Reinforced Online Parameter Adaptation Method for Population-based Metaheuristics

Vasileios A. Tatsis
Department of Computer Science and Engineering
University of Ioannina
GR-45110 Ioannina, Greece
vtatsis@cse.uoi.gr

Konstantinos E. Parsopoulos
Department of Computer Science and Engineering
University of Ioannina
GR-45110 Ioannina, Greece
kostasp@cse.uoi.gr

***Abstract*—Online parameter control has proved to offer significant performance boost in metaheuristics. Common parameterization methods (tuners) typically involve their own parameters that affect their dynamics. Although the potential of completely removing the user out of the tuning cycle is questionable, taking more informative autonomous decisions is highly desirable as part of the general pursuit of artificial intelligence. Towards this goal, we propose an enhanced version of a recently proposed gradient-based parameter adaptation method that can be used by any population-based metaheuristic. The enhancement lies in the use of reinforcement learning to adapt the tuner's parameters, offering overall performance gains. The REINFORCE family of algorithms is considered in the present paper as the proof-of-concept approach. The proposed method is demonstrated on the differential evolution algorithm, which stands among the most popular metaheuristics. Experimental assessment on an established test suite offers promising results, also verifying previous evidence regarding the tuner's robustness and simplicity.**

***Index Terms*—parameter tuning, metaheuristics, differential evolution, reinforcement learning**

I. INTRODUCTION

Metaheuristics have been long established as state-of-the-art solvers for demanding optimization problems. Their increasing popularity lies in their ability to detect (sub-)optimal solutions in manageable time, while requiring only trivial information on the problem at hand. Early works such as [9] have verified the dependence of metaheuristics' performance on their proper parameterization. To this end, parameter tuning has become an essential part of the relevant practice, which is frequently based on resource-consuming, trial-and-error procedures. A variety of tuning methods has been developed to support the user in the laborious parameter setting phase, which is conducted either offline or online.

Early parameter tuning methods operate offline. They are typically based on a preprocessing phase, which aims at identifying proper parameter values through preliminary experimentation of the metaheuristic on the studied problem or similar ones. Statistical analysis of the algorithm's performance can offer insights on parameter values that, on average, promote nice performance on problems of similar type and dimension. Among the most popular methods of this type are the design of experiments [2], F-race [4], and paramILS [17]. Nevertheless, the reusability of parameter values comes at the cost of resource-intensive procedures that may become more demanding than solving the studied problem itself.

In contrast to offline tuning, online parameter adaptation (also known as parameter control) methods are not based on preprocessing. Instead, they take advantage of online feedback of the algorithm's performance in order to adapt its parameters during its run. Obviously, the outcome is not reusable because it is tightly related to the specific performance profile of the metaheuristic each time. Nevertheless, dynamically modifying the parameters based on current performance has proved to offer efficiency and effectiveness to the metaheuristic. Most of the online parameter adaptation methods consist of *ad hoc* procedures suited to the specific metaheuristic. The reader is referred to [9], [10], for a comprehensive presentation.

Recently, an algorithm-independent parameter adaptation method was proposed in [33]. The advantage of the method is its generality, which renders it applicable to any population-based algorithm since it is not tightly related to the algorithm itself [32], [34], [35]. The core mechanism of this method lies on grid search in the discretized parameter space. The method was successfully demonstrated on two population-based metaheuristics, namely differential evolution and particle swarm optimization, with significant results [34], while its sensitivity was preliminarily studied in [36].

The method was further evolved by substituting the parameter grid search with an approximate gradient approach with line search [37], while maintaining some of its previous features to handle also categorical parameters. The method was assessed on established test suites, exhibiting competitive performance against dominant state-of-the-art algorithms [37].

Parameter control methods such as the ones above involve also a few parameters on their own, which influence their behavior. Thus, on the one hand, the main goal of parameter controllers is to ameliorate the burden of tuning the algorithm while, on the other hand, the controller needs proper setting itself. Although this necessity may seem somehow counter-intuitive, it can offer significant gains if the controller is less sensitive to its parameters than the tuned algorithm. The user's experience on the controller is proved to be beneficial for

978-1-7281-2547-3/20/$31.00 ©2020 European Union

Authorized licensed use limited to: University of Ioannina. Downloaded on April 27,2021 at 09:25:34 UTC from IEEE Xplore. Restrictions apply.

this purpose. Nevertheless, making more informative decisions with limited computational cost would be highly desirable.

In the era of artificial intelligence, this goal can be achieved by using online learning methodologies. Following this line, the present paper proposes an enhanced version of the aforementioned gradient-based parameter adaptation method, where reinforcement learning is incorporated into its core mechanism in order to adapt the parameters of the control method itself. The established REINFORCE algorithm is considered for this purpose. The derived method is demonstrated on the differential evolution algorithm, which was the main application algorithm of the method in previous works. Experimental assessment on an established test suite serves as proof-of-concept, verifying its competitive performance against its precursor as well as other tuned algorithms.

The rest of the paper is organized as follows: background information is provided in Section II. The proposed method is thoroughly described in Section III, and implementation details for the differential evolution algorithm are exposed in Section IV. Experimental results are presented in Section V, and the paper concludes in Section VI.

II. Background Information

Brief descriptions of the gradient-based parameter adaptation method and the differential evolution algorithm are provided in the following paragraphs for completeness reason. We assume that the algorithms are applied on the n-dimensional bound-constrained minimization problem:

$$\min_{x \in X} f(x), \tag{1}$$

where $X \subset \mathbb{R}^n$ is a continuous search space.

A. Differential Evolution

Differential evolution was introduced by Storn and Price [27] and it has been long established as one of the most popular metaheuristics for continuous optimization tasks. It employs a population of search points, $P = \{x_1, x_2, \ldots, x_N\}$, where each search point is a candidate solution in X:

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{in}) \in X, \quad i \in I \triangleq \{1, 2, \ldots, N\}.$$

The population is iteratively updated according to three evolutionary operators: mutation, crossover, and selection. Mutation and crossover are responsible for the production of new candidate solutions by combining existing ones, while selection retains the best candidate solutions in the population.

Each iteration t of the algorithm starts with mutation. For each individual $x_i^{(t)}$ of the current population, a new vector $u_i^{(t+1)}$ is produced following a mutation operator. There is a variety of mutation operators [11], [33]. The most common ones are as follows:

DE/Best/1:

$$u_i^{(t+1)} = x_{g_t}^{(t)} + F\left(x_{r_1}^{(t)} - x_{r_2}^{(t)}\right), \tag{2}$$

DE/Rand/1:

$$u_i^{(t+1)} = x_{r_1}^{(t)} + F\left(x_{r_2}^{(t)} - x_{r_3}^{(t)}\right), \tag{3}$$

where F is a user-defined parameter of the algorithm, also called the scale factor [7]. The index g_t denotes the best member of the current population in terms of function value:

$$g_t = \arg\min_{i \in I} \left\{ f\left(x_i^{(t)}\right) \right\},$$

while indices r_k are taken randomly in I such that:

$$r_k \neq r_l \neq i, \quad \forall\, k, l.$$

Crossover implements genetic information fusion between the new vector and the original one by combining their components. Thus, a new trial vector is componentwisely produced as follows:

$$v_{ij}^{(t+1)} = \begin{cases} u_{ij}^{(t+1)}, & \text{if } \texttt{rand}() \leqslant CR \text{ or } j = R_i, \\ x_{ij}^{(t)}, & \text{otherwise}, \end{cases} \tag{4}$$

for all $j = 1, 2, \ldots, n$, where CR is a user-defined parameter of the algorithm, $\texttt{rand}()$ is a uniform random number generator in the interval $[0, 1]$, and $R_i \sim \mathcal{U}(I)$ is a random index independently sampled from the set I for each i and t, which ensures that $v_i^{(t+1)}$ will not be identical to $x_{ij}^{(t)}$. This type of crossover is also called the binomial crossover. An alternative operator is the exponential crossover [33].

After the generation of all trial vectors $v_i^{(t+1)}$, $i \in I$, the iteration is completed with the selection phase, which updates the population as follows:

$$x_i^{(t+1)} = \begin{cases} v_i^{(t+1)}, & \text{if } f\left(v_i^{(t+1)}\right) \leqslant f\left(x_i^{(t)}\right), \\ x_i^{(t)}, & \text{otherwise}, \end{cases} \tag{5}$$

for all $i \in I$. The algorithm iterates until a termination condition is satisfied. Termination is usually related to a predefined solution quality or a prescribed maximum number of iterations, $t_{\max}$.

There is an abundance of variants of differential evolution in the relevant literature [12], [31]. Indicatively, we mention some relevant approaches that will be later considered for comparisons with the proposed method: in [42] a variant called jDElscop was introduced by implementing a population-reduction mechanism, combining different strategies. A self-adaptive and a memetic variant were proposed in [5] and [26], respectively. A different approach called JADE, which uses an external archive for the parameter adaptation, was proposed in [44]. In the same vein, a modified version of JADE, called SHADE [30], is one of the most popular self-adaptive variants, where the algorithm guides parameter adaptation by taking advantage of historical data. Finally, a different approach is adopted in EPSDE [22] in which, parameter adaptation is based on a survival procedure applied on pools of different mutation strategies and parameter settings.

B. Gradient-based Parameter Adaptation with Line Search

The method of gradient-based parameter adaptation with line search (denoted as GPALS) was proposed in [37], extending its grid-based predecessor [33]. The method applies online

Authorized licensed use limited to: University of Ioannina. Downloaded on April 27,2021 at 09:25:34 UTC from IEEE Xplore. Restrictions apply.

adaptation of the parameters of the considered algorithm through approximate gradient search in the parameter space. For this purpose, performance estimations based on short runs of the algorithm are used. Also, the step size on the approximate gradient direction is determined using line search.

Assume that we wish to control the real-valued, population-wide parameters, $\rho_1, \rho_2, \ldots, \rho_\zeta$, of a given algorithm during its run when solving a problem of the type of Eq. (1). For demonstration purposes, we will henceforth assume that we selected a population-based algorithm similar to differential evolution. Let N be the population size of the algorithm, and let the prescribed range for each parameter be:

$$\rho_i \in [l_i, u_i], \quad i = 1, 2, \ldots, \zeta.$$

Then, the parameter domain is defined as the hyperbox:

$$G = [l_1, u_1] \times [l_2, u_2] \times \cdots \times [l_\zeta, u_\zeta].$$

The population of the algorithm is randomly initialized in the search space X. Also, initial values are assigned to its parameters ρ_i, $i = 1, 2, \ldots, \zeta$, either randomly selected in G or according to previous experience (if any). The center of the parameter domain G may be used for non-informative initialization. The initial population is called the primary population and denoted as $P_{\rm pri}$. Also, the initial parameter vector is denoted as $\hat{\rho}_0 = (\rho_1^0, \rho_2^0, \ldots, \rho_\zeta^0) \in G$.

After its initialization, the primary population runs for a specific number of iterations, $t_{\rm pri}$, using the given parameter set. This phase is also called the *dynamic deployment phase*. The user-defined parameter $t_{\rm pri}$ depends on several factors such as the tuned algorithm, the available computational resources (function evaluations), the difficulty of the problem at hand, etc. In previous works [33], [37], $t_{\rm pri}$ is related to the problem's dimension n, with:

$$t_{\rm pri} = 10\, n,$$

being suggested as promising empirical value that offered nice results for the differential evolution algorithm on established test suites. Naturally, better values may exist for different experimental settings. Nevertheless, it is preferable to avoid very low or high values of $t_{\rm pri}$.

When the dynamic deployment phase finishes, an average quality measure of the primary population is calculated. In [37] the selected measure was the average objective value, defined as:

$$H(P_{\rm pri}, \hat{\rho}_c) = \frac{1}{N} \sum_{x \in P_{\rm pri}} f(x),$$

where $\hat{\rho}_c$ denotes the current parameter vector of the primary population (initially this is $\hat{\rho}_0$). The obtained average objective value is the outcome of $t_{\rm pri}$ iterations of the primary population $P_{\rm pri}$, using the parameter values $\hat{\rho}_c$. Thus, $\hat{\rho}_c$ is reasonably considered as an influential variable of the calculated performance measure.

The calculation of the average quality measure is the first step of the *performance gradient estimation phase*, where the goal is to estimate the negative gradient of the considered performance measure in $\hat{\rho}_c$. The estimation is based on the symmetric difference formula as follows:

$$-\nabla H(P_{\rm pri}, \hat{\rho}_c) = -\begin{pmatrix} \frac{\partial H(P_{\rm pri}, \hat{\rho}_c)}{\partial \rho_1^c} \\ \vdots \\ \frac{\partial H(P_{\rm pri}, \hat{\rho}_c)}{\partial \rho_\zeta^c} \end{pmatrix}, \qquad (6)$$

where the partial derivatives are defined as:

$$\frac{\partial H(P_{\rm pri}, \hat{\rho}_c)}{\partial \rho_i^c} = \frac{H(P_{\rm pri}, \hat{\rho}_c + \lambda e_i) - H(P_{\rm pri}, \hat{\rho}_c - \lambda e_i)}{2\,\lambda}, \qquad (7)$$

for all $i = 1, 2, \ldots, \zeta$, where e_i is the i-th row of the $\zeta \times \zeta$ identity matrix. The central difference was preferred against forward differences for accuracy reasons.

Although the gradient notation resembles analytical derivatives, we shall keep in mind that the performance measure:

$$H(P_{\rm pri}, \hat{\rho}_c \pm \lambda e_i)$$

is the average objective value of the primary population $P_{\rm pri}$ evolved with the specific parameters $\hat{\rho}_c$ and, hence, it is the outcome of a simulation procedure. Thus, 2ζ such simulations shall be conducted to derive the approximate gradient vector, either serially or in parallel. More specifically, following the algorithm in [37], the primary population $P_{\rm pri}$ is copied into 2ζ secondary populations, denoted as $P_{{\rm sec}_j}$, $j = 1, 2, \ldots, 2\zeta$. Each secondary population $P_{{\rm sec}_j}$ is also assigned its corresponding parameter values:

$$\hat{\rho}_{c_j} = \begin{cases} \hat{\rho}_c + \lambda e_j, & \text{if } j = 1, 2, \ldots, \zeta, \\ \hat{\rho}_c - \lambda e_{j-\zeta}, & \text{if } j = \zeta + 1, \zeta + 2, \ldots, 2\zeta, \end{cases} \qquad (8)$$

where λ is the step size, and $\hat{\rho}_c$ is the parameter vector of the existing primary population. In other words, each secondary population is assigned one of the perturbed parameter vectors required for the derivation of the partial derivatives.

The next step is the evolution of the secondary populations for a limited number of iterations, $t_{\rm sec}$. This is a simulation step that can be conducted either serially or in parallel. In [37], values of $t_{\rm sec}$ ranginf from 5 to 10 were found to be successful in established test suites, while the step size value was set to $\lambda = 0.1$. Both these parameters are user-defined and depend on the studied algorithm (differential evolution in the specific case). It is also noted that all secondary populations are evolved using the same sequence of random numbers [37]. Eventually, the produced approximate gradient vector is normalized in order to become unit vector, thereby alleviating scaling issues [37].

The obtained (normalized) negative gradient vector is subsequently used to update the primary parameter vector $\hat{\rho}_c$. The new primary parameter vector is obtained as follows:

$$\hat{\rho}_{c+1} = \hat{\rho}_c - s\, H(P_{\rm pri}, \hat{\rho}_c), \qquad (9)$$

where $s > 0$ is a proper step size determined through line search. In [37], the golden section method was selected

Authorized licensed use limited to: University of Ioannina. Downloaded on April 27,2021 at 09:25:34 UTC from IEEE Xplore. Restrictions apply.

due to its rapid convergence properties. Initially a bracketing procedure is applied, where four cut points of equal distance are defined, $s_1 < s_2 < s_3 < s_4$. The first one is $s_1 = 0$, which corresponds to the current parameter vector $\hat{\rho}_c$ in Eq. (9). The last one, s_4, corresponds to the intersection of the negative gradient vector with the boundary of the search space. This point can be easily estimated using a bisection search along the negative gradient vector. Then, the rest is defined as:

$$s_2 = s_4 - \psi\,\Delta, \qquad s_3 = s_1 + \psi\,\Delta,$$

where $\Delta = s_4 - s_1$, and $\psi = (\sqrt{5} - 1)/2$. These four points are adequate to apply the golden section method. Each point s_i corresponds to a different parameter setting (let us denote it as $\hat{\rho}_{s_i}$) derived from Eq. (9). The evaluation of s_i consists of the same simulation steps as previously for the gradient estimation. More specifically, a secondary population $P_{\mathrm{sec}_{s_i}}$ is initialized as equal to the primary one, and it is assigned the parameters $\hat{\rho}_{s_i}$. This population is evolved for t_{sec} iterations, and its performance measure, $H\left(P_{\mathrm{sec}_{s_i}}, \hat{\rho}_{s_i}\right)$, is used as the objective value for s_i. Using these values, the golden section method proceeds accordingly. The line search iterates until the subsequent inspected points become adequately close to each other. Naturally, the user may adopt different line search approaches.

Eventually, line search provides a desirable step size s^* on the direction of the negative gradient, and a new parameter vector $\hat{\rho}_{c+1}$ is obtained from Eq. (9). Also, the corresponding secondary population $P_{\mathrm{sec}_{s^*}}$ becomes the new primary population P_{pri}. This step completes a cycle of the method, which iterates again from the dynamic deployment phase, following the same steps as above.

The application of GPALS has proved to be very competitive against top-performing algorithms. The reader is referred to [37] for a thorough analysis.

III. Proposed Method

The presented GPALS method has three operational parameters, namely t_{pri}, t_{sec}, and λ. Their setting in [37] followed empirical rules based on previous evidence for the grid-based predecessor of the method [36]. According to that evidence, higher values of t_{pri}, λ, and lower values of t_{sec} were associated with superior performance. Although the method exhibits mild sensitivity to its parameters, the issue of *"tuning the tuner"* arises as in most parameter tuning methods. Since the method is based on data generated during the run of the algorithm, it is reasonable to wonder whether this data can be used to better understand the dynamics induced by different parameter values and adapt them accordingly.

In this framework, we propose an enhanced variant of the GPALS method, which aims to ameliorate its parameter-setting requirements. The proposed approach incorporates reinforcement learning, and it will be henceforth denoted as RL-GPALS. The use of reinforcement learning aims at exploiting the data produced during the run of the algorithm, in order to identify promising parameter values and modify the parameters of GPALS on-the-fly.

REINFORCE is a well-known family of reinforcement learning algorithms [3], [28]. Given a number of binary decision variables, it assigns a real-valued weight to each value (0 or 1) of each variable. The weights are used to proportionally determine the selection probabilities of 0 and 1 for each binary variable. Then, stochastic selection takes place to produce a binary vector, where each binary variable has been assigned a value. The system is evaluated and its quality is used to update the weights of the binary values, accordingly. This way the system is capable to learn probabilities that produce the best-performing binary vector [29]. The method has been proved to follow the stochastic hill-climbing property [40]. A variant for multivalued variables has been successfully studied in [19]. This approach is of our interest in the present paper.

Reinforcement learning is better understood under discretized decision spaces. Thus, in the present preliminary study, we consider only discretized values (referred to as levels) of the three parameters of the GPALS method. Putting it formally, let:

$$\xi_{\mathrm{pri}_i}, \quad \xi_{\mathrm{sec}_i}, \quad \xi_{\lambda_i}, \quad i = 1, 2, \ldots, q,$$

denote the i-th level value of t_{pri}, t_{sec}, and λ, respectively. Without loss of generality, we assume that each parameter has q candidate levels. Also, let:

$$w_i^{\varphi}, \quad i = 1, 2, \ldots, q, \quad \varphi \in \Phi \triangleq \{\text{“sec”}, \text{“pri”}, \text{“}\lambda\text{”}\},$$

be the corresponding positive, real-valued weight for each parameter level. Then, the selection probability of each parameter level is given as follows:

$$o_i^{\varphi} = \frac{\exp\left(w_i^{\varphi}\right)}{\sum_{j=1}^{q} \exp\left(w_j^{\varphi}\right)},$$

for all $i = 1, 2, \ldots, q$, and $\varphi \in \Phi$. Based on these selection probabilities, the values ξ_{pri}^*, ξ_{sec}^*, and ξ_{λ}^*, of t_{pri}, t_{sec}, and λ, respectively, are stochastically selected among the available ones through fitness proportionate selection (the well known roulette-wheel selection). Note that this selection type allows for some weaker parameter levels to survive the selection process, thereby promoting better exploration.

Having set its parameters, a full cycle of GPALS is applied as previously described. At this stage, a measure of goodness of the assigned values of GPALS is needed in order to serve as the reinforcement signal r that will be used for the adaptation of the learned weights. The measure shall reflect the quality of the GPALS outcome under the specific parameters, it shall be easy to compute, and avoid scaling issues during the different phases of the optimization procedure.

For the above reasons, in our demonstrative experiments with differential evolution, we recorded the number of trial points $v_i^{(t+1)}$ produced by Eq. (4) that were eventually admitted in the offspring population by replacing their original individuals $x_i^{(t)}$, for each iteration of differential evolution in the dynamic deployment phase. We call these points the *successful points* of the population, and their average number

Authorized licensed use limited to: University of Ioannina. Downloaded on April 27,2021 at 09:25:34 UTC from IEEE Xplore. Restrictions apply.

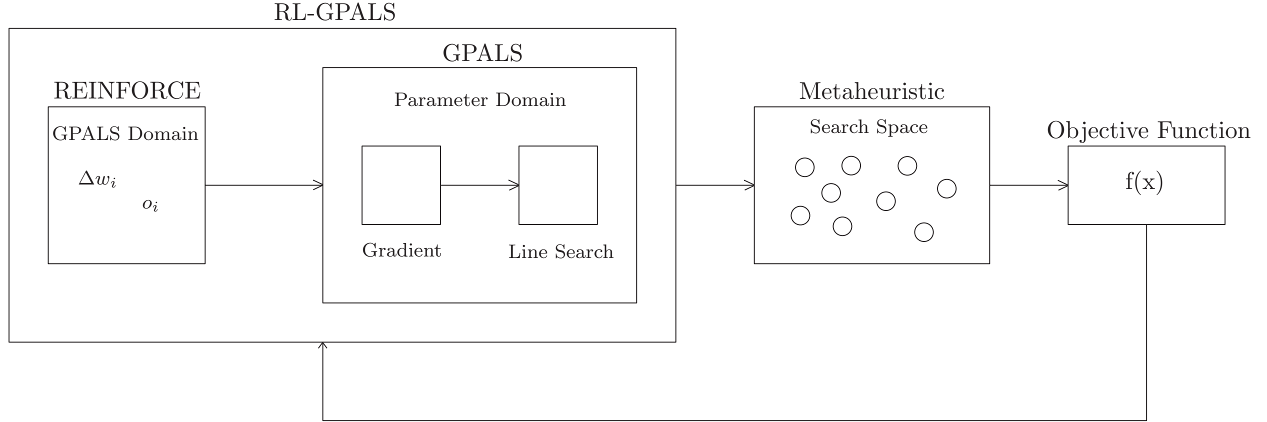

Fig. 1. Workflow of the proposed method.

TABLE I
PARAMETER LEVELS FOR THE REINFORCEMENT LEARNING PHASE OF RL-GPALS.

Parameter	Level 1	2	3	4
t_{sec}	5	10	15	20
t_{pri}	$5n$	$10n$	$15n$	$20n$
λ	0.1	0.2	0.3	0.4

over these iterations is subsequently used to provide the reinforcement signal. Naturally, this type of reinforcement signal was guided by the specific metaheuristic (differential evolution). Moreover, due to the hill climbing property of the REINFORCE algorithm [40], the reinforcement signal in our case needs to be reversed, because the average update in parameter space lies in a direction for which the expected value of r increases. Alternative measures of goodness can be used in case of algorithms of different type.

After the end of GPALS, the learned weights are updated as follows:

$$\Delta w_i^{\varphi} = \begin{cases} \alpha\,(r-\bar{r})\,o_i^{\varphi}\,(1-o_i^{\varphi}) - \delta w_i^{\varphi}, & \text{if } i = i^*, \\ -\alpha\,(r-\bar{r})\,o_i^{\varphi} o_k^{\varphi} - \delta w_i^{\varphi}, & \text{if } i \neq i^*, \end{cases} \tag{10}$$

for each $\varphi \in \Phi$. The index i^* denotes the selected level value of the corresponding parameter, and $0 < \delta < 1$. The positive parameter α is the learning rate, r is the reinforcement signal delivered by the GPALS method, and $\bar{r}$ is the reinforcement comparison:

$$\bar{r}(t) = \gamma\,\bar{r}(t-1) + (1-\gamma)\,r(t),$$

where $\gamma \in (0,1]$ is the decay rate. The specific update scheme promotes the selected parameter levels that responded with positive reinforcement signals, while weakening the rest of the values. Thus, it can dynamically drive the parameter setting of GPALS based on its online outcome.

IV. APPLICATION DETAILS ON DIFFERENTIAL EVOLUTION

The differential evolution algorithm is used for the demonstration of the proposed RL-GPALS method. The selection of the specific algorithm is motivated by its previous use with GPALS in [37]. We will henceforth denote the studied approach as RL-GPALS-DE. The prescribed ranges for the two scalar parameters of the algorithm are $F, CR \in (0,1]$. Thus, the parameter domain G is defined as:

$$G = (0,1] \times (0,1].$$

The RL-GPALS-DE method initializes the primary population P_{pri} in the problem's search space X, and assigns the center of the parameter domain space, $(CR, F) = (0.5, 0.5)$, as the current parameter vector $\hat{\rho}_c$ of the primary population. At the same time, reinforcement learning initializes all weights to $w_i^{\varphi} = 0$ for all i and φ, hence the selection probabilities become $o_i^{\varphi} = 0.25$. Drawing from the sensitivity analysis of the GPALS parameters in [36], the four possible value levels reported in Table I are assumed for each parameter. The parameters are randomly initialized on one of these values.

After initialization, the primary population is evolved for t_{pri} iterations. Then, the performance gradient estimation phase is initiated by evoking 4 secondary populations, in order to calculate the negative gradient according to Eq. (6). Then, line search is applied and a new primary population with its corresponding parameter vector is obtained. Simultaneously, the reinforcement signal r is calculated. Finally, the system calculates the selection probabilities and updates the GPALS parameters t_{pri}, t_{sec}, and λ. The method continues its iterations with the same sequence of steps as described in the previous sections until a termination condition is met. In our case, this criterion is the available computation budget in terms of function evaluations.

In our experiments (thoroughly presented in the following section) the default parameters of the reinforcement learning system:

$$a = 0.1, \quad \gamma = 0.9, \quad \delta = 0.02.$$

were used, as proposed in [19]. It shall be noted that, due to the properties of any REINFORCE algorithm, its parameters typically control the convergence speed. Albeit the absence of convergence theory, these algorithms tend to converge to local

Authorized licensed use limited to: University of Ioannina. Downloaded on April 27,2021 at 09:25:34 UTC from IEEE Xplore. Restrictions apply.

optima [40], [41]. Furthermore, as mentioned in [41], only the parameter a seems to have performance impact. Higher values of a tend to local optima convergence, while lower values promote the exploration of the system state space. The rest of the parameters were set on the above values that are commonly used in similar cases. A graphical representation of the proposed approach according to the above description is given in Fig. 1.

V. Experimental Results

The performance of the proposed RL-GPALS-DE method was assessed on a variety of test problems of various dimensions. Problems of higher dimension are usually more challenging. Thus, our reasonable expectation is that parameter-adaptation methods such as RL-GPALS can be more beneficial in such cases. For this reason, the test suite of the special issue on large scale continuous optimization problems in [21] was used. The test suite consists of 19 scalable, shifted, and hybrid composition test problems. The considered dimensions were:

$$n \in \{50, 100, 200\},$$

for all the problems, and the maximum computation budget in terms of function evaluations was:

$$\tau_{\max} = 5000\, n,$$

following the test suite rules for comparisons. Note that this budget is provided solely for running the algorithms, and does not take into consideration the computational effort spent for tuning the competitor algorithms.

The test suite setting recommends three base algorithms for comparisons, namely differential evolution with exponential crossover (denoted as $\mathrm{DE_{exp}}$), CHC [13], and GC-MAES [1]. Complete results for these algorithms are provided in the test suite to allow full statistical analysis. Moreover, mean error values for 13 additional algorithms are provided for comparisons, namely SOUPDE [39], DE-D^{40}+M^{m} [14], GaDE [43], jDElscop [6], SaDE-MMTS [45], MOS [18], MA-SSW-Chains [23], RPSO-vm [15], Tuned IPSOLS [24], EvoPROpt [8], EM323 [16], VXQR1 [25], and GODE [38].

The adopted performance measure was the solution value error defined as:

$$e_{\mathrm{alg}} = f\left(x_{\mathrm{alg}}\right) - f\left(x^{*}\right),$$

where x^* is the known optimal solution for each problem, and x_{alg} is the solution found by the algorithm. For each problem, 25 independent experiments were conducted, and the mean error, as well as the standard deviation were recorded for each one. In all cases, the population size was set to:

$$N = 60,$$

according to previous works [33], [37].

Comparisons among the algorithms were conducted in two phases. In the first phase, the proposed RL-GPALS-DE method was statistically compared against the base algorithms as well as against the standard GPALS method [37] on each test problem. Wilcoxon rank-sum tests at confidence level 95% were used for this purpose. In the second phase, the proposed method was compared against the 13 competitor algorithms in terms of their average error.

In the first phase, differential evolution algorithm with binomial crossover (denoted as $\mathrm{DE_{bin}}$) was included in the results, in addition to its counterpart with exponential crossover, due to its popularity. The quality of RL-GPALS-DE was assessed based on pairwise comparisons with the base algorithms on the 19 test problems using the Wilcoxon rank-sum test. For each test problem, a win (denoted as "$+$") was counted whenever RL-GPALS-DE had statistically different performance than the competitor algorithm and attained better average error value. Otherwise, a loss (denoted as "$-$") was recorded. Draws (denoted as "$=$") were counted when the compared algorithms had no statistical difference between them.

The results of the statistical tests are reported in Table II (analytical results are omitted due to space limitation). As we can observe, the proposed method exhibits superior average performance against the base algorithms for all dimensions. In most cases, the number of wins of RL-GPALS-DE is higher than double the number of losses and draws together. Moreover, the method significantly outperforms the $\mathrm{DE_{exp}}$ algorithm, which was reported to be the best-performing base algorithm of the test suite [21]. It shall be noted that the superiority of the proposed method was achieved against algorithms that were already tuned for the specific problems, without considering the additional computation cost.

Despite the superiority of the proposed method, similar performance was observed for GPALS in [37], thereby raising questions regarding the actual contribution of reinforcement learning. For this reason, we conducted further statistical comparisons of RL-GPALS-DE with the standard GPALS method. The results are reported at the bottom of Table II, and they clearly reveal that reinforcement learning is highly beneficial, especially in higher dimensions. It is worth noting that there was no test problem where the proposed method was outperformed by its predecessor, while the number of wins was always higher than that of draws.

In the second phase of experimentation, the provided average error values of the 13 additional competitor algorithms included in the test suite [20] were used for comparisons against the proposed method. Table III summarizes the results, reporting the number of test problems where the proposed method exhibited inferior (losses) or non-inferior performance (wins or draws). Evidently, the proposed method achieved competitive performance against all competitor algorithms, including rather sophisticated schemes such as jDElscop and GaDE. Also, it proved to be very competitive against different algorithms such as GODE, SaDE-MMTS, SOUPDE, and DE-D^{40}+M^{m}, especially as dimension increases. Moreover, it is worth noting that the proposed method outperforms also algorithms that are not based on differential evolution, such as EvoPROpt, MA-SSW-Chains, RPRSO-vm, Tuned IPSOLS, and VXQR1, in all dimensions.

Authorized licensed use limited to: University of Ioannina. Downloaded on April 27,2021 at 09:25:34 UTC from IEEE Xplore. Restrictions apply.

TABLE II

NUMBER OF WINS (+), LOSSES (−), AND DRAWS (=) OF RL-GPALS-DE AGAINST THE BASE ALGORITHMS AS WELL AS THE GPALS METHOD.

	Problem dimension								
	50			100			200		
Algorithm	+	−	=	+	−	=	+	−	=
DE_{bin}	13	3	3	17	1	1	17	1	1
DE_{exp}	11	6	2	13	4	2	13	5	1
CHC	19	0	0	19	0	0	19	0	0
GCMAES	16	3	0	16	3	0	16	3	0
GPALS	6	0	13	12	0	7	10	0	9

TABLE III

NUMBER OF PROBLEMS WHERE RL-GPALS-DE EXHIBITED INFERIOR (−) AND NON-INFERIOR (+ / =) AVERAGE SOLUTION VALUES AGAINST A VARIETY OF ALGORITHMS.

	+ / =			−		
	Dimension			Dimension		
	50	100	200	50	100	200
SOUPDE	11	13	13	8	6	6
DE-D^{40}+M^m	13	13	13	6	6	6
GODE	13	14	14	6	5	5
GaDE	11	13	11	8	6	8
jDElscop	10	12	10	9	7	9
SaDE-MMTS	14	14	14	5	5	5
MOS	11	12	10	8	7	9
MA-SSW-Chains	14	16	17	5	3	2
RPSO-vm	18	18	18	1	1	1
Tuned IPSOLS	15	14	14	4	5	5
EvoPROpt	18	18	18	1	1	1
EM323	15	15	16	4	4	3
VXQR1	15	16	16	4	3	3

VI. CONCLUSIONS

Parameter tuning and control is a crucial aspect of metaheuristic design. It requires time-consuming and challenging procedures, sometimes offering questionable results over different types of optimization problems. Moreover, in-depth knowledge of the considered algorithm is usually needed by the user in order to achieve competitive results. To this end, parameter control methods have been proposed for the online adaptation of the algorithm's parameters. However, even the control methods contain parameters or involve decisions from the user side.

The present paper is a first attempt to enhance a recently proposed parameter control method by using a simple reinforcement learning approach. The main goal is to ameliorate the decision burden of the user regarding its parameters. Reinforcement learning is used to learn and automatically adapt the internal parameters of the tuner. The performance of the enhanced method was proved to be significantly improved using the REINFORCE reinforcement learning system, without noticeable increase in running time of the overall approach. This was verified on an established high-dimensional test suite, following the setting of previous studies. The obtained results revealed that the proposed method can serve as a decision-support tool for the user toward the parameter setting and adaptation of metaheuristics.

The use of a reinforcement learning in the field of parameter control has opened various directions for further inquiry. A broader application of the proposed method on a variety of test problems and algorithms is under development. Furthermore, the application of the method on real-world problems is also under investigation.

ACKNOWLEDGEMENT

This research is co-financed by Greece and the European Union (European Social Fund - ESF) through the Operational Programme "Human Resources Development, Education and Lifelong Learning" in the context of the project "Reinforcement of Postdoctoral Researchers - 2nd Cycle" (MIS-5033021), implemented by the State Scholarships Foundation (IKY).

REFERENCES

[1] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC)*, pages 769–1776, 2005.

[2] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation*. Springer-Verlag, Berlin, 2006.

[3] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE Conference on Decision and Control*, volume 1, pages 560–564. IEEE, 1995.

[4] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-Race and iterated F-Race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, Berlin, Heidelberg, 2010.

Authorized licensed use limited to: University of Ioannina. Downloaded on April 27,2021 at 09:25:34 UTC from IEEE Xplore. Restrictions apply.

[5] J. Brest, B. Bošković, and A. Zamuda. Self-adaptive differential evolution algorithm with a small and varying population size. In *WCCI 2012 IEEE World Congress on Computational Intelligence*, 2012.

[6] J. Brest and M. S. Maucec. Self-adaptive differential evolution algorithm using population size reduction and three strategies. *Soft Computing*, 15:2157–2174, 2011.

[7] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, 2011.

[8] A. Duarte, R. Martí, and F. Gortazar. Path relinking for large scale global optimization. *Soft Computing*, 15:2257–2273, 2011.

[9] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[10] A. E. Eiben and S. K. Smit. Evolutionary algorithm parameters and methods to tune them. In Y. Hamadi, E. Monfroy, and F. Saubion, editors, *Autonomous Search*, chapter 2, pages 15–36. Springer, Berlin Heidelberg, 2011.

[11] T. Eltaeib and A. Mahmood. Differential evolution: A survey and analysis. *Applied Sciences*, 8(10):1945, 2018.

[12] M. G. Epitropakis, D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis. Enhancing differential evolution utilizing proximity-based mutation operators. *IEEE Transactions on Evolutionary Computation*, 15(1):99–119, 2011.

[13] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. *Foundations of Genetic Algorithms*, 2:187–202, 1993.

[14] C. García-Martínez, F. J. Rodríguez, and M. Lozano. Role differentiation and malleable mating for differential evolution: An analysis on large scale optimisation. *Soft Computing*, 15:2109–2126, 2011.

[15] J. García-Nieto and E. Alba. Restart particle swarm optimization with velocity modulation: A scalability test. *Soft Computing*, 15:2221–2232, 2011.

[16] V. Gardeux, R. Chelouah, P. Siarry, and F. Glover. EM323: A line search based algorithm for solving high-dimensional continuous non-linear optimization problems. *Soft Computing*, 15:2275–2285, 2011.

[17] H. H. Hoos. Automated algorithm configuration and parameter tuning. In Y. Hamadi, E. Monfroy, and F. Saubion, editors, *Autonomous Search*, chapter 3, pages 37–72. Springer, Berlin Heidelberg, 2011.

[18] A. LaTorre, S. Muelas, and J. Peña. A MOS-based dynamic memetic differential evolution algorithm for continous optimization a scalability test. *Soft Computing*, 15:2187–2199, 2011.

[19] A Likas. Multivalued parallel recombinative reinforcement learning: A multivalued genetic algorithm. *Proceedings of Fourth Hellenic-European Conference on Computer Mathematics and its Applications, (HERCMA)*, 98, 98.

[20] M. Lozano, F. Herrera, and D. Molina. Evolutionary algorithms and other metaheuristics for continuous optimization problems. http://sci2s.ugr.es/eamhco/, 2010.

[21] M. Lozano, F. Herrera, and D. Molina. Scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. *Soft Computing*, 15:2085–2087, 2011.

[22] R. Mallipeddi, P. N. Suganthan, Q. Pan, and M. F. Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied soft computing*, 11(2):1679–1696, 2011.

[23] D. Molina, M. Lozano, A. M. Sánchez, and F. Herrera. Memetic algorithms based on local search chains for large scale continuous optimisation problems: MA-SSW-Chains. *Soft Computing*, 15:2201–2220, 2011.

[24] M. A. Montes de Oca, D. Aydin, and T. Stützle. An incremental particle swarm for large-scale optimization problems: An example of tuning-in-the-loop (re)design of optimization algorithms. *Soft Computing*, 15:2233–2255, 2011.

[25] A. Neumaier, H Fendl, H. Schilly, and T. Leitner. VXQR: Derivative-free unconstrained optimization based on QR factorizations. *Soft Computing*, 15:2287–2298, 2011.

[26] A. P. Piotrowski. Adaptive memetic differential evolution with global and local neighborhood-based mutation operators. *Information Sciences*, 241:164–194, 2013.

[27] R. Storn and K. Price. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *J. Global Optimization*, 11:341–359, 1997.

[28] R. S. Sutton, A. G. Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[29] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[30] R. Tanabe and A. Fukunaga. Success-history based parameter adaptation for differential evolution. In *2013 IEEE Congress on Evolutionary Computation*, pages 71–78, 2013.

[31] D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis. Parallel differential evolution. In *Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753)*, volume 2, pages 2023–2029. IEEE, 2004.

[32] V. A. Tatsis and K. E. Parsopoulos. Grid search for operator and parameter control in differential evolution. In *9th Hellenic Conference on Artificial Intelligence*, SETN '16, pages 1–9. ACM, 2016.

[33] V. A. Tatsis and K. E. Parsopoulos. Differential evolution with grid-based parameter adaptation. *Soft Computing*, 21(8):2105–2127, 2017.

[34] V. A. Tatsis and K. E. Parsopoulos. Grid-based parameter adaptation in particle swarm optimization. In *12th Metaheuristics International Conference (MIC 2017)*, pages 217–226, 2017.

[35] V. A Tatsis and K. E Parsopoulos. Experimental assessment of differential evolution with grid-based parameter adaptation. *International Journal on Artificial Intelligence Tools*, 27(04):1–20, 2018.

[36] V. A. Tatsis and K. E. Parsopoulos. On the sensitivity of the grid-based parameter adaptation method. In *7th International Conference on Metaheuristics and Nature Inspired Computing (META 2018)*, pages 86–94, 2018.

[37] V. A. Tatsis and K. E. Parsopoulos. Dynamic parameter adaptation in metaheuristics using gradient approximation and line search. *Applied Soft Computing*, 74:368–384, 2019.

[38] H. Wang, Z. Wu, and S. Rahnamayan. Role differentiation and malleable mating for differential evolution: An analysis on large scale optimisation. *Soft Computing*, 15:2127–2140, 2011.

[39] M. Weber, F. Neri, and V. Tirronen. Shuffle or update parallel differential evolution for large scale optimization. *Soft Computing*, 15:2089–2107, 2011.

[40] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[41] R. J. Williams and J. Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

[42] Z. Yang, K. Tang, and X. Yao. Scalability of generalized adaptive differential evolution for large-scale continuous optimization. *Soft Computing*, 15(11):2141–2155, 2011.

[43] Z. Yang, K. Tang, and X. Yao. Scalability of generalized adaptive differential evolution for large-scale continuous optimization. *Soft Computing*, 15:2141–2155, 2011.

[44] J. Zhang and A. C. Sanderson. JADE: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13:945–958, 2009.

[45] S. Z. Zhao, P. N. Suganthan, and S. Das. Self-adaptive differential evolution with multi-trajectory search for large-scale optimization. *Soft Computing*, 15(11):2175–2185, 2011.

Authorized licensed use limited to: University of Ioannina. Downloaded on April 27,2021 at 09:25:34 UTC from IEEE Xplore. Restrictions apply.