Swarm and Evolutionary Computation 68 (2022) 100974

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Evolutionary reinforcement learning via cooperative coevolutionary negatively correlated search

Peng Yang [a], Hu Zhang [b,*], Yanglong Yu [a], Mingjia Li [a], Ke Tang [a]

[a] Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China

[b] Science and Technology on Complex System Control and Intelligent Agent Cooperation Laboratory, Beijing Electro-mechanical Engineering Institute, Beijing 100074, China

ARTICLE INFO

Keywords:
Evolutionary algorithms
Deep reinforcement learning
Cooperative coevolution

ABSTRACT

Evolutionary algorithms (EAs) have been successfully applied to optimize the policies for Reinforcement Learning (RL) tasks due to their exploration ability. The recently proposed Negatively Correlated Search (NCS) provides a distinct parallel exploration search behavior and is expected to facilitate RL more effectively. Considering that the commonly adopted neural policies usually involves millions of parameters to be optimized, the direct application of NCS to RL may face a great challenge of the large-scale search space. To address this issue, this paper presents an NCS-friendly Cooperative Coevolution (CC) framework to scale-up NCS while largely preserving its parallel exploration search behavior. The issue of traditional CC that can deteriorate NCS is also discussed. Empirical studies on 10 popular Atari games show that the proposed method can significantly outperform three state-of-the-art deep RL methods with 50% less computational time by effectively exploring a 1.7 million-dimensional search space.

1. Introduction

Reinforcement Learning (RL) is an attracting machine learning problem that actively learns the decision-making policy through environmental interactions with delayed and noisy rewards, without consuming labeled training data like traditional supervised learning [1]. In RL, an agent sequentially observes the environment state and takes an action to the environment suggested by the policy, a reward is then returned (from the environment) to the agent reflecting how it interacts with the environment. By iteratively conducting the above procedure, RL is targeted to learn the policy that can maximize the long-term reward [2].

For this purpose, the policy is expected to accurately react to the observed environment states. The arising Deep Reinforcement Learning (DRL) methods provide the possibility by employing deep neural networks as the policy [3]. DRL builds policy directly on the observed raw data, offering seamless interfaces to the environment, and facilitates the state-action decision-making with powerful non-linear inference ability for many successful real-world applications [4–6]. On the other hand, due to the non-uniform distribution of rewards, it is important to explore the action space to enlarge the coverage of the policy on the state space [7]. This requires to exploratively optimize the parameters of the policy, e.g., the connection weights of deep networks. Traditional gradient-based methods [8,9] are not always effective due to the lacks of exploration ability.

Recently, Evolutionary Algorithms (EAs) have been successfully applied to DRL by searching for the optimal policy in a population-based manner [10–12]. The population-based nature of EAs not only provides the urgent exploration ability to RL, but also offers other bonuses such as parallel acceleration [11], noisy-resistance [13], and compatibility of training non-differentiable policies (e.g., trees [14]). Due to those advantages, the resultant Evolutionary Reinforcement Learning (ERL) has drawn increasing research popularity from both academia and industries.

Actually, the balance between exploration and exploitation has been extensively studied in the EA community for decades [15]. Basically, it is widely acknowledged that the diversity of the population is an important indicator to guide the exploration. Rich volume of works have been proposed to effectively measure the diversity of the current population and adjust it accordingly to generate the next population. Until recently, Tang et al. [16] discusses that the diversity of the current population may not be closely related to the generation of the diversified next population, and proposes to directly measure the diversity of the next population. Based on this idea, the presented Negatively Correlated Search (NCS) is able to capture the on-going interactions between suc-

* Corresponding author.
E-mail address: jxzhanghu@126.com (H. Zhang).

https://doi.org/10.1016/j.swevo.2021.100974
Received 27 October 2020; Received in revised form 21 August 2021; Accepted 22 August 2021
Available online 7 September 2021

cessive iterations and introduce manageable diversity to the generation of the next population, successfully forming a fully parallel exploration search behavior where each individual in the population is managed to visit a different region of the search space with high enough objective quality. Due to its effectiveness, NCS has been successfully applied to several types of real-world applications [17–20]. This paper aims to apply NCS into ERL to offer a parallel exploration ability for policy training.

Note that training a deep neural network needs to optimize commonly millions or more connection weights. Such huge numbers of decision variables usually impose great challenges to EAs (e.g., NCS) [21]. Generally, to solve such large-scale problems, the Cooperative Coevolution (CC) framework is often beneficial for scaling up EAs by dividing the decision variables into multiple exclusive groups and addressing the smaller sub-problems (i.e., groups of decision variables) with EAs respectively [22]. One major difficulty of applying CC lies in the evaluation of partial solutions in each sub-problem with respect to the original objective function. Existing CC methods mostly complement all partial solutions in a sub-problem with the same complementary vector [22], and then evaluate the complemented solutions accordingly. This strategy is mostly effective for traditional EAs whose population tend to converge and the individuals in the population are somewhat similar at the later stage of the search. However, due to the parallel exploration behaviors, the population of NCS tend to be diverse. In this case, sharing the same complementary vector among all partial solutions may lead to significantly incorrect evaluations of some partial solutions, and thus fail the parallel exploration. To address this issue, a specified CC framework is devised to scale up NCS without compromising its exploration ability, resulting in the proposed Cooperative Coevolutionary Negatively Correlated Search (CCNCS).

CCNCS is applied to train a neural policy with 1.7 million connection weights on 10 popular Atari games to verify its effectiveness on RL problems. Empirical results show that CCNCS can outperform the state-of-the-art DRL methods (including both gradient-based and EA-based policy training methods) by scoring significantly higher scores within half less computational time. Furthermore, it has been shown that CCNCS can obtain scores in some games with very sparse reward distributions, while the state-of-the-art methods cannot break the tie. This should also be attributed to the well-preserved strong exploration ability while scaling up NCS with CC. The correctness of the proposed NCS-friendly CC is also verified by comparing the proposed CCNCS with the pure NCS and the CCNCS with traditional CC on 10 games. Furthermore, computational time costs comparions indicate that ERL is competitive to gradient-based DRL in terms of computational efficiency on CPUs. Lastly, the parameter sensitivity analysis show the robustness of the proposed CCNCS. As a result, this work can be regarded as an effective showcase of appling EAs to the large-scale optimization problem as well as the so called evolving neural networks.

The reminder of this paper is organized as follows. Section 2 briefly introduces the preliminaries of ERL, NCS, and CC. Section 3 details the proposed CCNCS method. Section 4 presents the empirical studies of CCNCS on Atari games. The conclusions are drawn in Section 5.

2. Preliminaries

This section briefly provides three aspects of preliminaries for the following sections.

2.1. Reinforcement learning

RL is a problem of learning a policy to make Markov decisions for maximizing the long-term rewards. In RL, the policy is learnt by iteratively interacting with the environment. At each time step, the agent obeserves the environment state and picks an action based on the suggestion from the policy, leading to a transition from origin state to the next state, then receives a reward as the feedback of the action to update the policy. The above steps keep going until terminated.

To maximize the expected cumulative discounted reward in long term, various RL methods have been proposed in the literature. Among them, the model-based methods [2,23,24] aim to first learn an accurate environment model, and then get the most helpful action-environment pair by looking up the environment model. Those methods usually have a solid mathematical support, while they can hardly scale-up well to the real-world cases, resulting in the environment model to be either inaccurate or insufficient. The value function based methods [2,4,25] strive to learn the value function which indicates the expectation of the accumulated rewards in the following interactions. Recently, the policy search based methods that adopt the deep neural networks as the policy model have drawn most research attentions due to their powerful performance [8,10]. The key problem for this type of methods turns into how to search for a deep neural network based policy in the policy parameters space. This process encounters three major problems. First, the search space of training the deep neural networks is highly large-scale and multi-modal; second, due to the Markov decision process nature of RL, the policy learning process is non-differentiable unless some derivable functions are specially designed (e.g., the critic function in A3C [8]); last, the delayed rewards may involve considerable noise. Evolutionary RL is a category of suitable methods for addressing the above difficulties in terms of their derivate-free, robust and parallel features. For more details of RL methods, please refer to Arulkumaran et al. [3].

2.2. Evolutionary reinforcement learning

ERL specifies the class of RL approaches that using EAs to directly optimize the parameters of the policy [10], e.g., deep neural networks in this paper, though can be trees as well[26]. The general flowchart of ERL can be seen in Fig. 1. Initially, each individual of an EA is represented as a vector of all the connection weights of the policy. The training phase is divided into multiple epochs. At each epoch, the agent starts from the beginning of a game and takes actions suggested by a policy model (i.e., an individual of EA) with respect to the environment states. After a game (i.e., an epoch) has been finished, the reward of the policy will be returned back to the agent as well as EA. Then EA takes the rewards of all candidate policies in the current population to generate the new population of policies for the next epoch. The above procedure repeats until the training budget runs out, and the best policy model at the final epoch will be output.

ERL enjoys multiple merits offered by EAs. First, the population-based search mechanism of EAs can be exploited to improve both the exploration and parallel acceleration of ERL [11]. Second, the search direction of EAs can be guided by the relative order of the individuals in terms of their objective qualities, which makes ERL less sensitive to the evaluation noise [13]. Lastly, EAs can be used to optimize non-differentiable models (e.g., trees [14]), thus ERL is not restricted to train neural policy.

Among the above merits, the exploration ability of EAs might be the most attracting feature for RL, as it has also been widely studied in RL [27], which allows a policy to discover the unknown knowledge about the environment to enrich its current knowledge, hopefully maximizing the long-term benefit. In the literature, Epsilon–Greedy [28] is a simple yet effective method to balance exploration and exploitation in a randomized way, where epsilon refers to the probability of choosing the exploration strategy during the training. Parameter/action space noise methods [29] introduce pre-defined noise into the generation of parameters of the policy, leading to a policy with more diverse behaviors. Gibbs sampling [30] aims to model the importance areas of the action space and sample new actions therefrom, so as to uncover new promising areas of the action space for improving the exploration. Curiosity-driven exploration methods [31] formulate the error of a policy as an external objective of curiosity to enable the exploration, which shows to be effective in the environment with sparse rewards. The novelty

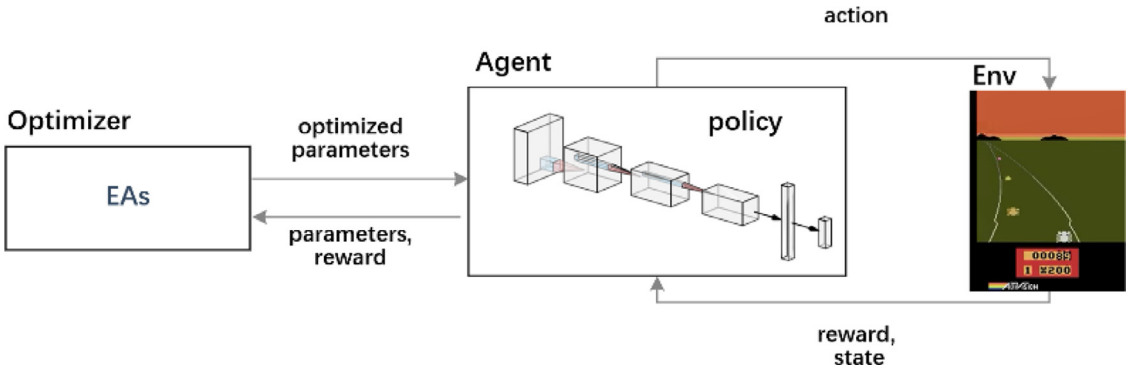

Fig. 1. The flowchart of evolutionary reinforcement learning.

search methods [32,33] from the derivate-free optimization community explicitly construct a new objective as the "novelty" to drive the search of the parameters to be diverse, while the original objective is seldom used. Another advantage of EAs might be their power in solving multi-objective optimization problems, which may emerge from many aspects of RL problems [34], such as multi-objective regularization of the policies [35,36].

Although those methods have shown effectiveness to the improvement of the exploration for RL, they still face great challenges in dealing with large-scale search space for optimizing the parameters of a policy.

2.3. Negatively correlated search

NCS is a recently proposed EA that features in its parallel exploration behavior [16,38]. Technically, NCS regards the population of λ individuals as λ exclusive search processes. Each search process is evolved with a traditional EA to maximize the objective quality of each individual. Meanwhile, the diversity among different search processes are explicitly measured and maximized to improve the exploration ability. Different from most existing EAs that measure the diversity as the differences among the individuals of the current population, NCS measures the diversity as the differences among individuals of the next population. By maximizing the diversity of the unsampled next population, it is straightforward to diversify the search processes in a controllable manner. As a result, each search process can visit a different region of the search space while preserving high objective quality, finally forming a parallel exploration search behavior (see Fig. 2 for the illustration of parallel exploration search behavior).

Take the NCS-C [16] (an instantiation of the NCS framework) as an example. The pseudo code of NCS-C is given in Algorithm 1 for illustration. At each iteration, each ith search process preserves a parent individual $\mathbf{x}_i$, and generates an offspring individual $\mathbf{x}_i'$ from the Gaussian distribution $\mathcal{N}(\mathbf{x}_i, \mathbf{\Sigma}_i)$. The covariance matrix $\mathbf{\Sigma}_i$ indicates the search step size of the evolution of $\mathbf{x}_i$. For simplicity, NCS-C ignores the inter-dependencies among variables. Thus, the covariance matrix is actually a diagonal matrix and is further simplified as a vector with the same dimensionality with $\mathbf{x}_i$. In practice, each dimension of $\mathbf{\Sigma}_i$ is usually initialized to the same value that is related to the search space range. The Gaussian distribution basically describes how the ith search process will generate the next individual. Thus, if the Gaussian distributions of different search processes can be explicitly forced to be different, they are highly likely to generate diversified individuals. To realize this idea, three steps are conducted in NCS-C. First, the difference among distributions is measured by the Bhattacharyya distance (Eqs. (1) and (2)), which considers the differences of both mean vectors and covariance matrices and is able to quantify the "overlap" of pairwise Gaussian distributions.

$$d(p_i) = \min_j \{-\ln\left(\int \sqrt{p_i(\mathbf{x})p_j(\mathbf{x})}\mathrm{d}\mathbf{x}\right) | j \neq i\} \tag{1}$$

$$d(p_i') = \min_j \{-\ln\left(\int \sqrt{p_i'(\mathbf{x})p_j(\mathbf{x})}\mathrm{d}\mathbf{x}\right) | j \neq i\} \tag{2}$$

Second, the diversity of the next population is modeled as the differences of all search processes and is maximized parallel with respect to each search process. Specifically, for each search process, either the parent distribution $p_i \sim \mathcal{N}(\mathbf{x}_i, \mathbf{\Sigma}_i)$ or the offspring distribution $p_i' \sim \mathcal{N}(\mathbf{x}_i', \mathbf{\Sigma}_i)$ is selected for the next iteration to maximize the whole diversity. Third, while selecting either the parent or offspring, their objective qualities are also considered (see steps 7 and 8 in Algorithm 1). Thus, a balance between the exploration (by maximizing the diversity) and the exploitation (by maximizing the objective quality) can be readily achieved by a trade-off parameter, which in turn is managed to control the distributions of search processes to be negatively correlated for parallel exploration. Lastly, the covariance matrix is updated based on the 1/5 successful rule (Eq. (3)), which increases the search step size by r times if the offspring successfully replaces the parent for c times over $epoch$ iterations, and decreases it otherwise. For more details of NCS, please refer to [16].

$$\sigma_i = \begin{cases} \sigma_i * r & \textbf{if } \frac{c}{epoch} > 0.2 \\ \frac{\sigma_i}{r} & \textbf{if } \frac{c}{epoch} < 0.2 \\ \sigma_i & \textbf{if } \frac{c}{epoch} = 0.2 \end{cases} \tag{3}$$

Algorithm 1 NCS-C.

1: **Initialize** λ solutions $\mathbf{x}_i$ (and their distributions $p_i \sim \mathcal{N}(\mathbf{x}_i, \mathbf{\Sigma}_i)$) randomly, $i = 1, \ldots, \lambda$.
2: **Evaluate** the λ solutions with respect to the objective function f.
3: **Repeat** until stop criteria are met:
4: **For** $i = 1$ **to** λ // Can be made in parallel
5: **Generate** an offspring solution $\mathbf{x}_i'$ from the distribution $p_i \sim \mathcal{N}(\mathbf{x}_i, \mathbf{\Sigma}_i)$.
6: **Calculate** $f(\mathbf{x}_i')$, $d(p_i)$ and $d(p_i')$, where $p_i' \sim \mathcal{N}(\mathbf{x}_i', \mathbf{\Sigma}_i)$
7: **If** $f(\mathbf{x}_i') + \varphi \cdot d(p_i') > f(\mathbf{x}_i) + \varphi \cdot d(p_i)$ // φ denotes the trade-off parameter
8: $\mathbf{x}_i = \mathbf{x}_i'$ and $p_i = p_i'$.
9: **Update** $\mathbf{\Sigma}_i$ according to the 1/5 successful rule.
10: **Output** the best solution ever found with respect to f.

2.4. Cooperative coevolution

EAs basically work by randomized sampling in the search space, which means their performance usually drops heavily when the search space enlarges significantly. To solve such large-scale problems with EAs, a commonly adopted method is CC [22]. CC follows the classic divide-and-conquer methodology that first divides the decision variables

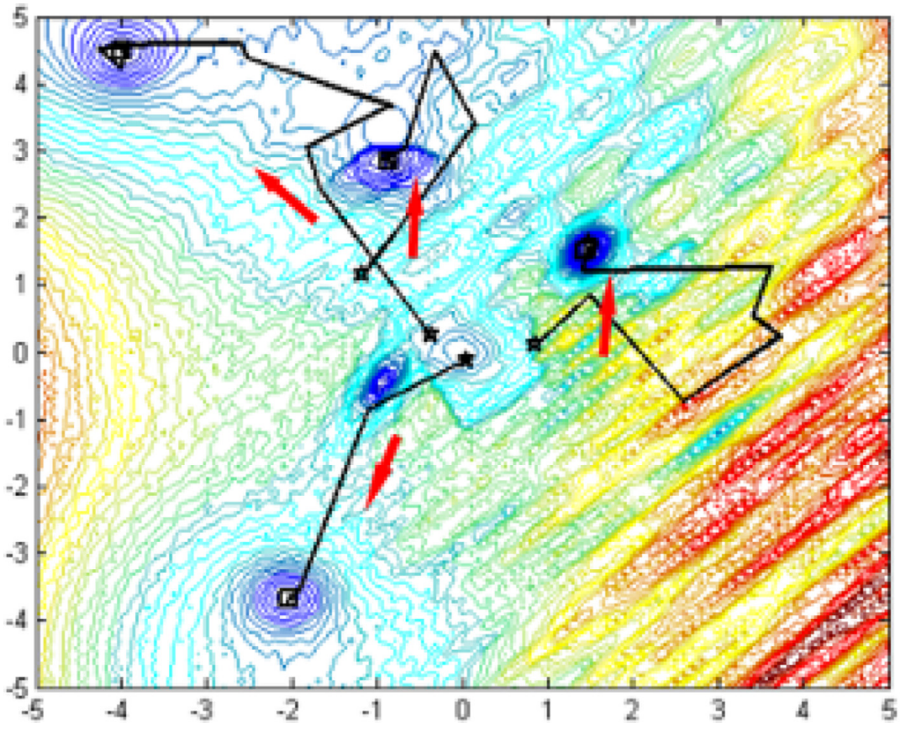

Fig. 2. The parallel exploration search pattern of NCS is illustrated on a 2-D multi-modal function, i.e., F19 in CEC'2005 continuous optimization functions benchmark [37], where 4 search processes (i.e., 4 individuals) explore different regions of the search space and successfully locate 4 optima.

into multiple small-scale groups, and optimizes each group separately with an EA. Ideally, each EA only needs to tackle a small-scale sub-problem while the original large-scale problem can still be effectively solved [22].

The major difficulty of applying the divide-and-conquer idea to EAs lies in how to evaluate the partial solutions in each sub-problem. To solve this problem, CC first complements the low-dimensional partial solutions to the dimensionality of the original objective function, and then evaluates them accordingly. However, different complementary vectors to a same partial solution can result in different objective qualities. Thus, rich amount of research efforts has been devoted to studying the complementing methods so as to improve the accuracy of the evaluation of partial solutions [39,40]. For example, Popovici et al. [41] comprehensively studied the complementing methods, categorizing them into individual-centric methods and population-centric methods. For the former cases, all the partial solutions are complemented with the same vector, which can be selected as the best performed or at random. Comparatively, the latter cases (such as shuffle-and-pair) conceptually allow the evaluation of all partial solutions with different complementary vectors, which, however, have not been frequently studied in nowadays CC field. There are also a plenty of approaches that have been proposed to the grouping of decision variables, aiming to make the sub-problems independent from each other so that even random complementary vectors can lead to accurate evaluations [42]. Also note that other methods that do not need to complement partial solutions also exist [21,43].

Despite of all the technical details, most CC methods basically complement all the partial solutions in a sub-problem with the same complementary vector [22], i.e., the individual-centric method. This is reasonable in many cases since keeping the complementary vector the same for all partial solutions can largely reduce the noise of calculating their relative order that guides the search direction of many EAs. However, it will be shown in the next section that such strategy does not work well for NCS due to its parallel exploration search behavior. For illustration, the framework of classic CC is briefly given in Algorithm 2.

Algorithm 2 Classic cooperative coevolution.

1: **Initialize** λ solutions $\mathbf{x}_i$ randomly, $i = 1, \ldots, \lambda$.
2: **Repeat** until stop criteria are met:
3: **Divide** the D-dimensional problem into M sub-problems.
4: **For** $j = 1$ **to** M
5: **Evolve** all the partial solutions $\boldsymbol{x}_{i,j}$ for one iteration with an EA, $i = 1, \ldots, \lambda$.
6: **For** $i = 1$ **to** λ
7: **Complement** each partial solution $\boldsymbol{x}_{i,j}$ with the same complementary vector $\boldsymbol{v}_j$ as $[\boldsymbol{x}_{i,j}; \boldsymbol{v}_j]$.
8: **Evaluate** each complemented solution $f[\boldsymbol{x}_{i,j}; \boldsymbol{v}_j]$ as the qualities of $\boldsymbol{x}_{i,j}$.
9: **Output** the best complemented solution ever found with respect to f.

3. Cooperative coevolutionary negatively correlated search

This section first discusses that the traditional CC framework is not suitable for NCS, and an NCS-friendly CC framework is proposed accordingly. The resultant CCNCS by incorporating NCS into the variant CC framework is also described in detail.

3.1. The NCS-friendly CC framework

As mentioned above, traditional CC complements all the partial solutions of a sub-problem with the same complementary vector and then evaluates the complemented solutions with respect to the objective function. This is useful for most EAs as their partial solutions in each sub-problem often tend to be similar at the convergence stage of the search, where the population highly likely locates at the same sub-space of a basin of attraction (i.e., they will converge to the same optimum). In this case, the differences of the evaluation qualities among all partial solutions can be made marginal by complementing them with the same complementary vector. By "marginal", we mean this optimum will be found by this population very likely, no matter by which specific individual.

Distinctly, NCS explicitly asks each search process to be negatively correlated, where the partial solutions in each sub-problem tend to be very different from each other at the later stage of the search and the population may locate multiple different basins of attraction. In this case, complementing all the partial solutions with the same complemen-

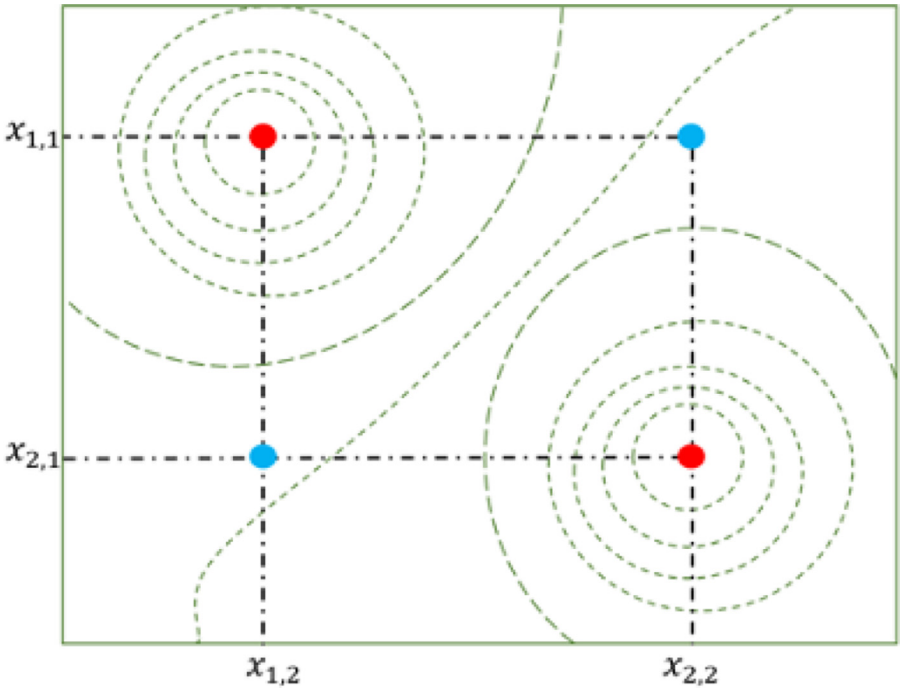

Fig. 3. Traditional CC that complements two partial solutions of a sub-problem, i.e., $x_{1,1}$ and $x_{2,1}$, with the same vector, i.e., either $x_{1,2}$ or $x_{2,2}$, is not suitable for NCS alike search methods as either optimum (marked as red solid circles) will be missed.

tary vector is problematic. For example, Fig. 3 illustrates a 2-D search space with two optima located at the left-upper corner and the right-lower corner. Suppose NCS has obtained two partial solutions in the first sub-problem, denoted as $x_{1,1}$ and $x_{2,1}$, respectively. And the concatenation of them is denoted as $[x_{2,1}; x_{1,2}]$. If they are complemented with $x_{1,2}$, it is clear that $f([x_{1,1}; x_{1,2}])$ is better than $f([x_{2,1}; x_{1,2}])$ and the left-upper optimum will be found. If they are complemented with $x_{2,2}$, the right-lower optimum will be found. In either case, the traditional CC will deteriorate the parallel exploration ability of NCS as either of the two optima will be missed out. As the iterations go on, the population of NCS will finally get converged to the partial solution who best fits the complementary vector, and thus fails to form the featured parallel exploration. One can also complement the partial solutions with multiple complementary vectors (e.g., both $x_{1,2}$ and $x_{2,2}$ in Fig. 3) to fit each promising partial solution with redundancy. However, it is computationally expensive as multiple times of objective function evaluations will be consumed.

In this paper, we propose to complement each partial solution of NCS with a different complementary vector to maximally preserve the parallel exploration search behavior. The only difference between the NCS-friendly CC and traditional CC lies in the step 7 of Algorithm 2. Specifically, in each jth sub-problem, each ith partial solution $\boldsymbol{x}_{i,j}$ will be complemented with a distinct complementary vector $\boldsymbol{v}_{i,j}$ in the new CC framework, rather than the same complementary vector $\boldsymbol{v}_j$ as for traditional CC. For the context of the ith complementary vector $\boldsymbol{v}_{i,j}$, we simply set it as $\boldsymbol{v}_{i,j} = \boldsymbol{x}_i / \boldsymbol{x}_{i,j}$. That is, each ith complementary vector in the jth sub-problem is calculated as the quotient set of $\boldsymbol{x}_i$ with respect to $\boldsymbol{x}_{i,j}$, and is composed of the current ith partial solution in other sub-problems. Under this setting, from the perspective of NCS, $\boldsymbol{x}_{i,j}$ and $\boldsymbol{v}_{i,j}$ are evolved together over iterations as $[\boldsymbol{x}_{i,j}; \boldsymbol{v}_{i,j}] = \boldsymbol{x}_i$. This means that $\boldsymbol{v}_{i,j}$ should be a suitable complementary vector of $\boldsymbol{x}_{i,j}$ heuristically. Thus, given NCS works by evolving each individual separately, CCNCS actually works by sequentially optimizing each sub-problem of each individual (see steps 8–11 of Algorithm 2 that $\boldsymbol{x}_{i,j}$ and $\boldsymbol{x}'_{i,j}$ are competed for survival with the same complementary vector $\boldsymbol{v}_{i,j}$). In other words, in the proposed CCNCS, CC is used to enhance the performance of the evolution of each $\boldsymbol{x}_i$ under the NCS framework. Therefore, $\boldsymbol{x}_{i,j}$ and $\boldsymbol{v}_{i,j}$ are practically recombined if they have the same indices of i and j. This strategy actually exploits the parallel exploration feature itself that different partial solutions will locate the sub-space of different basins of attraction. This helps to avoid the situation in Fig. 3 where one complementary vector cannot fit diversified promising partial solutions. Also note this strategy does not cost extra computational objective function evaluations as each partial solution is still evaluated once.

3.2. The details of CCNCS

In this section, NCS-C is incorporated into the variant CC framework to obtain the proposed CCNCS and the pseudo code is illustrated in Algorithm 3.

Algorithm 3 CCNCS.

1: **Initialize** λ solutions $\mathbf{x}_i$ randomly, $i = 1, \ldots, \lambda$.
2: **Repeat** until stop criteria are met:
3: **Divide** the D-dimensional problem into M sub-problems.
4: **For** $j = 1$ **to** M
5: **For** $i = 1$ **to** λ
6: **Generate** an offspring solution $\boldsymbol{x}'_{i,j}$ from the distribution $p_{i,j} \sim \mathcal{N}(\boldsymbol{x}_{i,j}, \boldsymbol{\Sigma}_{i,j})$.
7: **Calculate** the diversity of both parent and offspring as $d(p_{i,j})$ and $d(p'_{i,j})$, where $p'_{i,j} \sim \mathcal{N}(\boldsymbol{x}'_{i,j}, \boldsymbol{\Sigma}_{i,j})$.
8: **Complement** $\boldsymbol{x}_{i,j}$ and $\boldsymbol{x}'_{i,j}$ as $[\boldsymbol{x}_{i,j}; \boldsymbol{v}_{i,j}]$ and $[\boldsymbol{x}'_{i,j}; \boldsymbol{v}_{i,j}]$, respectively.
9: **Evaluate** $f([\boldsymbol{x}_{i,j}; \boldsymbol{v}_{i,j}])$ and $f([\boldsymbol{x}'_{i,j}; \boldsymbol{v}_{i,j}])$ as the qualities of $\boldsymbol{x}_{i,j}$ and $\boldsymbol{x}'_{i,j}$.
10: **If** $f([\boldsymbol{x}'_{i,j}; \boldsymbol{v}_{i,j}]) + \varphi \cdot d(p'_{i,j}) > f([\boldsymbol{x}_{i,j}; \boldsymbol{v}_{i,j}]) + \varphi \cdot d(p_{i,j})$
11: $\mathbf{x}_{i,j} = \mathbf{x}'_{i,j}$ and $p_{i,j} = p'_{i,j}$.
12: **Update** $\boldsymbol{\Sigma}_{i,j}$ according to the 1/5 successful rule in NCS-C.
13: **Output** the best complemented solution ever found with respect to f.

CCNCS first randomly initializes λ individuals (i.e., search processes) in the original D-dimensional search space. At each iteration of the main loop, the large-scale D decision variables are first decomposed in to M groups. Basically, decomposing the decision variables is to minimize

the interdependencies among them so that different groups of variables can be optimized independently. Intuitively, if it can be acquired how the decision variables interact with each other, the decomposition can made accurately [44]. However, in many cases, it is computational prohibited to learn the interdependencies among decision variables. Therefore, an alternative way for minimizing the interdependencies is to randomly group the variables. That is, the decision variables are grouped randomly, and the interdependencies among the groups are at random. As shown by Yang et al. [45], the more frequently random grouping is conducted, the lower the mathematical expectation of interdependencies among variables will be.

For simplicity, the random grouping with varied values of M is employed [45]. To be specific, when executing the step 3, the value of M is first randomly selected from a fixed pool with candidate values, and the D decision variables are randomly divided into M groups. Then NCS is run for optimizing each sub-problem (steps 5–12). As NCS conducts λ search processes in parallel, the detailed search operators can be executed with respect to each partial solution. Specifically, for the ith partial solution in the jth sub-problem $\boldsymbol{x}_{i,j}$, its offspring $\boldsymbol{x}'_{i,j}$ is first generated from the distribution $p_{i,j} \sim \mathcal{N}(\boldsymbol{x}_{i,j}, \boldsymbol{\Sigma}_{i,j})$. Then the Bhattacharyya distance between the parent distribution $p_{i,j} \sim \mathcal{N}(\boldsymbol{x}_{i,j}, \boldsymbol{\Sigma}_{i,j})$ and the distributions of other search processes $p_{k,j}$ is calculated and denoted as $d(p_{i,j})$ in Eq. (4), where $k = 1, \ldots, \lambda$ and $k \neq i$. Similarly, the Bhattacharyya distance of the offspring distribution $p'_{i,j} \sim \mathcal{N}(\boldsymbol{x}'_{i,j}, \boldsymbol{\Sigma}_{i,j})$ to the distributions of other search processes is calculated and denoted as $d(p'_{i,j})$ in Eq. (5).

$$d(p_{i,j}) = \min_j\{-\ln\left(\int \sqrt{p_{i,j}(\mathbf{x})p_{k,j}(\mathbf{x})}\mathrm{d}\mathbf{x}\right)|k \neq i\} \tag{4}$$

$$d(p'_{i,j}) = \min_j\{-\ln\left(\int \sqrt{p'_{i,j}(\mathbf{x})p_{k,j}(\mathbf{x})}\mathrm{d}\mathbf{x}\right)|k \neq i\} \tag{5}$$

To measure the qualities of $\boldsymbol{x}_{i,j}$ and $\boldsymbol{x}'_{i,j}$, they are first respectively complemented as $[\boldsymbol{x}_{i,j}, \boldsymbol{v}_{i,j}]$ and $[\boldsymbol{x}'_{i,j}, \boldsymbol{v}_{i,j}]$, and evaluated with respect to the objective function f. In fact, $f([\boldsymbol{x}_{i,j}, \boldsymbol{v}_{i,j}])$ can be obtained from the previous iteration, thus it will not cost function evaluations except for the first iteration. Then by considering both the objective qualities and the diversity, i.e., $f([\boldsymbol{x}'_{i,j}, \boldsymbol{v}_{i,j}]) + \varphi \cdot d(p'_{i,j})$ and $f([\boldsymbol{x}_{i,j}, \boldsymbol{v}_{i,j}]) + \varphi \cdot d(p_{i,j})$, either the parent partial solution or its offspring is selected to the next iteration. Accordingly, the distribution of either the parent or the offspring should also be passed to the next iteration as it describes the search behavior of the search process. After that, the distribution should be adjusted based on the feedback of the search process. Basically, the distribution mainly concerns the mean vector and covariance matrix. Between them, the mean vector is actually identical to the partial solution and should not be adjusted. The covariance matrix is adaptively adjusted via the 1/5 successful rule described in Eq. (3). After the main loop is terminated, the best complemented solution ever found will be output as the final solution of CCNCS.

4. Empirical studies

This section verifies the effectiveness of CCNCS on the popular RL problems, i.e., playing Atari Games. First, the Atari Games and the CCNCS-based ERL method are described. After that, the experimental protocol including the experimental settings and the compared state-of-the-art algorithms are briefly introduced. Finally, empirical results are analyzed to discuss the effectiveness of CCNCS.

4.1. CCNCS for playing Atari games

Atari 2600 is a set of video games that have been popular for benchmarking RL approaches [10,46,47]. Atari games successfully cover different types of difficult tasks, such as obstacle avoidance (e.g., Freeway and Enduro), shooting (Beamrider, and SpaceInvaders), maze (e.g., Alien and Venture), balls (e.g., Pong), Mario alike (e.g., Montezuma's Revenge), Sports (e.g. Bowling and DoubleDunk), and other types. Those types of games provide significantly different environmental settings and interaction ways for the agent, and thus are able to test the RL approaches with different tasks in terms of maximizing the long-term reward. Furthermore, the environment of Atari games can involve considerable randomness that the rewards may be noisy. For the underlying empirical studies, the mentioned-above 10 games are selected to assess the performance of CCNCS.

When playing Atari games, CCNCS fully follows the ERL flowchart depicted in Fig. 1. Specifically, each individual is represented as a vector of all connection weights of the neural policy model (i.e., one individual of CCNCS represents one candidate policy), and is evolved based on Algorithm 3. Each epoch of training is equivalent to each iteration of CCNCS, and one game playing with a candidate policy is actually one objective function evaluation of an individual of CCNCS. This means that the the reward is delayed for actions as it is only accumulated after a game is finished. As this empirical studies aim to show how different optimization methods perform on training a policy, rather than designing a sophisticated policy, we directly adopt the policy suggested in Mnih et al. [8] for simplicity, which is composed of three convolution layers and two full connection layers. The architecture of the network can be seen in Table 1, which involves nearly 1.7 million connection weights to be optimized. With the convolution layers, the policy is able to directly process the raw pixel data from the videos and no more efforts are needed for preprocessing the environmental states.

4.2. Experimental protocol

Three state-of-the-art DRL approaches are employed as the compared algorithms, i.e., A3C [8], PPO [9], and CES [10]. Among them, A3C and PPO are popular gradient-based methods that train the network with the traditional back-propagation. CES is a recently proposed ERL that utilizes the canonical Evolution Strategies to directly optimize the policy network. By comparing with those three methods, it suffices to assess how CCNCS performs against the state-of-the-arts. Besides, in order to show how CC facilitates NCS, we directly apply NCS to Atari games as the baseline. Lastly, to verify the effectiveness of the proposed NCS-friendly CC framework, we also compare the proposed CCNCS with the CCNCS under the traditional CC framework, denoted as CCNCS-t.

All the above five algorithms are targeted to train the same policy network with CCNCS. Each algorithm terminates the training phase in a game when the total budget runs out, and the best policy network at the final epoch will be returned for testing. The quality of the final policy is measured with the testing score, i.e., averaged score of 30 repeated games playing without the frame limitations. Given that the environment is randomly initialized with respect to a random number of "no-op" frames at the beginning of play (sampled over the interval 0 to 30), as suggested by Machado et al. [46], each run will be re-run for 10 times, i.e., each algorithm will obtain 10 testing scores on each game. And the performance of an algorihem is measured as the averaged quality over 10 repeated testing processes. The two-sided Wilcoxon rank-sum test at a 0.05 significance level has been conducted to see whether the averaged performance of two algorithms is statistically significantly different.

The total time budget is set as the total game frames that each training phase is allowed to consume. For the four ERL methods (i.e., CES, NCS, CCNCS-t, and CCNCS), the total game frames are set to 100 million. For A3C and PPO, as it works quite differently via back-propagation, it is unfair to set the same total game frames with the ERL methods. On this basis, we counted the game frames consumed by the well-established CES and A3C on the same CPU-cored hardware configurations with the same given computational run time in the same games. It has been found that the ratio of the consumed game frames between A3C and CES is about 2.5 within the same computational time budget. In other words, the back-propagation of policy parameters for gradient-based DRL consume 2.5 times more than the direct sampling of policy parameters for

Table 1
The network architecture of the policy.

	Input size	Output size	Kernel size	Stride	#filters	activation
Conv1	$4 \times 84 \times 84$	$32 \times 20 \times 20$	8×8	4	32	ReLU
Conv2	$32 \times 20 \times 204$	$64 \times 9 \times 9$	4×4	2	64	ReLU
Conv3	$64 \times 9 \times 9$	$64 \times 7 \times 7$	3×3	1	64	ReLU
Fc1	$64 \times 7 \times 7$	512	–	–	–	ReLU
Fc2	512	#Action	–	–	–	–

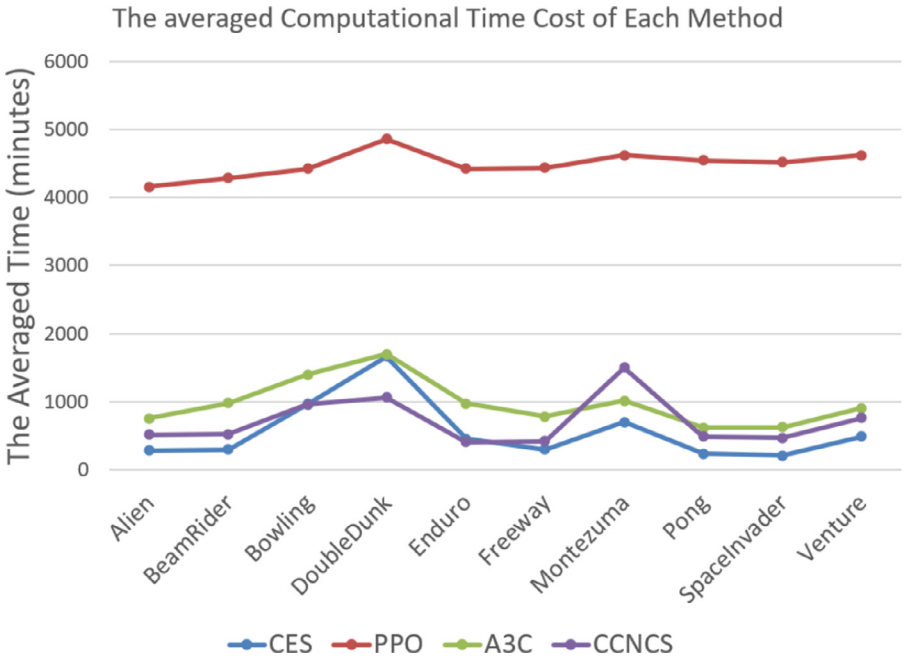

Fig. 4. The averaged computational time costs (in minutes) for CES, A3C, PPO, and CCNCS on 10 games. The game frame budget for A3C and PPO is set to 40 million, and for CES and CCNCS is set to 100 million. All algorithms are distributed on 16 CPU threads for each run.

ERLs on game frames. This is a reasonable ratio between the ERL and the gradient-based DRL on CPUs, and can be verified from Fig. 4. Though most gradient-based DRL methods can be accelerated on GPUs, the application scenarios (i.e., the environment simulators) may not easily fit GPUs well,[1] where the accelerations of GPUs can be heavily degenerated by the communications between simulators and policies. As a result, given the aboce comsumption ratio of game frames, the total game frames are set to 40 million for A3C and PPO for fairness. To discretize the games for agent's actions execution and states acquiring, the skipping frame is set to 4. In this case, in each training phase, the agent is allowed to take 25 million actions for derivate-free methods and 10 million actions for gradient-based methods under the totol budget of 100 million game frames.

For A3C, PPO, and CES, the hyper-parameters suggested in their original papers are directly employed for the comparisons and are listed in Table 2. For CCNCS, the population size λ is set to 6. The covariance of the Gaussian distribution is initialized as 0.2 for each dimensionality. The period of adaptively adjusting the covariance of the Gaussian distribution is set to 5. The adjusting ratio of the covariance of the Gaussian distribution is set to 1.2. The number of sub-problems M is randomly picked from the set of (2,3,4) at each iteration. The other hyper-parameters of CCNCS follow the original settings of NCS in Tang et al. [16]. To keep it simple, NCS and CCNCS-t share the same values of the corresponding hyper-parameters with the proposed CCNCS. For more details of fairly setting up the experimental protocol, please refer to [48].

[1] For all the empirical studies, the workstation with Intel(R) Xeon(R) CPU E7-4809 v4 @ 2.10 GHz (16Cores) is used.

Table 2
The major hyperparameter settings of A3C, PPO, and CES.

A3C [8]	PPO [9]	CES [10]
Async update = 5	Adam stepsize = 0.00025	Learning rate = 1
Actor num = 16	Actor num = 8	Sigma = 0.01
Gamma = 0.99	Gamma = 0.99	Parent size = 50
Learning rate = 0.0001	c1(loss weight) = 1	Pop size = 780
Entropy coefficient = 0.01	c2(loss weight) = 0.01	–
loss coefficient = 0.5	epsilon(clip) = 0.1	–
max grad normalization = 50	step num = 256	–

4.3. Results and analysis

Comparisons with well-established methods The best score, the worst score, the averaged score and the standard deviation of the 10 repeated testing scores of each algorithm on 10 games are shown in Table 3, where the best performance is marked in bold. The last row shows the statistics analysis of the two-sided Wilcoxon rank-sum test at a 0.05 significance level, where W-D-L indicates the number of games that the averaged score of CCNCS is statistically better, the same, or worse than that of the well-established methods. It can be seen that CCNCS generally performs the best among all compared algorithms. Specifically, for Alien, Beamrider, Freeway, Montezuma's Revenge, Pong, SpaceInvaders, Venture, Bowling and DoubleDunk, CCNCS can significantly outperform the 3 state-of-the-art methods within the same time budget. For Enduro, CCNCS can still perform better than A3C and CES while performs less satisfactorily than PPO. To further express the effectiveness of CCNCS, we run it with 50 million game frames which is 50% less than the time budget of the other algorithms. The results are listed in the rightmost column of Table 3. It is clearly seen that the advantages

Table 3
The testing scores of CCNCS and three well-established algorithms on 10 Atari games. The time budget indicates the total game frames can be used. The larger the scores in each run is, the better the algorithm performs. The last row shows the statistics analysis, where W-D-L indicates the number of games that the averaged score of CCNCS is statistically better, the same, or worse than that of the well-established methods.

Methods		CES [10]	PPO [9]	A3C [8]	CCNCS	CCNCS
Time Budget		0.1B	40M	40M	0.1B	0.05B
Alien	**Best**	653.7	813.7	1040.9	**2864.2**	1275.0
	Worst	416.0	301.0	277.4	**1203.8**	1190.0
	Average	561.8	638.2	766.0	**1818.3**	1242.4
	Std.	101.6	301.0	281.1	515.6	37.1
Beamrider	**Best**	508.2	657.4	884.0	**918.6**	903.2
	Worst	401.0	534.0	481.6	**772.4**	656.1
	Average	433.1	600.8	633.6	**864.0**	764.9
	Std.	37.8	47.4	62.2	61.7	58.1
Enduro	**Best**	8.4	**501.6**	0.0	63.7	53.1
	Worst	**6.2**	0.0	0.0	5.2	4.8
	Average	7.1	**186.4**	0.0	24.4	21.7
	Std.	1.1	221.0	0.0	26.8	24.2
Freeway	**Best**	17.2	22.0	0.0	**26.7**	26.3
	Worst	10.4	9.7	0.0	23.3	**23.5**
	Average	13.8	14.8	0.0	**24.8**	24.2
	Std.	4.8	4.4	0.0	1.6	1.5
Montezuma's	**Best**	0.0	0.0	0.0	**94.6**	68.1
	Worst	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
Revenge	**Average**	0.0	0.0	0.0	**27.5**	22.4
	Std.	0.0	0.0	0.0	33.1	30.8
Pong	**Best**	6.3	−19.8	−21.0	9.3	**12.9**
	Worst	−19.9	−20.0	−21.0	**−13.2**	−13.4
	Average	−6.4	−19.9	−21.0	**1.8**	−1.1
	Std.	8.5	0.1	0.0	6.4	7.0
SpaceInvaders	**Best**	679.1	622.5	1027.1	**1345.0**	1045.0
	Worst	275.0	399.2	421.6	**985.0**	900.0
	Average	433.8	488.1	626.0	**1131.6**	996.0
	Std.	212.0	199.6	291.7	101.5	75.0
Venture	**Best**	435.0	0.0	0.0	748.0	**791.8**
	Worst	200.0	0.0	0.0	**527.0**	522.8
	Average	343.0	0.0	0.0	618.0	**652.3**
	Std.	67.0	0.0	0.0	80.0	121.4
Bowling	**Best**	97.4	23.9	0.0	**112.8**	98.8
	Worst	20.0	22.2	0.0	**71.9**	47.0
	Average	56.8	23.4	0.0	**93.1**	77.8
	Std.	28.4	0.6	0.0	18.6	26.0
DoubleDunk	**Best**	−2.8	−3.0	−1.2	**0.0**	**0.0**
	Worst	−4.1	−3.8	−3.4	**−0.6**	**−0.6**
	Average	−3.4	−3.3	−2.2	**−0.2**	−0.3
	Std.	0.6	0.4	1.1	0.2	0.3
Statistics Analysis	**W-D-L**	9-1-0	8-1-1	9-1-0	–	–

still exist that CCNCS can gain as much as twice scores than the state-of-the-art methods in most of the tested games within half less time.

Among all the tested games, Montezuma's Revenge is almost the hardest one as traditional well-established methods (including the three compared ones) can hardly gain any score, unless human experience is incorporated to educate the policy [49] or the ensemble idea is introduced to multiple existing powerful policies [50]. This is mainly because the reward distribution of this game is very sparse that very limited sequence of actions can hit an effective reward. More intuitively, it can be explained as that the search space of Montezuma's Revenge appears to be very flat and there is very few information can be utilized to guide the search (i.e., to optimize the policy). On this basis, Montezuma's Revenge might be the most appropriate problem for assessing the exploration ability of RL methods. As CCNCS successufuly breaks the tie, the motivation of this paper that applying CCNCS to ERL can benefit the exploration ability is verified.

Performance analysis of CC component To verify the effectiveness of the proposed NCS-friendly CC, we compare the proposed CCNCS with the pure NCS (i.e., without CC) and the CCNCS-t (i.e, CCNCS with traditional CC). The same experimental setup with the first group of comparisons are empolyed here, and the empirical results as well as the statistical analysis are shown in Table 4. In general, CCNCS significantly outperforms NCS and CCNCS-t. Specifically, CCNCS is statistically better than NCS on all 10 games, and is superior to CCNCS-t on 7 games. On Alien, Montezuma's Revenge, and Venture, CCNCS almost achieves twice scores over NCS and CCNCS-t.

The discussions in Section 3.1 that traditional CC does not fit NCS well is intuitively verified in this group of comparisons. The discussions can be briefly summarized as that traditional CC will make NCS more greedy and thus may degenerate the effectiveness of searching different promising areas at the same time. By greedy, we can image that CCNCS-t is less explorative and its performance may be less varied. This can be observed on several games in Table 4. Specifically, on Enduro, CCNCS achieves the higher best score than CCNCS-t, but is much more varied. The similar situation can also be seen in Pong. In other games like Alien, Freeway, and Montezuma's Revenge, CCNCS-t is more stable than CCNCS according to the standard deviation, though the averaged performance is much worse than CCNCS. Interestingly, NCS can obtain scores on the very hard Montezuma's Revenge due to its parallel exploration ability [38], and CCNCS can amplify the advantage to score more with the proposed NCS-friendly CC. However, CCNCS-t cannot make any score, as the greedy strategy can hardly guide the search processes in the 'flat' search space. To summarize, the proposed NCS-friendly CC can indeed fit NCS better than traditional CC.

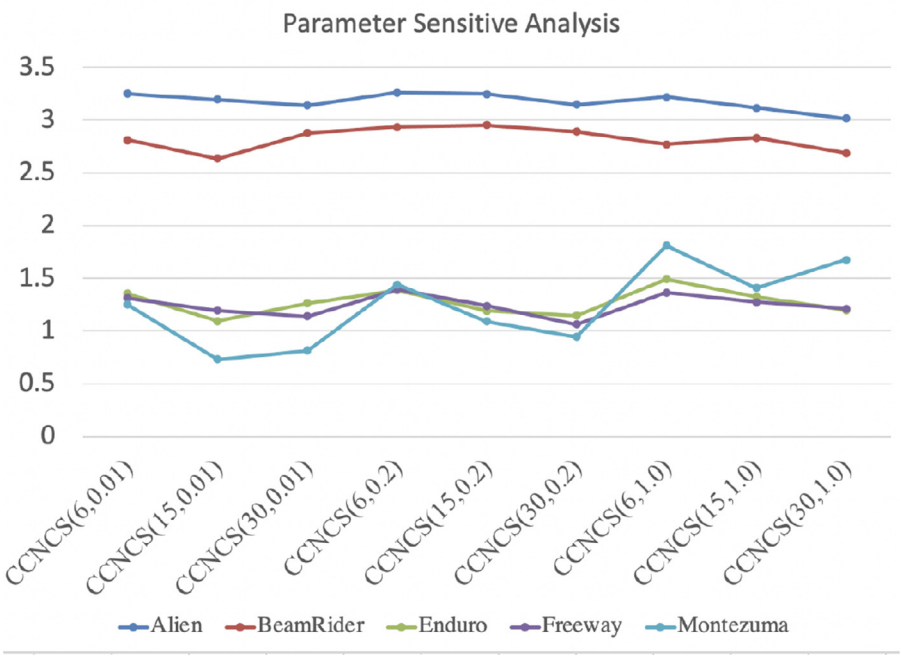

Fig. 5. The logarithm of the averaged performance of 9 variant CCNCS with different population size and initialized search step size are shown.

Table 4
The testing scores comparisons and the statistical analysis of NCS, CCNCS with traditional complementing strategy, and CCNCS with the proposed complementing strategy on 10 Atari games. The time budget is set to 0.1 billion game frames.

Methods		NCS	CCNCS-t	CCNCS
Alien	**Best**	1211.9	164.2	**2864.2**
	Worst	854.1	53.1	**1203.8**
	Average	1043.4	87.4	**1818.3**
	Std.	182.3	61.3	515.6
Beamrider	**Best**	782.6	808.5	**918.6**
	Worst	620.5	427.7	**772.4**
	Average	703.8	633.1	**864.0**
	Std.	79.4	196.8	61.7
Enduro	**Best**	12.4	40.9	**63.7**
	Worst	6.7	**35.8**	5.2
	Average	7.9	**37.7**	24.4
	Std.	2.1	2.2	26.8
Freeway	**Best**	19.7	23.79	**26.7**
	Worst	13.0	22.5	**23.3**
	Average	16.2	23.1	**24.8**
	Std.	3.6	0.4	1.6
Montezuma's	**Best**	10.3	0.0	**94.6**
	Worst	**0.0**	**0.0**	**0.0**
Revenge	**Average**	3.1	0.0	**27.5**
	Std.	5.8	0.0	33.1
Pong	**Best**	−9.1	**19.1**	9.3
	Worst	−9.9	**18.0**	−13.2
	Average	−9.5	**18.5**	1.8
	Std.	0.4	0.4	6.4
SpaceInvaders	**Best**	1047.4	1070.5	**1345.0**
	Worst	800.0	428.4	**985.0**
	Average	967.1	821.1	**1131.6**
	Std.	103.5	261.7	101.5
Venture	**Best**	628.0	384.0	**748.0**
	Worst	418.0	0.0	**527.0**
	Average	522.0	266.3	**618.0**
	Std.	109.0	109.7	80.0
Bowling	**Best**	76.5	**161.5**	112.8
	Worst	53.6	63.8	**71.9**
	Average	64.7	**114.3**	93.1
	Std.	12.4	25.4	18.6
DoubleDunk	**Best**	−0.8	−0.9	**0.0**
	Worst	−1.7	−2.0	**−0.6**
	Average	−1.3	−1.5	**−0.2**
	Std.	0.4	0.4	0.2
Statistics Analysis	**W-D-L**	10-0-0	7-0-3	–

Computational time of compared algorithms Fig. 4 shows the averaged computational time costs (in minutes) of CES, A3C, PPO, and CCNCS on 10 games. Based on some prelimilary tests, it has been found that the ratio of the consumed game frames between A3C and CES is about 2.5 within the same computational time budget. Thus, the game frame budget for A3C and PPO is set to 40 million, and for CES and CCNCS is set to 100 million. All algorithms are distributed on 16 CPU threads for each run. From Fig. 4, it can be seen that the computational time costs of ERL is quite competitive to the gradient-based DRL methods on CPUs, as both CES and CCNCS consume similar computational time to A3C. PPO costs much more computational time than the other three methods. This might be that the executed implementations of PPO are not suitable for CPUs but specifically for GPUs. Nevertheless, as many simulators of RL environments are developed based on CPUs, it suffices to show that ERL is promising not only for its solution performance but also the computational efficiency.

Sensitivity analysis of CCNCS parameters We select the population size and the initialized search step size (i.e., the covariance matrix) for sensitive analysis, as they are the most important and commonly analyzed parameters of EAs. For the population size, we set it to three different values as 6, 15, 30. For the initialized covariance matrix, we set it to three different values as 0.01, 0.2, and 1.0. We combine different values of the two parameters to generate 9 variants of CCNCS. We select the first 5 games (i.e., Alien, Beamrider, Enduro, Freeway, and Montezuma's Revenge) for sensitive analysis. Each configuration runs on each game for 10 times, and the logarithm of the averaged performance is depicted in Fig. 5. It can be seen that CCNCS is generally not very sensitive to these two important parameters. Specifically, CCNCS variants with larger population sizes tend to perform worse as the number of search iterations become much fewer. Larger initialized search step sizes are helpful for exploration, so those variants perform better on Montezuma's Revenge. On the other hand, for some games, larger initialized search step can lead to slow converge to an optimum.

5. Conclusions

This paper studies how to effectively enable DRL approaches the exploration ability. By focusing on the emerging ERL techniques that employs EAs to optimize the parameters of the policies, this paper discusses that the recently presented NCS can be more appropriate to ERL due to its parallel exploration search behavior. To apply NCS to optimize mil-

lions of connection weights of neural policies, this paper employs the CC framework to scale up NCS for large-scale search space. The adopted CC framework is modified to specially fit NCS alike methods who have parallel exploration search behaviors. Extensive studies are conducted on 10 Atari games to verify the proposed method. Empirical results show that scaled-up NCS can perform more effectively than three state-of-the-art DRL methods with 50% less computational time, including both gradient-based ones and derivate-free ones. Furthermore, the powerful exploration ability enabled by NCS successfully helps the agent gain scores on the difficult Montezuma's Revenge game, where most existing methods cannot get scored due to the highly sparse reward distribution. Then, this paper empirically studies how the proposed NCS-friendly CC framework facilitates the NCS better than traditional CC. This paper lastly studies the computational time costs and the parameters sensitivety of the proposed CCNCS.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Peng Yang: Conceptualization, Supervision, Methodology, Writing – original draft, Writing – review & editing. **Hu Zhang:** Methodology, Data curation, Visualization, Investigation. **Yanglong Yu:** Software, Validation. **Mingjia Li:** Software, Validation. **Ke Tang:** Conceptualization, Writing – review & editing.

Acknowledgments

This work was supported by the Natural Science Foundation of China (Grant nos. 61806090, 61703382, and 61672478), Guangdong Provincial Key Laboratory (Grant no. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant no. 2017ZT07X386), and Shenzhen Science and Technology Program (Grant no. KQTD2016112514355531), and the Stable Support Plan Program of Shenzhen Natural Science Fund (Grant no. 20200925154942002).

References

[1] L. Zhang, K. Tang, X. Yao, Log-normality and skewness of estimated state/action values in reinforcement learning, in: Advances in Neural Information Processing Systems, Long Beach, USA, 2017, pp. 1804–1814.
[2] J. Oh, S. Singh, H. Lee, Value prediction network, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA, 2017, pp. 6120–6130. Long Beach, California, USA
[3] K. Arulkumaran, M.P. Deisenroth, M. Brundage, A.A. Bharath, Deep reinforcement learning: a brief survey, IEEE Signal Process. Mag. 34 (6) (2017) 26–38.
[4] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[5] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484.
[6] N.C. Luong, D.T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, D.I. Kim, Applications of deep reinforcement learning in communications and networking: a survey, IEEE Commun. Surv. Tutor. 21 (4) (2019) 3133–3174.
[7] L. Zhang, K. Tang, X. Yao, Efficient exploration is crucial to achieving good performance in reinforcement learning, Advances in Neural Information Processing Systems, Vancouver, Canada, 2019.
[8] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: Proceedings of 2016 International Conference on Machine Learning, New York, USA, 2016, pp. 1928–1937.
[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347(2017).
[10] P. Chrabaszcz, I. Loshchilov, F. Hutter, Back to basics: benchmarking canonical evolution strategies for playing Atari, in: International Joint Conference on Artificial Intelligence, Stockholm, 2018, pp. 1419–1426.
[11] T. Salimans, J. Ho, X. Chen, S. Sidor, I. Sutskever, Evolution strategies as a scalable alternative to reinforcement learning, arXiv preprint arXiv:1703.03864(2017).
[12] M.M. Drugan, Reinforcement learning versus evolutionary computation: a survey on hybrid algorithms, Swarm Evol. Comput. 44 (2019) 228–246.
[13] C. Qian, Y. Yu, K. Tang, Y. Jin, X. Yao, Z.-H. Zhou, On the effectiveness of sampling for evolutionary optimization in noisy environments, Evol. Comput. 26 (2) (2018) 237–267.
[14] Z. Zhou, J. Feng, Deep forest: towards an alternative to deep neural networks, in: Proceedings of 2017 International Joint Conference on Artificial Intelligence, Australia, 2017, pp. 3553–3559.
[15] M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: a survey, ACM Comput. Surv. (CSUR) 45 (3) (2013) 1–33.
[16] K. Tang, P. Yang, X. Yao, Negatively correlated search, IEEE J. Sel. Areas Commun. 34 (3) (2016) 542–550.
[17] G. Li, C. Qian, C. Jiang, X. Lu, K. Tang, Optimization based layer-wise magnitude-based pruning for DNN compression, in: Proceedings of 2018 International Joint Conference on Artificial Intelligence, Stockholm, 2018, pp. 2383–2389.
[18] Y. Lin, H. Liu, G. Xie, Y. Zhang, Time series forecasting by evolving deep belief network with negative correlation search, in: Proceedings of 2018 Chinese Automation Congress (CAC), IEEE, Shaanxi, China, pp. 3839–3843.
[19] D. Jiao, P. Yang, L. Fu, L. Ke, K. Tang, Optimal energy-delay scheduling for energy harvesting WSNs with interference channel via negatively correlated search, IEEE Internet Things J. 7 (3) (2019) 1690–1703.
[20] P. Yang, K. Tang, J.A. Lozano, X. Cao, Path planning for single unmanned aerial vehicle by separately evolving waypoints, IEEE Trans. Robot. 31 (5) (2015) 1130–1146.
[21] P. Yang, K. Tang, X. Yao, Turning high-dimensional optimization into computationally expensive optimization, IEEE Trans. Evol. Comput. 22 (1) (2017) 143–156.
[22] Z. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Inf. Sci. 178 (15) (2008) 2985–2999.
[23] D. Ha, J. Schmidhuber, Recurrent world models facilitate policy evolution, in: Advances in Neural Information Processing Systems, NIPS'18, Curran Associates Inc., Red Hook, NY, USA, 2018, pp. 2455–2467. Montréal, Canada
[24] S. Zhong, Q. Liu, Z. Zhang, Q. Fu, Efficient reinforcement learning in continuous state and action spaces with Dyna and policy approximation, Front. Comput. Sci. 13 (1) (2019) 106–126, doi:10.1007/s11704-017-6222-6.
[25] H.v. Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI'16, AAAI Press, Phoenix, Arizona, 2016, pp. 2094–2100, doi:10.5555/3016100.3016191.
[26] D.G. Wilson, S. Cussat-Blanc, H. Luga, J.F. Miller, Evolving simple programs for playing Atari games, in: Proceedings of the 2018 Genetic and Evolutionary Computation Conference, 2018, pp. 229–236.
[27] H. Tang, R. Houthooft, D. Foote, A. Stooke, O.X. Chen, Y. Duan, J. Schulman, F. DeTurck, P. Abbeel, Exploration: a study of count-based exploration for deep reinforcement learning, in: Advances in Neural Information Processing Systems, 2017, pp. 2753–2762.
[28] V. Raykar, P. Agrawal, Sequential crowdsourced labeling as an epsilon-greedy exploration in a Markov decision process, in: Proceedings of the 2014 Artificial Intelligence and Statistics, 2014, pp. 832–840.
[29] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R.Y. Chen, X. Chen, T. Asfour, P. Abbeel, M. Andrychowicz, Parameter space noise for exploration, CoRR abs/1706.01905(2017).
[30] C. Andrieu, N. De Freitas, A. Doucet, M.I. Jordan, An introduction to MCMC for machine learning, Mach. Learn. 50 (1-2) (2003) 5–43.
[31] D. Pathak, P. Agrawal, A.A. Efros, T. Darrell, Curiosity-driven exploration by self-supervised prediction, CoRR abs/1705.05363(2017).
[32] J. Lehman, K.O. Stanley, Abandoning objectives: evolution through the search for novelty alone, Evol. Comput. 19 (2) (2011) 189–223.
[33] E. Conti, V. Madhavan, F.P. Such, J. Lehman, K. Stanley, J. Clune, Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents, in: Advances in Neural Information Processing Systems, 2018, pp. 5027–5038.
[34] Y. Guo, H. Yang, M. Chen, J. Cheng, D. Gong, Ensemble prediction-based dynamic robust multi-objective optimization methods, Swarm Evol. Comput. 48 (2019) 156–171.
[35] M. Gong, X. Jiang, H. Li, Optimization methods for regularization-based ill-posed problems: a survey and a multi-objective framework, Front. Comput. Sci. 11 (3) (2017) 362.
[36] Y.-N. Guo, X. Zhang, D.-W. Gong, Z. Zhang, J.-J. Yang, Novel interactive preference-based multiobjective evolutionary optimization for bolt supporting networks, IEEE Trans. Evol. Comput. 24 (4) (2020) 750–764.
[37] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization, Technical Report, Nanyang Technological University (NTU), Singapore, 2005.
[38] P. Yang, Q. Yang, K. Tang, X. Yao, Parallel exploration via negatively correlated search, Front. Comput. Sci. (2021), doi:10.1007/s11704-020-0431-0.
[39] X. Ma, X. Li, Q. Zhang, K. Tang, Z. Liang, W. Xie, Z. Zhu, A survey on cooperative co-evolutionary algorithms, IEEE Trans. Evol. Comput. 23 (3) (2019) 421–441, doi:10.1109/TEVC.2018.2868770.
[40] L. Panait, Theoretical convergence guarantees for cooperative coevolutionary algorithms, Evol. Comput. 18 (4) (2010) 581–615.
[41] E. Popovici, A. Bucci, R.P. Wiegand, E.D. De Jong, Coevolutionary Principles, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 987–1033.
[42] M.N. Omidvar, X. Li, Y. Mei, X. Yao, Cooperative co-evolution with differential grouping for large scale optimization, IEEE Trans. Evol. Comput. 18 (3) (2013) 378–393.
[43] P. Yang, K. Tang, X. Yao, A parallel divide-and-conquer-based evolutionary al-

gorithm for large-scale optimization, IEEE Access 7 (2019) 163105–163118, doi:10.1109/ACCESS.2019.2938765.
[44] S. Mahdavi, M.E. Shiri, S. Rahnamayan, Metaheuristics in large-scale global continues optimization: a survey, Inf. Sci. 295 (2015) 407–428.
[45] Z. Yang, K. Tang, X. Yao, Multilevel cooperative coevolution for large scale optimization, in: Proceedings of 2008 IEEE Congress on Evolutionary Computation, IEEE, 2008, pp. 1663–1670.
[46] M.C. Machado, M.G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, M. Bowling, Revisiting the arcade learning environment: evaluation protocols and open problems for general agents, J. Artif. Intell. Res. 61 (2018) 523–562.
[47] M.G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The arcade learning environment: an evaluation platform for general agents, J. Artif. Intell. Res. 47 (2013) 253–279.
[48] E. Osaba, E. Villar-Rodriguez, J. Del Ser, A.J. Nebro, D. Molina, A. LaTorre, P.N. Suganthan, C.A. Coello Coello, F. Herrera, A tutorial on the design, experimentation and application of metaheuristic algorithms to real-world optimization problems, Swarm Evol. Comput. 64 (2021) 100888.
[49] Y. Aytar, T. Pfaff, D. Budden, T. Paine, Z. Wang, N. de Freitas, Playing hard exploration games by watching youtube, in: Advances in Neural Information Processing Systems, Montréal, CANADA, 2018, pp. 2930–2941.
[50] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, D. Silver, Rainbow: combining improvements in deep reinforcement learning, in: Proceedings of the 2018 AAAI Conference on Artificial Intelligence, 2018, pp. 3215–3222.