Comparison of a spatially-structured cellular evolutionary algorithm to an evolutionary algorithm with panmictic population

Thomas Dittrich, Wilfried Elmenreich
Institute of Networked and Embedded Systems / Lakeside Labs
Alpen-Adria Universität Klagenfurt, Austria
Dizi.D@gmx.at, wilfried.elmenreich@aau.at

***Abstract*—Evolutionary Algorithms are metaheuristic optimization algorithms which are based on a population of individual candidate solutions. These solutions are evolved with the aim to solve a given problem. We compare two types of Evolutionary Algorithms, one with a panmictic population and one with a spatially-structured population. Previous works indicate that evolutionary algorithms with a spatially-structured population perform better that those with a panmictic population. In this work we will examine whether this holds true for evolving Artificial Neural Networks. For comparison we use two test problems, a simple XOR calculation and a complex task requiring self-organization among a number of agents. Our findings show that for the evaluated tasks, the algorithm with a spatially-structured population performs better than an algorithm with panmictic population.**

I. Introduction

An Evolutionary Algorithm (EA) is a metaheuristic optimization algorithm for finding the optimal solution for a certain problem within a given search space [1]. There are many different types of EAs. All EAs share some common properties [2]: An EA uses the collective learning process of individuals in a population, descendants of individuals are generated by randomized processes such as mutation and recombination and by means of evaluating individuals in their environment, a measure of quality or fitness value can be assigned to every individual. The process of this optimization, called *evolution*, can be described as follows. During each step of the evolutionary process, named a *generation*, the individuals are evaluated as a controller for a given problem and then ranked by their fitness values. A selection algorithm decides for every individual if it will be replaced by any other individual, by a mutation of another individual, by its offspring, or by a new individual generated using random settings.

Evolutionary algorithms (EAs) have been applied to solve or approximately solve many problems, for example finding Cellular Automata rules for Morphogenesis (CAM) [3], the Traveling Salesman Problem [4], and for evolving game players [5]. They have also been used to evolve a robot controller [6] and for designing the controller for a mobile robot for competitive games [7], [8].

As the simulation time increases for complex problems it is important to use an EA which finds a sufficient solution in acceptable time. Often, an EA implements a panmictic population, i.e., all individuals of a population are potential partners, although typically with different mating probabilities depending on their fitness. In contrast, in spatially-structured EAs the mating between individuals is restricted based on a graph or a network [9]. EAs with a structured population implement two processes, namely exploration and exploitation in the search space, which often leads to superior results in contrast to EAs with panmictic populations. Tomassini showed for genetic programming that an EA with panmictic population results in a lower genotypic diversity in the population than when an algorithm where the population is spatially structured is used. He also showed that the panmictic population is less efficient than a spatially-structured population [10].

Cellular EAs implement a structured population in a particular form of adjacent cells, where each cell represents an individual. Cells can be arranged as a one-dimensional line, a one-dimensional ring, or a 2D grid, optional with a toroidal topology [11]. Alba examined the influence of the shape of a spatially-structured population and the influence of the size of the neighborhood on the performance of the algorithm [12]. A combination of these two types of EAs, the Island model, was investigated by Neumann in [13]. Other examples of spatially-structured EAs include patchwork models for EA [14],

terrain-based genetic algorithms [15] and religion-based EA models [16].

In this work we will examine a spatially structured evolutionary algorithm for evolving artificial neural network (ANN) controllers. The EA with panmictic population that we use for comparison is the Neural Network Genetic Algorithm (NNGA) from the Framework for Evolutionary Design (FREVO) [17]. NNGA was configured to use only a single population, thus implementing a true panmictic population. We implemented an EA with spatially structured population as a Cellular Evolutionary Algorithm with two-dimensional Population (CEA2D) as an optimization method in FREVO. For ANNs, we used a Three Layered Neural Network (TLNN) and a Fully Meshed Neural Network (FMNN). These are all components available from FREVO. For the comparison of the two EAs we used the XOR and CAM problems as test problems.

Section II introduces the spatially-structured evolutionary algorithm. Section III describes the implementation of this algorithm. Section IV introduces the test problems and Section V shows the evaluation for the two algorithms. Section VI concludes the paper and suggests ideas for further work based on our results.

II. The Spatially-Structured Evolutionary Algorithm

In a spatially-structured evolutionary algorithm each of the individual candidate solutions has a limited amount of neighbors with whom it can interact, i.e., exchange genetic information or replace a neighbor. This structure is implemented as an undirected graph with the individuals being the nodes and the connections between the neighbors being the edges. The structure that we used is a two-dimensional grid whose rows and columns are wrapped onto a toroid surface, i.e., wrapped left–right and up–down. The evolutionary process of the spatially-structured algorithm is similar as described above. A notable difference is that the selection algorithm only uses neighbors of an individual to replace it by a mutation or by a recombination.

III. Implementation of the Spatially-Structured Algorithm

The algorithm is implemented as an optimization method module within FREVO. FREVO supports the implementation of various components as building blocks. These are genome representation, optimization method, optimization problem, and ranking algorithm [17]. The described algorithm will be implemented as an optimization method, which implements the population and its structure. An optimization method can be used generically with any type of individual (named a *controller representation* in FREVO) and applied for different problems.

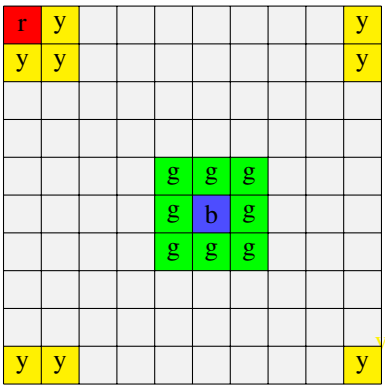

Fig. 1. A population with 10x10 candidates. The green (g) candidates are neighbors of the blue (b)) candidate and the yellow (y) candidates are neighbors of the red (r) candidate.

A. Structure of the population

The controller representations of a population are distributed over a two-dimensional area. The area that we use for this algorithm is a regular grid where every element represents one controller representation of the population. As Fig. 1 shows, the neighborhood of the controller representations is a Moore-Neighborhood which connects the borders of the grid so that it forms the surface of a toroid.

B. Selection of the controller representations

For every controller representation in the population we calculate a fitness ranking of itself and its neighbors. If this controller is ranked on position one it is selected to survive to the next generation, otherwise, it will be replaced either by a mutation of the neighbor with the highest fitness value, a recombination of itself with this neighbor or a new representation with random settings. At the beginning of the first generation all representations are initialized with random values. Configuration parameters are the percentage of candidates replaced by mutations, results of recombinations or created anew.

C. Neighborhood ranking

The controller representations are ranked by the ranking algorithm selected during the configuration of FREVO. Since the ranking returns only a list of representations ordered by fitness value, there is a potential

problem if multiple representations reach the highest fitness value in a population. For example if r_a, r_b and r_c are neighbors and have the same fitness value which is the best in their neighborhood the ranking could be the following:

ranking	for r_a	for r_b	for r_c
1.	r_b	r_c	r_a
2.	r_c	r_a	r_b
3.	r_a	r_b	r_c
	...	...	...

In these rankings, none of the three representations is elite in its own ranking and so all the representations will be replaced in the next iteration, effectively losing the individuals with the best fitness. In the implementation, this problem has been solved by disturbing each fitness value by a small random value before running the ranking algorithm, which breaks deadlocked situations such as those described above into a deterministic order among the different rankings. After ranking, these random additions are removed from the fitness values.

IV. Test Problems

For the comparison of the two evolutionary algorithms we used a setup with two test problems: The XOR problem and the Cellular Automaton Morphogenesis (CAM) problem.

A. XOR

The aim of this problem is to create a ANN wich has the functionality of the XOR function (exclusive or) as shown in the following truth table.

a	b	z
0	0	0
0	1	1
1	0	1
1	1	0

The fitness value for one representation is calculated as a negative mean squared error as follows:

$$F = -\frac{1}{4}\sum_{i=1}^{4}(o_i - z_i)^2 \tag{1}$$

with o_i being the output of the representation for the inputs a and b of the i-th line of the truth table and z_i being the expected output of this line.

B. Cellular Automaton Morphogenesis

In the Cellular Automaton Morphogenesis (CAM) Problem, a representation tries to display a reference picture. For this, it is duplicated to every position of a rectangular grid which has the same number of rows and columns as the picture has pixels (in height and width). The output of these duplicates is then converted to one of the colors that occur in the reference picture and the input is the color of their neighbors in a von Neumann neighborhood, i.e. the four cells within a Manhattan distance of one. The simulation is done in discrete time steps with the input of step i being the output of step $i-1$. At the end of every simulation step, a fitness value is calculated by the formula

$$f_i = 1 - \frac{1}{(N_C-1)^2 \cdot w \cdot h}\sum_{x=1}^{w}\sum_{y=1}^{h}\left(c_x^y - r_x^y\right)^2 \tag{2}$$

with N_C being the number of different colors in the reference picture, w and h being the width and height of the picture, c_x^y being the color output at position (x, y) and r_x^y being the reference color at this position.

To obtain a stable non-oscillating solution, the fitness is calculated as a weighted sum over N iterations

$$F = \sum_{i=1}^{N}\frac{f_i}{2^{N-i}} \tag{3}$$

with f_i being the value calculated in equation 2.

V. Evaluation

During all the simulations we used identical settings for the EAs which were chosen as follows:

Number of Generations	200
Population size	100
Percentage Elite	11
Percentage Mutation	59
Percentage XOver	30
Percentage Renew	0
Percentage Random Selection	0
Mutationseverity	30
Probability of Mutation	1

where $PercentageMutation = 11$ means, that there is exactly one elite controller representation in every neighborhood.

The data shown in each of the following figures is the average value over 100 runs with different seeds for the random number generator.

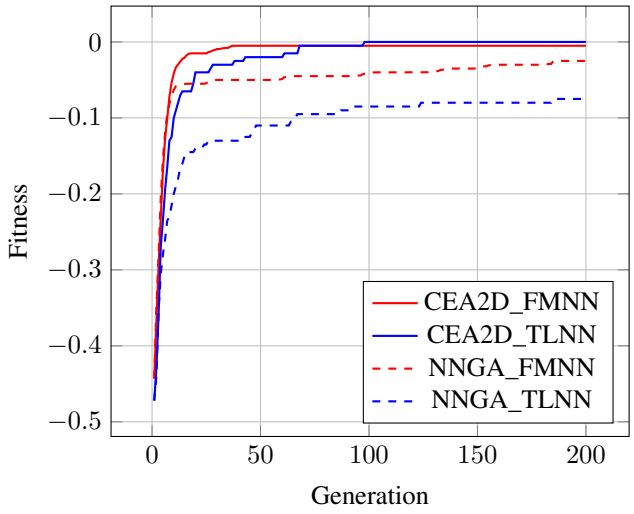

Fig. 2. Fitness against generation number for the XOR-Problem.

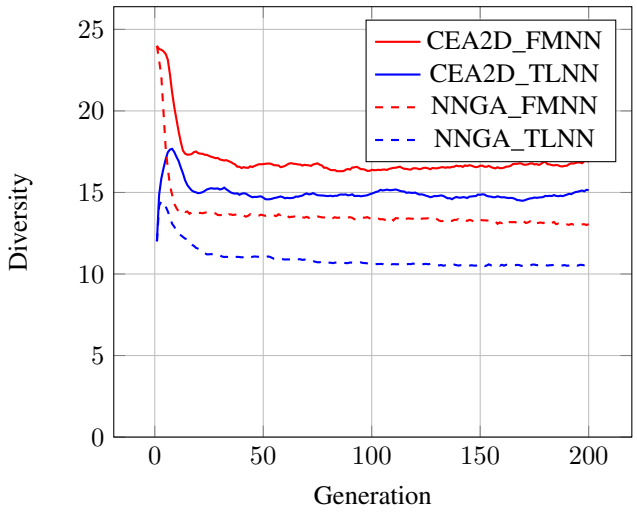

Fig. 3. Diversity against generation number for the XOR-Problem.

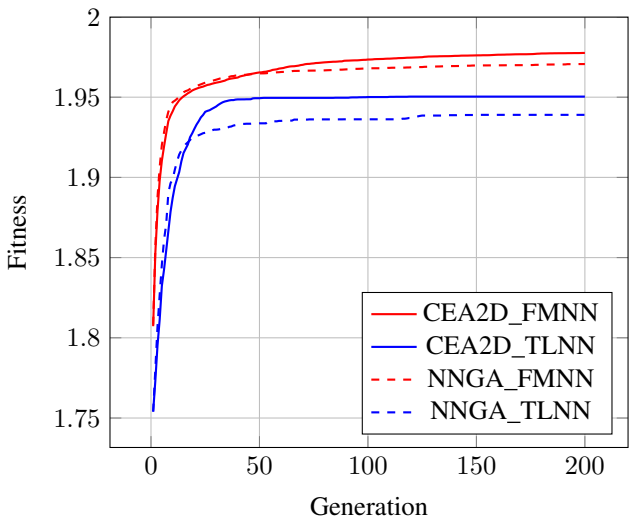

Fig. 4. Fitness against generation Number for the CAM-Problem

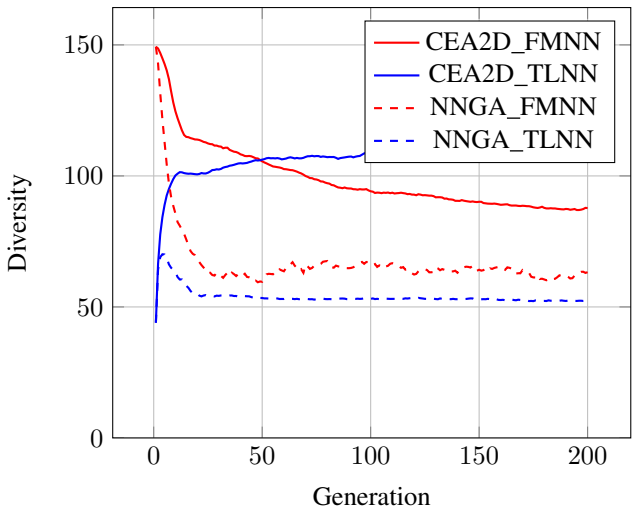

Fig. 5. Diversity against generation Number for the CAM-Problem

A. XOR

Figure 2 shows that the CEA2D performed significantly better than the NNGA for the XOR problem irrespective of whether a Fully Meshed Neural Network or a Three Layered Neural Network was used.

A major difference is that the CEA2D keeps, due to the spatially structured population, a higher diversity between the ANNs in the population than the NNGA. Figure 3 shows the average diversity between all the ANNs in one population.

B. CAM

As a reference picture for the CAM-Problem [3] we selected the Hungarian flag with 6 by 9 pixels and 20 simulation steps ($N = 20$ in Equ. 3). Figure 4 shows that, for this simulation, the NNGA performs similar to the CEA2D in the beginning but after 50 generations, it gets outperformed by the cellular algorithm. Again the CEA2D maintains a significantly higher diversity which is shown in Figure 5.

VI. Conclusion and Future Work

Using FREVO we simulated the XOR and the CAM problems with the two ANNs, FMNN, and TLNN, which were evolved by the EAs CEA2D and NNGA. These simulations showed that for our test cases, the CEA2D performs significantly better than NNGA. Although the two algorithms are comparable in their parameterization and local selection strategies, the CEA2D is able to maintain a higher diversity among its population, allowing the algorithm to find better solutions after 30-50 generations. For the future we are planning to apply the CEA2D for more complex problems, in particular for swarm robotic problems. Therefore, we will extend FREVO with an interface to the robotic simulator ARGoS [18]. We have further made the CEA2D algorithm available in the current FREVO release, available as open source under `http://frevo.sourceforge.net`.

References

[1] E. Alba and B. Dorronsoro. *Cellular Genetic Algorithms*. Operations research/computer science interfaces series. Springer, 2009.

[2] T. Baeck, D.B Fogel, and Z Michalewicz. *Handbook of Evolutionary Computation*. Taylor & Francis, 1997.

[3] W. Elmenreich and I. Fehérvári. Evolving self-organizing cellular automata based on neural network genotypes. In *Proceedings of the Fifth International Workshop on Self-Organizing Systems*, volume LNCS 6557, pages 16–25. Springer Verlag, 2011.

[4] Huai-Kuang Tsai, Jinn-Moon Yang, Yuan-Fang Tsai, and Cheng-Yan Kao. An evolutionary algorithm for large traveling salesman problems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(4):1718–1729, 2004.

[5] 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 on*, 37(4):583–593, 2007.

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

[7] I. Fehervari and W. Elmenreich. Evolving neural network controllers for a team of self-organizing robots. *Journal of Robotics*, 2010.

[8] A.L. Nelson, E. Grant, and T.C. Henderson. Evolution of neural controllers for competitive game playing with teams of mobile robots, 2004.

[9] Matteo De Felice, Sandro Meloni, and Stefano Panzieri. Effect of topology on diversity of spatially-structured evolutionary algorithms. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1579–1586, New York, NY, USA, 2011. ACM.

[10] M. Tomassini. *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time*. Natural Computing Series. Springer, 2005.

[11] E. den Heijer and A.E. Eiben. Maintaining population diversity in evolutionary art using structured populations. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 529–536, June 2013.

[12] E. Alba and José M. Troya. Cellular evolutionary algorithms: Evaluating the influence of ratio. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, and H. Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, volume 1917 of *Lecture Notes in Computer Science*, pages 29–38. Springer Berlin Heidelberg, 2000.

[13] F. Neumann, P. S. Oliveto, G. Rudolph, and D. Sudholt. On the effectiveness of crossover for migration in parallel evolutionary algorithms. In *GECCO*, pages 1587–1594, 2011.

[14] T. Krink, B. H. Mayoh, and Z. Michalewicz. A PATCHWORK model for evolutionary algorithms with structured and variable size populations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1321–1328, 1999.

[15] V. Scott Gordon, Rebecca Pirie, Adam Wachter, and Scottie Sharp. Terrain-based genetic algorithm (tbga): Modeling parameter space as terrain. In Wolfgang Banzhaf, Jason M. Daida, A. E. Eiben, Max H. Garzon, Vasant Honavar, Mark J. Jakiela, and Robert E. Smith, editors, *GECCO*, pages 229–235. Morgan Kaufmann, 1999.

[16] René Thomsen, Peter Rickers, and Thiemo Krink. A religion-based spatial model for evolutionary algorithms. In Marc Schoenauer, Kalyanmoy Deb, Günther Rudolph, Xin Yao, Evelyne Lutton, JuanJulian Merelo, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, volume 1917 of *Lecture Notes in Computer Science*, pages 817–826. Springer Berlin Heidelberg, 2000.

[17] A. Sobe, I. Fehervari, and W. Elmenreich. FREVO: A tool for evolving and evaluating self-organizing systems. In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2012 IEEE Sixth International Conference on*, pages 105–110, 2012.

[18] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012.