 19.57 (10)	134.23 ± 20.28 (10)
	Softmax	277.62 ± 69.64 (7)	229.16 ± 29.18 (5)	149.19 ± 36.14 (9)	167.65 ± 26.04 (9)	132.57 ± 27.56 (10)	136.18 ± 20.18 (10)
	Interval-based	241.82 ± 61.56 (6)	275.92 ± 23.06 (3)	192.28 ± 37.03 (10)	209.33 ± 69.02 (8)	155.73 ± 29.57 (10)	136.06 ± 19.38 (10)
wheel-1	ε-Greedy	279.95 ± 111.43 (9)	330.08 ± 88.50 (5)	174.39 ± 39.01 (10)	251.10 ± 94.08 (9)	170.16 ± 26.34 (9)	305.89 ± 125.85 (5)
	ε-Greedy-Improved	218.53 ± 41.69 (9)	205.67 ± 25.76 (8)	157.88 ± 25.21 (9)	146.84 ± 23.14 (10)	136.46 ± 16.38 (10)	144.26 ± 26.06 (10)
	Softmax	219.24 ± 38.84 (10)	259.63 ± 101.15 (9)	149.70 ± 26.88 (10)	162.52 ± 20.59 (10)	156.71 ± 45.43 (10)	152.00 ± 27.22 (10)
	Interval-based	193.85 ± 37.54 (8)	229.29 ± 51.75 (6)	177.48 ± 42.24 (10)	188.89 ± 62.54 (7)	156.15 ± 31.98 (10)	150.31 ± 19.77 (10)
wheel-2	ε-Greedy	359.86 ± 72.04 (5)	559.51 ± 39.50 (3)	284.33 ± 65.01 (10)	439.37 ± 87.77 (5)	289.18 ± 82.89 (10)	363.35 ± 105.20 (3)
	ε-Greedy-Improved	408.79 ± 116.08 (7)	390.42 ± 89.45 (8)	269.42 ± 100.93 (9)	247.26 ± 44.12 (10)	224.90 ± 70.25 (9)	240.01 ± 43.11 (10)
	Softmax	425.71 ± 39.08 (3)	455.42 ± 80.84 (6)	296.79 ± 91.99 (10)	277.66 ± 57.11 (9)	239.37 ± 35.06 (10)	300.44 ± 127.77 (10)
	Interval-based	365.01 ± 69.73 (7)	526.87 ± 7.97 (2)	324.04 ± 98.39 (10)	336.74 ± 122.25 (8)	294.20 ± 47.31 (9)	271.15 ± 89.24 (10)

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 catches up and, at the end, both

methods perform similarly with no statistically significant difference reported.

The second set of experiments focused on knowledge transfer. The champions evolved for each track were used to seed the evolution of drivers for different tracks. The results suggest that it is convenient to exploit previous knowledge: the new seeds speed up the learning process at the beginning and result in an overall improvement of the final performance. More interestingly, our results suggest that knowledge should be transferred from the drivers evolved on more difficult tracks.

The third 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: ε-greedy is significantly worse than ε-greedy-improved, softmax, and interval-based, which perform similarly. These results confirm the findings of Whiteson and Stone [44], who also noted that, on the typical reinforcement learning benchmarks, ε-greedy performed worse. In addition, as reported in [44], also our analysis shows no statistically significant difference between softmax and interval-based. In particular, with respect to the generalization capabilities, the results also show a difference in the performance of ε-greedy-improved, softmax, and interval-based, with a cluster containing what appear to be the *best technique* (ε-greedy-improved) and a cluster containing softmax and interval-based.

We performed another set of experiments to study how the length of the evaluation slot could affect the final performance. Our results show that short evaluation slots can boost the learning at the beginning, but might lead to the convergence to a suboptimal driver. Long evaluation slots (e.g., similar to the one that would be used for offline neuroevolution) result in an extremely slow learning at the beginning and very poor drivers at the end. Evaluation slots of 1000 or 2000 ticks appear to be the best choice in that they provide a reasonable learning speed and lead to a similar performance. In particular, we believe that 1000 ticks might be the best compromise providing the best learning speed initially, with no statistically significant decrease in the final performance.

In the last set of experiments, we compared the online approaches considered here with typical offline neuroevolution. Our results suggest that online neuroevolution can be competitive, notwithstanding the several challenges that online neuroevolution has to face. In fact, offline neuroevolution performs slightly better than online only in the easiest track (A-Speedway), while, as the track difficulty increases, the differences become blurred and, on the most difficult track, online approaches may outperform offline neuroevolution.

Overall, our results showed that, although a general solution to tackle the several challenges posed by realistic game platforms to online learning is still missing, it is possible to adapt existing approaches to obtain interesting performances. In fact, there are still several open issues that should be investigated to enable online learning in commercial games. A major long-term research direction regards the development of methods that can provide a principled solution to the issues discussed in this paper (e.g., the interdependent evaluation of candidate solutions using short time windows). With respect to the application of online neuroevolution to games, our future research directions include the use of adaptive evaluation selection strategies, to improve the performance during the last stage of the evolution, as well as the use of an adaptive evaluation slot size. We also plan to investigate how the number of reevaluations (the parameter k) influences the performance with respect to the characteristics of the track and the neuroevolution approach used. Finally, we wish to investigate the use of online neuroevolution in different domains, for instance, to adapt nonplayer character behaviors in first-person shooters.

APPENDIX

See Table IV.

REFERENCES

[1] "The open racing car simulator," 2008 [Online]. Available: http://torcs.sourceforge.net/

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2–3, pp. 235–256, 2002.

[3] S. Bakkes, P. Spronck, and E. O. Postma, "Team: The team-oriented evolutionary adaptability mechanism," in *Entertainment Computing—ICEC 2004*, ser. Lecture Notes in Computer Science, M. Rauterberg, Ed. Berlin, Germany: Springer-Verlag, 2004, vol. 3166, pp. 273–282.

[4] T. Beielstein and S. Markon, "Threshold selection, hypothesis tests, and doe methods," in *Proc. Congr. Evol. Comput.*, May 2002, vol. 1, pp. 777–782.

[5] B. D. Bryant, "Evolving visibly intelligent behavior for embedded game agents," Ph.D. dissertation, Dept. Comput. Sci., Univ. Texas, Austin, TX, 2006.

[6] L. Cardamone, "On-line and off-line learning of driving tasks for the open racing car simulator (TORCS) using neuroevolution," M.S. thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy, Nov. 2008.

[7] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Evolving competitive car controllers for racing games with neuroevolution," in *Proc. 11th Annu. Conf. Genetic Evol. Comput.*, New York, 2009, pp. 1179–1186.

[8] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Evolving drivers for TORCS using on-line neuroevolution," Illinois Genetic Algorithms Lab., Univ. Illinois at Urbana-Champaign, Urbana, IL, Tech. Rep. 2009008, 2009.

[9] L. Cardamone, D. Loiacono, and P. L. Lanzi, "On-line neuroevolution applied to the open racing car simulator," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 2622–2629.

[10] L. Cardamone, D. Loiacono, and P.-L. Lanzi, "On-line neuroevolution applied to the open racing car simulator," in *Proc. IEEE Congr. Evol. Comput.*, May 2009, pp. 2622–2629.

[11] S. A. Glantz and B. K. Slinker, *Primer of Applied Regression & Analysis of Variance*, 2nd ed. New York: McGraw-Hill, 2001.

[12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[13] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proc. 2nd Int. Conf. Genetic Algorithms Genetic Algorithms Appl.*, Mahwah, NJ, 1987, pp. 41–49.

[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 Symp. Comput. Intell. Games*, 2006, pp. 39–44.

[15] P. L. Lanzi, "Extending the representation of classifier conditions—Part I: From binary to messy coding," in *Proc. Genetic Evol. Comput. Conf.*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., Orlando, FL, Jul. 1999, pp. 337–344.

[16] P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds., *Learning Classifier Systems: From Foundations to Applications*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, April 2000, vol. 1813.
[17] A. Lazaric, "Knowledge transfer in reinforcement learning," Ph.D. dissertation, Dept. Electron. Inf., Politecnico di Milano, Milan, Italy, 2008.
[18] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated Car Racing Championship 2009: Competition Software Manual," Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy, Tech. Rep., 2009.
[19] D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, and Y. Saez, "The WCCI 2008 simulated car racing competition," in *Proc. IEEE Symp. Comput. Intell. Games*, 2008, pp. 119–126.
[20] S. J. Louis and J. McDonnell, "Learning with case-injected genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 8, no. 4, pp. 316–328, Aug. 2004.
[21] S. J. Louis and C. Miles, "Playing to learn: Case-injected genetic algorithms for learning to play computer games," *IEEE Trans. Evol. Comput.*, vol. 9, no. 6, pp. 669–681, Dec. 2005.
[22] S. Lucas, "Evolving a neural network location evaluator to play ms. pac-man," in *Proc. IEEE Symp. Comput. Intell. Games*, 2005, pp. 203–210.
[23] W. G. Macready and D. H. Wolpert, II, "Bandit problems and the exploration/exploitation tradeoff," *IEEE Trans. Evol. Comput.*, vol. 2, no. 1, pp. 2–22, Apr. 1998.
[24] J. K. Olesen, G. Yannakakis, and J. Hallam, "Real-time challenge balance in an RTS game using rtNEAT," in *Proc. IEEE Symp. Comput. Intell. Games*, Dec. 2008, pp. 87–94.
[25] E. Onieva, D. A. Pelta, J. Alonso, V. Milanés, and J. Perez, "A modular parametric architecture for the TORCS racing engine," in *Proc. IEEE Symp. Comput. Intell. Games*, 2009, pp. 256–262.
[26] M. Parker and G. B. Parker, "The evolution of multi-layer neural networks for the control of xpilot agents," in *Proc. IEEE Symp. Comput. Intell. Games*, 2007, pp. 232–237.
[27] S. Priesterjahn, "Online imitation and adaptation in modern computer games," Ph.D. dissertation, Dept. Comput. Sci., Univ. Paderborn, Paderborn, Germany, 2008.
[28] S. Priesterjahn, A. Weimer, and M. Eberling, "Real-time imitation-based adaptation of gaming behaviour in modern computer games," in *Proc. 10th Annu. Conf. Genetic Evol. Comput.*, New York, 2008, pp. 1431–1432.
[29] J. Reeder, R. Miguez, J. Sparks, M. Georgiopoulos, G. Anagnostopoulos, and Reeder, "Interactively evolved modular neural networks for game agent control," in *Proc. IEEE Symp. Comput. Intell. Games*, Dec. 2008, pp. 167–174.
[30] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma, "Adaptive game AI with dynamic scripting," *Mach. Learn.*, vol. 63, no. 3, pp. 217–248, 2006.
[31] P. Stagge, "Averaging efficiently in the presence of noise," in *Proc. 5th Int. Conf. Parallel Problem Solving From Nature*, London, U.K., 1998, pp. 188–200.
[32] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the nero video game," *IEEE Trans. Evol. Comput.*, vol. 9, no. 6, pp. 653–668, Dec. 2005.
[33] K. O. Stanley, R. Cornelius, and R. Miikkulainen, "Real-time learning in the nero video game," in *Proc. Artif. Intell. Interactive Digital Entertainment Conf.*, R. M. Young and J. E. Laird, Eds., 2005, pp. 159–160.
[34] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.
[35] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[36] C. H. Tan, J. H. Ang, K. C. Tan, and A. Tay, "Online adaptive controller for simulated car racing," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2008, pp. 2239–2245.
[37] M. E. Taylor and P. Stone, "Behavior transfer for value-function-based reinforcement learning," in *Proc. 4th Int. Joint Conf. Autonom. Agents Multiagent Syst.*, F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, Eds., New York, Jul. 2005, pp. 53–59.
[38] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, no. 1, pp. 1633–1685, 2009.
[39] M. E. Taylor, P. Stone, and Y. Liu, "Value functions for RL-based behavior transfer: A comparative study," in *Proc. 20th Nat. Conf. Artif. Intell.*, Jul. 2005, pp. 880–885.
[40] M. E. Taylor, S. Whiteson, and P. Stone, "Transfer via inter-task mappings in policy search reinforcement learning," in *Proc. 6th Int. Joint Conf. Autonom. Agents Multiagent Syst.*, May 2007, pp. 156–163.
[41] S. Thrun, "Is learning the n-th thing any easier than learning the first?," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 1996, pp. 640–646.
[42] J. Togelius, S. M. Lucas, H. D. Thang, J. M. Garibaldi, T. Nakashima, C. H. Tan, I. Elhanany, S. Berant, P. Hingston, R. M. MacCallum, T. Haferlach, A. Gowrisankar, and P. Burrow, "The 2007 IEEE CEC simulated car racing competition," *Genetic Programm. Evolvable Mach.*, vol. 9, no. 4, pp. 295–329, 2008.
[43] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, U.K., 1989.
[44] S. Whiteson and P. Stone, "On-line evolutionary computation for reinforcement learning in stochastic domains," in *Proc. 8th Annu. Conf. Genetic Evol. Comput.*, New York, 2006, pp. 1577–1584.
[45] S. Whiteson, M. E. Taylor, and P. Stone, "Empirical studies in action selection with reinforcement learning," *Adaptive Behavior*, vol. 15, no. 1, pp. 33–50, 2007.
[46] D. Whitley, K. Mathias, and P. Fitzhorn, "Delta coding: An iterative search strategy for genetic algorithms," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 77–84.
[47] M. Wittkamp, L. Barone, and P. Hingston, "Using neat for continuous adaptation and teamwork formation in pacman," in *Proc. IEEE Symp. Comput. Intell. Games*, Dec. 2008, pp. 234–242.
[48] G. N. Yannakakis and J. Hallam, "A generic approach for generating interesting interactive pac-man opponents," in *Proc. IEEE Symp. Comput. Intell. Games*, 2005, pp. 94–101.

Luigi Cardamone was born in Ispica (RG), Italy, in 1984. He received the first level degree *cum laude* and the second level degree *cum laude* in computer engineering from Politecnico di Milano, Milan, Italy, in 2006 and in 2008, respectively, where currently he is working towards the Ph.D. degree in computer engineering at the Department of Electronics and Information.

His research area is in computational intelligence and games.

Daniele Loiacono was born in Lecco, Italy, in 1980. He graduated *cum laude* in computer engineering from the Politecnico di Milano, Milan, Italy, in 2004 and received the Ph.D. degree in computer engineering from the Department of Electronics and Information, Politecnico di Milano, in 2008.

Currently, he is a Postdoctoral Researcher at the Department of Electronics and Information, Politecnico di Milano. His main research interests include machine learning, evolutionary computation, and computational intelligence in games.

Pier Luca Lanzi was born in Turin, Italy, in 1967. He received the Laurea degree in computer science from the Università degli Studi di Udine, Udine, Italy, in 1994 and the Ph.D. degree in computer and automation engineering from the Politecnico di Milano, Milan, Italy, in 1999.

Currently, he is an Associate Professor at the Department of Electronics and Information, Politecnico di Milano. His research areas include evolutionary computation, reinforcement learning, and machine learning. He is interested in applications to data mining and computer games.

Dr. Lanzi is a member of the editorial board of the *Evolutionary Computation Journal*, the IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, and *Evolutionary Intelligence*. He is also the Editor-in-Chief of the *SIGEVOlution*, the newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation.