FEFFuL: a Few-Examples Fitness Function Learner

Nicolo' Brandizzi [a], Andrea Fanti[a], Roberto Gallotta[a] and Christian Napoli[a]

[a]Department of Computer, Automation and Management Engineering, Sapienza University of Rome, via Ariosto 25 Roma 00185, Italy

Abstract

This paper presents *FEFFuL*, an architecture used to estimate the fitness value of a generated artifact in any Evolution Strategy (ES) system that would otherwise require human evaluation, i.e.: Interactive Evolutionary Computation (IEC) systems. By learning directly human preferences, the *FEFFuL* network aims to reduce user's fatigue to a minimum while also adapting to new emergent artifacts. We apply here *FEFFuL* in the context of evaluating generated structures in the popular game Minecraft.

Keywords

Evolution Strategy, Fitness Estimation, Human-in-the-Loop Control,

1. Introduction

In *Evolutionary Computation* (EC) and *Evolution Strategies* (ES), a global optimization problem is tackled by taking inspiration from biologic evolution: an initial population of solutions is evolved by selection and mutation, producing new solutions with higher values of the task's objective function. In this context, a particular solution is referred to as an *individual*, while its value of the objective function is its *fitness*. *Genetic Algorithms* (GA) are a family of algorithms in EC in which individuals are encoded by their *genotype*, which contains the information to reconstruct the actual solution, called *phenotype*[1, 2]. The genotypes are mutated and combined in different ways to produce the next generation of solutions.

There are several domains in which a fitness function is unknown or very hard to compute, e.g. visual [3, 4, 5, 6] or musical [7, 8] appeal. In *Interactive Evolutionary Computation* (IEC), this problem is overcome by using manual human evaluation to compute the fitnesses of individuals in each generation. One of the main issues of IEC is user fatigue, which greatly limits the amount of human evaluations available; this usually poses bounds both on the number of possible generations and the number of individuals that can be evaluated at each generation. To mitigate this problem the interactions are often carefully designed so as to minimize psychological fatigue [9]. Function approximators (e.g. neural networks) are also often used to learn the preferences of the human experimenter [9, 7], so that this information can be used in subsequent runs of the GA and reduce the amount of interaction needed.

In our experiment, we tackle the generation of interesting structures in the popular videogame *Minecraft*. Minecraft is a sandbox construction videogame set in a voxel-based environment with various basic building blocks, such as wood, stone, glass, water, etc. The major advantage of Minecraft over most Artificial Life (ALife) domains is that surprisingly complex and functional structures (e.g. moving robots, word processors, etc.) can all be built from the same basic building blocks, which aligns well with the few chemical building blocks that produce complex biological systems [10, 11, 12].

User fatigue is a major challenge in IEC, and since the human fatigue threshold cannot be improved, the available evaluations must be exploited as much as possible. Our approach to this issue builds on the techniques mentioned above, by alternating human evaluation and the training of a fitness estimator model in a single run of the genetic algorithm. This approach allows to use few human interactions while still perfoming a high number of generations. Moreover, since the fitness estimator is re-aligned with human preferences almost periodically, it can also adapt to artifacts only seen in later generations. As a convenient side-effect, more than one human (possibly with slightly different preferences) can contribute to a single run, which demonstrates how this approach can be extended from IEC to Collaborative Interactive Evolution (CIE) [13]. Unlike previous approaches such as [14] our approach leverages data collected through the evolution process to further reduce user fatigue even when in an IEC setting.

2. Related work

In this section we give a technical overview of our application of *FEFFuL* to the Minecraft environment.

SYSTEM 2021 @ Scholar's Yearly Symposium of Technology, Engineering and Mathematics. July 27–29, 2021, Catania, IT
brandizzi@diag.uniroma1.it (N. B.);
fanti.1650746@studenti.uniroma1.it (A. Fanti);
gallotta.1890251@studenti.uniroma1.it (R. Gallotta);
cnapoli@diag.uniroma1.it (C. Napoli)
0000-0002-3336-5853 (C. Napoli)

2.1. Artifacts generation

An artifact is a structure with precise width, height and depth. These values are user defined. The artifact is the phenotype of a genome. Each genome uniquely encodes information used to generate the artifact. We use the NeuroEvolution of Augmenting Topologies (NEAT) [15] to control how these genomes evolve over the generations. Each genome is part of a population of fixed size and, thanks to the NEAT algorithm, is mutated to increase the complexity of the resulting artifact. In fact, NEAT ensures an ever-increasing complexification of the artifacts and, thanks to speciation and the use of elitism, it also ensures a lasting diversification of phenotypes (diversity here is measured as genomic distance). Additionally we set a low stagnation level to ensure low-performing genomes (i.e.: genomes that generate uninstering artifacts for the user) are pruned away and leave space in the population for more interesting genomes. During evolution each genome is mutated and new genomes are created by combining two random parent genomes. Since genomes with high fitness are more likely to reproduce and pass on their properties to their offsprings, we ensure that the entire population slowly converges to diverse, high-fitness solutions.

The genomes are used to encode a Compositional Pattern Producing Networks (CPPNs) [16]. Such a network is composed of blocks of elementary activation functions connected together by weighted connections. The genomes encode the structure, weights and biases of the network. Mutations during evolution consist in adding or removing connections or blocks, perturbing the weights and biases values and changing the activation functions. The CPPNs thus directly map from genotype to phenotype without local interaction and are applicable on a infinite-sized input space (a typical application is, for instance, 2D image generation from pixel coordinates). Due to their architecture, CPPNs create patterned outputs and can develop pure symmetry and symmetry with variation, which in turn make for appealing artifacts.

In our experiment, we use the NEAT Python [17] library and the PyTorch-NEAT[1] to integrate both the NEAT evolutionary algorithm and the CPPNs architecture. The inputs to the networks are the scaled X, Y and Z coordinates ($\in [0, 1]$) of the artifact and the outputs are the block type and rotation (both values are output $\in [0, 1]$ and later scaled to the admissible values).

The artifact generation process can be formally expressed as follows: given a population P at the generation g of genomes Gs, we first define the CPPN network that the genome encodes as

$$CPPN_G = \mathrm{dec}(P_G^g)$$

[1] The PyTorch-NEAT library is available at https://github.com/uber-research/PyTorch-NEAT/

where dec is the decoding function. The artifact generation is then simply

$$o = CPPN_G(i)$$

where $i \in R^3$ and $o \in R^2$ are, respectively, the inputs and the outputs of the network, both constrained in the real-valued interval $[0, 1]$. In order to show the artifact as a structure to the user, o is then transformed. By assigning to b the number of admissible blocks and to r the number of admissible block rotations, we have that

$$o_1 = \lfloor o_1 * (b + 1) \rfloor$$

$$o_2 = \lfloor o_2 * (r + 1) \rfloor$$

The user has to choose the admissible blocks and values by modifying a configuration file. The resulting artifacts are then evaluated in the Minecraft game using the EvoCraft [10] interface.

2.2. Artifacts evaluation

The generated artifacts undergo an user defined Minimum Criterion (MC) step at each generation before evaluation. This step removes artifacts that don't satisfy this minimum requirement. Only a predefined number of artifacts sampled randomly from the artifacts that pass the MC step are then presented to the user for direct evaluation. In our experiment, the MC consisted in removing artifacts that didn't contain enough air blocks and enough solid blocks (both values were expressed as minimum percentages). The number of maximum artifacts that could be presented to the user was set to 24.

The user is then tasked to evaluate the generated artifacts. This is done by looking at the structures on the Minecraft client applet and choosing the most interesting ones. During manual (human) evaluation the program accepts a list of indexes that correspond to the selected artifacts. Since the mapping from genomes to artifacts is 1:1, we can assign the fitness value to the correct corresponding genome that generated the artifact. The possible fitness values are 1 for the genomes the user is interested in and 0 for the others. We note that we automatically assign a fitness value of 0 to all the genomes that didn't pass the MC step to discourage their traits to appear again later in the generations, as these are not interesting to the user. This ensures that only the phenotypically interesting genomes survive throughout the evolution process.

During human evaluation we save the artifacts and their fitness in a memory buffer. Once the buffer is at capacity, the *FEFFuL* network can be trained in a supervised fashion to directly estimate the user preferences given the artifacts. The network itself is rather simple: a 3D

Convolution maps the artifact to a sequence of feature maps that are later flattened and passed through multiple residual blocks that in the end output a value $\in [0, 1]$ that represent how likely the human would have marked the artifact as interesting. A graphical overview of the *FEFFuL* network can be found in 1.

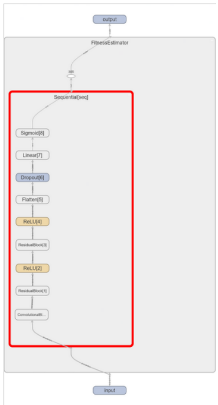

Figure 1: Architecture of the *FEFFuL* network

Due to the nature of the task, it is expected that the buffer would be unbalanced: it is more likely that there would be more examples of discarded artifacts than accepted artifacts. For this reason, we balance the dataset before training the network. We do this by both oversampling accepted artifacts and then downsampling discarded artifacts. The former is accomplished by augmenting artifacts by rotation: we rotate the structure along the Y-axis by 90. We can do this since the artifact must be interesting regardless of the orientation. The latter is instead accomplished by removing discarded artifacts from the dataset until a good ratio of accepted artifacts and discarded artifacts is reached.

The output value of the *FEFFuL* module is directly assigned as fitness value to the corresponding genome. This gives an additional significance to the fitness value: it not only marks the user preference but also the network's confidence in its output. Thus, a low-fitness genome would have been less likely to have been picked by the user than a high-fitness one. This process is important during the reproduction step of the NEAT algorithm as it heavily relies on the genome's fitness to order the possible parents of new individuals.

2.3. The alignment problem

Due to the ever-complexification of the genomes thanks to the NEAT algorithm, both the structures and the user's preferences change over time. An artifact that the user deemed interesting at generation 0 may not be as interesting if found at generation 100. This implies that also the *FEFFuL* network has to reflect this behavior. We solved this problem with two simple corrections to the network's behavior. First, we note that the buffer is constantly updated only with user's selected artifacts and fitnesses, effectively making it a constantly-updated dataset we can train the network from. The corrections are the following:

1. We first use the network to evaluate the artifacts for a given number of generations at a time. This value is increased as the evolution process goes on, thus leveraging the network more and more as time goes on.
2. We enforce fine-tuning of the network after the aforementioned number of generations. We do this by prompting the user to evaluate the current generation, collect the preferences in the buffer and finetuning the network on the updated dataset. At this point of the process there could be some disalignment between user preferences and network estimates. This would be harmful to the entire experiment as it would diverge the search. We solve this by activating the network only if its accuracy over unseen artifacts is higher than a given threshold. Otherwise the user is prompted again until the network can be activated.

3. Methodology and Results

We first report the experiment settings in order to reproduce our results at 1. These are taken from the *experiment* configuration file in our repository; we are not going to report the settings for the NEAT algorithm which can be found in the *neat* configuration file in our repository.

Artifact width	5
Artifact height	7
Artifact depth	5
Max number of artifact shown to the user	24
Min percentage of air blocks in an artifact	22%
Min percentage of solid blocks in an artifact	32%
Admissible rotations	NORTH, WEST, SOUTH, EAST, UP, DOWN
Admissible blocks	AIR, COBBLESTONE, STONE, STONE_STAIRS STONEBRICK, QUARTZ_BLOCK
RNG seed	42
Buffer capacity	512
Batch size	32
Number of convolution channels	16
Number of epochs	100
Interval of training	5
Activation threshold on test accuracy	0.5

Table 1
Experiment parameter summary

We then report the train and test accuracy and loss evaluated at different generations at 2 and 3 respectively. We note that, while the training metrics behave as expected, in the test we can see a rising trend for the loss metric. The accuracy however remains well over the 80%, which we consider a good value.

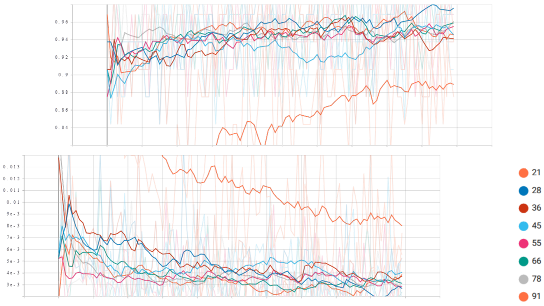

Figure 2: Train accuracy and loss of the *FEFFuL* network at different generations

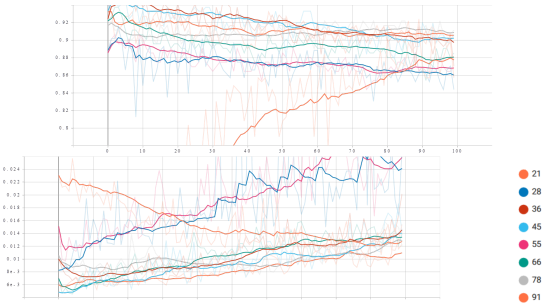

Figure 3: Test accuracy and loss of the *FEFFuL* network at different generations

We ran the experiment for 100 generations. While this is a rather low number of generations, the aim of this project was not to evolve structures to a certain level of user satisfiability but to show that it was possible to use a NN to approximate user's preferences, reduce user fatigue and save time. The metrics prove that we succeeded in the first part of the task. For what concerns user fatigue, it is known that a single user can evaluate around 20 generations worth of artifacts before starting to feel fatigued [9]. In the entire evolution process only 29 generations worth of artifacts where evaluated by the user, whereas the remaining 71 were evaluated by the *FEFFuL* network. We report a graph that shows which evaluator was used at each generation at 4.

The time gain over the same amount of generations is well apparent: we estimated that the human user takes from 2 to 3 minutes to evaluate a single batch of artifacts, whereas the *FEFFuL* network only a few milliseconds. In fact we noticed that the bottleneck of time consumption

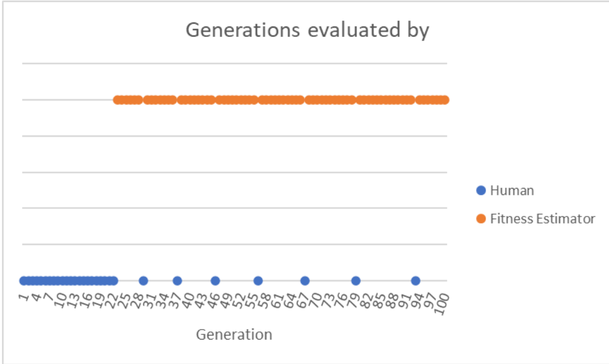

Figure 4: Overview of the active evaluator per generation

was not the evaluation anymore but the generation of the new genomes instead.

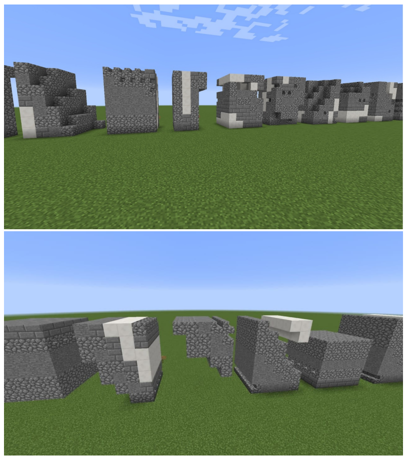

Figure 5: Artifacts produced on generation 0 (left) and on generation 100 (right)

Finally, we report a comparison between the artifacts generated on generation 0 and those generated on generation 100 at 5. We note how the artifacts are more complex on generation 100 as well as in line with user's preferences during evolution.

4. Conclusions and future work

The results shown in the previous section suggest that this approach can effectively be used to mitigate the user fatigue problem in IEC tasks. More specifically, most of the interactions with the human experimenter are in the first few generations, where the buffer is not completely full and the estimator cannot be trained. After the first alignment, the human evaluations were needed very sporadically, as the estimator maintained good alignment and usually only needed one generation to adapt to new artifacts chosen by the user. This also means that after the initial alignment, the cost to perform more generations, in terms of user evaluations needed is much lower. This can get even lower for further generations, even though this descending trend would be fairly moderate in our specific implementation.

However, these same results also suggest that a method is needed to avoid the fitness estimator from overfitting during the fine-tuning step, perhaps with an adaptive rule that either produces an optimized "expiration date" for the model or that dynamically stops the fine-tuning process when overfitting occurs.

We finally note two possible improvements to the *FEFFuL* network. *FEFFuL* could use a latent representation of the phenotypes instead of directly mapping the artifacts to their fitness values. This could benefit the re-alignment process if the generalization power of the model over unseen artifact improves. This comes with the added benefit of keeping a smaller buffer, thus requiring even less human evaluation at the beginning of the experiment. Finally The Minimum Criterion could be enforced only in generations evaluated by the human, so that the advantage of having a fitness estimator can be exploited to evaluate the entire population and not just a subset. This would also ensure that no interesting artifact is discarded *a priori* during the MC step, allowing a diverse set of good solutions to remain in the population.

References

[1] D. Połap, M. Woźniak, C. Napoli, E. Tramontana, R. Damaševičius, Is the colony of ants able to recognize graphic objects?, Communications in Computer and Information Science 538 (2015) 376–387. doi:10.1007/978-3-319-24770-0_33.

[2] D. Połap, M. Woźniak, C. Napoli, E. Tramontana, Real-time cloud-based game management system via cuckoo search algorithm, International Journal of Electronics and Telecommunications 61 (2015) 333–338. doi:10.1515/eletel-2015-0043.

[3] J. Secretan, et al., Picbreeder: evolving pictures collaboratively online, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2008, pp. 1759–1768.

[4] G. Capizzi, G. Lo Sciuto, C. Napoli, E. Tramontana, M. Woźniak, A novel neural networks-based texture image processing algorithm for orange defects classification, International Journal of Computer Science and Applications 13 (2016) 45–60.

[5] R. Avanzato, F. Beritelli, M. Russo, S. Russo, M. Vaccaro, Yolov3-based mask and face recognition algorithm for individual protection applications, volume 2768, 2020, pp. 41–45.

[6] M. Wozniak, C. Napoli, E. Tramontana, G. Capizzi, G. Lo Sciuto, R. Nowicki, J. Starczewski, A multiscale image compressor with rbfnn and discrete wavelet decomposition, volume 2015-September, 2015. doi:10.1109/IJCNN.2015.7280461.

[7] B. Johanson, R. Poli, Gp-music: An interactive genetic programming system for music generation with automated fitness raters, Genet. Program. (2000).

[8] J. Biles, et al., Genjam: A genetic algorithm for generating jazz solos, in: ICMC, volume 94, Ann Arbor, MI, 1994, pp. 131–137.

[9] H. Takagi, Interactive evolutionary computation: fusion of the capabilities of ec optimization and human evaluation, Proceedings of the IEEE 89 (2001) 1275–1296. doi:10.1109/5.949485.

[10] G. S. R. Djordje, et al., EvoCraft: A New Challenge for Open-Endedness, arXiv (2020). URL: https://arxiv.org/abs/2012.04751v1. arXiv:2012.04751.

[11] M. Woźniak, D. Połap, C. Napoli, E. Tramontana, Application of bio-inspired methods in intelligent gaming systems, Information Technology and Control 46 (2017) 150–164. doi:10.5755/j01.itc.46.1.13872.

[12] D. Połap, M. Wózniak, C. Napoli, E. Tramontana, Is swarm intelligence able to create mazes?, International Journal of Electronics and Telecommunications 61 (2015) 305–310. doi:10.1515/eletel-2015-0039.

[13] S. R. Szumlanski, A. S. Wu, C. E. Hughes, Conflict resolution and a framework for collaborative interactive evolution, in: AAAI, 2006, pp. 512–517.

[14] P. G. de Prado Salas, S. Risi, Collaborative interactive evolution in minecraft, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 127–128. URL: https://doi.org/10.1145/3205651.3205723. doi:10.1145/3205651.3205723.

[15] K. O. Stanley, Efficient Evolution of Neural Networks Through Complexification, Ph.D. thesis, Department of Computer Sciences, The University of Texas at Austin, 2004. URL: http://nn.cs.utexas.edu/?stanley:phd2004.

[16] K. O. Stanley, Compositional pattern producing networks: A novel abstraction of development, Genetic Programming and Evolvable Machines 8 (2007) 131–162. URL: https://doi.org/10.1007/s10710-007-9028-8. doi:10.1007/s10710-007-9028-8.

[17] S. Sarti, G. Ochoa, A neat visualisation of neuroevolution trajectories., in: EvoApplications, 2021, pp. 714–728.