TMLAI TRANSACTIONS ON MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE
SOCIETY FOR SCIENCE AND EDUCATION UNITED KINGDOM
VOLUME 2, ISSUE 6
ISSN: 2054 - 7390

Comparison of Metaheuristic Algorithms for Evolving a Neural Controller for an Autonomous Robot

[1]**Sergii Zhevzhyk and** [2]**Wilfried Elmenreich**
Institute of Networked and Embedded Systems, Alpen-Adria Universität Klagenfurt, Klagenfurt, Austria;
[1]sergii.zhevzhyk@aau.at; [2]wilfried.elmenreich@aau.at

ABSTRACT

Evolutionary algorithms are a possible way to automatically design the behavior of autonomous robots. In this paper we compare different evolutionary algorithms (EA), namely simple EA, two dimensional cellular EA, and random search, according to their performance in a simple simulation, where a phototaxis robot with two sensors of limited range has to find a light source in a closed area. In our experiments we studied the effects on performance of EA parameters, such as population size and number of generation. The results explain how the choice of the neural network (three-layered or fully-connected) may influence the quality of a final solution.

Our findings indicate that acceptable results can be achieved using all EAs but not with random search. The utilization of a fully-connected neural network allows achieving better results for all EAs as compared to a three-layered neural network. Two dimensional cellular EA and simple EA evolve the best strategies for a robot's behavior which allow the robot to reach the light source in almost all cases.

Keywords: Evolutionary algorithm; neural network; robot simulation.

1 Introduction

One way to create a control system for an autonomous robot is to apply an evolutionary approach for evolving a neural controller. EAs can be used to solve different problems especially in machine learning and function optimization domains [13], [17], [20]. A lot of performed experiments with evolving of adaptive behavior confirm that evolutionary algorithms can generate a number of successful system controllers [17], [9], [10]. In this work we compare three metaheuristic algorithms (simple EA, two dimensional cellular EA, and random search) in their ability to discover the most apt neural controller for a phototaxis robot.

The idea of using natural selection for evolving of control systems for robots was proposed by Turing in 1950s. In the next decades a set of metaheuristic algorithms have been developed [12], [15], but the actual usage of these algorithms started after 1990 by the continuous improvement of computer technology and the advent of evolutionary robotics [5]. A large number of EAs causes a need for a generally accepted methodology that allows comparing different EAs and exploring which parameters significantly affect performance [6]. Comparison of EAs on theoretical level was carried out by He and Yao applying Markov chains [14]. Empirical studies in this field were done by De Jong [7] and by Schaffer, et al. [19]. Their works are mostly applied to genetic programming. In our study we compare

DOI: 10.14738/tmlai.26.783
Publication Date: 21th December, 2014
URL: http://dx.doi.org/10.14738/tmlai.26.783

metaheuristic algorithms for evolving a neural controller. Czarn mentioned that the results of theoretical work may not comply with practical outcomes [6]. Therefore, here we focus on practical experiments and analysis of these results.

Programming a controller for a robot can become a challenging task because of algorithm complexity, processing of results from sensors, and actuator modeling [9]. A possible way to overcome this issue is to use an evolutionary algorithm to design a neural controller. Applying this approach raises the issue of selecting an appropriate metaheuristic algorithm and its settings since it can have a significant impact on the quality of the evolved behavior.

The purpose of this paper is to compare metaheuristic algorithms applicable to designing the behavior for an autonomous robot. Robot simulation provides a convenient test bed to compare the performance of differently evolved control algorithms. We used a computer simulation as application of metaheuristic algorithms. We defined a simple computer simulation of an autonomous robot, its neural controller and the environment. The simulation of the robot is based on the former experiment we used in [18] for comparing the evolvability of ANNs and Finite State Machines. The main objective is to compare different metaheuristic algorithms and determine dependencies (e.g., population size, number of generations) which may have an impact on performance. This knowledge can be helpful in selection of an appropriate evolutionary approach in future research. Section 2 of this paper provides background about evolutionary algorithms and briefly describes simple EA, two dimensional EA, and random search. Section 3 defines the problem definition and specifies configurations of the conducted experiments. In Section 4 we examine the results of EAs evaluations. Section 5 concludes the paper.

2 Evolutionary Algorithms

In the past decades a large number of evolutionary algorithms were presented, but in general they are united by the same idea: a population with limited resources competing for resources, therefore activating natural selection. Algorithm 1 shows a generalized functioning scheme of evolutionary algorithms. They work with a pool of candidates, each described by a candidate’s genotype. The initial population is filled either with randomly generated representations or with candidates which are specifically adapted for this problem. A parameter N denotes the population size in the evolutionary algorithms.

```
initialize population;
evaluate candidates;
while not (termination criterion) do
    parent selection;
    recombination;
    mutation;
    evaluate candidates;
    survivor selection;
end
```

Algorithm 1: General algorithm of EA.

Operators of variation (recombination, mutation) and selection are the two forces that drive evolution forward. The main role of variation operators is the generation of new candidates for the next evolutionary steps. Selection is used to choose individual genomes from the population for later breeding. The fitness function is a result of the candidates’ evaluation in one or multiple simulation runs and allows to measure a quality of a genotype [8]. The quality of a final solution for an evolutionary algorithm strongly depends on matching of representations, variation operators, and fitness function [11].

2.1 Simple EA

A simple EA is the reflection of the generalized scheme of evolutionary algorithms. Appropriate solutions are achieved through application of variation operators (crossover, mutation) and selection of the most fitted individuals. To use this algorithm, the parameters ρ_e, ρ_r, ρ_m, ρ_c, respectively representing the rate of elite candidates, the rate of randomly selected individuals, the rate of representations for mutation, and the rate of candidates as results of crossover, should be defined. Based on these parameters and known size of the population *N*, we calculate the number of elite candidates, the number of randomly selected representations, the number of individuals for mutation, the number of candidates for breeding, which are stored in the parameters n_e, n_r, n_m, and n_c, respectively.

```
$X \leftarrow$ randomly generated population;
while not (termination criterion) do
    foreach candidate $x_i$ of $X$ do
        run experiment for $x_i$;
        compute the fitness $f(x)$ of $x_i$;
    end
    descending sort of $X$ based on $f(x)$;
    $X_{new} \leftarrow$ empty population ;
    for $i \leftarrow 1$ to $n_e$ do
        add $x_i$ to $X_{new}$;
    end
    $k \leftarrow n_e$;
    for $i \leftarrow k$ to $k + n_r$ do
        $x_r \leftarrow$ randomly selected individual from $X$;
        add $x_r$ to $X_{new}$;
    end
    $k \leftarrow k + n_r$;
    for $i \leftarrow k$ to $k + n_m$ do
        $x_e \leftarrow$ randomly selected elite candidate;
        $x'_e \leftarrow$ mutate $x_e$;
        add $x'_e$ to $X_{new}$;
    end
    $k \leftarrow k + n_m$;
    for $i \leftarrow k$ to $k + n_c$ do
        $x_{e1} \leftarrow$ randomly selected elite candidate;
        $x_{e2} \leftarrow$ randomly selected elite candidate;
        $x_c \leftarrow$ mate $x_{e1}$ and $x_{e2}$;
    end
    $k \leftarrow k + n_c$;
    for $i \leftarrow k$ to $N$ do
        $x_n \leftarrow$ randomly generated individual;
        add $x_n$ to $X_{new}$;
    end
    $X \leftarrow X_{new}$;
end
```

Algorithm 2: The above pseudo-code outlines the algorithm of simple EA.

Typically, the elite candidates in a population having the highest fitness value are selected for the next generation. Accordingly, their offspring, as results of crossover and mutation, take the places of the less fit representations. However, a small percentage of individuals not belonging to the elite can also be selected to the next generation, because of their property or characteristic in their structure might be useful in the next generation. Moreover, non-elite candidates allow increase diversity of the population and thus increment a number of different unique solutions. Algorithm 2 is describing the implementation of simple EA.

2.2 Cellular EA

A cellular EA (cEA) [23] is a kind of evolutionary algorithms, in which the population is placed in a toroidal two dimensional space. Candidates can only communicate with their neighbors, what corresponds to the behavior of individuals in nature. There are many models of neighborhoods for cEA, such as Von Neumann (linear) neighborhood, Moore (compact) neighborhood, diamond neighborhood and others [16]. Usage of different models can lead to completely different strategies. In our experiments we use only Moore neighborhood with radius one, which means that only the closest neighbors are taken into consideration. The neighborhood R also includes the central candidate for which we calculate the neighborhood. In this evolutionary algorithm the parameters ρ_e, ρ_m, ρ_c, which respectively denote the rate of elite individuals, the probability of mutation, and the probability of crossover, are applied for the neighborhoods. The number of elite candidates n_e is calculated from ρ_e and the neighborhood size. Algorithm 3 shows the pseudo-code for a cellular EA.

```
$X \leftarrow$ randomly generated population;
while not (termination criterion) do
    $X_{new} \leftarrow$ empty population;
    foreach candidate $x_i$ of $X$ do
        run experiment for the neighborhood $R_i$ of $x_i$;
        compute the fitness $f(x)$ of $R_i$;
        descending sort of $R_i$ based on $f(x)$;
        if number of $x_i$ in $R_i \leq n_e$ then
            add $x_i$ to $X_{new}$;
        else
            generate random number $r \in [0, 1]$;
            if $r < \rho_m$ then
                $x_e \leftarrow$ randomly selected elite candidate;
                $x'_e \leftarrow$ mutate $x_e$;
                add $x'_e$ to $X_{new}$;
            else if $r < \rho_m + \rho_c$ then
                $x_e \leftarrow$ randomly selected elite candidate;
                $c_i \leftarrow$ mate $x_i$ and $x_e$;
                add $c_i$ to $X_{new}$
            else
                $x_n \leftarrow$ randomly generated individual;
                add $x_n$ to $X_{new}$;
            end
        end
    end
    $X \leftarrow X_{new}$;
end
```

Algorithm 3: The above pseudo-code outlines the algorithm of cellular EA

2.3 Random search

Random search finds a solution using an undirected search (see Algorithm 4). In each generation, all candidates of the population are replaced with randomly generated candidates, which are subsequently evaluated. A single candidate that has the highest fitness value is kept for the next generation. If the search space is small and the number of evaluations, i.e. generations times population size, is comparably high, then this algorithm has a chance to pick an acceptable solution. In case of large search space this chance goes down. Compared to other algorithms, random search does not try to improve candidates via mutation or crossover, therefore it can be treated as an undirected search. The random search approach gives a reference for the size of the search space.

```
$X \leftarrow$ randomly generated population;
while not (termination criterion) do
    foreach candidate $x_i$ of $X$ do
        run experiment for $x_i$;
        compute the fitness $f(x)$ of $x_i$;
    end
    descending sort of $X$ based on $f(x)$;
    for $i \leftarrow 2$ to $N$ do
        $x_r \leftarrow$ randomly generated candidate;
        replace i-th individual of $X$ by $x_r$;
    end
end
```

Algorithm 4: The above pseudo-code outlines the algorithm of random search.

3 Experiment Setup

3.1 Physical setup

Figure 1 sketches the simulation setup of our phototaxis robot searching for a light source. The testbed for our robot is a closed quadratic room. The start position is in the center of this room. The position of the light source is outside a restriction circle with the central point in the center of the room. The restriction circle prevents a finding of the light on the first steps. All environment settings are shown in Table 1.

For our experiments we used a differential wheeled robot with configuration described in Table 2. It has 2 sensors to detect whether the distance to the light source is within their sensing range.

Table 1: Configuration of the environment.

Parameter name	Parameter value
Width of the field	200 cm
Diameter of the light source	10 cm
Radius of the restricting circle	80 cm

Table 2: Robot configuration parameters.

Parameter name	Parameter value
Diameter of the robot	10 cm
Diameter of the wheels	5 cm
Range of the sensors	70 cm
Angle of the sensor vision	45˚
Maximum speed	12 cm/s

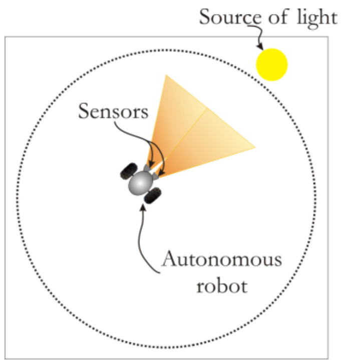


Figure 1: An autonomous robot is looking for the light source in the closed area.

3.2 Fitness function

The main task of the robot is to reach its target in a minimal amount of movements – on this basis we implemented the fitness function as in Equation 1.

$$F = k_t P_t + k_s P_s + k_l P_l \tag{1}$$

$$P_t = \begin{cases} 1 & \text{if robot reached the target} \\ 0 & \text{if robot did not reach the target} \end{cases} \tag{2}$$

$$P_s = \begin{cases} e^{\frac{r-d}{r}-1} & \text{if robot senses the target} \\ 0 & \text{if does not sense the target} \end{cases} \tag{3}$$

$$P_l = e^{\frac{m-l}{m}-1} \tag{4}$$

P_t is the reward for a successful strategy allowing to reach the target (see Equation 2). The value P_s shows how close the robot is to the target at the end of simulation (see Equation 3). This value is especially important in the beginning of an evolution to teach the robot to come closer to the light and finally reach it. The maximum range of the robot's sensors is represented as parameter *r*. The distance is encoded in parameter *d* in case the robot senses the light. Finally, Equation 4 represents how fast the robot can reach the target. The value *m* represents the maximum amount of time steps in the simulation. The number of time steps that is required to reach the target for the selected strategy is defined as *l*. Coefficients k_t, k_s and k_l describe the influence of P_t, P_s, and P_l on the fitness value. In our work they have been set to the following values:

k_t = 0:3; k_s = 0:3; k_l = 0:4

The fitness function is designed in a way that all possible fitness values should lay in the range [0;1]. The maximum number of time steps for our experiments is 300.

3.3 Evolvable control system

The robot was controlled by an ANN. In our simulations we used two different representations: a fully-connected ANN and a three-layered ANN.

The three-layered neural network is a time-discrete ANN which has a feed-forward structure. It means that each neuron of the input layer is connected to each neuron of the hidden layer which at the same time is connected to each neuron of the output layer. The fully-connected neural network is a discrete-time and recurrent ANN. Instead of feed-forward structure of three-layered neural network, each neuron of the fully-connected neural network is connected to every other neuron and itself, thus making it a recurrent artificial neural network [24]. A recurrent network can retain information about the past, but in general is hard to train [25]. In our case, the training of the two network types follows the same approach of mutating and recombining a genome consisting of weights and biases of the ANN. A fully-connected neural network has a larger search-space whereas it employs more connections between neurons. At the same time this feature and presence of recurrent connections might help to achieve more sophisticated behavior.

The number of inputs and the number of outputs are the same for both candidates. Two inputs which represent distances measured by sensors are connected to the input neurons. From two output neurons we receive information about the speed of robot’s wheels. With regard to the number of neurons in the hidden layer, there is no straightforward way to determine the optimal number of hidden neurons analytically. The optimal number depends on the complexity of the function to be approximated, and, therefore, indirectly on the number of input and output nodes. Besides a trial and error approach, there are some empirically derived rules-of-thumb, of these, the most commonly relied on is the optimal size of the hidden layer is usually between the size of the input and size of the output layers [2]. Swingler [22] and Berry [1] propose a maximum of two times the number of input nodes for the hidden nodes. Boger and Guterman [3] suggest that the number of hidden nodes should be 70%-90% of the number of input nodes. Caudill and Butler [4] recommend that the number of hidden nodes should be two third of the sum of input and output nodes. Since determining the optimal number of hidden nodes for a problem is outside the scope of this paper, we have chosen two hidden nodes in accordance with most of the rules of thumb given above.

In our experiments we apply metaheuristic algorithms to train these networks. The main idea of this training is to optimize the weights w_{ji}, where *j* represents the neurons which have incoming connection to *i*, and the bias b_i of each neuron *i* in the ANNs. We calculate the output of the neuron for step *k* by applying an activation function *F*:

$$o_i(k) = F(\sum_{j=0}^{n} w_{ji} o_j(k-1) + b_i) \quad (5)$$

where the sigmoid function is employed as activation function *F*:

$$F(x) = \frac{1}{1 + \mathrm{e}^{-x}} \quad (6)$$

3.4 Experiment parameters

All experiments are developed using the FREVO tool [21] which has a workflow for the selection of building blocks (problems, representations, evolutionary algorithms and ranking systems) and provides an easy setup for all necessary settings.

Settings of evolutionary algorithms have a huge impact on results of experiments. Information about used configurations is specified in tables 3 and 4.

We conducted a set of experiments with 2000, 5000, 10000, and 100000 evaluations. For each of these values we run experiments 100 times with different initial seeds in order to get sufficient statistical data. The results obtained from these experiments allow to watch an evolutionary process in detail. To check how the number of candidates in the population influences the results of evolutionary algorithms, we used the following population sizes: 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, which are the squares of natural numbers. This is the requirement of cellular EA that builds a toroidal two dimensional space. The number of evaluations equals the population size multiplied by the number of generations.

Table 3: Settings of simple EA.

Name	Value
Elite rate	0.1
Mutation rate	0.6
Crossover rate	0.1
Renew rate	0.1
Random selection rate	0.1
Mutation severity	0.3
Mutation probability	0.3

Table 4: Settings of cellular EA.

Name	Value
Elite rate	0.1
Probability of elite mutation	0.6
Probability of elite crossover	0.1
Renew probability	0.2
Mutation severity	0.3
Mutation probability	0.3

4 Experiment Setup

We have conducted a set of experiments on evolving the autonomous robot controller using different evolutionary algorithms. Since the runtime of the simulation accounts for the majority of time spent for evaluating solutions, we specified a given number of evaluations for each experiment. Figure 2 depicts the results after 2000 evaluations, which corresponds to a rather short time of evolution. Thus, this figure indicates which algorithm and parameter setting is preferable if there is no possibility for extensive simulation, e.g., there is a limit on run time. The fitness values (ranging from 0 to 1, according to the definition in Section III) show a large dispersion of results. The values for random search mark an inefficient algorithm, while cellular EA and simple EA show comparable good results for short time of evolution.

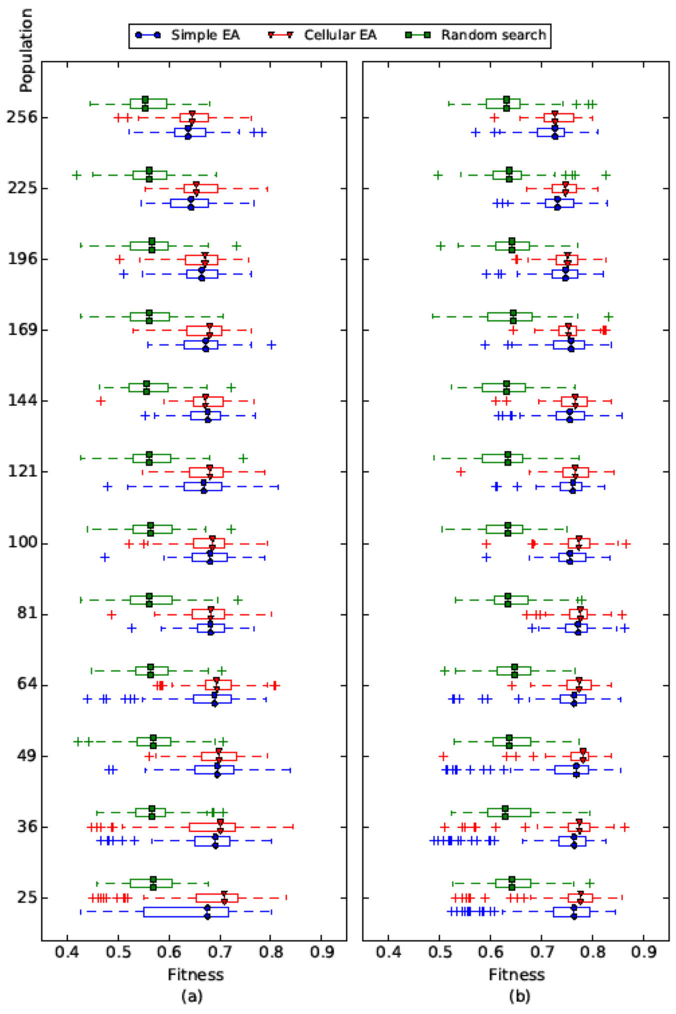


Figure 2: Box-and-whisker diagrams of the fitness values after 2000 evaluations for (a) three-layered ANN and (b) fully-connected ANN.

Figure 3 shows the results after 5000 evaluations which yield better fitness values than after 2000 evaluations. The relative effectiveness of the algorithms stayed the same.

Figures 4 and 5 extend the number of evaluations towards 10000 and 100000, respectively. The latter corresponds to a case where sufficient simulation time is available and the question shifts from w*hich algorithm provides good results the fastest?* to *which algorithm provides the best results if we wait long enough?*. The fitness values are more gathered after 100000 evaluations, but the performance of evolved controllers is good enough. The difference in terms of efficiency of neural networks after 10000 and 100000 evaluations is negligible compared to waiting time.

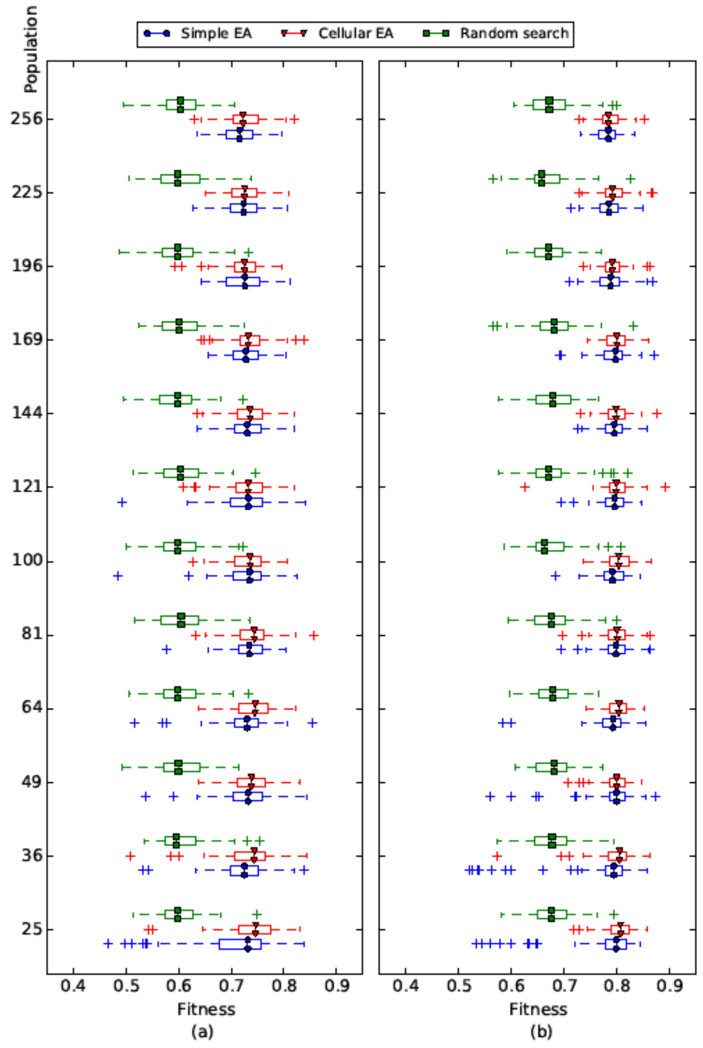


Figure 3: Box-and-whisker diagrams of the fitness values after 5000 evaluations for (a) three-layered ANN and (b) fully-connected ANN.

We can see that fully-connected ANN performs better than three-layered ANN employing all evolutionary algorithms, but with increasing number of evaluations this difference becomes insignificant.

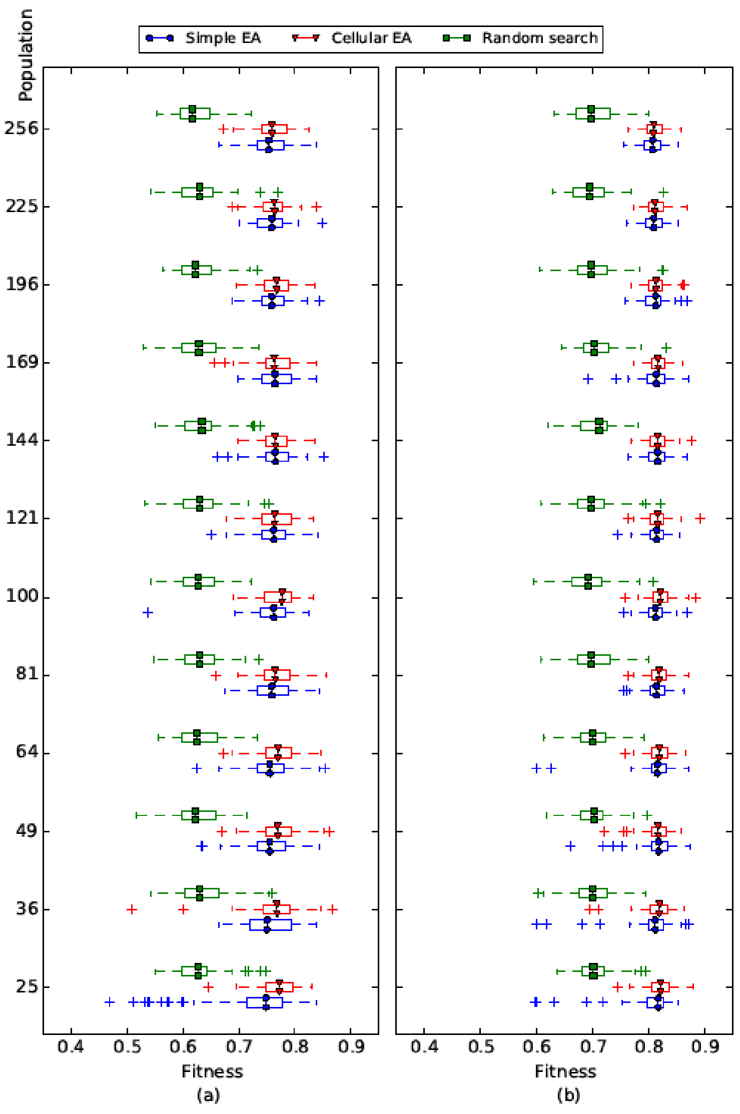


Figure 4: Box-and-whisker diagrams of the fitness values after 10000 evaluations for (a) three-layered ANN and (b) fully-connected ANN.

4.1 Evaluation of Significance

Considering that the results from the simulations are affected by random factors it is not so easy to affirmatively define which algorithm and settings work better and which show similar performance. To answer this question we model the fitness values for the two algorithms as two independent events – *X* for cEA and *Y* for simple EA:

$$X \sim N(\mu_X, \sigma_X^2),$$

$$Y \sim N(\mu_Y, \sigma_Y^2),$$

where μ_X, μ_Y are means and σ_X^2, σ_Y^2 are estimated variances of results measured using multiple simulation runs with different random seeds.

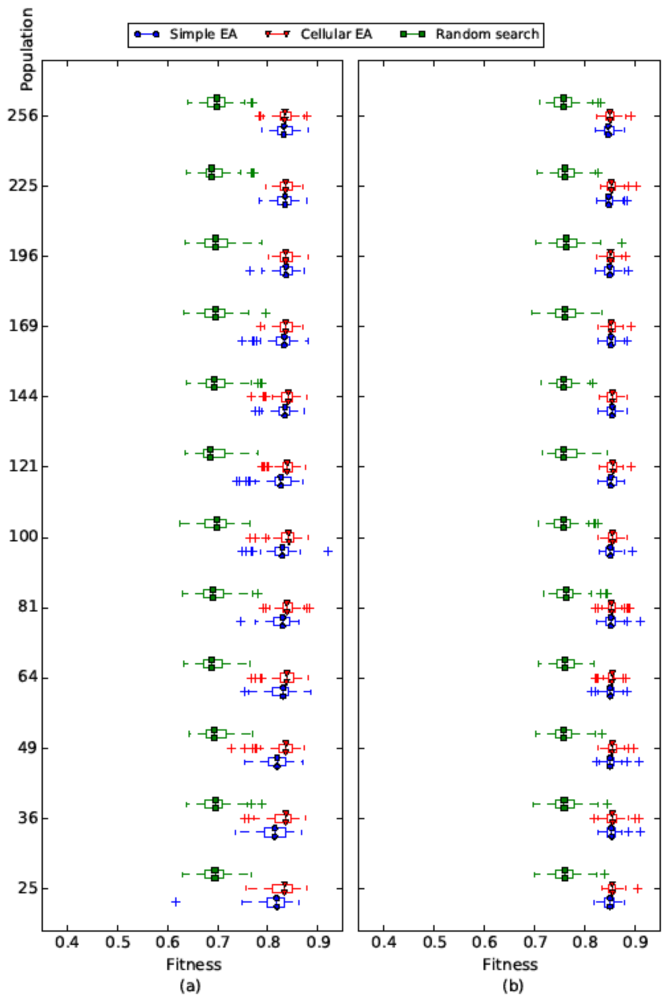


Figure 5: Box-and-whisker diagrams of the fitness values after 100000 evaluations for (a) three-layered ANN and (b) fully-connected ANN.

In the next step, we calculate the difference between two events:

$$Z \sim N(\mu_Z, \sigma_Z^2) \sim X - Y \sim N(\mu_Y, \sigma_Y^2) - N(\mu_X, \sigma_X^2) \sim N(\mu_Y - \mu_X, \sigma_Y^2 - \sigma_X^2) \tag{7}$$

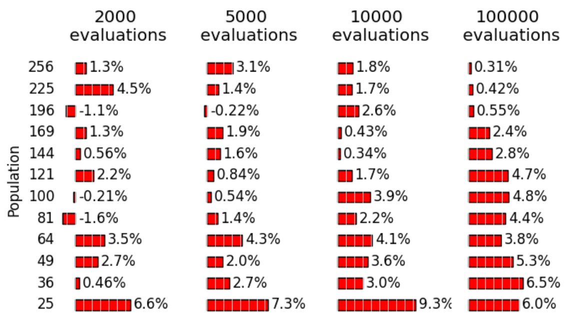
In order to compare cEA and simple EA, we calculate a chance, that probability of *Z* is less than 0:

$$P(Z \le 0) = \frac{1}{2} erfc(\frac{\mu_Z}{\sqrt{2}\sigma_Z}) \tag{8}$$

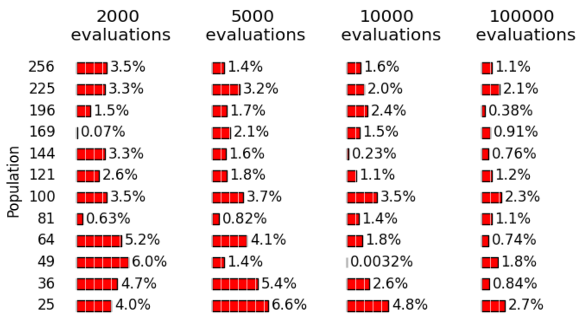
The probability $P(Z \le 0)$ corresponds to the probability that cEA is better than simple EA. The probability that simple EA is better than cEA can be obtained using Equation 9.

$$P(Z > 0) = 1 - P(Z \le 0) \tag{9}$$

Figure 6 shows the difference between calculated probabilities $P(Z \le 0)$ and $P(Z > 0)$, which at the same time allows observing how cellular EA is better than simple EA. The trends for different neural networks vary. Cellular EA employing three-layered ANN works better than simple EA for larger number of evaluations. For fully-connected ANN simple EA works better for small number of evaluations and with increasing number of evaluations this difference becomes insignificant. Figure 6 points that for fully-connected ANN cellular EA provides better results than simple EA. If we employ three-layered ANNs, the cellular EA also dominates over simple EA with a few exceptions.



(a)



(b)

Figure 6: Prevalence of cellular EA compared to simple EA for different neural networks: a) three-layered ANN; b) fully-connected ANN.

5 Conclusion

Two dimensional cellular EA and simple EA show acceptable results in evolving behavioral designs of an autonomous robot. Examination of outcome robot strategies using these algorithms shows that the light source can be found in the vast majority of experiments. Achieved performance results using different evolutionary algorithms demonstrate efficiency of metaheuristic approach for evolving of an autonomous robot.

The results of the experiments help to determine, that cEA and simple EA are the most applicable for evolving a neural controller. A fully-connected ANN outperforms three-layered ANN in all conducted experiments. Based on our findings, we recommend to use cEA and fully-connected ANN for problems that require short evaluation phase. For a large number of generations and population size the efficiency of both algorithms are approximately the same. In the experiments we measured an influence of population size and number of generations on performance of metaheuristic algorithms. The dependencies on these parameters are negligible. This information is important for the conduction of experiments. To accelerate a simulation, the population size should be the same as the number of cores on the server, where these experiments will be performed.

In future work we are planning to validate our results for different application scenarios and to extend our analysis to further parameters, for instance, mutation and crossover rate.

ACKNOWLEDGEMENT

This work was performed in the research cluster Lakeside Labs funded by the European Regional Development Fund, the Carinthian Economic Promotion Fund (KWF), and the state of Austria under grant 20214—21532—32604 (Project MESON) and 20214—22935—34445 (Project Smart Microgrid Lab). We would like to thank A. Monacchi for the feedback.

REFERENCES

[1]. M. Berry and G. Linoff. *Data Mining Techniques: For Marketing, Sales, and Customer Support*. Database management / Wiley. Wiley, 1997.

[2]. A. Blum. *Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems*. Wiley, 1 edition, 5 1992.

[3]. Z. Boger and H. Guterman. Knowledge extraction from artificial neural network models. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference*, volume 4, pages 3030–3035 vol.4, 1997.

[4]. M. Caudill and C. Butler. *Understanding Neural Networks; Computer Explorations*. MIT Press, Cambridge, MA, USA, 1992.

[5]. D. Cliff, P. Husbands, and I. Harvey. Explorations in evolutionary robotics. *Adaptive Behavior*, 2(1):73–110, 1993.

[6]. A. Czarn, C. MacNish, K. Vijayan, B. Turlach, and R. Gupta. Statistical exploratory analysis of genetic algorithms. *Evolutionary Computation, IEEE Transactions*, 8(4):405–421, 2004.

[7]. K. A. De Jong. Analysis of the behavior of a class of genetic adaptive systems. 1975.

[8]. A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2007.

[9]. W. Elmenreich and G. Klingler. Genetic evolution of a neural network for the autonomous control of a four-wheeled robot. In A. Gelbukh and A. F. Kuri Morales, editors, *Sixth Mexican International Conference on Artificial Intelligence*, pages 396–406. IEEE Computer Society, 2007.

[10]. D. Floreano and L. Keller. Evolution of adaptive behaviour in robots by means of darwinian selection. *PLoS biology*, 2010.

[11]. D. Fogel. What is evolutionary computation? *Spectrum, IEEE*, 37(2):26–28, 2000.

[12]. L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, 1966.

[13]. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[14]. J. He and X. Yao. From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms. *Evolutionary Computation, IEEE Transactions*, 6(5):495–511, 2002.

[15]. J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.

[16]. J. Kari. Theory of cellular automata: a survey. *Theoretical Computer Science*, 334(1):3–33, 2005.

[17]. A. L. Nelson, E. Grant, and T. C. Henderson. Evolution of neural controllers for competitive game playing with teams of mobile robots. *Robotics and Autonomous Systems*, (46):135–150, 2004.

[18]. A. Pinter-Bartha, A. Sobe, and W. Elmenreich. Towards the light – Comparing evolved neural network controllers and finite state machine controllers. In *Proceedings of the Tenth International Workshop on Intelligent Solutions in Embedded Systems*, pages 83–87, Klagenfurt, Austria, jul 2012.

[19]. J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 51–60. Morgan Kaufmann Publishers Inc., 1989.

[20]. M. Sipper, Y. Azaria, A. Hauptman, and Y. Shichel. Designing an evolutionary strategizing machine for game playing and beyond. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions*, 37(4):583–593, 2007.

[21]. A. Sobe, I. Fehervari, and W. Elmenreich. FREVO: A tool for evolving and evaluating self-organizing systems. In *Proceedings of the 1st International Workshop on Evaluation for Self-Adaptive and Self-Organizing Systems*, Lyon, France, Sept. 2012.

[22]. K. Swingler. *Applying Neural Networks: A Practical Guide*. Morgan Kaufmann, pap/dsk edition, 5 1996.

[23]. M. Tomassini. *Spatially structured evolutionary algorithms: artificial evolution in space and time (natural computing series)*. Springer-Verlag New York, Inc., 2005.

[24]. M. Schuster, and K. Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on 45*, no. 11: 2673-2681, 1997.

[25]. R. Pascanu, T. Mikolov, Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30 th International Conference on Machine Learning*, Atlanta, Georgia, USA, 2013