Hybridizing the 1/5-th Success Rule with Q-Learning for Controlling the Mutation Rate of an Evolutionary Algorithm

Arina Buzdalova, Carola Doerr, Anna Rodionova

Hybridizing the 1/5-th Success Rule with Q-Learning for Controlling the Mutation Rate of an Evolutionary Algorithm

Arina Buzdalova[1], Carola Doerr[2], and Anna Rodionova[1]

[1] ITMO University, 49 Kronverkskiy ave., Saint Petersburg, Russia, 197101
abuzdalova@gmail.com
[2] Sorbonne Université, CNRS, LIP6, Paris, France
Carola.Doerr@lip6.fr

Abstract. It is well known that evolutionary algorithms (EAs) achieve peak performance only when their parameters are suitably tuned to the given problem. Even more, it is known that the best parameter values can change during the optimization process. Parameter control mechanisms are techniques developed to identify and to track these values.
Recently, a series of rigorous theoretical works confirmed the superiority of several parameter control techniques over EAs with best possible static parameters. Among these results are examples for controlling the mutation rate of the $(1 + \lambda)$ EA when optimizing the OneMax problem. However, it was shown in [Rodionova et al., GECCO'19] that the quality of these techniques strongly depends on the offspring population size λ.
We introduce in this work a new hybrid parameter control technique, which combines the well-known one-fifth success rule with Q-learning. We demonstrate that our HQL mechanism achieves equal or superior performance to all techniques tested in [Rodionova et al., GECCO'19] and this – in contrast to previous parameter control methods – simultaneously for all offspring population sizes λ. We also show that the promising performance of HQL is not restricted to OneMax, but extends to several other benchmark problems.

Keywords: Parameter Control · Q-Learning · Offspring Population Size

1 Introduction

The problem of selecting suitable parameter configurations for an evolutionary algorithm is frequently considered to be one of the most essential drawbacks of evolutionary computation methods, and possibly a major obstacle towards wider application of these optimization techniques in practice [30].

Automated configuration techniques such as SPOT [3], irace [31], SMAC [23], hyperband [29], MIP-EGO [41], BOHB [21], and many others have been developed to assist the user in the decisive task of selecting suitable parameter configurations. These *parameter tuning* methods, however, require to test different parameter combinations before presenting a recommendation. They are therefore rather time-consuming, and are not applicable when the possibility for such training is not given, e.g., when the problem is truly black-box, with no/only little information about its fitness landscape structure.

An orthogonal approach to solve the algorithm configuration problem is *parameter control*, which does not require a priori training, and aims at identifying suitable parameter combinations *on the fly*, i.e., while executing the optimization [20, 26, 30]. Apart from being more generally applicable than parameter tuning, parameter control also bears the advantage of being able to adjust the search behavior of the evolutionary algorithm to the different stages of the optimization process. Most state of the art evolutionary algorithms therefore make use of parameter control, in particular in the continuous domain, where a decreasing search radius is needed to eventually converge towards an optimal point. However, one should not forget that parameter control mechanisms, too, introduce their own hyperparameters, which need to be adequately set by the user prior to running the algorithm. Here again one can apply parameter tuning (e.g., via so-called per-instance algorithm configuration [4]), but the general hope is that the setting of the hyperparameters is less critical to achieve reasonable performance.

However, while parameter control is routinely used in numerical optimization, its potential remains far from being well exploited in the optimization of problems with discrete decision variables, where it has only recently re-gained momentum as a now very active area of research. In particular in the sub-domain of runtime analysis, parameter control has enjoyed rising attention in the last years, as summarized in [9].

A particularly well-researched topic in the theory literature for parameter control in discrete optimization heuristics is the $(1 + \lambda)$ Evolutionary Algorithm (EA) with dynamic mutation rates and fixed offspring population size λ optimizing the OneMax problem (the problem of controlling λ has also been addressed, e.g., in [28], but has received much less attention so far). Not only was this problem one of the first ones for which dynamic mutation schemes were approximated [2], and not only is it frequently used as a test case for empirical works [6], but it is also one of the few problems for which we have a very solid theoretical understanding.

Extending the previous work from [17], we have presented at GECCO'19 a comparative empirical study of several mechanisms suggested in the theory literature [36]. Among other findings, we demonstrated that the efficiency of all benchmarked techniques depends to a large extent on the offspring population size λ. For example, we observed that the 2-rate $(1 + \lambda)$ EA suggested in [14] is the best among the tested algorithms when λ is smaller than 50. For larger offspring population sizes, however, this algorithm is outperformed by a $(1+\lambda)$ EA which uses the one-fifth success rule to control the mutation rate. We also observed in [36] that the ranking of the algorithms was identical for all tested dimensions $n \in [10^4..10^5]$.

Our Results. The results presented in [36] raise the question if one can achieve stable performance across all offspring population sizes λ. We address this problem by introducing a new parameter control scheme, which hybridizes the one-fifth success rule with Q-learning. More precisely, we first introduce the $(1 + \lambda)$ QEA, which uses Q-learning only to control the mutation rate. The $(1 + \lambda)$ QEA learns for each optimization state whether it should increase or decrease the current mutation rate (we use constant factor changes). We show that the $(1+\lambda)$ QEA performs efficiently on OneMax for all observed values of λ when an appropriate lower bound $p_{\min}$ for the mutation rate is used. In absence of a well-tuned lower bound, however, the performance of the $(1 + \lambda)$ QEA drops significantly. We show that this dependence on the value of $p_{\min}$ can be mitigated by a hybridization of the $(1 + \lambda)$ QEA with the one-fifth success rule. More precisely, the hybrid Q-learning EA (the $(1 + \lambda)$ HQEA) extends the $(1 + \lambda)$ QEA by using the one-fifth success rule in states that have not been visited before and for those for which the $(1 + \lambda)$ QEA is ambiguous with respect to the two available actions.

We show that, on OneMax, the $(1+\lambda)$ HQEA outperforms or at least performs on par with all algorithms tested in [36], and this simultaneously for all tested values of $\lambda \in [1..2^{12}]$ and also for both considered lower bounds for the mutation rate, $p_{\min} = 1/n$ and $p_{\min} = 1/n^2$, respectively. It therefore solves the issue of the other control mechanisms previously suggested in the theory literature. Note here that we do not have a theoretical convergence analysis of the $(1 + \lambda)$ HQEA. Given its complexity, it may be beyond the current state of the art in runtime analysis, as it requires to keep track of multiple states, which are highly dependent. We are nevertheless confident that the robust performance of the $(1 + \lambda)$ HQEA encourages further work on learning-based parameter control, and their hybridization with other classical control methods.

In the last parts of this paper we also show that the promising performance of the $(1 + \lambda)$ HQEA is not restricted to OneMax. More precisely, we show that it performs well also on the LeadingOnes function, as well as on several benchmark functions suggested in [16].

Related Work. We are not the first to use reinforcement learning (RL) as a parameter control technique. An exhaustive survey of RL-based parameter control approaches can be found in [26]. Particularly, there are parameter control approaches based on techniques for the *Multi-Armed Bandit Problem (MAB)*, see [22] (and references mentioned therein) and [12] for a theoretical investigation of MAB-based parameter control.

In many of the known approaches, RL algorithms are used to select the parameter values directly. For numerical parameters, however, most common techniques require to either discretize the value space [24] or to make use of quite sophisticated techniques [1, 19, 37], which are rather difficult to grasp without expert knowledge.

In contrast to such a direct selection of the parameter values, we use in this work an indirect approach which uses as actions the possibility to increase the current parameter value by some fixed multiplier, or decrease it. As we shall see below, this yields a simple, yet efficient, control mechanism. Like most common parameter control techniques, including those studied in this work, this indirect approach has the advantage of a smoother transition of the mutation rates between consecutive iterations. This behavior is beneficial if the optimal parameter values do not change abruptly, which is the case in many problems analyzed in theoretical works [10, 13], but also the case in many applications of evolutionary algorithms to machine learning problems, including hyperparameter optimization itself [34]. Exceptions to this rule exist, of course, and the jump functions [18] are a classical example for a problem requiring such an abrupt change. In such cases it may take the the parameter control mechanisms some time to adjust the mutation rate to the appropriate scale.

We note that a similar indirect control approach has been described in [33], where an indirect control of the step size of the (1+1) evolution strategy (ES) is described. In contrast to our work, however, this approach (which uses SARSA – another common reinforcement learning algorithm – instead of Q-learning) did not manage to outperform the (1+1) ES with suitably tuned static step sizes.

2 Previous $(1 + \lambda)$ EAs with Dynamic Mutation Rates

We briefly review the algorithms studied in [36] and summarize their main findings. We assume in our presentation that the algorithms operate on a problem $f : \{0, 1\}^n \to \mathbb{R}$, with the objective to maximize this function.

The $(1 + \lambda)$ EA. The standard $(1 + \lambda)$ EA is an elitist algorithm, which always keeps a current best solution x in its memory. The $(1 + \lambda)$ EA is initialized with a point chosen from the search space $\{0, 1\}^n$ uniformly at random. In each iteration, λ *offspring* are sampled by applying standard bit mutation to the *parent* x, i.e., the algorithm creates λ offspring $y^{(1)}, \ldots, y^{(\lambda)}$ by creating λ copies of x and flipping each bit in these copies with some probability $0 < p < 1$. The variable p is commonly referred to as the *mutation rate*. We set it to $p = 1/n$ in our experiments, which is a standard recommendation and often a fall-back value if no indication is given that larger values could be beneficial. The best of the λ offspring (ties broken uniformly at random) replaces the parent if it is at least as good. The $(1 + \lambda)$ EA continues until some user-defined termination criterion is met (see "implementation details" below for our setting).

The $(1+\lambda)$ EA(A, b). The $(1+\lambda)$ EA(A, b) extends the $(1+\lambda)$ EA by an adaptive choice of the mutation rate p. Its (1+1) variant was suggested in [15], and we use a straightforward extension to the $(1 + \lambda)$ EA by updating the mutation rate p by Ap if the best of the λ offspring is at least as good as the parent and by decreasing the mutation rate to bp otherwise. It is ensured that the mutation rate does not fall below some minimal mutation rate $p_{\min} > 0$ and that it does not exceed $p_{\max} = 1/2$, by capping the value of p appropriately where required. As argued in [11], this update rule is essentially a one-fifth success rule, even if this term was not mentioned in [15]. The one-fifth success rule was originally suggested in [8,35,38] and its interpretation for the discrete optimization is due to [27]. More precisely, the idea is that the mutation rate should remain constant if a certain ratio of iterations is successful (i.e., produces a solution of better than previous-best quality). In our work, this success ratio is 1/2, whereas the traditional rule suggests a success ratio of 1/5.

The $(1 + \lambda)$ EA(A, b) has three hyperparameters, A, b, and $p_{\min}$. In our experiments, we set $A = 2$, $b = 1/2$, and consider $p_{\min} \in \{1/n, 1/n^2\}$. We initialize p by $1/n$. Note that these values are not specifically tuned, but we chose them to be consistent with previous works, and in particular with [36]. The reader interested in the sensitivity of the performance of the $(1 + \lambda)$ EA(A, b) with respect to these parameters is referred to [15] and [11] for an empirical and a theoretical investigation, respectively.

The 2-rate $(1+\lambda)$ EA$_{r/2,2r}$. The $(1+\lambda)$ EA$_{r/2,2r}$ suggested in [14] uses two different mutation rates in each iteration: half the offspring are created with mutation rate $p/2$ and the other $\lambda/2$ offspring are sampled with mutation rate $2p$. The mutation rate is parametrized as $p = r/n$ in the $(1 + \lambda)$ EA$_{r/2,2r}$. The value of r is updated after each iteration by a random decision which gives preference to the rate by which the best offspring has been created. The latter is selected with probability 3/4, whereas the other one of the two

tested mutation rates is chosen with probability 1/4. As in the $(1+\lambda)$ EA(A,b), the mutation rate is capped at $p_{\min} \in \{1/n^2, 1/n\}$ and $p_{\max} = 1/2$, respectively.

Implementation details. We briefly summarize a few common assumptions made in all our algorithms.

Shift Mutation Strategy. All algorithms described above use standard bit mutation as variation operator. To avoid sampling offspring that are identical to the parent (these offspring would not bring any new information to our optimization process, and are therefore useless), we use the "shift" operation suggested in [5]. If an offspring equals its parent, this strategy simply flips a randomly chosen bit. We write $y \leftarrow \text{mutate}(x,p)$ if y is sampled by applying the shift mutation operator with mutation rate p to x.

Termination Criterion and Runtime Measure. We focus in this work on the *runtime* (also known as *optimization time*), which we measure in terms of generations that are needed until an optimal solution is evaluated for the first time. Since we only study algorithms with static offspring population size λ, the classical runtime in terms of function evaluations is easily obtained by multiplication with λ. As common in the academic benchmarking of EAs, our termination criterion is thus the state $f(x) = \max\{f(y) \mid y \in \{0,1\}^n\}$.

Strict vs. Non-Strict Update Rules. We have presented in the previous section the algorithms as originally suggested in the literature. However, in our initial experiments we have made an interesting observation that the $(1+\lambda)$ EA(A,b) can substantially benefit from a slightly different parameter update rule, which replaces p by Ap only if the best offspring y is *strictly better* than the parent, i.e., if it satisfies $f(y) > f(x)$. We perform all experiments for the strict and the classical (non-strict) update rules, which – together with the two lower bounds $p_{\min} = 1/n^2$ and $p_{\min} = 1/n$ – yields four different settings for each benchmark problem. For reasons of space we can only comment on a few selected cases below. The detailed results are available in the appendix. We mostly focus on the case of the strict update rule, if not stated otherwise.

3 Hybridizing Q-Learning and the 1/5-th Success Rule

The main contribution of our work is an algorithm that avoids the drawbacks of the above-mentioned $(1+\lambda)$ EA variants observed on OneMax, and shows stable performance for all values of λ. We will achieve this by hybridizing the $(1+\lambda)$ EA(A,b) with Q-learning.

Q-learning is a method that falls into the broader category of reinforcement learning (RL). Q-learning aims at learning, from the data that it observes, a policy that tells an *agent* which *action* to apply in a given situation. For this, it maintains a state-action matrix, in which it records its guess for what the expected *reward* of each action in each of the states is. For a given *state* s, the action a maximizing this expected reward is chosen and executed. The environment returns a numerical reward and a representation of its state. The reward is used to update the state-action matrix, according to some rules that we shall discuss in the next paragraphs. The Q-learning process repeats until some termination criterion is met. The goal of the agent is to maximize the total reward. A smooth introduction to RL can be found in [39].

The $(1+\lambda)$ QEA. We apply Q-learning to control the mutation rate of the $(1+\lambda)$ EA with fixed offspring population size λ. We first present in Alg. 1 the basic $(1+\lambda)$ QEA. Its hybridization with the 1/5-th success rule will be explained further below. The $(1+\lambda)$ QEA considers only two actions: whether to multiply the current mutation rate p by the factor $A > 1$ (action a_{mult}) or whether to multiply it by the factor $b < 1$ (action a_{divide}). As mentioned in the introduction, the advantage of this action space is a smooth transition of the mutation rates between consecutive iterations, compared to a possibly abrupt change when operating directly on the parameter values.

We use as reward the relative fitness gain, i.e., $(\max f(y^{(i)}) - f(x))/f(x)$ (where we use the same notation as in the description of the $(1+\lambda)$ EA, i.e., x denotes the parent individual and $y^{(1)}, \dots, y^{(\lambda)}$ its λ offspring). This reward is computed in line 12. Note here that several other reward definitions would have been possible. We tried different suggestions made in [25] and found this variant to be the most efficient. The new state s' is computed as the number of offspring $y^{(i)}$ that are strictly better than the parent (lines 13-16). With the reward and the new state at hand, the efficiency estimation $Q(s,a)$ is updated in line 18, through

Algorithm 1: The $(1+\lambda)$ QEA, Q-learning highlighted in blue font

1 **Input:** population size λ, learning rate α, learning factor γ;
2 **Initialization:**
3 $x \leftarrow$ random string from $\{0,1\}^n$;
4 $p \leftarrow 1/n$;
5 **for** *all states* $s_i \in [0 \dots \lambda]$ *and all actions* $a_i \in \{a_{mult}, a_{divide}\}$ **do** $Q(s_i, a_i) \leftarrow 0$;
6 $s, a \leftarrow$ undefined;
7 **Optimization: while** *termination criterion not met* **do**
8 **for** $i = 1, \dots, \lambda$ **do** $y^{(i)} \leftarrow \text{mutate}(x, p)$;
9 $x^* \leftarrow \arg\max_{y^{(i)}} f(y^{(i)})$;
10 $x_{\text{old}} \leftarrow x$;
11 **if** $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;
12 $r \leftarrow \frac{f(x^*)}{f(x_{\text{old}})} - 1$; `// reward calculation`
13 $s' \leftarrow 0$;
14 **for** $i = 1, \dots, \lambda$ **do**
15 **if** $f(y^{(i)}) > f(x_{old})$ **then**
16 $s' \leftarrow s' + 1$; `// state calculation`
17 **if** $s \neq$ *undefined* ***and*** $a \neq$ *undefined* **then**
18 $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma \max_{a'} Q(s', a') - Q(s,a))$;
19 $s \leftarrow s'$;
20 **if** $Q(s', a_{mult}) = Q(s', a_{divide})$ **then**
21 $a \leftarrow$ select a_{mult} or a_{divide} equiprobably;
22 **else**
23 $a \leftarrow \arg\max_{a'} Q(s', a')$;
24 $p \leftarrow ap$; `// update mutation rate`
25 $p \leftarrow \min(\max(p_{\min}, p), p_{\max})$; `// capping mutation rate`

a standard Q-learning update rule. Note here that action a is the one that was selected in the previous iteration (lines 20-23), and it resulted in moving from the previous state s to the current state s'.

After this update, the $(1+\lambda)$ QEA selects the action to be used in the next iteration, through simple greedy selection if possible, and through an unbiased random choice otherwise; see lines 20-23. The mutation rate p is then updated by this action (line 24) and capped to remain within the interval $[p_{\min}, p_{\max}]$ if needed (line 25).

Hyperparameters. The $(1+\lambda)$ QEA has six hyperparameters, the constant factors of the actions a_{mult} and a_{divide}, the upper and lower bounds for the mutation rate $p_{\min}$ and $p_{\max}$, and two hyperparameters originating from the Q-learning methodology itself (line 18), the *learning rate* α and the *discount factor* γ. In our experiments, we use $a_{\text{mult}} = 2$, $a_{\text{divide}} = 1/2$, $p_{\max} = 1/2$, $\alpha = 0.8$, and $\gamma = 0.2$. These values were chosen in a preliminary tuning step, details of which we have to leave for the full report due to space restrictions. For $p_{\min}$ we show results for two different values, $1/n^2$ and $1/n$, just as we do for the other parameter control mechanisms.

The $(1+\lambda)$ HQEA, the Hybrid Q-learning EA. In the hybridized $(1+\lambda)$ QEA, the $(1+\lambda)$ HQEA, we reconsider the situation when the $Q(s,a)$ estimations are equal. This situation arises in two cases: when the state s is visited for the first time or when the same estimation was learned for both actions a_{mult} and a_{divide}. In these cases, the learning mechanism cannot decide which action is better, and an action is selected uniformly randomly. The $(1+\lambda)$ HQEA, in contrast, borrows in this case the update rule from the $(1+\lambda)$ EA(A,b) algorithm, i.e., action a_{mult} is selected if the best offspring is strictly better than the parent, otherwise a_{divide} is chosen. Formally, we obtain the $(1+\lambda)$ HQEA by replacing in Alg. 1 line 21 by the following text:

$$\textbf{if } f(x^*) > f(x_{\text{old}}) \textbf{ then } a \leftarrow a_{\text{mult}} \textbf{ else } a \leftarrow a_{\text{divide}}. \tag{1}$$

Strict vs. Non-Strict Update Rules. As mentioned at the end of Sec. 2, we experiment both with a strict and a non-strict update rule. Motivated by the better performance of the strict update rule, the description of the $(1 + \lambda)$ QEA and the $(1 + \lambda)$ HQEA use this rule. The non-strict update rules can be obtained from Alg. 1 by replacing the strict inequality in line 15 by the non-strict one. Similarly, for the $(1 + \lambda)$ HQEA, we also replace "**if** $f(x^*) > f(x_{\text{old}})$" in (1) by "**if** $f(x^*) \geq f(x_{\text{old}})$".

4 Empirical comparison of parameter control algorithms

We now demonstrate that, despite the seemingly minor change, the $(1 + \lambda)$ HQEA outperforms both its origins, the $(1 + \lambda)$ QEA and the $(1 + \lambda)$ EA(A, b), on several benchmark problems. We recall that the starting point of our investigations were the results presented in [36], which showed that the performance of the $(1 + \lambda)$ EA variants discussed in Sec. 2 on OneMax strongly depends on (1) the offspring population size λ, and on (2) the bound $p_{\min}$ at which we cap the mutation rate. The $(1 + \lambda)$ HQEA, in contrast, is shown to yield stable performance for all tested values of λ and for both tested values of $p_{\min}$.

Experimental setup. All results shown below are simulated from 100 independent runs of each algorithm. We report statistics for the optimization time, i.e., for the random variable counting the number of steps needed until an optimal solution is queried for the first time. Since the value of λ is static, we report the optimization times as number of generations; classical running time in terms of function evaluations can be obtained from these values by multiplying with λ. For OneMax, we report average optimization times, for consistency with the results in [36] and with theoretical results. However, for some of the other benchmark problems, the dispersion of the running times can be quite large, so that we report median values and interquartile ranges instead. Please also note that we use logarithmic scales in all runtime plots.

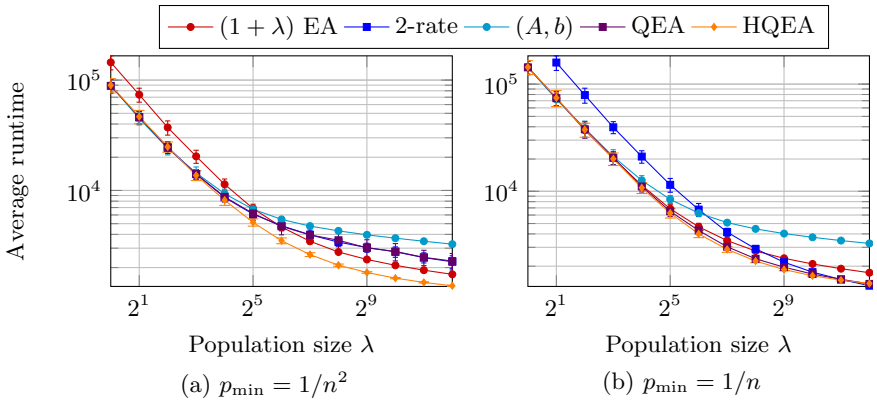
In the cases of large dispersion, we also performed the rank-sum Wilcoxon test to question statistical significance [7]. More precisely, we compared the $(1 + \lambda)$ HQEA to each of the other algorithms. As the input data for the test, the runtimes of all 100 runs of each of the two compared algorithms were used. The significance level was set to $p_0 = 0.01$.

The value of λ is parameterized as 2^t, with t taking all integer values ranging from 0 to 12 for OneMax and from 0 to 9 for all other problems. The problem dimension, in contrast, is chosen in a case-by-case basis. We recall that it was shown in [36] that the dimension did not have any influence on the ranking of the algorithms on OneMax. This behavior can be confirmed for the here-considered algorithm portfolio (results not shown due to space limitations).

4.1 Stable Performance on OneMax

Fig. 1 summarizes our empirical results for the 10^4-dimensional OneMax problem, the problem of maximizing the function $\text{Om} : \{0,1\}^n \to [0..n], x \mapsto \sum_{i=1}^{n} x_i$. For $p_{\min} = 1/n^2$, our key findings can be summarized as follows. **(i)** For small λ up to 2^4, all the parameter control algorithms perform similarly and all of them seem to be significantly better than the $(1 + \lambda)$ EA with static mutation rates. **(ii)** Starting from $\lambda > 2^5$ for the $(1 + \lambda)$ EA(A, b) and from $\lambda > 2^6$ for the $(1 + \lambda)$ QEA and the $(1 + \lambda)$ EA$_{r/2,2r}$, these algorithms are outperformed by the $(1 + \lambda)$ EA. **(iii)** The $(1 + \lambda)$ HQEA is the only parameter control algorithm that substantially improves the performance of the $(1+\lambda)$ EA for all considered values of λ. The advantage varies from 21% for $\lambda = 2^{12}$ to 38% for $\lambda = 1$.

For the less generous $p_{\min} = 1/n$ lower bound, we observe the following.**(i)** Overall, the performance is worsened compared to the $1/n^2$ lower bound. In particular, for small values of λ, most of the algorithms are indistinguishable from the $(1 + \lambda)$ EA, except for the $(1 + \lambda)$ EA$_{r/2,2r}$, which is even substantially worse. **(ii)** However, for $\lambda \geq 2^9$, the $(1 + \lambda)$ EA$_{r/2,2r}$ starts to outperform the $(1 + \lambda)$ EA, in strong contrast to the situation for the $1/n^2$ lower bound. **(iii)** Our $(1 + \lambda)$ HQEA is the only method which is never worse than the $(1 + \lambda)$ EA and still outperforms it for $\lambda > 2^6$. With the growth of λ, the advantage grows as well: while the $(1 + \lambda)$ EA with $\lambda = 2^{12}$ needs 1738 generations, on average, the $(1 + \lambda)$ HQEA only requires 1379 generations, an advantage of more than 20%. **(iv)** It is worth noting that the $(1 + \lambda)$ QEA in this case performs on par with the $(1 + \lambda)$ HQEA.



(a) $p_{\min} = 1/n^2$ (b) $p_{\min} = 1/n$

Fig. 1: Average number of generations and its standard deviation needed to locate the optimum of the OneMax problem

Overall, we thus see that the $(1 + \lambda)$ HQEA is the only considered parameter control algorithm, which stably performs on par or better than the $(1 + \lambda)$ EA and all of the other algorithms for all values of λ and for both values $p_{\min} \in \{1/n^2, 1/n\}$.

4.2 Stable Performance on Other Benchmark Problems

LeadingOnes. The LeadingOnes problem asks to maximize functions of the type $\text{Lo}_{z,\sigma} : \{0,1\}^n \to \mathbb{R}, x \mapsto \max\{i \in [n] \mid \forall j \leq i : x_{\sigma(i)} = z_{\sigma(i)}\}$, where σ is simply a permutation of the indices $1, \ldots, n$ (the classic Lo function uses the identity). We study the $n = 10^3$-dimensional variant of this problem.

For $p_{\min} = 1/n^2$ all the methods – including the $(1 + \lambda)$ EA– show very similar performance, with the difference between the best and the worst of the five algorithms varying from 3% to 6% for each offspring population size λ, which is of the same order as the corresponding standard deviations. For the $1/n$ lower bound, the situation is similar, except that the $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ performs substantially worse than the $(1 + \lambda)$ EA for all considered values of λ, and the difference varies from 45% to 93%.

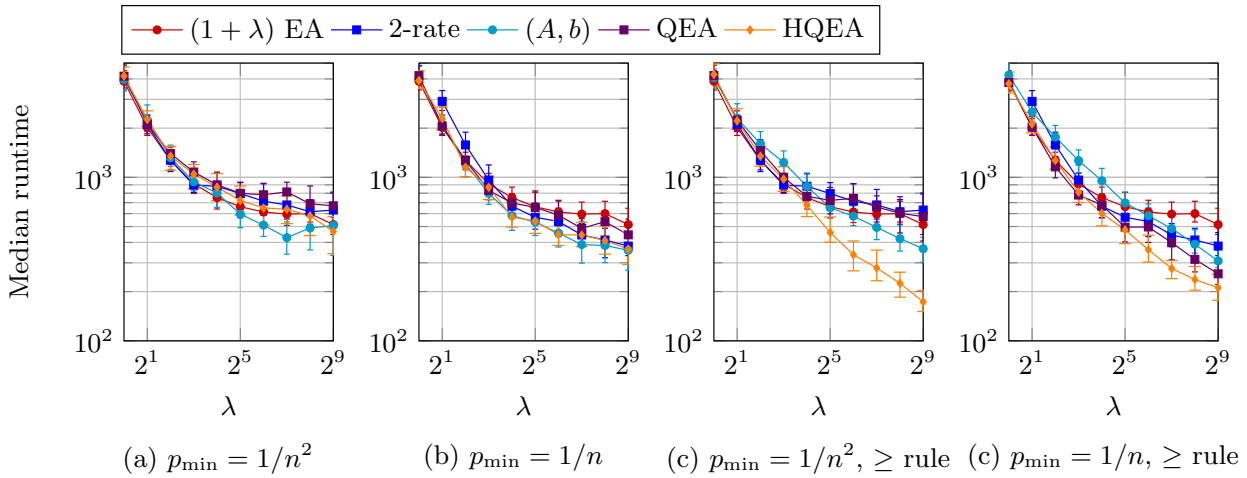
As a result, the $(1 + \lambda)$ HQEA generally performs on par with the $(1 + \lambda)$ EA for all considered values of λ and both considered lower bounds on the mutation rate. Particularly, for $p_{\min} = 1/n^2$ it is strictly better in 6 of the 10 cases, and in the other cases the disadvantages are 0.3%, 0.3%, 0.7%, and 1.1%.

Neutrality. The Neutrality function is a W-model transformation [42] that we apply to OneMax. It is calculated the following way: a bit string x is split into blocks of length k each, and each block contributes 0 or 1 to the fitness value according to the majority of values within the block. In line with [42] and [16] we considered $k = 3$. We study the $n = 10^3$-dimensional version of this problem. The results are summarized in Fig. 2.

For $p_{\min} = 1/n^2$ we obtain the following observations. Most of the parameter control methods perform poorly, i.e. worse than the $(1 + \lambda)$ EA. The exception is $(1 + \lambda)$ $\text{EA}(A, b)$, which performs better than the $(1 + \lambda)$ EA for several values of λ (in particular, $\lambda = 2^6, 2^7$).

The lower bound $p_{\min} = 1/n$ turns out to be preferable for all the algorithms: for large offspring population sizes λ, they all perform better than the standard $(1 + \lambda)$ EA. Our $(1 + \lambda)$ HQEA is usually one of the best algorithms, but however, for $\lambda = 2^7$ and $\lambda = 2^8$ it seems to be worse than the $(1+\lambda)$ $\text{EA}(A, b)$. The Wilcoxon test results did not confirm the significance of this difference though (the p-values are greater than 0.04 in both cases).

For this problem we also observe that switching from the strict update rule to the non-strict version is beneficial for the $(1 + \lambda)$ HQEA, the $(1 + \lambda)$ QEA, and the $(1 + \lambda)$ $\text{EA}(A, b)$, regardless of the value of $p_{\min}$. It is worth noting that with these values of hyper-parameters the $(1 + \lambda)$ HQEA performs significantly better on high values of the population size ($\lambda \geq 2^5$) than all the other considered methods (the p-values are between $1.6 \cdot 10^{-9}$ and $3.9 \cdot 10^{-18}$).



(a) $p_{\min} = 1/n^2$ (b) $p_{\min} = 1/n$ (c) $p_{\min} = 1/n^2$, $\geq$ rule (c) $p_{\min} = 1/n$, $\geq$ rule

Fig. 2: Median number of generations and the corresponding interquartile ranges needed to locate the optimum of the NEUTRALITY problem

Plateaus. Plateau is an extension of the W-model suggested in [16]. This transformation operates on the function values, by setting $\text{PLATEAU}(f(x)) := \lfloor f(x)/k \rfloor + 1$, for a parameter k that determines the size of the plateau. We superpose this transformation to ONEMAX, and study performances for dimension $n = 1000$.

Small plateaus, $k = 2$. For $k = 2$, $p_{\min} = 1/n^2$, and $2 \leq \lambda \leq 2^6$, all considered parameter control algorithms improve the performance of the $(1+\lambda)$ EA. For large values of λ (starting from $\lambda = 2^7$), however, the runtimes of the $(1 + \lambda)$ EA and the parameter control algorithms are hardly distinguishable. The only exception for large λ is the proposed $(1 + \lambda)$ HQEA, which performs a bit better than the $(1 + \lambda)$ EA. The Wilcoxon test suggests that the difference is significant with the p-values less than $3.9 \cdot 10^{-18}$.

The results obtained when using $p_{\min} = 1/n$ are less successful, as most of the parameter control methods just perform on par with the $(1 + \lambda)$ EA in this case. The $(1 + \lambda)$ HQEA shows nevertheless a stable and comparatively good performance for all offspring population sizes λ. The $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ performs worse than the $(1 + \lambda)$ EA in this case.

Plateaus with $k = 3$. We also considered a harder version of the problem with a larger size of the plateau, for which we use $k = 3$. As the total running time for this problem is much larger than for $k = 2$, we had to restrict our experiments to a smaller problem size $n = 100$.

For $p_{\min} = 1/n^2$ we cannot see any clear improvement of parameter control over the $(1 + \lambda)$ EA any more. Moreover, for $\lambda \geq 2^7$, the $(1 + \lambda)$ EA seems to be the best performing algorithm.

Interestingly, for the $1/n$ lower bound the situation is pretty similar to the $k = 2$ case. All the parameter control algorithms perform on par with the $(1+\lambda)$ EA (with only slight differences at $\lambda = 2^4, 2^6$), except for the $(1 + \lambda)$ $\text{EA}_{r/2,2r}$, which performs worse. It seems that as the problem gets harder, a larger lower bound is preferable, which seems to be natural, as with a bigger plateau, a higher mutation rate is needed to leave it. Let us also mention that the $(1 + \lambda)$ HQEA performs stably well for all considered values of λ in this preferable configuration.

Ruggedness. We also considered the W-Model extension F9 from [16], which adds local optima to the fitness landscape by mapping the fitness values to $r_2(f(x)) := f(x) + 1$ if $f(x) \equiv n \mod 2$ and $f(x) < n$, $r_2(f(x)) := \max\{f(x) - 1, 0\}$ for $f(x) \equiv n + 1 \mod 2$ and $f(x) < n$, and $r_2(n) := n$. This transformation is superposed on ONEMAX of size $n = 100$.

For $p_{\min} = 1/n^2$, all the considered parameter control algorithms significantly worsen the performance of the $(1+\lambda)$ EA. Even the $(1+\lambda)$ $\text{EA}_{r/2,2r}$, which, untypically, performs the best among all these algorithms, is still significantly worse than the $(1 + \lambda)$ EA.

The situation improves for $p_{\min} = 1/n$ and the parameter control algorithms show similar performance as the $(1 + \lambda)$ EA. The only exception is again $(1 + \lambda)$ $\text{EA}_{r/2,2r}$, whose performance did not change much compared to the case $p_{\min} = 1/n^2$.

5 Conclusions and Future Work

To address the issue of unstable performance of several parameter control algorithms on different values of population size reported in [36], we proposed the Q-learning based parameter control algorithm, the $(1 + \lambda)$ QEA, and its hybridization with the $(1 + \lambda)$ $\text{EA}(A, b)$, the $(1 + \lambda)$ HQEA. The algorithms were compared empirically on OneMax and five more benchmark problems with different characteristics, such as neutrality, plateaus and presence of local optima. Our main findings may be summarized as follows.

On simple problems, i.e. OneMax, LeadingOnes, and Plateau with $k = 2$ the $(1 + \lambda)$ HQEA is the only algorithm which always performs on par or better than the other tested algorithms for all the considered values of λ and both mutation rate lower bounds.

On the harder problems, i.e., Neutrality, Plateau with $k = 3$, and Ruggedness, the $(1 + \lambda)$ HQEA performance depends on the lower bound (the same is true for the other algorithms). For $p_{\min} = 1/n$, the $(1 + \lambda)$ HQEA still performs on par with or better than the other algorithms for all values of λ in almost all cases.

The $(1 + \lambda)$ QEA is usually worse than the $(1 + \lambda)$ HQEA. There are a number of examples where $(1+\lambda)$ $\text{EA}(A, b)$ is significantly worse as well. The hybridization of these two algorithms seems to be essential for the observed good performance of the $(1 + \lambda)$ HQEA.

As next steps, we plan on investigating *more possible actions* for the Q-learning part. For example, one may use several different multiplicative update rules, to allow for a faster adaptation when the current rate is far from optimal. This might in particular be relevant in *dynamic environments*, in which the fitness functions (and with it the optimal parameter values) change over time. We also plan on identifying ways to automatically select the configuration of the Q-learning algorithms, with respect to its hyper-parameters, but also with respect to whether to use the strict or the non-strict update rule. In this context, we are investigating exploratory landscape analysis [32, 40].

Acknowledgments. The reported study was funded by RFBR and CNRS, project number 20-51-15009, by the Paris Ile-de-France Region, and by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH.

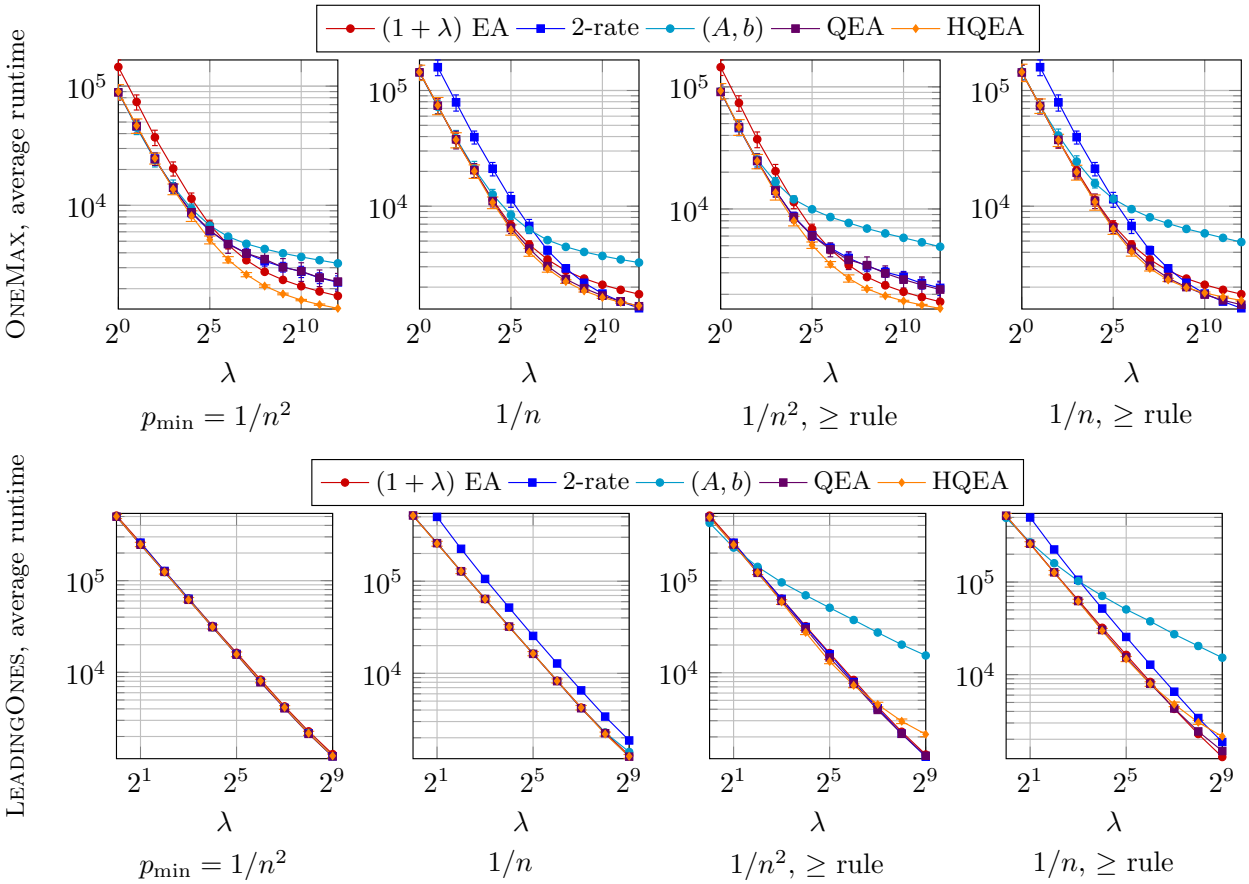
References

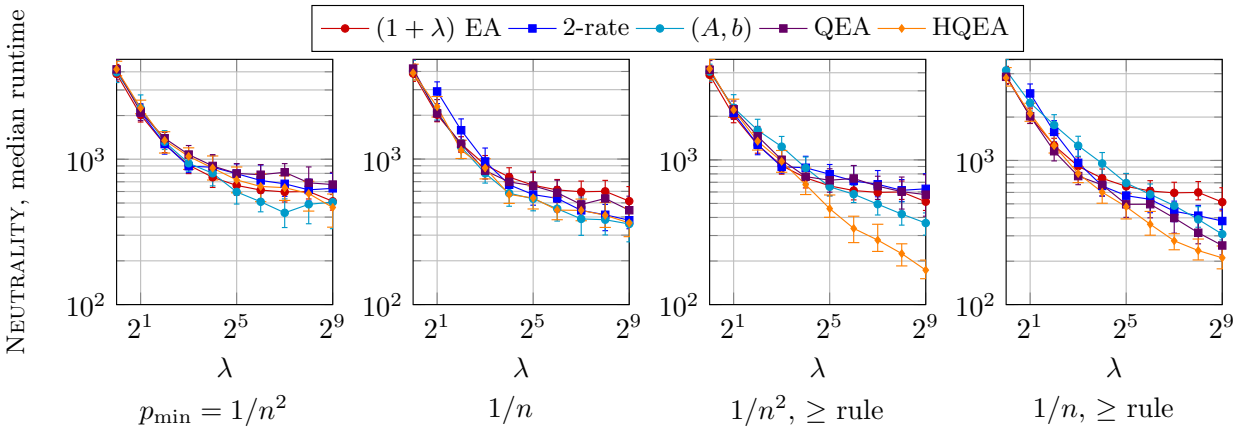
1. Aleti, A., Moser, I.: Entropy-based adaptive range parameter control for evolutionary algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'13). pp. 1501–1508 (2013)
2. Bäck, T.: The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In: Proc. of Parallel Problem Solving from Nature (PPSN'92). pp. 87–96. Elsevier (1992)
3. Bartz-Beielstein, T., Flasch, O., Koch, P., Konen, W.: SPOT: A toolbox for interactive and automatic tuning in the R environment. In: Proc. of the 20th Workshop on Computational Intelligence. pp. 264–273. Universitätsverlag Karlsruhe (2010)
4. Belkhir, N., Dréo, J., Savéant, P., Schoenauer, M.: Per instance algorithm configuration of CMA-ES with limited budget. In: Proc. of Genetic and Evolutionary Conference (GECCO'17). pp. 681–688. ACM (2017)
5. Carvalho Pinto, E., Doerr, C.: Towards a more practice-aware runtime analysis of evolutionary algorithms (2018), https://arxiv.org/abs/1812.00493
6. Costa, L.D., Fialho, Á., Schoenauer, M., Sebag, M.: Adaptive operator selection with dynamic multi-armed bandits. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'08). pp. 913–920. ACM (2008)
7. Derrac, J., Garcia, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation **1**(1), 3–18 (2011)
8. Devroye, L.: The compound random search. Ph.D. dissertation, Purdue Univ., West Lafayette, IN (1972)

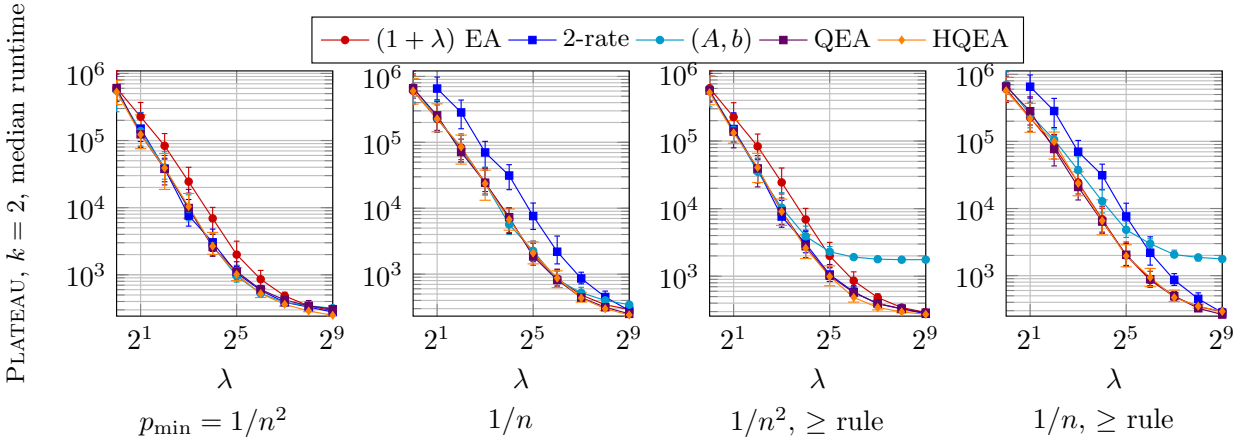
9. Doerr, B., Doerr, C.: Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In: Theory of Evolutionary Computation: Recent Developments in Discrete Optimization, pp. 271–321. Springer (2020), also available online at https://arxiv.org/abs/1804.05650
10. Doerr, B.: Analyzing randomized search heuristics via stochastic domination. Theoretical Computer Science **773**, 115–137 (2019)
11. Doerr, B., Doerr, C., Lengler, J.: Self-adjusting mutation rates with provably optimal success rules. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'19). ACM (2019)
12. Doerr, B., Doerr, C., Yang, J.: k-bit mutation with self-adjusting k outperforms standard bit mutation. In: Proc. of Parallel Problem Solving from Nature (PPSN'16). Lecture Notes in Computer Science, vol. 9921, pp. 824–834. Springer (2016)
13. Doerr, B., Doerr, C., Yang, J.: Optimal parameter choices via precise black-box analysis. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'16). pp. 1123–1130. ACM (2016)
14. Doerr, B., Gießen, C., Witt, C., Yang, J.: The $(1 + \lambda)$evolutionary algorithm with self-adjusting mutation rate. Algorithmica (2019)
15. Doerr, C., Wagner, M.: On the effectiveness of simple success-based parameter selection mechanisms for two classical discrete black-box optimization benchmark problems. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'18). pp. 943–950. ACM (2018)
16. Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O.M., Bäck, T.: Benchmarking discrete optimization heuristics with IOHprofiler. Applied Soft Computing **88**, 106027 (2020)
17. Doerr, C., Ye, F., van Rijn, S., Wang, H., Bäck, T.: Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: profiling $(1 + \lambda)$ EA variants on onemax and leadingones. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'18). pp. 951–958. ACM (2018)
18. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science **276**, 51–81 (2002)
19. Eiben, A.E., Horvath, M., Kowalczyk, W., Schut, M.C.: Reinforcement learning for online control of evolutionary algorithms. In: Proc. of the 4th International Conference on Engineering Self-Organising Systems. pp. 151–160 (2006)
20. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation **3**, 124–141 (1999)
21. Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and efficient hyperparameter optimization at scale. In: Proc. of International Conference on Machine Learning (ICML'18). pp. 1436–1445 (2018)
22. Fialho, Á., Costa, L.D., Schoenauer, M., Sebag, M.: Analyzing bandit-based adaptive operator selection mechanisms. Annals of Mathematics and Artificial Intelligence **60**, 25–64 (2010)
23. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. of Learning and Intelligent Optimization (LION'11). pp. 507–523. Springer (2011)
24. Karafotias, G., Eiben, Á.E., Hoogendoorn, M.: Generic parameter control with reinforcement learning. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'14). pp. 1319–1326 (2014)
25. Karafotias, G., Hoogendoorn, M., Eiben, A.E.: Evaluating reward definitions for parameter control. In: Proc. of International Conference on the Applications of Evolutionary Computation. Lecture Notes in Computer Science, vol. 9028, pp. 667–680 (2015)
26. Karafotias, G., Hoogendoorn, M., Eiben, A.: Parameter control in evolutionary algorithms: Trends and challenges. IEEE Transactions on Evolutionary Computation **19**, 167–187 (2015)
27. Kern, S., Müller, S.D., Hansen, N., Büche, D., Ocenasek, J., Koumoutsakos, P.: Learning probability distributions in continuous evolutionary algorithms - a comparative review. Natural Computing **3**, 77–112 (2004)
28. Lässig, J., Sudholt, D.: Adaptive population models for offspring populations and parallel evolutionary algorithms. In: Proc. of Foundations of Genetic Algorithms (FOGA'11). pp. 181–192. ACM (2011)
29. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. arXiv preprint arXiv:1603.06560 (2016)
30. Lobo, F.G., Lima, C.F., Michalewicz, Z. (eds.): Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence, vol. 54. Springer (2007)
31. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Stützle, T., Birattari, M.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
32. Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory landscape analysis. In: Proc. of Genetic and Evolutionary Conference (GECCO'11). pp. 829–836. ACM (2011)
33. Müller, S.D., Schraudolph, N.N., Koumoutsakos, P.D.: Step size adaptation in evolution strategies using reinforcement learning. In: Proc. of the 2002 Congress on Evolutionary Computation (CEC'02). pp. 151–156 (2002)

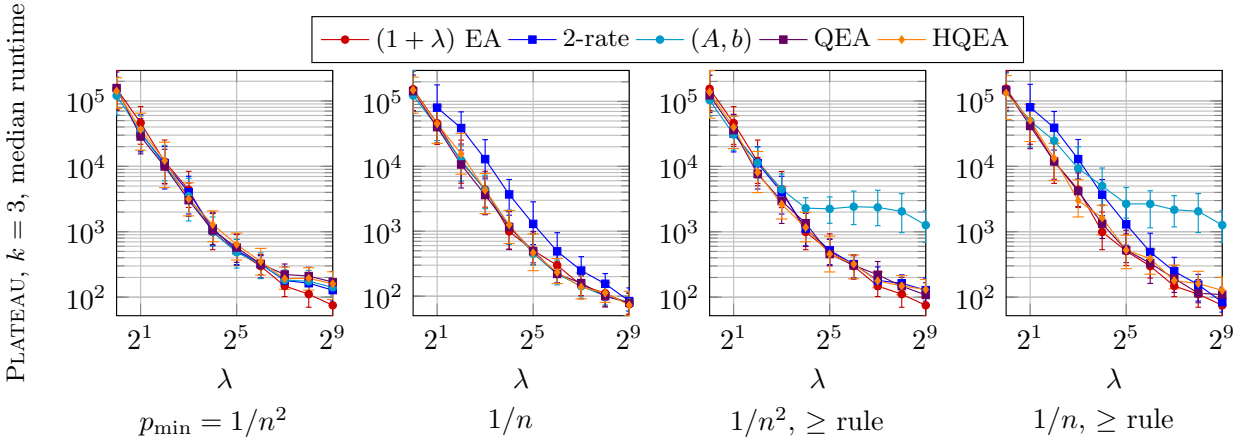
34. Pushak, Y., Hoos, H.H.: Algorithm configuration landscapes: - more benign than expected? In: Proc. of Parallel Problem Solving from Nature (PPSN'18). Lecture Notes in Computer Science, vol. 11102, pp. 271–283. Springer (2018)
35. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Fromman-Holzboorg Verlag, Stuttgart (1973)
36. Rodionova, A., Antonov, K., Buzdalova, A., Doerr, C.: Offspring population size matters when comparing evolutionary algorithms with self-adjusting mutation rates. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'19). pp. 855–863. ACM (2019)
37. Rost, A., Petrova, I., Buzdalova, A.: Adaptive parameter selection in evolutionary algorithms by reinforcement learning with dynamic discretization of parameter range. In: Proc. of Genetic and Evolutionary Computation Conference Companion (GECCO'16). pp. 141–142 (2016)
38. Schumer, M.A., Steiglitz, K.: Adaptive step size random search. IEEE Transactions on Automatic Control **13**, 270–276 (1968)
39. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press, Cambridge, MA, USA (1998)
40. Vérel, S.: Apport à l'analyse des paysages de fitness pour l'optimisation mono-objective et multiobjective : Science des systèmes complexes pour l'optimisation par méthodes stochastiques. (Contributions to fitness landscapes analysis for single- and multi-objective optimization : Science of complex systems for optimization with stochastic methods) (2016), https://tel.archives-ouvertes.fr/tel-01425127
41. Wang, H., Emmerich, M., Bäck, T.: Cooling strategies for the moment-generating function in bayesian global optimization. In: Proc. of Congress on Evolutionary Computation (CEC'18). pp. 1–8 (2018)
42. Weise, T., Wu, Z.: Difficult features of combinatorial optimization problems and the tunable w-model benchmark problem for simulating them. In: Proc. of Genetic and Evolutionary Computation Conference Companion (GECCO'18). pp. 1769–1776 (2018)

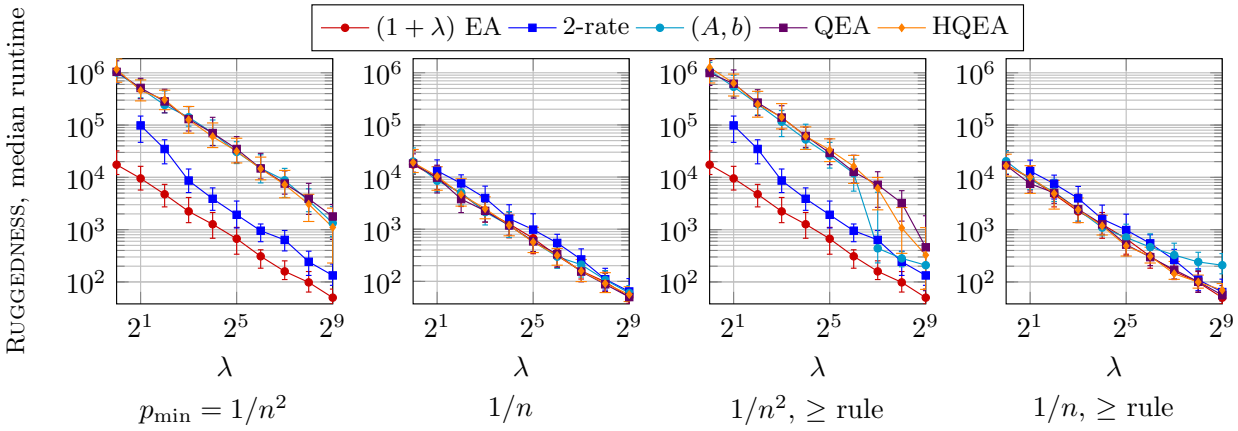
Appendix




(1 + λ) EA
2-rate
(A, b)
QEA
HQEA
Neutrality, median runtime
10^3
10^2
2^1
2^5
2^9
λ
p_min = 1/n^2
1/n
1/n^2, ≥ rule
1/n, ≥ rule


(1 + λ) EA
2-rate
(A, b)
QEA
HQEA
Plateau, k = 2, median runtime
10^6
10^5
10^4
10^3
2^1
2^5
2^9
λ
p_min = 1/n^2
1/n
1/n^2, ≥ rule
1/n, ≥ rule


(1 + λ) EA
2-rate
(A, b)
QEA
HQEA
Plateau, k = 3, median runtime
10^5
10^4
10^3
10^2
2^1
2^5
2^9
λ
p_min = 1/n^2
1/n
1/n^2, ≥ rule
1/n, ≥ rule


(1 + λ) EA
2-rate
(A, b)
QEA
HQEA
RUGGEDNESS, median runtime
10^6
10^5
10^4
10^3
10^2
2^1
2^5
2^9
λ
p_min = 1/n^2
1/n
1/n^2, ≥ rule
1/n, ≥ rule