Deep Reinforcement Learning for Adaptive Parameter Control in Differential Evolution for Multi-Objective Optimization

DOI:
10.1109/SSCI51031.2022.10022227

Document status and date:
Published: 30/01/2023

Document Version:
Accepted manuscript including changes made at the peer-review stage

Deep Reinforcement Learning for Adaptive Parameter Control in Differential Evolution for Multi-Objective Optimization

1st Robbert Reijnen
Dept. of Industrial Engineering
Eindhoven University of Technology
Eindhoven, The Netherlands
r.v.j.reijnen@tue.nl

2nd Yingqian Zhang
Dept. of Industrial Engineering
Eindhoven University of Technology
Eindhoven, The Netherlands
yqzhang@tue.nl

3rd Zaharah Bukhsh
Dept. of Industrial Engineering
Eindhoven University of Technology
Eindhoven, The Netherlands
z.bukhsh@tue.nl

4th Mateusz Guzek
Decision Science and Optimization
Goodyear Innovation Center Luxembourg
Colmar-Berg, Luxembourg
mateusz_guzek@goodyear.com

***Abstract*—Evolutionary algorithms (EA) are efficient population-based stochastic algorithms for solving optimization problems. The performance of EAs largely depends on the configuration of values of parameters that control their search. Previous works studied how to configure EAs, though, there is a lack of a general approach to effectively tune EAs. To fill this gap, this paper presents a consistent, automated approach for tuning and controlling parameterized search of an EA. For this, we propose a deep reinforcement learning (DRL) based approach called 'DRL-APC-DE' for online controlling search parameter values for a multi-objective Differential Evolution algorithm. The proposed method is trained and evaluated on widely adopted multi-objective test problems. The experimental results show that the proposed approach performs competitively to a non-adaptive Differential Evolution algorithm, tuned by grid search on the same range of possible parameter values. Subsequently, the trained algorithms have been applied to unseen multi-objective problems for the adaptive control of parameters. Results show the successful ability of DRL-APC-DE to control parameters for solving these problems, which has the potential to significantly reduce the dependency on parameter tuning for the successful application of EAs.**

***Index Terms*—Evolutionary Algorithms, Deep Reinforcement Learning, Multi-Objective Optimization Problems, Adaptive Parameter Control, Differential Evolution**

I. Introduction

The aim for Combinatorial Optimization Problems (COP) is to identify high-quality solutions in a large space of decision variables. Given the computational complexity of the problems, practical solution approaches often rely on heuristics. Such heuristics are typically fast, but lack the guarantee of finding optimal solutions, and remain dependent on trial-and-error approaches in their development. Resulting high cost of development and owing to recent advancements, the machine learning methodologies are increasingly being used for solving COP (e.g., [3]). With numerous developments in the field, more and better machine learning approaches are published to solve various optimization problems of increasing complexity. Yet, it remains challenging to define end-to-end learning approaches that generalize well to larger and more constrained or difficult problem variants.

Evolutionary Algorithms (EAs) are efficient heuristic-based search techniques that use randomness for solving problems in a stochastic manner. These EA approaches are traditionally applied to solve optimization problems that are hard to solve in polynomial time [1], [13]. The successful EA application requires the formulation of an effective search strategy and its imposed parameters. Common ways for addressing this configuration problem are based on conventions, experience, ad hoc choices, and by experimentation. The first three approaches are typically sub-optimal (as they include little to no knowledge of the specifics of a problem), while experimentation faces limitations as parameters interact, requiring extensive trial-and-error approaches such as grid- or random search, which are often impractical or infeasible for solving real-world problems with computationally expensive evaluation functions.

To address these issues, machine learning approaches have been proposed that find the optimal parameters of EAs, learning the interaction among parameters to optimize objective function (e.g., [4], [6]). With the successful application of Deep Reinforcement Learning (DRL) in a wide range of domains, it was successfully applied to learn to control parameter values for Differential Evolution (DE), an EA proposed by Storn and Price for optimizing real valued functions [25]. With the formulation of this tuning problem as a sequential decision making problem, DRL has been successfully leveraged to control operators and parameters of DEs over various

Supported by the Luxembourg National Research Fund (FNR) (15706426)

generations (e.g., [24], [29] and [30]). With this, DRL does not learn which operator is expected to perform best in the next generation of search, but learns to control operators to optimize the performance of the entire search procedure over various generations of evolutionary search.

In line with the successful applications of the technique, we focus in this work on the application of DRL to online control the scaling factor (f) and crossover probability parameters (cr) that are imposed on the search operators of DE for solving Multi-Objective Optimization Problems (MOOP). This type of problem involves more than one objective functions that simultaneously needs to be optimized. We train and evaluate our method on problems from the DTLZ benchmark of multi-objective problems [11], a problem-set that is widely used for benchmarking multi-objective evolutionary algorithms. In line with the experiments of [9], we focus on applications with limited number of evaluations, as in practice the large number of evaluations is the limiting factor for adoption of evolutionary algorithms. Extensive testing shows that our method outperforms a static DE parametrization approach, where parameters are selected using a grid search. Next to this, we observe competitive results when deploying a trained DRL model to unseen problem instances, comparing to the static DE approach tuned for each such instance.

The contributions of this work are summarized as follows:

- To the best of our knowledge, we are the first to propose a DRL adaptive parameter control approach for solving Multi-Objective Optimization Problems.
- Our experiment results show that the proposed approach is problem-independent, implying its potential to be applied to other Multi-Objective Optimization Problems without additional training or tuning.

II. Background and Related work

The formulation of an EA requires setting values for its parameters, which determine whether the performed search will be efficient or not. Common ways for addressing this tuning problem are based on conventions, experience, ad hoc choices, and experimentation. The main drawback of the first three approaches is that they typically result in a sub-optimal configuration, while experimentation is typically computationally expensive for complex optimization problems. Additionally, as the execution of an EA is an intrinsically dynamic, adaptive process, different parameters might be the most effective at different stages of an evolutionary process.

A. Adaptive Parameter Control of Evolutionary Algorithms

To address the issues of setting fixed values of parameters of EAs that remain static for the entire search process, the work of [12] defines three classes of adaptive parameter control in EAs, in which the setting of values is performed during a run: Deterministic Parameter Control, Self-Adaptive Parameter Control, and Adaptive Parameter Control. In Deterministic Parameter Control, the values of parameter are altered by a deterministic rule. Usually, a time varying schedule is used to update the parameters at specified intervals of the evolutionary search (e.g., [28]). In Self-Adaptive Parameter Control, the control of parameters is implemented in the evolutionary search. Here, the parameters that are online adapted are coded into the chromosomes that undergo recombination and mutation (e.g., [18]). Adaptive Parameter Control takes place when there is some form of feedback from the search used to determine the directions and/or magnitude of changes to the parameters. Approaches in this category typically involve a credit assignment scheme for rewarding parameters based on the impact of their recent applications on the search process, and an operator selection mechanism for deciding which parameters to apply, based on the credits assigned (e.g., [15], [21], [34]).

B. Parameter Configuration and Control of Differential Evolution

Differential Evolution (DE) is an EA proposed by Storn and Price for optimizing real valued functions [27]. The distinguishing feature of DE is its *differential mutation*. Given a population of candidate solutions in $\mathbb{R}^n$, a solution $\bar{x}'$ is produced by adding a permutation vector to an existing solution: $\bar{x}' = \bar{x} + \bar{p}$. This perturbation vector $\bar{p}$ is the scaled vector difference of two other randomly chosen solutions from the solution: $\bar{p} = f \cdot (\bar{y} - \bar{z})$, where the scaling factor f controls the rate at which the population evolves. After mutation, a crossover operator is applied to combine the new candidate solution created by mutation with another solution from the current population. This reproduction operator is typically a uniform crossover, and is subject to the crossover probability parameter cr, which defines the probability of copying the value of the gene from the candidate solution to the corresponding gene of the selected solution from the current population.

As in other EAs, the performance of DE depends on its algorithm control parameter values and its mutation strategy. The effect of the control parameters of DE is widely studied, and various ranges of values are suggested for setting the population size (NP), scaling factor (f) and crossover probability rate (cr). Storn and Price [26] propose NP = 10D, f $\in$ [0.5, 1] and cr $\in$ [0.8, 1], Gämperle et al. [14] suggest a population size between NP = 3D and NP = 8D, with a scaling factor f = 0.6 and crossover rate is cr $\in$ [0.3, 0.9] and Ronkkonen et al. [20] recommend NP = 2D and NP = 40D, f $\in$ (0.4, 0.95] and cr $\in$ (0, 0.2) when the objective function is separable. Further, Zielinski et al. [36] report best obtained results with a scaling factor f $\geq$ 0.6 and cr $\geq$ 0.6. As it can be observed, the suggested parameter combinations to use for DE are different from each other. Therefore, and according to the no free lunch theorem [33], there is no fixed parameter configuration that is generally used by practitioners, or that consistently outperforms others on wide range of problems. Therefore, trial-and-error or experimentation approaches are commonly used for the tuning of parameters, which is either non-comprehensive or costly. Setting fixed parameters for the whole evolutionary process duration has also intrinsic limitations, as different parameter settings may be required in order to achieve optimal performance at different stages of evolution [12].

Various extensions have been proposed to the DE algorithm that make use of an adaptive approach to control parameters during the search [8]. These approaches differ in terms of what they control and employed approach. Popular approaches are *jDE* [5] using a self-adaptive scheme, *SaDE* [19] assigns evolutionary strategies and control parameters based on probabilities that have been learned from the experience from previous generations, *JADE* [35] utilizes a Cauchy and normal distribution to automatically update control parameters, and in *SHADE* and *LSHADE* [31] parameters are updated based on the historical memory of successful solutions. A complete overview of these approaches can be found in [8].

C. Machine Learning Approaches for Differential Evolution Parameter Configurations and Control

Differential Evolution with adaptive control achieved good performance, yet make the programming fairly complex and run the risk of increasing the number of function evaluations [17]. Given this, and the notable successes in machine learning led practitioners to develop learning-based approaches for learning to tune the parameters of the Differential Evolution algorithm. Examples of recent machine learning applications are based on Bayesian hyper parameter optimization [4] and an Artificial Neural Network for predicting optimal parameter values [6].

Besides these promising approaches for tuning parameters based on popular machine learning methods, various research has recently been devoted on applying Deep Reinforcement Learning for both the tuning and the control of mutation operators and/or control parameters that are imposed in each generation of search of Differential Evolution. Examples are the work of [24] and [30] that use a Double Deep Q-Network (DDQN) agent for selecting different mutation operators within DE, and [29] that proposes a Policy Gradient method as DRL algorithm. All above-mentioned approaches are focused on single-objective optimization. The authors of [2] develop a DRL approach to solve a multi-objective online order batching Problem. However, in their approach, the DRL agent learns a policy that decides directly the decisions for the underlying optimization problem. Accordingly, the multiple objectives are modelled as reward functions.

III. Deep Reinforcement Learning for Adaptive Parameter Control for Multi-objective Optimization

Our proposed approach distinguishes itself from the prior discussed DRL-based approaches in two main aspects. First, we apply Deep Reinforcement Learning for the adaptive control of parameter values imposed on the mutation operators of the Differential Evolution algorithm. With this, we control the scaling factor f and crossover probability parameter cr imposed on the perturbation vector and the reproduction operators for the generation of solutions to optimize how new solutions are generated throughout the evolutionary search process. Second, we apply our approach to solve Multi-Objective Optimization Problems (MOOP), a branch of problems involving two or more objective functions. These types of problems are mathematically expressed as follows:

$$\begin{array}{ll} \text{Minimize} & F(x)=\{f_1(x), f_2(x), \ldots, f_m(x)\} \\ \text{Subject to} & g(x) \leq 0 \end{array} \tag{1}$$

where x is the vector of decision variables, $f_i(x)$ is the ith objective function, and $g(x)$ is the constraint function. The solution of MOOP is generally expressed as a set of Pareto-optimal solutions, which are the best trade-offs between objectives. A solution $x*$ is Pareto optimal if it is feasible and there exists no other solution x such that $f_i(x) \leq f_i(x^\star)$ and $f_j(x) < f_j(x^\star)$, for at least one objective $i, j \in \{1, 2, \ldots, m\}$ with $i \neq j$.

A. Solution Framework

We propose a Deep Reinforcement Learning based solution for adaptive parameter control in Differential Evolution called 'DRL-APC-DE'. In this approach, a decision-making policy is trained for tuning the parameters that control the evolutionary operators during the search process. This policy can be defined as the agent's way of behaving at a given time, mapping the states of the environment to actions to be taken. This enables the dynamic selection of parameters that are used in the different stages of the evolutionary search process.

The Differential Evolution algorithm is formulated with a 'DE/Rand/1' mutation strategy, as originally described in [25] and explained in II-B, and with a random uniform crossover strategy. To effectively deal with MOOP, we use an NSGA2 selection procedure, as described in [10]. A action-making policy π_θ is trained with Deep Reinforcement Learning algorithm to decide what scaling factor f is used in the mutation strategy and what crossover rate *cr* is applied in the cross-over operator. The values are repeatedly selected in every generation of the search, based on the state of the search. The pseudo-code of the described approach can be found in Algorithm 1 and the training of the DRL policy is described in section IV-A.

B. Markov Decision Process formulation

A Markov Decision Process (MDP) is formulated to create an environment where agents can learn a strategy for making these sequential decisions. An MDP framework to the problem is defined as a tuple $< S, A, R, P >$, where *S* represents the set of states, *A* the set of actions, *R* the reward function and *P* the state transition probability function. The state transition occurs after the execution of an action by the agent, such that the state S_t of environment at time step *t* transforms to S_{t+1} at time step *t+1*, and the agent receives a reward according to the reward function *R*. Based on this framework, we define the decision-making problem for adaptive parameter control in Differential Evolution as follows:

State space S - The state space is formulated as a one-dimensional vector in which the DRL agent is provided with information regarding the current status of the evolutionary

Algorithm 1 Deep Reinforcement Learning based Adaptive Parameter Control for Differential Evolution (DRL-APC-DE)

```
Initialize trained action-making policy $\pi_\theta$
Generate initial population of size NP
while termination condition is not satisfied do
    Select $cr$ and $f$ with policy $\pi_\theta$ based on state $s$
    for For each individual $j$ in the Population do
        Generate random integers, $r_1, r_2, r_3 \in (1, NP)$, with $r_1 \neq r_2 \neq r_3 \neq j$
        for For each parameter $i$ in individual $j$ do
            if if rand (0,1) $< cr$ then
                child $x'_{i,j} = x_{i,r_3} + f \times (x_{i,r_1} - x_{i,r_2})$
            else
                child $x'_{i,j} = x_{i,j}$
    Evaluate fitness of each child $j$ in the Population
    Select individuals to population with NSGA2 selection
return best found solutions
```

search toward the best set of solutions. The state space consists of 5 metrics: 1) Hole Relative Size, 2) spacing, 3) the number of cardinality points, 4) stagcount, and 5) the remaining budget. Hole Relative Size is used to identify the largest hole in a set of solution [7], and is given by:

$$(1/\bar{d}) \max_{i=1,2,\ldots,|S|} d_i, \tag{2}$$

where $d_i = \min_{(s_i,s_j)\in S, s_i\neq s_j} \|F(s_i) - F(s_j)\|_1$ is the l_1 distance between point $s_i \in S$ and its closest neighbor, and $\bar{d}$ the mean of the d_i. Spacing measures the distribution of individuals in the Pareto front [22] with:

$$\sqrt{\frac{1}{|S|-1}\sum_{i=1}^{|S|}\left(\bar{d}-d_i\right)^2}. \tag{3}$$

The number of cardinality points provides the total number of individual solutions present in the solution set, the remaining budget indicates how many generations of search are left in the search budget, and stagcount tracks the number of generations without improving the hypervolume (HV). The hypervolume metric is the volume of the objective space dominated by the Pareto front approximation [37]. All features have been normalized given the bounds of the initialized solution objectives, allowing for the deployment to different problems of different objective dimensions.

Action space A - The action space contains actions that set parameters of the DE algorithm. Each action is associated with possible parameter combinations that an agent can select for the DE to generate a new generation of solutions and are composed of different combinations of the scaling factor (f) and crossover rate (cr) parameters. The used parameters values that shape these actions are defined based on the recommended values from state of the art on the DE tuning [14], [20], [26], [36]. Table I provides an overview of the formulated action space.

TABLE I
DEEP REINFORCEMENT LEARNING ACTION SPACE

	Scaling Factor F	Crossover Rate CR
Action 1	0.6	0.5
Action 2	0.6	0.7
Action 3	0.6	0.9
Action 4	0.6	1.0
Action 5	0.8	0.5
Action 6	0.8	0.7
Action 7	0.8	0.9
Action 8	0.8	1.0
Action 9	0.95	0.5
Action 10	0.95	0.7
Action 11	0.95	0.9
Action 12	0.95	1.0

State Transition Function P - A selected action is used to parameterize the creation of a new generation of solutions by the DE algorithm. This action is imposed on the DE algorithm for one generation, which corresponds to one time step t of the DRL agent, transforming state S_t to S_{t+1}. This transition is assumed to satisfy the Markov property, as such that the transition probability only depends on state S_t and action A_t taken. In this work, the agent has no prior knowledge of this transition function and learns it by interacting with the environment.

Reward function R - A reward function is formulated to train DRL to select the best actions based on the state S of the evolutionary search. For this, we formulated a reward function for learning to take actions that close the gap between the initialized population and the true Pareto front, using a Generational Distance (GD) Performance metric [38]. This metric measures the distance from solutions $A = \{a_1, a_2, \ldots, a_{|A|}\}$ to the Pareto front $Z = \{z_1, z_2, \ldots, z_{|Z|}\}$, computed as:

$$\frac{1}{|A|}\left(\sum_{i=1}^{|A|} d_i^p\right)^{1/p} \tag{4}$$

where d_i represents the Euclidean distance $(\mathrm{p} = 2)$ from a_i to its nearest reference point in Z. With this, our reward function is formulated as follows:

$$GD = \frac{GD_{init} - GD_{final}}{GD_{init}} * 100\%, \tag{5}$$

where GD_{init} is the GD of initialized population and GD_{final} the GD of the finally found solution set. The reward is provided to the Deep Reinforcement Learning agent after the last generation of evolutionary search as an episodic reward to train the agent. This reward guides the agent in learning how to select actions that control the evolutionary search process to improve its final results.

C. Proximal Policy Optimization

We select the Proximal Policy Optimization (PPO) algorithm [23] for learning the action-making policies. PPO is a policy gradient DRL method that uses a probability ratio between two policies to maximize improvement without the risk of performance collapse. For this, it utilizes clipping to define how far away a new policy is allowed to deviate from the old one, preventing severe performance drops from occurring. Given its computational inexpensiveness, ease of implementation, and effectiveness for learning a wide range of sequential decision-making tasks, it is regarded as one of the most successful DRL algorithms and is widely used by practitioners.

IV. Experimentation

We test the performances of our proposed 'DRL-APC-DE' approach by evaluating its ability to solve both problems explored during the DRL agent's training and problems not included in this training. The performances of our proposed approach are compared with a grid-search tuned DE algorithm, tuned for solving individual problems. With this, we aim to evaluate to what extent our proposed approach can be used as an alternative to a classical grid-search approach and how DRL can learn a generalizable policy that can online control parameters of different problems. In this section, we first describe the training details of the DRL models and the configurations of the problems used for training and testing. Correspondingly, we describe the results of the experiments.

A. *Training*

We train four adaptive parameter control policies, one for each of the first four problems of the scalable DTLZ multi-objective problem set presented in [11]. This DTLZ is a test suite for multi-objective problems that are scalable to any number of objectives and decision variables. The objectives of the selected problems are non-separable, making it impossible to optimize them one at a time to find global optima. The number of decision variables of the different DTLZ problems is $M + k - 1$, where M is the number of objectives and k a parameter. For DTLZ1 we use $k = 5$ and for DTLZ2-4 we set $k = 10$. With three objectives, the number of variables is set to 7 for DTLZ1 and 12 for DTLZ2-4.

The training was performed for 250.000 steps, similar to the computational budget provided for a grid search procedure for optimizing the setup of the Differential Evolution algorithm with fixed control parameters. With this grid search, we attempt to compute the optimum values of control parameters by an exhaustive search with the same parameter options as the DRL approach. In each training episode, 100 generations of evolutionary search are performed with a population of 20 individuals. The training was performed on a Processor Intel(R) Core(TM) i7-6920HQ CPU @ 2.90GHz with 8.0GB of RAM with ten parallel environments, which took 45 minutes. The model parameters selected for training are described in [23]. The training traces are displayed in Figure 1, showing the mean episodic reward obtained during the training of the different models.

Figure 1 presents the average reward of different action-making agents trained on the four DTLZ problems. The agents learn a policy that learns to interact with the environment to obtain higher rewards from the rewards function, suggesting better convergence of solutions towards the true Pareto front.

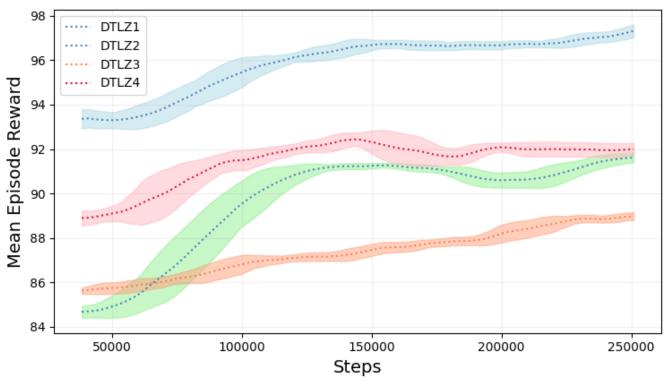

Fig. 1. Rolling mean and std. deviation of training episode reward over time for DTLZ1-4 training problems

In each case, the final policy obtained is better than its initial state, which corresponds to the random action policy. From the learning curves can also be observed that a random action-making policy for the DTLZ1 is most effective. Further, the policy trained for DTLZ2 managed to gain the most improvement compared to the random strategy they were initialized with. There is also a visible difference in the difficulty of the problems, e.g., the mean reward of the initialized policy for DTLZ1 is larger than trained policies for other algorithms.

B. *Methodology of Performance Evaluation on DTLZ*

We compare the DRL-based adaptive parameter control approach for Differential Evolution (DRL-APC-DE) with the non-adaptive, tuned Differential Evolution configuration. Both approaches are provided with the same parameter ranges, and an equal computational budget is used for training the DRL-APC-DE policies and tuning the static DE algorithm with a grid search. Due to the stochastic nature of the algorithms, each of them is run 100 times to solve each problem multiple times. To ensure a fair comparison, we use the same seeds, ensuring the initial populations to be the same.

Results are compared using three performance metrics: Hypervolume (HV), Generational Distance (GD) and Inverted Generational Distance (IGD) [16]. The IGD metric inverts the GD by measuring the distance of the Pareto front to the closest point in the solution population. Given the different dimensions of the optimization problems, HV is computed using the true ideal point [0,0,0] as the reference, and the GD and IGD are computed with the true Pareto fronts of the problems. Lower values are better for each of the performance metrics. Correspondingly, the differences of means (i.e., arithmetical averages) of the performance metrics of the two algorithms are tested for statistical significance with the Wilcoxon signed-rank test $(p < .05)$ [32]. The cells in the results tables are shaded gray when the difference in mean results is statistically significant.

C. Results of DRL-APC-DE Tuned and Applied on a Single Problem

As a first experiment, we evaluate the results of the grid-search tuned Differential Evolution with the DRL-APC-DE algorithm trained on the same problem. The results of the experiment are presented in Table II.

It can be observed that the DRL-based approach reaches competitive performance for most of the objectives, and for some problems and performance metrics, it outperforms the tuned DE approach. Comparing the performance of GD, the metric used for the training and tuning of the algorithms, we find that the proposed DRL-APC-DE approach is on average better in converging to the true Pareto front of the nonseparable objective functions of DTLZ1 and 3, while it reaches similar distances towards the true Pareto front for DTLZ2 and 4. This also holds for the IGD metric for DTLZ1, 2, and 4, finding similar distances between the true Pareto front and the found solutions. For DTLZ3 we observe lower HV and IGD values for the tuned DE, but lower GD for the DRL-APC-DE approach.

It demonstrates the successful application of our approach to optimize its objective to decrease the distance towards the true Pareto front (GD), which is inadvertently reached by sacrificing the performance metrics that were not rewarded in training. It is a common challenge of applying RL, which could be mitigated by future work on multi-objective reward function that would result in more balanced search behavior.

TABLE II
PERFORMANCE OF DE AND DRL-APC-DE, TUNED AND TRAINED ON THE SAME PROBLEM

	Tuned DE (DTLZ1)			DRL-APC-DE (DTLZ1)		
DTLZ1	HV	GD	IGD	HV	GD	IGD
mean	9.75E+01	5.27	4.00	9.55E+01	**5.21**	**3.75**
min	6.45E-01	1.14	0.92	**5.50E-01**	1.18	0.92
max	2.12E+03	**21.16**	**16.18**	3.06E+03	29.38	19.97
std	2.71E+02	**3.85**	2.91	2.75E+02	4.31	2.87
	Tuned DE (DTLZ2)			DRL-APC-DE (DTLZ2)		
DTLZ2	HV	GD	IGD	HV	GD	IGD
mean	0.41	0.06	0.17	0.42	0.06	0.17
min	0.34	0.04	0.14	0.32	0.05	0.14
max	0.48	0.09	0.20	0.50	0.09	0.21
std	0.03	0.01	0.01	0.03	0.01	0.01
	Tuned DE (DTLZ3)			DRL-APC-DE (DTLZ3)		
DTLZ3	HV	GD	IGD	HV	GD	IGD
mean	**2.22E+05**	168.41	**54.26**	3.18E+05	**109.30**	73.50
min	2.18E-27	43.99	9.58	2.05E-12	**18.41**	0.11
max	3.89E+06	240.42	206.22	1.84E+06	**182.65**	**147.47**
std	6.21E+05	33.52	48.92	**3.17E+05**	32.94	**25.03**
	Tuned DE (DTLZ4)			DRL-APC-DE (DTLZ4)		
DTLZ4	HV	GD	IGD	HV	GD	IGD
mean	0.33	0.04	0.18	0.32	0.05	0.19
min	0.00	0.02	0.14	0.00	0.02	0.14
max	0.46	0.08	0.54	0.47	0.09	0.35
std	0.12	0.01	0.07	0.14	0.02	0.05

D. Results of DRL-APC-DE Applied to Other Problems

In the second experiment, we aim to test how well a trained DRL policy can be used for adaptive control of parameter values to solve other MOOP problems (i.e., problems that are not presented to it during the training). During the training of DRL models, we found that the model trained on the DTLZ2 problem achieves the largest improvements when compared to the random decision-making strategy the learning was initialized with. Therefore, to evaluate the ability to solve unseen problems, we evaluate the performance of this model (the one trained on DTLZ2) to solve DTLZ1, 3, and 4. The methodology for comparison is the same as for the previous experiment. The results of this experiment are shown in Table III.

TABLE III
COMPARISON OF DE TUNED BY GRID-SEARCH ON THE PROBLEM VS. DRL-APC-DE TRAINED ON THE DIFFERENT DTLZ2 PROBLEM

	Tuned DE (DTLZ1)			DRL-APC-DE (DTLZ2)		
DTLZ1	HV	GD	IGD	HV	GD	IGD
mean	9.75E+01	5.27	4.00	**6.32E+01**	**4.42**	**3.38**
min	6.45E-01	1.14	0.92	**5.50E-01**	1.18	0.92
max	2.12E+03	21.16	16.18	**6.24E+02**	**15.82**	**13.12**
std	2.71E+02	3.85	2.91	**1.35E+02**	**3.61**	**2.75**
	Tuned DE (DTLZ3)			DRL-APC-DE (DTLZ2)		
DTLZ3	HV	GD	IGD	HV	GD	IGD
mean	**2.22E+05**	168.41	**54.26**	2.42E+05	**101.62**	64.53
min	2.18E-27	43.99	**9.58**	**1.78E-11**	**18.41**	10.24
max	3.89E+06	240.42	206.22	1.84E+06	**172.56**	**118.42**
std	6.21E+05	33.52	48.92	2.71E+05	32.46	**22.61**
	Tuned DE (DTLZ4)			DRL-APC-DE (DTLZ2)		
DTLZ4	HV	GD	IGD	HV	GD	IGD
mean	0.33	0.04	0.18	0.33	0.05	0.18
min	0.00	0.02	0.14	0.00	0.02	0.14
max	0.46	0.08	0.54	0.45	0.08	0.34
std	0.12	0.01	0.07	0.12	0.01	0.04

The proposed DRL-APC-DE approach, trained with only the DTLZ2 problem, can adaptively control parameter values used in the search operators to solve problems not presented to it during training. Comparing with a DE algorithm tuned with grid search, we observe that DRL-APC-DE is better for DTLZ1 and 3 and reaches similar performances for DTLZ4. Interestingly, the scores achieved by DRL-APC-DE trained on DTLZ2 and applied to DTLZ1 and 3 are better than those achieved by DRL-APC-DE trained on the same problem or those generated by the grid-search tuned DE. This demonstrates that it is possible to learn a meaningful search control system that is not problem-specific. From this, we identify the potential of Deep Reinforcement Learning for training an adaptive parameter control policy to control the parameter values of evolutionary algorithms for solving MOOP. This potentially reduces the need for expensive experimentation for the successful deployment of evolutionary search.

V. CONCLUSION AND FUTURE WORK

In this work we propose DRL-APC-DE, a Deep Reinforcement Learning based approach for online parameter control of Differential Evolution for solving multi-objective optimization problems. With this, we aim to learn to select parameters used in each generation of evolutionary search to generate new solutions for improving the overall search performance and efficiency. The approach can be considered an alternative to classical approaches (e.g., grid search) that tune evolutionary algorithms before deployment or can be used as an alternative for (self-) adaptive parameter control methods for controlling parameters during the evolutionary search.

The proposed method has been compared with a Differential Evolution that was tuned with a grid search. The Deep Reinforcement Learning and the tuned Differential Evolution approach were provided with an equal computational budget for the training and tuning. Results indicate that the proposed method compares competitively with the tuned Differential Evolution algorithm.

Subsequently, we applied the adaptive parameter control policy to control the parameters of problems not presented during its training. Results show that DRL-APC-DE can outperform the tuned approach of the static Differential Evolution algorithm, indicating that the proposed approach can learn shared characteristics of a class of problems and strategies to control the search process efficiently. In turn, this may significantly impact the reduction or elimination of the need to perform testing and tuning procedures per problem or instance.

Given that our approach is formulated problem- and algorithm-independent, we want future work to explore if our approach can be used to adaptively control parameters of other Evolutionary Algorithms to solve more complex problems. Further, we are interested in controlling more parameters.

References

[1] Thomas Back, David B. Fogel, and Zbigniew Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., GBR, 1st edition, 1997.

[2] Martijn Beeks, Reza Refaei Afshar, Yingqian Zhang, Remco Dijkman, Claudy van Dorst, and Stijn de Looijer. Deep reinforcement learning for a multi-objective online order batching problem. In *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022*, volume 32, pages 435–443. Association for the Advancement of Artificial Intelligence (AAAI), June 2022.

[3] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[4] Subhodip Biswas, Debanjan Saha, Shuvodeep De, Adam D Cobb, Swagatam Das, and Brian A Jalaian. Improving differential evolution through bayesian hyperparameter optimization. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 832–840. IEEE, 2021.

[5] Janez Brest, Sao Greiner, Borko Boskovic, Marjan Mernik, and Viljem Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE transactions on evolutionary computation*, 10(6):646–657, 2006.

[6] Manu Centeno-Telleria, Ekaitz Zulueta, Unai Fernandez-Gamiz, Daniel Teso-Fz-Betoño, and Adrián Teso-Fz-Betoño. Differential evolution optimal parameters tuning with artificial neural network. *Mathematics*, 9(4):427, 2021.

[7] Yann Collette and Patrick Siarry. Three new metrics to measure the convergence of metaheuristics towards the pareto frontier and the aesthetic of a set of solutions in biobjective optimization. *Computers & operations research*, 32(4):773–792, 2005.

[8] Swagatam Das, Sankha Subhra Mullick, and Ponnuthurai N Suganthan. Recent advances in differential evolution–an updated survey. *Swarm and evolutionary computation*, 27:1–30, 2016.

[9] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE transactions on evolutionary computation*, 18(4):577–601, 2013.

[10] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[11] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable test problems for evolutionary multiobjective optimization. In *Evolutionary multiobjective optimization*, pages 105–145. Springer, 2005.

[12] Agoston E Eiben, Zbigniew Michalewicz, Marc Schoenauer, and James E Smith. Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*, pages 19–46. Springer, 2007.

[13] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[14] Roger Gämperle, Sibylle D Müller, and Petros Koumoutsakos. A parameter study for differential evolution. *Advances in intelligent systems, fuzzy systems, evolutionary computation*, 10(10):293–298, 2002.

[15] Wenyin Gong, Álvaro Fialho, Zhihua Cai, and Hui Li. Adaptive strategy selection in differential evolution for numerical optimization: an empirical study. *Information Sciences*, 181(24):5364–5386, 2011.

[16] Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. Modified distance calculation in generational distance and inverted generational distance. In *International conference on evolutionary multi-criterion optimization*, pages 110–125. Springer, 2015.

[17] Sk Minhazul Islam, Swagatam Das, Saurav Ghosh, Subhrajit Roy, and Ponnuthurai Nagaratnam Suganthan. An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):482–500, 2011.

[18] Zbigniew Michalewicz, David B Fogel, and Thomas Bèack. *Evolutionary Computation. Vol. 2, Advanced Algorithms and Operators*. Taylor & Francis, 2000.

[19] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2):398–417, 2008.

[20] Jani Ronkkonen, Saku Kukkonen, and Kenneth V Price. Real-parameter optimization with differential evolution. In *2005 IEEE congress on evolutionary computation*, volume 1, pages 506–513. IEEE, 2005.

[21] Ruhul A Sarker, Saber M Elsayed, and Tapabrata Ray. Differential evolution with dynamic parameters selection for optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(5):689–707, 2013.

[22] Jason Ramon Schott. *Fault tolerant design using single and multicriteria genetic algorithm optimization*. PhD thesis, Massachusetts Institute of Technology, 1995.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[24] Mudita Sharma, Alexandros Komninos, Manuel López-Ibáñez, and Dimitar Kazakov. Deep reinforcement learning based parameter control in differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 709–717, 2019.

[25] Rainer Storn. Differrential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces. *Technical report, International Computer Science Institute*, 11, 1995.

[26] Rainer Storn and Kenneth Price. Minimizing the real functions of the icec'96 contest by differential evolution. In *Proceedings of IEEE international conference on evolutionary computation*, pages 842–844. IEEE, 1996.

[27] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[28] Gaoji Sun, Geni Xu, and Nan Jiang. A simple differential evolution with time-varying strategy for continuous optimization. *Soft Computing*, 24(4):2727–2747, 2020.

[29] Jianyong Sun, Xin Liu, Thomas Bäck, and Zongben Xu. Learning adaptive differential evolution algorithm from optimization experiences by policy gradient. *IEEE Transactions on Evolutionary Computation*, 25(4):666–680, 2021.

[30] Zhiping Tan and Kangshun Li. Differential evolution with mixed mutation strategy based on deep reinforcement learning. *Applied Soft Computing*, 111:107678, 2021.

[31] Ryoji Tanabe and Alex S Fukunaga. Improving the search performance of shade using linear population size reduction. In *2014 IEEE congress on evolutionary computation (CEC)*, pages 1658–1665. IEEE, 2014.

[32] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[33] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[34] Guohua Wu, Rammohan Mallipeddi, Ponnuthurai Nagaratnam Suganthan, Rui Wang, and Huangke Chen. Differential evolution with multi-

population based ensemble of mutation strategies. *Information Sciences*, 329:329–345, 2016.

[35] Jingqiao Zhang and Arthur C Sanderson. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation*, 13(5):945–958, 2009.

[36] Karin Zielinski, Petra Weitkemper, Rainer Laur, and K-D Kammeyer. Parameter study for differential evolution using a power allocation problem including interference cancellation. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1857–1864. IEEE, 2006.

[37] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International conference on parallel problem solving from nature*, pages 292–301. Springer, 1998.

[38] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.