Rapidly Evolving Soft Robots via Action Inheritance

Shulei Liu, Wen Yao, Handing Wang, *Member, IEEE,* Wei Peng, and Yang Yang

***Abstract*—The automatic design of soft robots characterizes as jointly optimizing structure and control. As reinforcement learning is gradually used to optimize control, the time-consuming controller training makes soft robots design an expensive optimization problem. Although surrogate-assisted evolutionary algorithms have made a remarkable achievement in dealing with expensive optimization problems, they typically suffer from challenges in constructing accurate surrogate models due to the complex mapping among structure, control, and task performance. Therefore, we propose an action inheritance-based evolutionary algorithm to accelerate the design process. Instead of training a controller, the proposed algorithm uses inherited actions to control a candidate design to complete a task and obtain its approximated performance. Inherited actions are near-optimal control policies that are partially or entirely inherited from optimized control actions of a real evaluated robot design. The action inheritance plays the role of surrogate models where its input is the structure and output is the near-optimal control actions. We also propose a random perturbation operation to estimate the error introduced by inherited control actions. The effectiveness of our proposed method is validated by evaluating it on a wide range of tasks, including locomotion and manipulation. Experimental results show that our algorithm is better than the other three state-of-the-art algorithms on most tasks when only a limited computational budget is available. Compared with the algorithm without surrogate models, our algorithm saves about half the computing cost.**

***Index Terms*—Soft robot design, surrogate-assisted evolutionary algorithm, morphological optimization, error estimation**

I. Introduction

In the last decade, the research on creating autonomous robots has observed important developments [1], [2]. Inspired from natural living systems, where body structure and brain are two key factors for completing any task in a real environment, an intelligent soft robot typically requires optimizing its structure and control simultaneously [3], [4]. Such a co-design problem has been challenging in robotics and machine learning communities [5].

The co-design of soft robots is a bi-level optimization problem [6], [7] in which the outer loop optimizes morphological structures while the inner loop optimizes the control for a given structure. In the outer loop, the encoding strategy of structures directly affects the expression of the morphological design space. A reasonable encoding strategy can result in an effective design space and maximize the retention of physical morphology [4]. There are two types of encoding strategies: indirect and direct encoding. Compositional pattern producing network (CPNN) [8] is the most classical indirect encoding scheme in which a neural network is used to learn the mapping relation between the spatial coordinate of a robot voxel and the type of that voxel. Through a series of perturbation operations, including adding or deleting nodes and edges, exchanging weights and activation functions, CPNN outputs a variety of structures [9]. Compared with indirect encoding, direct encoding takes a more intuitive scheme. A morphological structure is always represented as a tree [4], [10], grid [11], [12], or graph [13]. For complex robot design tasks, either the direct or the indirect encoding strategy will lead to a combinatorially huge search space. This poses a great challenge to the outer optimization methods [14].

A choice for outer optimization is evolutionary algorithms (EAs), due to their searching ability in the complex design space [15], [16]. Based on the principle of selective reproduction of the fittest, robots are viewed as autonomous artificial organisms that can develop their own structures by interacting with the environment. The fittest robots survive and reproduce until a robot that satisfies the performance criteria is produced. Considering the discrete nature of the design space, several improved EAs have been proposed, such as biodiversity enhanced mechanism [17], constrained evolution [18], neural graph evolution [19], etc.

In the optimization, the EA should allocate each newly generated robot design a unique controller. Then, the fitness evaluation is performed by using the actions provided by the controller to control a robot to fulfill a specific task. The inner control optimization directly affects the evolution direction. Many methods have been proposed to obtain an intelligent controller, broadly divided into two categories: model- and learning-based approaches [20]. Model-based approaches first find a hand-engineered mathematical model of the robot dynamics, such as a physical [21], [22] or fuzzy approximation [23] model. Then, model parameters are adjusted by repeatedly executing control actions and obtaining feedback. After that, a robot is typically derived using the re-parameterized model [24], [25]. One drawback of such methods is assuming a basic topology in advance, such as legged robots [26]. However, in some complex tasks, the prior knowledge about the basic topology is limited.

Recently, reinforcement learning (RL) has proven effective

This work was supported in part by the National Natural Science Foundation of China (No. 61976165,62376202).(*Corresponding author: Wen Yao, Handing Wang*)

S. Liu and H. Wang are with School of Artificial Intelligence, Xidian University, Xi'an 710071, China. (e-mail: shuleiliu@126.com, hdwang@xidian.edu.cn). S. Liu is also with the Defense Innovation Institute, Chinese Academy of Military Science, Beijing 100071, China. H. Wang is also with Collaborative Innovation Center of Quantum Information of Shaanxi Province, Xidian University, Xi'an 710071, China.

W. Yao, W. Peng, and Y. Yang are with the Defense Innovation Institute, Chinese Academy of Military Science, Beijing 100071, China (e-mail: wendy0782@126.com, weipeng0098@126.com, bigyangy@gmail.com).

at discovering control policies [4], [11], [27]. In learning-based methods, the RL paradigm is used to learn the mapping relation between environment observations and executed actions. Compared with model-based methods, learning-based methods are able to learn intelligent behaviors without prior knowledge. However, the process of RL often requires a large number of trajectory data, leading to a computationally expensive training procedure. The inner control optimization leads to a time-consuming whole optimization process. For instance, a control policy is learned after 5 million iterations in [4]. On the premise of using the parallel technology, it takes about 16 hours to train 288 controllers. The inner control optimization leads to a long iteration cycle of the whole optimization process.

With the extensive application of RL techniques, especially deep RL, the significant computational burden imposed by training controllers has become a major factor affecting the rapid development of robot automatic design [4], [19]. There are generally two strategies to accelerate the evolution of soft robots. One is to reduce the time spent on optimizing control. The simplest method of the first strategy is to use other control optimization methods, such as CPPN [28] and evolution strategies [29], instead of RL algorithms. Although these methods can reduce the time spent on optimizing control, this is a small contribution compared to the costed total time. Since the prolonged training process is the main reason for the high computational cost, many acceleration strategies have been proposed, such as model order reduction technique [30], group training [18], model-based reinforcement learning [13], and policy transfer [31]. However, they are only applicable to simple tasks or the implementation of these strategies requires a series of complex operations.

The other is to reduce the number of times to train controllers. The purpose of training controllers is to obtain the task performance of candidate robots. In the optimization process of EAs, there are always many eliminated robots. Once a robot is eliminated, it will not appear in the subsequent population and will not impact the optimization process. If these eliminated robots can be pre-selected without training controllers, then the number of times to train controllers can be reduced, thereby accelerating the evolution. Surrogate-assisted evolutionary algorithms (SAEAs) are a technique that accelerates the optimization process by reducing the number of computationally expensive evaluations [32], [33]. One general target of SAEAs is using a computationally cheap surrogate model to replace the computationally expensive fitness evaluations. The key in SAEAs is how to build accurate surrogate models [33]. Many excellent machine learning methods have been utilized, including random forest [34], radial basis function network [35], Kriging model [36], etc.

Although SAEAs have achieved good performance in dealing with many practical problems [37], [38], there are several challenges to using SAEAs to solve robot design problems. First, in most SAEAs, some generic surrogate models are directly trained by taking decision variables and objectives as input and output, respectively [39]. However, the robot design problem is a co-design or bi-level optimization problem with the objective related to outer decision variables and inner optimized control [40], [41]. For dealing with expensive bi-level problems, surrogate models can be useful to approximate the outer objective function. One approach is to train models using decision variables and the outer objective as input and output without considering inner control. This method is particularly effective when the inner optimization is straightforward. [40]. The automatic design of soft robots requires the inner controller to drive the robot to generate intelligent behavior, so its optimization is extremely complex. Second, although the surrogate model technology can approximate the inner control optimization in theory, there are still some practical difficulties. The optimized control is a series of intelligent actions generated by a well-trained controller after the robot interacts with the environment rather than a simple numerical vector. Moreover, the number of actions is relatively large. For example, in [11], controlling a robot to complete some complex tasks requires at least 1000 actions. Without a large amount of training data, it is not easier to train a generic model to approximate the mapping from a low-dimensional robot structure to its high-dimensional control actions. Generally, the key to using SAEAs to solve robot design problems is how to build an accurate surrogate model.

As soft robot design problems have the characteristics of bi-level optimization and complex control, it is challenging to build accurate generic surrogate models. Fortunately, there are still some well-trained controllers in the optimization process. For a real-evaluated robot, its well-trained controller can be abstractly described as the knowledge of how to control the robot to complete a particular task. Therefore, besides the evaluated data (structures and rewards), we can use this knowledge to build surrogate models. In soft robot design problems, different controllers inputs and outputs may differ. Without retraining, the transferred controller (a neural network) cannot directly act on a new robot design. The simplest way to utilize this knowledge is to use the actions provided by a well-trained controller to control a candidate robot and obtain its approximated task performance. Since the actions employed to obtain approximated performance are partially or fully inherited from these control actions of a real-evaluated robot, we refer to the process of utilizing a well-trained controller for evaluating candidate robot designs as action inheritance.

In this work, we take voxel-based soft robots as the research object and present an action inheritance-based evolutionary algorithm (AIEA) to accelerate the automatic soft robot design. The main contributions of the paper are summarized as follows.

1) For a candidate robot design, a rapid evaluation method is proposed to obtain its approximated task performance. Based on the structural similarity, a candidate robot first finds its parent among all the real-evaluated robots. Considering some differences between voxel positions and types in these two robots' structures, some easy-to-implement adjustments are made to the parent's control actions, such as extracting and filling, to obtain inherited actions. The approximated task performance is obtained using inherited actions to control the candidate robot. The action inheritance-based evaluation plays the role

of surrogate models in the proposed algorithm.

2) Since these inherited actions are near-optimal, there is an error between the task performance under the near-optimal control and its actual performance. A random perturbation operation is proposed to estimate the error introduced by action inheritance.
3) A local search method is integrated into the proposed algorithm to accelerate the evolution further. Combining surrogate model approximation and local search allows the proposed algorithm to search for more potential designs without leading to additional computational costs.

The rest of this paper is organized as follows. Section II briefly introduces the study object of this paper: voxel-based soft robot. Section III introduces the details of the proposed algorithm, and the experimental results of our algorithm with the other algorithms are presented in Section IV. Finally, Section V concludes the paper with a summary and a discussion of future work.

II. Voxel-based Soft Robot

Voxel-based soft robots (VSRs) are a kind of robots composed of several elastic cubic blocks (called voxels) [42]. Through the coordinated deformation among various voxels, the robot can emerge intelligent behaviors. Following [11], we consider a unified multi-material VSR in this work. When a VSR attempts to fulfill a task, such as locomotion and manipulation, it usually involves three main factors: structure, simulation, and action.

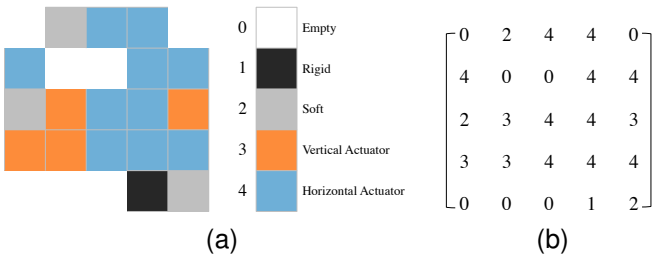

Fig. 1. Taking a VSR with (5×5) grid size as an example. (a) is the voxel representation. (b) is the material matrix.

The structure of a VSR describes how the voxels are arranged in a grid topology. A direct encoding strategy is applied. Each robot is specified as a material matrix of voxels. The entries of the material matrix are integers corresponding to a voxel type. Following [11], four types of voxels are considered: soft, rigid, vertical actuator, and horizontal actuator voxels, which are encoded as 1, 2, 3, and 4, respectively. Besides, we also allow the presence of empty voxels in VSRs. Correspondingly, 0 represents an empty voxel. Taking the (5×5) grid size as an example, Fig. 1 shows the voxel representation and material matrix a randomly generated VSR.

Simulation, also known as the modeling method of robot dynamics, is a fundamental part in VSR design problems [2]. The mass-spring system is a simple and flexible method for modeling VSRs in which robots and task environment are converted into a set of point masses and springs by turning each voxel into a cross-braced square [43].

At each time step, an action vector provided by the robot's controller is used to step the simulation. Each component of the action vector is associated with an actuator voxel (either horizontal or vertical) and instructs a deformation of that voxel. In other words, the length of an action vector is the same as the number of actuator voxels. Different robots may have different action lengths. Each component in an action vector is a real number that represents the ratio of the deformed length of the spring to its original length.

The goal of VSR autonomous design problems is to find the optimal structure $\boldsymbol{s^*}$ and control $\boldsymbol{a}^*_{\boldsymbol{s}^*}$:

$$\begin{aligned} &\boldsymbol{s}^*, \boldsymbol{a}^*_{\boldsymbol{s}^*} = \arg\max_{\boldsymbol{s} \in S} F(\boldsymbol{s}, \boldsymbol{a}^*_{\boldsymbol{s}}), \\ &s.t. \quad \boldsymbol{a}^*_{\boldsymbol{s}} \in \arg\max_{\boldsymbol{a}_{\boldsymbol{s}} \in A} f(\boldsymbol{s}, \boldsymbol{a}_{\boldsymbol{s}}), \end{aligned} \tag{1}$$

where $\boldsymbol{s}$ is the structure matrix of a robot design, S is the design space. $\boldsymbol{a}_{\boldsymbol{s}} \in \mathbb{R}^{t \times l}$ is the set of control actions where t is the number of simulation steps and l is the action length. $F(\cdot)$ represents the objective function of the outer loop where its evaluations require the optimized control of the inner loop. $f(\cdot)$ represents the objective function of the inner loop which is usually expressed as the learning process of intelligent controllers by RL algorithms, such as proximal policy optimization (PPO) [44]. More specifically, the RL algorithm needs to learn the mapping between the environmental observations and the suitable actions by constantly generating actions, executing actions, and obtaining simulation feedback. The learning process of RL requires tremendous simulation data, suffering from a heavy computational burden. Although RL can train an intelligent controller, its convergence cannot be always guaranteed. To be precise, the actions outputted by a well-trained controller is less-than-optimal. In this article, actions outputted by well-trained controllers are expressed as optimized actions.

In general, the decision variable $\boldsymbol{s}$ is identified as an $n \times m$ matrix where $s_{ij} \in \{0, 1, ..., v\} (1 \leq i \leq n, 1 \leq j \leq m)$. 0 represents an empty voxel and v is the type of voxels. In this work, $v = 4$. When the fitness of a candidate design needs to be evaluated during evolution, the optimized or trained control $\boldsymbol{a}^*_{\boldsymbol{s}}$ should be provided. That is to say, a unique controller must be trained before obtaining the performance of a robot design. Based on the above analysis, VSR design problems have two characteristics: the decision variable is discrete and the fitness evaluation is time-consuming. Therefore, the VSR design problem is a typical expensive combinatorial optimization problem.

III. Proposed Algorithm

SAEAs generally contain two types of essential operations: search strategies-related and surrogate models-related. In this section, we first introduce the basic flow of AIEA and then detail the search strategy used in AIEA, followed by the action inheritance-based surrogate model and an error estimation method.

A. Basic Flow

Fig. 2 illustrates the framework of AIEA, which follows the basic process of SAEAs, including initialization, real evalua-

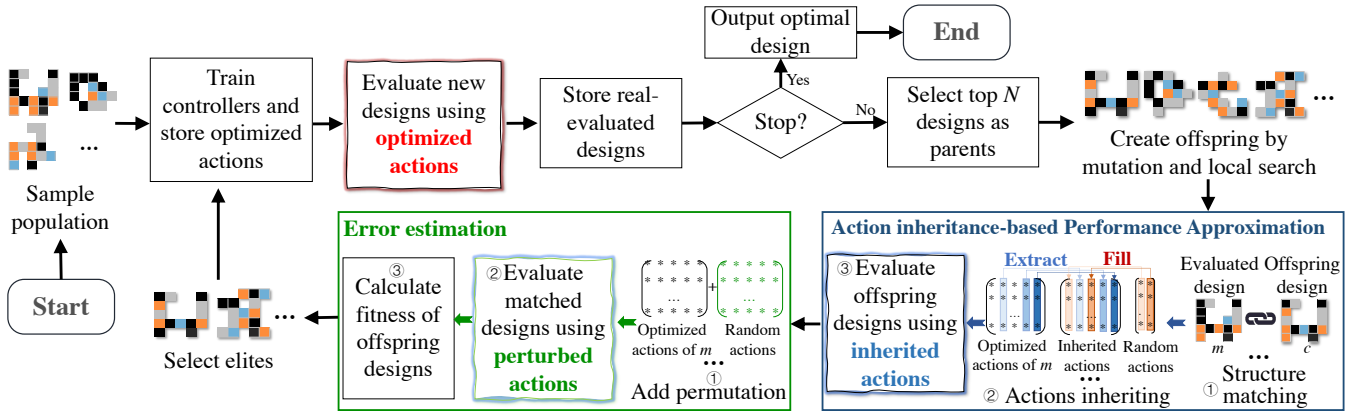

Fig. 2. Framework of the proposed AIEA. For a candidate robot design, action inheritance is used to rapidly evaluate its approximated performance by matching it with a real-evaluated design based on structural similarity, inheriting optimized control actions of the matched design based on the position and type of actuator voxels, and applying inherited actions to control the robot. Error estimation is used to make the rapidly-evaluated performance more accurate by adding random perturbation to optimized control actions of the matched design, evaluating the matched design using perturbed actions, and re-calculating fitness.

tion, variation, surrogate-based evaluation, and environmental selection. The first three parts are shown on the first layer of Fig. 2, the last two are on the second layer.

For a given task, a number of initial robot designs are first generated in the design space by random sampling. Before sampling a robot design, it's essential to determine its maximal size, such as the $(n \times m)$ grid. Once the robot size is specified, each voxel will be randomly sampled from all materials. This process results in a unique design every time. After that, the sampled design will be judged if it has actuators or is connected. Otherwise, a new design is resampled. After sampling, their performance are evaluated by the expensive fitness function. More specifically, the PPO algorithm is used to learn an optimized controller for each robot. Following that, the simulation is executed to obtain the task reward of each robot with the participating of its well-trained controller. Specifically, at each time step, the well-trained controller outputs an action vector according to the environment observation. The robot executes the action and the simulation returns a task-specific reward. The sum of rewards at all steps is applied as the fitness to measure the performance of the current robot and its control. After that, all the real-evaluated designs with their optimized control actions are stored in archives.

In each generation of AIEA, top N designs are selected from these stored designs to form a parent population. Next, a mutation operator and a local search method are performed to create offsprings (Section III-B). For each offspring, an action inheritance-based method is used to obtain its approximated performance (Section III-C). To make the approximated performance accurate, an error estimation method is employed (Section III-D). After that, the environmental selection is executed by taking the sum of the approximated performance and the error as the selection metric. The survivors are then re-evaluated using the real fitness function. All the real-evaluated designs are stored in archives. The above process repeats until the termination condition is satisfied.

B. Search Algorithm

Following [11], the genetic algorithm (GA) with a simple mutation operator is implemented to evolve the population of robot designs. For combinatorial optimization problems, a combination of local search and population-based search has been widely used to improve the convergence speed [45], [46]. Considering the discrete design spaces and arbitrarily complex tasks, we integrate a local search method into the GA to search promising designs.

The pseudo-code of AIEA is shown in Algorithm 1. In each generation of AIEA, N robot designs are firstly created by iteratively selecting one of N parents and mutating its structure (lines 11-15). N represents the population size. These progenies are generated based on multiple high-quality designs, which play a role in exploring the design space. Then, other N robot designs are created by taking the best design found so far as the only parent and iteratively mutating its structure (lines 16-20). By limiting the parent, these final N designs pay more attention to a local region and favor exploitation. In the above two processes, only the selection of parents is different. The approach of mutating parents structure is the same. Following [11], a simple mutation strategy is used to evolve the population of robot designs. The pseudocode of mutation is shown in Section S.I of the supplementary material. To avoid repeatedly exploiting a same local region, the parent chosen for local search should be different in different generations.

After variation, the performance of these $2N$ offspring designs is evaluated by a computationally cheap approximation method (lines 23-27). Next, the environmental selection is executed. Only those survivors will be re-evaluated by the expensive fitness function. Following [11], environmental selection keeps the top λN designs from the current population as survivors (lines 28, 29). $\lambda \in (0, 0.6]$ denotes the survival rate. All the real-evaluated designs, including their structures, controllers, and actions, are stored. AIEA outputs the optimal design among these stored designs when the termination condition is satisfied.

In each generation, the fitness of most designs is evaluated with low computational cost, and only a small number of designs are evaluated using an expensive fitness function. In the context of soft robot design problems, the computational time required for fitness evaluations (i.e., training controllers via reinforcement learning) is significantly longer than that of other algorithmic steps. Therefore, compared with GA, it does not lead to extra time costs even though AIEA creates more designs.

Algorithm 1 Pseudocode of AIEA

Input: N: Population size; N_{max}: Maximum number of real fitness evaluations.
Output: $\boldsymbol{s}^*$: the optimal design; $\boldsymbol{a}^*_{\boldsymbol{s}^*}$: the control action of $\boldsymbol{s}^*$.

Initialize the population $P = \{\boldsymbol{p}_1, ..., \boldsymbol{p}_N\}$
# Real evaluation
for each $\boldsymbol{p}$ in P **do**
    Optimize control
    Calculate the outer objective based the optimized control and structure
end for
Add the structures of real-evaluated designs to S
Add the control actions of real-evaluated designs to A
Set number of current used fitness evaluations $N_{cur} = N$
while $N_{cur} < N_{max}$ **do**
    $O_1, O_2 := \emptyset$
    # Variation by mutation
    while $|O_1| < N$ **do**
        Randomly select a parent p_r from P
        o_m := mutation(p_r)
        $O_1 \cup \{o_m\}$
    end while
    # Variation by local search
    Set p_b as the design with the best performance in S
    while $|O_2| < N$ **do**
        o_l := mutate(p_b)
        $O_2 \cup \{o_l\}$
    end while
    $O := O_1 \cup O_2$
    # Approximated Evaluation
    $\Omega := \emptyset$
    for each o in O **do**
        $\hat{r_o}$:= ApproximateEvaluation(o, S, A)
        e_o := ErrorEstimation(o, S, A)
        $\Omega \cup \{\hat{r_o} + e_o\}$
    end for
    $\lambda = 0.6 \times (1 - N_{cur}/N_{max})$
    O' := EnvironmentalSection($O, \Omega, \lambda N$)
    # Real Evaluation
    Evaluate survivors in O' by the real fitness function
    $N_{cur} = N_{cur} + |O'|$
    # Pool updating
    Add the structures of re-evaluated designs to S
    Add the control actions of re-evaluated designs to A
    Select top N designs as the parent population P
end while
Set $\boldsymbol{s}^*$ as the design with the best performance in S

C. Action Inheritance-based Performance Approximation

In automatic robot design problems, the performance evaluation of candidate robot designs is time-consuming. To accelerate the design process, we use a specific surrogate model to replace most expensive evaluations.

According to the learned mapping relation, there are generally two strategies for model construction. The first strategy is to construct an end-to-end model in which the surrogate model is used to approximate the mapping from structures to their performance. It can be formulated as

$$\hat{r_c} = \mathfrak{M}_1(\boldsymbol{s}_c), \tag{2}$$

where $\mathfrak{M}_1(\cdot)$ is the learned model, $\hat{r_c}$ is the predicted task performance, and $\boldsymbol{s}_c$ is the structure matrix of a candidate robot design c. According to Equation (1), the task performance of a robot design is not only related to its structure but also to control policies. Without considering control, it is difficult to accurately learn this mapping using generic machine learning models. Although this strategy is intuitive and straightforward, the forecasting precision is limited. The second strategy uses surrogate models to map a structure to its control actions. It can be formulated as

$$\hat{\boldsymbol{a}_c} = \mathfrak{M}_2(\boldsymbol{s}_c), \tag{3}$$

where $\mathfrak{M}_2(\cdot)$ is the learned model, $\hat{\boldsymbol{a}_c}$ is the predicted control actions. After that, the task performance can be obtained by $F(\boldsymbol{s}_c, \hat{\boldsymbol{a}_c})$, where $F(\cdot)$ is the objective function of the outer optimization. As described in Section II, the control actions are a series of intelligent behaviors after the robot interacts with the environment, rather than a simple numerical vector. With limited training data, it is challenging to implement this process using traditional machine learning methods. To overcome this, we propose an action inheritance-based model construction method.

In evolution, a morphology structure is generated by changing the structure from its parent, resulting in a certain similarity between these two structures. Given a task, the inner control optimization only relates to structures. The robots with a significant structural similarity show a high homologous in control policy. For a robot, the control of its similar robot can be seen as an approximation of its optimal control. The purpose of using the second strategy to construct surrogate models is to obtain near-optimal control actions quickly. Therefore, getting near-optimal control actions based on structural similarity can act as a surrogate model. There are two techniques to reuse the control policy of real-evaluated robot designs. One is to transfer the well-trained controller, and the other is to reuse actions provided by the controller. In VSR design problems, the input and output of each controller are different. Without retraining, the transferred controller cannot directly act on a new robot design. On the contrary, reusing the controller's output is much easier.

The action inheritance-based performance approximation mainly includes two steps: structure matching and actions inheriting. The structure matching is to find the most similar one to the current design from real-evaluated designs. Actions inheriting enables control actions of the found design to act on the current design through some easy-to-implement operations. For a candidate robot design c, its material matrix of physical structure is denoted as $\boldsymbol{s}_c$. The matching process can be described as

$$\boldsymbol{s}_m = \arg\min_{\boldsymbol{s}_i \in S} h(\boldsymbol{s}_i - \boldsymbol{s}_c), \tag{4}$$

where $\boldsymbol{s}_i$ is the structure matrix of robot i. S is an archive. All structures of real-evaluated designs are stored in S. $h(\cdot)$

is used to calculate the number of non-zero elements in a matrix. Fewer non-zero elements in the difference between the two structure matrices show a more remarkable similarity. $\boldsymbol{s}_m$ represents the structure matrix of real-evaluated design m which has the most similarity to $\boldsymbol{s}_c$.

Suppose that the optimized control actions of m is expressed as $\boldsymbol{a}_m^* \in \mathbb{R}^{t \times l_m}$ where t represents the number of simulation steps and l_m represents the action length. At each time step, an action vector of $\boldsymbol{a}_m^*$ is applied to step the simulation. Since each component of the action vector is associated with an actuator voxel, the action length equals to the number of actuator voxels in a structure. For different designs, their simulation steps are identical in the same task environment, but their action length differs. Structure matching considers the similarity of all voxels while action inheriting focuses more on actuator voxels, including their positions and types. Even though the two designs, c and m, have the most significant structural similarity, in most cases, the optimized control actions of m cannot be directly used to control c. Take Fig. 3 as an example. If $\boldsymbol{a}_m^*$ is directly used to control c, the actions applied on the 9th voxel of m will apply on the 12th voxel of c (The 9th voxel of m and the 12th voxel of c are both the first actuator voxel in their structures). It is inappropriate because the positions of these two actuator voxels are different. The color of the 12th voxel of m and c is blue, indicating that they are both horizontal actuator voxels. Ideally, the actions applied on the 12th voxel of m act on the 12th voxel of c since the positions and types of these two actuator voxels are the same.

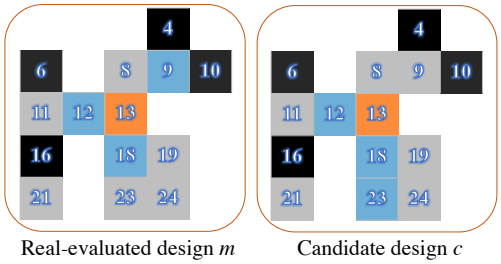

Fig. 3. An example shows that the optimized control actions of m cannot be directly used to control c. Blue, orange, gray, and black represent horizontal actuator voxels, vertical actuator voxels, soft voxels, and rigid voxels.

In VSRs, the control actions only apply to actuator voxels. Considering the difference of actuator voxels in position and type, some adjustments need to be performed before using optimized actions of a real-evaluated robot m to control a candidate robot c. In a structure matrix, we use 3 or 4 to represent a vertical or horizontal actuator voxel, respectively. The adjustment includes the following two basic operations.

1) **Extract**. If the actuator voxels at some positions of $\boldsymbol{s}_m$ is the same as that of $\boldsymbol{s}_c$, then the actions applied on these voxels of $\boldsymbol{s}_m$ can be associated with the corresponding voxels of $\boldsymbol{s}_c$. Assume that $\boldsymbol{\nu}_m$ represents the set of actuator positions and types in $\boldsymbol{s}_m$. Taking Fig. 4a as an example, $\boldsymbol{\nu}_m = \{(8,4),(11,4),(12,3),(17,4),(22,4)\}$, each element of the set is a tuple consisting of the position number and material type. The tuple $(8,4)$ indicates that the eighth voxel in $\boldsymbol{s}_m$ is a horizontal actuator. From Fig. 4a, we can observe that $\boldsymbol{\nu}_c = \{(8,4),(11,4),(12,3),(22,4)\}$ and $\boldsymbol{\nu}_c \cap \boldsymbol{\nu}_m = \boldsymbol{\nu}_c$. In this case, the actions applied on the 8th, 11th, 12th, and 22nd voxels of m can be extracted and applied to the voxels at the corresponding positions of c. Specifically, the first to third and fifth dimensions of $\boldsymbol{a}_m^*$ are extracted to form a new action matrix to control c. This process is shown in Fig. 4a.
2) **Fill**. For a candidate robot design, the dimension of extracted actions is usually smaller than that of its actual control actions. Taking Fig. 4b as an example, $\boldsymbol{\nu}_m = \{(8,4),(11,4),(12,3),(17,4),(22,4)\}$, $\boldsymbol{\nu}_c = \{(8,4),(11,4),(12,3),(22,3)\}$, $(\boldsymbol{\nu}_m \cap \boldsymbol{\nu}_c) \subseteq \boldsymbol{\nu}_c$. Since the type of 22nd voxel of m is different from that of c, the actions applied on the 22nd voxel of c cannot be inherited from the $\boldsymbol{a}_m^*$. Considering the time cost, we use random actions to fill the missing part. In this case, the first three dimensions of $\hat{\boldsymbol{a}_c}$ are extracted from $\boldsymbol{a}_m^*$ and the fourth dimension is filled with random actions where $\hat{\boldsymbol{a}_c}$ is the inherited actions to control c.

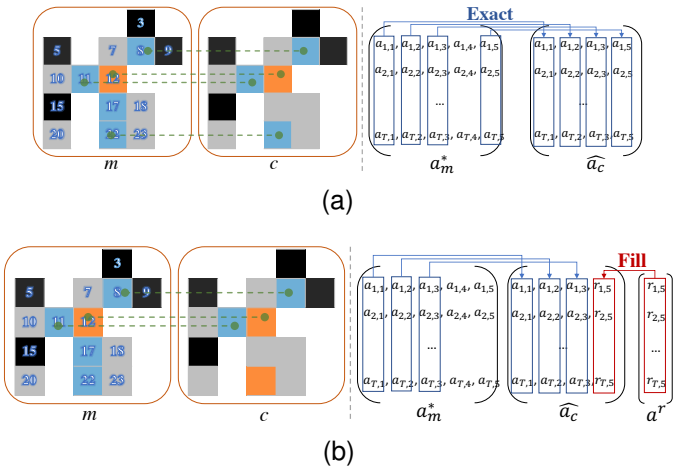

Fig. 4. Action inheritance diagram. m is a real-evaluated robot design, c is a candidate robot design. The structural similarity between m and c is the largest. $\boldsymbol{a}_m^*$ and $\hat{\boldsymbol{a}_c}$ represent the optimized control actions of m and the actions inherited from m, respectively. a^r is a random action. (a) Actions inheriting only contains extraction. (b) Actions inheriting contains extraction and filling.

The pseudo-code of the action inheritance-based performance approximation is shown in Algorithm 2, which includes three parts: structure matching (lines 1-7), actions inheriting (lines 8-20) and evaluation (line 21). It is worth noting that in VSRs design problems, the process of training controllers is time-consuming, and evaluating task performance based on structure and control actions is not time-consuming.

The action inheritance is a process of applying the optimized control policy of a real-evaluated robot to an un-evaluated robot. Once a candidate robot is assessed as promising by the action inheritance-based performance approximation, RL will be used to train its controller. If the opposite situation occurs, it will not appear in the subsequent optimization process. Therefore, action inheritance is actually to pre-select some promising robot designs that use RL to optimize their controllers. It does not participate in optimizing the control

mechanism of any robots. On the other hand, inherited actions disregard the robot's observations. However, this is not a bad situation for AIEA. Environmental selection requires comparing the performance of a parent and its offspring to preserve either. Suppose an offspring performs better under the control that disregards observations than its parent under the optimized control. In that case, it is more likely that the design structure of the descendant is superior to that of the parent. It is also possible to quickly distinguish robots who can enter the next generation without training an enormous number of controllers. The ingenious cooperation of environmental selection and action inheritance has achieved the goal of rapid robot evolution.

Algorithm 2 Pseudocode of Action Inheritance

Input: S: the set of structures of real-evaluated designs; A: the set of control actions of real-evaluated designs; c: the candidate design.
Output: $\hat{r_c}$: the approximated performance of c.

```
    # Structure Matching
    D := ∅
    s_c := the structure of c
    for each s_i in S do
        D := D ∪ {h(s_i − s_c)}
    end for
    m := argmin(D)
    Set s_m as the structure of m
    # Action Inheritance
    Set a*_m as the optimized control actions of m
    â_c := Initialize a random matrix
    k_1, k_2 := 0
    for each s_c^ij in s_c do
        # Extract
        if s_m^ij = s_c^ij = 3 or s_m^ij = s_c^ij = 4 then
            â_c[:,k_1] = a_m[:,k_2]
        end if
        # Fill
        if (s_c^ij = 3 or s_c^ij = 4) and s_m^ij ≠ s_c^ij then
            â_c[:,k_1] = random()
        end if
        k_1 + 1 if s_c^ij=(3 or 4)
        k_2 + 1 if s_m^ij=(3 or 4)
    end for
    # Evaluation
    r̂_c = F(s_c, â_c)
```

D. Error Estimation

In AIEA, the performance of most designs is evaluated by an approximate method. The approximated evaluation can be expressed as

$$\hat{r_c} = F(\boldsymbol{s}_c, \hat{\boldsymbol{a}_c}), \tag{5}$$

where c is a candidate robot design, $\boldsymbol{s}_c$ is the structure, $\hat{\boldsymbol{a}_c}$ is the inherited control actions, $F(\cdot)$ is the objective function of the outer optimization, $\hat{r_c}$ is the close task performance. Since the control actions $\hat{\boldsymbol{a}_c}$ are near-optimum, there is an error e_c between $\hat{r_c}$ and its actual performance r_c. Intuitively, e_c is the performance difference of c under the optimal and near-optimal control. If the difference between these two control policies is measurable, the error can be accurately evaluated. However, the optimal control is not available without training a controller. In the case that the accurate method is not feasible, we adopt an approximate method to make $\hat{r_c}$ as close as possible to r_c.

In evolution, the candidate design c is created by a mutation operator. There must be a parent m similar to it. Based on structural similarity, we assume that the performance difference of m under the optimal and near-optimal control is approximately equal to that of c. The error e_c can be estimated as

$$e_c \approx e_m = F(\boldsymbol{s}_m, \boldsymbol{a}^*_m) - F(\boldsymbol{s}_m, \hat{\boldsymbol{a}_m}), \tag{6}$$

where $\boldsymbol{a}^*_m$ is the optimized control actions of m. $\hat{\boldsymbol{a}_m}$ is an approximation of $\boldsymbol{a}^*_m$ which is obtained by adding random perturbation to $\boldsymbol{a}^*_m$. Since m is a real-evaluated design, its optimized control actions are available. Compared with the time cost of training a controller, the time cost of obtaining $\hat{\boldsymbol{a}_m}$ is almost negligible when $\boldsymbol{a}^*_m$ is available.

If the perturbation is added to all components of $\boldsymbol{a}^*_m$, the obtained $\hat{\boldsymbol{a}_m}$ may not be an approximation of $\boldsymbol{a}^*_m$ but a matrix entirely unrelated to $\boldsymbol{a}^*_m$. Conversely, if the perturbation is only added to a particular column of $\boldsymbol{a}^*_m$, it may result in a significant error difference when different columns are chosen. To overcome this, we add random perturbation to each column of $\boldsymbol{a}^*_m$ in turn and take the average as the approximate error.

For a candidate morphology design c, the perturbation-based error estimation can be described as

$$e_c \approx \frac{l_m \times F(\boldsymbol{s}_m, \boldsymbol{a}^*_m) - \sum_{j=1}^{l_m} F(\boldsymbol{s}_m, \boldsymbol{a}^*_m + \boldsymbol{r}_j)}{l_m}, \tag{7}$$

where m is a real-evaluated robot design. $\boldsymbol{s}_m$ is the structure of m which is most similar to that of c. $\boldsymbol{a}^*_m$ is the optimized control actions of m. t represents the number of simulation steps and l_m represents the number of actuator voxels in m, that is, the length of each action vector. $\boldsymbol{r}_j$ is a matrix in which the values of the jth column are random numbers and the other values are 0. $F(\cdot)$ is the objective function of the outer optimization. After error estimation, the task performance of c is calculated as $\hat{r_c} + e_c$. After that, the value of $\hat{r_c} + e_c$ is employed the metric for environmental selection.

IV. Experimental Study

A. Benchmark

We use Evolution Gym [11] as the test suite to empirically validate the performance of compared algorithms. Evolution Gym (often abbreviated to EvoGym) is a comprehensive benchmark for co-optimizing the structure and control of VSRs. EvoGym employs a unified multi-material voxel-based representation which is able to construct a wide diversity of morphologies. Each morphology is specified as a grid-like structure composed of rigid, soft, horizontal actuator, vertical actuator, and empty voxels. The grid size is set as (5×5). In EvoGym, a state-of-the-art RL algorithm, PPO, is used to train controllers. The reward that a robot can achieve is applied as the fitness to measure the performance of the current structure and the control action. The larger the reward a robot achieves, the better its structure and control are. Under the premise of using the same number of expensive fitness evaluations, a large reward means a fast convergence speed.

In addition to expressive structure design space and effective integration of RL technologies, EvoGym also presents various

tasks with different difficulty levels that span from locomotion to manipulation. In this work, we adopt 31 tasks, including 11 easy, 11 medium, and 9 hard tasks[1]. Table S.I in the supplementary material summarizes the properties of these adopted tasks. Additional information about these tasks, such as the definition of reward functions and the values of simulation parameters, can be found in [11].

The termination condition of all compared algorithms is set as the maximum number of expensive fitness evaluations. For easy, medium and hard tasks, the numbers of expensive evaluations are set as 100, 150, and 200, respectively. The evaluations of compared algorithms are performed on a server with Intel Xeon Silver 4214 CPU @ 2.19GHz on Linux 4.15.0. GPU is not required.

B. Ablation Experiment

Our proposed algorithm AIEA has two components: action inheritance-based performance approximation and error estimation. A detailed illustration of the effectiveness of inheritance-based performance approximation is presented in Section IV-C. In this section, we conduct an ablation experiment to investigate the effect of error estimation. A variant of AIEA without performing error estimation, namely AIEA-noE, is used for performance comparison. Except for the error estimation, the other parameters of AIEA and AIEA-noE are the same. Nine tasks with different difficulty levels are selected as the benchmark.

For comparing the performance of AIEA and AIEA-noE, it is essential to focus on both the final optimization results and the impact of error estimation on the optimization process. The presence or absence of the error evaluation operator directly affects the approximately evaluated fitness values. In SAEAs, since the approximated fitness is only used to select promising candidate designs, it is not strictly required to be infinitely close to its actual value as long as the approximated values can correctly distinguish the performance differences of candidate designs [47]. Therefore, compared to the commonly used prediction error, the rank of promising candidate designs is more suitable to clarify the impact of error estimation on the optimization process.

Fig. 5 shows the reward curve of AIEA and AIEA-noE. Among the nine tasks, AIEA achieves the best performance on seven tasks. For the remaining two tasks, AreaMinimizer-v0 and Thrower-v0, the performance between AIEA and AIEA-noE is quite comparable. To clarify the impact of error estimation, we compare the environmental selection results of AIEA and AIEA-noE in some generations. Take the population at the first generation as an example. There are $2N$ designs where N is the population size. After the environmental selection, $\lambda N(\lambda \in (0, 0.6])$ survivals are enter the next generation. Ψ_{wE} and Ψ_{noE} represents the set of survivals in AIEA and AIEA-noE, respectively. To compare the differences between Ψ_{wE} and Ψ_{noE}, we first re-evaluate $2N$ designs using real fitness function and sort them in descending order based on their actual rewards. Next, each design is assigned a rank that

[1]EvoGym consists of 32 tasks. But we found a bug in the simulation of the task Traverser-v0. Only 31 of these tasks are used in this paper.

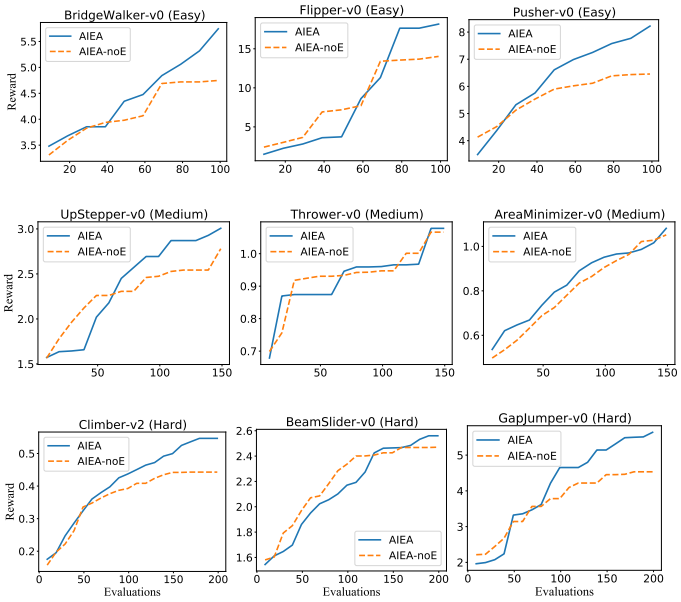

Fig. 5. The reward curve of AIEA and AIEA-noE over expensive evaluations on 9 tasks. All the curves are averaged 5 independent runs.

represents the position in the descending order. For instance, the rank of the best design is 0, and the rank of the worst design is $2N$. Finally, we calculate the average rank of designs in Ψ_{wE} and Ψ_{noE}. Considering the time cost, we choose Pusher-v0, AreaMinimizer-v0, and GapJumper-v0 as the analysis cases.

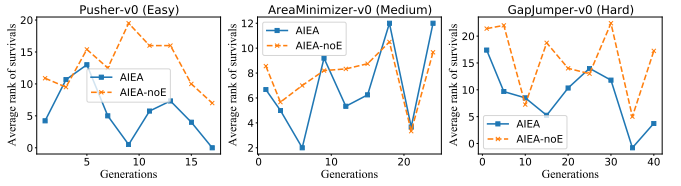

Fig. 6. The average rank of survivals in AIEA and AIEA-noE.

Fig. 6 shows the average rank of survivals at different generations. The smaller the average rank, the better the performance of the selected designs. On tasks Pusher-v0 and GapJumper-v0, a weak statistical test shows that these designs selected by AIEA are better than those selected by AIEA-noE in most generations. On task AreaMinimizer-v0, the average rank of survivals selected by AIEA is roughly the same as that of AIEA-noE in most generations. This result appears to that the disparity between these survivals selected by AIEA and AIEA-noE is less distinctive. Overall, the results in Figs. 5 and 6 illustrate that error estimation has played a positive role in improving the performance of AIEA.

C. Comparative Experiments

In this section, we compare AIEA with the following three algorithms.

1) Genetic algorithm (GA). Considering the characteristics of black-box evaluation in automatic design, GAs are

widely used to evolve VSRs [11], [19], [48]. GAs achieve morphological evolution by iteratively generating new designs and eliminating underperforming designs. At each generation of GA, top λN performing designs are first chosen as parents and then $\lceil(1-\lambda)N\rceil$ offsprings are created by iteratively sampling and mutating one of those parents. After that, the control optimization performs and provides a unique controller for each offspring. While the termination condition is satisfied, GA outputs the optimal design in the current population.

2) Bayesian optimization (BO). In VSRs design problems, control optimization is time-consuming. Moreover, the fitness evaluation requires optimized control as input, which makes the process expensive. BO is well suited when the fitness evaluations are expensive. Compared with GA, BO uses a computationally cheap surrogate model to replace expensive fitness evaluations. BO is selected as a comparison algorithm in this work. Followed by [49], we use Gaussian process as the surrogate model, batch Thompson sampling for extracting the acquisition function.
3) Random forest-assisted evolutionary algorithm (RFEA). As mentioned in [34], [38], random forest is suitable for directly accepting the discrete data as input. Compared with BO, RFEA uses a random forest model as the surrogate model to learn the mapping between structures and rewards. Expect for the selection of surrogate models, other settings are same as that of GA.

BO, RFEA, and AIEA belong to SAEAs. BO and RFEA use the modeling method mentioned in Equation (3) while AIEA uses the modeling method mentioned in Equation (4). For all compared algorithms, the population size is set as 20. PPO is applied for control optimization. Followed by [11], mutation is only applied to create offsprings. The probability of mutation is set to 0.1. The value of survival rate λ is set as $0.6 \times (1 - N_{cur}/N_{max})$ where N_{cur} and N_{max} are the current and maximum number of real fitness evaluations. The maximal real fitness evaluations on easy, medium, and hard tasks are set as 100, 150, and 200, respectively. In RFEA, the number of trees is set as 300. The other parameters in the optimization are the same as those provided in [11].

TABLE I
MEAN (STANDARD DEVIATION) OF REWARDS OBTAINED BY GA, BO, RFEA, AND AIEA ON 11 EASY TASKS. ALL THE VALUES ARE AVERAGED 5 INDEPENDENT RUNS. THE BEST RESULTS ARE HIGHLIGHTED.

Task	GA	BO	RFEA	AIEA
Walker-v0	9.37(1.09)	9.02(1.49)	9.07(1.33)	**10.32(0.53)**
BridgeWalker-v0	3.58(0.33)	3.60(0.35)	4.50(0.55)	**5.74(1.08)**
Carrier-v0	3.36(1.07)	2.80(0.76)	5.55(2.96)	**8.69(1.56)**
Pusher-v0	6.21(1.06)	6.46(0.67)	6.79(1.08)	**8.22(0.95)**
BeamToppler-v0	2.76(0.05)	3.57(1.98)	3.01(0.34)	**5.86(2.49)**
DownStepper-v0	4.61(0.30)	4.68(0.60)	4.85(0.68)	**6.59(1.26)**
AreaMaximizer-v0	1.74(0.42)	1.45(0.17)	1.56(0.15)	**2.37(0.17)**
WingspanMazimizer-v0	0.56(0.07)	0.50(0.11)	0.57(0.16)	**0.83(0.09)**
Flipper-v0	11.06(8.75)	2.97(1.03)	4.65(0.87)	**18.16(10.27)**
Jumper-v0	**0.81(0.41)**	0.43(0.11)	0.45(0.16)	0.46(0.08)
Balancer-v0	0.08(0.02)	0.08(0.01)	0.09(0.01)	**0.10(0.03)**
Average Rank	2.91	3.55	2.45	1.09

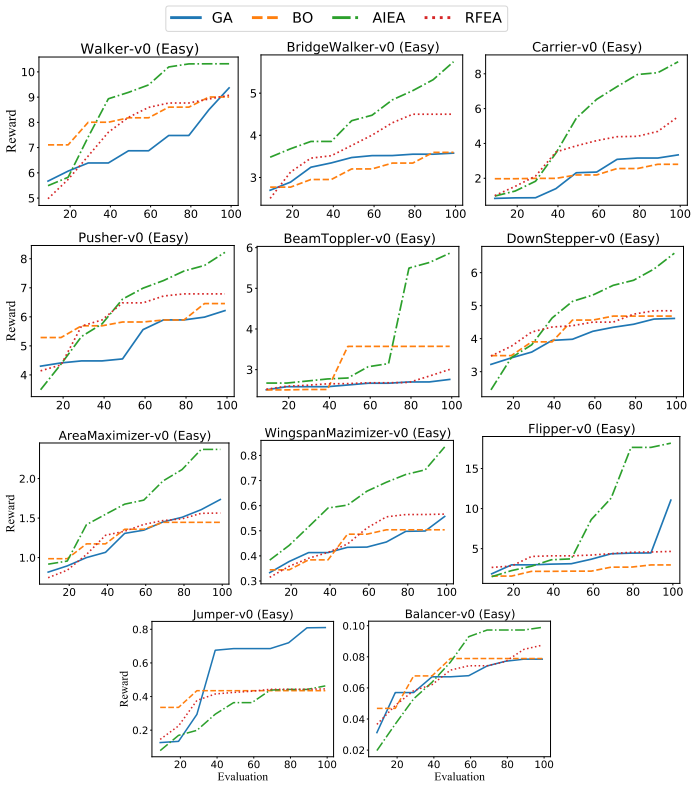

Fig. 7. The reward curve of GA, BO, RFEA, and AIEA over evaluations on 11 easy tasks. All the curves are averaged 5 independent runs.

The average rewards and reward curves of GA, BO, RFEA, and AIEA on easy, medium, and hard tasks are shown in Tables I, II, III and Figs. 7, 8, 9, respectively. To clearly and intuitively demonstrate the performance of optimal robots obtained by GA, BO, RFEA, and AIEA on various tasks, we have uploaded the simulation animation to GitHub[2] and included the boxplots of obtained final rewards in the supplementary material. A weak statistical test shows that AIEA outperforms the other three algorithms on most tasks, indicating that AIEA has a faster convergence speed with fewer fitness evaluations.

A weak statistical test shown in Table I and Fig. 7 illustrates that AIEA achieves the best performance on 10 out of 11 easy tasks. Based on the simulation animation and the results in Table I, we can find the optimal robot obtained by AIEA completes most easy tasks, except for the task Jumper-v0. Moreover, compared with GA, AIEA saves about half the computing cost. On task Jumper-v0, we compare the performance of the optimal robots obtained by AIEA and given in [11], finding that these two robots did not jump to the height mentioned in [11]. Even given an optimal physical structure, PPO cannot obtain a well-trained controller on Jumper-v0. Therefore, the reason for the poor performance of AIEA on task Jumper-v0 is the need for optimal control policy rather than AIEA cannot find an optimal physical structure. In other tasks, AIEA evolves intelligent robots that can complete corresponding goals. The next is RFEA, with BO performing the worst. Overall, on easy tasks, AIEA can evolve a fully

[2]https://github.com/HandingWangXDGroup/AIEA/tree/main/animation

successful robot.

TABLE II
MEAN (STANDARD DEVIATION) OF REWARDS OBTAINED BY GA, BO, RFEA, AND AIEA ON 11 MEDIUM TASKS. ALL THE VALUES ARE AVERAGED 5 INDEPENDENT RUNS. THE BEST RESULTS ARE HIGHLIGHTED.

Task	GA	BO	RFEA	AIEA
BidirectionalWalker-v0	3.44(0.52)	3.24(0.30)	4.19(0.31)	**4.75(0.41)**
Pusher-v1	0.41(0.24)	0.18(0.05)	0.51(0.32)	**1.35(0.94)**
Thrower-v0	0.97(0.11)	1.06(0.12)	**1.16(0.32)**	1.08(0.14)
Climber-v0	0.23(0.05)	0.20(0.02)	0.26(0.04)	**0.29(0.06)**
Climber-v1	0.27(0.05)	0.23(0.04)	0.31(0.09)	**0.39(0.06)**
UpStepper-v0	2.46(0.19)	2.31(0.16)	2.60(0.57)	**3.01(0.76)**
ObstacleTraverser-v0	2.45(0.30)	2.57(0.14)	3.04(0.98)	**3.54(0.74)**
CaveCrawler-v0	2.61(0.90)	1.90(0.01)	3.73(0.74)	**5.03(0.99)**
AreaMinimizer-v0	0.79(0.02)	0.61(0.03)	0.90(0.07)	**1.08(0.10)**
HeightMaximizer-v0	0.28(0.02)	0.28(0.03)	0.33(0.04)	**0.42(0.04)**
Balancer-v1	0.50(0.04)	0.42(0.03)	0.52(0.07)	**0.56(0.05)**
Average Rank	3.18	3.81	1.81	1.18

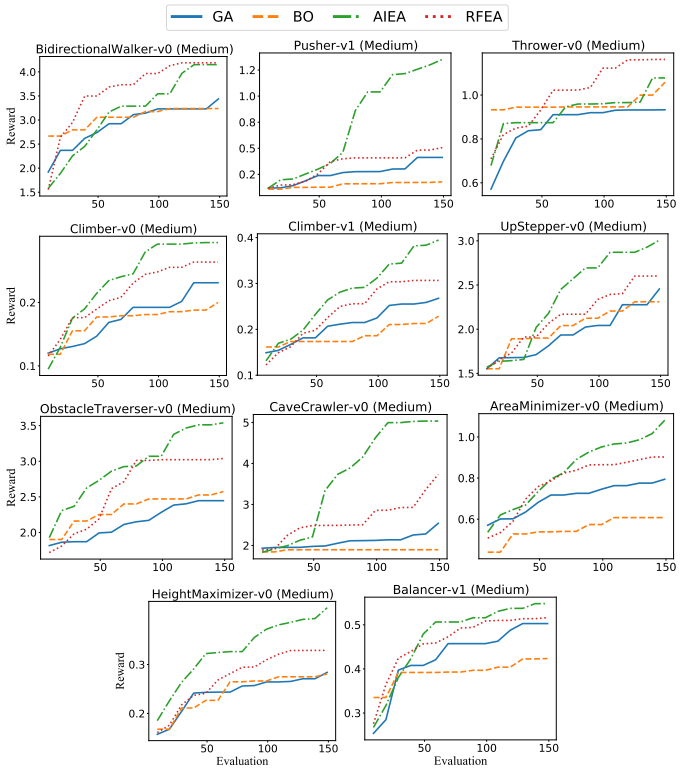

Fig. 8. The reward curve of GA, BO, RFEA, and AIEA over evaluations on 11 medium tasks. All the curves are averaged 5 independent runs.

A weak statistical test shown in Table II and Fig. 8 illustrates that AIEA achieves the best performance on 10 out of 11 medium tasks, followed by RFEA and GA, BO performs worst. On task Thrower-v0, the results in Table IV indicate that all algorithms have the same performance. However, by observing Fig. S.II in the supplementary material, we can find that the upper bound of the rewards obtained by RFEA is the largest. Thrower-v0 requires the robot to throw a soft rectangular box as far as possible without moving itself significantly from its original position. By querying the information in Table S.I of the supplementary material, we can find that Thrower-v0 is a task of medium difficulty, but its simulation step is the smallest. That is, robots need to complete a relatively tricky goal with few control actions. In this case, whether the control action is optimal significantly impacts the task performance. AIEA uses a near-optimal control for evaluating the performance of most robots, which results in poor reliability of the performance approximation. However, RFEA directly uses the surrogate model to learn the mapping between structures and rewards, which to some extent, weakens the influence of the optimal control. Therefore, the average reward obtained by RFEA in five independent runs is larger than that of AIEA on task Thrower-v0. A week statistical test shows the outstanding performance of RFEA on task Thrower-v0. Through the simulation animation, we can also discover that although the best design obtained by AIEA outperforms that of the other three algorithms on two climbing tasks, it still needs to evolve further to find a successful robot. One reason is that climbing is a relatively complex task requiring the robot to climb as high as possible through a vertical channel. The width of this channel is equal to the robot's horizontal width. Robots need to learn how to grasp the wall before they can climb upwards. However, through the simulation animation, it can be seen that even the optimal design is not able to get traction on the wall, resulting in almost identical performance among different designs in the population. The lack of environmental selection pressure is the main reason for the failure of evolution.

TABLE III
MEAN (STANDARD DEVIATION) OF REWARDS OBTAINED BY GA, BO, RFEA, AND AIEA ON 11 HARD TASKS. ALL THE VALUES ARE AVERAGED 5 INDEPENDENT RUNS. THE BEST RESULTS ARE HIGHLIGHTED.

Task	GA	BO	RFEA	AIEA
Carrier-v1	3.49(0.28)	3.40(0.12)	3.14(0.01)	**3.65(0.02)**
Catcher-v0	-1.43(1.42)	-0.82(1.85)	-2.23(1.55)	**0.19(1.48)**
BeamSlider-v0	2.24(0.30)	1.62(0.05)	1.91(0.35)	**2.56(0.07)**
Lifter-v0	0.40(0.31)	0.08(0.09)	0.61(0.53)	**0.86(0.42)**
Climber-v2	0.42(0.05)	0.31(0.07)	0.46(0.04)	**0.55(0.08)**
ObstacleTraverser-v1	1.59(0.08)	1.66(0.12)	**1.75(0.27)**	1.66(0.05)
Hurdler-v0	1.36(0.11)	1.34(0.11)	1.24(0.27)	**1.49(0.42)**
PlatformJumper-v0	1.81(0.09)	1.73(0.04)	1.94(0.09)	**2.07(0.14)**
GapJumper-v0	2.77(0.65)	3.99(0.54)	3.20(0.73)	**5.64(0.95)**
Average Rank	3.00	3.22	2.55	1.22

A weak statistical test shown in Table III and Fig. 9 illustrates that AIEA achieves the best performance on 8 out of 9 hard tasks, followed by RFEA. As shown in Fig. S.III of the supplementary material, except for BO, the other three algorithms have almost the same median reward and reward upper bound on task Carrier-v1. The upper bound of rewards obtained by BO is the smallest. Carrier-v1 requires the robot to carry a box to a table and place the box on the table. The robots obtained by the four algorithms can successfully carry the box to the table, but none of them can place the box on the table. This is mainly because there is a significant difference in the morphological structures required to complete carrying and placing. The controller first learns the control policy for carrying, these algorithms tens to search for the structure that completes the first half of the task. On task ObstacleTraverser-v1, RFEA outperforms AIEA. Among all the locomotion tasks, the terrain of ObstacleTraverser-v1 is the bumpiest. The entire landscape is covered with many rigid protrusions of different shapes. Especially in the first half of the terrain, there is a major terrain depression. In this task, the robot

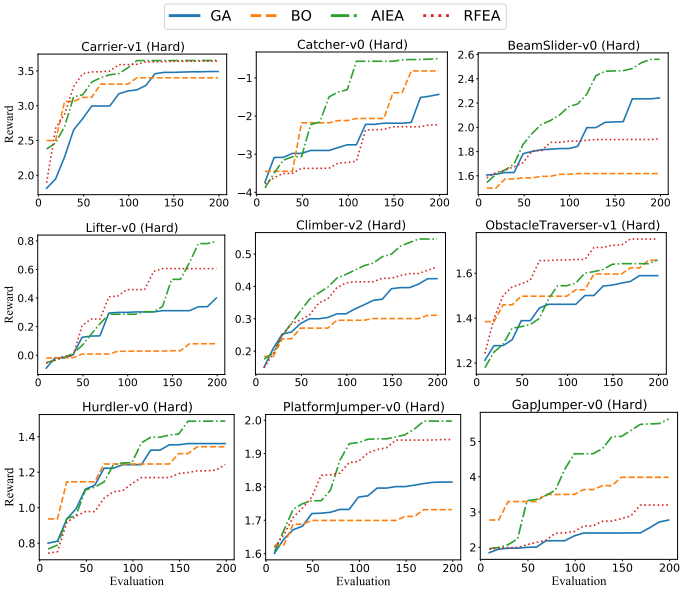

Fig. 9. The reward curve of GA, BO, RFEA, and AIEA over evaluations on 11 hard tasks. All the curves are averaged 5 independent runs.

walks through the bumpy terrain and maintains its balance as much as possible to avoid getting stuck in a depression. The task requires the controller to be able to apply appropriate deformations to each actuator voxel accurately. AIEA uses near-optimal action to control candidate robots to complete the task-related goal, causing most candidate robots to be trapped in a depression and unable to continue moving forward. Most robots have the same approximated performance. AIEA cannot find promising robots from these candidate robots. On the contrary, RFEA uses a random forest model to learn the mapping between structure and task performance. To some extent, RFEA ignores the impact of optimal control on task performance. Although the prediction error of the random forest model may be relatively large, it can at least distinguish these robots without causing chaos. That is why RFEA is better than AIEA on task ObstacleTraverser-v1. Overall, on most hard tasks, AIEA outperforms the other algorithms with few expensive fitness evaluations.

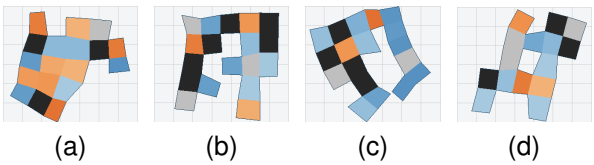

Fig. 10. Optimal robot designs on task ObstacleTraverser-v0. (a), (b), (c), and (d) are the structures of optimal designs obtained by GA, BO, RFEA, and AIEA, respectively.

In summary, we can obtain the following observations from Tables I-III, Figs. 7-9, and the simulation animation on GitHub.

1) Under the verification of a weak statistical test, AIEA shows the best performance on most tasks with limited computational resources. The optimal designs evolved by AIEA achieve satisfying performance for all easy tasks (The simulation animation on GitHub can illustrate the conclusion). Although the optimal designs obtained by AIEA did not reach the final goal for most medium tasks, they all evolved basic features to adapt to task environments. Taking the task ObstacleTraverser-v0 as an example, it requires the robot to walk across increasingly bumpy terrain. As shown in Fig. 10, the optimal robot obtained by AIEA has evolved trunk-leg-like features that can help the robot step over obstacles of different heights while maintaining balance - much like how humans move their legs to walk. Although AIEA needs to evolve a fully successful robot for most hard tasks, it achieves the best performance under the same number of fitness evaluations.
2) We compare the performance of three SAEAs, AIEA, BO, and RFEA. The difference is that RFEA and BO use the surrogate model to predict the results of the outer objective function (The first modeling method introduced in Section III-C), while AIEA uses the surrogate model to predict the results of the inner objective function (The second modeling method introduced in Section III-C). In most tasks, AIEA appears to perform better than BO and RFEA, indicating that the second modeling method is more effective than the first one in automatic soft robots design problems. However, the above analysis is based on the results of 5 independent runs, and further analysis is required.
3) In most tasks, the average reward obtained by RFEA is larger than that of BO. This shows that random forest is more suitable for combinatorial optimization problems than Gaussian process. The reasons for the poor performance of BO can be summarized as follows. Firstly, the Gaussian process is a similarity-based model [50]. When the decision variable is continuous, distance, such as the Euclidean distance, is often used to measure the similarity between two solutions in the decision space. However, in soft robot design problems, the decision variable is the type of voxels. Strictly speaking, there is no concept of distance between two voxel types, causing the inaccurate similarity measurement between two robots in the decision space. Secondly, BO has advantages in solving low-dimensional problems [51]. In robot design problems, the dimension of decision variables is 25. For BO, this is already a relatively high-dimensional problem. Last, in the evaluation process of robots, RL is used to train controllers. Due to the randomness in RL, the evaluation is noisy [11]. The noise done by RL is also a reason for the deterioration of BO.

D. Further Investigations

1) Effect of independent runs: Although we propose an algorithm to accelerate the robot design process, its optimization process still requires some time. Taking the simplest task Walker-v0 as an example, on a cloud server configured with 24-core Intel Xeon Silver CPU, running the AIEA with a single thread takes about one day, and running with multiple threads takes about 2 hours. The running time of main steps in

GA and AIEA is shown in Section S.II of the supplementary material. Considering the time cost, we set the number of independent runs to 5, which makes it impossible to perform statistical tests on these results.

Without statistical tests, the description of one algorithm performing better than another is not rigorous. To more accurately demonstrate the effectiveness of AIEA, all compared algorithms are independently run 20 times on 14 tasks. The results are analyzed by the Wilcoxon signed-rank test [52]. The significance level is set as 0.05. The results of the statistical test are shown in Table IV. We can find that AIEA significantly outperforms GA and BO on 12 tasks. AIEA significantly outperforms RFEA on 9 tasks. AIEA and RFEA perform equally in the remaining 5 tasks. From the results of statistical tests, AIEA shows the best performance on most tasks.

TABLE IV
MEAN OF REWARDS OBTAINED BY GA, BO, RFEA, AND AIEA ON 14 TASKS. ALL THE VALUES ARE AVERAGED 20 INDEPENDENT RUNS. THE BEST RESULTS ARE HIGHLIGHTED.

Task	Difficulty	GA	BO	RFEA	AIEA
Walker-v0	Easy	8.84+	8.86+	9.04+	**10.03**
BridgeWalker-v0		3.81+	3.61+	4.24+	**5.69**
Carrier-v0		3.80+	2.89+	6.02+	**8.31**
Pusher-v0		6.45+	6.54+	7.01≈	**7.98**
DownStepper-v0		4.66+	4.25+	5.08+	**5.93**
Thrower-v0	Medium	0.95≈	1.02≈	**1.03**≈	1.01
UpStepper-v0		2.47+	2.35+	2.07+	**2.97**
HeightMaximizer-v0		0.30+	0.27+	0.31+	**0.40**
Balancer-v1		0.49+	0.46+	0.53≈	**0.56**
CaveCrawler-v0		2.63+	1.91+	3.32+	**5.14**
Catcher-v0	Hard	-0.91≈	-1.86≈	-0.97≈	**-0.74**
BeamSlider-v0		2.22+	1.63+	2.18+	**2.47**
PlatformJumper-v0		1.84+	1.81+	1.98≈	**2.02**
Hurdler-v0		1.39+	1.35+	1.45+	**1.62**
+/≈		12/2	12/2	9/5	

Symbols + and ≈ indicate that the result obtained by AIEA is significantly better and similar to that obtained by other algorithms, respectively.

2) Effect of reproduction operators: In this work, offspring designs are generated by a sample mutation operator. As well known, the reproduction operator directly affects the searching ability of colony evolution-based algorithms [53]. We construct a set of comparison experiments to explore the impact of different reproduction operators on solving robot design problems. GA and AIEA are selected as test algorithms, which belong to EAs and SAEAs, respectively. In GA and AIEA, only mutation is used to generate offspring designs. In GAC and AIEAC, mutation and uniform crossover are used to create offspring designs. The probabilities of crossover and mutation are set to 0.9 and 0.1, respectively. In uniform crossover, the independent probability for each voxel to be exchanged is set to 0.3. Four tasks are chosen as the test suits.

Fig. 11 shows the reward curve of GA, GAC, AIEA, and AIEAC. We can observe several interesting conclusions.

1) On task Flipper-v0, AIEA outperforms AIEAC, and GA outperforms GAC. That is, algorithms without crossover perform better than those with crossover.
2) On task Thrower-v0, AIEA beats AIEAC, while GA and GAC perform almost identically.
3) On task Hurdler-v0, algorithms with crossover perform better than those without crossover, which is precisely the opposite of the conclusion obtained on Flipper-v0.

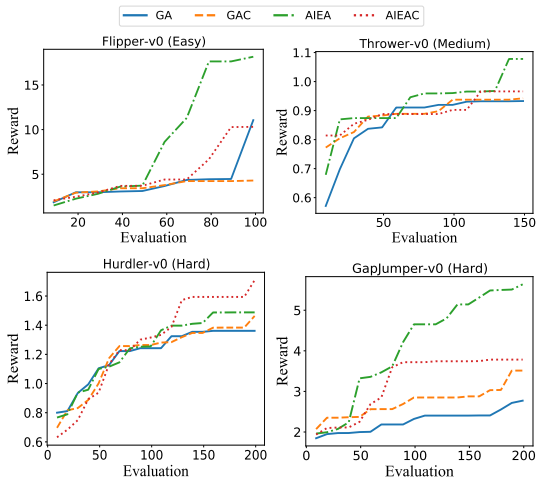

Fig. 11. The reward curve of GA, GAC, AIEA, and AIEAC on 4 tasks. All the curves are averaged 5 independent runs.

4) On task GapJumper-v0, AIEA outperforms AIEAC, while GAC outperforms GA.

Four different conclusions are obtained on four different tasks. On Flipper-v0 and Hurdler-v0, whether using a crossover operator or not has entirely different effects. On GapJumper-v0, the performance of GA improves by introducing a crossover operator, which does not occur in AIEA. Overall, the simple fusion of crossover and mutation operators can improve optimization performance on some tasks, but it is only sometimes effective. This is mainly because the decision space of robot design problems is discrete, and sometimes blindly expanding the search scope is not a wise choice, especially in situations where the number of fitness evaluations is limited. To improve optimization performance by designing effective reproduction operators, it's essential to carefully consider both the task and the algorithm used to solve it. One possible solution could be to train a generator as part of the optimization process. This approach requires a well-designed training process that takes into account all aspects of the task and algorithm.

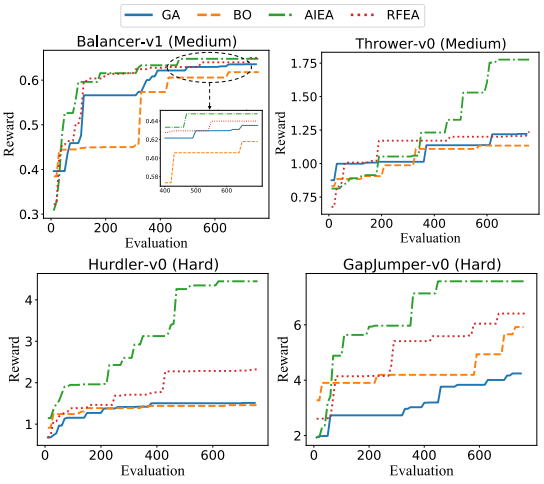

Fig. 12. The reward curve of GA, BO, RFEA, and AIEA over 750 evaluations on 2 medium and 2 hard tasks. All the curves are averaged 2 independent runs.

3) Effect of maximum evaluations: In soft robot design problems, the running time of compared algorithms mainly depends on the number of expensive fitness evaluations. Considering the time cost, we set the maximum number of expensive fitness evaluations to 200. In this section, we construct a set of comparative experiments to demonstrate whether our proposed algorithm still has superiority even with a large number of evaluations. Four tasks are chosen as the test suits. We set the maximum number of expensive fitness evaluations to 750, a value suggested in [11]. From the reward curve shown in Fig. 12 and the simulation animation uploaded to GitHub, we can obtain two observations.

1) On task Balancer-v1, although all the compared algorithms have found the optimal design (the obtained average reward is greater than 0.6 and the simulation animation on Github can support the conclusion), AIEA converges the fastest. More specifically, AIEA obtains its maximum reward around 400 fitness evaluations while other algorithms obtain their maximum rewards after 600 evaluations.
2) On the other three tasks, the average reward obtained by AIEA is always the largest. Compared with the other three algorithms, AIEA fins a better design using the same number of fitness evaluations.

By comparing the results of Figs.7, 8, 9 and 12, it can be found that whether the maximum number of evaluations is set to 200 or 750, the superiority of AIEA can be reflected.

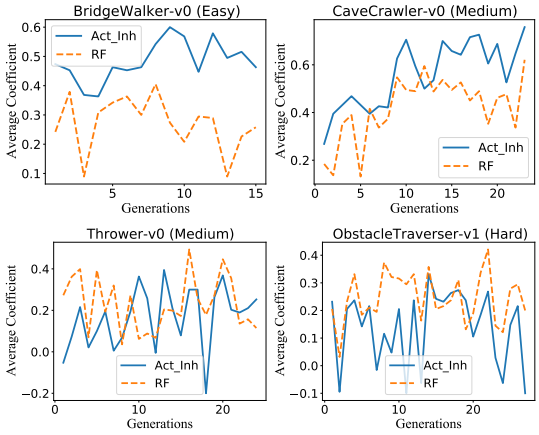

Fig. 13. Approximation accuracy with respect to different surrogate models. Two models are tested: random forest (RF) and action inheritance (Act_Inh). All the curves are averaged 5 independent runs.

4) Approximation accuracy of surrogate models: The comparison with RFEA demonstrates that AIEA improves over RFEA in terms of final policy and morphology, however that is not necessarily caused by AIEA being a more accurate surrogate of fitness. To illustrate whether the accuracy of surrogate models is the main reason AIEA outperforms RFEA, we compare the correlation coefficients between the values predicted by different surrogate models and the actual objective values. Specifically, in each generation, we first use an RF model and the action inheritance process to predict all candidate designs objective values. Then, we use the expensive objective function to evaluate their actual objective values. Finally, we calculate the Kendall rank correlation coefficient between the values predicted by RF and actual values and the coefficient between the values predicted by action inheritance and actual values. τ is used to represent the correlation coefficient, with a value range of -1 to 1. The higher the value of τ is, the stronger the correlation between different ranking results for the same population.

We select four tasks that can be simply divided into two categories. AIEA is superior to RFEA on tasks BridgeWalker-v0 and CaveCrawler-v0, while the opposite is true on tasks Thrower-v0 and ObstacleTraverser-v1. The average coefficients of five independent runs are shown in Fig. 13. We can find that the action inheritance has a higher approximation accuracy than RF on tasks BridgeWalker-v0 and CaveCrawler-v0, consistent with the conclusion that AIEA is superior to RFEA. Moreover, the accuracy gradually increases with the training data. On the other two tasks, the accuracy of RF is higher than that of action inheritance. As analyzed earlier, Thrower-v0 and ObstacleTraverser-v1 are two tasks that require extremely precise control policies. RFEA uses a random forest model to learn the mapping between structure and task performance. To some extent, RFEA ignores the impact of optimal control on task performance. Although the prediction error of RF may be relatively large, it can at least distinguish these robots without causing chaos.

V. Conclusion

In this paper, we propose an action inheritance-based evolutionary algorithm (AIEA) [3] to accelerate the automatic soft robots design. For a candidate design, AIEA uses an inheritance method to obtain a group of near-optimal actions. Then, these inherited actions are employed to control the robot to complete the task and the reward achieved by this robot is taken as the approximated fitness value. The action inheritance-based performance approximation plays the role of surrogate models. Its input and output are the structure of the robot and the near-optimal action, respectively. During the optimization process, the reward of most candidate designs are evaluated using the action inheritance-based approximate method, while only a small number of candidate designs use the reinforcement learning-based real evaluation method. In addition, we also propose an error estimation method to make the approximated rewards close to their actual values.

To evaluate the performance of the proposed algorithm, we compare it with three state-of-the-art algorithms on 31 tasks with various difficulty levels. The experimental results show the effectiveness of our proposed method. However, we can also observe that AIEA is not able to evolve a fully successful robot on some tasks, such as Climber-v2 and BeamSlider-V0. This is mainly because we only used a simple mutation operator to generate new designs during the evolution, resulting in a limited search ability. From the results in Fig.11 we can find that the simple fusion of crossover and mutation operators can improve optimization performance on

[3]The source code of AIEA is available at https://github.com/HandingWangXDGroup/AIEA.

some tasks, but it is only sometimes effective. For our future research, we will consider introducing a self-guided variation operator in AIEA to increase its search efficiency, such as using a generator to generate new designs, extracting implicit features of promising designs, and multitask optimization of simple and hard tasks. Another research direction is to realize transfer learning between different control networks, including parameter sharing, structural fine-tuning, etc.

REFERENCES

[1] H. Oh, A. Ramezan Shirazi, C. Sun, and Y. Jin, "Bio-inspired self-organising multi-robot pattern formation: A review," *Robotics and Autonomous Systems*, vol. 91, pp. 83–100, 2017.

[2] C. Armanini, F. Boyer, A. T. Mathew, C. Duriez, and F. Renda, "Soft robots modeling: A structured overview," *IEEE Transactions on Robotics*, 2023.

[3] Y. Jin and Y. Meng, *Morphogenetic Robotics: A New Paradigm for Designing Self-Organizing, Self-Reconfigurable and Self-Adaptive Robots*, pp. 61–87. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[4] A. Gupta, S. Savarese, S. Ganguli, and L. Fei-Fei, "Embodied intelligence via learning and evolution," *Nature communications*, vol. 12, no. 1, p. 5721, 2021.

[5] D. Jin and L. Zhang, "Embodied intelligence weaves a better future," *Nature Machine Intelligence*, vol. 2, no. 11, pp. 663–664, 2020.

[6] G. Zardini, D. Milojevic, A. Censi, and E. Frazzoli, "Co-design of embodied intelligence: A structured approach," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7536–7543, IEEE, 2021.

[7] P.-Q. Huang, Q. Zhang, and Y. Wang, "Bilevel optimization via collaborations among lower-level optimization tasks," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2022.

[8] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genetic programming and evolvable machines*, vol. 8, pp. 131–162, 2007.

[9] N. Cheney, J. Bongard, V. SunSpiral, and H. Lipson, "Scalable co-optimization of morphology and control in embodied machines," *Journal of The Royal Society Interface*, vol. 15, no. 143, p. 20170937, 2018.

[10] G. Lan, M. De Carlo, F. van Diggelen, J. M. Tomczak, D. M. Roijers, and A. Eiben, "Learning directed locomotion in modular robots with evolvable morphologies," *Applied Soft Computing*, vol. 111, p. 107688, 2021.

[11] J. Bhatia, H. Jackson, Y. Tian, J. Xu, and W. Matusik, "Evolution gym: A large-scale benchmark for evolving soft robots," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2201–2214, 2021.

[12] F. Pigozzi, Y. Tang, E. Medvet, and D. Ha, "Evolving modular soft robots without explicit inter-module communication using local self-attention," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 148–157, 2022.

[13] J. Whitman, M. Travers, and H. Choset, "Learning modular robot control policies," *arXiv preprint arXiv:2105.10049*, 2021.

[14] D. Howard, A. E. Eiben, D. F. Kennedy, J.-B. Mouret, P. Valencia, and D. Winkler, "Evolving embodied intelligence from materials to machines," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 12–19, 2019.

[15] S. G. R. Prabhu, R. C. Seals, P. J. Kyberd, and J. C. Wetherall, "A survey on evolutionary-aided design in robotics," *Robotica*, vol. 36, no. 12, pp. 1804–1821, 2018.

[16] N. Bredeche, E. Haasdijk, and A. Prieto, "Embodied evolution in collective robotics: a review," *Frontiers in Robotics and AI*, vol. 5, p. 12, 2018.

[17] E. Medvet, A. Bartoli, F. Pigozzi, and M. Rochelli, "Biodiversity in evolved voxel-based soft robots," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 129–137, 2021.

[18] Z. Wang, B. Benes, A. H. Qureshi, and C. Mousas, "Co-design of embodied neural intelligence via constrained evolution," *arXiv preprint arXiv:2205.10688*, 2022.

[19] T. Wang, Y. Zhou, S. Fidler, and J. Ba, "Neural graph evolution: Towards efficient automatic robot design," *arXiv preprint arXiv:1906.05370*, 2019.

[20] R. J. Alattas, S. Patel, and T. M. Sobh, "Evolutionary modular robotics: Survey and analysis," *Journal of Intelligent & Robotic Systems*, vol. 95, pp. 815–828, 2019.

[21] D. Bruder, A. Sedal, R. Vasudevan, and C. D. Remy, "Force generation by parallel combinations of fiber-reinforced fluid-driven actuators," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3999–4006, 2018.

[22] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Data-driven control of soft robots using koopman operator theory," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 948–961, 2021.

[23] Y. Pan, P. Du, H. Xue, and H.-K. Lam, "Singularity-free fixed-time fuzzy control for robotic systems with user-defined performance," *IEEE Transactions on Fuzzy Systems*, vol. 29, no. 8, pp. 2388–2398, 2020.

[24] D. Bruder, B. Gillespie, C. D. Remy, and R. Vasudevan, "Modeling and control of soft robots using the koopman operator and model predictive control," *arXiv preprint arXiv:1902.02827*, 2019.

[25] G. Bravo-Palacios, A. Del Prete, and P. M. Wensing, "One robot for many tasks: Versatile co-design through stochastic programming," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1680–1687, 2020.

[26] G. Bledt and S. Kim, "Extracting legged locomotion heuristics with regularized predictive control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 406–412, 2020.

[27] J. Whitman, R. Bhirangi, M. Travers, and H. Choset, "Modular robot design synthesis with deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 10418–10425, 2020.

[28] F. Corucci, N. Cheney, F. Giorgio-Serchi, J. Bongard, and C. Laschi, "Evolving soft locomotion in aquatic and terrestrial environments: effects of material properties and environmental transitions," *Soft robotics*, vol. 5, no. 4, pp. 475–495, 2018.

[29] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017.

[30] C. Schaff, A. Sedal, and M. R. Walter, "Soft robots learn to crawl: Jointly optimizing design and control with sim-to-real transfer," *arXiv preprint arXiv:2202.04575*, 2022.

[31] X. Liu, D. Pathak, and K. M. Kitani, "Revolver: Continuous evolutionary models for robot-to-robot policy transfer," *arXiv preprint arXiv:2202.05244*, 2022.

[32] H. Wang, Y. Jin, and J. O. Jansen, "Data-driven surrogate-assisted multiobjective evolutionary optimization of a trauma system," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 6, pp. 939–952, 2016.

[33] M. Cui, L. Li, M. Zhou, and A. Abusorrah, "Surrogate-assisted autoencoder-embedded evolutionary optimization algorithm to solve high-dimensional expensive problems," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 4, pp. 676–689, 2022.

[34] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 350–364, 2020.

[35] S. Liu, H. Wang, W. Peng, and W. Yao, "A surrogate-assisted evolutionary feature selection algorithm with parallel random grouping for high-dimensional classification," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 5, pp. 1087–1101, 2022.

[36] Z. Song, H. Wang, C. He, and Y. Jin, "A kriging-assisted two-archive evolutionary algorithm for expensive many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 6, pp. 1013–1027, 2021.

[37] X. Ji, Y. Zhang, D. Gong, and X. Sun, "Dual-surrogate-assisted cooperative particle swarm optimization for expensive multimodal problems," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 794–808, 2021.

[38] H. Wang and Y. Jin, "A random forest-assisted evolutionary algorithm for data-driven constrained multiobjective combinatorial optimization of trauma systems," *IEEE Transactions on Cybernetics*, vol. 50, no. 2, pp. 536–549, 2020.

[39] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[40] L. Chen, H.-L. Liu, K. Li, and K. C. Tan, "Evolutionary bi-level optimization via multi-objective transformation-based lower level search," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2023.

[41] R. Said, M. Elarbi, S. Bechikh, C. A. C. Coello, and L. B. Said, "Discretization-based feature selection as a bi-level optimization problem," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2022.

[42] E. Medvet, A. Bartoli, A. De Lorenzo, and S. Seriani, "2d-vsr-sim: A simulation tool for the optimization of 2-d voxel-based soft robots," *SoftwareX*, vol. 12, p. 100573, 2020.

[43] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson, "Physically based deformable models in computer graphics," in *Computer graphics forum*, vol. 25, pp. 809–836, Wiley Online Library, 2006.