Reinforcement Learning-Based Differential Evolution for Global Optimization

Iztok Fister, Dušan Fister, and Iztok Fister Jr.

Abstract Reinforcement learning is a computational approach that mimics learning from interaction and supplements the existing supervised and unsupervised learning methods within the machine learning field. It bases on the mapping of a given situation to the action, and each action is evaluated by a reward. Of crucial concern, here is that the mapping is performed using suitable policies that correspond to a set of the so-called psychological stimulus-response rules (associations). However, in reinforcement learning, we are not interested in immediate rewards, but in a value function that specifies how good the rewards were in the long run. Reinforcement learning differential evolution is proposed in this study. On the one hand, a Q-learning algorithm capable of ensuring the good behavior of the evolutionary search process by explicit strategy exploration is engaged to collect the more prominent mutation strategies within an ensemble of strategies. On the other, the reinforcement learning mechanism selects among the strategies incorporated from the original L-SHADE algorithm using the 'DE/current-to-pbest/1/bin' mutation strategy toward the iL-SHADE to jSO using the 'DE/current-to-pbest-w/1/bin' mutation strategies. Testing the proposed RL-SHADE algorithm was conducted on the well-established function benchmark suites from the popular CEC special session/competition on real-parameter single-objective optimization during the last decade, where three different benchmark suites were issued. We expected that the results of the proposed RL-SHADE algorithm would outperform the results of the three original algorithms in solving all the observed benchmarks.

I. Fister · I. Fister Jr. (✉)
Faculty of Electrical Engineering and Computer Science, Koroška Ul. 43, 2000 Maribor, Slovenia
e-mail: iztok.fister1@um.si

I. Fister
e-mail: iztok.fister@um.si

D. Fister
Faculty of Economics and Business, Razlagova ul. 14, 2000 Maribor, Slovenia
e-mail: dusan.fister1@um.si

B. V. Kumar et al. (eds.), *Differential Evolution: From Theory to Practice*, Studies in Computational Intelligence 1009,
https://doi.org/10.1007/978-981-16-8082-3_3

1 Introduction

The classical artificial intelligence (AI), that was valid in the sixties of the last century, carried out the intelligence as a product of vast numbers of special purpose tricks, procedures, and heuristics [1]. Brooks [2], one of the main critics of the so-called strong AI, established that the intelligent behavior is not disembodied within computer systems, but it is the result of an interaction of a subject with the environment. Instead of strong methods, the modern AI stems in learning from interaction as a main idea underlying all theories of learning and intelligence. This approach is known under the name of weak AI.

Formally, reinforcement learning (RL) represents a computational approach to learning from interaction. Human environment interaction represents the basic idea of natural learning. Children, for instance, discover concepts about the real world through play that is nothing but an interaction with their environment. Indeed, RL looks for a solution of how to map a situation to an action in order to maximize the rewarding signal. Thus, the learner is not told which action to take, but to discover which actions yield the most reward by trying them [1].

RL belongs to the domain of machine learning (ML) and supplements the already existing supervised and unsupervised learning methods. Thus, supervised learning refers to learning from a set of labeled examples provided by a knowledgeable external supervisor, while the unsupervised learning is to find structures hidden in a collection of unlabeled data. Actually, RL combines the trial-and-error well-known method from the psychology of intelligence [3] with the delayed reward in order to maximize the value function.

Two issues characterized RL in the past: learning by trial-and-error and solving the problem of the optimal control value function and dynamic programming. Trial-and-error learning goes as far back as 1852 to Alexander Bain [4]. Minsky [5] influenced the AI community by connecting the trial-and-error learning with AI, including prediction, expectation, and the basic credit assignment mechanism. Holland et al. [6] connected trial-and-error learning in its non-associative form with genetic algorithms (GA). He introduced the classifier system that represents the true RL system, which uses the so-called bucket-brigade algorithm for credit assignment. The GA serves for evolving classification rules. Later, Klopf [7] revised the trial-and-error RL with AI. Barto and Sutton [8] showed that RL differs from the supervised learning. On the other hand, the term optimal control emerged in the late 1950s and refers to the problem of designing the controller for minimizing or maximizing a measure of a dynamical system behavior over time [1]. Bellman [9] proposed the class of methods for solving optimal control problems using dynamic programming. The same author also introduced discrete stochastic version of optimal control known as the Markovian decision process (MDP) [10]. Both methods represent essential elements of modern RL [1].

Recently, evolutionary algorithm's (EAs) have affected solving the optimization problems community dramatically [11]. They are stochastic population-based nature-inspired algorithms that explore the principle of trial-and-error. Their evolutionary process mimics the natural evolution founded by Darwin [12] during searching for a new solution in the search space. Differential evolution (DE) developed by Storn and Price [13] has gained a lot of attention due to its simplicity of use and the quality of results obtained especially by applying it to continuous optimization problems.

Two main processes enable the RL to work properly, i.e., exploitation and exploration. Although the processes of the same names can also be found in the EAs community [14], they have another meaning in the sense of RL. The exploitation in RL refers to already experienced actions in order to retain reward, while the exploration to make better action selection in the future [1]. The other differences from the EAs are as follows: EAs do not interact with the environment, and they evaluate rather the value the solutions in the immediate-term (i.e., reward) than in the long-term sense (i.e., value function). On the other hand, the RL works with agents having explicit goals that are capable of sensing their environment and can choose actions with which they influence their environment. These agents are typical components of the longer behavior system. In general, this means that they are not necessary robots acting in physical environments, but also software components working in programming environments.

The DE algorithm applies static policies (i.e., the so-called mutation strategies) for exploring the search space. Typically, these do not change during the lifetime of the evolutionary search process. A fitness landscape normally changes during the evolutionary runs. Therefore, the strategy used at the beginning does not explore the search space efficiently when the evolutionary process becomes matured. In line with this, the DE that uses ensemble strategies, capable of changing strategies according to feedback from the search process, has become popular in recent years [15, 16].

The present comparative study captured ten nature-inspired algorithms (also state-of-the-art ones) that achieved good results on the CEC special session/competition on real-parameter single-objective optimization by optimizing the benchmark function suites in the last decade. Indeed, there were three different suites published in the years 2013 [17], 2014 [18], and 2017 [19], since competitions in 2016 and 2018 were also organized using the benchmark suites published in years 2014 and 2018, respectively.

This chapter presents in some way a continuation of the already published paper by Fister et al. [20], where the impact of selecting the benchmark suites on the estimation of an algorithm's quality was examined. Authors concluded that the best algorithm, capable of overcoming the results of the rest of the algorithms in tests by solving all the benchmark suites, does not exist. In this chapter we go a step further by assuming that such an algorithm could be found when the RL method is incorporated within the DE algorithm.

The following algorithms were selected carefully for our study, where both families of nature-inspired algorithms were observed: Thus, the DE and its variants belong to the EA family, while the artificial bee colony (ABC) [21] and its self-adaptive variant SSEABC [22] to the swarm intelligence (SI)-based algorithm family. Among DE variants, the self-adapted versions of DE, like SaDE [23] and jDE [24], were taken into consideration, to which a family of success-history-based adaptive DE (SHADE) [25] were also included. The following improved state-of-the-art versions of SHADE-based algorithms were applied in our study: L-SHADE [26], iL-SHADE [27], jSO [28], and LSHADE-RSP [29].

The structure of the remainder of the chapter is as follows: Sect. 2 deals with the basic information needed for readers to understand the subjects that follow. The proposed RL algorithm incorporated within the L-SHADE is illustrated in Sect. 3. The experiments and obtained results are the subjects of Sect. 4. The chapter is concluded with Sect. 5, where the performed work is summarized and directions are outlined for the future.

2 Basic Information

The section is devoted to explain the basic information necessary for following the matters in the remainder of the paper. Actually, it is divided to two parts: The first part is dedicated for presenting the basic concepts of RL, while the second describes the characteristics of the particular nature-inspired algorithms used in our experimental study.

2.1 *Concepts of Reinforcement Learning*

An essential piece of the RL system is a complete, interactive, goal-seeking agent that is capable of sensing its environment, on the one hand, and taking actions, selected by its decision-making process, on the other. The majority of the RL problems can be formed as a MDP [30], where the agent makes the decision which actions A_t to perform in order to gain the maximum reward R_{t+1} and senses its environment to change its Internal state S_{t+1} accordingly (Fig. 1).

Let us notice that the agent is in the State S_0 at the beginning. MDP and the agent generate a sequence of states, actions, and rewards in the following form: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots$ Then, an expected return G_t is evaluated as:

$$G_t \doteq R_{t+1} + R_{t+2} + \cdots + R_T \doteq \sum_{k=t+1}^{T}, \qquad (1)$$

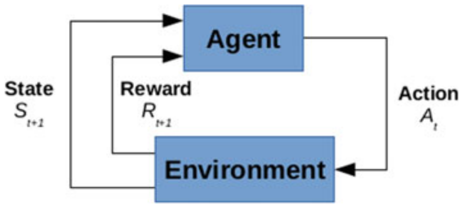


Fig. 1 Interaction of the agent with the environment in the Markovian decision process

where a sequence of environment interactions is also called episodes. Each episode is terminated by the Terminal state T.

Beside the agent embodied into the environment, the RL system consists of the following elements [1]:

- a policy,
- a reward signal,
- a value function,
- a model of the environment.

The policy is a mapping from current states of the environment to actions to be taken [1]. In psychology, this selection can be considered as a situation to action rules, or association, while in the RL process, the policy can be a simple function or demand for extensive computation in the case of a complex search process. However, the policies may be either deterministic or stochastic regarding their nature.

The reward signal can be either positive in the case of a successful feedback after performing the action or negative when the feedback was unsuccessful. Actually, the rewards are analogous to the experience of the pleasure or pain in biological systems [1]. While the reward signal estimates action in an immediate sense, the value function evaluates a sequence of actions in the long run. This means that the value function sums the total amount of reward signals in the longer period and can predict the rewards accumulated over the future.

The last element of the RL systems is a model of the environment that is optional. In some RL systems using planning, it simulates the behavior of the environment. Actually, the model is able to predict the next state by the known current state and the chosen action. Such systems are known under the name model-based, while those that do not use the planning are also named model-free.

Let us notice that we employ the model-free RL system in our study (the model-based systems are discussed here only to supplement a big picture to the readers). Interestingly, there are two kinds of RL algorithms as follows:

- value-based,
- policy-based.

The former tries to maximize a value function, where an agent expects long-term rewards of the current states using a specific policy. The latter tries to learn a policy that will lead to the maximum possible reward in the future for each action taken in every state [30].

Q-learning is one of the more applicable policy-based RL algorithms proposed by Watkins [31]. This algorithm applies the following policy for learning how to act optimally in a controlled MDP:

$$Q(S, A) = Q(S, A) + \alpha(R + \gamma \cdot \max_a Q(S', a) - Q(S, A)), \tag{2}$$

that predicts the optimal action-value for each $\langle$state,action$\rangle$ pair in the long run. Equation (2) consists of three terms. One of them is the difference between the discounted maximum action-value and the current action-value. This difference, scaled by the step size α, is then added to the current action-value in order to obtain the new action-value. The parameter γ in the equation denotes discount value that weakens the influence of the maximum action-value during the run. The discounted maximum action-value denotes a feedback from the learning process and, indeed, represents the so-called associative learning.

The pseudo-code of the Q-learning algorithm is illustrated in Algorithm 1 [1], from which it can be seen that the algorithm is controlled by three parameters: step

Algorithm 1: Q-learning algorithm.

```
Algorithm parameters: $\alpha \in (0, 1], \epsilon > 0$;
forall the $s \in S^{+} \wedge a \in A(s)$ do                /* Initialization */
    Q(s,a)=0; N(a)=0;
forall the episode do
    Initialize $S$;
    forall the $t \in T$ do
        $A = \begin{cases} \text{argmax}_n Q(s, a), & \text{with probability} 1 - \epsilon, \\ \text{random action}, & \text{otherwise.} \end{cases}$
        $\langle R, S' \rangle = \text{Take action}(A)$;
        $Q(S, A) = Q(S, A) + \alpha(R + \gamma \cdot \max_a Q(S', a) - Q(S, A))$;
        $S + S'$;
```

size α, discount value γ, and probability ϵ. After initialization, where the Q action-values and the number of action calls are initialized, two main loops are started. The outer for-loop is executed for each episodes, while the inner for-loop for each step of the episode. In the inner for-loop, either the ϵ-greedy action is selected with the probability ϵ, or the current best policy is applied with probability $1 - \epsilon$. The reward R and the new state S' are obtained after applying the action A into the environment, while both variables enter into the Q action-value function. Finally, the state S is replaced by the new state S'.

2.2 Algorithms in the Study

Ten different stochastic nature-inspired population-based algorithms are compared with the proposed variants of RL incorporated within DE algorithms. Nature-inspired algorithms are comprised of two algorithm families: EAs and SI-based. The former found their inspiration in the Darwinian struggle for existence [12], according to which the fitter individuals have more chances of surviving and transferring their good traits into the next generation. The latter mimics the behavior of animals and insects by performing vital functions to survive (e.g., foraging, reproduction) [32]. Both mentioned behaviors can be treated as an optimization process by Holland et al. [6].

Obviously, the better state-of-the-art algorithms appropriate for global optimization are observed in the study. Especially, two algorithms of the mentioned families serve as a foundation for the development of these powerful algorithms, i.e., DE and ABC. In the remainder of the chapter, both the mentioned algorithms are described in detail together with their more efficient variants representing the former as well as the present state-of-the-art algorithms.

2.2.1 Original DE and Its Variants

The original DE emerged in 1995, and various adaptive and self-adaptive variant of this algorithms have appeared since then. Although its variation operators are based on well-defined mathematical operations of vector differences, these can be interpreted in the sense of Darwinian mutation and crossover. Thus, vectors represent individuals in a population of solutions.

In our study, the following DE variants are considered:

- Original DE [13],
- Self-adaptive DE (SaDE) [23, 33],
- Self-adaptive DE (jDE) [24],
- Success-history-based adaptive DE (SHADE) [25].

Especially, the SHADE algorithm is distinguished in EC community, because of its conceptual simplicity and implementation integrity. Therefore, it is no wonder that the majority of variants based on this algorithm were winners of the CEC special session/competition on real-parameter optimization in the last decade. In our study, we are focused on the following ones:

- SHADE with linear population size reduction (L-SHADE) [26],
- Improved L-SHADE (iL-SHADE) [27],
- Single-objective real-parameter optimization algorithm (jSO) [28],
- L-SHADE algorithm with a rank-based selective pressure strategy (LSHADE-RSP) [29].

In the remainder of the chapter, all the mentioned algorithms are illustrated in detail.

The Original DE

DE was developed by Storn and Price [13] and attracted the attention of the evolutionary research community quickly by achieving good results especially in continuous optimization problem solving. Latter, its applicability was widened to solve discrete and real-world problems, where the algorithm was distinguished by achieving solid results.

The main characteristic of this EA is the use of the real-valued representation of solution. The search space is explored by DE using variation operators like mutation and crossover, while selection ensures that the best vectors in the current population are preserved to the next generation in the evolutionary cycle. The original DE modifies vectors in the population using the so-called mutation strategies that determine how the search space must be explored. Due to the high number of strategies, a special notation was proposed in order to describe them. The basic mutation strategy 'DE/rand/1/bin', for example, takes the scaled difference between two randomly selected vectors and adds them to the third vector, in other words:

$$\mathbf{u}_i^{(t)} = \mathbf{x}_{r0}^{(t)} + F \cdot (\mathbf{x}_{r1}^{(t)} - \mathbf{x}_{r2}^{(t)}), \tag{3}$$

where the scaling factor $F \in [0.0, 1.0]$ determines the rate of modification, the population size NP limits the number of individuals, and the randomly selected numbers $r0,\ r1,\ r2$ capture values in the interval $1, \ldots, NP \wedge i \neq r_0 \neq r_1 \neq r_2$. Effectively, these random numbers determine the vectors that are then entered into the mutation strategy.

The crossover parameter CR regulates the number of parameter values that are included in the operation. In effect, there are two ways of how the parameter values are handled: binomial or exponential. The binomial crossover selects the parameter values to the trial vector uniformly from the trial or target vectors. On the other hand, the exponential crossover is similar to the one- or two-point crossover in classical genetic algorithms (GA) [34], where the parameters are taken from mutant vector as long as $\text{rand}(0, 1) \leq \text{CR}$, while all the others from the target vector until the first occurrence of $\text{rand}(0, 1) > \text{CR}$. Mathematically, the binomial crossover is expressed as:

$$w_{i,j}^{(t)} = \begin{cases} u_{i,j}^{(t)}, & \text{rand}_j(0, 1) \leq \text{CR} \vee j = j_{\text{rand}}, \\ x_{i,j}^{(t)}, & \text{otherwise}, \end{cases} \tag{4}$$

where the crossover rate $\text{CR} \in [0.0, 1.0]$ controls those part of parameter values that are copied from the mutant vector to the trial solution. Furthermore, the condition $j = j_{\text{rand}}$ ensures that the trial solution $\mathbf{w}_i^{(t)}$ differs from the target solution $\mathbf{x}_i^{(t)}$ in at least one element.

The DE selection operator is known under the name 'one-to-one' selection, because the best between target and trial vector is preserved to the next generation. Mathematically, the selection is expressed as follows:

$$\mathbf{x}_i^{(t+1)} = \begin{cases} \mathbf{w}_i^{(t)}, & \text{if } f(\mathbf{w}_i^{(t)}) \leq f(\mathbf{x}_i^{(t)}), \\ \mathbf{x}_i^{(t)}, & \text{otherwise}. \end{cases} \tag{5}$$

SaDE

DE supports many mutation strategies that are applied according to the problem to be solved, but knowing which mutation strategy is the best for the particular problem is far from being discovered easily. A lot of experiments need to be conducted, but the true answer to the question cannot be found, especially, if the fitness landscape changes over time (which usually does). This means that the mutation strategy used at the beginning of the search process is not necessarily optimal when the process matures.

Therefore, Qin and Suganthan [23] proposed SaDE capable of changing the two DE mutation strategies (i.e., 'DE/rand/1/bin' and 'DE/best/2/bin') depending on the feedback from the search process. Thus, the specific strategy is selected with regard to probabilities p_1 and $p_2 = 1 - p_1$. Initially, both probabilities are set as $p_1 = p_2 = 0.5$, but during the evolutionary run, the proper strategy is selected using the following equations:

$$\mathbf{u}_i^{(t)} = \begin{cases} \mathbf{x}_{r0}^{(t)} + F \cdot (\mathbf{x}_{r1}^{(t)} - \mathbf{x}_{r2}^{(t)}), & \text{if } \mathrm{rand}_1 \leq \mathrm{CR} \wedge \mathrm{rand}_2 \leq p_1, \\ \mathbf{x}_i^{(t)} + F \cdot (\mathbf{x}_{\mathrm{best}}^{(t)} - \mathbf{x}_i^{(t)}) + F \cdot (\mathbf{x}_{r0}^{(t)} - \mathbf{x}_{r1}^{(t)}), & \text{if } \mathrm{rand}_1 \leq \mathrm{CR} \wedge \mathrm{rand}_2 > p_1, \\ \mathbf{x}_i^{(t)} & \text{if } \mathrm{rand}_1 > \mathrm{CR}, \end{cases} \tag{6}$$

where rand_1 and rand_2 denote random numbers drawn from uniform distribution in the interval [0, 1] and probabilities p_1 and p_2 are calculated using the expressions:

$$\begin{aligned} p_1 &= \frac{ns_1 \cdot (ns_2 + nf_2)}{ns_2 \cdot (ns_1 + nf_1) + ns_1 \cdot (ns_2 + nf_2)}, \\ p_2 &= 1 - p_1. \end{aligned} \tag{7}$$

The variables ns_1 and nf_1 designate the number of successful and unsuccessful modifications of trial solution, when the first DE mutation strategy is used, respectively, and ns_2 and nf_2 are the same values by application of the second DE mutation strategy.

Indeed, the original DE is controlled by three algorithm parameters: scale factor F, crossover rate CR, and population size NP, which remain fixed during the algorithm's run. Interestingly, two of these parameters (i.e., F and CR) are self-adapted in SaDE. Both mentioned parameters are attached to each individual in the population separately.

For each i-th vector, the scale parameter is modified in the interval $F_i \in [0, 2]$ according to the following equation:

$$F_i = 2 \cdot \mathcal{N}(0.5, 0.3), \tag{8}$$

where the function $\mathcal{N}(0.5, 03)$ denotes a random number drawn from the normal distribution with mean 0.5 and standard deviation 0.3.

The crossover rate CR_i is calculated more sophisticated and needs the presence of memory CR_m that is initialized to value 0.5. The new value of CR_i is determined according to the equation:

$$\mathrm{CR}_i = \mathcal{N}(\mathrm{CR}_m, 0.1). \tag{9}$$

Here, the CR_i is assigned to the random number drawn from the normal distribution with mean of CR_m and standard deviation 0.1. Interestingly, each CR_i value stays alive for five generations, during which its successful modifications are recorded.

After 25 generations, the new CR_m value is calculated as an average of recorded CR_i values.

jDE

Similarly as in SaDE, Brest et al. [24] in jDE also proposed the self-adaptation of scale factor F and the crossover rate CR that are not hold fixed during the evolutionary cycle, but are incorporated into the representation of individuals and modified by variation operators. The individuals in jDE are represented as follows:

$$\mathbf{x}_i^{(t)} = (x_{i,1}^{(t)}, x_{i,2}^{(t)}, \ldots, x_{i,D}^{(t)}, F_i^{(t)}, \text{CR}_i^{(t)}).$$

Indeed, the representation of individuals in jDE is divided into problem variables and control parameters. While the problem variables are modified using the particular DE mutation strategy, the control parameters F_i and CR_i are modified using the following two equations:

$$F_i^{(t+1)} = \begin{cases} F_\text{l} + \text{rand}_1 \cdot (F_\text{u} - F_\text{l}), & \text{if } \text{rand}_2 < \tau_1, \\ F_i^{(t)}, & \text{otherwise}, \end{cases} \tag{10}$$

$$\text{CR}_i^{(t+1)} = \begin{cases} \text{rand}_3, & \text{if } \text{rand}_4 < \tau_2, \\ \text{CR}_i^{(t)}, & \text{otherwise}, \end{cases} \tag{11}$$

where variables rand_i for $i = 1, \ldots, 4$ denote random values drawn from uniform distribution in the interval [0, 1], learning rates τ_1 and τ_2 affect the speed of learning, while the lower and upper bounds F_l and F_u limit the boundaries of feasible values for parameter F_i, respectively.

SHADE

The SHADE algorithm by Tanabe and Fukunaga [25] utilizes the historical memories $\mathbf{M}_\text{CR}$ and $\mathbf{M}_F$ to adapt the control parameters CR and F. Interestingly, both memory variables are vectors denoting elements $\mathbf{M}_\text{CR} = \{M_{\text{CR}_i}\}$ and $\mathbf{M}_F = \{M_{F_i}\}$ for $i = 1, \ldots, H$ that are initially set to 0.5. During the run, the control parameters are modified w.r.t. the following equations:

$$\begin{aligned} \text{CR}_i &= \mathcal{N}(M_{\text{CR},r_i}, 0.1), \\ F_i &= \mathcal{C}(M_{F,r_i}, 0.1), \end{aligned} \tag{12}$$

where function $\mathcal{N}(\mu, \sigma)$ denotes the random value drawn from normal distribution with mean μ and standard deviation σ, and function $\mathcal{C}(\mu, \sigma)$ the random value drawn from the Cauchy distribution of the same parameters μ and σ. In Eq. (12), variable r_i denotes the random value drawn from uniform distribution in the interval $[1, H]$.

Modifying the historical memories depends on success histories S_CR and S_F maintaining the number of successfully changed individuals in each generation. Mathematically, the calculation is expressed as:

$$M_{\mathrm{CR},k}^{(t+1)} = \begin{cases} \mathrm{mean}_{WA}(S_{\mathrm{CR}}), & \text{if } S_{\mathrm{CR}} \neq 0, \\ M_{\mathrm{CR},k}^{(t)}, & \text{otherwise}, \end{cases} \tag{13}$$

$$M_{F,k}^{(t+1)} = \begin{cases} \mathrm{mean}_{\mathrm{WL}}(S_F), & \text{if } S_F \neq 0, \\ M_{F,k}^{(t)}, & \text{otherwise}, \end{cases} \tag{14}$$

where k determines those elements in the historical memory that need to be updated. Initially, this variable is assigned to one and incremented, until $k \leq H$, where the value is reset back to $k = 1$. The function $\mathrm{mean}_{\mathrm{WA}}(S_{\mathrm{CR}})$ in Eq. (13) denotes the weighted arithmetic mean and is calculated as:

$$\mathrm{mean}_{\mathrm{WA}}(S_{\mathrm{CR}}) = \sum_{k=1}^{|S_{\mathrm{CR}}|} w_k \cdot S_{\mathrm{CR},k}, \tag{15}$$

where

$$w_k = \frac{\Delta f_k}{\sum_{k=1}^{|S_{\mathrm{CR}}|} \Delta f_k}, \tag{16}$$

and Δf_k is expressed as $\Delta f_k = |f(\mathbf{u}^{(t)}) - f(\mathbf{x}^{(t)})|$. The function $\mathrm{mean}_{\mathrm{WL}}(S_F)$ in Eq. (14) represents a weighted Lehmer mean that is expressed as:

$$\mathrm{mean}_{\mathrm{WL}}(S_F) = \frac{\sum_{k=1}^{|S_F|} w_k \cdot S_{F,k}^2}{\sum_{k=1}^{|S_F|} w_k \cdot S_{F,k}}. \tag{17}$$

The SHADE algorithm also utilizes the novel 'DE/current-to-pbest/1/bin' mutation strategy expressed as:

$$\mathbf{v}_i^{(t)} = \mathbf{x}_i^{(t)} + F_i \cdot (\mathbf{x}_{\mathrm{pbest}}^{(t)} - \mathbf{x}_i^{(t)}) + F_i \cdot (\mathbf{x}_{r_0}^{(t)} - \mathbf{x}_{r_1}^{(t)}), \tag{18}$$

where a scale factor F_i scales not only a single, but instead two different terms in the corresponding equation, i.e., social and random component. Here, the social component is referred to a difference between the randomly selected vector $\mathbf{x}_{\mathrm{pbest}}^{(t)}$ among the top $p_i \cdot NP$ vectors in the current population and the target vector $\mathbf{x}_i^{(t)}$, while the random component has the same meaning as in the 'DE/rand/1/bin' strategy. Let us mention that the p_i variable is not fixed, but randomly selected from the uniform distribution in the interval $[p_{\min}, 0.2]$, in other words:

$$p_i = \mathcal{U}(p_{\min}, 0.2), \tag{19}$$

where $p_{\min} = 2/NP$ is picked such that the x_{pbest} individual can be selected between the two best members in the current population.

L-SHADE

To improve the performance of the SHADE algorithm, it was further enhanced with the L-SHADE proposed by Tanabe and Fukunaga [26]. This was focused on the influence of the third DE parameter, i.e., the population size *NP*, where the linear population size reduction (LPSR) was introduced to decrease the population size linearly when the number of generations increases. However, this is not the first attempt to modify a population size during the evolutionary run. Arabas et al. [35] proposed the so-called variable population size within GA (GAVaPS) that modifies a population size according to the feedback from the evolutionary search process, where the population size can be increased or decreased similarly as in the natural biological environments.

The LPSR in the L-SHADE algorithm decreases the population size as follows:

$$NP^{(t+1)} = \left\lceil \left(\frac{NP_{\min} - NP_{\text{init}}}{MAX\_NFE} \cdot NFE + NP_{\min} \right) \right\rceil, \tag{20}$$

where $NP_{\min}$ is normally set to the smallest feasible value enabling the DE mutation strategy to be accomplished successfully (e.g., $NP_{\min} = 4$), NFE denotes the current number of fitness function evaluations, NP_{init} the initial size of population, and MAX_NFE is referred to the maximum number of fitness function evaluations.

iL-SHADE

Brest et al. [27] proposed iL-SHADE, an improved variant of the L-SHADE, for the CEC-16 special session/competition on real-parameter single-objective optimization. The algorithm incorporated the following improvements against its predecessor:

- Initialization of the historical memories: In place of $\mathbf{M}_{\text{CR}} = 0.5$, originally set by L-SHADE, the corresponding historical memory was initialized as $\mathbf{M}_{\text{CR}} = 0.8$ for each of the H elements. Moreover, at least one element k of the corresponding historical memories needs to be as $M_{\text{CR},k} = M_{F,k} = 0.8$.
- Updating the historical memory $\mathbf{M}_{\text{CR}}$: In place of the weighted arithmetic mean function $\text{mean}_{\text{WA}}(S_{\text{CR}})$, the Lehner mean function $\text{mean}_{\text{WL}}(S_{\text{CR}})$ was employed in the calculation of historical memory $M_{\text{CR},k}$.
- Adapting the values $\text{CR}_i^{(t)}$ and $F_i^{(t)}$: These values are adapted regarding the maturity of the evolutionary search process: The smaller values of those parameters are favored at the beginning and the higher at the end of the optimization process.
- Adapting the ratio of the top solutions p_i used in 'DE/current-to-pbest/1/bin' mutation strategy: For this calculation, the authors proposed the equation as follows:

$$p_i = \frac{p_{\max} - p_{\min}}{\text{MAX\_NFE}} \cdot \text{NFE} + p_{\min}, \tag{21}$$

  where $p_{\min}$ and $p_{\max}$ denote the predefined minimum and maximum feasible values of these constants, respectively. As can be seen from the equation, the value p_i increases from $p_{\min}$ toward $p_{\max}$ in each generation linearly.

Let us mention that all the other L-SHADE features (e.g., LPSR) remain unchanged in the proposed iL-SHADE.

jSO

The jSO variant of the L-SHADE extended the original iL-SHADE and was adjusted to the CEC-17 special session/competition on real-parameter single-objective optimization. The algorithm is distinguished against its predecessor using the following feature:

- Introduction of the 'DE/current-to-pbest-w/1/bin' mutation strategy expressed as:

$$\mathbf{v}_i^{(t)} = \mathbf{x}_i^{(t)} + F_{w_i} \cdot (\mathbf{x}_{\text{pbest}}^{(t)} - \mathbf{x}_i^{(t)}) + F_i \cdot (\mathbf{x}_{r0}^{(t)} - \mathbf{x}_{r1}^{(t)}), \tag{22}$$

  where

$$F_{w_i} = \begin{cases} 0.7 \cdot F_i, & \text{if NFE} < 0.2 \cdot \text{MAX\_NFE}, \\ 0.8 \cdot F_i, & \text{if NFE} < 0.4 \cdot \text{MAX\_NFE}, \\ 1.2 \cdot F_i, & \text{otherwise}. \end{cases} \tag{23}$$

  Interestingly, Eq. (22) introduced two scale factors F_{w_i} and F_i affecting the amount of contribution regarding the social and random components, respectively. While the scale factor F_i acts similarly as in the original L-SHADE, the scale factor F_{w_i} prefers the smaller changes of the trial's position at the beginning of the optimization process, and, vice versa, the higher changes are preferred at the end of the optimization process.

  Thus, the motivation behind the use of such mechanism is to increase the population diversity in latter phases when the one disappears gradually.

Although the changes to the iL-SHADE seem minor, the jSO has improved the results of the CEC-17 special session/competition on real-parameter single-objective optimization crucially.

LSHADE-RSP

The LSHADE-RSP presents another variant of the L-SHADE algorithm that was proposed for the CEC-18 special session/competition on real-parameter single-objective optimization by Stanovov et al. [29]. The main feature of this algorithm is introducing the 'DE/current-to-pbest/r/bin' mutation strategy that replaced the 'DE/current-to-pbest/1/bin' strategy used in the original L-SHADE. Using the proposed DE mutation strategy, the vectors entering into the random component of Eq. (18) are not selected randomly, but by considering their ranks, similarly as in GAs [34]. Indeed, the proposed mutation strategy is defined using the following equation:

$$\mathbf{v}_i^{(t)} = \mathbf{x}_i^{(t)} + F_i \cdot (\mathbf{x}_{\text{pbest}}^{(t)} - \mathbf{x}_i^{(t)}) + F_i \cdot (\mathbf{x}_{\text{pr0}}^{(t)} - \mathbf{x}_{\text{pr1}}^{(t)}). \tag{24}$$

The motivation behind using the selection is to prevail the premature convergence that typically emerges in conditions of losing the population diversity.

Let us mention that the ranking selection in GA operates in four steps: (1) sorting the solutions according to their fitness function values, (2) assigning the corresponding rank values to each individual, (3) calculating the probability of selection, and (4) applying the proportional selection operator.

In the first step, the individuals are sorted according to their fitness function values ascendingly. In the second step, the rank values are assigned to the ordering w.r.t. the following expression:

$$\text{rank}_i = k \cdot (NP - i) + 1, \tag{25}$$

where variable k presents a scaling factor that favors the better solutions by assigning the higher ranks. In the third step, the probability of selection is calculated for each solution according to the following equation:

$$pr_i = \frac{\text{rank}_i}{\sum_{j=1}^{NP} \text{rank}_j}. \tag{26}$$

The aim of using the proportional selection operator in the fourth step is to select two solutions for entering into the mutation strategy proportional to their rank values.

2.2.2 ABC and Its Self-adaptive Variant

SI-based algorithms present an additional family belonging to a class of stochastic nature-inspired population-based algorithms. Although a huge amount of this class members have been published recently, only a few can be confronted successfully with the hardest global optimization problems. On the other hand, the majority of the newly developed SI-based algorithms prefer spectacular nature inspiration before introducing their internal novelties. This means that such algorithms bring nothing new to the research community, because these, typically, present only better or worse copies of already developed ones. Fortunately, the flood of new nature-inspired algorithms has slowed dramatically after criticism by Sörensen [36].

Anyway, the focus of our study is on the SI-based algorithms that also achieved good results in solving the global optimization problems. Consequently, we selected two members of this family as follows:

- the original ABC by Karaboga and Basturk [21],
- the self-adaptive search equation-based ABC (SSEABC) by Yavuz et al. [22].

Interestingly, the ABC is one of the rare case of the SI-based algorithms that also supports crossover beside the mutation and thus proves how important this operator is for the efficiency of the results. Interestingly, this operator is applied using a probability of $1/D$, if variable D describes the length of the individuals. On the one hand, the low probability ensures slow convergence, while on the other, it avoids the search process becoming stuck in the local optima. In the remainder of the chapter, the mentioned algorithms are illustrated in detail.

ABC

The ABC algorithm was proposed by Karaboga and Basturk [21]. Similarly as in a natural bee colony, ABC also consists of three bee groups: employed, onlooker, and scout bees. In natural bee colony, the tasks of these bee groups are specific: The employed bees, for instance, are responsible for visiting the closest food sources and scanning for the amount of nectar. These then inform the onlooker bees in the same colony using the so-called waggle dance, which according to the probability of visiting a food source, select the most appropriate one for foraging. Employed bees with an empty food source that is normally foraged by employed or onlooker bees become scouts.

The search process within the ABC algorithm is divided into three phases that reflect the various roles of the virtual bee groups within the colony, i.e.,:

- employed,
- onlooker,
- scout.

In the employed phase, the employed virtual bees within the ABC algorithm select the proper food source and determine its nectar amount. Thus, it is assumed that each food source is occupied by one employed bee. The onlooker virtual bees select the proper food source based on information obtained from the employed bees. Finally, the virtual scouts arise when the fitness function values of either employed or onlooker bees are not improved for a predefined number of generation and are dedicated for discovering the new food sources.

Similarly as in EAs, the ABC search process also operates in cycles, where the individuals are evolved and thus improved their traits. However, each cycle is started with initialization and terminated according the termination condition that determines when to stop the optimization process. The maximum number of function evaluations MAX_NFE is typically applied for this purpose.

Individuals in the ABC algorithm are modified w.r.t. the following mutation strategy:

$$\mathbf{v}_i^{(t)} = \mathbf{x}_i^{(t)} + \phi_{i,j}^{(t)} \cdot (\mathbf{x}_i^{(t)} - \mathbf{x}_k^{(t)}), \tag{27}$$

where a scaling factor $\phi_{i,j}^{(t)}$ is randomly drawn from the uniform distribution in the interval $[-1.0, 1.0]$, a vector $\mathbf{x}_k^{(t)}$ is randomly selected from bee colony, whereby the following inequality relation $i \neq k$ needs to be fulfilled. The probability of visiting the particular food source by the onlooker bee is expressed as:

$$p_i = \frac{f(\mathbf{x}_i)}{\sum_{j=1}^{NP} f(\mathbf{x}_j)}. \tag{28}$$

Let us notice that Eq. (27), determining a move of an artificial bee within the search space, is actually applied twice in each optimization cycle: (1) for each employed bee and (2) for those onlooker bees having the probability of visiting the food source higher than the assigned probability is in Eq. (28). However, only one randomly selected element of solution vector is modified using Eq. (27). This means that either one, or a maximal two, changes are applied to each individual in each generation. Finally, each individual, fulfilling conditions to become scout, is reinitialized. Actually, such a scout can potentially ensure for increasing the population diversity.

SSEABC

The SSEABC algorithm emerged in the CEC-16 special session/competition on real-parameter single-objective optimization and was proposed by Yavuz et al. [22]. Actually, it utilizes three strategies for exploring the problem search space: (1) determination of the self-adaptive search equation, (2) selection of the competitive local search, and (3) increment of the population size. Individuals in this algorithm $\mathbf{x}_i^{(t)} = \{x_{i,j}^{(t)}\}$ for $j = 1, \ldots, m$, and variable $m \in [1, D]$ determines the number of modified elements within the trial vector, are modified using the mutation strategy as follows:

$$x_{i,j}^{(t)} = \text{term}_1 + \phi_1 \cdot \text{term}_2 + \phi_2 \cdot \text{term}_3 + \phi_3 \cdot \text{term}_4, \tag{29}$$

where

term_1 = the origin vector that can be either current, or best, or randomly selected,
$\text{term}_2, \text{term}_3, \text{term}_4$ = various combinations of the second term in Eq. (27) (e.g., $(\mathbf{x}_{\text{best}} - \mathbf{x}_{r1})$, $(\mathbf{x}_{r0} - \mathbf{x}_{r1})$, etc.),
$\phi_1, \phi_2, \phi_3,$ = scale factors

The mentioned exploration strategies implemented within the SSABC are implemented as follows: The first strategy maintains a set of available components, from which the particular terms in Eq. (28) can be constructed. Each successful application of the specific component increases its success rate. Then, the components that have the lowest success rates are excluded from the set.

The second strategy supports two local search heuristics that compete between each other during the predefined number of fitness function evaluations. The result of this compete phase is determining the better local search heuristic that is applicable in a deployment phase. The deployment phase is terminated, when the population begins to stagnate. However, this condition causes that the compete phase is taking place again.

The last strategy was inspired by the incremental social learning (ISL) proposed by Montes De Oca et al. [37] and adds a new individual after a predefined number of generations to the existing population. However, the new individual $\mathbf{x}_{\text{new}}^{(t)}$ is created w.r.t. the following equation:

$$x_{\text{new},j}^{(t)} = x_{\text{ini},j}^{(t)} + \phi_{i,j} \cdot (x_{\text{best},j}^{(t)} - x_{\text{ini},j}^{(t)}), \tag{30}$$

where $\mathbf{x}_{\text{ini}}^{(t)}$ denotes the target vector generated randomly and $\mathbf{x}_{\text{best}}^{(t)}$ the current best solution.

3 The Proposed RL-SHADE

The motivation behind the design and implementation of the proposed RL-SHADE algorithm was to develop an algorithm that would be able to achieve the best results on all the CEC benchmark suites for real-parameter single-objective optimization in the last decade. In line with this, we focused on three state-of-the-art algorithms of their time that were founded on the same origin.

As eligible candidates for our study, the following algorithms seemed to be the more appropriate: L-SHADE, iL-SHADE, and jSO. All three mentioned algorithms based on the original SHADE and were developed in different periods. While the L-SHADE was the best adapted to the CEC-14 benchmark suite in 2014, the iL-SHADE achieved brilliant results by solving the same benchmark problems in 2016, while the jSO was excellent at solving the CEC-17 benchmark suite one year later. However, in general, no algorithm is suitable to achieve the best results on all the mentioned benchmarks.

Therefore, the idea is to overcome the posted problem by using the RL mechanism incorporated into one algorithm that could be suitable to cover the characteristics of all three mentioned SHADE-based algorithms under the same umbrella. At first, we need to identify the characteristics (also the exploration strategies) of each particular algorithm and then apply the RL Q-learning mechanism for adjusting the particular exploration strategy to different fitness landscapes caused by different benchmark problems to be solved, in order to achieve the best long-term value.

Obviously, the fitness landscape is changing dynamically, and consequently, this changing must be followed by an exploring strategy online. For instance, the jSO strategy of exploring search space is good in one moment, while the L-SHADE one is better in another. Furthermore, the fitness landscape depends on the benchmark problem suite. In summary, the RL mechanism enables selecting the more successful strategies during a longer period. Within the proposed RL-SHADE algorithm, this serves as a mechanism for adapting the exploration search process to the fitness landscape. Although this adaptation is not sensitive to the rapid changes in the environment, we expect that using the RL mechanism would bring long-term benefits as well.

The strategy of exploring the search space in the SHADE algorithm is not limited only to the used DE mutation strategy. Indeed, the L-SHADE and iL-SHADE use the same 'DE/current-to-pbest/1/bin', while the jSO the slightly modified 'DE/current-to-pbest-w/1/bin'. The main differences are in other components of the basic algorithm, like initialization, or the way of selecting the proper values of parameters F_i and CR_i, or the sizes of the historical memories.

Beside the initialization, the SHADE algorithm consists of three different phases within the evolutionary cycle that are retained in all the mentioned SHADE variants:

- generation of trial vectors,
- selection of the better between target and trial solutions and maintaining the success histories,
- updating historical memories.

Algorithm 2: RL-SHADE algorithm.

```
Input: pop_strategy
$G = 1, N_G = N^{init}, A = \emptyset$;
$N[a] = 1, Q[a] = 0$ for $a \in [1, K]$ ;                                   /* RL-step 1 */
Initialize population $\mathbf{P}_G = (\mathbf{x}_{1,G}, \ldots, \mathbf{x}_{N,G})$ randomly;
if $pop\_strategy = lSHADE$ then
    Set all values in $\mathbf{M}_{CR}, \mathbf{M}_F$ to 0.6; $H = 5$; $p = 0.11$;   /* L-SHADE */
else if $pop\_strategy = iL-SHADE$ then
    Set all values in $\mathbf{M}_{CR}$ to 0.8, $\mathbf{M}_F$ to 0.5;$p_{\max} = 0.2$;$p_{\min} = 0.1$;$H = 6$;   /* iL-SH */
else
    Set all values in $\mathbf{M}_{CR}$ to 0.8, $\mathbf{M}_F$ to 0.3; $p_{\max} = 0.25$;$p_{\min} = 0.1$;$H = 5$;   /* jSO */
while Termination criteria not met do
    $S_{CR} = 0, S_F = 0, a = pop\_strategy$;
    for $i = 1$ to $N$ do                                                   /* RL-step II */
        $a = \begin{cases} \text{argmax}_{a=1,\ldots,K} Q[a], & \text{if } \text{rand}(0,1) < 1 - \epsilon, \\ \text{rand}(1, K) \wedge k \neq \text{ord}(a) \wedge \neg\text{hold}(max\_try), & \text{otherwise.} \end{cases}$
        $r_i$ = Select from $[1, H]$ randomly;
        if $a \in \{iL-SHADE, jSO\}$ then                                     /* iL-SHADE,jSO */
            if $r_i = H$ then
                $M_{F,r_i} = 0.9$;  $M_{CR,r_i} = 0.9$;
        if $M_{CR,r_i} = \perp$ then                                         /* L-SHADE,iL-SHADE,jSO */
            $CR_{i,G} = 0$;
        else
            $CR_{i,G} = \text{randn}_i(M_{CR,r_i}, 0.1)$;
        if $a \in \{iL-SHADE, jSO\}$ then                                     /* iL-SHADE,jSO */
            if $g < 0.25G_{MAX}$ then
                $CR_{i,G} = [\max(CR_{i,G}, 0.5) | \max(CR_{i,G}, 0.7)]$;
        else if $g < 0.5G_{MAX}$ then
            $CR_{i,G} = [\max(CR_{i,G}, 0.25) | \max(CR_{i,G}, 0.6)]$;
        $F_{i,G} = \text{randc}_i(M_{F,r_i}, 0.1)$;
        if $a = iL-SHADE$ then                                               /* iL-SHADE */
            if $g < 0.25 \cdot G_{MAX}$ then
                $F_{i,G} = \min(F_{i,G}, 0.7)$;
            else if $g < 0.5G_{MAX}$ then
                $F_{i,G} = \min(F_(i,G), 0.8)$;
            else if $g < 0.75G_{MAX}$ then
                $F_{i,G} = \min(F(i,G), 0.9)$;
        if $a = jSO$ then                                                    /* jSO */
            if $g < 0.6 \cdot G_{\max} \wedge F_{i_G} > 0.7$ then
                $F_{i,G} = 0.7$;
        $\mathbf{u}_{i,G}$ = Generate trial vector($\mathbf{x}_{i,G}, a$);
        Update Reinforcement Learning($\mathbf{u}_{i,G}, a$) ;                  /* RL-step III */
    for $i = 1$ to $N$ do
        if $f(\mathbf{u}_{i,G}) \leq f(\mathbf{x}_{i,G})$ then
            $\mathbf{x}_{i,G+1} = \mathbf{u}_{i,G}$;
        else
            $\mathbf{x}_{i,G+1} = \mathbf{x}_{i,G}$;
        if $f(\mathbf{u}_{i,G}) \leq f(\mathbf{x}_{i,G})$ then
            $A = A \cup \mathbf{x}_{i,G}$;$S_{CR} = S_{CR} \cup CR_{i,G}$;$S_F = S_F \cup F_{i,G}$;
```

Algorithm 2:

```
while do
    Update archive size to $|A|$;
    Update memories $\mathbf{M}_{CR}$ and $\mathbf{M}_F$ (pop_strategy) ;    /* Algorithm 3 */
    Calculate new $N_{G+1}$;    /* Optional LPSR strategy. */
    if $N_G < N_{G+1}$ then
        Sort individuals in $\mathbf{P}$ according their fitness values and delete lowest $N_G - N_{G+1}$
        members;
        Resize archive size $|A|$ according to new $|P|$;
    if $pop\_strategy \in \{iL-SHADE, jSO\}$ then    /* iL-SHADE,jSO */
        Update $p$ using: $p = \frac{p_{\max}-p_{\min}}{max\_nfes} \cdot nfes + p_{\min}$;
    G++;
```

The L-SHADE is obtained when the LPSR operation is added to the original SHADE algorithm. Obviously, this feature, although slightly modified, is also applied in the iL-SHADE and jSO algorithms.

The pseudo-code of the proposed RL-SHADE algorithm is illustrated in Algorithm 2, from which it can be seen that the algorithm is called with an argument pop_strategy. The argument affects primarily the sizes of historical memories and their updating process. The first for-loop statement inside the evolutionary cycle (while-loop) is devoted to the generation of trial vectors, the second for-loop implements the one-to-one selection and ensures maintaining the success histories, while the remainder of the algorithm serves for the updating archive solutions and historical memories (Algorithm 3). Finally, the LPSR algorithm follows.

In the pseudo-code Algorithm 2, the changes in the original SHADE algorithm, referred to the RL mechanism, are denoted in red. All the other changes represent an implementation of ensemble strategies and are self-explanatory. Indeed, the RL mechanism demands three changes in the proposed RL-SHADE algorithm:

- initialization,
- action selection,
- update of the RL structures.

The RL mechanism introduces two vectors: (1) the number of particular strategy applications $N(a)$ and (2) the Q-values corresponding to each strategy $Q(a)$. In our study, the ensemble strategies consisted of three strategies (i.e., $K = 3$) determining the behavior of the original algorithms, in other words:

$$a = \{\text{lSHADE,iL-SHADE,jSO}\}.$$

A variable a is selected for each ith trial vector using the ϵ-greedy selection method determining a selected action. Typically, the RL mechanism allows only a single application of the ϵ-greedy selected action. In our study, we introduced the parameter max_try enabling the ϵ-greedy selected action to be used several times before being replaced (function ‘hold()’). In this case, the ϵ-greedy selected action has more

Algorithm 3: Algorithm for updating the historical memories $\mathbf{M}_{\mathrm{CR}}$ and M_F.

```
Input: strategy
if $S_{CR} \neq 0$ and $S_F \neq 0$ then
    if $M_{CR,k,G} = \perp$ or $\max(S_{CR}) = 0$ then
        $M_{\mathrm{CR},k,G+1} = \perp$;
    else
        if strategy = lSHADE then
            $M_{\mathrm{CR},k,G+1} = \mathrm{mean}_{WL}(S_{\mathrm{CR}})$;                 /* L-SHADE */
        else
            $M_{\mathrm{CR},k,G+1} = (\mathrm{mean}_{WL}(S_{\mathrm{CR}}) + M_{\mathrm{CR},k,G})/2$;   /* iL-SHADE,jSO */
    if strategy = lSHADE then
        $M_{F,k,G+1} = \mathrm{mean}_{WL}(S_F)$;                            /* L-SHADE */
    else
        $M_{F,k,G+1} = (\mathrm{mean}_{WL}(S_F) + M_{F,k,G})/2$;              /* iL-SHADE,jSO */
    $k = k + 1$;
    if $k > H$ then
        $k = 1$;
else
    $M_{\mathrm{CR},k,G+1} = M_{\mathrm{CR},k,G}$;
    $M_{F,k,G+1} = M_{F,k,G}$;
```

chances to improve its Q-value and becomes the current best policy. The selected action affects adjusting the parameters F_i and CR_i and, consequently, generating the trial vector. After evaluation of trial solutions, the RL structures need to be updated. In line with this, the following equation is employed for updating the Q-values:

$$Q(S, A) = Q(S, A) + \alpha(R + \gamma \cdot Q_{\max} - Q(S, A)), \tag{31}$$

where

$$\begin{aligned}
\alpha &= \frac{1}{N(A)}, && \text{step size,}\\
R &= \frac{f(\mathbf{x}_i) - f(\mathbf{u_i})}{f(\mathbf{x}_i)}, && \text{reward calculation,}\\
\gamma &= \frac{MAX\_NFE - NFE}{MAX\_NFE}, && \text{discount value,}\\
Q_{\max} &= \frac{f(\mathbf{x}_{\text{best}}) - f(\mathbf{u_i})}{f(\mathbf{x}_{\text{best}})}, && \text{associative learning.}
\end{aligned}$$

Interestingly, the step size α, determining the learning rate, decreases in the interval $\alpha \in \left[1, \frac{1}{MAX\_NFE}\right]$, and the discount value γ regulates the influence of associative learning in the interval $\gamma \in [0, 1]$.

4 Experiments and Results

The purpose of our experimental work was to show that the following hypothesis holds: “The RL-SHADE algorithm can outperform the results of the other algorithms in the study on all the CEC special session/competition on real-parameter single-objective optimization in the last decade”. A lot of experiments were conducted, where the characteristics of the proposed RL-SHADE algorithm were analyzed first, and then an extensive comparative study was performed that tried to accept or reject the hypothesis set.

In the remainder of the chapter, the structure of the section is as follows: First, the algorithm setup follows together with summarizing the main features distinguished by this. Next, the experimental setup describes the characteristics of various benchmark suites. Then, measures are illustrated for comparing the quality of the results achieved by different algorithms. Finally, the obtained results are presented in detail.

4.1 Algorithm Setup

The present study includes those algorithms, whose implementations in C++ programming language can be found on the Internet. The algorithm setup used in the study is presented in Table 1, which shows the special features that are implemented by the particular algorithms. Obviously, the proposed RL-SHADE is also included in the collection.

Let us mention that the parameter setup of the algorithm is the same as proposed by the authors of the implemented algorithms. Moreover, all the algorithms were compiled using the same g++ compiler on Linux Ubuntu 20.04 running on a personal computer with an AMD Ryzen 7 1700 Eight-Core Processor and 16 GB memory. Furthermore, all the algorithms were run under the same conditions, e.g., they used the same termination condition. In this way, we wanted to make the comparative study as fair as possible.

4.2 Characteristics of the Benchmark Suites

The CEC special sessions/competitions are organized as a part of general CEC in order to provide the general test bed for comparing the new developed state-of-the-art nature-inspired algorithms [20]. These benchmarks are typically arisen annually, and their complexity is increasing from year to year. These competitions include various types of benchmark problem suites, like: single-objective, large-scale, noisy, multi-objective, and constrained optimization [38].

This study focuses on the CEC special session/competition on real-parameter single-objective optimization benchmark function suites. Obviously, several compe-

Table 1 Collection of stochastic nature-inspired population-based algorithms in the study

Org.	Variant	Branch	Adaptation			Ens.	Selection		Memory		Archive
alg.			*F*	CR	*NP*	strat.	Stand.	Rank	$\mathbf{M}_F$	$\mathbf{M}_{\mathrm{CR}}$	
DE	Original						✓				
	SaDE		✓	✓		✓	✓			✓	
	jDE		✓	✓			✓				
	SHADE	Original	✓	✓			✓		✓	✓	✓
		L-SHADE	✓	✓	✓		✓		✓	✓	✓
		iL-SHADE	✓	✓	✓		✓		✓	✓	✓
		jSO	✓	✓	✓		✓		✓	✓	✓
		LSHADE_RSP	✓	✓	✓			✓	✓	✓	✓
		RL-SHADE	✓	✓	✓	✓	✓		✓	✓	✓
ABC	Original						✓				
	SSEABC		✓		✓	✓	✓				

Table 2 Function benchmark suites for the CEC special sessions/competitions on real-parameter single-objective optimization in the last decade

Function	CEC'13		CEC'14/16		CEC'17/18	
type	*NF*	*D*	*NF*	*D*	*NF*	*D*
Unimodal	5	{10,30,50,100}	3	{10,30,50,100}	2	{10,30,50,100}
Multi-modal	15	{10,30,50,100}	13	{10,30,50,100}	7	{10,30,50,100}
Composition	8	{10,30,50,100}	8	{10,30,50,100}	10	{10,30,50,100}
Hybrid	n/a	n/a	6	{10,30,50,100}	10	{10,30,50,100}
$\sum$	28	4	30	4	29	4

titions were organized over time. Interestingly, definitions of benchmark functions as well as results of the particular competitions together with the source codes of competitive algorithms were eagerly collected by Prof. Suganthan [39]. He is also the main driving force of EC for developing the new nature-inspired algorithms capable of solving the increasingly difficult problems, with which humans are confronted daily.

Although those special sessions/competitions were organized on an annual basis, the same benchmark suites were arisen twice only. For instance, the benchmark suite for the session/competition in the year 2014 [18] arose again in the year 2016, while the benchmark suite from the year 2017 occurred again in the year 2018. Interestingly, this CEC special session/competition was transformed to the 100-digit challenge special session and the competition on single-objective numerical optimization in 2019 and 2020 [40]. Interestingly, the functions in the benchmark suites are implemented in several programming languages (i.e., MATLAB, C, and Java). This means that development of the state-of-the-art algorithms is independent on the programming languages and, therefore, open for wide range of competitors.

Table 2 shows characteristics of the observed benchmark suites, from which it can be seen that they are presented according to the function types and corresponding issues of CEC special session/competition benchmark suite. Each benchmark suite is presented with the number of the functions *NF* and their available dimensions D. For more information about function types, the interested reader is invited to look the appropriate literature provided [17–19]. Although the benchmark suites enable solving the functions of dimensions $D = 10$, $D = 30$, $D = 50$, and $D = 100$, only the first two dimensions were observed in our preliminary study due to the time complexity, while a termination condition for algorithms in tests obeyed the regulations of the CEC special sessions/competitions that imposes MAX_NFE $\cdot$ D.

4.3 Measures

Two statistical tests were employed for comparing the results of different algorithms as follows:

- the Friedman non-parametric,
- the Spearman correlation.

Friedman non-parametric statistical tests are conducted on so-called classifiers, where each classifier merges the results of a particular algorithm per dimension of the functions included into the specific benchmark suite. Thus, each classifiers represents the results in the sense of five statistical measures (i.e., *best, worst, mean, median,* and *StDev*) averaged over 51 independent runs of the algorithm per benchmark suite. In summary, the classifier length is $5 \times NF$, where NF designates the number of functions in the benchmark suite.

The [41] statistical tests are conducted in order to compare the results of different stochastic nature-inspired algorithms collected into statistical classifiers. These non-parametric tests are founded on an analysis of variances by ranks. Thus, the null hypothesis assumes that the medians between ranks of all algorithms are equal. These statistical tests are divided into two steps. The first step is devoted to calculation of the statistics, from which ranks of particular algorithms are determined. In our study, the higher the rank value, the better the particular algorithm. The second step, which is conducted only if the posted hypothesis is rejected, demands performing the post-hoc statistical tests based on the ranks calculated in the first step. Interestingly, the second step was omitted in our study because the comparison was performed on the results of the first step (i.e., ranks). The significance level of 0.05 was employed by performing these tests.

Similarly as Friedman test, the Spearman non-parametric test also starts with an assumption that statistical classifiers do not correlate (formally written $r_s = 0$). In other words, it is assumed that a relationship between two classifiers is not linear, but arbitrarily curved.

Fister et al. [20] proposed the Spearman rank correlation for comparing ranks of algorithms obtained by optimizing benchmark functions suite X with the same results obtained by optimizing benchmark function suite Y. In our study, the ranks are calculated using the Friedman's non-statistical tests in step one. Let us suppose that two distributed variables r_f and $r_f^{'}$ are obtained as a result of a Friedman's test (also ranking lists). Then, the idea behind using the Spearman correlation is that when the connection between both non-normally distributed variables r_f and $r_f^{'}$ is significant, and the algorithm's rank in r_X is the same as in r_Y.

4.4 Results

In order to accept or reject the hypothesis set at the beginning of the section, three experiments were conducted, in which we investigated:

- influence of the parameter pop_strategy,
- influence of the parameter max_try,
- influence of selecting the benchmark suite.

The purpose of the first two experiments was to get an insight into the behavior of the proposed RL-SHADE. Then, the impact on the results was inspected by selecting the particular benchmark suite. At the end of the chapter, the obtained results are also discussed in detail.

4.4.1 Influence of the Parameter pop_strategy

The parameter pop_strategy determines the initial value of the RL-SHADE exploring strategy and has a crucial impact on the produced results. Beside selecting the initial strategy, this parameter also affects the initialization of the history memories $\mathbf{M}_{\mathrm{CR}}$ and $\mathbf{M}_F$, their sizes, and, consequently, the LPSR behavior. Practically, the size of the history memory in the L-SHADE and the iL-SHADE was set to $H = 6$, while in the jSO to $H = 5$.

Indeed, the results of this experiment are presented as the ranks calculated after the Friedman non-parametric tests. The ranks of particular algorithms in the test, obtained by varying the parameter $pop\_strategy \in$ {L-SHADE, iL-SHADE, jSO}, are presented in Fig. 2, which is divided into two parts according to the dimension of benchmark functions. Actually, ranks of three original SHADE-based algorithms (i.e., L-SHADE, il-SHADE, jSO) are compared with three RL-SHADE variants using three different initial exploration strategies. These were obtained by various setting of the pop_strategy denoted as ST-1 for $pop\_strategy = lSHADE$, ST-2 for $pop\_strategy = iL - SHADE$, and ST-3 for $pop\_strategy = jSO$. Thus, each part consists of a diagram and a table, where the former illustrates the ranks of algorithms graphically, and the latter presents the same results numerically. Let us notice that the best ranked algorithms are denoted in the red colored bars, while the best RL-SHADE variants in the blue colored bars. The results were obtained by setting parameter $max\_try = 1$.

As can be seen from the figure, there are no significant differences between the six SHADE-based algorithms by solving the CEC'13 benchmark suite on functions of dimension $D = 10$. However, the RL-SHADE using the jSO exploration strategy (i.e., ST-3) exposed the best results by optimizing the other benchmark suites on functions of the same dimension. The situation is changed dramatically by solving the benchmark suites on functions of dimension $D = 30$. Here, the best ranks overall were obtained by iL-SHADE for the CEC-13 and CEC-14, while the CEC-17 was solved the best by the jSO algorithm. The best overall results among the RL-SHADE variants were exposed by the ST-1 that dominated the ranks of the other algorithm variants (i.e., ST-2 and ST-3) by solving the benchmark suites CEC-14 and CEC-17. The ST-2 achieved slightly better rank by solving the CEC-13 benchmark suite.

4.4.2 Influence of the Parameter *max_try*

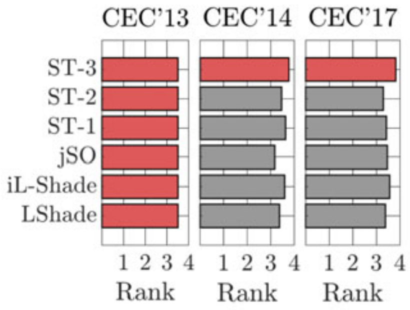
The results of the previous experiment actually rejected our hypothesis set at the beginning of the experimental section. Actually, they showed that there is no RL-SHADE algorithm that could be ranked higher than the other original SHADE-based algorithm by solving the benchmark suite on functions of dimension $D = 30$.

However, introducing the parameter max_try could help us to overcome the problem. The parameter determines how many generations the ϵ-greedy selected action is held before being replaced with the current best policy. Indeed, extensive experiments have shown that holding the ϵ-greedy for $max\_try > 1$ generations can improve the behavior of the RL-SHADE algorithm crucially. Obviously, the proper setting of this parameter is the subject of the experimental work.

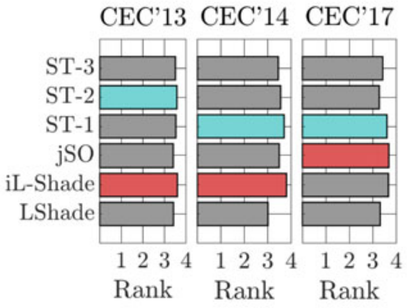
Although many settings of this parameter were tested during the experiments, the setting $max\_try = 4$ was distinguished by the quality of the obtained results. Therefore, the ranking of the algorithms in the test is presented in Fig. 3 using the mentioned setting.

Interestingly, the outline of the figure is similar as in the previous test: Ranks of the same algorithms are presented by two observed dimensions in graphical and numerical form.

As can be seen from the presented ranks of the algorithms by solving all the benchmark suites on functions of dimension $D = 10$, the overall best ranked was



a: Graphical representation for $D = 10$.



b: Graphical representation for $D = 30$.

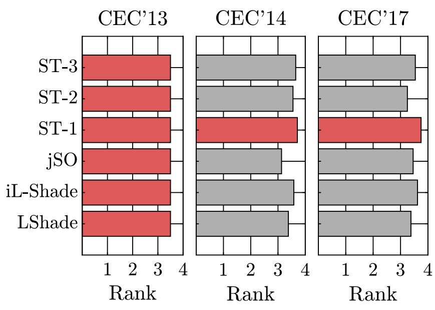
Nr.	Algorithms	Benchmark suite		
		CEC'13	CEC'14	CEC'17
1	L-SHADE	3.50	3.37	3.39
2	iL-SHADE	3.50	3.59	3.57
3	jSO	3.50	3.17	3.47
4	ST1	3.50	3.63	3.43
5	ST2	3.50	3.46	3.30
6	ST3	3.50	**3.77**	**3.83**

c: Ranking obtained by $D = 10$.

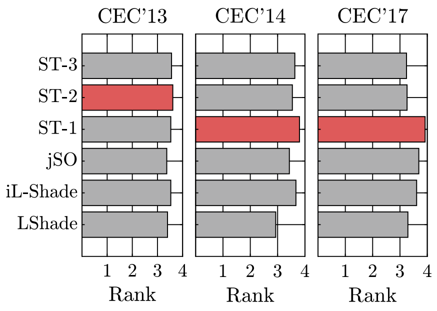
Nr.	Algorithms	Benchmark suite		
		CEC'13	CEC'14	CEC'17
1	L-SHADE	3.41	3.01	3.32
2	iL-SHADE	**3.59**	**3.80**	3.67
3	jSO	3.40	3.49	**3.69**
4	ST1	3.52	3.70	3.61
5	ST2	3.57	3.55	3.28
6	ST3	3.50	3.45	3.43

d: Ranking obtained by $D = 30$.

Fig. 2 Influence of the parameter *pop_strategy*



a: Graphical representation for D = 10.



b: Graphical representation for D = 30.

Nr.	Algorithms	Benchmark suite		
		CEC'13	CEC'14	CEC'17
1	L-SHADE	3.50	3.38	3.38
2	iL-SHADE	3.50	3.58	3.62
3	jSO	3.50	3.13	3.46
4	ST1	3.50	**3.71**	**3.75**
5	ST2	3.50	3.55	3.25
6	ST3	3.50	3.65	3.54

c: Ranking obtained by D = 10.

Nr.	Algorithms	Benchmark suite		
		CEC'13	CEC'14	CEC'17
1	L-SHADE	3.40	2.93	3.29
2	iL-SHADE	3.52	3.67	3.61
3	jSO	3.37	3.43	3.69
4	ST1	3.53	**3.80**	**3.92**
5	ST2	**3.61**	3.54	3.26
6	ST3	3.56	3.63	3.24

d: Ranking obtained by D = 30.

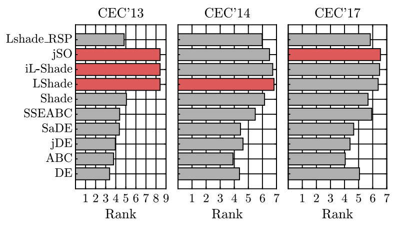
Fig. 3 Influence of the parameter *max_try*

the ST-1 algorithm (i.e., RL-SHADE using the *lSHADE* exploration strategy). The situation was even improved by optimizing the benchmark suites on functions of dimension $D = 30$, where again the best ranked algorithm on CEC-14 and CEC-17 was the ST-1 variant of the RL-SHADE. On the CEC-13 benchmark suite, however, the ST-2 variant was better.

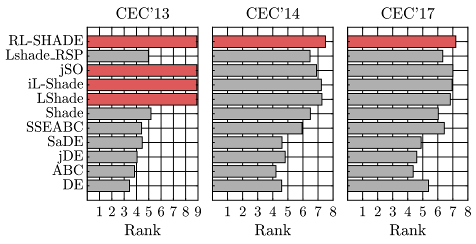
4.4.3 Influence of Selecting the Benchmark Suite

The purpose of this experiment was to investigate the influence of the benchmark suite selection on the ranking of the particular algorithm in tests. In line with this, the original ranking of ten stochastic nature-inspired population-based algorithms as obtained by Fister et al. [20] was compared with refined ranking considering also the results of the proposed RL-SHADE. Obviously, the best variant of the RL-SHADE (i.e., RL-SHADE using the *lSHADE* exploration strategy denoted as ST-1) was considered in the experiment.

The ranking of the algorithms obtained by optimizing the CEC benchmark suites of dimension $D = 10$ is depicted in Fig. 4. The latter is divided into the original ranking as proposed by Fister et al. [20] and the refined ranking by including the RL-SHADE algorithm. Both rankings were illustrated graphically and numerically, while the best algorithm was presented as red colored bars in the diagrams.



a: The original graphics (D = 10).



b: The refined graphics (D = 10).

Nr.	Algorithms	Benchmark suite		
		CEC'13	CEC'14	CEC'17
1	DE	3.39	4.37	5.06
2	ABC	3.78	3.91	4.05
3	jDE	3.96	4.62	4.39
4	SaDE	4.36	4.44	4.65
5	SSEABC	4.39	5.49	5.93
6	SHADE	5.06	6.15	5.67
7	L-SHADE	**8.41**	**6.82**	6.39
8	iL-SHADE	**8.41**	6.73	6.49
9	jSO	**8.41**	6.50	**6.55**
10	LSHADE_RSP	4.83	5.98	5.83

c: The original ranking (D = 10).

Nr.	Algorithms	Benchmark suite		
		CEC'13	CEC'14	CEC'17
1	DE	3.44	4.59	5.38
2	ABC	3.84	4.21	4.36
3	jDE	4.05	4.82	4.60
4	SaDE	4.47	4.61	4.89
5	SSEABC	4.42	5.94	6.43
6	SHADE	5.18	6.49	6.03
7	L-SHADE	**8.91**	7.26	6.83
8	iL-SHADE	**8.91**	7.22	6.95
9	jSO	**8.91**	6.90	7.00
10	LSHADE_RSP	4.97	6.47	6.32
11	RL-SHADE	**8.91**	**7.49**	**7.20**

d: The refined ranking (D = 10).

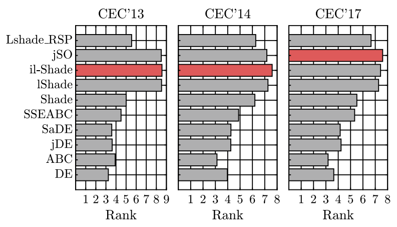
Fig. 4 Comparing the results of the original versus the refined study ($D = 10$)

The original study revealed that none of the algorithms is capable of solving all the benchmark suites at best ranks in general. The L-SHADE was indeed the best ranked algorithm by solving the benchmark suites CEC-13 and CEC-14, but the jSO is performed better at the benchmark suite CEC-17. However, if considering the results of the RL-SHADE, it can be concluded that this algorithm performed at best ranks at all the benchmark suites.

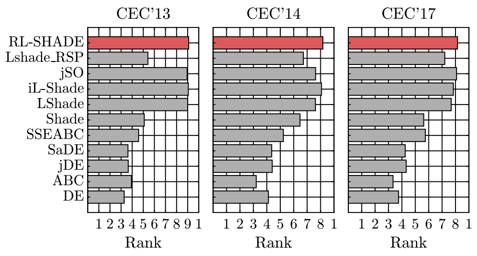
The results of the algorithms by optimizing the benchmark suites on functions of dimension $D = 30$ are illustrated in Fig. 5.

Here, the fact revealed by Fister et al. [20] is again valid for the dimension of functions $D = 30$. Actually, the iL-SHADE achieved the best rank by optimizing the CEC-13 and CEC-14 benchmark suites, while the jSO was the best ranked by solving CEC-17 benchmark suite. Contrarily, after the proposed RL-SHADE was included into the ranking, this one became the best ranked algorithm in general.

In the next experiment, we were interested in how the selection of the benchmark influenced the ranking of the particular algorithm. Consequently, we combined the ranks obtained after the Friedman non-parametric tests for all observed function dimensions according to different benchmark suites into classifiers. These classifiers were entered into Spearman correlation tests.



a: The original graphics (D = 30).



b: The refined graphics (D = 30).

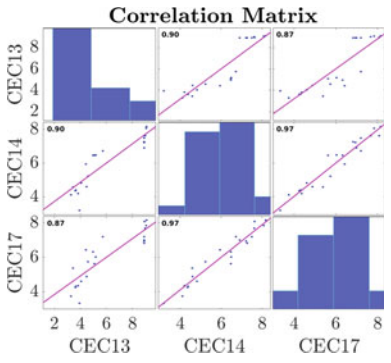
Nr.	Algorithms	Benchmark suite		
		CEC'13	CEC'14	CEC'17
1	DE	3.24	3.97	3.64
2	ABC	3.92	3.13	3.18
3	jDE	3.62	4.25	4.23
4	SaDE	3.58	4.26	4.16
5	SSEABC	4.50	4.88	5.33
6	SHADE	5.01	6.20	5.51
7	L-SHADE	8.52	7.26	7.27
8	iL-SHADE	**8.56**	**7.60**	7.43
9	jSO	8.49	7.17	**7.60**
10	LSHADE_RSP	5.55	6.27	6.66

c: The original ranks (D = 30).

Nr.	Algorithms	Benchmark suite		
		CEC'13	CEC'14	CEC'17
1	DE	3.28	4.12	3.75
2	ABC	3.94	3.21	3.34
3	jDE	3.65	4.40	4.32
4	SaDE	3.62	4.36	4.25
5	SSEABC	4.58	5.23	5.75
6	SHADE	5.09	6.47	5.63
7	L-SHADE	9.00	7.62	7.68
8	iL-SHADE	9.06	8.07	7.84
9	jSO	8.95	7.63	8.08
10	LSHADE_RSP	5.74	6.72	7.20
11	RL-SHADE	**9.09**	**8.17**	**8.15**

d: The refined ranks (D = 30).

Fig. 5 Comparing the results of the original vs. the refined study ($D = 30$)



Nr.	Benchm.	CEC'13	CEC'14	CEC'17
1	CEC'13	1.00	0.90	0.87
2	CEC'14	0.90	1.00	0.97
3	CEC'17	0.87	0.97	1.00

a: Spearman correlation test.

Fig. 6 Spearman correlation tests with corresponding correlation matrix based on the results of optimizing the benchmark suites of $D = 10$ and $D = 30$

The results of the Spearman tests are presented in Fig. 6 in the form of a correlation matrix.

Figure is again divided into a diagram and a table representing the same results in graphical as well as numerical form.

The diagram in the figure shows that the relationship between rankings obtained by optimizing the CEC-14 and CEC-17 was strongly correlated (Spearman correlation coefficient equaled $r_s = 0.97$). Relationships between the ranks CEC-13 - CEC-14 ($r_s = 0.90$) and CEC-13 - CEC-17 ($r_s = 0.87$) showed as little smaller, but still significant.

4.5 Discussion

The main hypothesis asserted in our study was that an RL-SHADE algorithm exists which is the best ranked on all ranking lists obtained after the Friedman non-parametric tests on the results of optimizing the various CEC benchmark suites. Experimental results of ranking the eleven stochastic nature-inspired population-based algorithms in the refined study, illustrated in Figs. 4 and 5, confirmed that the proposed RL-SHADE (precisely the RL-SHADE using the *lSHADE* exploration strategy) is the best ranked algorithm in all the observed ranking lists by solving the problems of dimensions $D = 10$ and $D = 30$. This means that the hypothesis can be accepted for such problem dimensions. However, to expand the same hypothesis to other problem dimensions, additional effort must be made in the future.

The results of the Spearman correlation tests need to be analyzed in order to show how the rankings of the other algorithms in ranking lists are changed w.r.t. selection of the benchmark suite. In this sense, we can assert that no crucial differences exist between testing the newly developed stochastic nature-inspired population-based algorithms. The better ranked algorithms according to solving the CEC-14 benchmark suite gained a similar rank at solving the CEC-17 benchmark and vice versa. The phrase ‘similar rank’ in the last statement is used intentionally, because the phrase ‘equal rank’ could be used if the Spearman correlation coefficient would achieve the value $r_s = 1$.

Although we obtained values less than one, these are still very high (statistically, they might even not differentiate from one). Consequently, one can observe only little changes in the ranking of a particular algorithm on different ranking lists. For instance, let us take the ranking lists in Table 5c into consideration. There, the iL-SHADE is the best ranked algorithm by solving the CEC-14 benchmark suite, while the jSO is the best ranked by solving CEC-17 benchmark suite. This fact can be explained by the No-Free-Lunch theorem by Wolpert and Macready [42] on the one hand, and the stronger adaptation of both algorithms to the different CEC benchmark suites on the other.

From Figs. 4 and 5, it can be seen that four algorithms achieved the best ranks by optimizing the CEC-13 benchmark suites on functions of dimension $D = 10$. This means that this benchmark suite is an easy task for the state-of-the-art algorithms nowadays. Therefore, this fact should be considered by the developer of the new nature-inspired algorithms searching for the test bed for their new developed algorithms.

5 Conclusion

Looking for a general problem solver that could be capable of solving all classes of problems has been the eternal desire of many researchers in the past. Obviously, we did not pursue this goal that is normally impossible to be reached due to the NFL theorem. Indeed, in the study, we searched for an algorithm that could be able to solve all benchmark suites issued by the CEC special session/competition on real-parameter single-objective optimization in the last decade. In line with this, we selected the better state-of-the-art SHADE-based algorithms, identified their exploring strategies, and incorporated these under the umbrella of the RL mechanism. The result of this integration was the RL-SHADE algorithm that uses a Q-learning algorithm for selecting the successful exploration strategy of discovering the search space.

The purpose of our experimental work was to show that the proposed RL-SHADE algorithm is able to be the first ranked on the different ranking lists created by optimizing the CEC benchmark suites on functions of dimensions $D = 10$ and $D = 30$. The results of the refined study showed that the goal can be achieved with the RL-SHADE algorithm using the *lSHADE* exploration strategy.

There are many directions for the future work, among which, obviously, verification of the RL-SHADE algorithm on the CEC benchmark suites with functions at higher dimensions is the most urgent to be realized. Applying the algorithm to other problem classes remains another direction for the future.

Acknowledgements Iztok Fister Jr. is grateful the Slovenian Research Agency for the financial support under Research Core Funding No. P2-0057. Dušan Fister is grateful the Slovenian Research Agency for the financial support under Research Core Funding No. P5-0027. Iztok Fister thanks the Slovenian Research Agency for the financial support under Research Core Funding No. P2-0042 - Digital twin.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA (2018)
2. Brooks, R.A.: Elephants don't play chess. Robot. Auton. Syst. **6**(1), 3–15 (1990)
3. Piaget. J.: The Psychology of Intelligence/by Jean Piaget. Routledge and Kegan Paul London (1950)
4. Woodworth, R.S., Schlosberg, H., Kling, J.W., Riggs, L.A.: Woodworth and Schlosberg's Experimental Psychology, 3rd edn. Methuen London (1972)
5. Minsky, M.: Steps toward artificial intelligence. Proc IRE. (1961)http://web.media.mit.edu/~minsky/papers/steps.html
6. Holland, J.H.: Adaptation. In: Rosen, R., Snell, F. (eds.) Progress in theoretical biology, vol. 4, pp. 263–293. Academic Press, New York (1976)

7. Klopf, A.: Brain function and adaptive systems: a heterostatic theory. Tech. rep, Air Force Cambridge Research Laboratories, Bedford, MA (1972)
8. Barto, A.G., Sutton, R.S.: Landmark learning: an illustration of associative search. Biol. Cybern. **42** (1981). https://doi.org/10.1007/BF00335152
9. Bellman, R.: Dynamic Programming. Dover Publications (1957)
10. Bellman, R.: A Markovian decision process. J. Math. Mech. **6**(5), 679–684 (1957b). http://www.jstor.org/stable/24900506
11. Das, S., Suganthan, P.N.: Differential evolution: a survey of the state-of-the-art. IEEE Trans. Evol. Comput. **15**(1), 4–31 (2011). https://doi.org/10.1109/TEVC.2010.2059031
12. Darwin, C.: On the origin of species by means of natural selection. Murray, London, or the Preservation of Favored Races in the Struggle for Life (1859)
13. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**(4), 341–359 (1997). https://doi.org/10.1023/A:1008202821328
14. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: a survey. ACM Comput. Surv. **45**(3) (2013). https://doi.org/10.1145/2480741.2480752
15. Fister, I., Suganthan, P., Kamal, S., Al-Marzouki, F., Perc, M., Strnad, D.: Artificial neural network regression as a local search heuristic for ensemble strategies in differential evolution. Nonlinear Dynam. **84**, 895–914 (2016)
16. Wu, G., Mallipeddi, R., Suganthan, P.N.: Ensemble strategies for population-based optimization algorithms–a survey. Swarm Evol. Comput. **44**, 695–711 (2019)
17. Liang, J.J., Qu, B.Y., Suganthan, P.N., Hernández-Díaz, A.G.: Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization. Tech. rep., Technical Report, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Technical Report, Nanyang Technological University, Singapore (2013b)
18. Liang, J.J., Qu, B.Y., Suganthan, P.N.: Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. Tech. rep., Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Technical Report, Nanyang Technological University, Singapore (2013a)
19. Awad, N.H., Ali, M.Z., Liang, J.J., Qu, B.Y., Suganthan, P.N.: Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Bound Constrained Real-Parameter Numerical Optimization. Technical Report, Nanyang Technological University, Singapore, Tech. rep (2016)
20. Fister, I., Brest, J., Iglesias, A., Gálvez, A., Deb, S., Jr.: IF: on selection of a benchmark by determining the algorithms' qualities. IEEE Access **9**, 51166–51178 (2021)
21. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. Glob. Optim. **39**(3), 459–471 (2007). https://doi.org/10.1007/s10898-007-9149-x
22. Yavuz, G., Aydin, D., Stützle, T.: Self-adaptive search equation-based artificial bee colony algorithm on the CEC 2014 benchmark functions. In: 2016 IEEE Congress on Evolutionary Computation. CEC vol. 2016, 1173–1180 (2016). https://doi.org/10.1109/CEC.2016.7743920
23. Qin, A. K., Suganthan, P.N.: Self-adaptive differential evolution algorithm for numerical optimization. In: 2005 IEEE Congress on Evolutionary Computation, IEEE CEC 2005. Proceedings, vol. 2, pp. 1785–1791 (2005). https://doi.org/10.1109/cec.2005.1554904
24. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Trans. Evol. Comput. **10**(6), 646–657 (2006)
25. Tanabe, R., Fukunaga, A.S.: Evaluating the performance of SHADE on CEC 2013 benchmark problems.: IEEE Cong. Evol. Comput. CEC **2013**(1), 1952–1959 (2013). https://doi.org/10.1109/CEC.2013.6557798
26. Tanabe, R., Fukunaga, A.S.: Improving the search performance of SHADE using linear population size reduction. In: Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014, pp. 1658–1665 (2014). https://doi.org/10.1109/CEC.2014.6900380