Generic Parameter Control with Reinforcement Learning

Giorgos Karafotias
VU University
Amsterdam, Netherlands
g.karafotias@vu.nl

A.E. Eiben
VU University
Amsterdam, Netherlands
a.e.eiben@vu.nl

Mark Hoogendoorn
VU University
Amsterdam, Netherlands
m.hoogendoorn@vu.nl

ABSTRACT

Parameter control in Evolutionary Computing stands for an approach to parameter setting that changes the parameters of an Evolutionary Algorithm (EA) on-the-fly during the run. In this paper we address the issue of a generic and parameter-independent controller that can be readily plugged into an existing EA and offer performance improvements by varying the EA parameters during the problem solution process. Our approach is based on a careful study of Reinforcement Learning (RL) theory and the use of existing RL techniques. We present experiments using various state-of-the-art EAs solving different difficult problems. Results show that our RL control method has very good potential in improving the quality of the solution found without requiring additional resources or time and with minimal effort from the designer of the application.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods, Control theory

Keywords

Evolutionary algorithms; parameter control; reinforcement learning

1. INTRODUCTION

Controlling the parameters of Evolutionary Algorithms (EA) on-the-fly has been on the research agenda of the Evolutionary Computing community since the late nineties, when Eiben et al. put the issue in the spotlight [4]. Over the last decade and a half, the field has made great progress and several good parameter control mechanisms have been invented. This progress is reviewed and related trends are identified in a recent survey [12]. However, as noted in this survey, the field suffers from the lack of generic control methods that can be applied to (m)any parameter(s). This causes *the patchwork problem*: "if one is to control more (all) parameters of an EA, then for each parameter she has to choose from a parameter specific set of existing methods and mix them into one system. Unfortunately, little is known about the joint effects of control mechanisms, thus there are no good guidelines about how to create good combinations. Therefore, the resulting mix is necessarily ad hoc and likely suboptimal."

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
Copyright 2014 ACM 978-1-4503-2662-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2576768.2598360 .

In this paper we investigate a possible approach to mitigate this issue. Our approach is based on using reinforcement learning (RL) [21] [27]. The main idea is to use RL as a generic control mechanism that can be employed as a 'universal plugin' to adjust the values of any parameter of any EA on-the-fly.

Even though it is not common in the related parameter control literature, let us try to position this approach by discussing its possible niche, i.e., the use cases where it can (expectedly) provide an advantage. To this end, it is important to note that the performance of EAs can be greatly improved by tuning their parameters before the run. Over the last decade, this issue has received much attention and several very good parameter tuners are available, see, for instance [5] for an overview. Such tuning techniques have their niche in repetitive problems (cf. [Chap 13] in [6]) that occur with little variations and the possible extensive tuning costs are compensated by the repeatedly occurring performance benefits. On the other hand, parameter control mechanisms offer benefits in cases when tuning is not possible or not economic. For instance, in online evolutionary robotics or other real-time applications, where the problem can change and human intervention to adjust the EA is not feasible. Or, in case of one-off problems that need to be solved by just a few EA runs without time for tuning.

The main research question of this paper is the following:

> Can an RL-based parameter controller improve the performance of an off-the-shelf EA without the need for tailoring the controller towards the given problem?

To answer this research question we develop an RL controller and perform experiments using it as a 'plug-in' for several EAs that are tested on a number of different problems. To gain a good base for conclusions we selected good and freely available EAs with varying levels of sophistication, ranging from a straightforward ES, through some competitive GAs and ending with the highly developed IPOP CMA-ES. As for selecting the test problems, for each algorithm we chose problems from a collection for which the given EA has been developed. This guaranteed that there was no trivial gain for the controller. All together we use 4 EAs and 11 test problems, including 4 real world problems.

2. RELATED WORK

Reinforcement learning has been employed for operator selection by Sakurai et al. [18], Chen et al. [2] and Pettinger and Everson [17]. Muller et al. [16] used temporal difference learning to control the mutation step size for real valued encodings. These methods use feedback from the EA to define a state and choose actions that set parameters, thus, model parameter control as a RL problem. However, all these methods are simplistic in their approach and further limited by the fact that they are specific to a single parameter. A reduced version of reinforcement learning, i.e. the multi-armed bandit approach, has been extensively used for adaptive operator selection (for a list of such publications refer to [12]).

The only case we are aware of where RL was used for generic parameter control is by Eiben et al. [3]. Fitness based metrics were used to define the state while actions were mapped to set any/all parameters. However, they used an unmotivated combination of on-policy and off-policy learning while the experimental results were discouraging.

3. FORMULATION OF THE PROBLEM

In this section we formulate EA parameter control as a reinforcement learning problem. We start by discussing the state of an EA, how parameter control affects it and the definition of a parameter control design. Subsequently, we map this EA parameter control design onto an RL problem.

The complete state of an EA can be defined as:

$$S_{EA} = \{G, \bar{p}\} \tag{1}$$

where G is the set of all the genomes in the population and $\bar{p}$ is the vector of current parameter values. A pair S_{EA} uniquely specifies the state of the search process for a given evolutionary algorithm (the design and specific components of the EA need not be included in the state since they are the same during the whole run) in the sense that S_{EA} fully defines the search results so far and is the only observable factor that influences the search process from this point on (though not fully defining it, given the stochastic nature of EA operators). Time is not part of S_{EA} as it is irrelevant to the state itself; it introduces an artificial uniqueness and a property that is unrelated to the evolution. Of course, state transitions are not deterministic.

The state of the EA, and in extension the outcome of the search process, can be affected by changing G or $\bar{p}$. Evolutionary operators change G; every time variation or selection is applied the genotypes in the population change. Controlling the parameters changes $\bar{p}$, thus also affecting the next state(s) of the EA. The goal of such control is to maximize the final performance (at the end of a number of available evaluations). However, the long term effects of a certain $\bar{p}$ applied cannot be known, only the current S_{EA}.

The definition of an EA parameter control method requires three components [13]: (i) the choice of parameters (including the frequency and time of updates), (ii) the observables that are derived from the search state S_{EA} and are used as input to the controller and (iii) the specific mechanism/algorithm (either static or dynamic) that maps the current observables' values to parameter values.

There is an apparent analogy between the EA parameter control problem and the full RL problem and a Markov Decision Process (MDP). The EA state observables suggest a state space, the parameters controlled and their ranges point to an action space while a dynamic control mechanism component could be implemented by a specific RL algorithm. Although this high level mapping is straightforward, the exact formulation of the RL problem is far from trivial. To define an MDP we need to define a state space, an action space, a transition function and a reward function [21] [23]. These are explained below.

The most important and challenging is the formulation of the state space. Ideally, we would like our state definition to give our process the Markov property, i.e.:

$$P(s_{t+1}, r_{t+1} | s_t, r_t, s_{t-1}, r_{t-1}, ..., s_1, r_1, s_0) = P(s_{t+1}, r_{t+1} | s_t, r_t) \tag{2}$$

Such a state definition would entirely contain all the necessary information for deciding the most appropriate action without taking into account the complete history [23].

Defining the state of the RL problem directly as S_{EA} is problematic. On one hand, using S_{EA} directly is impractical and not useful; it is unlikely such a state will be repeated, thus whatever is learned there is useless, while approximations of a state value function seem rather difficult. On the other hand, if we choose our observations to be other more compact measures derived from S_{EA} we are considering a Partially Observable MDP (PO-MDP) setting because multiple S_{EA} states can give the same observable values [20]. The alternative would be to define the state space using derived observables as its dimensions. The likely risk in this case is that the Markov property is lost and the problem becomes non-stationary. If the observables used are too simple or weak then not only the current state will not contain enough information but also the optimal actions for a state will quickly change. Furthermore, since such observables would probably be continuous, so will the state space. Prior discretization of the state space can be highly inefficient since the distribution of values of the observables cannot be known beforehand. Consequently, special techniques are required to deal with the continuous state space [8].

Regardless the exact definition of the state space, the transition function is of course unknown because of the unknown effect of parameter values and noise due to the influence of the stochastic operators on the state of the EA.

Second, we must define the actions of the RL problem that will give parameter values. There are two general options for approaching this: (i) actions that increase or decrease (or maintain) the current parameter value and (ii) actions that set a specific value to the parameter. These options pose a tradeoff. Defining actions to increase/decrease the current parameter values makes the next values dependent on the current ones. Subsequently, we should expand the state definition to include the current parameter vector or we should cope with another source of uncertainty about the current state making the PO-MDP problem described earlier stronger. On the other hand, actions that define direct values for parameters mean that the action space must be the Cartesian product of the domains of all parameters controlled. Such an action space also poses the problem of continuity since actions correspond to values of continuous parameters. However, unlike the state space, discretisation of the action space may be reasonable. Though we can expect that certain regions of parameter values may deserve more attention and finer granularity, some parameter have wide "good areas" of values (e.g. crossover rate of GAs). Even so, a reasonable amount of discretisation intervals would be required resulting in a higher number of total actions than if we used option (i), thus slowing down learning because more exploration would be required.

Finally, we define reward. According to Sutton and Barto [21] reward must indicate what we want accomplished without being biased by the designer's intuition about the "how". The difference between the previous and the current fitness would be an obvious option but it would result in disproportionate rewards: an EA tends to make large steps in the beginning even if parameters are not good while near the end very small fine-tuning improvements are very hard to achieve. This could be solved if we know the target fitness of the problem but we cannot make this assumption in general. Consequently, we try to alleviate the problem by defining reward as the ratio of the previous fitness to the current (for a minimisation problem). Since the population and offspring sizes may be among the parameters controlled, a reward should be normalised according to the effort (number of evaluations) spent for the improvement made. Thus, we make the following definition for the reward:

$$R(s_t, a_t) = C \cdot \frac{\frac{f_b^{t+1}}{f_b^t} - 1}{Evals_{t+1} - Evals_t} \tag{3}$$

where f_b^i is the best fitness at time i, $Evals_i$ is the number of evaluations spent until time i and C is a scaling constant.

A model of the reward function is not available because of the unknown effect of parameter values and the stochastic nature of the evolutionary operators.

4. A GENERIC RL CONTROLLER

In this section we present a concrete design of a generic parameter controller based on reinforcement learning to tackle the problem as defined in the previous section. First we present a concrete list of observables for defining state, subsequently the manner actions are treated and, finally, the specific algorithm used for learning.

The choice of observables that will define the state of the EA is a challenging issue as was explained in the previous section. So far there has been almost no research exploring the usefulness of different observables - the only exceptions we are aware of are by Veerapen et al. [25] and Whitacre et al. [26] - while almost all controllers found in literature use fitness and/or a form of diversity as feedback from the EA without providing any motivation or justification for this choice [12]. A systematic approach for the definition of observables was suggested by Karafotias et al. [13].

For our initial design of a generic controller we have chosen a set of simple observables that we consider appropriate and intuitively useful (since as we explained a better informed decision is impossible at the moment):

1. Genotypic diversity
2. Phenotypic diversity (when different from genotypic)
3. Fitness standard deviation
4. Fitness improvement
5. Stagnation counter

When looking at the observables, two types can be distinguished: those that measure the current state of the population (the first three items) and the ones measuring the progress made over a number of generations (the last two). As for the first type, the first two regard diversity which is generally accepted as an important descriptor of the state of an EA (see [12]). However, not only is the measurement of diversity dependent on the representation used by the EA but the concept of diversity is itself not well-defined [24]. For this paper, we conducted experiments with numeric optimisation, thus we used Euclidean distance for measuring diversity, though there are alternative options (see for example [19] and [15]). The standard deviation of fitness values in the population is used as a secondary measure of diversity and convergence. The observables measuring the progress include the fitness improvement and the stagnation counter. The fitness improvement is the only real fitness measure we use (defined the same as the reward (3)). The reason we did not include absolute fitness values is that such observables would be less useful to a non-restarting EA: states defined by absolute fitness values are not likely to occur again making anything learned from those states unusable in the future. The number of generations that have passed without any improvement is a metric that might be related to taking drastic exploratory action.

Control actions are defined as setting each controlled parameter to a certain value. To simplify our design we cope with the continuous action space by discretising parameter ranges to equal intervals. The number of discretisation intervals is the same for all parameters, regardless of the parameter's range. When the controller picks an interval to set a parameter's value, the exact value is taken uniformly randomly from within that interval. Values of symbolic parameters are treated each separately. Each action of the controller specifies the intervals for all numeric parameters and the values for all symbolic parameters. Consequently, the total number of control actions is equal to the product of the numbers of intervals/values of all parameters. Depending on the EA controlled, the number of parameters will change. The granularity of the discretisation is important here. The smaller the intervals, the more fine-grained the controller can set parameter values, but the larger the action space requiring more time to explore. The number of discretisation bins decides the number of actions; it is left as a choice for each specific application depending on the EA (i.e. how many parameters it has) and problem at hand (i.e. how long a run is possible).

The core of the controller is based on Temporal Difference (TD) learning with dynamic state space segmentation and eligibility traces. The dynamic state segmentation method used is based on the work by Uther and Veloso [22]. The controller represents states as a binary decision tree. Internal nodes are decision nodes; they segment space with an inequality on one of the observables. Leaf nodes represent actual discrete states. Each such state node S_i includes $Q(S_i, a_j)$ values and eligibility traces $e(S_i, a_j)$ for all actions a_j and a $V(S_i)$ value of the state itself. $Q(S_i, a_j)$ denotes the estimated state-action value, i.e. the expected long-term return of taking action a_j when in state S_i. These Q values are used when the controller selects actions. Eligibility traces are a way to assign credit (or blame) to actions when their influence can extend to several steps after they are taken. At each time t, every state-action pair S_i, a_j has an eligibility trace $e(S_i, a_j) \in [0, 1]$ that shows how much responsibility action a_j from state S_i is taking for the reward $R(t)$ that is presently received. The more recently action a_j was taken from state S_i the higher $e(S_i, a_j)$ is and vice versa.

At each control step (which can occur every one or more generations of the EA) the controller receives a set of observables values $\vec{o}(t)$ (as defined earlier) and a reward $R(t)$ (as defined in (3)). It derives the corresponding current discrete state by starting at the root of the state tree and traversing the decision nodes based on the observables values down to a state node $S(t)$. It then selects the current action $a(t)$ using an ϵ-greedy strategy: with probability ϵ a random action is chosen, otherwise it selects the action with the highest Q value: $a(t) = argmax_j Q(S(t), a_j)$. It then calculates the target value δ for the update of the previous action based on the reward $R(t)$ received. The target δ is derived according to the SARSA on-policy rule [21]:

$$\delta = R(t) + \gamma \cdot Q(S(t), a(t)) - Q(S(t-1), a(t-1)) \tag{4}$$

where γ is the discount rate. Before applying the δ target, the eligibility trace of the previous action is updated to one: $e(S(t-1), a(t-1)) = 1$. Subsequently, all eligible state-action pairs are updated according to target δ:

$$\begin{gathered} Q(S_i, a_j) = Q(S_i, a_j) + \alpha_t \cdot \delta \cdot e(S_i, a_j), \\ \forall S_i, a_j : e(S_i, a_j) > e_{min} \\ \alpha_t = \begin{cases} \alpha_0 & R(t) = 0 \\ \alpha & otherwise \end{cases} \end{gathered} \tag{5}$$

where e_{min} is the minimum eligibility and α is the step-size parameter. The value of α decides how much influence the target δ will have on the current Q values. Because rewards (fitness improvements) tend to be zero a lot of the time, we use a different (and much lower) α_0 to avoid Q values quickly dropping to zeros.

The value of the state is updated to be the maximum action value within that state:

$$V(S_i) = maxQ(S_i, a_j) \tag{6}$$

Finally, the eligibility traces of all state-action pairs are updated:

$$e(S_i, a_j) = e(S_i, a_j) \cdot \gamma \cdot \lambda \tag{7}$$

where γ is the discount rate and λ is the trace decay parameter.

After the next action is selected and all updates are performed the state tree is updated. The state tree starts with only one root node which is a state node representing one universal state; any possible observables $\vec{o}$ will map to that state. Every time a control step is made a transition occurs from an observation and action to a reward and a new observation $T_t = (\vec{o}(t), a(t), R(t+1), \vec{o}(t+1))$. Every state maintains an archive of such transitions and at each control step the transition T_t is added to the archive of state $S(t)$. If the state's archive is larger than a threshold A_m and with probability p_s the state is checked for splitting into two new states, which means that the current node becomes an internal decision node and two new state nodes are added as its children. To convert the state node into a decision node a choice of observable and a splitting value are required to form the corresponding inequality.

The purpose of this process is to divide a state into two parts with significantly different value estimates. First, each transition in the state's archive is assigned a value equal to the reward of the transition plus the current value estimation of the state that currently corresponds to the *end* observation of the transition (the $\vec{o}(t+1)$ for T_t). Subsequently, all observables are checked as candidates for the splitting criterion. For each observable:

- Transitions in the archive are sorted according to the current observable in the transition's *starting* observation (the $\vec{o}(t)$ for T_t).
- Taking the sorted list of transitions we consider the values of the current observable. The mid-point between the observable values of every two subsequent transitions in the sorted list is a candidate for a splitting point[1]. Splitting points are only considered if they split a transition list to fractions that are larger than a A_f.
- Given the split point, the values of the transitions of the two parts form two samples. A Kolmogorov-Smirnov double test is run with these two samples and the resulting D value of the test is saved.
- The above process is repeated for all observables.

After all observables are checked, the smallest D value is taken. If it is smaller than D_{max} then the node is split at the corresponding observable and splitting point. Two new state nodes are created and their Q, V and e values are set to the values of the parent node as an initial estimation. The archive of the parent node is split according to the chosen split point and the parts are given to the corresponding children nodes. The parent node becomes a decision node with an inequality using the observable and split point derived above.

The RL controller has ten parameters due to the TD learning rule, the dynamic state tree and the eligibility traces. These parameters and their default values are shown in Table 1.

5. EXPERIMENTAL SETUP

The experiments presented in this section aim at evaluating the RL controller described earlier. To this end, we selected four different EAs (described later) and four different parameter control mechanisms: none (using the default parameter values of the given EA), PRAM [28], our RL, and a mechanism that changes parameter values randomly. In order to get results as realistic and meaningful as possible, each EA was tested with functions it was designed for or is considered competent solving.

[1]If the transitions are too many then only 100 equally spaced points are checked

Table 1: Controller Parameters

TD		State tree		Traces	
Parameter	Value	Parameter	Value	Parameter	Value
ϵ	0.1	A_m	60	λ	0.8
γ	0.8	A_f	0.2	e_{min}	0.001
α	0.9	D_{max}	0.05		
α_0	0.2	p_s	0.1		

5.1 EAs to be controlled

We decided to use algorithms and problems from the classic numeric optimisation domain which do originate from real world applications but are also widely used and understood in the research community. This also allows us to study a wide range of problems as well as the impact of the level of sophistication of the algorithm upon the performance of the RL controller. For our experiments we selected four different EAs ranging in sophistication from very simple to state-of-the-art. In particular we used:

1. A simple self-coded Evolution Strategy (ES) with real valued representation, Gaussian mutation with one σ, uniform crossover and tournament selection for both parents and survivors. The parameters controlled are the population size μ, the generation gap g (the ratio of offspring to population size), the mutation step size σ and the tournament size for survivor selection k_s (the parent tournament size is fixed to two). Being an unspecialised "naive" algorithm, the Simple ES was tested with three of the standard numeric optimisation test functions frequently used in literature: Rastrigin, Schaffer and Fletcher & Powell.

2. The Cellular GA [1] implemented for the BBOB2013[2] competition by Holtschulte and Moses [9] [3]. The parameters controlled are the choice of crossover (two-point or arithmetic), the crossover probability p_c, the choice of mutation (Gaussian, uniform, decreasing Gaussian or alternating uniform and Gaussian), the mutation rate p_m and the mutation variance m_{var}. The parent and survivor selection operators are fixed (ranking and select best) and not parameterisable. The Cellular GA was implemented for the BBOB contest thus we tested it with BBOB functions 21 through 24 (we chose the harder functions that would justify long runs and resemble the real functions tackled in one-off applications).

3. The GA MPC (GA with Multi-Parent Crossover) by Elsayed et al. [7] that was the winner of the CEC2011 competition on real world applications[4]. The parameters controlled are the population size μ, the maximum size of the parent tournament k_{max}, the randomisation probability p_r, the percentage of the population that is put in the archive f_a and the mean β_m and standard deviation β_{std} of the normal distribution used to draw the β weight used in the multi-parent crossover. The survivor selection is "select best" and involves no parameters. The GA MPC was tested with the test suite it was

[2]http://coco.gforge.inria.fr/doku.php?id=bbob-2013
[3]The source code was acquired directly from the authors
[4]http://www3.ntu.edu.sg/home/epnsugan/index_files/CEC11-RWP/CEC11-RWP.htm. The source code of GA MPC is available at the same competition page.

designed for (CEC2011); we used two problems it performed very well and two for which it performed poorly (12/13 and 7/11.8 respectively).

4. A recent CMA ES variant: the IPOP-10DDr CMA-ES by Liao and Stützle [14] that took part in the BBOB 2013 competition[5]. The parameters controlled are the backward time horizon for distribution cumulation c_c, the step size cumulation c_s, the step size damping parameter d, the mixing between rank-one and rank-mu update μ_{cov} and the recombination type (equal, linearly decreasing or super-linearly decreasing weights). The population and offspring sizes are controlled at restart by the IPOP-10DDr mechanism and cannot be changed during actual runtime. The restart conditions were kept to defaults. We used the BBOB test suite where the CMA ES variants are especially competent; we chose the same functions as the Cellular GA (for the same reason).

5.2 Control mechanisms used

Each algorithm was run in combination with four different control mechanisms. The baseline mechanism was the use of no control, that is, using static parameters set to default values as provided in the source codes or specified in the relevant publications listed above. (For our Simple ES we set parameters to "standard" values).

All algorithms were also run with the RL controller. The parameters of the RL controller were set to the default values shown in Table 1 for all experiments. The only option of the RL controller changed was the number of discretisation bins: since the number of parameters and the length of runs differ among EA and problem settings, the amount of discretisation bins was chosen for each experiment separately to yield a reasonable number of control actions.

Third, we run all algorithms with one of the few existing generic controllers, the Probabilistic Rule-driven Adaptive Model (PRAM) by Wong et al. [28]. PRAM can only handle numeric parameters, thus, any symbolic parameters were kept static to their default values when running with PRAM. For all PRAM experiments the epoch length was set to 150, 20% of which was the training phase. PRAM's step sizes were separately adjusted for each algorithm and parameter since parameter ranges differ drastically.

Finally, we also run all algorithms with their parameters being randomly varied (see [10] for the motivation). In these tests the ranges of parameters were the same as for the RL controller but values were drawn uniformly randomly.

Each algorithm/problem/control mode combination was run 30 times. Table 2 summarises the experimental setup[6] showing all algorithms, their parameters and their ranges and default values, the test problems for each algorithm, the number of action discretisation bins used for the RL controller and the step sizes of the PRAM.

6. RESULTS AND ANALYSIS

The results of all experiments are shown in Table 4. The RL controller significantly improves over the static mode in 8 out of 15 cases while it has a better best run in additional 3 cases. For PRAM these number are 6 and 0 and for random 8 and 2. The RL controller and random modes are both significantly better than static in 8 cases, in 3 of these the RL control is significantly better than random and in additional 3 it has a better best run.

In several cases the performance of the EA is improved when using the RL controller compared to the static mode while there are only two cases where performance is significantly worsened with RL control. Furthermore, as we stated in Section 5, the settings of the RL controller were the same across all experiments (with the trivial exception of the number of action bins). Based on these results we can conclude that an RL-based controller can indeed improve the performance of an off-the-shelf EA with no need of tailoring to the given problem.

Looking at the results of Table 4 we can see that the RL controller has a much better effect for some EAs (notably the Simple ES) while it is not particularly advantageous for others such as the CMA-ES. One way to interpret this is by considering the margin of improvement for each algorithm when solving a specific problem, i.e. how well tailored is the algorithm to the problem at hand. The Simple ES is very crude while the Cellular GA, though more complex, is certainly not a competent numeric optimiser. Neither of them is particularly fit for the problems they are solving. On the other hand, the GA MPC is an efficient optimiser (winner of the CEC 2011 competition) but notice that it ranked low for the first two problems while it ranked first and second for the last two problems. Finally, the CMA-ES is considered the most competent numeric optimiser and consistently performs well for the BBOB competitions. Table 3 summarises[7] these observations along with a simple comparison between the static mode and the RL control. It shows that when there is margin for improvement for the specific EA/problem combination, the RL controller will exploit it.

Another factor that may contribute to the performance differences among EAs when using the RL controller is monotonicity (decided by restarting and elitism). The Simple ES is the only non-elitistic. The Cellular GA and GA MPC are elitistic and do not restart while the IPOP-10DDr CMA-ES typically restarted only four or five times per run. This is relevant if we look into the observables' values (we cannot provide graphs due to lack of space[8]). It is frequently seen that, during a run, observables will follow a monotonous increasing or decreasing curve (unless the EA restarts or is non-elitistic). This means that specific observations only occur once, thus, states do not repeat unless the EA is restarted[9]. Consequently, elitism combined with no restarting could mean that anything learned during a run is never actually used and the controller degenerates to solving a dynamic multi-armed bandit problem.

Looking at the parameter values set by the RL controller we could not discern any obvious pattern. Even when looking at one of the best performing runs for the RL control (Simple ES with the Rastrigin problem) in Figure 1 no apparent trends can be seen. However, that is not necessarily a problem considering the combined effect of multiple observables, the dynamic segmentation of the state space and the effects of exploratory actions.

For all experiments, the settings of the RL controller were fixed to the default values shown in Table 1 without making any further adjustments or tests. Though that is not enough to derive conclusions about the robustness of the RL controller, it shows that any improvements in performance reported in this paper were achieved with no additional effort for applying the RL controller to the EAs.

When comparing the performance with PRAM, the benchmark parameter controller, it can be seen that the RL controller outperforms PRAM in most cases, although for each algorithm there is

[5]The source code for the (IPOP) CMA-ES was acquired from https://www.lri.fr/ hansen/cmaes_inmatlab.html. The 10DDr variation was added manually.

[6]The source code for this experiment is available for download at http://www.few.vu.nl/ gks290/resources/gecco2014.tar.gz

[7]Directly derived from the raw data in Table 4.

[8]All graphs are available for download at http://www.few.vu.nl/ gks290/resources/gecco2014.tar.gz

[9]Non-elitism can result in "weak" restarting with the diversity and fitness hopping to large/small values every time the best individuals do not survive.

Table 2: Experimental Setup

EA	Parameters, ranges, default values	Problems	RL Control settings	PRAM settings
Simple ES	$\mu \in [10, 80]$, $\mu = 20$ $g \in (0, 7]$, $g = 2$ $\sigma \in (0, 2]$, $\sigma = 0.8$ $k_s \in [2, 80]$, $k_s = 2$	Rastrigin, Fletcher&Powell, Schaffer, Schwefel in 10 dimensions	Action bins: 5	$s(\mu) = 8$ $s(g) = 0.25$ $s(\sigma) = 0.1$ $s(k_s) = 3$
Cellular GA	$xover \in \{2p, Ar\}$, $xover = 2p$ $p_c \in [0.6, 1]$, $p_c = 0.9$ $mut \in \{G, U, G_d, C\}$, $mut = G$ $p_m \in (0, 0.4]$, $p_m = 0.1$ $m_{var} \in (0, 0.4]$, $m_{var} = 0.2$	BBOB2013 f_{21}, f_{22}, f_{23}, f_{24} in 40 dimensions	Action bins: 4	$s(p_c) = 0.04$ $s(p_m) = 0.04$ $s(m_{var}) = 0.04$
GA MPC	$\mu \in [50, 130]$, $\mu = 90$ $k_{max} \in [3, 15]$, $k_{max} = 3$ $\beta_m \in [0.3, 1.1]$, $\beta_m = 0.7$ $\beta_{std} \in (0, 0.5]$, $\beta_{std} = 0.1$ $p_r \in (0, 0.4]$, $p_r = 0.1$ $f_a \in [0.3, 0.7]$, $f_a = 0.5$	CEC2011 f_7, $f_{11.8}$, f_{21}, f_{22}	Action bins: 4	$s(\mu) = 6$ $s(k_{max}) = 1$ $s(\beta_m) = 0.05$ $s(\beta_{std}) = 0.05$ $s(p) = 0.04$ $s(f_a) = 0.04$
IPOP-10DDr CMA ES	$c_c \in [0, 1]$, $c_c = 0.0909$ $c_s \in [0, 1]$, $c_s = 0.1375$ $d \in [1, 5]$, $d = 1.1375$ $\mu_{cov} \in [1, 20]$, $\mu_{cov} = 4.5409$ $xover \in \{E, D_l, D_{sl}\}$, $xover = D_{sl}$	BBOB2013 f_{21}, f_{22}, f_{23}, f_{24} in 40 dimensions	Action bins: 4	$s(c_c) = 0.1$ $s(c_s) = 0.1$ $s(d) = 0.5$ $s(\mu_{cov}) = 2$

a problem where PRAM has superior performance to the RL controller. Finally, it is noteworthy that random variation of parameter values had, in many cases, a positive effect on performance, even for the more complex GA MPC. Such an effect has been observed before [11],[10] and is again confirmed here.

Table 3: RL Control vs static: ++ (or - -) shows that the ABF is significantly better (or worse) while + (or -) shows that ABF is not significantly different but the result of the best run is better (or worse). A 0 means no difference. Grey cells denote that the EA is particularly fit to the specific problem (see explanation in the text).

Simple ES	++	++	++	
Cellular GA	++	++	++	+
GA MPC	++	++	+	- -
CMA ES	0	0	- -	+

7. CONCLUSIONS

In this paper, we presented an RL-based generic parameter controller for EAs. The controller can easily be applied to any EA by simply specifying the parameters to be controlled, their ranges and the desired level of precision. We conducted experiments with four different EAs and 15 problems. The results show that the RL controller can enhance performance without the need for tweaking its parameters. A detailed analysis shows that the RL controller can exploit an existing margin of improvement, i.e. when the EA has not been fully tailored towards the particular problem at hand. Compared to two benchmark controllers (PRAM and random variation) the RL controller generally outperforms both of them. An analysis of values selected by the controller shows that, though the controller is able to improve performance, a pattern in the selection of parameter values is hard to discern. A surprising - but in retrospect logical - observation is that the EA's monotonicity can have an influence on the efficiency of the RL controller. Finally, another interesting observation is that random variation of parameter values can enhance the performance compared to static default values.

Regarding future work, we believe that a systematic exploration and evaluation of different observables is important. Currently, very little is known about the information contained in observables and their usefulness for controlling parameter values. In this work, we have selected observables that are frequently used in literature but with no underlying argumentation. Analysis of our results shows that these observables are inappropriate unless certain conditions hold (non-elitism or frequent restarts). Other future work includes the improvement of the design of the controller, an analysis of the sensitivity of the controller to its own parameters and to noise in observables as well as more rigorous testing with other EA/problem combinations.

8. REFERENCES

[1] E. Alba and B. Dorronsoro. *Cellular Genetic Algorithms*. Springer, Berlin, Heidelberg, New York, 1st edition, 2008.

[2] F. Chen, Y. Gao, Z. Chen, and S. Chen. SCGA: Controlling genetic algorithms with Sarsa(0). In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, volume 1, pages 1177–1183, 2005.

[3] A. Eiben, M. Horvath, W. Kowalczyk, and M. Schut. Reinforcement learning for online control of evolutionary algorithms. In Brueckner, Hassas, Jelasity, and Yamins, editors, *Proceedings of the 4th International Workshop on Engineering Self-Organizing Applications (ESOA'06)*, volume 4335, pages 151–160. Springer, 2006.

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

[5] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[6] A. E. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin Heidelberg, 2003.

[7] S. Elsayed, R. Sarker, and D. Essam. GA with a new multi-parent crossover for solving IEEE-CEC2011 competition problems. In *Proceedings of the 2011 IEEE*

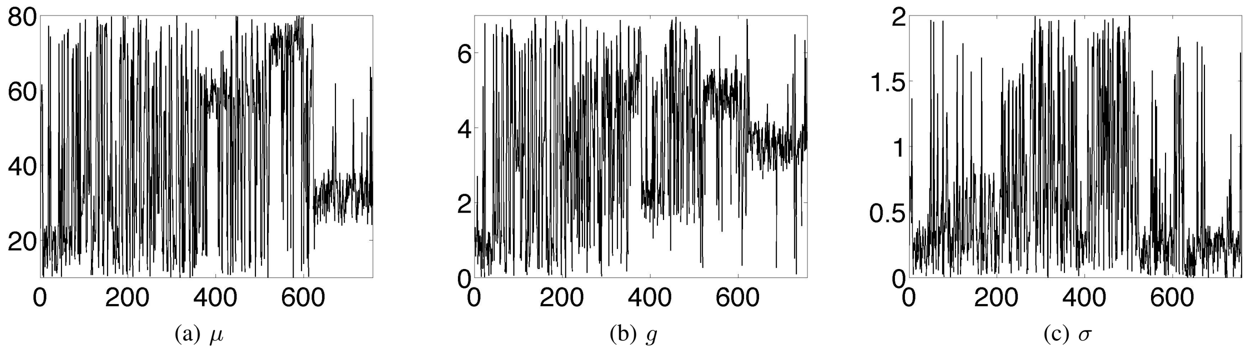

Figure 1: Parameter values over time for the best run of the Simple ES solving the Rastrigin function with RL control.

Congress on Evolutionary Computation, pages 1034–1040, New Orleans, USA, 2011. IEEE Press.

[8] H. Hasselt. Reinforcement learning in continuous state and action spaces. In Wiering and Otterlo [27], pages 207–251.

[9] N. J. Holtschulte and M. Moses. Benchmarking cellular genetic algorithms on the BBOB noiseless testbed. In *Proceeding of the Fifteenth Annual Conference Companion on Genetic and Evolutionary Computation Conference Companion*, GECCO '13 Companion, pages 1201–1208. ACM, 2013.

[10] G. Karafotias, M. Hoogendoorn, and A. Eiben. Why parameter control mechanisms should be benchmarked against random variation. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*, pages 349–355, Cancun, Mexico, 2013. IEEE Press.

[11] G. Karafotias, M. Hoogendoorn, and A. E. Eiben. Parameter control: strategy or luck? pages 215–216, New York, NY, USA, 2013. ACM.

[12] G. Karafotias, M. Hoogendoorn, and A. E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, to appear, 2014.

[13] G. Karafotias, S. Smit, and A. Eiben. A generic approach to parameter control. In C. Di Chio *et al*, editor, *Proceedings of EvoApplications 2012: Applications of Evolutionary Computation*, number 7248 in Lecture Notes in Computer Science, pages 366–375. Springer, Berlin, Heidelberg, New York, 2012.

[14] T. Liao and T. Stützle. Bounding the population size of IPOP-CMA-ES on the noiseless BBOB testbed. In *Proceeding of the Fifteenth Annual Conference Companion on Genetic and Evolutionary Computation Conference Companion*, pages 1161–1168. ACM, 2013.

[15] B. McGinley, J. Maher, C. O'Riordan, and F. Morgan. Maintaining healthy population diversity using adaptive crossover, mutation, and selection. *Evolutionary Computation, IEEE Transactions on*, 15(5):692 –714, 2011.

[16] S. Muller, N. Schraudolph, and P. Koumoutsakos. Step size adaptation in evolution strategies using reinforcement learning. In *2002 Congress on Evolutionary Computation (CEC'2002)*, pages 151–156, Honolulu, USA, 12-17 May 2002. IEEE Press, Piscataway, NJ.

[17] J. Pettinger and R. Everson. Controlling genetic algorithms with reinforcement learning. In W. Langdon *et al*, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 692–. Morgan Kaufmann, San Francisco, 9-13 July 2002.

[18] Y. Sakurai, K. Takada, T. Kawabe, and S. Tsuruta. A method to control parameters of evolutionary algorithms by using reinforcement learning. In *Signal-Image Technology and Internet-Based Systems (SITIS), 2010 Sixth International Conference on*, pages 74–79, 2010.

[19] S. Smit and A. E. Eiben. Population diversity index: A new measure for population diversity. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2011.

[20] M. Spaan. Partially observable markov decision processes. In M. Wiering and M. Otterlo, editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 387–414. Springer Berlin Heidelberg, 2012.

[21] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998.

[22] W. B. Uther and M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 769–774. American Association for Artificial Intelligence, 1998.

[23] M. van Otterlo and M. Wiering. Reinforcement learning and Markov Decision Processes. In Wiering and Otterlo [27], pages 3–42.

[24] M. Črepinšek, S.-H. Liu, and M. Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Comput. Surv.*, 45(3):35:1–35:33, 2013.

[25] N. Veerapen, J. Maturana, and F. Saubion. A comparison of operator utility measures for on-line operator selection in local search. In *Proceedings of the 6th international conference on Learning and Intelligent Optimization*, LION'12, pages 497–502. Springer-Verlag, 2012.

[26] J. Whitacre, T. Pham, and R. Sarker. Use of statistical outlier detection method in adaptive evolutionary algorithms. In M. Keijzer, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006)*, pages 1345–1352. Morgan Kaufmann, San Francisco, 2006.

[27] M. Wiering and M. Otterlo, editors. *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*. Springer Berlin Heidelberg, 2012.

[28] Y.-Y. Wong, K.-H. Lee, K.-S. Leung, and C.-W. Ho. A novel approach in parameter adaptation and diversity maintenance for genetic algorithms. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 7:506–515, 2003.

Table 4: Results of all experiments. There are four sections, one for each algorithm. Every section includes three or four subtables with the results for each test problem. Subtables show the performance of the algorithm with that problem with four parameter modes: static default, controlled with the RL controller, controlled with PRAM, and randomly varied. Performance is shown in terms of final fitness averaged over all repeats (ABF), final fitness of the best and worst runs and the number of evaluations until the best fitness of the run is reached averaged over all repeats (AEB). In the ABF and AEB columns underlined values are significantly better than static and bold values denote the winner(s) (not significantly worse than the best). Significant difference was decided when a two-sided Kolmogorov-Smirnov test with $\alpha = 0.05$ rejected the null hypothesis that two samples came from the same distribution. All problems are minimisation.

	Simple ES											
	Rastrigin				Schaffer				Fletcher & Powell			
	ABF	Best	Worst	AEB	ABF	Best	Worst	AEB	ABF	Best	Worst	AEB
static	27.681	20.718	34.808	**510033**	22.036	6.934	42.749	**664785**	7935.4	1982.5	11004	**505854**
RL	<u>**0.672**</u>	0.039	2.423	**491921**	<u>9.851</u>	1.079	29.226	**579073**	<u>**166.211**</u>	3.424	726.75	**586305**
PRAM	<u>11.762</u>	2.184	32.039	**533587**	<u>4.253</u>	0.908	16.492	**708175**	<u>**1022.72**</u>	15.557	7548.2	752212
random	<u>4.645</u>	1.630	8.325	**591095**	<u>**3.071**</u>	1.982	10.527	**667068**	<u>420.260</u>	187.938	768.02	**575475**

	Cellular GA											
	BBOB f_{21}				BBOB f_{22}				BBOB f_{23}			
	ABF	Best	Worst	AEB	ABF	Best	Worst	AEB	ABF	Best	Worst	AEB
static	40.891	40.805	41.139	**973913**	-998.82	-999.08	-998.49	979100	9.062	8.573	9.472	764053
RL	<u>40.800</u>	40.784	40.863	998730	<u>-999.18</u>	-999.26	-999.05	999350	**8.486**	8.148	8.906	906923
PRAM	<u>**40.780**</u>	40.780	40.781	**943080**	<u>**-999.29**</u>	-999.49	-999.20	<u>**900960**</u>	9.282	8.443	9.999	<u>**495420**</u>
random	<u>40.843</u>	40.795	40.973	999210	<u>-998.91</u>	-999.11	-998.69	999220	<u>8.643</u>	8.296	9.094	923773
	BBOB f_{24}											
	ABF	Best	Worst	AEB								
static	**471.809**	435.148	511.297	899713								
RL	**484.621**	412.925	565.928	986693								
PRAM	493.931	412.911	598.025	<u>**502450**</u>								
random	490.169	422.008	544.902	996706								

	GA MPC											
	CEC2011 f_7				CEC2011 $f_{11.8}$				CEC2011 f_{12}			
	ABF	Best	Worst	AEB	ABF	Best	Worst	AEB	ABF	Best	Worst	AEB
static	1.677	0.837	2.028	**100626**	1952741	1510399	2505116	**87535**	**16.300**	14.110	22.675	**149491**
RL	<u>**0.963**</u>	0.627	1.396	124440	<u>**947843**</u>	941267	955770	119430	**16.443**	12.589	20.512	149829
PRAM	1.552	0.586	1.969	**95914**	<u>1562083</u>	941087	3438794	**98474**	**16.213**	14.232	17.678	149968
random	<u>**0.893**</u>	0.584	1.219	142052	<u>**946728**</u>	939736	957187	132381	**16.541**	13.718	20.338	149698
	CEC2011 f_{13}											
	ABF	Best	Worst	AEB								
static	**15.211**	8.609	22.207	149335								
RL	19.329	14.119	26.764	**149873**								
PRAM	17.342	8.861	25.672	<u>**148652**</u>								
random	17.797	8.787	24.957	**149828**								

	IPOP-10DDr CMA ES											
	BBOB f_{21}				BBOB f_{22}				BBOB f_{23}			
	ABF	Best	Worst	AEB	ABF	Best	Worst	AEB	ABF	Best	Worst	AEB
static	**41.390**	40.780	43.252	**163539**	**-997.666**	-999.308	-985.415	**115916**	**6.889**	6.870	6.977	**604791**
RL	**41.689**	40.780	43.252	312667	**-997.753**	-999.308	-981.711	**193861**	6.902	6.875	6.980	**673839**
PRAM	41.985	40.780	43.949	**196793**	**-997.626**	-999.308	-981.711	**184533**	6.910	6.874	6.974	**705917**
random	**41.751**	40.780	43.949	396004	**-997.876**	-999.308	-985.415	**238268**	**6.942**	6.871	7.268	**568693**
	BBOB f_{24}											
	ABF	Best	Worst	AEB								
static	**131.541**	108.027	150.419	**416381**								
RL	**128.907**	106.931	152.292	**483242**								
PRAM	**137.570**	109.712	187.557	**594935**								
random	**126.457**	107.754	153.392	561017								