Curing ill-Conditionality via Representation-Agnostic Distance-Driven Perturbations

Kirill Antonov
LIACS, Leiden University
Leiden, The Netherlands
k.antonov@liacs.leidenuniv.nl

Anna V. Kononova
LIACS, Leiden University
Leiden, The Netherlands
a.kononova@liacs.leidenuniv.nl

Thomas Bäck
LIACS, Leiden University
Leiden, The Netherlands
t.h.w.baeck@liacs.leidenuniv.nl

Niki van Stein
LIACS, Leiden University
Leiden, The Netherlands
n.van.stein@liacs.leidenuniv.nl

ABSTRACT

The objective value of an ill-conditioned function may significantly change with a minor shift of the argument in the search space. Ill-conditioned functions do not have at all or exhibit very few hints towards better solutions and, thus, they are usually difficult to optimize with randomized search heuristics. However, problems that emerge in practical applications are likely to be formulated as ill-conditioned functions, as often Euclidean metric is used to measure distance in the search space. At the same time, it may be possible to use domain-specific knowledge to define such a metric in the search space so that the function stops being ill-conditioned. We consider finite search spaces and propose two mutation operators that leverage such metric to optimize the function more efficiently. The first operator assumes prior knowledge about the distance, the second operator uses the distance as a black box. Those operators apply an estimation of distribution algorithm to find the best mutant according to the defined function, which employs the given metric. For pseudo-Boolean and integer optimization problems, we experimentally show that both mutation operators speed up the search on most of the functions when applied in considered evolutionary algorithms and random local search. Moreover, those operators can be applied in any randomized search heuristic which uses perturbations. However, our mutation operators increase wall-clock time and so are helpful in practice when distance is (much) cheaper to compute than the real objective function.

ACM Reference Format:
Kirill Antonov, Anna V. Kononova, Thomas Bäck, and Niki van Stein. 2023. Curing ill-Conditionality via Representation-Agnostic Distance-Driven Perturbations. In *Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms (FOGA '23), August 30-September 1, 2023, Potsdam, Germany*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3594805.3607138

1 INTRODUCTION

Randomized search heuristics (RSH) are convenient off-the-shelf solvers of difficult optimization problems [6, 53]. While they do not necessarily guarantee to locate the exact solution to any problem, they aim to approximate a solution of sufficient quality within a feasible time frame. Applications of such algorithms require the user to define a search space X, where possible solutions are located, and the objective function f with domain X. This function defines the considered problem and usually is expensive to evaluate (in terms of computation cost, licenses or resources). In the scope of the current work, we consider finite set $X \subseteq \mathbb{Z}^n$, function f with codomain $\mathbb{R}$ and algorithms that aim to *maximize* function f, meaning finding such $x^* \in X$ that

$$\max\{f(x) \mid x \in X\} - f(x^*) = 0$$

Evolutionary algorithms (EAs) are RSH inspired by the process of natural evolution. Such algorithms are supported by theory [18, 27, 37], show promising applications to various ranges of problems including pseudo-Boolean/integer problems [19, 22, 38] and real-world scenarios [7, 24, 58]. One of the possible assumptions on the function optimized by means of EA is that the function is not ill-conditioned. This is also classically formulated as strong causality [48] of the optimization problem, contiguity, and locality [45]. It means that a small change in the candidate solution leads to a solution with similar performance according to the objective function. The significance of the change is measured according to the distance metric $d : X^2 \to \mathbb{R}$ [28], for example, Hamming distance in the case of pseudo-Boolean domains, or Euclidean distance in the case of $\mathbb{R}^n$ domains. The classical definition of an optimization landscape [51] is a triple of the search space, the metric defined for this search space, and the function being optimized. We will say that the landscape is well-conditioned if any two points in the search space with a small distance between them produce similar values of the objective function (this is defined more formally in Definition-2.2). Let us consider a set of functions $\mathcal{F}$ defined on the same search space X. If there exists such metric d on X that landscapes $\{(X, d, f) \mid f \in \mathcal{F}\}$ are well-conditioned then $\mathcal{F}$ is not closed under permutations of the search space [29]. In this case, the NFL theorem [56] guarantees the existence of an algorithm that outperforms other algorithms on average for functions in $\mathcal{F}$. So the existence of such metric is sufficient to design a problem solver for $\mathcal{F}$ that performs above average.

The well-conditioned optimization landscapes $\{(X, d, f) \mid f \in \mathcal{F}\}$ may exist for a set of functions $\mathcal{F}$ that a practitioner solves. However, obtaining the metric d for these landscapes usually requires a certain level of expertise in the domain. It means that a well-conditioned landscape gives away some information on the structure of the considered problems. While EAs do not require any information on the domain of the problem, it is well known that problem-specific knowledge significantly increases the performance of EAs [21]. We identify 4 main categories of approaches to take advantage of different kinds of domain-specific information in EA:

1) Asymmetric mutation operator. Methods in this category use the information about the problem to modify the mutation operator employed by an EA. Doerr et al [14, 15] consider the Eulerian cycle problem and rigorously show that asymmetry in mutation operator, for example avoiding certain modifications, leads to a performance increase compared to EA with classical mutations. Raidl et al [46] show the benefits of EA with a mutation operator that chooses rather edges with smaller weights to solve the minimum spanning tree problem and the traveling salesmen problem. Jansen and Sudholt analyzed the influence of a bias in the mutation operator when it is applied to pseudo boolean functions [30]. They considered a mutation operator that flips 1 with a higher probability when a solution contains more ones and vice-versa. The work rigorously shows that such bias speeds up $(1+1)$-EA equipped with such operator on three of the functions they considered, and slows down the algorithm on another function. Their work was extended by proposing an adaptation of the degree of asymmetry during the optimization process [47].

2) Enhanced representation. Traditionally, there are no specific restrictions on the used representation, but there are a number of recommendations [52]. In the EA community, the mainstream is to simply use the most natural representation of the individual [54]. However, finding such a representation may already require expert-level knowledge in the domain of the problem. Many works with a focus on practical applications prove this, for example, Yu et al applied an EA to the object decomposition for 3D printing [64], where they used a Binary Space Partitioning tree encoded as an array of continuous numbers. The design of such representation requires an understanding of the principles of object decomposition. In the area of spatial design creation, the challenging problem of defining a search space that does not significantly restrict the space of all possible solutions was subjected to comprehensive research in [42]. Moreover, standard EA may work significantly slower when the representation is not carefully chosen. Doerr et al developed such a representation for the Eulerian cycle problem that allowed to develop an EA with the arguably best reachable by RSH expected run-time [16].

3) Data-driven approaches. This category covers automated algorithm selection [25, 33, 35, 39, 63] and dynamic algorithm configuration [1, 9], where automatically extracted features of the problem are used to suggest an optimization method or a configuration of the algorithm.

4) Diverse set of candidate solutions. Diversity measure of the set of candidate solutions can be evaluated using a domain-specific distance, as it is suggested in [59]. It is known, that more diverse sets of parents in EA facilitate global exploration properties of the algorithm, reduce the risk to get stuck in local optima and allow obtaining multiple dissimilar high-quality solutions [60]. Niching methods were developed to maintain diverse sets of individuals in EA, for a comprehensive overview see [11, 43, 44]. Distance-based niching methods have a great number of successful applications, which shows the advantage of distance knowledge. For example, Dynamic Peak Identification niching [57] maintains high-performing solutions with a pairwise distance larger than the given lower bound. This method was successfully applied to find a number of high-performing designs of lenses in a single run of an EA in [34]. In the domain of airfoil optimization, a variation of niching was proposed, where the algorithm alternates between pure quality and distance-based diversity optimizations which allows to obtain better-performing solutions [49, 50].

As we see from the examples of approaches in categories 1 and 2, the user needs to have deep knowledge of both: the domain of the problem, and the optimization method. Approaches in category 3 have the disadvantage that the predicted algorithms/configurations do not always perform well, especially when a new (unseen) problem is dissimilar from the problems that the approaches are trained for. Approaches in category 4 leave the hard work to pick a distance that represents the problem well to the expert in the domain and allows using this distance as a black-box in the optimization algorithm. However, this distance has influence only on the selection part of EA. Since distance gives away information on the problem, it can also be leveraged in other parts of an EA to speed up the optimization even more. At the moment, there exist classical requirements to the design of crossover and mutation operators for an EA in a metric space [20, 21, 28]. Apart from this, there is a number of works, where neighborhood structure and distance were applied explicitly or via group theory, for example in the works [8, 40, 41, 55]. We observe in those works that a practitioner, who wants to use metrics in EA, needs to redesign the operators accordingly, which requires the application of methods from categories 1 and 2.

Our contribution: We propose two versions of the mutation operator which employs a given metric in the search space. It is possible to change the metric used in our mutation operators without amending the rest of the algorithm. The first developed mutation operator relies on the given distribution over mutation strengths. The second operator applies the distribution which is created automatically and allows control of its parameters. The application of such mutation operators in an EA does not require any additional effort from a practitioner to switch one metric to another and has great benefits where a natural representation makes the problem ill-conditioned. In this case, our mutation operator can *cure* the problem, if a domain-specific distance is known.

2 PRELIMINARIES

Evolutionary algorithms (EAs) are a class of search and optimization algorithms inspired by the principles of natural selection and Darwinian evolution. These algorithms are based on the idea that a population of solutions to a problem can evolve and improve over time through the application of selection, reproduction, and variation operators.

During the optimization process an EA samples elements of the search space X and evaluates the values of the objective function f in the sampled points. While doing it, EA ideally balances exploration/exploitation. It explores the search space by sampling elements with significantly different structures, which can be seen as gaining knowledge of the problem. The significance of the structural difference between sampled solutions can be measured with metric d, which is usually considered as the distance between elements in X. Despite the fact that the optimization problem is completely specified when X and f are given, it is convenient to talk about the landscape of the problem when the exploration/exploitation capabilities of an EA are analyzed. The landscape of the optimization problem is classically defined as follows.

Definition 2.1 (Landscape [51]). *For the problem f defined on the metric space X with metric d the landscape $\mathcal{L}$ is a triple (X, d, f).*

In this work, we distinguish between ill-conditioned and well-conditioned landscapes. In order to clearly define the latter ones we need to refer to small values in order to restrict the significance of the change in argument and in the objective value. Of course, the significance is always relative, so we assume that the knowledge of what is small already exists. In the following definition, we assume that this knowledge is given as sets of constants for the corresponding sets of real numbers.

Definition 2.2 (Well-conditioned landscape). *For the given search space X, metric $d : X^2 \to \mathbb{R}$ and the objective function $f : X \to \mathbb{R}$ consider sets of problem-dependent, small constants $\{C_{x_0,d(x_0,\cdot)} \mid x_0 \in X\}$, $\{C_{x_0,|f(\cdot)-f(x_0)|} \mid x_0 \in X\}$, that are given by an oracle. Given those constants, optimization landscape (X, d, f) is well-conditioned and metric d is well-conditioning if*

$$\forall x_0 \in X, \forall x \in \{x \mid d(x_0, x) < C_{x_0,d(x_0,\cdot)}\} \implies |f(x) - f(x_0)| < C_{x_0,|f(\cdot)-f(x_0)|}$$

The following algorithms are used in this work for the basis of our proposed mutation operators:

$(1+\lambda)$ *EA* is an elitist evolutionary algorithm without crossover. Given the objective function $f : X \to \mathbb{R}$, such an algorithm maintains the fittest solution and iteratively samples modifications of this solution. Each modification is obtained using the given parameterized distribution over the solution candidates [13].

$(1+(\lambda,\lambda))$ *EA* is an extension of $(1+\lambda)$ EA which uses crossover with bias c. At first, this algorithm samples λ solutions using mutation, then it picks the best mutant and recombines it λ times with the best-so-far solution. Each recombination applies crossover in such a way that for every component the bit is taken from the best-so-far solution with probability $(1-c)$, and from the best mutant with probability c. The best-so-far solution is updated with the fittest individual after such recombinations. This algorithm has proven itself well in theory [2, 3].

Randomized Local Search (RLS) is a stochastic local search algorithm. RLS iteratively explores the neighborhood of a current solution and selects a new solution based on a randomized criterion that takes into account the quality of the neighboring solutions. RLS is characterized by its simplicity, efficiency, and ease of implementation. In this work, we apply an extension of this algorithm to integer search spaces, so-called RLS-ab [12]. With this extension, the algorithm maintains the candidate solution, as a vector of n integers, and an additional vector of n integers that identify the step size with which every component of the solution is updated. This additional vector is called the vector of velocities. Values of velocities are adjusted independently during the optimization process. When a new mutant is fitter than the best-so-far solution, the corresponding velocity is increased, otherwise, it is decreased. We give the pseudocode of RLS-ab in Algorithm 3.

$(1+\lambda)$ EA_{ab} This is our simple extension of RLS-ab. The update adds a possibility to create λ individuals on every iteration and sample the number of modified components from the binomial distribution with success probability $1/n$. The operator to make mutation for every selected component stays the same, including the update rule of the parameters. For simplicity, we propose a pseudocode of a particular case of $(1+\lambda)$ EA_{ab} with $\lambda = 1$ in Algorithm 5.

Univariate Marginal Distribution Algorithm (UMDA) [36] is another population-based stochastic optimization algorithm. UMDA operates by sampling and updating the probability distribution of each variable in the solution vector independently, and generating new candidate solutions using a random sampling process based on the updated probability distributions. Every iteration it samples λ individuals and uses μ best of them to update the probability distribution. This algorithm does not use the notions of the neighborhood and so it has the potential to be efficiently applied even when a well-conditioned landscape does not exist for the problem. However, it is suggested to use a large enough value of λ to solve problems with difficult properties [17], so this algorithm may not be practically applicable right away to the initial optimization problem. We apply this algorithm to a relatively cheap function when we design our mutation operator. In this work, UMDA is adapted to handle integer domains. To achieve this we maintain the estimated probability of every integer in every component. We propose a pseudocode of this extension in Algorithm 4.

3 HIGH-LEVEL OVERVIEW OF THE PROPOSED APPROACHES

Given Definition 2.1 of a landscape, we say that an algorithm becomes landscape aware if it uses the specific distance d for defining the variation operators. In the scope of this work, we only propose ways to use this distance in the mutation operator. We will call a mutation operator *distance-driven* if it uses such distance to generate mutants. Accordingly, we will say that an algorithm is distance-driven if it uses a distance-driven mutation operator.

Let us consider RSH $\mathcal{A}$ which applies a mutation operator when it solves an optimization problem in the landscape $\mathcal{L}' = (X, f, d')$. We make an assumption that $\mathcal{L}'$ is not well-conditioned. At the same time, we assume that well-conditioned landscape $\mathcal{L} = (X, f, d)$ is given to us by an oracle. In this section, we propose a way to change $\mathcal{A}$ to a distance-driven algorithm dd-$\mathcal{A}$ which solves the optimization problem (at least partially) in the landscape $\mathcal{L}$. The optimization is performed in the new landscape completely when $\mathcal{A}$ uses only mutations to sample new solutions. However, we leave distance-driven crossover operators for future work, so if $\mathcal{A}$

uses recombination in addition to mutation, then dd-$\mathcal{A}$ partially works in the old landscape.

Both our distance-driven mutation operators rely on a parameterized family of distributions over all possible step sizes. In Section 4 we explain what we exactly mean by this. In Section 4 we assume that the distribution is defined by hand, which is possible when the so-called structure of the distance is known. Then in Section 5, we show how to create this distribution automatically for any given distance. The distance-driven mutation operator itself is given in Algorithm 1 and it is applied in both proposed methods, but with different distributions. Consequently, given a well-conditioned landscape $\mathcal{L}$ and distribution $\mathcal{D}$ defined in Section 4 we transform $\mathcal{A}$ to dd-$\mathcal{A}$ by substituting the usual mutation operator in $\mathcal{A}$ by the distance-driven mutation operator.

4 PROPOSED METHOD FOR THE DISTANCE WITH KNOWN STRUCTURE

Traditionally, mutation creates a structural change in the individual. The change is usually defined by the probabilistic distribution over different structural changes. The shape of the distribution reflects the intuition about the search. For example, in local search methods, the distribution is defined over only small mutations, that likely do not lead to quitting the basin of attraction. In global search methods, such as Evolutionary Strategies (ES), the distribution is defined over all the search space, but smaller changes happen with higher probability. Even though the mutation itself is sampled from this distribution, the significance of this change is defined w.r.t. the distance metric. Classically, it is assumed that the distribution is equipped with such a distance metric, that considers components independently and a small change in the value of every component contributes a small addition to the total value of distance. When the metric is different, but the same algorithm is applied, the mutations sampled from the same distribution likely do not give the properties that are expected from the perturbations.

Let us consider mutation operator μ of an optimization algorithm as a stochastic rule which chooses some mutant y for every candidate solution x such that $(x, y) \in X^2$. We will write $\mu(x)$ to denote a mutation of the individual x. We will say that *step size* of mutation w.r.t. distance d is the value $s \coloneqq d(x, \mu(x))$. The probabilistic distribution that is used in the mutation operator defines the step size of the operator. This step size may be sampled from the distribution explicitly, or implicitly, as happens in ES when the mutation itself is sampled. In the mutation operator that we design, we do not use anything from the distribution except the information on the step size. Hence, we consider the distribution defined over step sizes.

Let us define what we denote as the distribution over step sizes more precisely. First of all, we denote the set of all pairwise distances between points in the search space $S = \{d(x, y) \mid x, y \in X\}$ as *structure of the distance*. In this section, we assume that S is known to the designer of the algorithm. In order to allow balancing exploration/exploitation in the algorithm we add parametrization to the distribution, which significantly improves the performance of EA [4, 23]. All the parameters of the distribution are denoted with α. This vector controls the shape of the distribution and is updated outside of the mutation operator according to the rules defined in the distance-driven algorithm. For the given individual $x \in X$ we work with elementary outcomes $\{y \mid y = \mu(x)\}$ and probabilistic measure $\Pr_\alpha$ parameterized by a vector α. Given parameters α, individual $x \in X$ which is being perturbed and metric d, the distribution over step sizes from x is:

$$\mathcal{D}_{x,\alpha} \coloneqq \{\Pr_\alpha [d(x, \mu(x)) = d(x, y)] \mid y \in X\} \tag{1}$$

For the given α, distance-driven algorithms possess the following set of distributions in order to apply a distance-driven mutation operator at any point of X:

$$\mathcal{D}_\alpha \coloneqq \{\mathcal{D}_{x,\alpha} \mid x \in X\} \tag{2}$$

Then the parametrized family of sets of distributions is given by:

$$\mathcal{D} \coloneqq \{\mathcal{D}_\alpha \mid \alpha\} \tag{3}$$

We will say that $\mathcal{D}$ defined in Eq. 3 is *distribution family*. The choice of this distribution family is left to the user. To choose $\mathcal{D}$ it is sufficient to define $\mathcal{D}_\alpha$ in Eq. 2 and so it is sufficient to define $\mathcal{D}_{x,\alpha}$ in Eq. 1. Therefore it is sufficient to define the parameterized probability of mutation to the distance from any element in X to any element in X. In order to do this, it is sufficient to know the set S.

Our distance-driven mutation operator generates another individual that steps away from the initial one to the distance closest to the given step size. To implement this generation we solve another optimization problem as written in Algorithm 1. The goal of solving the problem in line 2 is to locate any such $y \in X$ that the distance between y and x is closest possible to the sampled step size s. If several pairwise different individuals are solutions to the optimization problem in line 2, then we do not distinguish between them and pick an arbitrary one of them.

Algorithm 1 Distance-driven mutation operator of individual $x \in X$ given distance $d : X^2 \to \mathbb{R}$, distribution family $\mathcal{D}$ and distribution parameters α

```
1: $s \sim \mathcal{D}_{x,\alpha}$                          ▷ Sample step size
2: $y \leftarrow \arg\min_y \{|d(x, y) - s|\}$          ▷ Internal optimization
3: return $y$
```

4.1 Analysis of internal optimization problem

In line 2 of Algorithm 1 the optimization problem is solved. In particular cases of constrained S it might be possible to solve this problem analytically or using a deterministic algorithm. However, when the size of S is too big or S is not known (as in Section 4) we can solve this problem with RSH. In order to pick an optimization algorithm to solve this problem we notice the following property.

Proposition 1. *If the landscape (X, h, f) is not well-conditioned, but landscape (X, d, f) is well-conditioned, then for all points $x \in X$ there exists $s \in \mathbb{R}$ such that the landscape $(X, h, g_{x,s})$ is not well-conditioned, where $g_{x,s}(y) \coloneqq |d(x, y) - s|$.*

Proof. Consider any $x \in X$ and $s = \max\{d(x, y) \mid y \in X\}$. Note that for this s and every point $y \in X$ we have

$$|g_{x,s}(y) - g_{x,s}(x)| = ||d(x, y) - s| - |d(x, x) - s|| = d(x, y)$$

Now let us assume that the landscape $(X, h, g_{x,s})$ is well-conditioned with this s for all $x \in X$. Then

$$\forall y_0 \in X, \forall y \in \{y \mid h(y_0, y) < C_{y_0,h(y_0,\cdot)}\} \implies |g_x(y) - g_x(y_0)| < C_{y_0,|g_x(\cdot)-g_x(y_0)|}$$

For $y_0 = x$ the latter transforms to $d(x, y) < C_{x,d(x,\cdot)}$. Since (X, d, f) is well-conditioned it implies that

$$|f(y) - f(x)| < C_{x,|f(\cdot)-f(x)|}$$

Combining these together, we obtain:

$$\forall x \in X, \forall y \in \{y \mid h(x, y) < C_{x,h(x,\cdot)}\} \implies |f(y) - f(x)| < C_{x,|f(\cdot)-f(x)|}$$

It means that (X, h, f) is well-conditioned, which contradicts the initial condition of the proposition. □

We assumed that the landscape (X, h, f) is not well-conditioned when h is Hamming distance. Then, following the Proposition 1, we see that the internal optimization problem may not be well-conditioned for some values of s. To optimize this function we pick an optimizer that does not use a notion of a neighborhood. One of the simplest such algorithms is UMDA. We applied it with a large λ and a big budget. In practical applications, λ evaluations of the objective function from the line 2 can be done in parallel, which reduces the wall-clock time spend for internal optimization.

Application of this version of the mutation operator requires the user to know the structure of distance and define the distribution family. Obtaining knowledge of the distance structure may require expertise in the domain of the problem. Moreover, the size of S may be extremely big, so defining the distribution family may not be feasible in some cases. To free the user from the necessity to learn this structure, we propose the following extension of the mutation operator, which automatically does both: explores the structure of distance and creates a distribution family.

5 PROPOSED METHOD FOR BLACK-BOX DISTANCE

The main purpose of this section is to propose a method that is able to extend the perturbation operator for the distance with an unknown structure. This allows the decoupling of an expert in optimization and an expert in the field of the domain. We expect that the distance created by an expert in the domain is well-conditioning and satisfies certain structural properties, which we summarize in the following assumption.

Assumption 1. *For any points $x, y \in X$:*

1. *Variability:* $|\{d(x, z) \mid z \in X\}| \gg 1$;
2. *Consistency:* $\min\{d(x, z) \mid z \in X \setminus \{x\}\} \approx \min\{d(y, z) \mid z \in X \setminus \{y\}\}$; *moreover* $\max\{d(x, z) \mid z \in X\} \gg \min\{d(x, z) \mid z \in X \setminus \{x\}\}$;
3. *Cheapness: wall-clock time required for computation of $d(x, y)$ is significantly smaller than the wall-clock time required for computation of the objective function at any point.*

The most important assumption for our methods is cheapness because the inner optimization problem is solved for every application of the mutation operator, which is normally called as often as the objective function. When the metric does not satisfy this property our method becomes very time-consuming. If the variability assumption is not upheld for many points, then the practitioner, who knows this fact, has already enough knowledge to use the previous method and create distributions family $\mathcal{D}$ based on their tastes. The consistency assumption is needed to save computational resources when we explore the space of possible values of the given metric. Our method is still applicable if this condition is not upheld for a subset of the search space, but for every point in this subset, we need to run a new exploration procedure, which becomes time-consuming.

In the case of black-box distance, it is impossible to define a distribution family $\mathcal{D}$ in advance. Here we propose a method to create such a distributions family that can be used in Algorithm 1. Designing this distribution family for the general case of the metric is challenging because of the two following problems. 1) *Different scales:* For different search points x, y and different metrics d_1, d_2 it may be the case that $d_1(x, y)/d_2(x, y) \gg 1$. 2) *Different gaps:* For different search points $x, y \in X$ and different metrics d_1, d_2, the values $v_i := |\{z \in X \mid \max\{d_i(x, z), d_i(y, z)\} < d_i(x, y)\}|$ may be significantly different for $i \in \{1, 2\}$.

At first, we propose a general transformation of the distance metric to solve both problems. Then we define the distributions family over the transformed values of distance. This transformed distance and distributions family are used in Algorithm 1 in the same way as in the previous subsection.

5.1 Transformation of the distance

Given metric d, for every point $x \in X$ we map $A_{d,x} = \{d(x, y) \mid y \in X\}$ to a subset of $(0, 1)$ with a monotonic function

$$\tau_\gamma(x, y) = 1 - \exp(-\gamma^2 d^2(x, y)) \tag{4}$$

The second term in the mapping is the well-known Gaussian function with codomain $(0, 1)$. Hence, the codomain of $\tau_\gamma(x, y)$ is $(0, 1)$. Let us define the set $\mathcal{I}_\gamma = \{\tau_\gamma(x, y) \mid y \in X \setminus \{x\}\}$. Such mapping transforms $A_{d,x}$ to $\mathcal{I}_\gamma \subset (0, 1)$ for any metric d and so allows to have the same ranges of distances, which solves the first problem we mentioned. The parameter γ of Eq. 4 is chosen in such a way that $\mathcal{I}_\gamma$ is as spread as possible. This requirement ensures that all the values of $\mathcal{I}_\gamma$ do not collapse to a very dense segment where different values of distance are not distinguishable given the limitations of the representation of floating point numbers.

To satisfy this requirement we consider the following problem regarding $\varepsilon_1, \varepsilon_2$:

$$\min \quad \varepsilon_1 \tag{5}$$
$$\min \quad \varepsilon_2 \tag{6}$$
$$\text{s.t.} \quad \min \mathcal{I}_\gamma \le \varepsilon_1 \tag{7}$$
$$\max \mathcal{I}_\gamma \ge 1 - \varepsilon_2 \tag{8}$$

The constraint Eq. 7 defines the upper bound on the smallest mapped distance, analogously constraint Eq. 8 defines the lower bound for the largest mapped distance.

For the purposes of our application, we specify the requirements for the solution of the problem. We aim to find sufficiently small ε_1 and ensure that ε_2 is small as well. Specifically, we iterate over values of ε_1 in the range $(10^{-4}, 10^{-2})$ with step 10^{-4} and stop when we find smallest $\varepsilon_1 \geq \varepsilon_2$ such that $\varepsilon_1, \varepsilon_2$ satisfy the constraints Eq. 7, Eq. 8.

To check if the given value of ε_1 satisfies the constraints we simplify them as follows. Using monotonic property of Eq. 4 we have:

$$\begin{cases} 1 - \exp\left(-\gamma^2 (\min A_{d,x})^2\right) \leq \varepsilon_1 \\ 1 - \exp\left(-\gamma^2 (\max A_{d,x})^2\right) \geq 1 - \varepsilon_2 \end{cases}$$

Which transforms to:

$$\frac{\sqrt{-\ln(\varepsilon_2)}}{\max A_{d,x}} \leq \gamma \leq \frac{\sqrt{-\ln(1-\varepsilon_1)}}{\min A_{d,x}} \tag{9}$$

It implies the following lower-bound for ε_2:

$$(1-\varepsilon_1)^{\left(\frac{\max A_{d,x}}{\min A_{d,x}}\right)^2} \leq \varepsilon_2$$

Then there exists ε_2 which satisfies our additional condition $\varepsilon_2 \leq \varepsilon_1$ if and only if ε_1 satisfies:

$$(1-\varepsilon_1)^{\left(\frac{\max A_{d,x}}{\min A_{d,x}}\right)^2} \leq \varepsilon_1 \tag{10}$$

To estimate the values $\zeta_{\max} := \max A_{d,x}$ and $\zeta_{\min} := \min A_{d,x}$ we consider fixed values $t, T \in \mathbb{R} : 0 < t < 1 < T$ and such sequence $(z_i)_{i=-k,\dots,k}$ that:

$$\begin{cases} z_i \sim \text{u.a.r.}(X \setminus \{x\}), & \text{if } i = 0 \\ d(x, z_i) \geq T d(x, z_{i-1}), & i > 0 \\ d(x, z_i) \leq t d(x, z_{i+1}), & i < 0 \end{cases}$$

As approximation of $\zeta_{\max}$ and $\zeta_{\min}$ we take $\widehat{\zeta_{\max}} := d(x, z_k)$, $\widehat{\zeta_{\min}} := d(x, z_{-k})$ accordingly. We substitute the values to Eq. 10 in order to check if ε_1 satisfies our constraints. When we obtain a solution $(\varepsilon_1^*, \varepsilon_2^*)$ we evaluate γ^*. It is taken as an average of its bounds obtained from Eq. 9. Steps above to compute $\varepsilon_1^*, \varepsilon_2^*, \gamma^*$ are summarised in Algorithm 2.

Algorithm 2 Computation of the parameters $\varepsilon_1, \varepsilon_2, \gamma$ for the given point $x \in X$

```
1:  $z_0 \leftarrow \text{u.a.r.}(X \setminus \{x\})$
2:  for $i \leftarrow 1, \dots, k$ do
3:      $z_i \leftarrow z \in \{z \in X \mid d(x,z) \geq T \cdot d(x, z_{i-1})\}$    ▹ Optimize
4:      $z_{-i} \leftarrow z \in \{z \in X \setminus \{x\} \mid d(x,z) \leq t \cdot d(x, z_{i+1})\}$    ▹ Optim.
5:  end for
6:  $\widehat{\zeta_{\max}} \leftarrow d(x, z_k)$
7:  $\widehat{\zeta_{\min}} \leftarrow d(x, z_{-k})$
8:  for $j \leftarrow 1, \dots, c$ do
9:      $\varepsilon_1 \leftarrow 10^{-4} j$
10:     $\varepsilon_2 \leftarrow \exp\left[\left(\frac{\widehat{\zeta_{\max}}}{\widehat{\zeta_{\min}}}\right)^2 \ln(1-\varepsilon_1)\right]$
11:     if $\varepsilon_2 \leq \varepsilon_1$ then break end if
12: end for
13: $\gamma \leftarrow \frac{1}{2}\left(\frac{\sqrt{-\ln(\varepsilon_2)}}{\widehat{\zeta_{\max}}} + \frac{\sqrt{-\ln(1-\varepsilon_1)}}{\widehat{\zeta_{\min}}}\right)$
14: return $\{\varepsilon_1, \varepsilon_2, \gamma\}$
```

In our implementation, we used the following constants: $k = 10, T = 1.2, t = 0.5, c = 100$. To find an element from the set defined in line 3 we apply UMDA to maximize the objective function $f_1(z) := d(x,z) - T \cdot d(x, z_{i-1})$ with stopping criteria that, apart from limiting the number of function evaluations, has condition $f_1(z) \geq 0$. We stop optimizing the function once the termination condition is satisfied for some solution. We return this solution as the element from the set. Since $T \cdot d(x, z_{i-1})$ can be an arbitrary number, we follow Proposition 1 and motivate the application of UMDA the same as earlier, i.e. in Section 4.1. Similarly, to compute z_{-i} in line 4 we apply UMDA to maximize the objective function $f_2(z) := t \cdot d(x, z_{i+1}) - d(x,z)$ and stopping criteria that includes conditions $f_2(z) \geq 0$ and $f_2(z) < t \cdot d(x, z_{i+1})$. The second additional condition is needed to ensure that $z_{-i} \neq x$. The first solution which satisfies the termination criteria is returned as the element from the set. It is possible that for some i and some $j \in \{1,2\}$ we have $\forall z : f_j(z) < 0$. It means that, the set defined in line 3 for $j = 1$ and in line 4 for $j = 2$ is empty. In this case we set z_i to z_{i-1} for $j = 1$, and set z_{-i} to z_{i+1} for $j = 2$.

Algorithm 2 takes as input a point from X. When a certain point $p \in X$ is passed as input to Algorithm 2 we say that the output values $\varepsilon_1, \varepsilon_2, \gamma$ are *computed for* the point p. Following our consistency assumption, the values of ε_1 and ε_1' are similar when computed for different points $x, x' \in X$. Using the same assumption we have: $d(x, z_k) \gg d(x, z_{-k})$. When the optimization problems in lines 3, 4 of Algorithm 2 are solved precisely we obtain: $\widehat{\zeta_{\max}} \gg \widehat{\zeta_{\min}}$. So we neglect the small change in $\left(\frac{\widehat{\zeta_{\max}}}{\widehat{\zeta_{\min}}}\right)^2$ and argue that values ε_2 and ε_2' are very close, when computed as in line 10 of Algorithm 2 for different points $x, x' \in X$. We take advantage of this and compute $\varepsilon_1, \varepsilon_2, \gamma$ only once for a point $x \in X$ chosen uniformly at random.

Following the variability assumption, $|A_{d,x}| \gg 1$ which makes elements of $\mathcal{I}_\gamma = \{\tau_\gamma(x,y) \mid y \in X\}$ located very close to each other in $\mathcal{I}_\gamma$. It means that there are no significant gaps between values in $\mathcal{I}_\gamma$, which solves the second problem.

5.2 Distribution over the transformed values

In this subsection, we derive distribution $\mathcal{D}_{x,\alpha}$ for arbitrary element $x \in X$ and parameters α. As we discussed before, if the distance function satisfies our assumptions, then the values $\varepsilon_1, \varepsilon_2, \gamma$ are practically the same for all the choices of $x \in X$. Hence, the distributions $\mathcal{D}_{x,\alpha}$ are the same for all x when parameters α are fixed. To be more concise, in this section, we will denote $\mathcal{D}_{x,\alpha}$ as $\mathcal{D}$.

The distribution over $\mathcal{I}_\gamma$ is chosen to be continuous to avoid computation of the exact value $|\mathcal{I}_\gamma|$. The fact that $\mathcal{I}_\gamma$ has finite size does not corrupt the algorithm, because of the big density of points in $\mathcal{I}_\gamma$. Moreover, the optimization problem in Algorithm 1 may not be solved precisely even when the distribution is discrete. It means that for a random variable $\xi \sim \mathcal{D}$ the value $\xi - \min\{|d(x,y) - \xi|\}$ may not be zero in practice, but close to zero in the case of discrete and continuous distributions. Then, the distribution is chosen as such $\mathcal{D}^*$ that have maximal differential entropy $H(\mathcal{D})$ over continuous distributions with fixed mean $m \in \mathcal{I}_\gamma$:

$$\mathcal{D}^* = \arg\max_{\mathcal{D}} \{H(\mathcal{D}) \mid \mathrm{E}(\xi \sim \mathcal{D}) = m\} \tag{11}$$

The maximal entropy is enforced to obtain an unbiased mutation operator [54]. We fix the mean of this distribution to allow controlling the step size during the optimization process. Distributions with fixed moments of higher order and other properties are left for future works. This is done because enforcing other properties will make it much more complex to solve the optimization problem defined in Eq. 11. On the other hand, distribution with a fixed mean and maximal entropy is already enough to efficiently solve problems with different complex properties, when the algorithm adjusts the mean during the optimization process. For example, one of $(1+\lambda)$ EA with mutation rate control optimizes a number of difficult pseudo-boolean functions faster than competitors [19]. The step size of mutation in $(1+\lambda)$ EA is sampled from the Binomial distribution. Mutation rate control is equivalent to controlling only the mean of this distribution. At the same time, Binomial distribution is the maximal entropy distribution among discrete distributions with the fixed mean and fixed support [26]. This motivates the hope, that our mutation operator with distribution from Eq. 11 has the potential to assist optimization better than mutation operators with other distributions, given that a smart enough method to control the mean is used.

In order to find the probability density function (pdf) $\mathcal{P}$ of distribution defined in Eq. 11 we consider the following optimization problem:

$$\arg\max_{\mathcal{P}} \quad -\int_{\varepsilon_1}^{1-\varepsilon_2} \mathcal{P}(x)\ln\mathcal{P}(x)dx \tag{12}$$

$$\text{s.t.} \quad \int_{\varepsilon_1}^{1-\varepsilon_2} \mathcal{P}(x)dx = 1 \tag{13}$$

$$\int_{\varepsilon_1}^{1-\varepsilon_2} \mathcal{P}(x)xdx = m \tag{14}$$

The following solution of Eq. 12 is unique for such $\lambda_0, \lambda_1 \in \mathbb{R}$ that satisfy the constraints Eq. 13, Eq. 14 [10, 31, 32]:

$$\mathcal{P}(x) = \exp(\lambda_0 + \lambda_1 x) \tag{15}$$

Substitution of Eq. 15 to Eq. 13, Eq. 14 and integration, gives us the following system:

$$\begin{cases} \frac{\exp(\lambda_0)}{\lambda_1}\left(e^{\lambda_1(1-\varepsilon_2)} - e^{\lambda_1\varepsilon_1}\right) = 1 \\ \frac{\exp(\lambda_0)}{\lambda_1^2}\left(e^{\lambda_1(1-\varepsilon_2)}(\lambda_1(1-\varepsilon_2)-1) - e^{\lambda_1\varepsilon_1}(\lambda_1\varepsilon_1 - 1)\right) = m \end{cases}$$

Combining those two equations into one with condition that $\lambda_1 \neq 0$ gives us the following:

$$e^{\lambda_1(1-\varepsilon_2)}(\lambda_1(1-\varepsilon_2)-1) - e^{\lambda_1\varepsilon_1}(\lambda_1\varepsilon_1 - 1) = \lambda_1 m\left(e^{\lambda_1(1-\varepsilon_2)} - e^{\lambda_1\varepsilon_1}\right) \tag{16}$$

If the solution of Eq. 16 exists regarding λ_1 then it is unique because the solution of Eq. 12 is unique. The left part of Eq. 16 converges to 0 for $\lambda_1 \to -\infty$ and converges to 0 for $\lambda_1 \to 0$. The right part of Eq. 16 is infinitely big for $\lambda_1 \to -\infty$ and converges to 0 for $\lambda_1 \to 0$. Moreover for $\lambda_1 \to -0$ the left part is greater than the right part. Since the functions of $\lambda_1 < 0$ in both parts are continuous there exists a solution of Eq. 16 for $\lambda_1 < 0$. To find the solution we apply binary search with bounds $(-K, 0)$, where K is sufficiently big.

6 EXPERIMENTS

In this section we outline the different experimental setups used to validate the proposed contributions. This is done using two experiments: binary OneMax-based problems to show the effectiveness of using distance-driven permutation in an easy-to-analyze use-case, and a discretized version of the BBOB benchmark suite to show the generalizability of the proposed approach.

6.1 Experimental Setup

Binary problems. In order to benchmark Algorithm 1 as it is, we consider a binary OneMax-based problem in n dimensions (i.e. with search space $X = \{0,1\}^n$). We adopt Ruggedness problem based on OneMax and implement it as in *W-Model* suite [62]. In our experiments, we define OneMax as $l_1(x) := |x|$ (L_1-norm of binary string x). Ruggedness uses a given permutation π and applies it to the objective values of the underlying function, which is OneMax in our case. Application of this permutation reorders the objective values. We define such Ruggedness as: $r_\pi(x) := \pi^{-1}(l_1(x))$. We define distance for this problem as $d(x,y) := |r_\pi(x) - r_\pi(y)|$. Note that this is not a metric since $d(x,y)$ can be zero even when $x \neq y$. However, this distance satisfies all other requirements of the metric because it is based on the absolute value. Moreover, the landscape (X, d, r_π) is well-conditioned. The permutation is stored implicitly, such that for every integer $0 < v < n$:

$$\begin{cases} r(x) = l_1(x) + v - 2(l_1(x) \bmod v) - 1, & \text{if } \lfloor l_1/v \rfloor \neq \lfloor n/v \rfloor \\ r(x) = l_1(x) + v_1 - 2k - 1, & \text{if } \lfloor l_1/v \rfloor = \lfloor n/v \rfloor \end{cases}$$

Here $v_1 \equiv n \pmod{v}$ and $k = (l_1(x) \bmod v_1) - ((\lfloor n/v \rfloor v) \bmod v)$. In our implementation, we considered $v \in [1,5]$. Note that with $v = 1$ Ruggedness is OneMax. Greater values of v create dissipative intervals of size v, for example, for $v = 5$ the objective value 99 of OneMax is permuted to 95, 99 to 96, …, 95 to 99.

We extended both $(1+\lambda)$ EA and $(1+(\lambda,\lambda))$ EA with our distance driven mutation operator. The modified algorithms are called dd-$(1+\lambda)$ EA and dd-$(1+(\lambda,\lambda))$ EA accordingly. Both algorithms are considered without any parameter control. In the implementation of Algorithm 1 we implemented Binomial distribution with n Bernoulli experiments with success probability $1/n$. Note that the value sampled from this distribution may be larger than the maximal distance reachable from point x (for example when it is of the form 11..100..0). In this case, the optimization in line 2 of Algorithm 1 is terminated when it runs out of budget. The parameters of UMDA taken in the experiments on binary problems are the following: $\mu = 50, \lambda = 100$, budget $= 1000$.

Integer problems. We consider integer problems to demonstrate the advantages of the distance-driven mutation operator when the distance satisfies the assumptions. The domain of those problems is

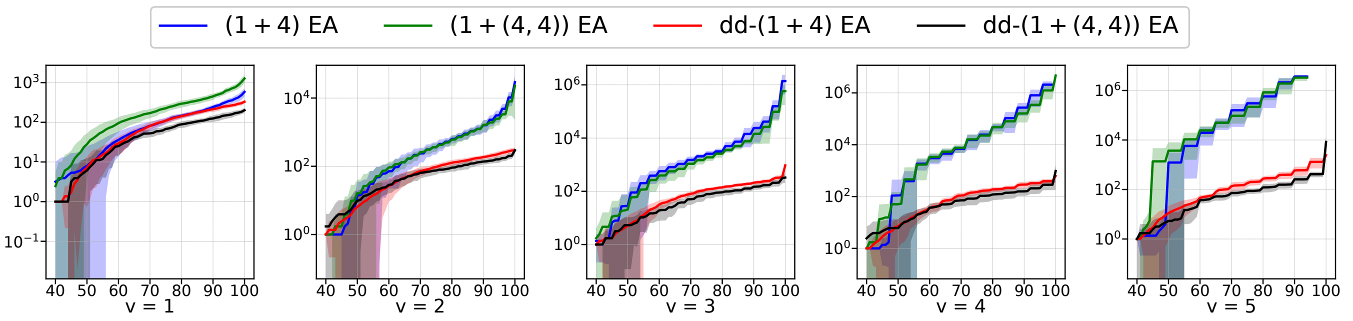

Figure 1: Average parallel optimization time and standard deviation of the considered algorithms (x-axis: best-so-far value of the objective function; y-axis: average number of iterations) on the Ruggedness with $v \in [1, 5]$. Dimensionality $n = 100$, 11 runs for every algorithm. UMDA is used as an internal optimizer of distance-driven algorithms with parameters $\mu = 50, \lambda = 100$, budget = 1000. The limit on the number of iterations is $5 \cdot 10^6$.

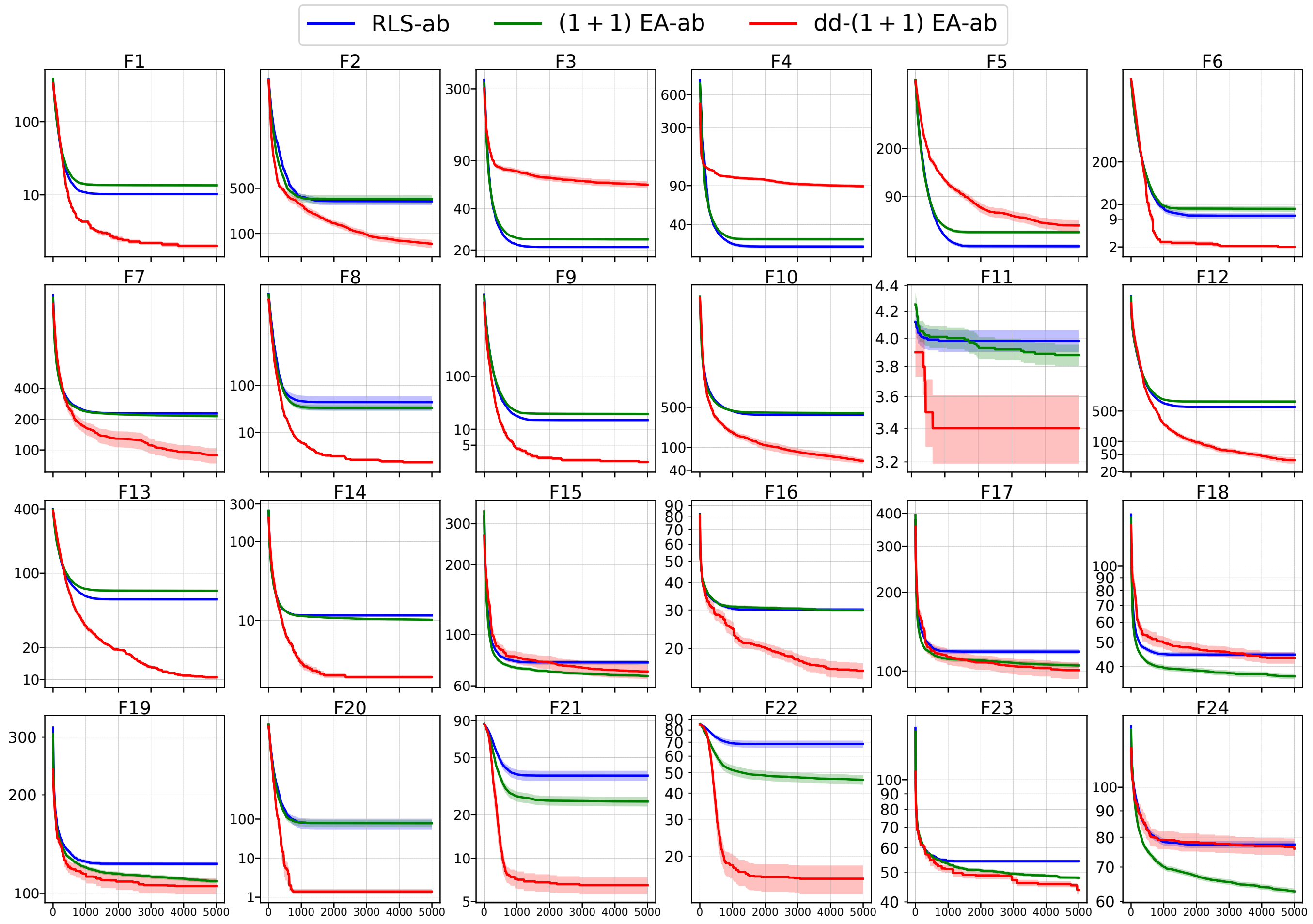

Figure 2: Convergence of the considered algorithms (x-axis: number of the objective function evaluations; y-axis: best-so-far value of regret and its standard error) on the transformed COCO/BBOB-MIXINT suite. Dimensionality $n = 40$, cardinality of every component is 100, 100 runs for RLS-ab and $(1 + 1)$ EA-ab, 10 run for dd$(1 + 1)$ EA-ab. UMDA is used as an internal optimizer of the distance-driven algorithm with parameters $\mu = 10^2, \lambda = 10^3$, budget = $4 \cdot 10^4$. The limit on the number of objective function evaluations is 5000.

$X = \Pi_{i=1}^{n}\{0, \ldots, c_i - 1\}$, where $(c_i)_{i=1\ldots n}$ are cardinalities. The problems are taken from COCO/BBOB-MIXINT suit [61]. We extended the problems to get rid of continuous variables and support the arbitrary cardinality of every component, the rest details on these problems were left unchanged. To show the advantages of our mutation operator, we transform those problems in our experiments. For the given permutation π and function f from BBOB-MIXINT we define the transformed function $Tf(x) := f\left[(\pi^{-1}(x_i))_{i=1,\ldots,n}\right]$. Here x_i stands for the i-th component of x. Note that here the same permutation is used to reorder integers in every dimension of the search space.We implemented this transformation because in our experiments the cardinalities of all the dimensions are equal. When permutation is applied to the search space the neighbourhood structures are destroyed in every component, so Euclidean distance becomes ill-conditioned. We define distance for the transformed problem as $d(x, y) := l_2(\left[(\pi^{-1}(x_i))_{i=1,\ldots,n}\right] - \left[(\pi^{-1}(y_i))_{i=1,\ldots,n}\right])$. This distance takes inverse permutations of every component and computes L_2-norm of the difference of obtained vectors. In all of the experiments, the same permutation was used for search space transformation.

In the experiments we consider $n = 40$ dimensions, each one with cardinality $c = 100$ for every problem in the suit COCO/BBOB-MIXINT. We extended UMDA to the domains where the cardinality of every component is greater than 2. The parameters of this extended UMDA taken in the experiments on those problems are the following: $\mu = 10^2, \lambda = 10^3$, budget = $4 \cdot 10^4$.

In our experiments we maximized negative regret, meaning the value $f(x^*) - f(x)$, where f is a problem, x^* is the global optimum of f, x is a candidate solution. At the same time, we plot positive regret in Figure 2 to avoid confusion.

We extended RLS algorithm with our mutation operator. Note that the new algorithm does not do local search anymore because the distribution is defined over all found values of distances. We control the mean of the distribution in the following way. Assume that the mutation operator produced individual y, and the best-so-far solution is x. Then if $f(y) > f(x)$ we increase the mean (which increases the expected step size) $m \leftarrow a \cdot m$, where $a > 1$. If $f(y) < f(x)$ then we reduce the mean $m \leftarrow b \cdot m$, where $b < 1$. The rest of the steps (except mutation and parameter update) are the same as in RLS algorithm. We call this modified algorithm dd-$(1 + 1)$ EA$_{\text{ab}}$. In the implementation, we take values $a = 1.001, b = 0.999$. To simplify the numerical solution of Eq. 16 we took $\varepsilon_1 = 0, \varepsilon_2 = 1$ at that equation (and only there). In this case the solution regarding λ_1 always belongs to $[-1/m, 0)$.

The implementation of all considered algorithms, problems and experiments can be accessed at Zenodo [5].

6.2 Results

Binary problems. In Figure 1 we compare the performance of the described algorithms on Ruggedness problem with different values of v. Algorithms with modified mutation operators performed much better than their analogues in all of the considered cases of $v > 1$. For $v = 1$ the performance of the algorithm is almost the same, indeed distance-driven mutation operator uses the same step sizes because the distribution is the same for all the algorithms. However, dd $-$ $(1 + (4, 4))$ EA slightly outperforms dd-$(1 + 4)$ EA on most of the points, except the final points for $v = 4$ and $v = 5$. When an individual is in the local optima, the mutation-only algorithm waits in this optima until a sufficiently big step size is sampled from the distribution. At the same time, crossover adds additional variability which helps to escape the local optima. This is the reason for the slight advantage of dd $-$ $(1 + (4, 4))$ EA on the most of optimization trajectory. At the same time, the performance of both algorithms is significantly better than their analogues and the gap between them grows with the growth of v.

Integer problems. In Figure 2 we compare the performance of dd-$(1 + 1)$ EA$_{\text{ab}}$ against two algorithms that are not aware of well-conditioning metrics for the problems. Despite the fact that we maximized the negative regret, we plotted the value of positive regret. So the lower the curve is in Figure 2 the better the performance of the corresponding algorithm. As expected, for the majority of functions, specifically for F1, F2, F6-F14, F16, and F20-F22 the increase in performance of dd-$(1 + 1)$ EA$_{\text{ab}}$ is significant compared to the other algorithms. It shows that our algorithm manages to explore the search space well enough to find the values of parameters that allow the mutation operator to generate small distances as well as big ones. However, we observe that performance is much worse on functions F3 and F4. For function F4, this may happen because both competitor algorithms got lucky with a permutation of the search space, which allowed them to jump to the local optima of better quality at the beginning, where they converged to the basin of attraction and got stuck. For function F3, the distance-aware algorithm may lose because it treats all the directions evenly. When it finds the ridge of F3, it continues to sample solutions outstanding to the sampled step size s. Vectors of all possible perturbations make a hyper ball of radius s which intersects many other points apart from the points on the ridge. So the probability to improve the value of the objective function becomes very small. This is the reason why we observe a very slow convergence of distance-driven algorithms on this function. For function F24 distance-driven algorithm also loses to the competitor that is able to make jumps to bigger step sizes in the search space. The reason for this might be the fact that basins of attraction on F24 are small, so the algorithm needs to make small steps to not jump from one valley to another. This is the weak spot of the distance-driven algorithm because in order to find smaller values of distance it needs to spend much more time on internal optimization, which makes the algorithm significantly slower. On top of that, the algorithm stops making jumps to other basins of attraction, because, after the initial convergence, the mean of the distribution becomes too small. It happens because the parameter control method that we pick for dd-$(1 + 1)$ EA$_{\text{ab}}$ is rather simple. On the rest of the functions, the distance-driven algorithm slightly wins or shows approximately the same performance as competitors.

7 CONCLUSION

In this work, we propose two extensions of the perturbation operator. Both operators are applicable to a broader class of algorithms than just EAs. They can extend other RSHs that use the notion of distance to make changes in the solution, for example, Simulated-Annealing. The first extension relies on the structure of the distance. The user of this approach is expected to design the distribution over step sizes that they target to see in the perturbation operator. When the step size is sampled from the distribution, it is used to solve the inner optimization problem, where the mutant is generated. The goal of the inner optimization is to generate such an individual that is a number of steps away from the given individual to a distance close to the sampled step size.

The second mutation operator uses distance as a black-box. It explores the search space prior to defining the distribution. The information about the problem that was obtained during the exploration is then used to define the distribution over the step size with maximal entropy and fixed value of the mean. To estimate the parameters of this distribution we solve the numerical optimization problem and repeat this optimization every time the mean changes. This allows us to control the strength of changes that are made in the mutation operator.

We observed the performance of the first operator on pseudo-boolean problems and the performance of the second operator on integer problems to empirically evaluate the proposed operators. From the results, distance-driven operators were found to do better on most functions. It shows the advantage of being aware of the distance during the optimization. However, the performance on some functions is worse, because of imperfections of parameter control, specificities of the considered problem, and due to the fact that our operator treats all the directions evenly.

In future work, we plan to identify directions in the metric space to assist the mutations to proceed in the most promising direction with higher probability. This should improve the performance of the functions F3. Moreover, it will allow us to apply the most advanced methods of parameter control that exist in Evolution Strategies in our mutation operators. Apart from this, we plan to investigate the possibility to create a surrogate for well-conditioning distance. The creation of such a model will significantly loosen the current constraints on the optimization problem for which our mutation operators are efficient.

ACKNOWLEDGMENTS

We would like to thank our collaborators at SRON for the real-world motivation that inspired this work.

REFERENCES

[1] Steven Adriaensen, André Biedenkapp, Gresa Shala, Noor Awad, Theresa Eimer, Marius Lindauer, and Frank Hutter. 2022. Automated dynamic algorithm configuration. *Journal of Artificial Intelligence Research* 75 (2022), 1633–1699.

[2] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. 2019. A tight runtime analysis for the (1+(λ, λ)) GA on LeadingOnes. In *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*. 169–182.

[3] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. 2020. The (1+(λ, λ)) GA is even faster on multimodal problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 1259–1267.

[4] Kirill Antonov, Maxim Buzdalov, Arina Buzdalova, and Carola Doerr. 2021. Blending Dynamic Programming with Monte Carlo Simulation for Bounding the Running Time of Evolutionary Algorithms. In *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 878–885.

[5] Kirill Antonov, Anna V. Kononova, Thomas Bäck, and Niki van Stein. 2023. *distance-aware-ea: FOGA release*. https://doi.org/10.5281/zenodo.7897290

[6] Anne Auger and Benjamin Doerr. 2011. Theory of randomized search heuristics: Foundations and recent developments. (2011).

[7] Thomas Bäck, Martin Schütz, et al. 1995. Evolution Strategies for Mixed-Integer Optimization of Optical Multilayer Systems.. In *Evolutionary Programming*. 33–51.

[8] Marco Baioletti, Alfredo Milani, and Valentino Santucci. 2020. Variable neighborhood algebraic differential evolution: An application to the linear ordering problem with cumulative costs. *Information Sciences* 507 (2020), 37–52.

[9] André Biedenkapp, H Furkan Bozkurt, Theresa Eimer, Frank Hutter, and Marius Lindauer. 2020. Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework. In *ECAI 2020*. IOS Press, 427–434.

[10] Thomas M Cover. 1999. *Elements of information theory*. John Wiley & Sons.

[11] Swagatam Das, Sayan Maity, Bo-Yang Qu, and Ponnuthurai Nagaratnam Suganthan. 2011. Real-parameter evolutionary multimodal optimization—A survey of the state-of-the-art. *Swarm and Evolutionary Computation* 1, 2 (2011), 71–88.

[12] Benjamin Doerr, Carola Doerr, and Timo Kötzing. 2018. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica* 80 (2018), 1732–1768.

[13] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. Optimal parameter choices via precise black-box analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 1123–1130.

[14] Benjamin Doerr, Nils Hebbinghaus, and Frank Neumann. 2006. Speeding up evolutionary algorithms through restricted mutation operators. In *Parallel Problem Solving from Nature-PPSN IX: 9th International Conference, Reykjavik, Iceland, September 9-13, 2006, Proceedings*. Springer, 978–987.

[15] Benjamin Doerr, Nils Hebbinghaus, and Frank Neumann. 2007. Speeding up evolutionary algorithms through asymmetric mutation operators. *Evolutionary Computation* 15, 4 (2007), 401–410.

[16] Benjamin Doerr and Daniel Johannsen. 2007. Adjacency list matchings: an ideal genotype for cycle covers. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. 1203–1210.

[17] Benjamin Doerr and Martin S Krejca. 2020. The univariate marginal distribution algorithm copes well with deception and epistasis. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 17–18.

[18] Benjamin Doerr and Frank Neumann. 2019. Theory of evolutionary computation: Recent developments in discrete optimization. (2019).

[19] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M Shir, and Thomas Bäck. 2019. Benchmarking discrete optimization heuristics with IOHprofiler. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1798–1806.

[20] Stefan Droste and Dirk Wiesmann. 2000. Metric based evolutionary algorithms. In *Genetic Programming: European Conference, EuroGP 2000, Edinburgh, Scotland, UK, April 15-16, 2000. Proceedings 3*. Springer, 29–43.

[21] Stefan Droste and Dirk Wiesmann. 2003. On the design of problem-specific evolutionary algorithms. *Advances in evolutionary computing: theory and applications* (2003), 153–173.

[22] Arkadiy Dushatskiy, Marco Virgolin, Anton Bouter, Dirk Thierens, and Peter AN Bosman. 2021. Parameterless Gene-pool Optimal Mixing Evolutionary Algorithms. *arXiv preprint arXiv:2109.05259* (2021).

[23] Ágoston E Eiben, Robert Hinterding, and Zbigniew Michalewicz. 1999. Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation* 3, 2 (1999), 124–141.

[24] IE Grossmann and Nikolaos V Sahinidis. 2002. Special issue on mixed integer programming and its application to engineering, part I. *Optim. Eng* 3, 4 (2002), 52–76.

[25] Xin Guo, Bas van Stein, and Thomas Bäck. 2019. A new approach towards the combined algorithm selection and hyper-parameter optimization problem. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2042–2049.

[26] Peter Harremoës. 2001. Binomial and Poisson distributions as maximum entropy distributions. *IEEE Transactions on Information Theory* 47, 5 (2001), 2039–2041.

[27] Thomas Jansen. 2013. *Analyzing evolutionary algorithms: The computer science perspective*. Springer.

[28] Thomas Jansen. 2013. Evolutionary Algorithms and Other Randomized Search Heuristics. *Analyzing Evolutionary Algorithms: The Computer Science Perspective* (2013), 7–29.

[29] Thomas Jansen. 2013. General limits in black-box optimization. *Analyzing Evolutionary Algorithms: The Computer Science Perspective* (2013), 45–84.

[30] Thomas Jansen and Dirk Sudholt. 2010. Analysis of an asymmetric mutation operator. *Evolutionary Computation* 18, 1 (2010), 1–26.

[31] Edwin T Jaynes. 1957. Information theory and statistical mechanics. *Physical review* 106, 4 (1957), 620.

[32] Edwin T Jaynes. 1957. Information theory and statistical mechanics. II. *Physical review* 108, 2 (1957), 171.

[33] Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. 2019. Automated algorithm selection: Survey and perspectives. *Evolutionary computation* 27, 1 (2019), 3–45.

[34] Anna V Kononova, Ofer M Shir, Teus Tukker, Pierluigi Frisco, Shutong Zeng, and Thomas Bäck. 2021. Addressing the multiplicity of solutions in optical lens design as a niching evolutionary algorithms computational challenge. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1596–1604.

[35] Ana Kostovska, Anja Jankovic, Diederick Vermetten, Jacob de Nobel, Hao Wang, Tome Eftimov, and Carola Doerr. 2022. Per-run algorithm selection with warm-starting using trajectory-based features. In *Parallel Problem Solving from Nature–PPSN XVII: 17th International Conference, PPSN 2022, Dortmund, Germany, September 10–14, 2022, Proceedings, Part I.* Springer, 46–60.
[36] Pedro Larrañaga and Jose A Lozano. 2001. *Estimation of distribution algorithms: A new tool for evolutionary computation.* Vol. 2. Springer Science & Business Media.
[37] Johannes Lengler. 2020. Drift analysis. *Theory of evolutionary computation: Recent developments in discrete optimization* (2020), 89–131.
[38] Rui Li, Michael TM Emmerich, Jeroen Eggermont, Thomas Bäck, Martin Schütz, Jouke Dijkstra, and Johan HC Reiber. 2013. Mixed integer evolution strategies for parameter optimization. *Evolutionary computation* 21, 1 (2013), 29–64.
[39] Fu Xing Long, Bas van Stein, Moritz Frenzel, Peter Krause, Markus Gitterle, and Thomas Bäck. 2022. Learning the characteristics of engineering optimization problems with applications in automotive crash. In *Proceedings of the Genetic and Evolutionary Computation Conference.* 1227–1236.
[40] Alberto Moraglio and Riccardo Poli. 2004. Topological interpretation of crossover. In *Genetic and Evolutionary Computation Conference.* Springer, 1377–1388.
[41] Alberto Moraglio, Julian Togelius, and Sara Silva. 2013. Geometric differential evolution for combinatorial and programs spaces. *Evolutionary computation* 21, 4 (2013), 591–624.
[42] Ksenia Pereverdieva, Michael Emmerich, André Deutz, Tessa Ezendam, Thomas Bäck, and Hèrm Hofmeyer. 2023. The Prism-Net Search Space Representation for Multi-objective Building Spatial Design. In *Evolutionary Multi-Criterion Optimization: 12th International Conference, EMO 2023, Leiden, The Netherlands, March 20–24, 2023, Proceedings.* Springer, 476–489.
[43] Mike Preuss. 2015. Groundwork for Niching. In *Multimodal Optimization by Means of Evolutionary Algorithms.* Springer, 55–73.
[44] Mike Preuss, Michael Epitropakis, Xiaodong Li, and Jonathan E Fieldsend. 2021. Multimodal Optimization: Formulation, Heuristics, and a Decade of Advances. In *Metaheuristics for Finding Multiple Solutions.* Springer, 1–26.
[45] Günther R Raidl and Jens Gottlieb. 2005. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary computation* 13, 4 (2005), 441–475.
[46] Günther R Raidl, Gabriele Koller, and Bryant A Julstrom. 2006. Biased mutation operators for subgraph-selection problems. *IEEE Transactions on Evolutionary Computation* 10, 2 (2006), 145–156.
[47] Amirhossein Rajabi and Carsten Witt. 2020. Evolutionary algorithms with self-adjusting asymmetric mutation. In *Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I.* Springer, 664–677.
[48] Ingo Rechenberg. 1989. Evolution strategy: Nature's way of optimization. In *Optimization: Methods and Applications, Possibilities and Limitations: Proceedings of an International Seminar Organized by Deutsche Forschungsanstalt für Luft-und Raumfahrt (DLR), Bonn, June 1989.* Springer, 106–126.
[49] Edgar Reehuis, Markus Olhofer, Bernhard Sendhoff, and Thomas Bäck. 2013. Novelty-guided Restarts for Diverse Solutions in Optimization of Airfoils. *A Bridge between Probability, Set-Oriented Numerics, and Evolutionary Computation, EVOLVE* 2013 (2013).
[50] Edgar Julius Jeronimus Reehuis. 2013. *Guiding evolutionary search towards innovative solutions.* Ph. D. Dissertation. Leiden University.
[51] Colin R Reeves. 2000. Fitness landscapes and evolutionary algorithms. In *Artificial Evolution: 4th European Conference, AE'99, Dunkerque, France, November 3-5, 1999. Selected Papers 4.* Springer, 3–20.
[52] Franz Rothlauf and Franz Rothlauf. 2006. *Representations for genetic and evolutionary algorithms.* Springer.
[53] Grzegorz Rozenberg, Thomas Bäck, and Joost N Kok. 2012. *Handbook of natural computing.* Springer.
[54] Günter Rudolph. 2012. *Evolutionary Strategies.* Springer Berlin Heidelberg, Berlin, Heidelberg, 673–698.
[55] Valentino Santucci, Marco Baioletti, and Alfredo Milani. 2020. An algebraic framework for swarm and evolutionary algorithms in combinatorial optimization. *Swarm and Evolutionary Computation* 55 (2020), 100673.
[56] Chris Schumacher, Michael D Vose, L Darrell Whitley, et al. 2001. The no free lunch and problem description length. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001).* 565–570.
[57] Ofer M Shir and Thomas Bäck. 2009. Niching with derandomized evolution strategies in artificial and real-world landscapes. *Natural Computing* 8 (2009), 171–196.
[58] Adam Slowik and Halina Kwasnicka. 2020. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications* 32 (2020), 12363–12379.
[59] Andrew R Solow and Stephen Polasky. 1994. Measuring biological diversity. *Environmental and Ecological Statistics* 1 (1994), 95–103.
[60] Dirk Sudholt. 2020. The benefits of population diversity in evolutionary algorithms: a survey of rigorous runtime analyses. *Theory of evolutionary computation: Recent developments in discrete optimization* (2020), 359–404.
[61] Tea Tušar, Dimo Brockhoff, and Nikolaus Hansen. 2019. Mixed-integer benchmark problems for single-and bi-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference.* 718–726.
[62] Thomas Weise and Zijun Wu. 2018. 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.* 1769–1776.
[63] Furong Ye, Carola Doerr, and Thomas Bäck. 2021. Leveraging benchmarking data for informed one-shot dynamic algorithm selection. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* 245–246.
[64] Eric A Yu, Jin Yeom, Cem C Tutum, Etienne Vouga, and Risto Miikkulainen. 2017. Evolutionary decomposition for 3D printing. In *Proceedings of the Genetic and Evolutionary Computation Conference.* 1272–1279.

A PSEUDOCODES

Algorithm 3 RLS-ab [12] for maximization of function $f : [0,\dots,r-1]^n \to \mathbb{R}$ with constants a, b such that $a > 1$ and $b < 1$

```
1:  $x \leftarrow$ u.a.r.$\{0,\dots,r-1\}^n$                ▷ Choose uniformly at random
2:  $v \leftarrow$ u.a.r.$\{1,\dots,\lfloor r/4 \rfloor\}^n$        ▷ Choose velocities
3:  for $t \leftarrow 1, 2, \dots$ do                      ▷ Repeat while having budget
4:      $y \leftarrow x$
5:      $i \leftarrow$ u.a.r.$\{0,\dots,n-1\}$              ▷ Choose position
6:      $k \leftarrow$ u.a.r.$\{0,1\}$
7:      if $k = 0$ then                                ▷ Pick a branch equiprobably
8:          $y_i \leftarrow x_i - \lfloor v_i \rfloor$
9:      else
10:         $y_i \leftarrow x_i + \lfloor v_i \rfloor$
11:     end if
12:     if $f(y) > f(x)$ then                          ▷ Candidate $y$ is fitter than $x$
13:         $v_i \leftarrow \min\{av_i, \lfloor r/4 \rfloor\}$      ▷ Increase the velocity $v_i$
14:     else
15:         $v_i \leftarrow \max\{1, bv_i\}$                ▷ Reduce the velocity $v_i$
16:     end if
17:     if $f(y) \geq f(x)$ then
18:         $x \leftarrow y$                            ▷ Update the parent
19:     end if
20: end for
```

Algorithm 4 Extension of UMDA [17] for integer search spaces to maximize the function $f : [0,\dots,r-1]^n \to \mathbb{R}$. Parameters λ and $\mu \leq \lambda$ are constants.

```
1:  $\left(p_i^{(0)}\right)_{i=1,\dots,n} \leftarrow \left(\frac{1}{(r-1)n}\right)_{i=1,\dots,n}$      ▷ Initialize the distribution
2:  $b \leftarrow \frac{1}{r}$                                ▷ Set lower bound for probability
3:  for $t \leftarrow 1, 2, \dots$ do                      ▷ Repeat while having budget
4:      for $i \leftarrow 1, 2, \dots, \lambda$ do              ▷ Produce population
5:          $x^{(i)} \sim p^{(t)}$                         ▷ Sample candidate solution
6:      end for
7:      $\left(y^{(i)}\right)_{i=1,\dots,\mu} \leftarrow \textsc{select}\left\{\left(x^{(i)}, f\left(x^{(i)}\right)\right)_{i=1,\dots,\lambda}\right\}$
8:      for $i \leftarrow 1, 2, \dots, n$ do
9:          $p_i^{(t+1)} \leftarrow \min\left\{1-b, \max\left\{b, \frac{1}{\mu}\sum_{j=1}^{\mu} y_i^{(j)}\right\}\right\}$
10:     end for
11: end for
```

Algorithm 4 uses operator SELECT which picks μ individuals with the greatest value of the objective function f from the given set of λ candidate solutions. One of the possible implementations of this operator could sort $\left(x^{(i)}\right)_{i=1,\dots,\lambda}$ in descending order according to the values $\left(f\left(x^{(i)}\right)\right)_{i=1,\dots,\lambda}$ and return first μ elements from the sorted list.

Algorithm 5 uses operator $\mathcal{B}_{0\to 1}(n, 1/n)$ which denotes Bernoulli Scheme of n independent experiments, each with probability of success $1/n$. The operator returns the indexes of successful experiments in the range $[0, n-1]$. If none of the experiments is successful, then the operator returns an integer in $[0, n-1]$ uniformly at random. The latter is denoted by $0 \to 1$ in our notation for this operator.

Algorithm 5 (1 + 1) EA-ab as extension of RLS-ab [12] for maximization of function $f : [0, \dots, r-1]^n \to \mathbb{R}$ with constants a, b such that $a > 1$ and $b < 1$

```
1:  $x \leftarrow$ u.a.r.$\{0, \dots, r-1\}^n$                 ▷ Choose uniformly at random
2:  $v \leftarrow$ u.a.r.$\{1, \dots, \lfloor r/4 \rfloor\}^n$                    ▷ Choose velocities
3:  for $t \leftarrow 1, 2, \dots$ do                  ▷ Repeat while having budget
4:      $y \leftarrow x$
5:      $(p_i)_{i=1,\dots,s} \sim \mathcal{B}_{0\to 1}(n, 1/n)$                 ▷ Sample positions
6:      $k \leftarrow$ u.a.r.$\{0, 1\}$
7:      for $i \leftarrow 1, 2, \dots, s$ do         ▷ Iterate every sampled position
8:          if $k = 0$ then                    ▷ Pick a branch equiprobably
9:              $y_{p_i} \leftarrow x_{p_i} - \lfloor v_{p_i} \rfloor$
10:         else
11:             $y_{p_i} \leftarrow x_{p_i} + \lfloor v_{p_i} \rfloor$
12:         end if
13:     end for
14:     for $i \leftarrow 1, 2, \dots, s$ do         ▷ Iterate every sampled position
15:         if $f(y) > f(x)$ then             ▷ Candidate $y$ is fitter than $x$
16:             $v_{p_i} \leftarrow \min\{a v_{p_i}, \lfloor r/4 \rfloor\}$ ▷ Increase the velocity $v_{p_i}$
17:         else
18:             $v_{p_i} \leftarrow \max\{1, b v_{p_i}\}$         ▷ Reduce the velocity $v_{p_i}$
19:         end if
20:     end for
21:     if $f(y) \geq f(x)$ then
22:         $x \leftarrow y$                                 ▷ Update the parent
23:     end if
24: end for
```