Fast Re-Optimization of LeadingOnes with Frequent Changes

Nina Bulanova, Arina Buzdalova, Carola Doerr

Fast Re-Optimization of LeadingOnes with Frequent Changes

Nina Bulanova
ITMO University
Saint Petersburg, Russia
ninokfox@gmail.com

Arina Buzdalova
ITMO University
Saint Petersburg, Russia
abuzdalova@gmail.com

Carola Doerr
Sorbonne Université
CNRS, LIP6, Paris, France
carola.doerr@lip6.fr

***Abstract*—In real-world optimization scenarios, the problem instance that we are asked to solve may change during the optimization process, e.g., when new information becomes available or when the environmental conditions change. In such situations, one could hope to achieve reasonable performance by continuing the search from the best solution found for the original problem. Likewise, one may hope that when solving several problem instances that are similar to each other, it can be beneficial to "warm-start" the optimization process of the second instance by the best solution found for the first. However, it was shown in [Doerr et al., GECCO 2019] that even when initialized with structurally good solutions, evolutionary algorithms can have a tendency to replace these good solutions by structurally worse ones, resulting in optimization times that have no advantage over the same algorithms started from scratch. Doerr et al. also proposed a diversity mechanism to overcome this problem. Their approach balances greedy search around a best-so-far solution for the current problem with search in the neighborhood around the best-found solution for the previous instance.**

In this work, we first show that the re-optimization approach suggested by Doerr et al. reaches a limit when the problem instances are prone to more frequent changes. More precisely, we show that they get stuck on the dynamic LeadingOnes problem in which the target string changes periodically. We then propose a modification of their algorithm which interpolates between greedy search around the previous-best and the current-best solution. We empirically evaluate our smoothed re-optimization algorithm on LeadingOnes instances with various frequencies of change and with different perturbation factors and show that it outperforms both a fully restarted (1+1) Evolutionary Algorithm and the re-optimization approach by Doerr et al.

I. Introduction

It is not uncommon in real-world optimization tasks that one needs to solve several problem instances of very similar type, e.g., when a routine task such as postal deliveries needs to be planned and the problem instance changes only marginally from one day to the next. Similarly, it may happen that the problem instance that we solve changes during the optimization process, e.g., because new information becomes available or some factors that determine the quality of the solution candidates change. In both situations one may hope to be able to benefit from the good solutions that were identified for the previous problem instance or from the instance before the change, respectively. Ideally, we would hope to warm-start the algorithms with the best previously found solutions, and to speed up the search compared to searching from scratch. While this hope may be true in most real-world optimization scenarios [1], it was shown in [2] that evolutionary algorithms may not always be able to benefit from such warm-starting procedure, even when the solution(s) that are transferred from one instance to the next are structurally very good. An extreme case for such inefficient use of information that was highlighted in [2] is the LeadingOnes problem: it was proven that even when initialized at Hamming distance one from the optimum, the number of fitness evaluations that the (1+1) Evolutionary Algorithm (EA) needs to sample the optimum for the first time can be quadratic in the problem dimension. Even worse, it was shown that this happens with constant probability when the fitness value of the solution used for warm-starting is at most $n/2$. Motivated by this inefficient behavior of the $(1+1)$ EA on dynamic LeadingOnes, Doerr et al. introduced a re-optimization algorithm (REA) that balances the search between sampling around the current-best solution (as done in the $(1+1)$ EA) and around the previous-best solution. The key idea of the REA is that not only the previous-best solution is kept in the memory, but also one point per each Hamming distance around the previous-best solution. This way, the algorithm does not suffer any more that much if the greedy search takes it on an unfavorable path away from the optimum – the search around the previous-best solutions helps to overcome such a distraction, by re-centering the search in the interesting part of the search domain.

Our Results. The theoretical analyses in [2] focus on re-optimization time only, i.e., on the average number of evaluations needed until a solution is found that is at least as good as the best one before the change happened. In this work, we consider a different setup, in which the fitness function is subject to frequent changes. More precisely, we (empirically) investigate how well the $(1+1)$ EA and the REA optimize the LeadingOnes problem when the optimum is changed periodically, every τ iterations, by flipping k uniformly chosen bits in the target string. We observe that the REA is indeed more efficient than the $(1+1)$ EA in the first iterations after such a fitness function perturbation. However, the advantage of the REA decreases and eventually vanishes as the number of iterations increases. At some point, the $(1+1)$ EA outperforms the REA. Based on these observations, we propose in this work a modified REA that dynamically changes the probability to sample either in the neighborhood of the previous-best

solution or around the current best solution. More precisely, the probability that our smoothREA selects the current-best solution as center of search slowly increases by an additive $1/(sn^2)$ term in each iteration. Here, the parameter s is a "smoothness" parameter that determines the speed by which the smoothREA converges towards the $(1+1)$ EA. We show that the smoothREA is more efficient than both the $(1+1)$ EA and the REA for broad ranges of s.

Related Work. Dynamic evolutionary computation is a well-studied problem. The surveys [1] and [3] provide a large number of relevant pointers to empirical and theoretical results, respectively, so that we mention here only a few works that have appeared since then and that are most closely related to our work.

Lengler et al. perform empirical [4] and theoretical [5] analysis of the dynamic BinVal problem. Just like LEADINGONES, BinVal is a classic, well-studied function in the runtime analysis domain. It is a linear function with weights of the form 2^{n-i}, $i = 1, ..., n$. In the dynamic version analyzed in the two mentioned works, the assignments of these weights to the bit positions are chosen uniformly at random in each iteration. Lengler et al. showed that an increased population size can be beneficial, in the sense that it increases the range of possible mutation rates for which the $(\mu + 1)$ EA can efficiently solve the dynamic BinVal problem. It was also shown that crossover can extend the range of efficient parameter values further. Two main differences to our work exist: (1) the optimum remains the same in the dynamic BinVal function, whereas it is changed by k bits with every change in our dynamic LEADINGONES problem, and (2) we investigate different frequencies of change, whereas Lengler et al. change the fitness function after each generation.

Guidelines to set mutation rates and population size of *non-elitist* evolutionary algorithms for the dynamic BinVal problem are provided in [6]. Extending previous work on a hand-crafted "moving Hamming-balls" function [7] the work also shows the benefit of non-trivial population sizes.

A dynamic version of LEADINGONES was studied in [8]. In that work, perturbations may happen in each iteration. More precisely, in each iteration (independently of all previous decisions), the optimum z is changed by (a) a 1-bit flip or (b) standard bit mutation with probability p. It is shown that if $p \leq c\ln(n)/n^2$ (in case (a)) and $p \leq c\ln(n)/n^3$ (in case (b)), then the expected time until the $(1 + 1)$ EA evaluates a then-optimal solution is at most $(\delta + o(1))n^{2+\delta c}$, where $\delta = (e - 1)/2 \approx 0.86$.

Related to the dynamic setting are also noisy optimization problems, and in particular models with a priori noise, in which the solution is perturbed with some probability prior to its evaluation. The main differences to our dynamic setting is that, in the noisy setting, the changes are independent in each iteration, whereas they remain fixed for the interval τ in the dynamic case studied here in this work. We nevertheless note that it has been shown that the $(1 + 1)$ EA is very sensitive to such a priori noise when optimizing the LEADINGONES function; see [9] for a sharp analysis and a summary of related works on the efficiency of EAs for noisy optimization.

II. PRELIMINARIES

We introduce the problem and the two algorithms that build the starting point for our work. Throughout this paper, we denote by $[a..b]$ the set of all integer values $r \in [a, b]$. For two bit strings $x, y \in \{0, 1\}^n$ we denote by $H(x, y) = |\{i \in [1..n] \mid x_i \neq y_i\}|$ the Hamming distance of x and y.

A. Dynamic LeadingOnes with Frequent Changes

The original (static) LEADINGONES problem is the problem of maximizing the function $\textsc{Lo} : \{0, 1\}^n \to \mathbb{R}, x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\}$, which simply assigns to each bit string the number of initial ones. Given a search point x with $\textsc{Lo}(x) = i$, it must hold that $x_1 = \ldots = x_i = 1$ and $x_{i+1} = 0$. The only way to improve the fitness of x is hence by creating a solution that is identical to x in the first i positions, but has the $(i + 1)$-st entry flipped (i.e., replaced by $1 - x_{i+1}$).

LEADINGONES was introduced in [10] to disprove a previous conjecture that for each unimodal function the number of function evaluations needed by the $(1 + 1)$ EA (see Section II-B) to find an optimal solution grows sub-quadratically in the problem dimension. This was formally proven in [11]. What makes LEADINGONES an interesting problem for theoretical works is that it is a non-separable function, i.e., the influence that a bit has on the overall quality of the solutions may depend on the setting of other bits.

It is well understood, and formalized in the so-called *unbiasedness* notion introduced in [12], that many algorithms, and in particular most mutation-only evolutionary algorithms, are indifferent with respect to optimizing this function or any of the $\textsc{Lo}_{z,\sigma}$ defined as follows. For a given *target string* $z \in \{0, 1\}^n$ and a given permutation σ of $[1..n]$, the LEADINGONES function $\textsc{Lo}_{z,\sigma}$ assigns to each string x the function value $\textsc{Lo}_{z,\sigma}(x) := \max\{i \in [0..n] \mid \forall j \leq i : x_{\sigma(j)} = z_{\sigma(j)}\}$, the longest prefix (in the order prescribed by σ) of x that agrees with z. It is not difficult to see that $\textsc{Lo}_{z,\sigma}$ has a unique optimum, which is the target string z. In this work, we are interested in the behavior of evolutionary algorithms when this target string changes. We do not consider changes in σ. To ease the presentation of our work, we can therefore safely assume that σ is equal to the identity. That is, all LEADINGONES functions appearing in our work are of the form $\textsc{Lo}_z : \{0, 1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = z_j\}$.

In our *dynamic variant of* LEADINGONES *with k-bit inversion and frequency of change* τ, the target string z is replaced by a new target string z_{new} after every τ steps. The Hamming distance of z and z_{new} is equal to k, i.e., we assume that exactly k bits are changed at the end of each period. We further assume that these k bits are chosen uniformly at random (u.a.r.).

Note that even a 1-bit inversion can reduce the fitness of the current-best solution very drastically: in the extreme case, it decreases from n to 0 if the first bit of the target string is flipped. This example also shows that LEADINGONES

Algorithm 1: The $(1+1)$ EA with shift mutation and mutation rate $0 < p < 1$ maximizing a function $f : \{0,1\}^n \to \mathbb{R}$.

1 **Initialization:** Sample $x \in \{0,1\}^n$ u.a.r. and evaluate $f(x)$;
2 **Optimization: for** $t = 0, 1, 2, \ldots$ **do**
3 Sample ℓ from $\mathrm{Bin}_{0\to1}(n,p)$, sample $y \leftarrow \mathrm{flip}_\ell(x)$ and evaluate $f(y)$;
4 **if** $f(y) \geq f(x)$ **then** $x \leftarrow y$;

suffers from a bad *fitness-distance correlation*, in that a small fitness value does not necessarily imply a large distance to the optimum. It is this property that may lead a greedy algorithm lose track of a good solution after a target string change. More precisely, a greedy algorithm is likely to accept solutions of better fitness but larger distance from the optimum. That this situation occurs with a non-negligible probability is at the heart of the negative re-optimization result proven in [2].

B. The (1+1) EA with Shift Mutation

The $(1+1)$ EA algorithm (Alg. 1) is initialized by sampling a search point $x \in \{0,1\}^n$ u.a.r. It then proceeds in rounds. In each iteration, one offspring y is created from x by so-called *standard bit mutation*. Standard bit mutation creates a copy from x and then changes each bit with probability p, independently of the status of all other bits. The *offspring* y replaces x if it is at least as good, i.e., if $f(y) \geq f(x)$ holds. The algorithm proceeds this way until a termination criterion is met. In our work, we run all algorithms for a certain number of iterations.

The standard setting for the *mutation rate* p is $1/n$. With this choice, one bit is changed on average, which can be easily verified by considering that the number of bits that change are binomially distributed. However, for $p = 1/n$ it happens with probability $(1-1/n)^n \approx 1/e \approx 36.8\%$ that none of the bits is flipped. We follow a suggestion made in [13] and apply the so-called *shift* operator: when none of the bits is changed, we flip a randomly selected bit. Put differently, our mutation operator first selects a mutation strength $\ell \in [1..n]$ by sampling from $\mathrm{Bin}_{0\to1}(n,p)$, which assigns probability $\mathrm{Bin}(n,p)(k) = \binom{n}{k}p^k(1-p)^{n-k}$ to each $k \in [2..n]$, and probability $\mathrm{Bin}(n,p)(0)+\mathrm{Bin}(n,p)(1) = (1-p)^n+np(1-p)^{n-1}$ to $k=1$. The offspring y is then created from x by choosing ℓ pairwise different positions $i_1, \ldots, i_\ell \in [1..n]$ and by setting $y_j = 1 - x_j$ for $j \in \{i_1, \ldots, i_\ell\}$ and by setting $y_j = x_j$ otherwise. This procedure is implemented by the flip_ℓ operator mentioned in line 3 of Alg. 1.

C. The Original REA Algorithm

It was shown in [2] that the $(1+1)$ EA can lose track of the good solutions when a k-bit inversion happens on LEADINGONES, in the sense that the average time required to regain a solution of previous fitness can be almost as long as when started from scratch, and this even when only a single bit changed. Doerr et al. therefore suggested a *Re-optimization EA (REA)*, which works as follows (see also Alg. 2).

Algorithm 2: The original REA for the re-optimization (here: maximization) of a function $f : \{0,1\}^n \to \mathbb{R}$, which emerged from the function f_{old} by a dynamic change.

1 **Input:** Solution x_{old};
2 **Initialization:** $x^0, x^* \leftarrow x_{\text{old}}$;
3 **for** $i = 1, 2, \ldots, \gamma+1$ **do** $x^i \leftarrow$ undefined, $f^i \leftarrow -\infty$;
4 **Optimization: for** $t = 0, 1, 2, \ldots$ **do**
5 Select parent x by choosing x^* with probability $1/2$ and uniformly at random from $\{x^i \mid i \in [0..\gamma+1]\} \setminus \{x^*\}$ otherwise;
6 Sample ℓ from $\mathrm{Bin}_{0\to1}(n,p)$, sample $y \leftarrow \mathrm{flip}_\ell(x)$ and evaluate $f(y)$;
7 **if** $f(y) \geq f(x^*)$ **then** $x^* \leftarrow y$;
8 $i \leftarrow \min\{H(y, x_{\text{old}}), \gamma+1\}$;
9 **if** $f(y) \geq f^i$ **then** $x^i \leftarrow y$, $f^i \leftarrow f(y)$;

The REA takes as an input the individual x_{old} with the best fitness $f_{\text{old}}(x_{\text{old}})$ achieved during the previous period before the fitness function f_{old} was changed to f. Then a special set of individuals is initialized in lines 2-3 and updated in lines 7-9. It consists of individuals which are at certain Hamming distances from x_{old}. More precisely, one individual x^i per each Hamming distance from 0 to γ is stored, and there is also one additional individual $x^{\gamma+1}$ which can be at any distance greater than γ. The parameter γ needs to be chosen by the user, and is usually an upper bound for the estimated difference between the old and the new optimum. In our work, we will assume that the number k of bit inversions is known, and set $\gamma = k$. This is the optimal choice of γ.

In each iteration, the REA selects one *parent* x from the current population, and creates one offspring y using the same shift mutation operator flip_ℓ as the $(1+1)$ EA. Two greedy selection steps follow: the current-best solution x^* is replaced by y if $f(y)$ is at least as large as $f(x^*)$. In addition, y replaces the best-so-far solution x^i, $i = \min\{H(x,y), \gamma+1\}$, if $f(y) \geq f^i$. The parent selection is done as follows (line 5): with probability $1/2$ the current-best solution x^* is chosen. Otherwise, the algorithm selects u.a.r. among all other members of the population $\{x^i \mid i \in [0..\gamma+1]\} \setminus \{x^*\}$.

Thus, in each iteration, the REA behaves like the $(1+1)$ EA with probability $1/2$ and it decides to search around a point with possibly small fitness value otherwise. It was shown in [2] that the REA needs at most $\min\{2e(\gamma+1)kn, 2en^2\}$ iterations to find again a solution of fitness at least as large as the best fitness found before the k-bit inversion, provided that $\gamma \geq k-1$ holds. Note that this result was for the REA version that uses standard bit mutation and not the shifted mutation, but it is not difficult to see that the shift mutation changes only the constants in the bound (this can be derived similarly as in [13]). We omit the details, as we are mainly interested in

the global picture.

III. Modified and Smooth REA for Frequent Changes of the Fitness Function

While the main focus in [2] is on the re-optimization time, i.e., on the time needed to find again a search point that is at least as good as the best one that was evaluated before the dynamic perturbation of the fitness function, we are interested in this work in situations in which the k-bit inversion happens with a certain frequency τ. We therefore need to extend the REA by switching off the re-optimization part when a search point of fitness at least $f_{\text{old}}(x_{\text{old}})$ is found. When this is the case, i.e., when the REA has found a solution that is at least as good as the best one found for the function before the k-bit inversion, the algorithm cannot benefit any more from the population $\{x^i \mid i \in [1..\gamma+1]\}$. In this situation, we therefore let the REA continue as a $(1+1)$ EA (line 18 of Alg. 3). For reasons of space, we do not provide the full pseudo-code of the REA adjusted to our setting, but it is the same as Alg. 3 with line 16 replaced by line 5 from Alg. 2. Despite the extension from the original REA proposed in [2], we continue to refer to this algorithm as the REA (there is no risk of confusion since we only consider this extended version in this work).

During preliminary experiments, it was repeatedly noticed that the REA has a clear advantage over the $(1+1)$ EA in optimization speed immediately after the fitness function perturbation, but that it begins to fall behind the $(1+1)$ EA after some time. This behavior is also visible in the plots presented in Section IV, e.g., Figure 1. It inspired us to gradually change the probability by which the REA resembles the $(1+1)$ EA. We use a simple linear probability adjustment: in iteration t after the last change, the probability to select the best-so-far solution x^* is set to $\min\{1, t/(sn^2)\}$, where $s > 0$ is a hyper-parameter of our modified REA. We call s the *smoothness parameter*. It determines the speed by which the modified REA converges from uniform selection among the points that are not the current-best[1] towards greedy selection of the latter. The smaller the value of s, the faster the modified REA converges to the $(1+1)$ EA. Note that we have chosen the normalization by n^2 since the optimization time of the $(1+1)$ EA on LeadingOnes is $\Omega(n^2)$ [11]. We refer to this modified REA as the smoothREA.

Algorithm 3: The smoothREA for maximizing a dynamic function that changes every τ iterations. The (extended) REA is identical to this algorithm after replacing line 16 with line 5 of Alg. 2.

1 **Input:** Smoothness parameter $s > 0$
2 **Initialization:** re-optimization flag $r \leftarrow$ false;
3 Sample $x^* \in \{0,1\}^n$ u.a.r. and evaluate $f(x^*)$
4 $x_{\text{old}} \leftarrow x^*$
5 $f_{\text{best}} \leftarrow f(x^*)$
6 **Optimization: for** $t = 0, 1, 2, \ldots$ **do**
7 **if** $t \mod \tau = 0 \wedge t \neq 0$ **then**
8 $x_{\text{old}} \leftarrow x^*$;
9 $f_{\text{best}} \leftarrow f(x_{\text{old}})$;
10 FitnessFunctionPerturbation(f)
11 k-bit inversion of target string;
12 $r \leftarrow$ true;
13 $x^0 \leftarrow x_{\text{old}}$;
14 **for** $i = 1, 2, \ldots, \gamma+1$ **do** $x^i \leftarrow$ undefined, $f^i \leftarrow -\infty$;
15 **if** r **then**
16 Select parent x by choosing x^* with probability $\min\{1, t/(sn^2)\}$ and uniformly at random from $\{x^i \mid i \in [0..\gamma+1]\} \setminus \{x^*\}$ otherwise;
17 **else**
18 $x \leftarrow x^*$;
19 Sample ℓ from $\text{Bin}_{0\to1}(n,p)$, sample $y \leftarrow \text{flip}_\ell(x)$ and evaluate $f(y)$;
20 **if** $f(y) \geq f(x^*)$ **then** $x^* \leftarrow y$;
21 **if** r **then**
22 $i \leftarrow \min\{H(y, x_{\text{old}}), \gamma+1\}$;
23 **if** $f(y) \geq f^i$ **then** $x^i \leftarrow y$, $f^i \leftarrow f(y)$;
24 **if** $f(x^*) \geq f_{best}$ **then**
25 $r \leftarrow$ false;

IV. Results

A. Experimental Setup

We compare the $(1+1)$ EA, the REA, and the smoothREA on the following dynamic LeadingOnes instances and with the following settings:

- dimension of the problem: $n \in \{100, 200\}$,
- frequency of fitness function perturbations $\tau \in \{500\} \cup \{1000i \mid i \in [1..10]\}$,
- strength of the perturbation, i.e., number of bits flipped in every fitness function perturbation: $k \in \{3, 5, 10\}$,
- size of the Hamming set: as motivated in Section II-C we use the optimal choice $\gamma = k$,
- smoothness parameter $s \in \{0.05, 0.1, 0.4, 0.8, 1.2, 1.6, 2.0\}$,
- total budget of function evaluations: 50 000 (the visible part on the plots may be less),
- results are averaged based on 100 runs for the performance plots (Figures 1 and 4) and the histograms in Figure 2. To increase the smoothness of the curves, they are based on 1000 runs for the cumulative plots shown in Figure 3.

Although we do not explicitly report additional statistical measures, such as deviation, they can be assessed using the histograms in Figure 2.

The source code used to perform this experimental study may be found at https://github.com/Ninokfox/REA.

[1] One may wonder if it would not be better to select uniformly among all points in the population, but we did not find evidence in our preliminary experiments that such a strategy would be advantageous over the one adopted in Alg. 3.

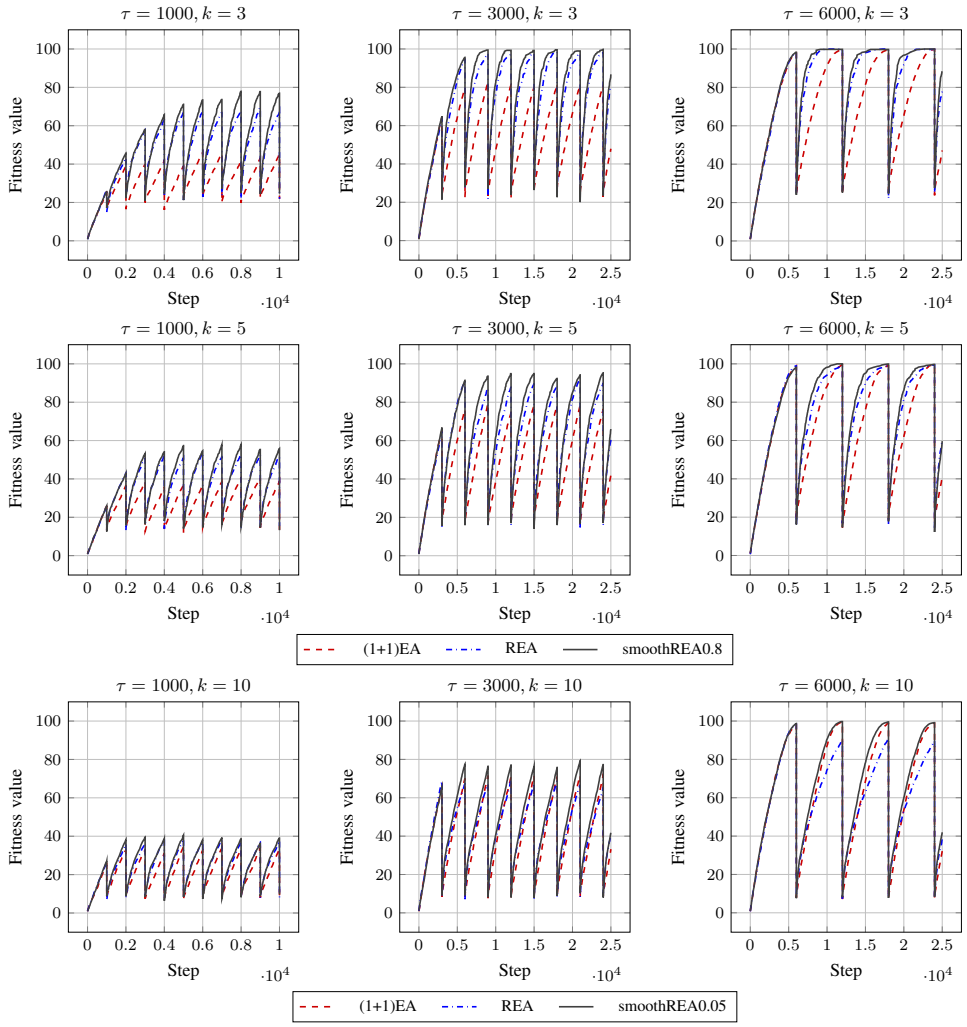

Fig. 1: Average function values of the $(1+1)$ EA, REA, and smoothREA for different instances of the dynamic LEADINGONES problem with k-bit inversion and problem size $n = 100$. Results are grouped by k in the rows (top: $k = 3$, middle: $k = 5$, bottom: $k = 10$) and for different frequencies of change in the columns. The smoothness parameter s is set to 0.8 for $k \in \{3, 5\}$ and we show results for $s = 0.05$ for $k = 10$.

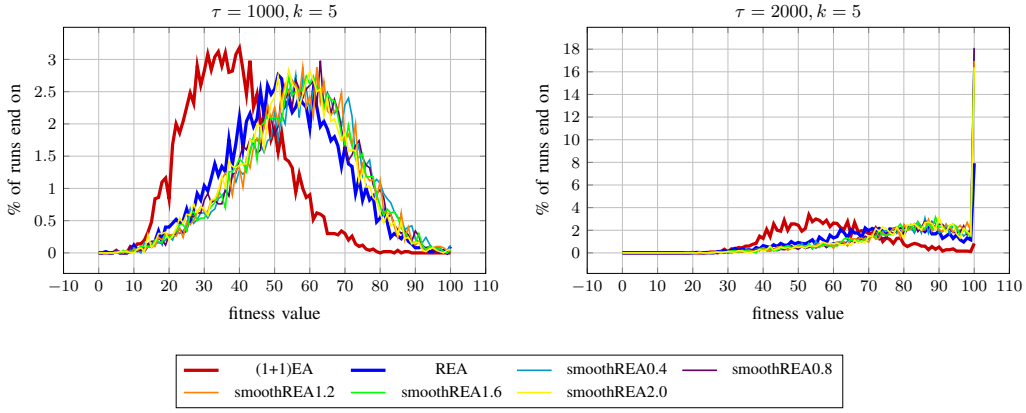

Fig. 2: Histograms of best fitness values found in each period between two fitness function perturbations, for dynamic LEADINGONES with frequency of change $\tau = 1000$ (left) and $\tau = 2000$ (right), perturbation strength $k = 5$, problem dimension $n = 100$, and 100 independent runs of the algorithms.

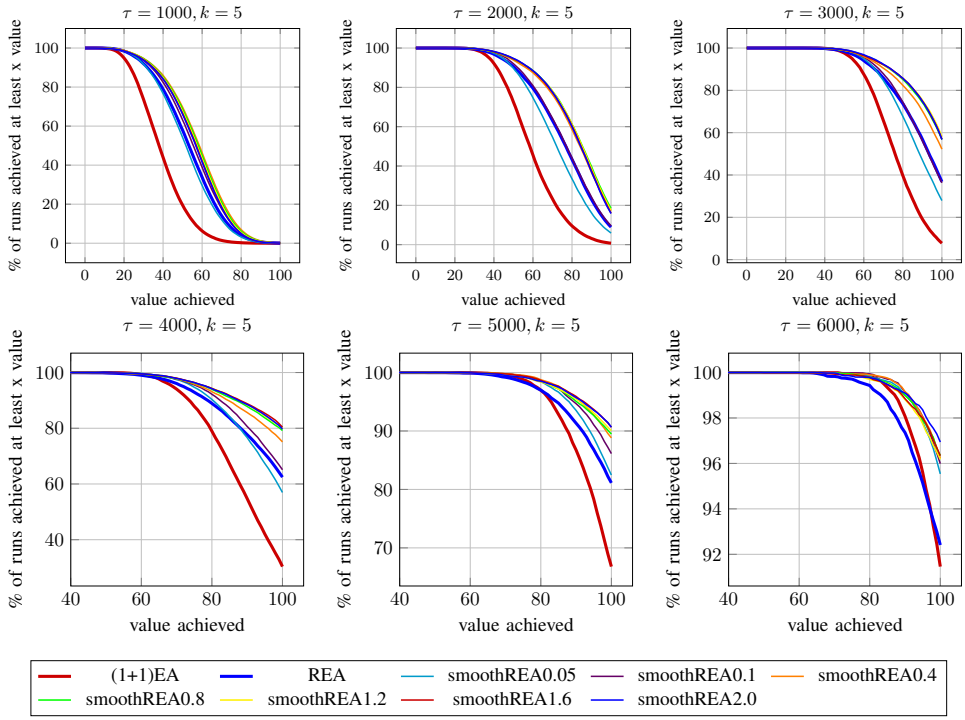

Fig. 3: Cumulative plots of the values plotted in Figure 2, for 1000 runs. Each point in the plot shows the fraction of values that is at least as large as the value indicated by the x-axis. All results are for the dynamic LEADINGONES problem with perturbation strength $k = 5$, dimension $n = 100$, and for frequencies of change ranging from $\tau = 1000$ to $\tau = 6000$.

B. Overview of Results, $n = 100$

Figure 1 shows the average quality of the best-so-far points obtained by the three algorithms in dependence of the number of solutions that have been evaluated, for different frequencies of change ($\tau \in \{1000, 3000, 6000\}$) and for different perturbation strengths ($k \in \{3, 5, 10\}$). For the smoothREA we plot here only the results for two selected values, $s = 0.8$ for $k \in \{3, 5\}$ and $s = 0.05$ for $k = 10$. These values were selected because they had decent performance. Results for other smoothness values will be discussed below.

Overall, REA performs better than the $(1+1)$ EA for most of the considered cases, but not for the setting with $k = 10$ and $\tau \geq 3000$. This is in line with the theoretical result mentioned at the end of Section II-C: the larger k, the less likely the REA is to select a good starting point, hence its reduced performance. Our proposed smoothREA outperforms the other two algorithms in all cases, but the advantage is almost negligible for $k = 10$.

We also observe that for all three algorithms the average best fitness value stagnates when the frequency of change is too large ($\tau = 1000$ for $k = 3$, $\tau \leq 3000$ for $k = 5$ and $k = 10$). The value at which the algorithms stagnate is typically lowest for the $(1 + 1)$ EA and largest for the smoothREA.

In order to trace the influence of the smoothness parameter s on the efficiency of the smoothREA, we consider the fitness values of best-found solutions at the last iteration before a perturbation of the fitness function happens. We plot these fitness values in histograms in Figure 2. More precisely, we plot the results for the $(1 + 1)$ EA, the REA, and for the smoothREA algorithms with smoothness values $s \in \{0.4, 0.8, 1.2, 1.6, 2.0\}$, for the case when $k = 5$ bits are flipped in the target string. The plot on the left shows the distribution of values for $\tau = 1000$ and the one on the right for $\tau = 2000$. Each plot is based on 100 (number of runs) times $50\,000/\tau$ fitness values, i.e, the plot on the left shows the distribution for $5\,000$ values, and the one on the right for $2\,500$ values.

For $\tau = 1000$, the distributions look like normally distributed ones for each of the seven algorithms. The mean of the values found by the $(1 + 1)$ EA are smaller than those found by the REA and the smoothREA algorithms. The difference between the original REA and the family of smoothREA algorithms is noticeable as well, but is much smaller than the difference to the $(1 + 1)$ EA.

For both frequencies of changes plotted in Figure 2 the distribution of the values look very similar for the different smoothness parameters s.

For $\tau = 2000$, the difference between the REA and the family of smoothREA algorithms is more pronounced, especially when considering the number of runs that hit the optimum (fitness equal to 100). Since this effect is not very well visible in this plot, we show a cumulative version of the plots in Figure 3, for different frequencies of change. More precisely, we plot in Figure 3 which percentage of the values (y-axis) are at least as large as the value indicated on the x-

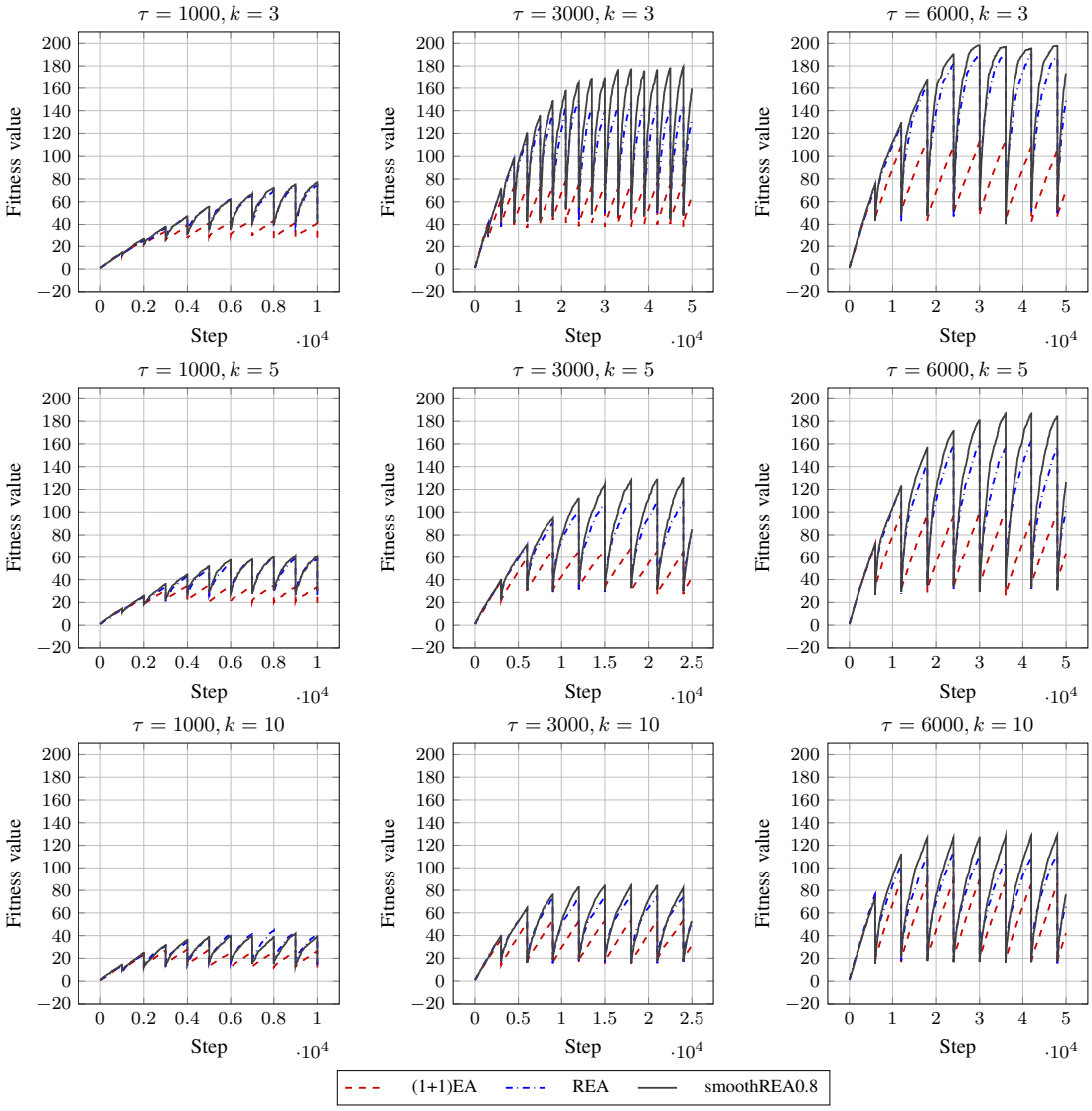

Fig. 4: Average function values of the $(1+1)$ EA, REA, and smoothREA for different instances of the dynamic LEADINGONES problem with k-bit inversion and problem size $n = 200$. Results are grouped by k in the rows (top: $k = 3$, middle: $k = 5$, bottom: $k = 10$) and for different frequencies of change in the columns. The smoothness parameter s is set to 0.8 for all settings.

axis.

When the perturbations are less frequent (i.e., when τ increases), the number of periods in which the algorithms reach the optimum increases noticeably. The $(1 + 1)$ EA (red line) is clearly worse than the REA (thick blue line) and all the smoothREA variants (thin lines), with the exception of the case $\tau = 6000$ where the REA achieves the optimum more often than the $(1 + 1)$ EA, but the $(1 + 1)$ EA has a larger fraction of periods in which it reaches fitness values up to 95.

We also observe that the REA and the smoothREA algorithms are very similar for high frequencies of change, especially for $\tau = 1000$. But the difference between them increases with increasing τ.

Overall, the visualization in Figure 3 clearly indicates that the dispersion of the fitness values reached at the end of each period is larger for the REA than for the $(1 + 1)$ EA, which explains the poor average performance plotted in Figure 1.

Comparing the different smoothREA variants, we observe that for high frequency of change $\tau = 1000$ the algorithms with $s = 0.4$ and $s = 0.8$ are the most efficient, for the frequency $\tau = 2000$ smoothness values $s = 0.8$ and 1.2 are the best choices, for $\tau = 4000$ one should pick $s = 1.6$ and 2.0 and so on. That is, the smoothness parameter s should increase with increasing periods between two fitness function perturbations. Put differently, the less frequent the change occurs, the less pronounced the benefit of the REA mechanism, and the faster one should change to the greedy selection (i.e., to the $(1 + 1)$ EA).

C. Overview of Results, $n = 200$

Figure 4 shows the average best-so-far fitness that the three algorithms achieved on different instances of the dynamic LEADINGONES problem in dimension $n = 200$. For a better comparability, the frequencies of change and perturbation strengths are identical to those plotted for the case $n = 100$ in Figure 1.

As expected, the average best fitness ever found is smaller for the 200-dimensional problem compared to the 100-dimensional one. For all nine settings, none of the three algorithms is able to reliably locate the optimal solution. Naturally, if the frequency and strength of perturbation stay the same, the percentage of changed bits relative to the length of the individual is halved in the $n = 200$ setting compared to $n = 100$. This makes the optimization for $k = 10$ and $n = 200$ similar to the optimization for $k = 5$ and $n = 100$. The same smoothness parameter turned out to be efficient in both cases. We also observe that the REA outperforms the $(1 + 1)$ EA in all cases, i.e., even for all three cases with perturbation strengths $k = 10$.

As mentioned in Section IV-A, we have tested various smoothness values. While $s = 0.8$ is among the best for most experiments, smaller values become beneficial when the frequency of change increases. For example, for $\tau = 1000$ and $k = 3$, the smoothREA with $s = 0.05$ is more efficient than the one with $s = 0.8$, but the difference in performance is not very large.

V. CONCLUSION

We have analyzed the performance of the REA proposed in [2] in the context of high frequencies of change. We first modified the REA for this setting, by switching off the REA elements when a solution is found that is at least as good as the best solution that was found for the fitness function before the last perturbation. In this case, the modified REA continues as a $(1 + 1)$ EA. We then observed that this modified REA is efficient only for the first steps after the fitness function perturbation, but falls behind the efficiency of the $(1+1)$ EA after a certain number of steps. We have therefore proposed a family of smoothREA algorithms, which interpolate the REA with the $(1 + 1)$ EA. We show that this version improves over the original REA for a broad range of smoothness parameters $s > 0$.

In contrast to the work presented in [2], our analysis is purely empirical. Even though our algorithms use the shift mutation operator suggested in [13], it should not be too difficult to compute the exact point at which one should ideally switch off the REA elements and continue with the $(1+1)$ EA. Likewise, it should not be too difficult to compute the optimal smoothness parameter s in dependence of n and k.

All algorithms investigated in this work struggle to find solutions that are better than a certain threshold that depends on n, k, and τ. There is hence room for further improvements of the re-optimization strategy.

Leaving the world of LEADINGONES instances, we plan on investigating the basic working principles of the REA on dynamic combinatorial problems such as dynamic scheduling or dynamic routing problems [14], [15].

ACKNOWLEDGMENTS

This work was supported by the Analytical Center for the Government of the Russian Federation (IGK 000000D730321P5Q0002), agreement No. 70-2021-00141. We thank Danil Shkarupin for initial explorations.

REFERENCES

[1] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 6, pp. 1–24, 2012. [Online]. Available: https://doi.org/10.1016/j.swevo.2012.05.001

[2] B. Doerr, C. Doerr, and F. Neumann, "Fast re-optimization via structural diversity," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*. ACM, 2019, pp. 233–241. [Online]. Available: https://doi.org/10.1145/3321707.3321731

[3] F. Neumann, M. Pourhassan, and V. Roostapour, "Analysis of evolutionary algorithms in dynamic and stochastic environments," in *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, B. Doerr and F. Neumann, Eds. Springer, 2020, pp. 323–357.

[4] J. Lengler and J. Meier, "Large population sizes and crossover help in dynamic environments," in *Proc. of Parallel Problem Solving from Nature (PPSN'20)*, vol. 12269. Springer, 2020, pp. 610–622. [Online]. Available: https://doi.org/10.1007/978-3-030-58112-1_42

[5] J. Lengler and S. Riedi, "Runtime analysis of the (μ + 1)-ea on the dynamic binval function," in *Proc. of Evolutionary Computation in Combinatorial Optimization (EvoCOP'21)*, ser. LNCS, vol. 12692. Springer, 2021, pp. 84–99. [Online]. Available: https://doi.org/10.1007/978-3-030-72904-2_6

[6] P. K. Lehre and X. Qin, "More precise runtime analyses of non-elitist eas in uncertain environments," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO'21)*. ACM, 2021, pp. 1160–1168. [Online]. Available: https://doi.org/10.1145/3449639.3459312

[7] D. Dang, T. Jansen, and P. K. Lehre, "Populations can be essential in tracking dynamic optima," *Algorithmica*, vol. 78, no. 2, pp. 660–680, 2017. [Online]. Available: https://doi.org/10.1007/s00453-016-0187-y

[8] R. Dang-Nhu, T. Dardinier, B. Doerr, G. Izacard, and D. Nogneng, "A new analysis method for evolutionary optimization of dynamic and noisy objective functions," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*. ACM, 2018, pp. 1467–1474. [Online]. Available: https://doi.org/10.1145/3205455.3205563

[9] D. Sudholt, "Analysing the robustness of evolutionary algorithms to noise: Refined runtime bounds and an example where noise is beneficial," *Algorithmica*, 2020, to appear. [Online]. Available: https://doi.org/10.1007/s00453-020-00671-0

[10] G. Rudolph, *Convergence Properties of Evolutionary Algorithms*. Kovac, 1997.

[11] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 51–81, 2002.

[12] P. K. Lehre and C. Witt, "Black-box search by unbiased variation," *Algorithmica*, vol. 64, no. 4, pp. 623–642, 2012.

[13] E. Carvalho Pinto and C. Doerr. (2018) Towards a more practice-aware runtime analysis of evolutionary algorithms. [Online]. Available: https://arxiv.org/abs/1812.00493

[14] M. Xu, F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with archive for dynamic flexible job shop scheduling," in *Proc. of IEEE Congress on Evolutionary Computation (CEC'21)*, 2021, pp. 2117–2124.

[15] Y. Diao, C. Li, S. Zeng, M. Mavrovouniotis, and S. Yang, "Memory-based multi-population genetic learning for dynamic shortest path problems," in *Proc. of IEEE Congress on Evolutionary Computation (CEC'19)*, 2019, pp. 2276–2283.