Annals of the University of Craiova, Mathematics and Computer Science Series
Volume 39(2), 2012, Pages 309–321
ISSN: 1223-6934

Optimization of Gear Changing using Simulated Annealing

Alexandru P. Becheru and Cătălin Stoean

Abstract. The current paper puts forward a study conducted on a racing simulator, with the purpose of finding appropriate revolutions per minute (RPM) values for changing the gears of a car. The intricacy of the problem is given by many factors: it has 12 dimensions, 6 values for increasing and 6 for decreasing the gears, a circuit may contain straight and twisty parts, so distinct RPM values may be optimal for different sections, the move to another circuit makes the previous parameters suboptimal, while adding other competitors increases the amount of noise. Overall, we probably have all the ingredients of a real-world problem and we tackle it with two approaches that are economical as concerns the number of simulations needed and conduct to some interesting findings. We consider as study cases 2 different circuits, and racing alone and against 2 opponents. This paper is a second approach to optimize gear changing values used in the game TORCS. Previously we used Hill Climbing algorithm to give proof that gear changing learning does reduce lap times. Now by using Simulated Annealing we improved even more the lap times and got better correlated sets of gear values.

2010 Mathematics Subject Classification. Primary 68T05; Secondary 68T20.
Key words and phrases. Artificial intelligence, machine learning, simulated annealing, optimization, simulation, gear changing.

1. Introduction

Humans are intelligent beings, capable of using all sorts of tools to manage different tasks. Computers have become the ultimate tool, used all over the world in many different circumstances. In fact, advanced societies would not exist without them. Just imagine doing all the calculations needed for the New York Stock Exchange by hand. They help us in almost every step of our life, from cooking to buying gifts. One may ask why computers had such a stunning success in such a short time? Because they always help in shortening our work, for entertaining or even in taking decisions. Computers do that, calculations that would have taken ages are now done in a matter of seconds, repetitive tasks which all of us hate are in their care, data managing software helps us understand real life scenarios while advising us on taking up the best solution.

In the last decades, we did not even notice how, slowly but surely, AI integrates into our lives: web search engines most of the time provide the best answers on the first page, shopping websites come with the most appropriate buying suggestions, GPS navigation devices become more and more indispensable, systems for automatic patient diagnosis based on various indicators are used worldwide, various driving assistance mechanisms are integrated into cars and so on. Such applications became so common in everyday life, that we do not even consider them as having intelligent methods behind. However, there are also very enthusiastic applications, in which robots play the violin, grab objects, run or look and move almost exactly like a

Received November 29, 2012.

human being, that still seem extracted from sci-fi movies. How about a car that drives itself in traffic? Although that appears a bit unrealistic, even since October 2010 we have witnessed cars driving over 200 000 km with only occasional human control and over 1 500 km without any human intervention [6]. Also in the United States of America the legislative of Nevada introduced a law which allows driverless cars to be tested in some designated area [4]. Therefore, we are not far from seeing many such driverless cars around us, which indicates the currently suggested research topic as a very hot one.

One area where computers are indispensable is simulating real life scenarios. Different scenarios can be tested in a fraction of the time normally needed and with a very low cost, thus helping the process of decision making. Even if we cannot simulate every parameter of the real world, we can see the big picture, which is a valuable feedback. Gaming is one important part of our digital world, used for entertaining and recently for real world simulations. Yesterday games were considered child play, nowadays researchers use them to solve real life problems. Real life simulation on modern computers gives an excellent feedback on a new idea in a fraction of the time used before, and with a very low budget.

From youngsters to elders, everybody wants to have a better car. In the real life, it cannot be easily customizable as the changes cannot be immediately performed and tested. Nevertheless, customization is possible with considerable investments as regards both, money and time. This is where computer games come in help, giving the player a taste of what it's like race like a professional and offering the possibility to customize your own car from scratch. Every parameter of the car being decisive in winning or losing a game and the effects of the changes can be observed in a matter of seconds or minutes after some fast simulations.

The platform that is used for racing simulations in the current work, The Open Race Car Simulator (TORCS) [18], allows us to configure a great amount of parameters for the car. We focus however on only a part of these settings and search for a good gear change calibration in various conditions. To the best of our knowledge, there is not such a specific study for the moment.

In the following sections will talk about the previous work done in this domain. The software that we used is described in the Tools section. Finally we will present the some results and the concluding regards.

2. Previous work and motivation

TORCS is an AI research dedicated platform that has its routes in the academic world being developed by the *University of Wurzburg* and the *Politecnico di Milano*. This platform is used in 3 competitions every year starting from 2010 held during the following major conferences [19]: EVO*, ACM GECCO (Genetic and Evolutionary Computation Conference), and IEEE CIG (Computational Intelligence in Games), although previous competitions existed. Besides being awarded as *the most capable open race car simulator* by *Linux Journal* [3] in 2007, it is also a very enjoyable game to play.

During the last years there have been a lot of attempts to create artificial intelligent agents capable of racing. Trying to imitate another car war carried out in [7]. Although the imitation of a human controlled car din not have good result, they succeed in imitating with good results the 2008 Simulated Car Racing Championship. But they had problems with gear changing learning. Rule based approaches where

tried in 2 papers [9, 10]. They where able to produce sets of rules for different circuits that reduced the lap time. Also ant colony implementation was tried in [2]. More methods are incorporated in Mr. Racer, the champion of the 2011 championship, like learning the target speeds depending on a parts of the circuit, function of curvatures [14]; next, the focus was on noise removal from the environment [12]. There are numerous other papers focused on developing a great artificial driver, but to the best of our knowledge all of them are concentrated on other features than gear changing, and used some preset values of some simple if-then rules for these 12 parameters. This encouraged us to investigate wether some potential speed boost is hidden within these parameters.

In a previous attempt regarding gear change learning [1], we used the Hill Climbing (HC) algorithm to find out proper values for RPM. After tweaking the algorithm to suit our scenario we were able to reduce the lap time even by 30%. After several tests we got enough data to conclude that gear changing should be a matter of interest for everyone developing intelligent agents running in TORCS. But we also encountered some drawbacks: on a fast circuit (that resembled a highway) little or no progress was done, also in some scenarios data sets were not correlated.

In this paper we employ the Simulated Annealing (SA) algorithm [11] with the aim of improving the correlation between data sets obtained for slightly different configurations, and reduce the lap time. By having a correlation of data sets for different settings of the same car and the same circuit, proper gear values are found by mathematical meanings. The aim on the long term is to succeed in creating a gear box that adapts to the environment and to the driving mode automatically.

3. SoftwareTools

The current section describes the most important pieces of software used in the experiments. Besides the software described in the following subsections, we used Microsoft Excel, Dev-C++ [20].

3.1. The Open Race Car Simulator. The most important piece of software is represented by the car racing simulator TORCS [17, 18], which represent an amazing research platform. According to [8], it has the following advantages:

- Besides being a good car racing game (see Figure 1), it offers a fully customizable environment, like the ones typically used by computational intelligence researchers for benchmark purposes.
- It features sophisticated engine (aerodynamics, fuel consumption, traction, etc.) as well as 3D graphics engine for the visualization of the races.
- It was not conceived as a free alternative to commercial racing games, but it was specially devised to make it as easy as possible to develop your own controller.

For research purposes, each race can be simulated in a matter of seconds, letting out the visualization and just returning the outputs (average speed, top speed, fuel consumption, covered distance etc.), so a huge number of rapid race simulations under various car configurations can be obtained in a matter of hours.

3.2. Simulated Car Racing Championship. The Simulated Car Racing Championship [19] (SCRC) is a competition consisting in 9 races taking place during 3 conferences. The purpose is to develop agents that can race on previously unknown circuits. Points are awarded based on the position at the end of the race. In the end, the car with the most point wins. There is no boundary of implementation, the

FIGURE 1. TORCS screenshot.

agents can be hard coded, they can use methods from computational intelligence, or whichever approach imagined.

The software developed for this competition comes in addition to TORCS making the combined platform better. Here are some key features:

- The obtained problem is very similar to the real world applications.
- The platform offers the possibility for the human players to interact with AI players.
- There is the possibility to study and see in action many agents developed.
- Real time racing.
- Fair competition.

The car is controlled only by a set of sensors and actuators [5]. Sensors give feedback to the agent during the race, i.e. opponents position. In fact, within TORCS, there were so many outputs from the car to the agent, so much information is fed, that it makes it unreal, as in the real world one cannot have access to all this data. To make things more real, within the competition, Gaussian noise is fed into this sensors. Actuators also give feedback, but they offer the possibility to change the value of the feedback, this being the method by which the car is controlled. An example of actuator is acceleration, by modifying it the cars speeds up or down.

This software platform is client-server based. This makes developing agents more easy, as the user does not have to configure everything and the simulations happen faster. The agent is run on a computer as the client, meanwhile all the physics engine is done on a server which communicates with the client. This packages includes a basic agent, which drives the car at city speeds.

3.3. AutoIt. AutoIt [21] is a BASIC (Beginners All-purpose Symbolic Instruction Code) like programming language, conceived to automate the Graphical User Interface (GUI) of Microsoft Windows, but in can be used for general programming. It simulates the presence of a user by controlling the mouse and the keyboard, thus making possible the manipulation of program windows and the automatization of user programmed tasks.

It was initially developed to automate the configurations necessary for millions of computers before being shipped to clients. Over time other modules were added for

better inter-operability with other programs which run on the Microsoft Windows platform, i.e. Microsoft Excel. It also has the possibility to create GUIs by using the following components: edit boxes, check boxes, list boxes, buttons, status bars and combo boxes. This feature makes it more user friendly.

4. The Proposed Approach

The optimization problem we treat herein is to find a proper gear change values in order to reduce the lap time, so we employ and algorithm that can rapidly search in vast data spaces and retrieve results that minimize a given function. More concrete, the search space is composed out of the possible values that the gears can take. The car has 6 gears and we have to set values both for shifting up and down, so we have to configure 10 gears, each can take a value from 1 to 8000. Of the 10 gear, 5 are for shifting the gear up and 5 for shifting the gear down. For example, the second gear up represents the RPM value of the motor when we shift from the 2^{nd} to the 3^{rd} gear. The objective function that needs to be minimized is the lap time. Actually the function is the duration of the race itself, the lap time being the indicator of how good or bad was the race in comparison with another.

With this idea in mind we set ourself to develop a system where we could run several races with different sets of gears. The feedback from each race will serve as the input for the approaches which will adjust the gear values for the next race. Finally we should be able to find gear values that make the car faster.

In this section we present the software that integrates the algorithm with the SCRC software. Also we discuss the adjustments to the SA algorithm made to adjust it to our scenario.

Next, the following concepts will be used throughout the article:

- A *generation* is a race consisting in a fixed number of laps; in our experiments, a generation is actually one lap.
- A *series* represents a preset number of generations with the same configuration.
- By *configuration* we mean choosing the track, the algorithm and the number of opponents.

In our first tests we intended to establish an optimum number of generations per series and we found it to be 200 generations. If this number was smaller then we may not always reach the vicinity of good results, but if it was bigger, no major improvement was achieved.

4.1. SCRC and Algorithm Integration. The program that integrates the HC and SA algorithms with the SCRC[1] offers the following facilities:

- A GUI for ease of use (see Figure 2). It offers the possibility to setup the tests and to continually see the results state. It is composed of two frames with different functionalities:
  - The top frame is made up of two tabs, where the user is able to interact with the software. The *General* tab offers the possibility to start testing or to stop it after the current generation or the current series. The *Options* tab is where the user sets the algorithm to be used, the name of the file where the tests data are saved, also the number of series and the number of generations per one series.

[1]Available for download and use `http://inf.ucv.ro/~cstoean/diploma.html`

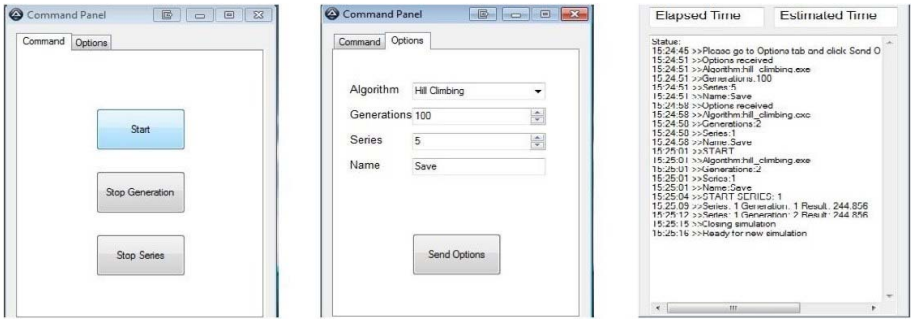

FIGURE 2. From right to left the first two images represent *General* and *Options* tabs of the top frame, while the last one depicts the bottom frame with intermediary and concisely results.

  - The bottom frame has the purpose of offering information regarding the tests: the elapsed and estimated time, the current generation result, the number of the current series and generation, and also environment messages.
- TORCS and SCRC software are integrated together with the optimization algorithms. Therefore, a user only has to hit the start and stop buttons. Automatically the programs and the algorithms are started, stopped and paused without any further intervention from the user.
- Data management, by assuring the flow of data between programs and by saving those information in a file for later use. All of this is done by:
  - Retrieving lap times after each generation, by extracting them from an XML file created by TORCS.
  - Transferring data from the algorithm towards the agent who controls the car. To make sure that the data is correct and successfully transferred, the algorithm and the the other pieces of software involved are paused and started alternatively. When a race is running, the algorithm waits for it to finish, next the race is paused and the time is retrieved and passed to the algorithm. Then the algorithm starts, reads the new data and after doing the necessary calculations passes the new set of values to the agent controlling the car, and pauses.
  - Saving the gear values of each generations among with their corresponding results to a Microsoft Excel file for ease in further manipulation. Each series of a test is written in a different sheet and, after each series is finished the best result, the worst and the average one in that series are written, so that the user can see the big picture when the data file is opened.

4.2. Adjusting the Simulated Annealing Algorithm. For adjusting the employed algorithms, we ran about 5000 generations. The races were simulated on the same computer, having a 2.8 GHz dual core processor, 4 GB of RAM and a graphics card with 256 MB of shared memory. For each generation there were necessary around 30 seconds.

In the description of the Algorithm 1, the following notions are used:

- vectorRpmOptim: is the vector of RPM values with the best result. The #Result represents the lap time associated with those values.

- vectorRpm: is the vector that is constructed out of the vectorRpmOptim and is due to be used on the next race. Also #Result attached represents the result associated with it.
- X: is the distribution parameter that is computed in Algorithm 2.
- plan: value specific to the SA algorithm by which worse result acceptance is controlled.

The plan of the SA method is to be elitist in the first generations (plan is set to 1) and then allow worse solutions depending on the difference between the previous lap time and the new one. If the difference is minor for the worse, then it has higher chances to be selected and as the difference increases, the chances to move to the worse setting decrease (lines 2, 17 and 19). As it can be observed in line 10, changes in the RPM values are more likely to appear if the lap time is high and the better the time (meaning the lower the value), the higher the chances to appear only small perturbations in the current solution.

Algorithm 1 Simulated Annealing

1: $mutationStrength \leftarrow 500$
2: $plan \leftarrow 1$
3: $vectorRpmOptim \leftarrow randomly\_generated\_vector\_with\_values\_between[1, 8000]$
4: $vectorRpmOptim\#Result \leftarrow 1000$
5: $vectorRpm\#Result \leftarrow 0$
6: $vectorRpm \leftarrow vectorRpmOptim$
7: $currentExecution \leftarrow 1$
8: **while** $currentExecution \leq maximum\_number\_of\_executions$ **do**
9: **for** $i = 1 \rightarrow length(vectorRpm)$ **do**
10: **if** $randomly\_generated\_number\_between[1, 1000] \leq 2{*}vectorRpm\#Result$ **then**
11: $deviation \leftarrow mutationStrength * X$
12: $vectorRpm[i] \leftarrow vectorRpmOptim[i] + deviation$
13: **end if**
14: **end for**
15: During the tests the algorithm is paused here, waiting for TORCS to finish the race and retrive the lap time
16: $vectorRpm\#Result \leftarrow LapTime$
17: $plan \leftarrow plan * 0.98$
18: $difference \leftarrow vectorRpm\#Result - vectorRpmOptim\#Result$
19: **if** $(difference \leq 0)$ or $(randomly\_generated\_number\_between[1, 10000] \leq exp\left(\frac{difference}{10*plan}\right))$ **then**
20: $vectorRpmOptim \leftarrow vectorRpm$
21: $vectorRpmOptim\#Result \leftarrow vectorRpm\#Result$
22: **end if**
23: $currentExecution \leftarrow currentExecution + 1$
24: **end while**

4.2.1. *The distribution of the mutation.* The mutation is the mean by which the set of gears is modified. The distribution of the mutation as shown in [1] has a big impact on the results. We compare herein the normal distribution with the proposed one with SA algorithm. As noticed, in Figure 3 the normal distribution's best results came close to 100, but the one proposed by us gave better results having reached 93. In some

similar tests the normal distribution managed to give better results, but overall the proposed distribution gave better ones. Algorithm 2 shows the calculations used for the proposed distribution. Note that X can be positive or negative, so the values of the RPM in line 12 of Algorithm 1 can be increased or decreased, correspondingly.

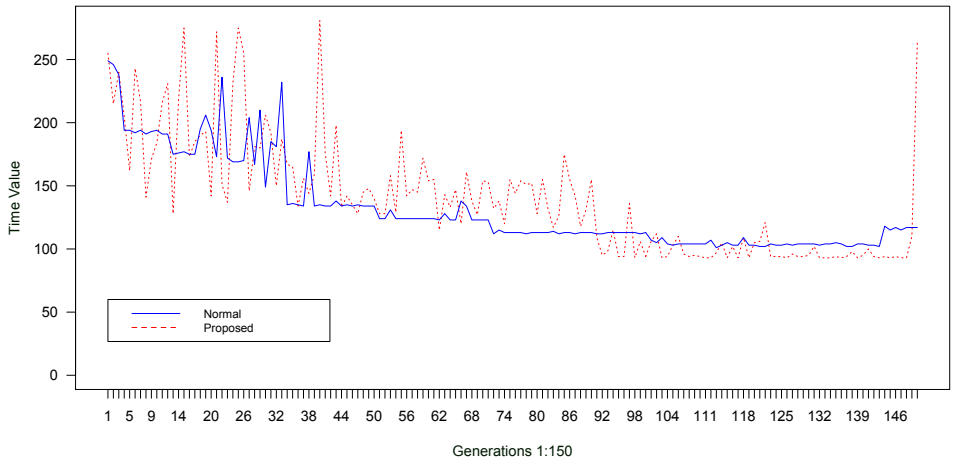

FIGURE 3. Normal vs proposed distribution.

Algorithm 2 Method for computing the proposed distribution.

1: $s \leftarrow 1$
2: **while** $s \geq 1$ **do**
3: $\quad u \leftarrow random[1, 1000]$
4: $\quad v \leftarrow \frac{2*u}{1000} - 1$
5: $\quad s \leftarrow 2 * v^2$
6: **end while**
7: $X \leftarrow v * \sqrt{-2 * lns}$

4.2.2. *Probability of Choosing Worse Solutions.* As mentioned above, this algorithm allows worst results to be chosen by some probability, this way it enlarges its search area and avoids local peaks. With every generation this probability shrinks by a percent value. We established that this percent should be 98%. As you can see in Table 1, if we used 97% by the 100^{th} generation this possibility did not even count, but by using 99%, even by the 200^{th} generation this counted too much. By using 98% rate we managed to overcome both problems.

4.3. Circuit Selection. We choose two tracks for experiments. The first one (Figure 4 left side) resembles a major city artery with a width of 15m and medium to fast corners; the second one (Figure 4 right side) looks a lot like a highway with a width of 30m and fast corners. The tracks length is not important as we will not compare the two of them directly. In choosing the tracks we consulted [12].

Number of generations	0.97	0.98	0.99
50	0.21	0.36	0.6
100	0.04	0.13	0.36
150	0.01	0.04	0.22
200	0.002	0.01	0.13

TABLE 1. Values for different probability rates of accepting worse results. The header line contains the values used for decreasing the *plan* variable in Algorithm 1.

FIGURE 4. From left to right, the city and highway tracks.

5. Experimental Results

The results are compared with the ones obtained using HC. The results spiking to the value 500 can be explained by the impossibility to run the simulated race. Impossibility can occur due to car accident or even due to some reason outside the game like losing the connection with the server.

5.1. Lap time. The charts from Figures 5 and 6 represent the lap time of the solution over 200 generations. The horizontal axis represents the generation number, and the vertical axis depicts the lap time. All the charts are made up with R statistical computing language [16].

Our expectations that SA algorithm surpasses HC are met, but this does not happen under any circumstances. However, SA results are in the worst case scenario very similar to the ones of HC, but not weaker. It can be observed in Figures 5 and 6 that as generations pass, the lap time tends be to smaller, so the aim of the optimization process in achieved.

On the city track with no other opponents, the best lap time results are around 90. This is very similar to what HC obtained. Next we introduced two other cars on the circuit. Surprisingly, the results were also around 90, so the other cars din not have a major influence on the results. However, although the final outcome is the same, we observed that it took a few generations more to get to those results (Figure 5). When using HC, the other cars had a major influence making the car slower, so it can be concluded that SA wins this round.

On the highway track SA maintains the supremacy over HC. While racing without other cars the best result is around 110, an major improvement over HC which obtains only around 120. As this track is very wide, we expected that by introducing other cars the results should not be influenced. This indeed happens, as the results were maintained around 110, fact that can be observed also in Figure 6.

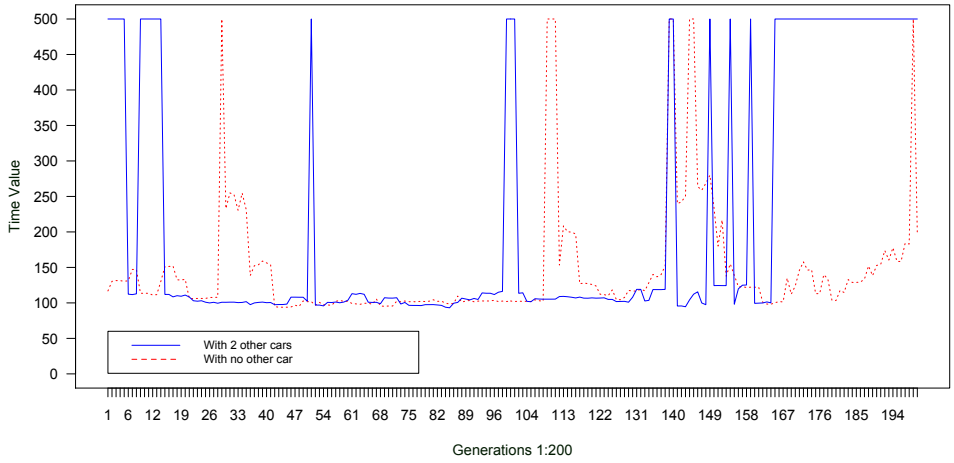

FIGURE 5. Results obtained on the city track, with and without other cars.

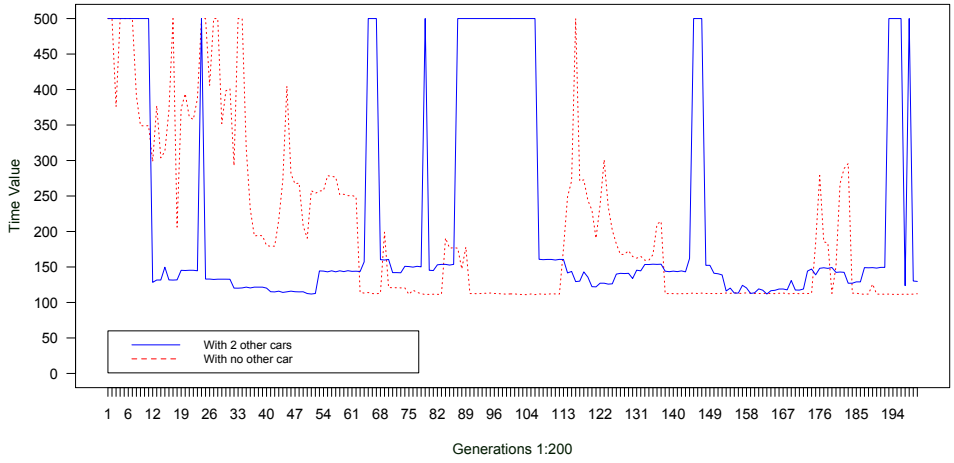

FIGURE 6. Results obtained on the highway track, with and without other cars.

5.2. Gear values. In Figures 7 and 8, the horizontal axis represent the set of gears, while the vertical axis contains the found RPM values for the corresponding gear changes. The gears are denoted by "Up" or "Down" followed by two numbers. For example "Down 2-3" represents the RPM value for changing the gear downward from the 3^{rd} to the 2^{nd} gear, the same with "Up 2-3" only the gear is changed upward. The charts from Figures 7 and 8 contain each 3 distinct sets of gear values, that conducted to good lap times for each scenario.

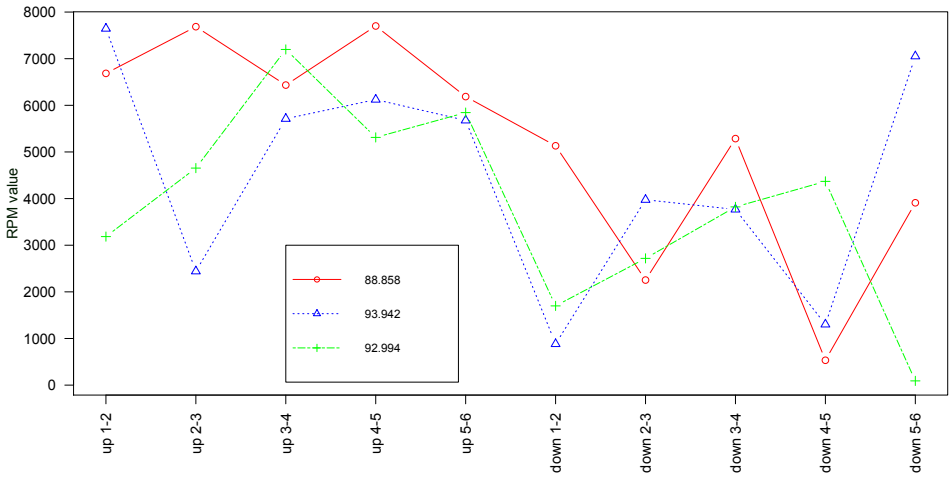

FIGURE 7. Chart of gear change RPM values on the city track, with no other cars.

Up 1-2	Up 2-3	Up 3-4	Up 4-5	Up 5-6	Lap time
6685	7686	6434	7701	6187	88.858
7645	2441	5712	6123	5677	93.942
3185	4654	7198	5312	5844	92.994
Down 1-2	Down 2-3	Down 3-4	Down 4-5	Down 5-6	Lap time
5132	2253	5286	533	3910	88.858
882	3975	3767	1302	7052	93.942
1699	2718	3822	4369	91	92.994

TABLE 2. Upward (first half of the table) and downward (in the second half) changing gear values showed in Figure 7.

Figures 7 and 8 should be analyzed together with the Tables 2 and 3. In opposition to the results obtained with the HC algorithm in [1], the current outcomes indicate an obvious correlation for both considered circuits. In both charts it can be seen that the peeks follow the same pattern, fact that proves the consistency of the results, similar settings are detected in distinct runs, so we'll give another point to the SA algorithm.

6. Concluding Remarks

The current paper treats the problem of finding a good RPM setting for a car under different circumstances. We tested a car for two different circuits, without any opponents, i.e. as in practice mode, or with two opponents. We tried two different optimization techniques that are both economical with respect to the number of fitness

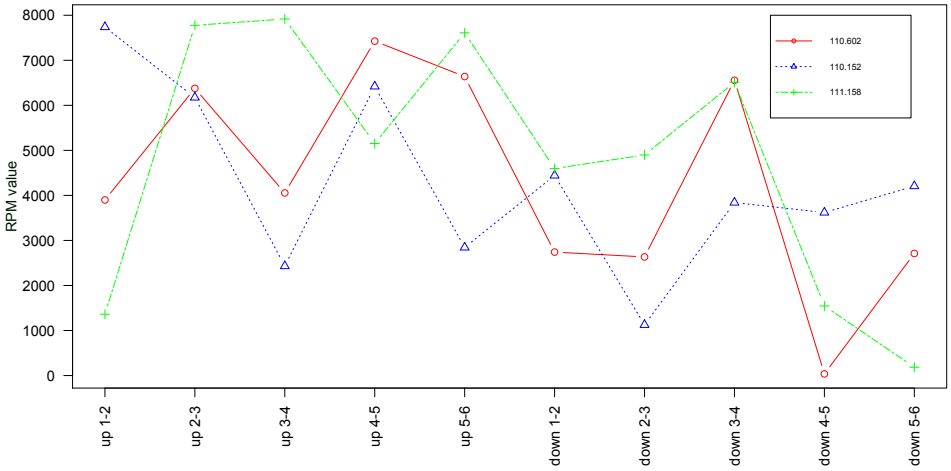

FIGURE 8. Chart of gear change RPM values on the highway track, with 2 other opponent cars.

Up 1-2	Up 2-3	Up 3-4	Up 4-5	Up 5-6	Lap time
3899	6376	4055	7426	6639	110.602
7738	6174	2430	6420	2844	110.152
1361	7777	7917	5152	7609	111.858
Down 1-2	Down 2-3	Down 3-4	Down 4-5	Down 5-6	Lap time
2741	2634	6556	37	2711	110.602
4442	1125	3841	3620	4206	110.152
4596	4899	6511	1546	185	111.858

TABLE 3. Upward (first half of the table) and downward (in the second half) changing gear values showed in Figure 8.

calls (as each simulation can last around 30 seconds), but are also powerful enough to find interesting solutions.

We establish that SA can be successfully used for finding appropriate RPM values when the task is to improve the lap time. Also we come to the conclusion that for the undertaken problem, as it generally happens, when the choice is between SA and HC methods, the former is preferred. One reason to support this decision is that in 3 of our 4 scenarios it gave better results, and in the 4^{th} it was as good as HC. A second rationale is that it provides sets of gear values which are more correlated than those returned by HC.

In the future we aim to focus also on the fuel consumption objective alone and even in conjunction with lap time improvement.

References

[1] A. P. Becheru and C. Stoean, Machine learning in gear changing, *Proceedings of the Second International Students Conference on Informatics, Imagination Creativity Design Development ICDD*, Sibiu, Romania, April 26-28, 2012, 32-43.

[2] L. delaOssa, J. A. Gamez and V. Lopez, Improvement of a car racing controller by means of Ant Colony Optimization algorithm, *Procedings of the IEEE Symposium on Computational Intelligence and Games (CIG'08)*, Perth, Australia, December 15-18, 2008, 365-371.

[3] M. Diehl, Free Games for Linux, *Linux Journal*, October 18, 2007.

[4] A. Knapp, Nevada Passes Law Authorizing Driverless Cars, *Forbes*, June 25, 2011.

[5] D. Loiacono, L. Cardamone and P. L. Lanzi, Simulated Car Racing Championship 2010 Competition Software Manual. Politecnico di Milano, Dipartimento di Elettronica e Informazione, Italy, may 2010, *Procedings of the IEEE Symposium on Computational Intelligence and Games*, 2011.

[6] J. Markoff, Google Cars Drive Themselves, in Traffic, *The New York Times*, October 11 2010.

[7] J. Munoz, G. Gutierrez and A. Sanchis, Controller for TORCS created by imitation, *Procedings of the IEEE Symposium on Computational Intelligence and Games (CIG'09)*, Milano, Italy, September 7-10, 2009, 271-278.

[8] E. Onieva, D. A. Pelta, J. Alonso, V. Milanes and J. Perez, A Modular Parametric Architecture for the TORCS Racing Engine, *Procedings of the IEEE Symposium on Computational Intelligence and Games(CIG'09)*, Milano, Italy, September 7-10, 2009, 256-262.

[9] D. Perez, Y. Saez, G. Recio and P. Isasi, Evolving a rule system controller for automatic driving in a car racing competition, *Procedings of the IEEE Symposium on Computational Intelligence and Games (CIG'08)*, Perth, Australia, December 15-18, 2008, 336-342 .

[10] D. Perez, G. Recio, Y. Saez and P. Isasi, Evolving a Fuzzy Controller for a Car Racing Competition, *Procedings of the IEEE Symposium on Computational Intelligence and Games (CIG'09)*, Milano, Italy, September 7-10, 2009, 263-270.

[11] D. L. Poole and A. K. Mackworth, *Articial Intelligence Foundations of Computational Agents*, Cambridge University Press, 2010, `http://artint.info/html/ArtInt.html`.

[12] M. Preuss, J. Quadflieg and G. Rudolph, TORCS Sensor Noise Removal and Multi-objective Track Selection for Driving Style Adaptation, *Procedings of the IEEE Symposium on Computational Intelligence and Games (CIG'11)*, Seoul, South Korea, August 31-September 3 , 2011, 337-344.

[13] M. Preuss, G. Rudolph and S. Wessing, Tuning optimization algorithms for real-world problems by means of surrogate modeling. *Proceedings of the 12th annual conference on Genetic and evolutionary computation (GECCO '10)*, New York, USA, ACM Press (2010), 401-408.

[14] J. Quadflieg, M. Preuss and G. Rudolph, Driving Faster Than a Human Player. *Proceedings of the 2011 international conference on Applications of evolutionary computation*, Springer-Verlag Berlin, Heidelberg (2011), vol. 1, 143-152.

[15] C. Stoean and R. Stoean, *Evoluţie şi inteligenţă artificială. Paradigme moderne şi aplicaţii*, Editura Albastra, 2010.

[16] R Development Core Team, *R: A language and environment for statistical computing. R Foundation for Statistical Computing*, Vienna, Austria, 2012, ISBN 3-900051-07-0, `http://www.R-project.org/`.

[17] B. Wymann, *T.O.R.C.S. Manual installation and Robot tutorial*, `http://berniw.org/`.

[18] `http://torcs.sourceforge.net/`.

[19] `http://cig.ws.dei.polimi.it/`.

[20] `http://www.bloodshed.net/devcpp.html`.

[21] `http://www.autoitscript.com/site/autoit/`.

(Alexandru P. Becheru, Cătălin Stoean) Department of Computer Science, University of Craiova, 13 A.I. Cuza Street, Craiova, 200585, Romania

E-mail address: `alex.becheru@inf.ucv.ro, catalin.stoean@inf.ucv.ro`