Neural-Network based Adaptation of Variation Operators' Parameters for Metaheuristics*

Tymoteusz Dobrzański$^{[0000-0001-5156-8585]}$, Aleksandra Urbańczyk$^{[0000-0002-6040-554X]}$, Tomasz Pełech-Pilichowski$^{[0000-0003-2212-7806]}$, Marek Kisiel-Dorohinicki$^{[0000-0002-8459-1877]}$, and Aleksander Byrski$^{[0000-0001-6317-7012]}$

AGH University of Science and Technology
tdobrzanski@student.agh.edu.pl,{aurbanczyk,tomek,doroh,olekb}@agh.edu.pl

Abstract. The paper presents an idea of training an artificial neural network a relation between different parameters observed for a population in a metaheuristic algorithm. Then such trained network may be used for controlling other algorithms (if the network is trained in such way, that the knowledge gathered by it becomes agnostic regarding the problem). The paper focuses on showing the idea and also provides selected experimental results obtained after applying the proposed algorithm for solving popular benchmark problems in different dimensions.

Keywords: adaptation of variation operators' parameters · evolutionary algorithms · neural networks.

1 Introduction

Maintaining the balance between exploitation and exploration in metaheuristic is an important task in order to avoid premature loss of convergence and getting stuck in a local extremum by a population processed. The exploitation and exploration are rather volatile notions, but can be approximated by measuring the diversity, thus based on diversity one might propose way for controlling the search of a metaheuristic algorithm in order to avoid pitfalls (like already mentioned premature convergence). Different approaches were proposed in order to adapt the parameters of variation operators (based on dedicated rules, or e.g. modifying the parameters of mutation by the same algorithm which was used for solving the problem), but the intuition points out that the actual relation between different parameters observed in the population and the features of exploitation/exploration is complex.

* The research presented in this paper has been financially supported by: Polish National Science Center Grant no. 2019/35/O/ST6/00570 "Socio-cognitive inspirations in classic metaheuristics." (A.U.) and Polish Ministry of Education and Science funds assigned to AGH University of Science and Technology (T.P-P., M.K-D., A.B.)

Why don't we use ANNs (Artificial Neural Networks) to grasp this complex relation and provide the metaheuristic with the knowledge required for reasonable (based on many experiments, not arbitrarily chosen rules) adaptation of the variation operators parameters? This is actually the idea proposed in this paper: to find a way for train an artificial neural network, gathering necessary knowledge and being able to reuse this knowledge in a similarly-defined (but not the same) problems.

In the course of this paper, after referring to the state of the art regarding neural networks and adaptation of the variation operators of metaheuristics, we discuss the proposed algorithm clearly, then show the experimental setting and provide the insight into first efficacious experiment results, finally we conclude our paper and point out the future work directions.

2 Neural networks in prediction and control

For prediction and control purposes a number of methods and approaches can be exploited. The goal is to perform reliable data processing based on input data, initially preprocessed. An application of a predictor is based on adjusting its parameters (in a fixed or adaptive way). One of prediction approaches is to use neural networks instead of numerical formulas. Considering general architecture of neural networks, a neuron processes a set of input data, according to fixed weights and a bias. Supervised or unsupervised training is a process of obtaining network parameters. For example, simple time series one-step-ahead prediction with a neural network containing one input layer is comparable to a linear regression, nevertheless, prediction can be performed based on results of data classification.

A number of neural models applied for prediction purposes have been proposed. The presented approaches differ in architectures, objective functions or a paradigm used [6]. An application of a model relies on the prediction attributes, time-horizon and input data quality (sampling, consistency, statistical and frequency characteristics). For a given time horizon, predictions – as one step or multistep - can be computed as point forecasts (results in point estimates of values for a fixed horizon) or probabilistic ones (the distribution of future data is produced). For prediction purposes Multilayer Perceptron (MLP) networks, Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) can be used [8], [29]. MLPs are a class of backpropagation neural networks used for prediction and regression of structured input data. A key feature is an ability of mapping from input datasets to output variables. MLPs can be used i.a. for time series classification and prediction – as mapping from input data to output data [27].

Neural networks (NNs) are applicable for dynamic process control (in particular, in industrial applications) [30, 22] where processes are complex and they have to be operated with a small number of operators (computer systems are widely used for a process control and supervision) and automatic adjusting values of processes is required. Process control is based on data processing (i.a.

set points, estimated states) aimed at obtaining present and future values, for example for adaptive control purposes. Prediction of a system future behavior is essential, and a set of constraints is satisfied (in particular, in real-time systems [23]). Model Predictive Control (MPC) is focused on finding (computing) control by solving optimization problem in a loop according to predicted output (the behavior of the system) and fixed time-horizon. Neural networks can be exploited to adjust parameters of controllers of applied algorithms based on analytical calculations due to their limitations arising from a large number of variables, parameters and control loops as well as a need of performing computation in a real time systems according to specified time limits. NNs can be developed for solving optimal control problems and for tasks requiring the use of control loop, in particular – as NN-MPC (Neural Network based Model Predictive Control).

3 Adaptation of evolutionary search parameters

In the group of evolutionary algorithms (Genetic Algorithms, Evolution Strategies and Evolutionary Programming and many hybridised versions of them) the key to success is good parameters setting. In the classic versions of them certain values for parameters (selection, mutation and crossover) were set on the beginning, very often by trial and error. Many researchers realised that setting them without the knowledge about the structure of the problem lead to sub-optimal solutions and result in another trial (known as "parameter tuning problem" [28]). Besides other solutions [13], the idea of parameter control grew on popularity. [17]. It states that instead of tuning the parameters in advance, these can be observed and adjusted during the run. This shift had many advantages, from releasing researchers from duty of initial parameters' setting, to broadening the spectrum of problems that can be solved by EAs (Evolutionary Algorithms).

The overview and classification of all possibly controlled parameters is extensively described in [17]. One of the axis of the classification is whether the method of control is dependent on the parameters itself or it is rather generic one.

There are numerous control methods intended for specific parameters. They can be based on one or few parameters combined. One of the most explored method related to one parameter is controlling and adaptation of mutation step size (σ), originally employed in ES (Evolution Strategies). The early ideas circled around modifying the step size proportionally to the distance to the optimum [26],[3] or increasing/decreasing it by 0.2 accordingly to the success rate (*one-fifth rule*) [25],[5]. The creation of the self-adaptive mutation was the next great leap in adaptive ESs: the mutation parameters (both the step-size and the covariance matrix) are tied to each individual and are also subject to mutation [9]. However, there is an algorithm that outperforms all of them and it is based on the adaptation of the full covariance matrix [16],[14]. The core idea behind CMA-ES (Covariance Matrix Adaptation ES) is to use the algorithm's route to deterministically update the various mutation parameters. If the algorithm has taken a sequence of steps in the same direction, the step size should be increased to allow for greater steps and faster processing. The covariance matrix update is affected by similar ideas. Aside from single-parameter techniques, there have been numerous attempts to improve EAs by creating ways for controlling multiple parameters at the same time (e.g. [4],[19]. However, they usually change components other than just variation parameters, therefore a detailed description is beyond the scope of this article.

The group of generic parameter control methods is the one with which we associate our work. In the recent extensive review of those methods by Gomes Pereira De Lacerda et al. [15] there are two main groups of general parameter controllers - one based on reinforcement learning and the other on prediction of probability of success for parameters at each point of time. The other attempts to design generic parameter control were based on Dynamic Programming under Markov assumption [2], fuzzy logic [18] and Bayesian networks [10]. We found no evidence of attempting to incorporate ANN as a generic controller for EA in the review or through our own research.

4 ANN-based adaptation of evolutionary search

The idea of adaptating the parameters of metaheuristics' variation operators is not new, and we have cited several seminal papers in Section 3 to give the background on this. Observing the state-of-the-art algorithms we stated, that those approaches usually utilize the information perceived in the population, based on fixed rules. A notable approach is the one applied in evolution strategies, where the parameters of the probability distribution of mutation undergoes similar evolution as the genetic information contained in the individuals. Not degrading any of the approaches, we would like to express our deep conviction, that apparently, a relation between the observations of certain parameters of the population and the current (desired) parameters of the variation operators (which would help in attaining balance between exploration and exploitation) is complex. Thus, instead of assuming certain fixed rules, we propose to teach a dedicated neural network in a way agnostic to the actual problem, in order to use the knowledge gathered in the network to control the variation operator's parameters for similar problems.

The idea of the proposed approach is shown in Fig. 1. The idea consists in training a dedicated neural network, to learn the complex relation between a number of different population-related parameters and the parameters of the variation operators. The presented approach assumes that the neural network is trained in a supervised way. The trained data are the selected parameters of the population (e.g. diversity-related measures), and they are paired with the desired parameters of the variation operators (e.g. probability of applying crossover and mutation, mutation range).

In this paper we have focused on supervised training (in the future we will also consider unsupervised approaches). Now, in order to train the network, which should grasp the relation between the observed parameters and the parameters of the variation operators, we assume that those parameters (in the discussed case)

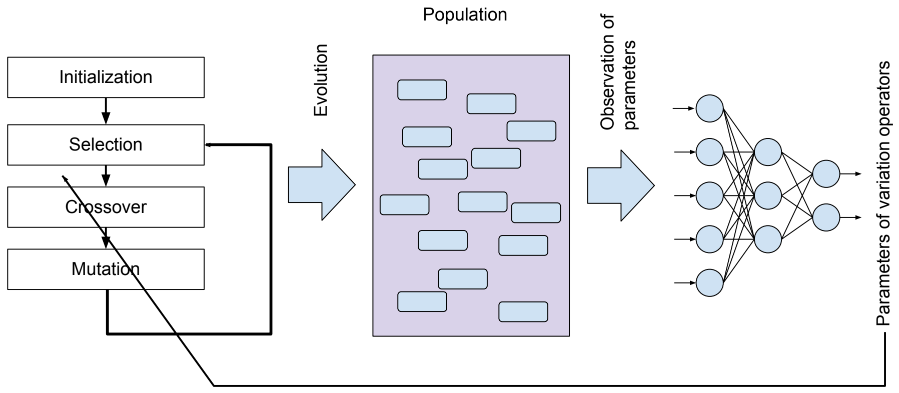


Fig. 1: Neural adaptation of variation operators' parameters

should have certain fixed relation with the exploration/exploitation phenomena. We assume, that an arbitrarily chosen function should, in the training phase, describe this relation (as a reasonable one) and become the template which will be later generalized by the network, applied to different problems. The function used in the presented approach is actually sigmoid-like (the details are given in Section 5) starting with high values, and gradually descending to lower values (yet not zero), assuming that a certain loss of diversity must be observed (as the population gradually ceases to show a "monte-carlo like" behavior and focuses on exploitation, still having certain exploration capabilities). The idea of training the network is shown in Fig. 2. Thus the network is trained on several

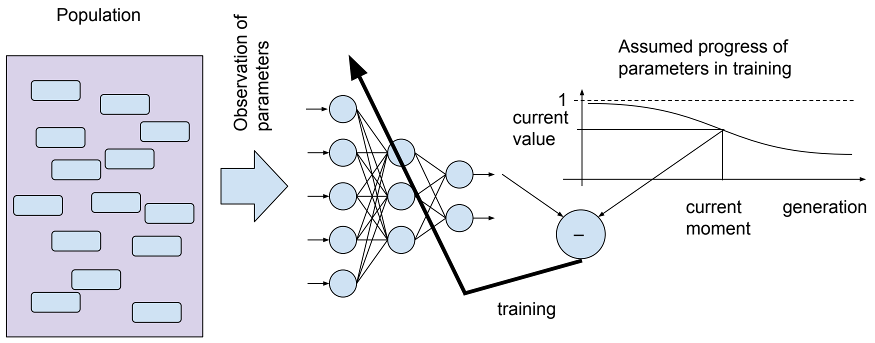

Population
Observation of parameters
Assumed progress of parameters in training
1
current value
current moment
generation
−
training

Fig. 2: Training the network

selected benchmark problems, different chosen functions can be used for showing the progress of the parameters in training, and all the training related parameters are prepared in a way as agnostic as possible to a considered problem, the actual shape and dimension of the search space etc. so the knowledge gathered in the network (regarding the relation between the parameters and the variation operators' parameters) can be reused.

5 Experimental results

In our experiments we have used as our metaheuristic algorithm a classic version of evolution strategy (μ, λ) algorithm applied to solving real-value optimization problems [20]. We have used apopulation of 100 individuals and as a stopping condition – reaching 10000 evaluations of the fitness function. We have repeated each experiment 1000 times, except results shown in fig 3 and 4, where we show result from single algorithm run.

Artificial Neural Networks (feed-forward fully connected Neural Network, with Rectified Linear Unit activation in hidden layers, and sigmoid activation in output layer) used in our work were built and trained using TensorFlow framework [1]. Evolution Strategy was implemented based on jMetalPy framework [7]. The benchmarks and ANN training process were evaluated on Windows based machine with AMD Ryzen 5 3600 (3.59GHz) CPU, Nvidia GeForce GTX 1650 GPU and 16 GB of RAM memory.

The observed parameters of the population (after each generation) were:

- Current evaluation number.
- Standard deviation diversity (stddev) [31] - standard deviation of fitness values across population (P). Computed as:

$$stddev(P) = \sqrt{\frac{\sum_{i=1}^{n}(f_i - \overline{f})}{n-1}}$$

  Where: P – population, n – population size, f_i – fitness value of i individual, $\overline{f}$ – mean fitness value.
- Phenotype diversity (ptype) [31] - number of unique values (U) of fitness across population. Computed as:

$$ptype(P) = \frac{U-1}{n-1}$$

  Where: U – number of unique fitness value, n – population size This value was not used as input for ANN.
- Distance diversity (distance) - mean distance between all pairs of individuals. Computed as:

$$distance(P) = \frac{2}{(n-1)n}\sum_{i=1}^{n-1}\sum_{k=i+1}^{n} dist(P_i, P_k)$$

  Where: $dist$ – distance metric, P_i – genotype of i individual, n – population size. We can use different distance metric (e.g. Hamming for binary genotype) to compute this type of diversity measure.

- Moment of Inertia diversity (MoI) [21] - calculating system moment of inertia to define diversity. Computed as:
  Centroid (central point of system) defined as vector $C = (c_1, c_2, \ldots, c_d)$:

  $$c_i = \frac{\sum_j^n x_{ij}}{n}$$

  Where: x_{ij} – value of i gen in j individual, n – population size
  Moment of Inertia:

  $$MoI(P) = \sum_i^d \sum_j^n (x_{ij} - c_i)$$

  Where: x_{ij} – value of i gen in j individual, n – population size, d - problem dimensionality. In Artificial Neural Network as an input value, we used:

  $$Input = MoI_c - MoI_r$$

  Where: MoI_c – MoI in current population, MoI_r – desired MoI diversity, calculated from sigmoid function multiplied by starting value of MoI.
- Current Mutation probability.
- Current Mutation range – determines range in which parameter can mutate
- Current Crossover probability.
- Current Crossover distribution index – parameter used in Simulated binary (SBX) crossover [11], to determine how far from parents, children's will be placed. Higher value means closer placement.
- Percentage loss of MoI diversity from start of algorithm runtime.
- Percentage loss of MoI diversity from last generation.

It is important to notice that almost all mentioned parameters are highly dependent on selected problem. Size of fitness function space or values range will have impact on computed parameters as well as prepared model. In order to make trained Neural Networks problem agnostic, diversity measurements have to be normalized. In our work, selected values were normalized by starting value of that parameter. This solution allow values to exceed 1, but it's not considered as a problem for trained ANN.

The biggest challenge in our work, was to create problem agnostic Artificial Neural Network, that can be used on any optimization function in any dimesionality, despite fact that selected parameters are size and dimension depended. For example MoI diversity in Ackley problem reaches 110 units, but in Griewank functions, that values can exceed 1400 units (in 100 dimensions). To achieve agnostic model we used normalization by starting value, that may not be optimal solution, but currently it works fine. In further work we will try to propose another, more efficient, normalization algorithm. Taking this information into consideration all size dependent parameters (e.g. Distance diversity, Standard deviation diversity) were normalize to create input in range [0, 1], other values that does not depend on problem (e.g. variation operators, percentage loss of MoI diversity) were used without preprocessing.

We have focused on popular benchmark functions: Ackley, Griewank, Rastrigin, Rosenbrock, Schwefel and De Jong [12]. The Ackley function (in 100 dimensions) was first used for training the network. All the problems were set in 100 dimensions except Rosenbrock which was set in 10, 20, 50, 100, 200 and 500 dimensions.

The parameters of the sigmoid function $sigm(x) = \frac{1}{1+e^{c_1 * x - c_2}}$ describing the progress of the variation operators' parameters were: $c_1 = \frac{10}{m_e}$, m_e - number of maximum evaluations, $c_2 = 5$. Values range of this variant of sigmoid function is [0.99, 0.01]. That should demonstrate balance between exploration and exploitation phase of algorithm.

We have started our experiments with training the selected feed-forward networks having 10 input neurons observing the above-mentioned parameters, 2 output neurons producing the probabilities of crossover and mutation and

- Variant 1 - Hidden Layers size: 24-64-92-24-56-32-24-12
- Variant 2 - Hidden Layers size: 36-64-128-64-18-32-24-12
- Variant 3 - Hidden Layers size: 36-64-128-64-18-32-24-12 It has same hidden size that Variant 2, but training dataset was prepared in different way. Normally variations operators were adapted by sigmoid function, in this case they were generated randomly after each algorithm generation.
- Variant 4 - Hidden Layers size: 36-64-128-64-6-32-24-12.

We have used Ackley function for training. Hidden size was set up using trial and error method. Training dataset was prepared by evaluating 2000 runs of algorithm with adaptive mutation and crossover probability parameters and saving those mentioned above parameters.

Artificial Neural Network training parameters:

- Dataset size: 200000 records
- Validation dataset size: 20% of dataset
- Epochs: 4000
- Batch size: 512
- Learning rate: 0.001
- Optimizer: Adam
- Loss function: Mean Square Error (mse) [24]. Computed as:

$$mse = \frac{\sum_{i=1}^{n}(y_i - \lambda(x_i))^2}{n}$$

Where: n - number of outputs, y_i - true value, $\lambda(x_i)$ - model output.

The trained network was evaluated on the Ackley benchmark (applied to adapt the parameters of the variation operators) and we provide the insight into the process of optimization and the final values obtained at the end of computing.

In Fig. 3 the progress of the best fitness observed for different versions of neural network adapting the parameters of the variation operators is shown. Apparently the idea of ANN-controlled adaptation taught based on previously given sigmoid progress function works – the baseline algorithm (without the adaptation) does not get so close as the algorithms using adaptation. Moreover, we have also tested the algorithm with adaptation of variation parameters without the neural network used (so it was actually the algorithm used for teaching the ANNs – the results were similar as in the case of the baseline algorithm (see Fig. 5a, Adaptive ES version of the algorithm). Thus we claim the idea makes sense and delve into more detailed experiments. Let us check, what is actually

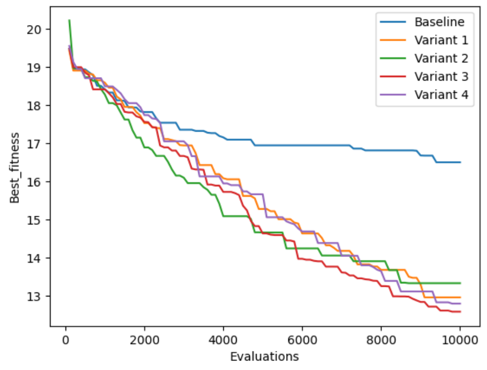
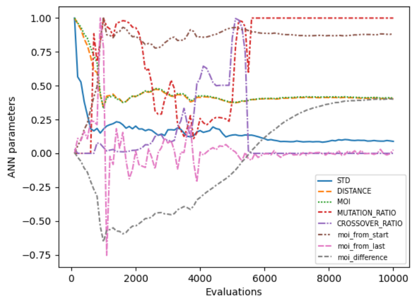
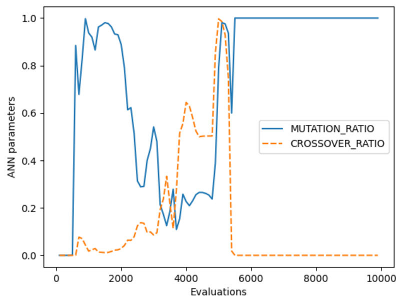


Fig. 3: Best fitness dependent on the number of evaluation, Ackley, 100 dimensions

happening in the course of computation, how (and if at all) does the ANN work (see Fig. 4). One can see that the actual relation between the observed parameters and the output (probabilities of mutation and crossover) is complex (as expected). Mutation and crossovers observed at the input are similar to the ones



(a) Input

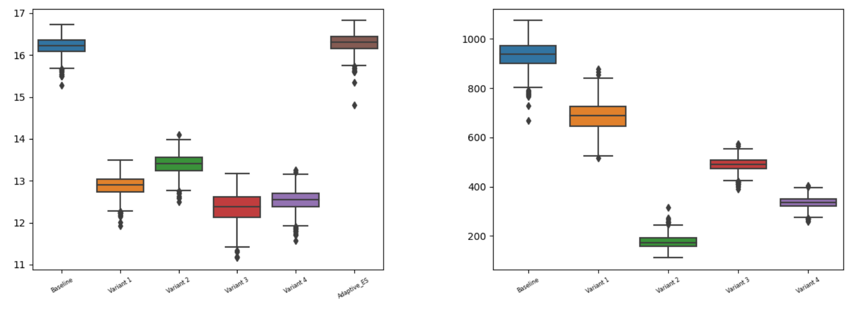


(b) Output

Fig. 4: Monitoring input and output parameters for the ANN used, Ackley 100 dimensions

observed at the output (as these values are in fact controlled by the network). One can observe the increasing mutation probability in the beginning of computation, then for some time the crossover becomes the more frequently used operator, however finally the crossover is apparently discarded (at the moment when the measurement of MoI-From-Last becomes stable, later no peaks are observed in the graph showing this measure). This situation seems to be normal, after some time in this experiment the diversity of the search is lost, and because of that the mutation is used for escaping the local minimum (probably attained), in other words, to escape the situation when most (or all) of individuals are very similar.

In Fig. 5a we can observe the final values obtained at the end of computing for Ackley problem. It is easy to see, that the neural versions of the algorithm are significantly better than the version without the neural adaptation. At the same time, in Fig. 5b we can observe the final results obtained for Griewank problem. This time there is one neural network which turned out to be the best in adaptation. This is completely normal and highly possible situation, that only one network prevailed. Usually choosing the optimal parameters of neural network is realized in a tedious process of trial and error. Apparently the most

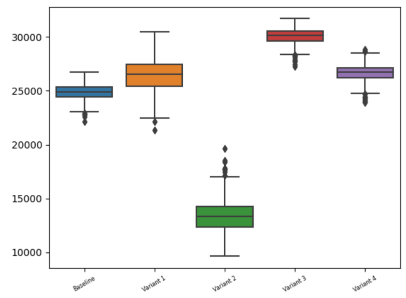


(a) Ackley (b) Griewank

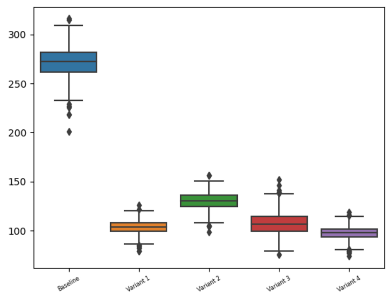
Fig. 5: Final fitnesses for Ackley and Griewank problems, 100 dimensions

complex problem is Schwefel (see Fig. 6a, as the results are far from optimum, however still one network prevails in this case. This might be an apparent case for more difficult problems, that only one network will prevail. Actually this observation is confirmed when observing the final results obtained for Sphere function (Fig. 6b), which is the easiest problem considered (this is a convex function). This time again the ANN-adapted versions of the algorithm prevail and it is difficult to tell, which structure of the network was the best.

Finally let us check what happens if we change the dimensionality of the problem. Thus let us focus on Rosenbrock problem in different dimensions (see Fig. 7). Now it is easy to see that again, for simpler problems (20 and 50 dimensions) the algorithms show similar efficacy, while for more difficult problems



(a) Final fitness for different ANNs, Schwefel, 100 dimensions



(b) Final fitness for different ANNs, Sphere, 100 dimensions

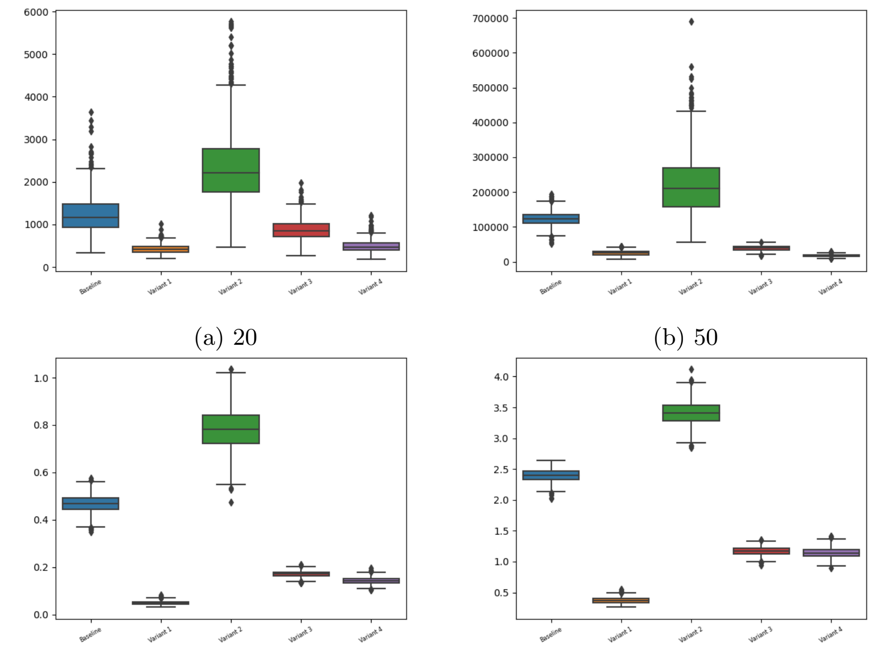
Fig. 6: Final fitnesses for Schwefel and Sphere problems, 100 dimensions

(200 and 500 dimensions) one or several algorithms prevail. Moreover – it seems that the adaptation helped a lot in the case of those more difficult problems (the results are much closer to the global optimum than in the case of 20 and 50), at the same time acting mediocre (or being more or less useless) for simpler problems.

6 Conclusion

We have presented a novel approach for adaptation of variation operators' parameters in metaheuristic algorithms, based on artificial neural networks. The idea consists of training the network on dedicated benchmark problems, working on the parameters of the search (and of the network) to make the knowledge gathered by the network agnostic to the problem, search space etc. and applying such network to adapt the parameters in other problems. We believe that the results presented are interesting, but we are sure that much more is to be done to further tune the algorithm and generalize it. Our plans for the future research are as follows:

– We have used a sigmoid function as a function showing the rational progress of the variation parameters – we will try experimenting also with other functions, to further explore our assumptions related the reasonable control over exploration and exploitation.
– We have followed the supervised learning paradigm, in the future we will also try to use unsupervised learning (e.g. hebbian learning) in order to avoid assuming concrete parameter progress functions.
– We will try to explore the relation between the network and the number of evaluation of the fitness function, perhaps this relation might be also generalized or omitted.
– In the presented research we have used one ANN for both parameters of variation operators, this was a simplification – we are sure we should apply different ANNs for each parameter controlled.



(a) 20 (b) 50

(c) 200 (d) 500

Fig. 7: Final fitnesses for Rosenbrock for different dimensions (20,50,200,500)

– We will aim at attaining agnostic knowledge gathered in the network in order to be able to easily reuse the trained network.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
2. Aine, S., Kumar, R., Chakrabarti, P.: Adaptive parameter control of evolutionary algorithms under time constraints. In: Applications of Soft Computing, pp. 373–382. Springer (2006)
3. Auger, A., Le Bris, C., Schoenauer, M.: Dimension-independent convergence rate for non-isotropic $(1, \lambda)$ - es. In: Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation: PartI. p. 512–524. GECCO'03, Springer-Verlag, Berlin, Heidelberg (2003)

4. Bäck, T., Eiben, A.E., van der Vaart, N.A.: An emperical study on gas “without parameters”. In: International conference on parallel problem solving from nature. pp. 315–324. Springer (2000)
5. Bassin, A., Buzdalov, M.: The 1/5-th rule with rollbacks. Proceedings of the Genetic and Evolutionary Computation Conference Companion (Jul 2019). https://doi.org/10.1145/3319619.3322067, http://dx.doi.org/10.1145/3319619.3322067
6. Benidis, K., Rangapuram, S., Flunkert, V., Wang, B., Maddix, D., Turkmen, A.C., Gasthaus, J., Bohlke-Schneider, M., Salinas, D., Stella, L., Callot, L., Januschowski, T.: Neural forecasting: Introduction and literature overview. https://arxiv.org/abs/2004.10240 (2020)
7. Benítez-Hidalgo, A., Nebro, A.J., García-Nieto, J., Oregi, I., Del Ser, J.: jmetalpy: A python framework for multi-objective optimization with metaheuristics. Swarm and Evolutionary Computation **51**, 100598 (2019). https://doi.org/https://doi.org/10.1016/j.swevo.2019.100598, https://www.sciencedirect.com/science/article/pii/S2210650219301397
8. Botalb, A., Moinuddin, M., Al-Saggaf, U.M., Ali, S.S.A.: Contrasting Convolutional Neural Network (CNN) with Multi-Layer Perceptron (MLP) for big data analysis. In: 2018 International Conference on Intelligent and Advanced System (ICIAS). pp. 1–5 (2018). https://doi.org/10.1109/ICIAS.2018.8540626
9. Bäck, T.: Self-adaptation in genetic algorithms. In: Proceedings of the First European Conference on Artificial Life. pp. 263–271. MIT Press (1992)
10. Corriveau, G., Guilbault, R., Tahan, ·.A., Sabourin, ·.R.: Bayesian network as an adaptive parameter setting approach for genetic algorithms. Complex & Intelligent Systems **2**, 1–22 (2016). https://doi.org/10.1007/s40747-016-0010-z
11. Deb, K., Agrawal, R.B., et al.: Simulated binary crossover for continuous search space. Complex systems **9**(2), 115–148 (1995)
12. Digalakis, J., Margaritis, K.: An experimental study of benchmarking functions for evolutionary algorithms. International Journal of Computer Mathemathics **79**, 403–416 (2002)
13. Eiben, A., Smit, S.: Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation **1**, 19–31 (03 2011). https://doi.org/10.1016/j.swevo.2011.02.001
14. Eiben, A.E., Smith, J.E., Michalewicz, Z., Schoenauer, M., Smith, J.E.: Parameter control in evolutionary algorithms. Studies in Computational Intelligence **54**(January), 19–46 (2007)
15. Gomes Pereira De Lacerda, M., Filipe De Araujo Pessoa, L., Buarque De Lima Neto, F., Ludermir, T.B., Kuchen, H.: A systematic literature review on general parameter control for evolutionary and swarm-based algorithms. Swarm and Evolutionary Computation **60**, 100777 (2021). https://doi.org/10.1016/j.swevo.2020.100777, www.elsevier.com/locate/swevo
16. Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of IEEE International Conference on Evolutionary Computation. pp. 312–317 (1996). https://doi.org/10.1109/ICEC.1996.542381
17. Karafotias, G., Hoogendoorn, M., Eiben, A.E.: Parameter control in evolutionary algorithms: Trends and challenges. IEEE Transactions on Evolutionary Computation **19**(2), 167–187 (2015). https://doi.org/10.1109/TEVC.2014.2308294
18. Maturana, J., Saubion, F.: On the design of adaptive control strategies for evolutionary algorithms. In: International Conference on Artificial Evolution (Evolution Artificielle). pp. 303–315. Springer (2007)

19. McGinley, B., Maher, J., O'Riordan, C., Morgan, F.: Maintaining healthy population diversity using adaptive crossover, mutation, and selection. IEEE Transactions on Evolutionary Computation **15**(5), 692–714 (2011)
20. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer (1996)
21. Morrison, R., De Jong, K.: Measurement of population diversity. vol. 2310, pp. 31–41 (10 2001)
22. Narendra, K.S., Parthasarathy, K.: Neural networks and dynamical systems. International Journal of Approximate Reasoning **6**(2), 109–131 (1992). https://doi.org/10.1016/0888-613X(92)90014-Q
23. Paternain, S., Morari, M., Ribeiro, A.: Real-time model predictive control based on prediction-correction algorithms. In: 2019 IEEE 58th Conference on Decision and Control (CDC). pp. 5285–5291. IEEE (2019). https://doi.org/10.1109/CDC40024.2019.9029408
24. Sammut, C., Webb, G.I. (eds.): Mean Squared Error, pp. 653–653. Springer US, Boston, MA (2010)
25. Schumer, M., Steiglitz, K.: Adaptive step size random search. Automatic Control, IEEE Transactions on **AC13**, 270–276 (07 1968). https://doi.org/10.1109/TAC.1968.1098903
26. Schwefel, H.P.: Numerical Optimization of Computer Models. John Wiley & Sons, Inc., USA (1981)
27. Shiblee, M., Kalra, P.K., Chandra, B.: Time series prediction with multilayer perceptron (MLP): A new generalized error based approach. In: "Köppen, M., Kasabov, N., Coghill, G. (eds.) Advances in Neuro-Information Processing. pp. 37–44. Springer Berlin Heidelberg (2009)
28. Smit, S.K., Eiben, A.E.: Comparing parameter tuning methods for evolutionary algorithms. In: 2009 IEEE congress on evolutionary computation. pp. 399–406. IEEE (2009)
29. Wang, J., Li, X., Li, J., Sun, Q., Wang, H.: NGCU: A new RNN model for time-series data prediction. Big Data Research **27**, 100296 (2022). https://doi.org/doi.org/10.1016/j.bdr.2021.100296
30. Werbos, P.J.: Consistency of HDP applied to a simple reinforcement learning problem. Neural Networks **3**(2), 179–189 (1990). https://doi.org/10.1016/0893-6080(90)90088-3
31. Zhu, K.Q., Liu, Z.: Population diversity in permutation-based genetic algorithm. In: European Conference on Machine Learning. pp. 537–547. Springer (2004)