Car setup optimization via evolutionary algorithms

Carlos Cotta[1], Antonio J. Fernández[1], Alberto Fuentes Sánchez[2], and Raúl Lara-Cabrera[1]

[1] Dept. Lenguajes y Ciencias de la Computación, ETSI Informática, University of Málaga, Campus de Teatinos, 29071 - Málaga, Spain.
{ccottap,afdez,raul}@lcc.uma.es
[2] elvetto@hotmail.com

Abstract. Car racing is a successful genre of videogames, as proved, for example, by the racing simulator saga, Gran Turismo. In this genre of games, players not only race but they are also involved in the process of setting up the car, assuming the role of a technician/mechanic/engineer. Generally, this configuration deals with a large set of parameters that range from the amount of fuel loaded into the car to the tire pressure and type. This article compares different proposals for optimizing this process using evolutionary computation techniques to make several suggestions for a simulated international competition for car racing setup optimization.

1 Introduction

Artificial intelligence (AI) in games has become very important so much so that there are even international conferences that only focus on this topic, or provide tracks and special sessions devoted to this area. These conferences frequently include competitions in order to compare and discuss different proposals of AI applied to videogames.

The GECCO conference proposed simulating the days before a race, when mechanics and drivers work together on the car setup to discover which car setup is the best for the circuit. The goal of this competition was to design an evolutionary algorithm to search for the best car setup in a racing car simulator thereby replacing the team of mechanics and drivers, in the search for the best car setup in a racing car simulator. This article describes different algorithms and analyzes their performance within the framework of the aforementioned competition.

There is a related competition, the *Simulated Car Racing Championship*, where each participant has to submit a controller to drive a car. This is a very active competition with a lot of controllers submitted and a wide range of AI techniques are used to develop them. [1] presents an autonomous racing car controller with several context-dependent behavioral modules the parameters of which have been optimized by evolutionary strategies. For example, [6] exhibits a tuned-by-hand modular architecture combined with a simple fuzzy system,

while in [3, 7] the authors use fuzzy systems in order to implement the controller. Another example that uses an on-line neuroevolution system is presented in [2].

2 The competition

The competition is linked to *TORCS*[3] (The Open Racing Car Simulator), which is a multi-platform racing simulator, that can also be used as both a game in itself and as a research platform because it includes artificial players and comparisons between them. The competition, called "Optimizing Car Setup", is based on 3 tracks or grand prizes. The participants have to provide evolutionary algorithms to optimize the best car setup for each track. The contest is divided into two consecutive phases: optimization and evaluation. During the optimization phase, the evolutionary algorithm should optimize the car parameters for a fixed amount of playing time (i.e. game tics): 10 million tics. Then, the best car setup is used to compete for a certain number of tics (time units), the maximum distance covered by a car using this configuration. Then, all the distances of the participants are compared with each other and these participants are ranked according to the distance reached; the player with the longest distance earns 10 points, while the second and third earn eight and six points, respectively. The rest of the players from the fourth to the eighth position receive five to one point, respectively.

A real-valued vector of 50 elements represents the parameters of a car setup. The competition software provides an API to evaluate these parameters in a race and returns useful information such as the best lap time, the maximum speed, the distance raced and the damage the car has suffered during the race. Through the API, it is possible to specify the amount of game tics to use in the assessment of a car setup. The game tics spent for an evaluation will be subtracted from the total amount of playing time available. The evaluation process ends when it reaches 10 million games tics or it spends more than 2 hours of computing time.

2.1 The architecture

TORCS is available as a standalone application, in which the bots (non-human players, in other words, the cars) are compiled as separate modules of the game and then loaded into the main memory prior to the start of a race. The cars available in TORCS have several parameters that should be optimized for each circuit, such as the angle of the wings or the suspension's parameters. These parameters make TORCS an ideal testing framework for optimization algorithms based on evolutionary computation. However, this architecture has two problems: (a) there is no separation between the bots and the simulation engine so that bots have full access to all the data structures that represent the track and the car; this information might be used to create game cheating Artificial Intelligence (AI), and (b) TORCS architecture restricts the choice of the programming language used to develop the bots.

[3] http://sourceforge.net/projects/torcs/

The competition software modified TORCS to cope with these problems and it extended the original architecture as follows:

1. TORCS is structured as a client-server application: the bots run as an external process and connect to the race server through an UDP connection.
2. It is possible to change the car parameters during the race; in the original TORCS the parameters were loaded at the start of each race.
3. The competition software creates a physical separation between the optimizer and the server of the race, building an abstraction layer, which removes the restriction of which programming language is used for the bots. Moreover, it only allows access to a set of parameters defined by the contest designer, and it prevents the possibility of exploiting some specific domain knowledge.

An optimization process involves a server and a client. At the beginning of the process, the client identifies the server and establishes a connection between them; then the latter sends some information to the former, namely the number of parameters to be optimized and the number of available ticks. After this, the simulation begins and the individuals sent by the client are evaluated.

For each evaluation, the client sends the server a request to evaluate a list of values of the parameters to be optimized as well as the duration of the evaluation expressed as a number of game tics. The evaluation (i.e. the race) is in itself a black box and the designer does not have access to it; this aspect of the game raises an interesting and hard challenge in the design of the game's AI. As soon as the evaluation is complete, the server returns the result to the client encoded as four values: *distraced* (i.e., distance raced), *topspeed* (i.e., maximum speed reached during the race), *damage* (i.e., damage to the car), and *bestlap* (i.e., best lap time)

Once the simulation is over, the client must send the server the best solution found. The final solution is the only result considered for the competition. During the competition, the final solution produced by each competitor will be compared with the others to assess the best optimization strategy and this will be scored according to the rules shown in the previous section.

3 A Steady-state Algorithm

The first proposal is an evolutionary steady-state algorithm. Individuals are selected in pairs from a population, using a binary tournament method of two parents (i.e. the parent with the best fitness is selected), subsequently they are combined into one individual and mutated (with a probability of 1.0). This new individual is added to the new population and the whole process is repeated a certain number of generations.

Each parameter was a real value and each of them was encoded as a 10-bit binary value. There were 50 parameters to optimize so each individual of the population was an array of 500 bits (50 * 10-bit real genes). With this

representation, the implementation of the converter had a way to translate the binary representation to a real-value vector.

On the other hand, we had to establish a fitness function to evaluate the quality of each individual. As mentioned above, the server returns four values: *bestlap*, *distraced*, *topspeed* and *damage*. At this point, we considered a single fitness to optimize:

$$C_1 * \text{distraced} + C_2 * \text{topspeed} + C_3 * (1000 - \text{bestlap}) + C_4 * \text{damage} \quad (1)$$

C_i are real constants that provide weights to the four values to optimize in Eq. (1). The value of this fitness corresponds to the mono-objective version of our evolutionary algorithm.

4 Experimental Analysis

This section includes a description of the experiments conducted in order to evaluate the performance of the mono-objective algorithm that was described in the last section. An analysis of the performance of this proposal during the contest held at the conference EVO-GAMES 2010 is also presented here.

For the experiments, we decided to set the following parameters: 10 runs of the algorithm with a population of 50 individuals and 20 generations. The number of tics for the simulation was set to 10000 and 5000 for the evaluation of each individual in the population and the evaluation for each solution of the optimization respectively.

For a competitive solution, first we had to find the appropriate values for the constants C_i (for $i \in [1, 4]$) in the expression (1), and for that we considered various combinations of values. The results of the fitness functions were tested on four different circuits (i.e., Speedway, wheel1, Olethros, and E-track 4). Table 1 shows the fitness function that were proposed for the study. As there are a lot of combinations to consider, we only present the data from some of these. The left hand column shows the fitness function used to train our proposal and the right hand column shows the total score after training in the four aforementioned circuits.

Prior to running the experiments, we had to find the appropriate values for the constants C_i in the fitness definition (1). Table 1 shows the score obtained from different combinations of these values.

The best performance of the mono-objective algorithm was achieved with the combination of values ($C_1 = 0.6$, $C_2 = 2.5$, $C_3 = 0.15$ and $C_4 = 0.05$), so the mono-objective algorithm's fitness function was the following:

$$0.6 * \text{distraced} + 2.5 * \text{topspeed} + 0.15 * (1000 - \text{bestlap}) + 0.05 * \text{damage} \quad (2)$$

We submitted the mono-objective evolutionary algorithm to the EVO-* competition, which specified the following rules:

Fitness Function (1)	Total Points
$0.6*distraced+2.5*topspeed+0.15*(1000-bestlap)+0.05*damage$	29
$0.6 * distraced + 3 * topspeed - bestlap - 0.1 * damage$	19
$0.7 * distraced + 1.5 * topspeed - 0.1 * bestlap - 0.05 * damage$	19
$0.6 * distraced + 1.5 * topspeed - 0.25 * bestlap - 0.05 * damage$	17
$0.25 * distraced + 0.25 * topspeed - 0.25 * bestlap - 0.25 * damage$	12
$0.25 * distraced + topspeed - 0.25 * bestlap - 0.25 * damage$	12
$distraced + 0.25 * topspeed - 0.25 * bestlap - 0.25 * damage$	8
$distraced + topspeed - 0.25 * bestlap - 0.25 * damage$	8
$0.6 * distraced + 0.25 * topspeed - 0.15 * bestlap - 0.05 * damage$	6
$0.25 * distraced + 10 * topspeed - 0.25 * bestlap - damage$	5
$0.6*distraced+2.5*topspeed+0.15*(1000-bestlap)-0.05*damage$	5
$0.25 * distraced + 0.25 * topspeed + 0.25 * bestlap + 0.25 * damage$	0
$0.25 * distraced + 0.25 * topspeed - 0.25 * bestlap + 0.25 * damage$	0
$0.25 * distraced + 0.25 * topspeed + 0.25 * bestlap - 0.25 * damage$	0

Table 1. Score obtained by using different C_i values in the fitness function

- The contest involved 3 tracks.
- Only 22 parameters out of 50 to optimize.
- The simulation time was 1 million games tics.
- The optimization algorithm ran 10 times in each circuit.
- The best solution was scored according to the distance covered in 10000 game tics.

The results for each of the three tracks considered in the competition are presented in Table 2 (note that Fuentes/Cotta/Fdez/Cab is our mono-objective algorithm).

Competitor	CG Track	Poli-Track	Dirt-3	Distance	Points
Muñoz (MOEA)	9831.83 (10)	7654.01 (6)	6128.29 (8)	23614.13	24
García-Sáez (PSO)	8386.77 (6)	7979.86 (10)	5021.41 (5)	21388.04	21
Walz (PSO)	8408.25 (8)	7304.54 (5)	5336.88 (6)	21049.77	19
Fuent-Cotta-Fdez-Cab (GA)	7553.21 (4)	5931.47 (4)	6263.40 (10)	19748.08	18
Muñoz-Martín-Sáez (EA)	8167.60 (5)	7718.36 (8)	4629.33 (4)	20515.29	17

Table 2. Distance Average and final classification in the 2010-competition: Distance(Points)

Each track was won by a different competitor. Our driver won in *Dirt-3* and the difference between the second and the fourth classified, was really small (only 3 points). The winner of the competition used a multi-objective algorithm (MOEA) to optimize the four parameters considered in the competition (i.e., distraced, topspeed, bestlap and damage).

5 Multi-objective Evolutionary proposals

We also considered two multi-objective algorithms that are described next. Initially we considered several possibilities and finally decided to create two multi-objective algorithms using SPEA2 [8], specifically the implementation of this algorithm provided by the ECJ [4] library.

In order to use a multi-objective algorithm, it is neccessary to add a new field to the population that corresponds to Pareto file size. This file corresponds to a part of the subpopulation and the size of the file cannot be larger than the size of the population.

Another peculiarity of the SPEA2 implementation from the ECJ library is that the fitness includes two elements: an array of floating point values representing the fitness values (0.0 is the worst and infinity is the best) and another value representing the SPEA2 fitness that identifies the individual status. The individuals are sorted by fitness value, so the best individual is the one with the lowest value.

In addition to this algorithm, we also propose a modification of SPEA2 multi-objective algorithm that is capable of finding a set of characteristics for the best individual in the Pareto archive, using a feature selection algorithm.

The feature selection algorithm used for this study consists of a Principal Component Analysis (PCA) [4, 5]. To implement PCA, we used the method included in Matlab[5]. This method, which has been applied to each candidate, returns several variables among which a set of characteristics can be found.

The method to obtain the pattern vector is as follows. Candidate solutions are executed ten times in four tracks/races and we consider a Pareto file with size 10. Each individual for each circuit should have a file of 100 individuals resulting from the simulation. Then, the Pareto front of non-dominated individuals is computed.

Each of these individuals are then evaluated in all four tracks, obtaining their distances raced. These distances are grouped into a matrix that is passed to the Matlab function in order to calculate the principal components. These values are involved in the process of selecting the best individual.

6 Experimental Analysis including MOEAs

In this section, we have compared our results obtained in the aforementioned experiments and those obtained by the participants of the competition held in GECCO 2009.

For the multi-objective evolutionary algorithm, we consider a population of 50 individuals, 20 generations and 10000 game tics for the simulation and evaluation. Different candidates were run 3 times on the same tracks that were

[4] http://cs.gmu.edu/ eclab/projects/ecj/
[5] http://www.mathworks.es/help/toolbox/stats/princomp.html

previously used for single objective. Table 3 shows some of the fitness combinations that were considered (first column) and the score obtained by SPEA2 to optimize in this context.

Fitness combinations	Total
$f[0] = \min$ bestlap, $f[1] =$ Single-objective fitness	36
$f[0] = \min$ bestlap	32
$f[0] = \min$ bestlap, $f[1] = \max$ distraced	27
$f[0] = \min$ bestlap, $f[1] = \min f[0] -$ damage	16
$f[0] = \min$ bestlap, $f[1] = \max$ topspeed	15
$f[0] =$ distraced	13
$f[0] =$ distraced, $f[1] =$ topspeed, $f[2] = \min$ bestlap, $f[3] = \min$ damage	13

Table 3. Results obtained using different fitness combinations

The best combination for our multi-objective algorithm was minimizing Bestlap and maximizing the fitness used by the mono-objective algorithm explained above.

The experiments with a variation of the multi-objective algorithm (based on PCA) had the following characteristics: two versions of the proposals were considered: one with a Pareto file size of 5 and another with a size of 10. Each version, in turn, had two versions: one version with the fitness previously considered in SPEA2, and the second one considered by directly optimizing the four main parameters of the competition (i.e., distraced, topspeed, bestlap, and damage). The evaluation was performed with the same scenario as in the previous experiments. Table 4 contains all the results.

Drivers	Total
4 fitness and 5 not dominated	25
4 fitness and 10 not dominated	27
2 fitness and 5 not dominated	36
2 fitness and 10 not dominated	28

Table 4. Score obtained using different parameters

The best variation of the multi-objective algorithm is the result of combining a Pareto file size 5 with the fitness: minimizing bestlap and maximizing the fitness used by the mono-objective algorithm. Table 5 shows the overall score obtained by the three proposals described above and the three drivers submitted to the contest GECCO-2009. In this table, the best performance corresponds to our SPEA2 multi- objective algorithm (without PCA).

Note that the winner in the evo-*2010 competition, as displayed in Section 4, is a multi-objective algorithm (MOEA) similar to our best proposal, as shown here. In fact, our two multi-objective proposals were much more competitive

Driver	Speedway	ETRACK	Olethros	Wheel	Total
Multi-objective	10	5	8	8	31
V&M&C	4	8	5	10	27
Jorge	8	4	10	4	26
Multi-objective PCA	3	10	6	6	25
Single-objective	5	6	4	5	20
Luigi	6	3	3	3	15

Table 5. Overall score of the proposals submitted to GECCO 2009

than the single-objective proposal that was sent to the competition. So we are led to think that either of these two proposals could have obtained better results.

6.1 Additional comments

Three parameters were disabled in the experiments: fuel consumption, car damage and the time limit per turn. The reader may wonder why and the answer is simple: the objective of this contest was to find the best solution for a car, but, during the training, the car may suffer variations. One of these changes is the damage done to the car. If a car collides with an object it is likely to deform the chassis and all the data obtained from then may not be reliable because it is conditioned by the crash. Fuel consumption leads to the same data reliability problem. In each lap the car consumes fuel and in turn it weights less, becoming easier and faster to drive.

7 Conclusions

In this article, we have considered different proposals based on evolutionary computation to set up a car in a racing simulator. In order to achieve this, we have used an international contest in which there were several parameters to be optimized.

The nature of the competition and experimental analysis has shown that multi-objective evolutionary algorithms are a good solution to the problem. Focusing on our proposals, the best proposal is based on the SPEA2, although it is true that we can not underestimate the work done by the single-objective algorithm. That defined the fitness function and this was used in our best multi-objective algorithm as one of the objectives to be optimized.

The results obtained by a variant of our multi-objective algorithm with search patterns was good (better than the results obtained by the single-objective algorithm) and this makes us think that the calculation of the principal components is a good technique to take into account for the future, although it requires further in depth studies.

There are several lines open such as trying new fitness functions for our single-objective algorithm or searching a classification algorithm to find better patterns for the multi-objective.

Acknowledgements

This work is partially supported by Spanish MICINN under project ANYSELF[6] (TIN2011-28627-C04-01), and by Junta de Andalucía under project P10-TIC-6083 (DNEMESIS[7]).

References

1. Butz, M., Lonneker, T.: Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on. pp. 317–324 (2009)
2. Cardamone, L., Loiacono, D., Lanzi, P.L.: On-line neuroevolution applied to the open racing car simulator. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on. pp. 2622–2629 (2009)
3. Ho, D., Garibaldi, J.: A fuzzy approach for the 2007 cig simulated car racing competition. In: Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On. pp. 127–134 (2008)
4. Jolliffe, I.: Principal Component Analysis. Series in Statistics, Sprimger, New York, USA, 2nd edn. (2002)
5. Krzanowski, W.: Principles of Multivariate Analysis: A User's Perspective. Oxford University Press, New York, USA (1988)
6. Onieva, E., Pelta, D., Alonso, J., Milanes, V., Perez, J.: A modular parametric architecture for the torcs racing engine. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on. pp. 256–262 (2009)
7. Perez, D., Recio, G., Saez, Y., Isasi, P.: Evolving a fuzzy controller for a car racing competition. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on. pp. 263–270 (2009)
8. Zitzler, E., Laumanns, M., Thiele, L.: Spea2: Improving the strength pareto evolutionary algorithm. Tech. rep., Swiss Federal Institute of Technology (ETH) Zurich (2001)

[6] http://anyself.wordpress.com/
[7] http://dnemesis.lcc.uma.es/wordpress/