



Mutation Rate Control in the $(1 + \lambda)$ Evolutionary Algorithm with a Self-adjusting Lower Bound

Kirill Antonov¹, Arina Buzdalova^{1(✉)}, and Carola Doerr²

¹ ITMO University, Saint Petersburg, Russia
abuzdalova@gmail.com

² Sorbonne Université, CNRS, LIP6, Paris, France

Abstract. We consider the 2-rate $(1 + \lambda)$ Evolutionary Algorithm, a heuristic that evaluates λ search points per each iteration and keeps in the memory only a best-so-far solution. The algorithm uses a dynamic probability distribution from which the radius at which the λ “offspring” are sampled. It has previously been observed that the performance of the 2-rate $(1 + \lambda)$ Evolutionary Algorithm crucially depends on the threshold at which the mutation rate is capped to prevent it from converging to zero. This effect is an issue already when focusing on the simple-structured OneMax problem, the problem of minimizing the Hamming distance to an unknown bit string. Here, a small lower bound is preferable when λ is small, whereas a larger lower bound is better for large λ .

We introduce a secondary parameter control scheme, which adjusts the lower bound during the run. We demonstrate, by extensive experimental means, that our algorithm performs decently on all OneMax problems, independently of the offspring population size. It therefore appropriately removes the dependency on the lower bound. We also evaluate our algorithm on several other benchmark problems, and show that it works fine provided the number of offspring, λ , is not too large.

Keywords: Parameter setting · Evolutionary computation · Metaheuristic · Algorithm configuration · Mutation rate

1 Introduction

Evolutionary algorithms (EAs) are a class of iterative optimization heuristics that are aimed to produce high-quality solutions for complex problems in a reasonably short amount of time [5, 9, 16]. They are applied to solve optimization problems in various areas, such as industrial design and scheduling [2], search-based software engineering [12], bioinformatical problems (for example, protein folding or drug design) [11], and many more.

The reported study was funded by RFBR and CNRS, project number 20-51-15009.

EAs perform black-box optimization, i.e. they learn the information about the problem instance by querying the *fitness function* value of a solution candidate (also referred to as *individual* within the evolutionary computation community). The structure of an EA is inspired by the ideas of natural evolution: *mutation* operators make changes to individuals, *crossover* operators create offspring from parts of the existing individuals (*parents*), while a *selection* operator decides which individuals to keep in the memory (*population*) for the next iteration (*generation*).

The performance of an EA depends on the setting of its parameters. Existing works on choosing these parameters can be classified into two main categories: **Parameter tuning** aims at fitting the parameter values to the concrete problem at hand, typically through an iterated process to solve this meta-optimization problem. While being very successful in practice [13], parameter tuning does not address an important aspect of the parameter setting problem: for many optimization problems, the optimal parameter settings are not static but change during the optimization stages [3, 10, 14]. **Parameter control** addresses this observation by entailing methods that adjust the parameter values during the optimization process, so that they not only aim at *identifying* well-performing settings, but to also *track* these while they change during the run [6].

Our work falls in the latter category. More precisely, in this work we address a previously observed shortcoming of an otherwise successful parameter control mechanism, and suggest ways to overcome this shortcoming. We then perform a thorough experimental analysis in which we compare our algorithm with previously studied ones.

We continue previous work summarized in [18], where we have studied the so-called 2-rate $(1 + \lambda)$ EA $_{r/2, 2r}$ algorithm suggested in [4]. The 2-rate $(1 + \lambda)$ EA $_{r/2, 2r}$ is an EA of $(1 + \lambda)$ EA type with self-adjusting mutation rates. Intuitively, the mutation rate is the expected value of the search radius at which the next solution candidates are sampled. More precisely, this radius is sampled from a binomial distribution $\text{Bin}(n, p)$, where n is the problem dimension and p the mutation rate. After sampling the search radius from this distribution, the “mutation” operator samples the next solution candidate uniformly at random among all points at this distance around the selected parent (which, in the context of our work, is always a best-so-far solution).

It has been proven in [4] that the 2-rate $(1 + \lambda)$ EA $_{r/2, 2r}$ achieves the asymptotically best possible expected running time on the ONEMAX problem, the problem of minimizing the Hamming distance to an unknown bit string $z \in \{0, 1\}^n$. In evolutionary computation, as in other black-box settings, the running time is measured by number of function evaluations performed before evaluating for the first time an optimal solution.

The main idea of the 2-rate $(1 + \lambda)$ EA $_{r/2, 2r}$ algorithm is to divide the offspring population in two equal subgroups and to create the offspring in each subgroup with a group-specific mutation rate. The offspring in the first group are generated by mutating each bit of the parent individual with probability $r/2$, whereas a mutation rate of $2r$ is applied to the second group. At the end of

each iteration, the mutation rate r is updated as follows: with probability $1/2$, it is updated to the rate used in the subpopulation that contains the best among all offspring (ties broken uniformly at random), and with probability $1/2$ the mutation rate is updated to $r/2$ or $2r$ uniformly at random.

When running the 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$, it is convenient to define a lower bound \mathbf{lb} for the mutation rate, to prevent it from going to zero, in which case the algorithm would get stuck. The original algorithm in [4] uses as lower bound $\mathbf{lb} = 1/n$. In our previous work [18], however, we observed that a more generous lower bound of $1/n^2$ can be advantageous. In particular, we observed the following effects on ONEMAX, for problem dimension n up to 10^5 and offspring population size λ up to $32 \cdot 10^3$:

- it is more preferable to use $1/n^2$ as a lower bound for the mutation rate when the population size is small ($5 \leq \lambda \leq \lambda_1 \in (50, 100)$), while for large population sizes ($\lambda_1 \leq \lambda \leq \lambda_2 \in (800, 3200)$) a lower bound of $1/n$ seems to work better;
- these results hold for all considered problem dimensions.

We consider the first property as a disadvantage of the 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ because its efficiency depends on the lower bound and for different values of λ different lower bounds should be used. The aim of the current paper is to propose and to test an algorithm based on the 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$, which manages to adapt the lower bound \mathbf{lb} automatically.

The main idea of the 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ improvement which we propose in this paper could be briefly described in the following way. First we start with the higher lower bound of $1/n$. Then in each population, we count the amount of individuals that were better than the parent separately for the cases of a higher mutation rate and for a lower one, which we call as the number of *votes* for a certain mutation rate. If a certain number of total votes is reached, we check if there are enough votes for the lower mutation rate among them, and decrease the value of the lower bound \mathbf{lb} in this case. Hence, the lower bound tend to become more generous closer to the end of optimization, as it never increases and has a chance to decrease.

The proposed algorithm is shown to be efficient on ONEMAX for all considered population sizes λ , which we vary from 5 to 3200. Our approach therefore solves the disadvantageous behavior of the 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ described above. We also tested our modification on other benchmark problems and observed good efficiency at least on sufficiently small population sizes, i.e. on $5 \leq \lambda \leq 20$ for LEADINGONES and on $5 \leq \lambda \leq 16$ for W-MODEL transformations of ONEMAX.

2 Description of the Proposed Algorithm

2.1 Preliminaries

Throughout the paper we consider the maximization of a problem that is expressed as a “fitness function” $f : \{0, 1\}^n \rightarrow \mathbb{R}$. That is, we study single-

objective optimization of problems with n binary decision variables. Particularly, we study maximization of the benchmark functions ONEMAX, LEADING-ONES, and some problems obtained through so-called W-MODEL transformations applied to ONEMAX. These functions will be formally introduced in Sects. 3.1, 3.2, and 3.3, respectively.

All the considered algorithms are based on the $(1 + \lambda)$ EA $_{0 \rightarrow 1}$. Just like the 2-rate $(1 + \lambda)$ EA $_{r/2, 2r}$, the algorithm keeps in its memory only one previously evaluated solution, the most recently evaluated one with best-so-far “fitness” (ties among the λ points evaluated in each iteration are broken uniformly at random). Each of the λ “offspring” is created by the *shift mutation* operator discussed in [17], which – instead of sampling the search radius from the plain binomial distribution $\text{Bin}(n, p)$ – shifts the probability mass from 0 to 1. That is, the search radius is sampled from the distribution $\text{Bin}_{0 \rightarrow 1}(n, p)$, which assigns to each integer $0 \leq k \leq n$ the value $\text{Bin}_{0 \rightarrow 1}(n, p)(k) = \text{Bin}(n, p)(k)$ for $1 < k \leq n$, but sets $\text{Bin}_{0 \rightarrow 1}(n, p)(0) = 0$ and $\text{Bin}_{0 \rightarrow 1}(n, p)(1) = \text{Bin}(n, p)(1) + \text{Bin}(n, p)(0)$. Given a search radius k , the offspring is then sampled uniformly at random among all the points at Hamming distance k from the parent (i.e., the point in the memory). For each offspring, the search radius k is sampled from $\text{Bin}_{0 \rightarrow 1}(n, p)$ independently of all other decisions that have been made so far. Note that shifting the probability mass from 0 to 1 can only improve $(1 + \lambda)$ -type algorithms, as evaluating the same solution candidate is pointless in our static and non-noisy optimization setting.

As mentioned previously, our main performance criterion is optimization time, i.e., the number of evaluations needed before the algorithm evaluates an optimal solution. In all our experiments, however, the value of λ remains fixed, so that – for a better readability of the plots – we report *parallel optimization times* instead, i.e., the number of iterations (generations) until the algorithm finds an optimal solution. Of course, the parallel optimization time is just the (classical, i.e., sequential) optimization time divided by λ .

2.2 General Description

Using the mentioned observations from the paper [18], we developed the 2-rate $(1 + \lambda)$ EA $_{r/2, 2r}$ with voting that is aimed to solve the issues of the conventional 2-rate $(1 + \lambda)$ EA $_{r/2, 2r}$. The pseudocode of the proposed algorithm is shown in Algorithm 1. Let us first introduce the notation and then explain the algorithm following its pseudocode:

- Voting - individuals from a population *vote* for the mutation rates. Only individuals which are better than the parent vote. Each voting individual votes for the mutation rate with which it was obtained;
- *v* - the number of individuals voted for decreasing of the mutation rate used on the current optimization stage;
- *cnt* - the total number of voted individuals;
- *quorum* - a constant value which is calculated as described in Sect. 2.3. It actually depends on n and λ but those are fixed during the optimization stage;

Algorithm 1: 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ with voting

```

1 Initialization: Sample  $x \in \{0, 1\}^n$  uniformly at random and evaluate  $f(x)$ ;
2  $r \leftarrow 2/n$ ;  $v \leftarrow 0$ ;  $\text{cnt} \leftarrow 0$ ;  $\text{lb} \leftarrow 1/n$ ;
3 Optimization: for  $t = 1, 2, 3, \dots$  do
4    $\text{voices}[r/2] \leftarrow 0$ ;  $\text{voices}[2r] \leftarrow 0$ ;
5   for  $i = 1, \dots, \lfloor \lambda/2 \rfloor$  do
6     Sample  $\ell^{(i)} \sim \text{Bin}_{0 \rightarrow 1}(n, r/2)$ , create  $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$ , and evaluate
        $f(y^{(i)})$ ;
7     if  $f(y^{(i)}) > f(x)$  then  $\text{voices}[r/2] \leftarrow \text{voices}[r/2] + 1$ ;
8   for  $i = \lfloor \lambda/2 \rfloor + 1, \dots, \lambda$  do
9     Sample  $\ell^{(i)} \sim \text{Bin}_{0 \rightarrow 1}(n, 2r)$ , create  $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$ , and evaluate
        $f(y^{(i)})$ ;
10    if  $f(y^{(i)}) > f(x)$  then  $\text{voices}[2r] \leftarrow \text{voices}[2r] + 1$ ;
11   $v \leftarrow v + \text{voices}[r/2]$ ;
12   $\text{cnt} \leftarrow \text{cnt} + \text{voices}[r/2] + \text{voices}[2r]$ ;
13   $x^* \leftarrow \arg \max\{f(y^{(1)}), \dots, f(y^{(\lambda)})\}$  (ties broken u.a.r.);
14  if  $f(x^*) \geq f(x)$  then  $x \leftarrow x^*$ ;
15  Perform one of the following two actions equiprobably;
16  ▶ replace  $r$  with the mutation rate that  $x^*$  has been created with;
17  ▶ replace  $r$  with either  $2r$  or  $r/2$  equiprobably;
18  if  $r < \text{lb}$  then
19    if  $\text{cnt} \geq \text{quorum}$  then
20      if  $v \geq d \cdot \text{quorum}$  then
21         $\text{lb} \leftarrow \max(k \cdot \text{lb}, \text{LB})$ ;
22       $\text{cnt} \leftarrow 0$ ;  $v \leftarrow 0$ ;
23     $r \leftarrow \text{lb}$ ;
24  if  $r > \text{UB}$  then
25     $r \leftarrow \text{UB}$ ;

```

- lb - the current lower bound of the mutation rate. In our algorithm, we interpret this bound as a parameter and adjust it during the optimization. Offspring votes are used as feedback for the adjustment;
- LB - the all-time lower bound of the mutation rate. It stays constant during all the execution time;
- UB - the all-time upper bound of the mutation rate. Analogically with LB it stays constant during all the execution time;
- voices - the map where we store voices;
- $0 < d < 1$ - the portion of votes which is needed to decrease the current lower bound lb ;
- $0 < k < 1$ - the multiplier which is used to decrease lb .

Let us explain the algorithm following the pseudocode. At the initialization stage a current solution x is generated as a random bit string of length n and

the mutation rate r is initialized with an initial value $2/n$ (we use this value following [4]). Also we initialize the variables that are used at the optimization stage.

The optimization stage works until we end up with the optimal solution or until a certain budget is exhausted. In the first cycle in line 6 we perform mutation of the first half of the current population using the mutation rate $r/2$. Also we calculate the number of individuals which are better than their parents (line 7). The next cycle in line 9 does the same for the second half of the population using the mutation rate $2r$.

After all the mutations are done we update the number of individuals v voted for decreasing the mutation rate (line 11) and the total amount `cnt` of offspring who appeared to be better than the parent (line 12). Then we pick the individual with the best fitness function among the generated offspring as a candidate to become a new parent (line 13). In line 14 we check if the candidate is eligible for that by comparing its fitness $f(x^*)$ with the fitness of the current parent $f(x)$. After that in lines 15–17 we perform the usual for 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ stage which adapts the value of the mutation rate.

Then in line 18 we check whether the adapted mutation rate r is less than the current lower bound `lb`. In this case we update r in line 23 after adapting the lower bound `lb` in lines 19–22.

The adaptation of the lower bound is the key contribution of our approach. It is adapted on the basis of offspring votes. We decrease `lb` in line 21 if there are enough voted individuals (`cnt` \geq `quorum`) and a sufficient portion of them voted for decreasing of the mutation rate ($v \geq d \cdot \text{quorum}$). Note that at the same time `lb` may not become lower than the all-time lower bound `LB`. If the required number of voted individuals (`cnt` \geq `quorum`) is reached, regardless whether `lb` was changed or not, we also always start the calculation of `cnt` and v votes from scratch in line 22. Finally, in lines 24, 25 we make sure that the mutation rate does not exceed the all-time upper bound `UB`.

Let us notice that the described approach is not restricted to be applied only in the 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ algorithm, but in principle may also be applied in other parameter control algorithms with simple update rules, such as one-fifth rule, to control lower bounds of the adapted parameters.

2.3 Selection of quorum

The `quorum` value used in Algorithm 1 is calculated as $\text{quorum}(n, \lambda) = A(n, \lambda) \cdot B(\lambda)$, where $A(n, \lambda) = (8n/9000 + 10/9)\lambda$ and $B(\lambda) = (1 + (-0.5)/(1 + (\lambda/100)^2))^2$. Below we describe how this dependency was figured out.

In general the formula for `quorum` consists of two parts: $A(n, \lambda)$ and $B(\lambda)$ which were obtained experimentally. Part $A(n, \lambda)$ is linear in n and λ , while part $B(\lambda)$ depends on $1/\lambda^4$.

In early experiments on ONEMAX only the $A(n, \lambda)$ part was used. Let us describe how we obtained it. We observed $n = 100$, $n = 1000$, $n = 10000$ and run experiments on ONEMAX with various values of λ and different values of `quorum`. For every tested pair (n, λ) we gained the `quorum` that gave the best

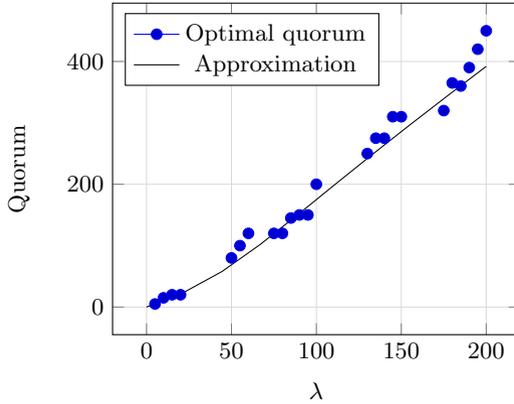


Fig. 1. Dependency of the optimal quorum on λ for $n = 1000$ (Color figure online)

runtime and plotted the dependency $\text{quorum}(\lambda)$ for a fixed n . We noticed that this dependency was close to linear. One of such dependencies is shown in Fig. 1 with blue points (which were acquired during experiments). We also plotted the dependency $\text{quorum}(n)$ for constant λ and noticed that it was close to linear as well. That gave us the target formula for $A(n, \lambda)$.

However, it turned out that $\text{quorum}(n, \lambda) = A(n, \lambda)$ is not very efficient on the LEADINGONES problem. Multiplying $A(n, \lambda)$ by $B(\lambda)$ allows the formula to work on small λ on both LEADINGONES and ONEMAX problems. At the same time, on large values of λ , $B(\lambda)$ part does not affect $A(n, \lambda)$ because $B(\lambda)$ approaches its limit which is 1. Both of these parts were chosen during the experiments but $B(\lambda)$ was chosen later on to make our method work on LEADINGONES at least for small values of λ and to not mess anything up on ONEMAX. The final approximation is shown with the black line in Fig. 1.

3 Empirical Analysis

In this section we present the results of empirical analysis of the proposed algorithm on ONEMAX, LEADINGONES, and some W-MODEL benchmark problems.

In all the experiments, the same parameter setting is used. The all-time bounds are $\text{LB} = 1/n^2$ and $\text{UB} = 1/2$. The parameters used for the lower bound adaptation are $d = k = 0.7$, they were determined in a preliminary experiment. We use $\text{quorum} = (8n/9000 + 10/9)\lambda \cdot (1 + (-0.5)/(1 + (\lambda/100)^2))^2$, as described in Sect. 2.3. All the reported results are averaged over 100 runs.

3.1 Benchmarking on ONEMAX

The classical ONEMAX problem is that of counting the number of ones in the string, i.e., $\text{OM}(x) = \sum_{i=1}^n x[i]$. The algorithms studied in our work are “unbiased” in the sense introduced in [15], i.e., their performance is invariant with

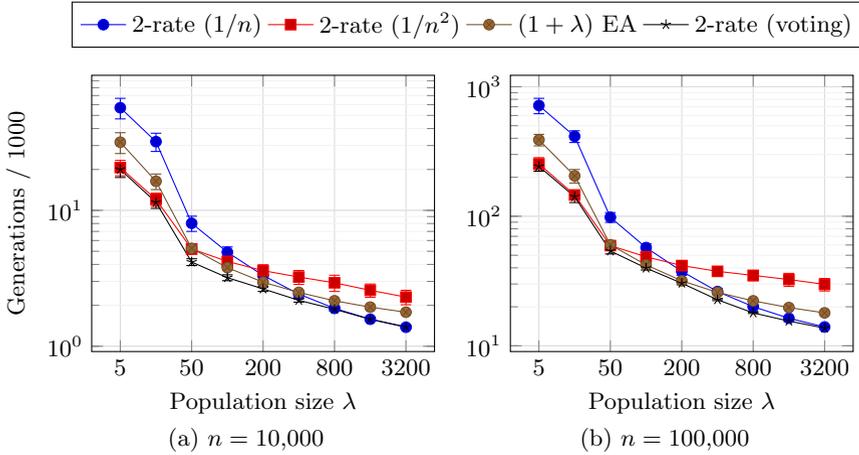


Fig. 2. Average parallel optimization time and its standard deviation for the different $(1 + \lambda)$ EA variants to find the optimal solution on ONEMAX problem. The averages are for 100 independent runs each.

respect to all Hamming automorphisms of the hypercube $\{0, 1\}^n$. From the point of view of the algorithms, the ONEMAX problem is therefore identical to that of minimizing the Hamming distance $H(\cdot, z) : \{0, 1\}^n \rightarrow \mathbb{R}, x \mapsto |\{i \mid x[i] = z[i]\}|$ – we only need to express the latter as the maximization problem $n - H(\cdot, z)$. Since the algorithms treat all these problems indistinguishably, we will in the following focus on OM only. All results, however, hold for any ONEMAX instance $n - H(\cdot, z)$.

Comparison with the Existing Methods. We compare the proposed 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ with voting, 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ with two different fixed lower bounds, and the conventional $(1 + \lambda)$ EA $_{0 \rightarrow 1}$ with no mutation rate adaptation. All the algorithms run on ONEMAX using the same population sizes as in [18]. The corresponding results are presented in Fig. 2 for $n = 10,000$ (left) and $100,000$ (right) problem sizes. The plots show average number of generations, or parallel optimization time, needed to find the optimum for each population size λ . An algorithm is more efficient if it has a lower parallel optimization time and hence if its plot is lower.

Let us first compare the previously known algorithms: the $(1 + \lambda)$ EA $_{0 \rightarrow 1}$ with a fixed mutation rate $1/n$, the 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ which adjusts mutation rate with respect to the $1/n$ lower bound and its 2-rate $(1 + \lambda)$ EA $_{r/2,2r}(1/n^2)$ version which uses a more generous $1/n^2$ lower bound. For both considered problem sizes, a similar pattern is observed:

- for $5 \leq \lambda \leq 50$ the 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ performed the best;
- then for $50 < \lambda \leq 400$ the $(1 + \lambda)$ EA $_{0 \rightarrow 1}$ became the best performing algorithm;

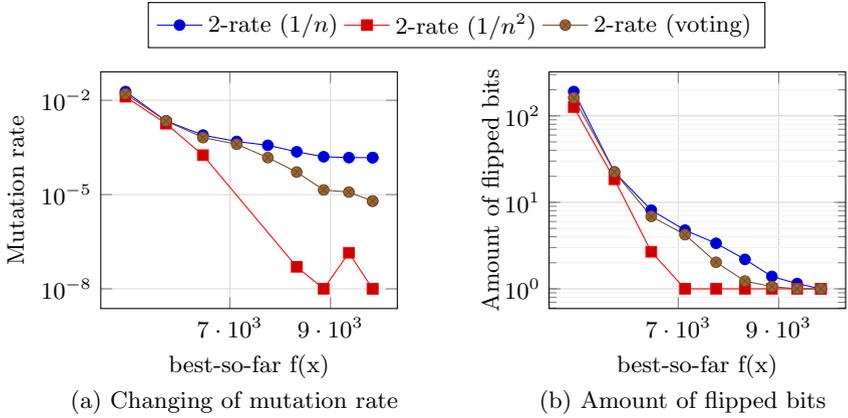


Fig. 3. Changing of mutation rate and number of flipped bits for $\lambda = 50$ and $n = 10,000$

- for $400 < \lambda \leq 3200$ the previous winner gave way to the 2-rate $(1 + \lambda)$ $EA_{r/2,2r}(1/n^2)$.

These observations confirm the results from [18], i.e. efficiency of the considered methods are strongly dependent on λ .

Let us now consider the results obtained with the proposed 2-rate $(1 + \lambda)$ $EA_{r/2,2r}$ with voting. For both problem sizes and each considered value of λ the proposed algorithm is at least as good as the previous leader observed in the corresponding area of λ values. Hence, it is quite efficient regardless the value of λ . Particularly, for $\lambda = 50, 100, 200$ and $n = 10,000$ (Fig. 2, left) the proposed algorithm is substantially better than the $(1 + \lambda)$ $EA_{0 \rightarrow 1}$, which was the previous best performing algorithm in this area. And the proposed algorithm is never worse than both 2-rate $(1 + \lambda)$ $EA_{r/2,2r}$ and 2-rate $(1 + \lambda)$ $EA_{r/2,2r}(1/n^2)$ in all the considered cases.

According to these observations, the proposed algorithm seem to be quite promising. In the next sections, we analyze its behavior in more deep on ONE-MAX and then test it on other benchmark problems.

Analysis of Bound Switching. Let us analyze the mutation rate dependency on the current best offspring to get deeper understanding of how the switching between the lower bounds really happens. Let us observe Fig. 3, where the average mutation rates chosen during the optimization process (a) and the corresponding average number of flipped bits (b) are shown.

In the beginning of optimization, the mutation rate obtained using the 2-rate $(1 + \lambda)$ $EA_{r/2,2r}$ with voting is the same as in the 2-rate $(1 + \lambda)$ $EA_{r/2,2r}$ (see Fig. 3(a)) and so the amount of flipped bits is the same approximately until the point $7 \cdot 10^3$ on the horizontal axis (Fig. 3(b)). When more and more offspring generated with rate $r/2$ are better than the parent, algorithm detects this and relaxes the lower bound to allow lower mutation rates. The switching process

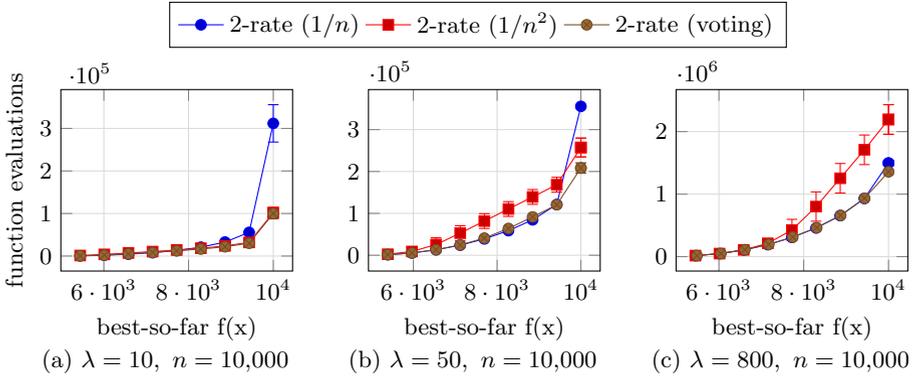


Fig. 4. Any-time performance on ONEMAX

is shown between points $7 \cdot 10^3$ and $9 \cdot 10^3$ on the horizontal axes. You can see how mutation rate is changing during this in Fig. 3(a). This leads to decrease of number of flipped bits (which is shown in Fig. 3(b)) and consequently switching to the 2-rate $(1 + \lambda) EA_{r/2,2r}(1/n^2)$. Voting algorithm finally starts to flip the same amount of bits as the 2-rate $(1 + \lambda) EA_{r/2,2r}(1/n^2)$ after the point $9 \cdot 10^3$.

Note that while mutation rates obtained with 2-rate $(1 + \lambda) EA_{r/2,2r}$ with voting and 2-rate $(1 + \lambda) EA_{r/2,2r}(1/n^2)$ in the end of optimization process are still different, the number of flipped bits tends to be the same for both algorithms, namely, one bit. This is explained by the fact that after reaching a sufficiently small mutation rate, both algorithms enter the regime of flipping exactly one randomly chosen bit according to the shift mutation operator described in Sect. 2.1. In this regime actual mutation rate values do not influence the resulting performance, so 2-rate $(1 + \lambda) EA_{r/2,2r}(1/n^2)$ does not get sufficient information to control the rate. Thus the rates are chosen randomly and the deviation of the mutation rate increases. This may explain the excess of the 2-rate $(1 + \lambda) EA_{r/2,2r}(1/n^2)$ plot in Fig. 3(a) at the end of optimization process.

Any-Time Performance Analysis. To further investigate how the optimization process goes, let us observe the number of fitness function evaluations which an algorithm needs to perform in order to reach a particular fitness value. We refer to this point of view as *any-time performance*, as shown in Fig. 4 for the 2-rate $(1 + \lambda) EA_{r/2,2r}$, 2-rate $(1 + \lambda) EA_{r/2,2r}(1/n^2)$ and 2-rate $(1 + \lambda) EA_{r/2,2r}$ with voting.

Let us first observe the Fig. 4(b) which corresponds to a medium population size $\lambda = 50$. One can see that starting at some point the 2-rate $(1 + \lambda) EA_{r/2,2r}(1/n^2)$ works better than the 2-rate $(1 + \lambda) EA_{r/2,2r}$, and the proposed 2-rate $(1 + \lambda) EA_{r/2,2r}$ with voting detects this and tries to act like the best algorithm on each segment.

The any-time performance on a small population size $\lambda = 10$ is shown in Fig. 4(a). One can see that the segment of the 2-rate $(1 + \lambda) EA_{r/2,2r}$ leadership

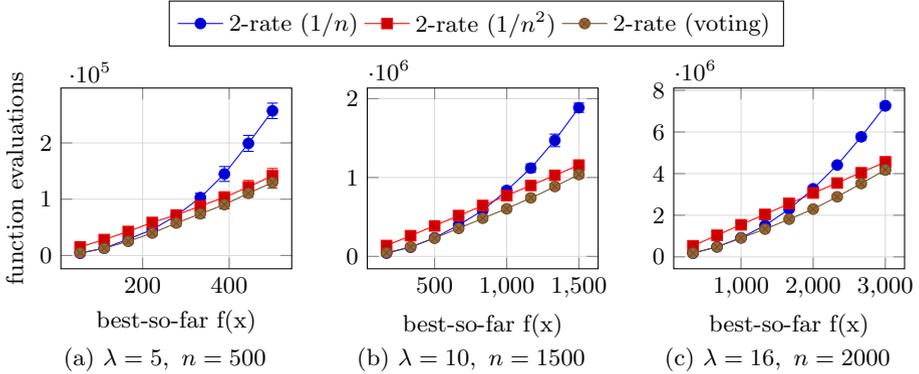


Fig. 5. Any-time performance on LEADINGONES

is quite small, hence most of the time 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ with voting keeps 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}(1/n^2)$ and so the final runtime of the proposed adaptive algorithm is close to the 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}(1/n^2)$ here.

Finally, we consider the performance on large population sizes on the example of $\lambda = 800$ in Fig. 4(c). The larger λ is the shorter becomes the segment of 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}(1/n^2)$ efficiency. Hence 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}(1/n^2)$ does not converge to the optimum faster than the 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ at any segment except probably the final mutation stage. The 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ with voting detects this and never switches to the 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}(1/n^2)$ so the runtime is close to the 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}$. To conclude, the observations of this section illustrate how 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ with voting turns out to be never worse than 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ and 2-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}(1/n^2)$ for different population sizes, as was previously seen in Fig. 2.

3.2 Benchmarking on LEADINGONES

LEADINGONES is another set of benchmark functions that is often used in theoretical analysis of evolutionary algorithms [5]. The classical LEADINGONES function assigns to each bit string $x \in \{0, 1\}^n$ the function value $\text{LO}(x) := \max \{i \mid \forall j \leq i : x[j] = 1\}$. As mentioned in Sect. 3.1, the algorithms studied in our work are invariant with respect to Hamming automorphisms. Their performance is hence identical on all functions $\text{LO}_{z,\sigma} : \{0, 1\}^n \rightarrow \mathbb{R}, x \mapsto \max \{i \mid \forall j \leq i : x[\sigma(j)] = z[\sigma(j)]\}$. This problem has also been called “the hidden permutation problem” in [1].

Unlike ONEMAX, the LEADINGONES problem is not separable; the decision variables essentially have to be optimized one by one. Most standard EAs have a quadratic running time on this problem, and the best possible query complexity is $\Theta(n \log \log n)$ [1].

We have analyzed the adaptation process with parameters $\lambda = 10$ and $n = 1500$. The results of any-time performance are shown in Fig. 5. We see that

the adaptation works nice here as well and 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ with voting achieves the best any-time performance.

In order to make sure that such positive results are invariant to λ and n we tested 2-rate $(1 + \lambda)$ EA $_{r/2,2r}$ with voting on $\lambda = 5, 10, 20$ and $n = 500, 1500, 2000$. On every pair of these parameters we gained the similar results. Example for corner cases are shown in Fig. 5(b). Nevertheless, the adaptation with the same `quorum` formula does not work for larger values of λ . Actually, it appeared that `quorum` grows much slower when optimizing LEADINGONES because in this case much less offspring vote.

To sum up, for small values of λ up to 20, the proposed adaptation works well on both ONEMAX and LEADINGONES. However, for greater values of λ the `quorum` formula has to be tuned to work well on LEADINGONES.

3.3 Benchmarking on W-MODEL Problems

Problem Description. W-MODEL problems were proposed in [19] as a possible step towards a benchmark set which is both accessible for theoretical analysis and also captures features of real-world problems. These features are:

- **Neutrality.** This feature means that different offspring may have the same fitness and be unrecognizable for EA. In our experiments, the corresponding function is implemented by excluding from the fitness calculation 10% of randomly chosen bits.
- **Epistasis.** In the corresponding problems the fitness contribution of a gene depends on other genes. In the implementation of the fitness function the offspring string is perturbed. The string is divided in the blocks of subsequent bits and special function is applied to each block to perform this perturbations. A more detailed description is given in [8].
- **Ruggedness and deceptiveness.** The problem is rugged when small changes of offspring lead to large changes of fitness. Deceptiveness means that gradient information might not show the right direction to the optimum. We used ruggedness function r_2 from [8], which maps the values of the initial fitness function $f(x)$ to $r_2(f(x)) := f(x) + 1$ if $f(x) \equiv n \pmod 2$ and $i < n$, $r_2(f(x)) := \max\{f(x) - 1, 0\}$ for $f(x) \equiv n + 1 \pmod 2$ and $i < n$, and $r_2(n) := n$.

These properties are supported in the IOHprofiler [7] and we benchmarked our algorithms with help of this tool. Details about each function implementation are described in [19]. We used IOHprofiler for our experiments, and all the parameter values are the same as described in [8]. The $F5$, $F7$ and $F9$ functions from the IOHprofiler were used, which means that the described W-MODEL transformations were applied to the ONEMAX function.

Results. We have analyzed problem dimensions $n = 500, \dots, 2000$ with step 500 and population sizes $\lambda = 5, \dots, 20$. In Figs. 6, 7 we show our results on the example of $\lambda = 10$, similar results hold for all $5 \leq \lambda \leq 16$. For $\lambda > 16$ the results of the

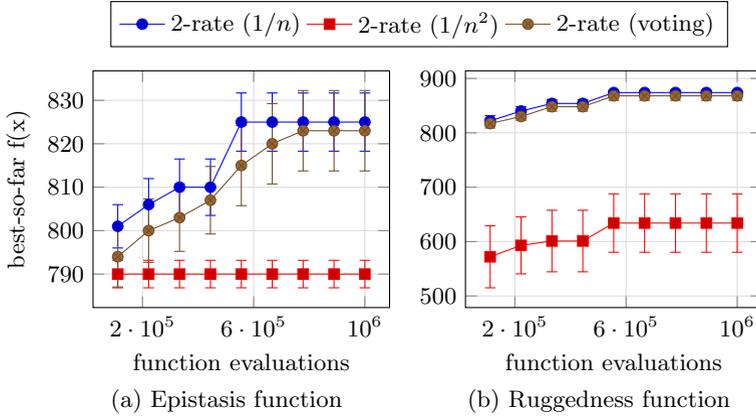


Fig. 6. Fixed budget results for the Epistasis (a) and the Ruggedness (b) functions, $\lambda = 10$, $n = 1000$

proposed algorithm get worse, we probably again need to additionally tune the quorum formula, as it was observed for LEADINGONES.

Due to the high complexity of Epistasis and Ruggedness, 2-rate $(1 + \lambda)$ $EA_{r/2,2r}$ takes too long to find the optimal solution, so we analyzed its performance from the fixed budget perspective. In the fixed budget approach, we do not pursue the goal to find the optimal solution, but we only limit the amount of fitness function evaluations and observe the best fitness values which can be reached within the specified budget. In the current work, we considered budget equal to the squared dimension size.

According to the fixed budget perspective, we had to use transposed axes compared to the previous plots in the paper, as shown for the Epistasis and Ruggedness functions in Fig. 6(a), (b) correspondingly. In this case, a higher plot corresponds to a better performing algorithm. The considered functions appeared to be too hard for 2-rate $(1 + \lambda)$ $EA_{r/2,2r}(1/n^2)$, so we did not find any segment, where 2-rate $(1 + \lambda)$ $EA_{r/2,2r}(1/n^2)$ appeared to be better than 2-rate $(1 + \lambda)$ $EA_{r/2,2r}$. Hence, for this situation results are positive when 2-rate $(1 + \lambda)$ $EA_{r/2,2r}$ with voting does not make 2-rate $(1 + \lambda)$ $EA_{r/2,2r}$ worse. And this is true for our case.

Finally, let us look at the Neutrality problem (see Fig. 7). This problem was easier to solve in a reasonable time, so we were able to use the usual stopping criterion of reaching the optimal solution, and the order of axes here is the same as in the rest of the paper. The Neutrality problem appeared to be not so hard for the 2-rate $(1 + \lambda)$ $EA_{r/2,2r}(1/n^2)$, so the 2-rate $(1 + \lambda)$ $EA_{r/2,2r}$ with voting used its advantages and worked well. Unlike for the two above problems, the promising behavior of 2-rate $(1 + \lambda)$ $EA_{r/2,2r}$ with voting was observed for $\lambda > 16$ as well, which may be explained by the fact that the considered Neutrality transformation does not change the complexity of the ONEMAX function drastically.

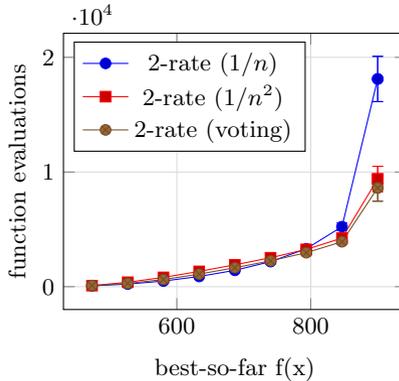


Fig. 7. Any-time performance on the Neutrality function, $\lambda = 10$, $n = 1000$

4 Conclusion

We proposed a new mutation rate control algorithm based on the 2-rate $(1 + \lambda)$ $EA_{r/2, 2r}$. It automatically adapts the lower bound of the mutation rate and demonstrates an efficient behavior on the ONEMAX problem for all considered population sizes $5 \leq \lambda \leq 3200$, while the efficiency of the initial 2-rate $(1 + \lambda)$ $EA_{r/2, 2r}$ algorithm with a fixed lower bound strongly depends on λ .

We have also applied the proposed algorithm to the LEADINGONES problem and noticed that adaptation of the lower bound worked there as well for $5 \leq \lambda \leq 20$. Finally, we tested the proposed algorithm on the W-MODEL problems with different fitness landscape features and observed that it behaves similarly to the best algorithm mostly on sufficiently small values of the population size $5 \leq \lambda \leq 16$.

Our next important goal is to further improve the efficiency of our algorithm for large population sizes λ . In the long term, we are particularly interested in efficient techniques for controlling two and more algorithm parameters. Despite some progress in recent years, the majority of works still focus on controlling a single parameter [3, 14].

References

1. Afshani, P., Agrawal, M., Doerr, B., Doerr, C., Larsen, K.G., Mehlhorn, K.: The query complexity of a permutation-based variant of Mastermind. *Discrete Appl. Math.* **260**, 28–50 (2019)
2. Bäck, T., Emmerich, M., Shir, O.M.: Evolutionary algorithms for real world applications [application notes]. *IEEE Comput. Intell. Mag.* **3**(1), 64–67 (2008)
3. Doerr, B., Doerr, C.: Theory of parameter control for discrete black-box optimization: provable performance gains through dynamic parameter choices. In: Doerr, B., Neumann, F. (eds.) *Theory of Evolutionary Computation*. NCS, pp. 271–321. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-29414-4_6. <https://arxiv.org/abs/1804.05650>

4. Doerr, B., Giessen, C., Witt, C., Yang, J.: The $(1+\lambda)$ evolutionary algorithm with self-adjusting mutation rate. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1351–1358. ACM, New York (2017)
5. Doerr, B., Neumann, F. (eds.): Theory of Evolutionary Computation: Recent Developments in Discrete Optimization. NCS. Springer, Cham (2020). <https://doi.org/10.1007/978-3-030-29414-4>
6. Doerr, C.: Non-static parameter choices in evolutionary computation. In: Proceedings of Genetic and Evolutionary Computation Conference Companion, pp. 736–761. ACM, New York (2017)
7. Doerr, C., Wang, H., Ye, F., van Rijn, S., Bäck, T.: IOHprofiler: a benchmarking and profiling tool for iterative optimization heuristics (2018). <https://arxiv.org/abs/1810.05281>. IOHprofiler. <https://github.com/IOHprofiler>
8. Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O.M., Bäck, T.: Benchmarking discrete optimization heuristics with IOHprofiler. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1798–1806. ACM, New York (2019)
9. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. NCS. Springer, Heidelberg (2015). <https://doi.org/10.1007/978-3-662-44874-8>
10. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. *IEEE Trans. Evol. Comput.* **3**, 124–141 (1999)
11. Fogel, G., Corne, D. (eds.): Evolutionary Computation in Bioinformatics. Morgan Kaufmann, San Francisco (2003)
12. Harman, M., Mansouri, A., Zhang, Y.: Search-based software engineering: trends, techniques and applications. *ACM Comput. Surv.* **45**, 1–78 (2012)
13. Hutter, F., Kotthoff, L., Vanschoren, J. (eds.): Automated Machine Learning: Methods, Systems, Challenges. SSCML. Springer, Heidelberg (2019). <https://doi.org/10.1007/978-3-030-05318-5>
14. Karafotias, G., Hoogendoorn, M., Eiben, A.: Parameter control in evolutionary algorithms: trends and challenges. *IEEE Trans. Evol. Comput.* **19**, 167–187 (2015)
15. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. *Algorithmica* **64**(4), 623–642 (2012). <https://doi.org/10.1007/s00453-012-9616-8>
16. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge (1996)
17. Pinto, E.C., Doerr, C.: Towards a more practice-aware runtime analysis of evolutionary algorithms (2018). <https://arxiv.org/abs/1812.00493>
18. Rodionova, A., Antonov, K., Buzdalova, A., Doerr, C.: Offspring population size matters when comparing evolutionary algorithms with self-adjusting mutation rates. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 855–863. ACM, New York (2019)
19. Weise, T., Wu, Z.: Difficult features of combinatorial optimization problems and the tunable w-model benchmark problem for simulating them. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1769–1776. ACM, New York (2018)