Improving Mutation-Based Evolving Artificial Neural Network via Population Partitioning for Topology and Parameter Mutations

Kenta Okumura*, Motoaki Hiraga†, Daichi Morimoto‡, Kazuhiro Ohkura§, Nanako Miura¶, and Arata Masuda¶
*Graduate School of Science and Technology, Kyoto Institute of Technology, Kyoto, Japan
Email: m25623023@edu.kit.ac.jp
†Faculty of Mechanical Engineering, Kyoto Institute of Technology, Kyoto, Japan
Email: hiraga@kit.ac.jp
‡Department of Mechanical and Control Engineering, Kyushu Institute of Technology, Fukuoka, Japan
§Graduate School of Advanced Science and Engineering, Hiroshima University, Hiroshima, Japan
¶Faculty of Mechanical Engineering, Kyoto Institute of Technology, Kyoto, Japan

***Abstract*—Topology and Weight Evolving Artificial Neural Networks (TWEANNs) can simultaneously optimize neural network structures and parameters. Mutation-Based Evolving Artificial Neural Network (MBEANN) is a TWEANN algorithm that uses only mutations for genetic variation. In conventional MBEANN, the rapid growth of network topology can outpace parameter optimization, leading to bloated networks and an inefficient search of network topologies. This study proposes a population partitioning strategy that divides a population into two subgroups. One subgroup focuses on parameter optimization, whereas the other focuses on topological exploration, thereby balancing the two search processes. Experimental results on MuJoCo locomotion benchmarks demonstrate that the proposed approach achieves higher fitness values than the conventional MBEANN algorithm.**

I. Introduction

Neuroevolution is the study of the design and optimization of neural networks using evolutionary computation methods [1]–[3]. Traditional machine learning methods require explicit targets or gradients, whereas neuroevolution can be applied to problems in which these are unavailable or unreliable, such as reinforcement learning, robot control, and game playing. Moreover, the use of a population-based search of evolutionary computation enables a broader exploration of the solution space, which can mitigate the risk of trapping in local optima.

Neuroevolution can optimize both network parameters and architectures simultaneously; such algorithms are known as Topology and Weight Evolving Artificial Neural Networks (TWEANNs) [3], [4]. These approaches focus on the automated architecture search of neural networks, potentially discovering designs that a human might not consider. The evolution of neural network topologies introduces several challenges, such as performing crossover between networks with different topologies and avoiding the premature elimination of novel structures before their weights are sufficiently optimized to reveal their full capabilities. NeuroEvolution of Augmenting Topologies (NEAT) [4] is a representative method within the TWEANN algorithms that has addressed these problems by using historical markers to enable meaningful crossover between different topologies and speciation to protect newly introduced structures during evolution.

This study focuses on Mutation-Based Evolving Artificial Neural Network (MBEANN) [5], a TWEANN algorithm that uses only mutations for genetic variation, thereby avoiding the challenges associated with crossover operators. MBEANN is characterized by two main features. First, mutations that modify neural network topologies are designed to be neutral, allowing structural innovations to be introduced without disrupting existing functionality. Second, it adopts a genotype encoding composed of multiple subgenomes, referred to as operons, each corresponding to a subnetwork. Structural innovations occur independently within each operon, leading to modularized network structures. MBEANN performed better than NEAT on tasks such as the double-pole balancing problem [5], automatic controller design for robotic swarms [6]–[8], and MuJoCo locomotion benchmarks [9].

Despite its effectiveness, MBEANN still faces challenges in balancing structural exploration and parameter optimization. In the conventional MBEANN, topology mutations are applied independently within each operon, whereas add-node mutations can introduce new operons, causing the network structure to grow increasingly as evolution progresses. Moreover, topology mutations are designed to be behaviorally neutral, allowing newly introduced topologies to persist without immediate fitness degradation. As a result, structural growth may be accepted prematurely, leading to the gradual accumulation of redundant components and bloated network structures.

This paper proposes an extension of MBEANN that balances topology search and parameter optimization by partitioning the population into two subgroups with different mutation schemes. One subgroup applies only parameter mutation, focusing on refining existing network structures, whereas

The work was partly supported by the collaborative research program 2025 Information Initiative Center, Hokkaido University Sapporo, Japan.

the other applies both topology and parameter mutations to promote structural exploration. This population partitioning allows topological changes to be propagated when they lead to fitness improvement, while prioritizing parameter refinement when weight optimization is more effective, thereby enabling a balanced evolutionary search. Experimental results on the MuJoCo locomotion benchmarks provided in Gymnasium [10] demonstrate that this population partitioning leads to better performance compared to the conventional MBEANN algorithm.

The remainder of this paper is organized as follows. Section II provides an overview of MBEANN, and Section III introduces the proposed method. Section IV presents the experimental setups, and Section V reports the experimental results. Finally, Section VI concludes the paper.

MBEANN is implemented using the pyMBEANN library[1], which is publicly available on GitHub. The proposed method is a simple extension of this implementation and can be reproduced based on the released code.

II. Mutation-Based Evolving Artificial Neural Network

Mutation-Based Evolving Artificial Neural Network (MBEANN) [5] is a Topology and Weight Evolving Artificial Neural Network (TWEANN) algorithm that omits crossover and uses only mutations for genetic variation. MBEANN employs two types of topology mutations: add-node and add-link mutations, which are designed to minimize their impact on the behavior of the neural network. In addition, parameter mutations are applied to perturb weight and bias values. The remainder of this section describes the genetic encoding, mutation operators, and overall evolutionary process of MBEANN.

A. Genetic Encoding

MBEANN adopts a genotype encoding method that is composed of subgenomes called operons. Each operon consists of node and link genes, each of which contains associated genetic information. Each node gene has an identification number and node type that specify its belonging layer (e.g., input, hidden, or output). In addition to this genetic information, each hidden and output node is assigned a bias value. Each link gene includes an identification number, identification numbers of the nodes it connects, and a weight value. The genome representation can be described as follows:

$$\text{genome} = \{\text{operon}_0, \text{operon}_1, \cdots, \text{operon}_m\} \quad (1)$$

$$\text{operon}_i = \{\text{node}_j | j \in O_{Ni}\} \cup \{\text{link}_k | k \in O_{Li}\} \quad (2)$$

where m is the maximum index of operons in the genome, node_j is a node gene with identification number j, O_{Ni} is a set of node identification numbers in operon_i, link_k is a link gene with identification number k, and O_{Li} is a set of link identification numbers in operon_i.

[1]https://github.com/motoHiraga/pyMBEANN

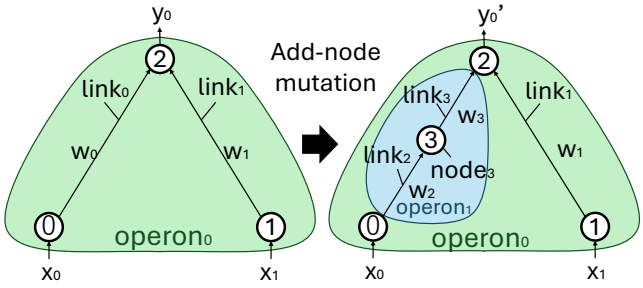


Fig. 1. Example of the add-node mutation

In the MBEANN algorithm, a genome is initialized only with operon_0, which consists of input nodes, output nodes, and links that connect them. New operons are generated when hidden nodes are added via add-node mutation. Connections between operon_i and operon_j ($i \neq j$, $i \neq 0$, $i \neq 0$) are prohibited to enable individuals to form modularized subnetworks.

B. Add-Node Mutation

Add-node mutation is applied to each operon with a probability of p_{node}. This mutation replaces a randomly selected link in the operon with a new node and two new links. If this mutation is applied to operon_0, a new operon is generated with a new node and links. An example of this mutation is shown in Fig. 1.

Topology mutations are designed to minimize the effect on the behavior of the neural network. In the example shown in Fig. 1, w is a weight value, x is an input value, and y is an output value, and the add-node mutation is applied so that $y_0 \simeq y_0'$. When the sigmoid activation function $\varphi(u) = 1/(1+e^{\alpha(\beta-u)})$ is applied to the hidden and output nodes, y_0 and y_0' are calculated as shown in the following equations:

$$y_0 = \varphi(w_0 x_0 + w_1 x_1), \quad (3)$$

$$y_0' = \varphi(w_3 \varphi(w_2 x_0) + w_1 x_1). \quad (4)$$

When $w_0 = w_3$, the condition $y_0 \simeq y_0'$ becomes $x_0 \simeq \varphi(w_2 x_0)$. Therefore, the error $L(x_0)$ can be expressed as follows:

$$L(x_0) = |x_0 - \varphi(w_2 x_0)| = \left| x_0 - \frac{1}{1 - e^{\alpha(\beta - w_2 x_0)}} \right|. \quad (5)$$

This error L is minimized when $\alpha = 4.63/w_2$ and $\beta = 0.5 w_2$ ($w2 \neq 0$) [9]. For simplicity, $w_2 = 1$ ($\alpha = 4.63$ and $\beta = 0.5$) is often used in MBEANN.

C. Add-Link Mutation

Add-link mutation is applied to each operon with a probability of p_{link}. This mutation generates a new link that connects two nodes from operon_i or one node each from operon_i and operon_0. The weight value of the new link is set to 0 to minimize the effect on the neural network behavior. An example of this mutation is shown in Fig. 2.

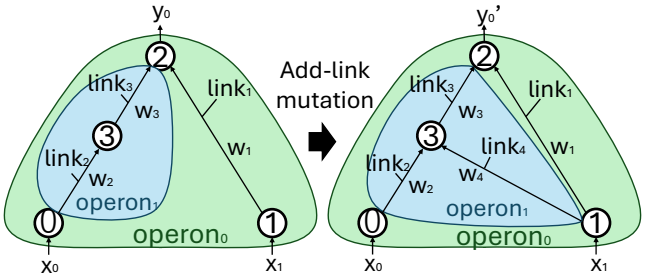


Fig. 2. Example of the add-link mutation

Algorithm 1 Conventional MBEANN

1: Initialize population with λ individuals
2: **while** termination condition not satisfied **do**
3: Evaluate each individual using fitness function
4: Select λ individuals using tournament selection based on their fitness
5: Apply mutations to each selected individual
6: **end while**

D. Parameter Mutation

Parameter mutation is applied to each weight or bias value with a probability of p_{param}. When this mutation is applied to a link gene link_i with a weight value w_i, its weight is mutated to w_i' using the following equation:

$$w_i' = w_i + \varsigma N_i(0,1) \tag{6}$$

where ς is the step size that determines the magnitude of the mutation and $N_i(0,1)$ is a value independently drawn from a standard normal distribution for each parameter i. Bias values are mutated using the same equation.

E. Evolutionary Algorithm of MBEANN

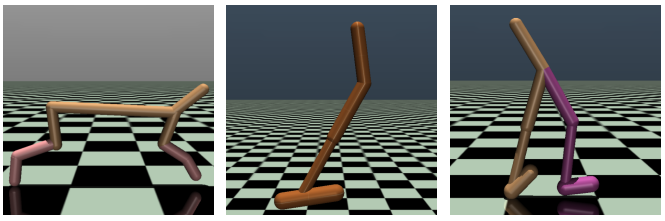
The evolutionary process of MBEANN proceeds as follows. First, a population of λ individuals is initialized, where each individual starts with a minimal topology consisting of only operon_0. Next, each individual is evaluated using a fitness function that reflects its performance on the target task. Parent individuals are then selected using tournament selection. In tournament selection, a fixed number of individuals, called the tournament size, are randomly sampled from the population, and the individual with the highest fitness among them is selected as a parent. This selection process is repeated until λ parents are chosen. Each parent is mutated using add-node, add-link, and parameter mutations to produce an offspring individual. These λ offspring are carried over to the next generation. This evolutionary cycle is repeated until the termination condition is satisfied. The overall evolutionary process of MBEANN is summarized in Algorithm 1.

III. Proposed Method: MBEANN with Population Partitioning for Topology and Parameter Mutations

This section describes the details of the method proposed in this study. In the conventional MBEANN algorithm, topology mutations occur independently within each operon and are designed to be neutral, which tends to cause the gradual accumulation of redundant components and bloated network structures. In the proposed method, instead of applying topology and parameter mutations to all individuals within the population, the population is divided into two subgroups: one subgroup applies both topology and parameter mutations, and the other applies only parameter mutation. The selection mechanism of the evolutionary algorithm operates on the union of these two subgroups. This population partitioning and selection method enables the balancing of topology search and parameter optimization. In particular, if a topological change leads to fitness improvement, that change will become dominant within the population. By contrast, if parameter optimization is more beneficial than topology search, individuals with only parameter mutation are more likely to survive to the next generation. The overall evolutionary process of the proposed method is summarized in Algorithm 2.

Algorithm 2 MBEANN with population partitioning

1: Initialize population with λ individuals
2: **while** termination condition not satisfied **do**
3: Evaluate each individual using fitness function
4: Select λ_T and λ_P individuals from λ individuals using tournament selection based on their fitness
5: Apply topology and parameter mutations to each individual in λ_T
6: Apply parameter mutation to each individual in λ_P
7: Generate new population λ by combining λ_T and λ_P
8: **end while**



(a) HalfCheetah-v5 (b) Hopper-v5 (c) Walker2d-v5

Fig. 3. MuJoCo tasks in Gymnasium

IV. Experimental Setups

In this study, experiments are conducted on the HalfCheetah-v5, Hopper-v5, and Walker2d-v5 tasks provided in the Gymnasium reinforcement learning environment [10]. Screenshots of the robots in these tasks are shown in Fig. 3. The objective of these tasks is to make the robot move forward effectively. The observation space consists of the vertical coordinate of the body, along with the velocities, angles, and angular velocities of various parts of the robot. The action space consists of the torques applied to the hinge joints. The observation and action spaces have 17 and 6 values in HalfCheetah-v5, 11 and 3 values in Hopper-v5, and 17 and 6 values in Walker2d-v5, respectively. The observation space

values are provided as inputs to the neural network, and the network outputs are mapped to the action space values. The default reward settings and episode termination conditions are used in the experiments. The cumulative reward obtained per episode is used as the fitness value.

A. Input Normalization

The OpenAI-ES algorithm [11] uses virtual batch normalization, a technique similar to batch normalization, in which a reference batch used to calculate normalization statistics is chosen at the start of training and is fixed [11], [12]. In environments such as MuJoCo locomotion tasks, some sensor inputs (e.g., joint velocities) may vary widely during the evolution process. In such tasks, normalizing the inputs of the neural network using virtual batch normalization has been demonstrated to be an effective technique in neuroevolution methods [13]. Input normalization improves performance by stabilizing the distribution of network inputs and reducing sensitivity to differences in scale across sensors, thereby facilitating more effective optimization during evolution. In this study, input normalization is implemented following the code provided in the OpenAI-ES GitHub repository[2].

B. Algorithm Settings

In this study, the conventional MBEANN [5] and MBEANN with input normalization, referred to as IN-MBEANN, are used as the baseline methods. The proposed method that uses population partitioning for topology and parameter mutations is referred to as SPLIT-MBEANN because it *splits* the population into two subpopulations. Note that SPLIT-MBEANN also uses input normalization.

The hyperparameters are set based on preliminary experiments and prior work [9]. The evolutionary process terminates when reaching the maximum number of generations $G_{\max}$. HalfCheetah-v5 is a relatively easy task compared to Hopper-v5 and Walker2d-v5. Therefore, the population size λ, the maximum number of generations $G_{\max}$, and the mutation step size ς are set to $\lambda = 200$, $G_{\max} = 200$, and $\varsigma = 0.01$ for HalfCheetah-v5, respectively. For Hopper-v5 and Walker2d-v5, these parameters are set to $\lambda = 500$, $G_{\max} = 500$, and $\varsigma = 0.005$. Across all benchmark tasks and algorithms, the add-node mutation probability is set to $p_{\text{node}} = 0.03$, the add-link mutation probability to $p_{\text{link}} = 0.3$, the parameter mutation probability to $p_{\text{param}} = 1.0$, and the tournament size for selection to $\lambda/10$.

In the proposed method, the population is divided into two subpopulations with sizes of λ_T and λ_P. For simplicity, the subpopulation sizes are set such that $\lambda_T : \lambda_P = 1 : 1$. That is, $\lambda_T = \lambda_P = 100$ for HalfCheetah-v5 and $\lambda_T = \lambda_P = 250$ for Hopper-v5 and Walker2d-v5. The proposed method applies topology mutations only to the subpopulation λ_T, which is half the size of λ. This decreases the number of individuals that undergo topology mutations. Therefore, an additional variant of the proposed method is implemented by doubling the mutation probabilities to $p_{\text{node}} = 0.06$ and $p_{\text{link}} = 0.6$, which is referred to as SPLIT-MBEANN-2.

[2]https://github.com/openai/evolution-strategies-starter

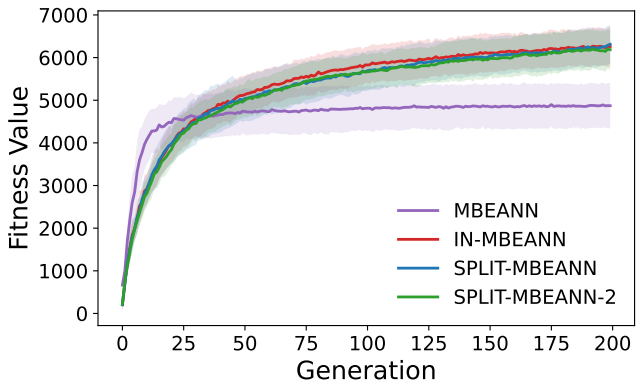
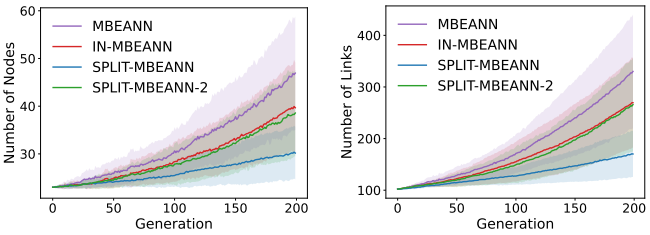


Fig. 4. Fitness transition of the best individual in the population on HalfCheetah-v5, where each line represents the mean of the fitness values over 30 trials and the shaded regions indicate the standard deviations



(a) Number of nodes (b) Number of links

Fig. 5. Transition of the topology size of the best individual in the population on HalfCheetah-v5, where each line represents the mean topology size over 30 trials and the shaded regions indicate the standard deviations

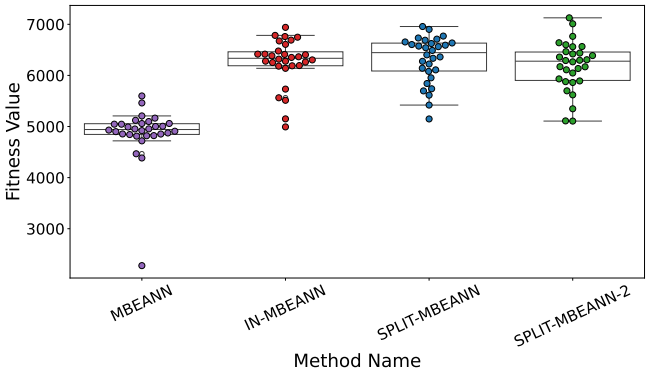


Fig. 6. Highest fitness achieved in the final generation of each evolutionary trial on HalfCheetah-v5

V. Results

Each algorithm is executed on each task over 30 independent evolutionary trials. The results on HalfCheetah-v5 are shown in Figs. 4–6, those on Hopper-v5 in Figs. 7–9, and those on Walker2d-v5 in Figs. 10–12. The fitness transition of the highest fitness value achieved in the population at each generation is shown in Figs. 4, 7, and 10. The transition of the

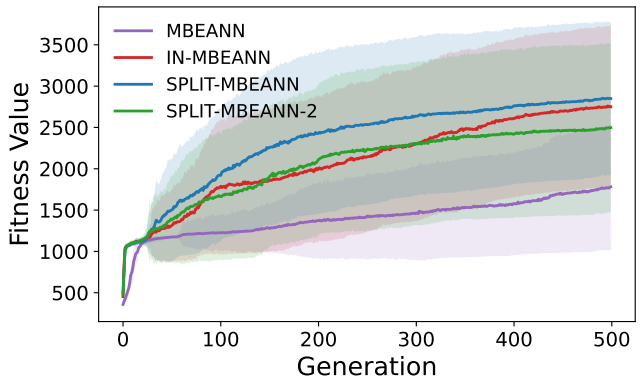


Fig. 7. Fitness transition of the best individual in the population on Hopper-v5, where each line represents the mean of the fitness values over 30 trials and the shaded regions indicate the standard deviations

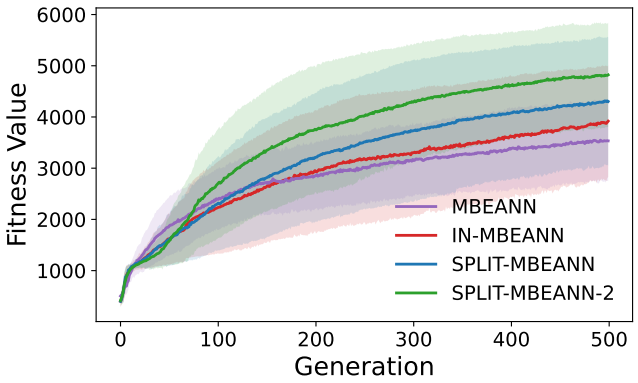
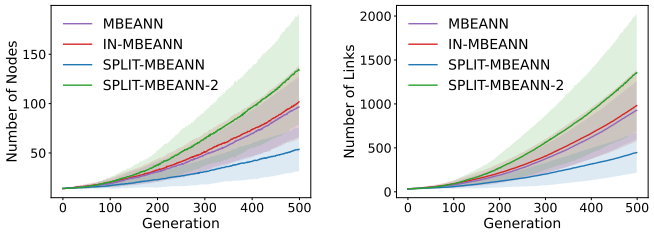
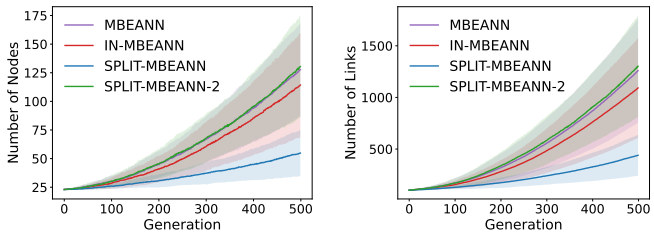


Fig. 10. Fitness transition of the best individual in the population on Walker2d-v5, where each line represents the mean of the fitness values over 30 trials and the shaded regions indicate the standard deviations



(a) Number of nodes (b) Number of links

Fig. 8. Transition of the topology size of the best individual in the population on Hopper-v5, where each line represents the mean topology size over 30 trials and the shaded regions indicate the standard deviations



(a) Number of nodes (b) Number of links

Fig. 11. Transition of the topology size of the best individual in the population on Walker2d-v5, where each line represents the mean topology size over 30 trials and the shaded regions indicate the standard deviations

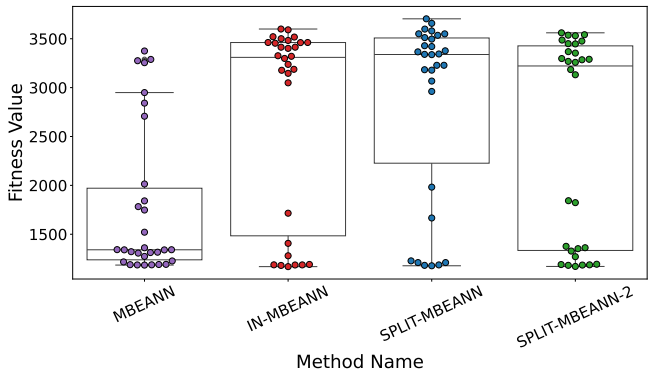


Fig. 9. Highest fitness achieved in the final generation of each evolutionary trial on Hopper-v5

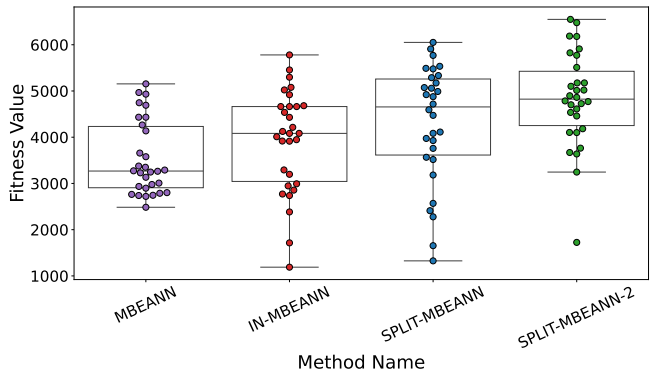


Fig. 12. Highest fitness achieved in the final generation of each evolutionary trial on Walker2d-v5

topology size of the corresponding best individuals is shown in Figs. 5, 8, and 11. In addition, Figs. 6, 9, and 12 show the highest fitness achieved in the final generation of each evolutionary trial. The two-sided Mann–Whitney U test with Bonferroni correction is performed at a significance level of 0.05 to compare the performance of the algorithms.

In HalfCheetah-v5, IN-MBEANN, SPLIT-MBEANN, and SPLIT-MBEANN-2, which applied input normalization, achieved higher fitness values than MBEANN, as shown in Fig. 4. SPLIT-MBEANN obtained the smallest topology among all algorithms, while MBEANN obtained the largest topology on HalfCheetah-v5, as shown in Fig. 5. In Fig. 6, significant differences were observed between MBEANN and the other algorithms, while no significant differences were observed between IN-MBEANN, SPLIT-MBEANN, and SPLIT-MBEANN-2.

In Hopper-v5, the highest fitness was achieved by SPLIT-MBEANN, as shown in Fig. 7. Again, MBEANN achieved

the lowest fitness among all algorithms. SPLIT-MBEANN-2 showed a more rapid growth in neural network topology, whereas SPLIT-MBEANN resulted in smaller topologies, as shown in Fig. 8. In Fig. 9, significant differences were observed between MBEANN and IN-MBEANN and between MBEANN and SPLIT-MBEANN, whereas no significant difference was observed between MBEANN and SPLIT-MBEANN-2. No significant differences were observed between IN-MBEANN, SPLIT-MBEANN, and SPLIT-MBEANN-2. Although SPLIT-MBEANN-2 tended to achieve slightly lower fitness than IN-MBEANN and SPLIT-MBEANN, the difference was not statistically significant.

In Walker2d-v5, SPLIT-MBEANN-2 achieved the highest fitness, followed by SPLIT-MBEANN, IN-MBEANN, and MBEANN, as shown in Fig. 10. SPLIT-MBEANN obtained the smallest topology among all algorithms, whereas MBEANN and SPLIT-MBEANN-2 evolved the largest topologies, as shown in Fig. 11. In Fig. 12, no significant differences were observed between MBEANN and IN-MBEANN, between IN-MBEANN and SPLIT-MBEANN, or between SPLIT-MBEANN and SPLIT-MBEANN-2, whereas significant differences were found between the remaining pairs.

Overall, the experimental results across all benchmark tasks indicate that the proposed methods SPLIT-MBEANN and SPLIT-MBEANN-2 generally outperform MBEANN and IN-MBEANN. SPLIT-MBEANN obtained the smallest topology among all algorithms, which is expected because only half of the population undergoes topology mutations. Interestingly, despite the smaller network topologies, SPLIT-MBEANN achieved high fitness values. Moreover, as no significant difference in fitness was observed between SPLIT-MBEANN and IN-MBEANN, population partitioning did not appear to hinder performance.

In SPLIT-MBEANN-2, the topological mutation probabilities are doubled to increase the number of individuals that undergo topology mutations. This resulted in evolving neural networks with topology sizes comparable to those of IN-MBEANN or MBEANN in HalfCheetah-v5 and Walker2d-v5. Moreover, SPLIT-MBEANN-2 achieved the highest fitness among all algorithms in Walker2d-v5. However, in Hopper-v5, SPLIT-MBEANN-2 obtained slightly lower fitness values than IN-MBEANN and SPLIT-MBEANN. In Hopper-v5, SPLIT-MBEANN-2 showed rapid growth in the neural network topology, which may cause parameter optimization to fall behind topological expansion, leading to performance degradation. These results indicate that increasing the topological mutation probabilities in a method with population partitioning may have a negative impact on performance. Further tuning of the mutation probabilities is required to improve the proposed method, which is left for future work.

In addition, MBEANN achieved the lowest fitness among all algorithms across all benchmark tasks. This indicates that input normalization is a beneficial technique to improve the performance of MBEANN.

VI. Conclusion

This study extended MBEANN by partitioning the population into two subgroups, in which one subgroup applies both topology and parameter mutations and the other applies only parameter mutation. In addition, the normalization technique used in OpenAI-ES was incorporated into MBEANN to further improve performance. The experimental results on the MuJoCo locomotion benchmarks provided in Gymnasium revealed that the proposed methods effectively improved the MBEANN algorithm.

The proposed methods using population partitioning outperformed both the conventional MBEANN and MBEANN with input normalization. However, the growth of topology and its impact on fitness varied across tasks. Future work will focus on more detailed investigations of the proposed method, including the validation on tasks that require larger topologies and analysis of how the ratio of subpopulations affects performance, with the goal of developing a more effective strategy for topology search.

References

[1] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[2] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[3] S. Risi, Y. Tang, D. Ha, and R. Miikkulainen, *Neuroevolution: Harnessing Creativity in AI Agent Design*. Cambridge, MA: MIT Press, 2025.

[4] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[5] K. Ohkura, T. Yasuda, Y. Kawamatsu, Y. Matsumura, and K. Ueda, "MBEANN: Mutation-based evolving artificial neural networks," in *Proceedings of the 9th European Conference on Artificial Life*. Springer, 2007, pp. 936–945.

[6] K. Ohkura, T. Yasuda, and Y. Matsumura, "Coordinating the adaptive behavior for swarm robotic systems by using topology and weight evolving artificial neural networks," in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.

[7] M. Hiraga and K. Ohkura, "Topology and weight evolving artificial neural networks in cooperative transport by a robotic swarm," *Artificial Life and Robotics*, vol. 27, no. 2, pp. 324–332, 2022.

[8] Y. Katada, T. Hirokawa, M. Hiraga, and K. Ohkura, "MBEANN for robotic swarm controller design and the behavior analysis for cooperative transport," *Journal of Robotics and Mechatronics*, vol. 35, no. 4, pp. 997–1006, 2023.

[9] M. Hiraga, M. Komura, A. Miyamoto, D. Morimoto, and K. Ohkura, "Improving the performance of mutation-based evolving artificial neural networks with self-adaptive mutations," *PLOS One*, vol. 19, no. 7, 2024, Art. no. e0307084.

[10] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2025.

[11] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[12] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[13] P. Pagliuca, N. Milano, and S. Nolfi, "Efficacy of modern neuro-evolutionary strategies for continuous control optimization," *Frontiers in Robotics and AI*, vol. 7, p. 98, 2020.