International Journal of
Science and Engineering Investigations vol. 1, issue 6, June 2012
ISSN: 2251-8843

Comparison of Evolution Strategy, Genetic Algorithm and Their Hybrids on Evolving Autonomous Game Controller Agents

Hidehiko Okada [1], Jumpei Tokida [2], Yuki Fujii [3]
[1,2,3] Department of Intelligent Systems, Faculty of Computer Science and Engineering, Kyoto Sangyo University
([1] hidehiko@cc.kyoto-su.ac.jp)

***Abstract*-** Researchers have been applying artificial/computational intelligence (AI/CI) methods to computer games. In this research field, further researches are required to compare AI/CI methods with respect to each game application. In this paper, we report our experimental results on the comparison of two evolutionary algorithms (evolution strategy and genetic algorithm) and their hybrids, applied to evolving autonomous game controller agents. The games are the CIG2007 simulated car racing and the MarioAI 2009. In the application to the simulated car racing, premature convergence of solutions was observed in the case of ES, and GA outperformed ES in the last half of generations. Besides, a hybrid which uses GA first and ES next evolved the best solution among the whole solutions being generated. This result shows the ability of GA in globally searching promising areas in the early stage and the ability of ES in locally searching the focused area (fine-tuning solutions). On the contrary, in the application to the MarioAI, GA revealed its advantage in our experiment, whereas the expected ability of ES in exploiting (fine-tuning) solutions was not clearly observed. The blend crossover operator and the mutation operator of GA might contribute well to explore the vast search space.

***Keywords*-** *evolutionary algorithm; autonomous game controller agent; neuroevolution.*

I. Introduction

Researchers have been applying artificial/computational intelligence (AI/CI) methods to computer games, and reporting their research results in conferences including IEEE Conference on Computational Intelligence and Games (CIG)[1] and IEEE Congress on Evolutionary Computation (CEC)[2]. In these conferences, competitions on autonomous game AI agents have been held. For example, competitions on Simulated Car Racing[3], Mario AI[4], Ms. Pac-Man[5], etc., were held in CIG 2011[6]. To develop high performance agents, AI/CI methods such as artificial neural networks, fuzzy sets, evolutionary algorithms, swarm intelligence and enforcement learning have been applied. In this research field, further researches are required to compare AI/CI methods with respect to each game application; to investigate which methods can derive better agents than others for which application and why.

In this paper, we report our experimental results on the comparison of two evolutionary algorithms (evolution strategy (ES) [1] and genetic algorithm (GA) [2] and their hybrids applied to evolving autonomous game controller agents. The games are the CIG2007 simulated car racing and the MarioAI 2009. We select ES and GA because these are the representatives of the evolutionary algorithms.

II. Game Applications

We selected the CIG2007 simulated car racing and the MarioAI 2009 as the game applications because these games provided sample controller agents (written in Java) on the web[7,8]. The sample agents are neural network based ones; we expect sample agents will perform well as we tune values of their unit connection weights and unit biases. We apply evolutionary algorithms to the tuning of the weights and the biases. Training neural networks by means of evolutionary algorithms is known as neuroevolutions [3,4]. Unlike training with the back propagation algorithm, neuroevolutions do not require training data sets and gradient information of error functions.

A. CIG 2007 Simulated Car Racing

Fig.1 shows a screenshot of the CIG2007 simulated car racing. An autonomous agent controls its associated car to “visit as many way-points as possible in a fixed amount of time [5].”

A starter kit has been provided on the web[9]. Samples of car controller agents are included in `simplerace/classes/simplerace`. The agents are provided as Java classes. Source codes of the agents are also provided. We utilized the agent `RMLPController` (`simplerace/classes/simplerace/RMLPController.class`) in our research, because the agent performed better than other sample agents in

[1] http://www.ieee-cig.org/.
[2] http://cec2011.org/.
[3] http://cig.ws.dei.polimi.it/?page_id=175
[4] http://www.marioai.org/
[5] http://cswww.essex.ac.uk/staff/sml/pacman/PacManContest.html
[6] http://cilab.sejong.ac.kr/cig2011/?page_id=100
[7] http://julian.togelius.com/cig2007competition/
[8] http://julian.togelius.com/mariocompetition2009/gettingstarted.php
[9] http://julian.togelius.com/cig2007competition/simplerace.zip

our experiment.

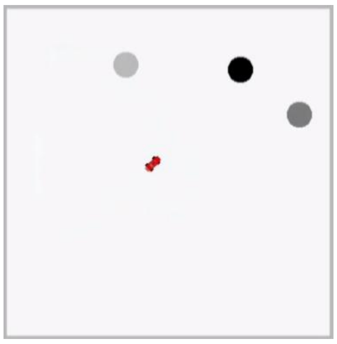


Figure 1. Screenshot of CIG2007 simulated car racing.

The following command starts car racing simulation[7]:

```
> java simplerace.Play evolved.xml
```

The argument of the `simplerace.Play` class, `evolved.xml`, is an XML-formatted file. The XML file includes an `<object>` element with which the agent class used as the controller in the simulation is specified. For example, the following example of description:

```
<object type="simplerace.RMLPController" id="0">
```

denotes that the class `simplerace.RMLPController` is used as the controller agent.

This `RMLPController` agent is implemented with a recurrent multi-layer perceptron (RMLP); as the inputs, the RMLP receives data of car environment captured by car sensors, and the RMLP outputs data to actuate (control) its car. Values of RMLP weights and biases are specified with `<array>` elements in the XML file. Thus, better `RMLPController` will be evolved as the values of `<array>` elements are tuned. We experimentally compare the ability of evolutionary algorithms and their hybrids on this `RMLPController` neuroevolutions.

B. *MarioAI 2009*

Fig.2 shows a screenshot of Mario game played by a MarioAI agent. An autonomous agent controls Mario to “win as many levels (of increasing difficulty) as possible.”[8]

A starter kit has been provided on the web[10]. Samples of Mario controller agents are included in `marioai/classes/ch/idsia/ai`. The agents are provided as Java classes. Source codes of the agents are also provided. We experimentally utilized the agent `SmallSRNAgent` (`marioai/classes/ch/idsia/ai/agents/ai/SmallSRNAgent.class`) in this research.

Figure 2. Screenshot of MarioAI 2009 game play.

The following command starts game play simulation[8]:

```
> java ch.idsia.scenarios.Play evolved.xml
```

The argument of the `ch.idsia.scenarios.Play` class, `evolved.xml`, is an XML-formatted file. The XML file includes an `<object>` element with which the agent class used as the controller in the simulation is specified. For example, the following example of description:

```
<object type="ch.idsia.ai.agents.ai.SmallSRNAgent" id="0">
```

denotes that the class `ch.idsia.ai.agents.ai.SmallSRNAgent` is used as the controller agent.

This `SmallSRNAgent` is implemented with a recurrent multi-layer perceptron (RMLP); as the inputs, the RMLP receives data of environmental state captured by Mario sensors, and the RMLP outputs data to actuate (control) Mario. Values of RMLP weights and biases are specified with `<array>` elements in the XML file. Thus, better `SmallSRNAgents` will be evolved as the values of `<array>` elements are tuned. We experimentally compare the ability of evolutionary algorithms and their hybrids on this `SmallSRNAgent` neuroevolutions.

III. Applying Evolutionary Algorithms to Game Controllers

A solution of the optimization problem is a 162 dimensional real vector $\vec{x} = (x_1, x_2, \ldots, x_{162})$ in the application to the CIG2007 simulated car racing and a 405 dimensional real vector $\vec{x} = (x_1, x_2, \ldots, x_{405})$ in the application to the MarioAI 2009. Each x_i is a variable for an `<array>` element in the XML files.

A. *Evolution Strategy*

The steps of evolution by means of ES in our research are shown in Fig.3.

[10] http://julian.togelius.com/mariocompetition2009/marioai.zip

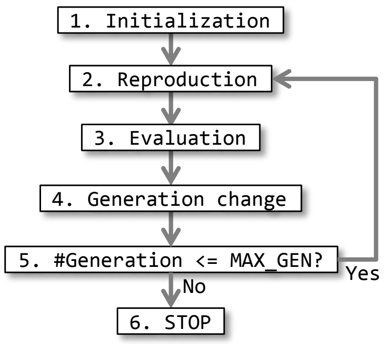


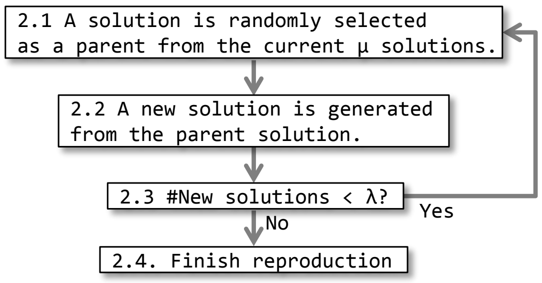
Figure 3. Steps of evolution by means of ES.

Initialization

First, μ solutions $\vec{x}^1$, $\vec{x}^2$, …, $\vec{x}^\mu$ are randomly generated. Values of x_i^j (i =1, 2, …, 162 (or 405); j = 1, 2, …, μ) are sampled from the normal Gaussian distribution with mean=0 and S.D.=1.

Reproduction

New λ offspring solutions are produced by using the current μ parent solutions. Fig.4 shows the steps of reproduction by means of ES.



2.1 A solution is randomly selected as a parent from the current μ solutions.
2.2 A new solution is generated from the parent solution.
2.3 #New solutions < λ?
Yes
No
2.4. Finish reproduction

Figure 4. Steps of reproduction by means of ES.

In the step 2.2 in Fig.4, a new offspring solution $\vec{x}_c$ is generated from the parent solution $\vec{x}_p$ as:

$$\vec{x}_c = \vec{x}_p + \vec{d}, \tag{1}$$

where,

- $\vec{d}$ is also a 162 (or 405) dimensional real vector, i.e., $\vec{d} = (d_1, d_2, \cdots, d_{162}$ (or d_{405})), and
- d_i is a random real value where $|d_i|$ is small or zero.

In our experiment, d_i is sampled from the normal Gaussian distribution with mean=0 and S.D.=1.

Evaluation

In this step, fitness of each solution is evaluated. The fitness in this research is the game score played by each autonomous controller in which values of x_i (i = 1, 2, ..., 162 (or 405)) is utilized as the associated `<array>` values in the XML file. In our experiment with the simulated car racing, we obtain the fitness score by utilizing the `simplerace.Stats` class which gives us the number of waypoints that the car (controlled by the agent specified in the XML file) could visit on 200 trials[7]. Besides, in our experiment with the MarioAI, we obtain the fitness score by utilizing the `ch.dsia.scenarios.CompetitionScore` class. This class gives us the total score of the games with level 0, 3, 5 and 10 stages[11].

Generation change

In this step, next-generation μ solutions are selected from the population of the current μ solutions and the newly generated λ solutions. Two different methods for this selection are known as (μ+λ)-ES and (μ, λ)-ES [1]. As the next-generation solutions, (μ+λ)-ES selects the best μ solutions among the μ+λ solutions, while (μ, λ)-ES selects the best μ solutions among the new λ solutions. We experimentally applied both methods and found that, for the optimization problem in this research, (μ+λ)-ES could evolve better solutions than (μ, λ) ES could.

The steps 2 to 5 in Fig.3 are repeated MAX_GEN times where MAX_GEN is a predefined number of generations.

B. Genetic Algorithm

The steps of evolution by means of GA in our research are shown in Fig.5.

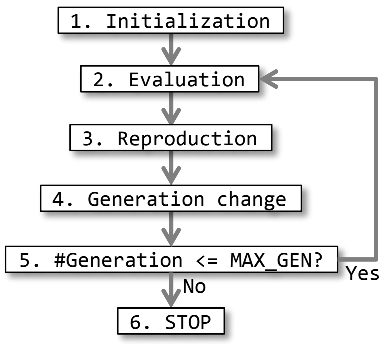


Figure 5. Steps of evolution by means of GA.

The steps 1, 2, and 5 are the same as those for ES.

Reproduction

Figs.6 and 7 show the steps of reproduction and crossover by means of GA respectively. New $(1 - e) * \lambda$ offspring solutions are produced by using the current λ parent solutions. Note that $e*\lambda$ solutions are copied from/to the current/next generation by the elitism operation (so that the reproduction

[11] marioai/src/ch/idsia/scenarios/CompetitionScore.java

process produces only (1−e)∗λ new solutions).

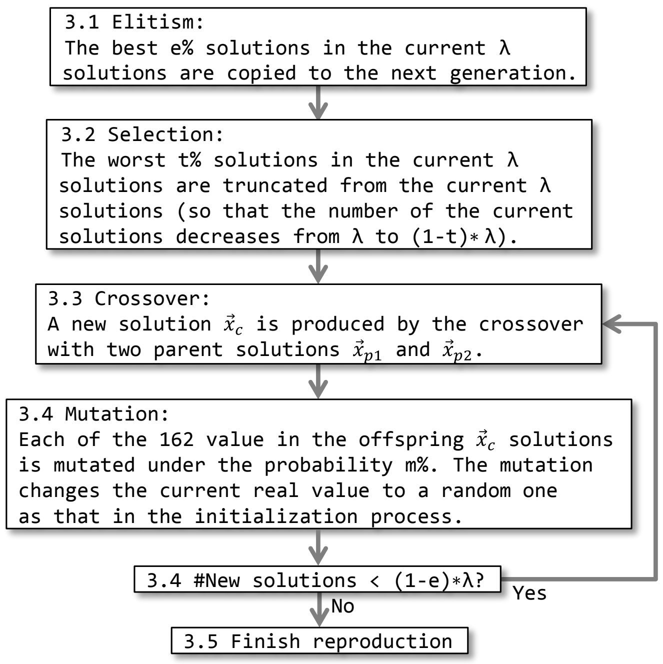

3.1 Elitism:
The best e% solutions in the current λ solutions are copied to the next generation.
3.2 Selection:
The worst t% solutions in the current λ solutions are truncated from the current λ solutions (so that the number of the current solutions decreases from λ to (1-t)∗λ).
3.3 Crossover:
A new solution $\vec{x}_c$ is produced by the crossover with two parent solutions $\vec{x}_{p1}$ and $\vec{x}_{p2}$.
3.4 Mutation:
Each of the 162 value in the offspring $\vec{x}_c$ solutions is mutated under the probability m%. The mutation changes the current real value to a random one as that in the initialization process.
3.4 #New solutions < (1-e)∗λ?
Yes
No
3.5 Finish reproduction

Figure 6. Steps of reproduction by means of GA.

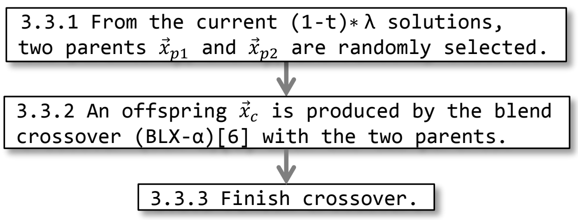

3.3.1 From the current (1-t)∗λ solutions, two parents $\vec{x}_{p1}$ and $\vec{x}_{p2}$ are randomly selected.
3.3.2 An offspring $\vec{x}_c$ is produced by the blend crossover (BLX-α)[6] with the two parents.
3.3.3 Finish crossover.

Figure 7. Steps of crossover by means of GA.

C. ES/GA Hybrids

As hybrids of ES and GA, we switch the application of the two algorithms between the first/last half of the total generations. For example, GA is applied in the first half of the total generations, and then ES takes over from GA in the last half of the total generations.

IV. Experimental Comparisons of Evolutionary Algorithms

A. Applied to CIG2007 Simurated Car Racing

To fairly compare the algorithms ((μ+λ)-ES, GA, and their switching hybrid), we should make consistent the total number of solutions being generated and tested by each algorithm. In our experiment, the total number of generations was set to 1,000, and the population size (the value of λ) was set to 10. Thus, the total solutions being tested was 10,000 (= 10∗1,000). The value of μ for (μ+λ)-ES was experimentally set to 5, and the parameter values for GA were experimentally set to:

- Blend crossover: α =0.5,
- Elitism: e =10%,
- Truncation: t =70%, and
- Mutation: m =1%.

These values performed the best than other values in our experiment.

In the case of GA→ES switch, GA with the above setting was applied in the first 500 generations, and the 10 offspring solutions by GA in the 500th generation were taken over to ES as the parent solutions in the 501th generation (the best 5 solutions among the 10 inherited solutions were actually used as the parents because we utilized (5+10)-ES).

Fig.8 and Table I show the result, where the fitness scores are the best ones among the 10 solutions in respective generations. Fig.8 plots the fitness scores per 25 generations. In Table I, values in the “max” row are the best scores among the total 10,000 solutions by respective algorithms.

Fig.8 and Table I revealed the followings.

- In the first 25 generations, all of the three algorithms could improve solutions rapidly. On the contrary, in the following generations after the 26th, they could improve solutions very slowly.
- It seemed that the solutions by ES resulted in undesirable premature convergence: the solutions were little improved in the latter generations.

These might due to the fact that the optimization problem in this experiment was a large dimensional one (i.e., searching in the 162 dimensional real-valued space) and that the population size was relatively small (i.e., 10)[12]. By the mutation operator GA could explore solutions globally even after the solutions had gathered to some local minimum, but ES could only exploit in a local minimum because ES could not generate offspring solutions that were far enough from their parents in the search space. Besides, the blend crossover operator might contribute for GA to inhibit premature convergence, because the operator could not only exploit between the two parents but also explore outside of the two parents.

In addition, Fig.8 and Table I revealed that, in the last 500 generations, GA→ES switch improved solutions better than ES and GA did. GA→ES switch could evolve the best solution (which scored 3,894 in the racing simulation) among all of the 30,000 solutions. This shows that the ability of ES to searching solutions locally (fine-tuning solutions) in the last generations was better than that of GA. ES applied in our experiment changed only one x_i of the 162 values in a solution $\vec{x}$ (see III.A) so that an offspring solution $\vec{x}_c$ was very close to its parent solution $\vec{x}_p$. This might contribute to exploiting locally better solutions; the 162 values of x_1, x_2, …, x_{162} are weights and biases of a neural network so that conservative

[12] Under the condition that the total number of solutions was 10,000, evolutions by 10 solutions ∗ 1,000 generations were better than evolutions by 100 solution ∗ 100 generations in our experiment.

modifications are appropriate in the final stage of fine-tuning weights and biases.

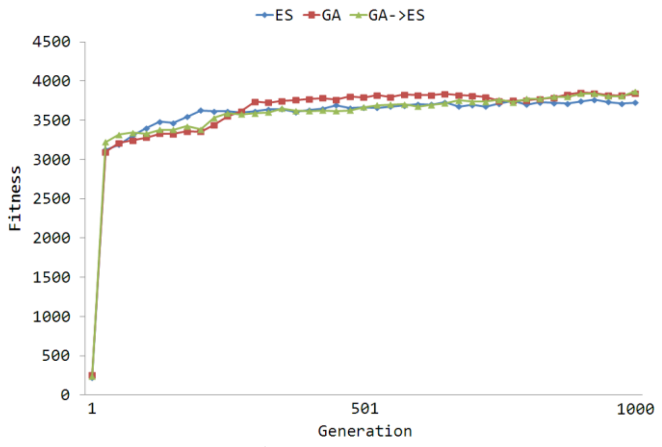


Figure 8. Result of evolutions by the three algorithms.

TABLE I FITNESS SCORES BY THE THREE ALGORITHMS

Generation	ES	GA	GA→ES
1	221	254	240
50	3,191	3,210	3,316
100	3,399	3,282	3,332
500	3,665	3,793	3,667
1,000	3,724	3,840	3,868
max	3,768	3,889	3,894

B. Applied toMarioAI 2009

Again, to fairly compare the algorithms we should make consistent the total number of solutions being generated and tested by an algorithm. In our experiment, the total number of generations was set to 500, and the population size (the value of λ) was set to 20. Thus, the total solutions being tested was 10,000 (= 20 ∗ 500). The value of μ for (μ+λ)-ES was experimentally set to 4, and the parameter values for GA were experimentally set to:

- Blend crossover: α =0.5,
- Elitism: e =10%,
- Truncation: t =60%, and
- Mutation: m =1%.

These values performed the best than other values in our experiment.

In the case of GA→ES switch, GA with the above setting was applied in the first 250 generations, and the offspring 20 solutions by GA in the 250th generation were taken over to ES as the parent solutions in the 251th generation (the best 4 solutions among the 20 inherited solutions were actually used as the parents because we utilized (4+20)-ES). Similarly, in the case of ES→GA switch, ES with the above setting was applied in the first 250 generations, and the offspring 20 solutions by GA in the 250th generation were taken over to GA as the parent solutions in the 251th generation.

Fig.9 and Table II show the result for comparing ES, GA and the two switches (ES→GA and GA→ES), where the fitness scores are the best ones so far at each generation (e.g., the fitness scores at the 250 generation in Fig.9 and Table II show the best scores during the 1st-250th generations) by the respective method.

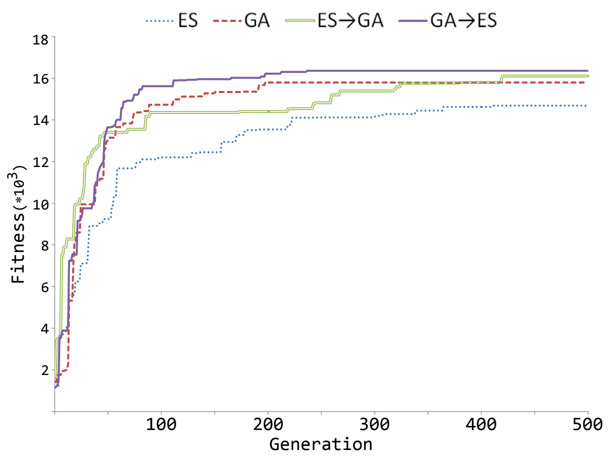


Figure 9. Result of evolutions by the four algorithms.

TABLE II FITNESS SCORES BY THE FOUR ALGORITHMS

Generation	ES	GA	ES→GA	GA→ES
1	1152	1426	1654	1155
25	7101	9948	10212	9284
50	9239	13001	13405	13642
100	12206	14729	14355	15618
200	13540	15793	14398	16218
250	14122	15793	14822	16358
300	14122	15793	15384	16358
400	14625	15793	15791	16358
500	14686	15793	16104	16358

Fig.9 and Table II revealed the followings.

- In the total 500 generations, GA→ES switch found a better solution than the other three algorithms. Note that the score by GA→ES was not improved in the last half of generations. Thus, the best score 16,358 was a result of GA, not of the GA→ES switch.
- At the 250th generation, the scores were better for GA and GA→ES than for ES and ES→GA. Thus, GA outperformed ES in the first half of generations.

These might due to the high dimensionality of the search space and the nature of ES/GA. In our application, the search space is a 405 dimensional real valued one ($\mathbf{R}^{405}$) so that the search efficiency by an algorithm will depend much on its ability of exploration in the early stage of generations. The blend crossover operator might contribute for GA to explore broader area in the search space, because the operator could not only exploit between the two parents but also explore outside of the two parents. The mutation operation might also contribute for GA to explore the space. On the contrary, the search by ES is neighborhood oriented (due to its reproduction process) so that ES was likely to contribute better for exploitation than for exploration.

We expected that GA→ES would perform the best among the four algorithms, because GA→ES would first explore promising area by GA and then exploit the promising area by ES, but the result was not consistent with the expectation. Further investigations are required on balancing the exploration and exploitation by mixtures of evolutionary algorithms. Recently, hybrid uses of evolutionary algorithms and local search algorithms have been researched, known as memetic algorithms [7-9]. Our future work includes application and evaluation of the memetic algorithms.

V. Conclusion

In this paper, we experimentally compared effectiveness of ES, GA, and their switching hybrids (ES→GA and GA→ES) on the optimization problem of the neuro-based autonomous game controllers. In the application to the CIG2007 simulated car racing, premature convergence of solutions was observed in the case of ES, and GA outperformed ES in the last half of generations. The blend crossover operator and the mutation operator of GA might contribute to inhibit undesirable premature convergence. Besides, the GA→ES hybrid (in which GA/ES was applied in the first/last half of generations) evolved the best solution among the entire 30,000 solutions being generated. This result shows the ability of GA in globally searching promising areas in the early stage and the ability of ES in locally searching the focused area (fine-tuning solutions). On the contrary, in the application to MarioAI 2009, GA revealed its advantage in this optimization problem, whereas the expected ability of ES in exploiting (fine-tuning) solutions was not clearly observed. The blend crossover operator and the mutation operator of GA might contribute well to explore the vast search space.

Future work includes application and evaluation of memetic algorithms and other AI/CI methods to this optimization problem.

Acknowledgment

This research was supported by Kyoto Sangyo University Research Grants.

References

[1] H.-P. Schwefel, Evolution and Optimum Seeking. New York: Wiley & Sons, 1995.

[2] D. E. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning. Addison Wesley, 1989.

[3] X. Yao, "A review of evolutionary artificial neural networks," International Journal of Intelligent Systems, vol.4, pp.539–567, 1993.

[4] K.O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," Evolutionary Computation, vol.10, no.2, pp.99–127, 2002.

[5] S. Lucas, and J. Togelius, "Point-to-point car racing: an initial study of evolution versus temporal difference learning," Proc. of IEEE Conference on Computational Intelligence and Games (CIG) 2007, pp.260–267, 2007. http://cswww.essex.ac.uk/cig/2007/papers/2071.pdf

[6] L.J. Eshelman, "Real-coded genetic algorithms and interval-schemata," Foundations of Genetic Algorithms 2, pp.187–202, 1993.

[7] Y.S. Ong, M.H. Lim, N. Zhu and K.W. Wong, "Classification of adaptive memetic algorithms: a comparative study," IEEE Transactions on Systems Man and Cybernetics - Part B, vol.36, no.1, pp.141–152, 2006.

[8] J.E. Smith, "Coevolving memetic algorithms: a review and progress report," IEEE Transactions on Systems Man and Cybernetics - Part B, vol.37, no.1, pp.6–17, 2007.

[9] F. Neri, C. Cotta, and P. Moscato (eds), Handbook of Memetic Algorithms. Springer, 2011.

Hidehiko Okada is currently a Professor with the Department of Computer Science and Engineering, Kyoto Sangyo University, Kyoto, Japan. He received the B.S. degree in industrial engineering and the Ph.D. degree in engineering from Osaka Prefecture University in 1992 and 2003, respectively. He had been a researcher with NEC Corporation from 1992 to 2003, and since 2004 he has been with the university. His current research interests include computational intelligence and human-computer interaction.

Jumpei Tokida and **Yuki Fujii** received the B.S. degrees in computer science and engineering from Kyoto Sangyo University, Kyoto, Japan, in 2012. Their research interests included evolutionary algorithms and autonomous game controller agents.