# Mixed Binomial Distributions for Binary Mutation Operators

Brahim Aboutaib
University of Minnesota Duluth
Duluth, Minnesota, USA

Andrew M. Sutton
University of Minnesota Duluth
Duluth, Minnesota, USA

## ABSTRACT

Mutation operators are crucial for evolutionary algorithms to make progress through a search landscape. Sometimes a mutation strategy that works in one part of the landscape is less effective in other regions of the landscape. If nothing is known about the best mutation operator, many strategies (such as self-adaptation, heavy-tailed mutation, variable neighborhood search) exist to overcome this. However, in some cases, some limited information may be available, either a priori or after probing. In this paper, we study the setting of a mixture of binomial distributions for pseudo-Boolean optimization. We show that, when a limited amount of information is available, evolutionary algorithms using mutation based on a mixture of binomial distributions can hill-climb and escape local optima efficiently.

## CCS CONCEPTS

• **Theory of computation** → **Design and analysis of algorithms**.

## KEYWORDS

Mutation Operators, Evolutionary Algorithms, Multimodal Optimization.

**ACM Reference Format:**
Brahim Aboutaib and Andrew M. Sutton. 2024. Mixed Binomial Distributions for Binary Mutation Operators. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3638529.3654010

## 1 INTRODUCTION

Designing efficient optimization algorithms is a challenging task. Multiple factors complicate this task, for example, how much information is known about the problem's structure and how much time is available to come up with a solution. Off-the-shelf heuristics like evolutionary algorithms usually require tuning before being deployed. This tuning is problem-sensitive and is influenced by the search landscape at hand. A general assumption about the landscapes of hard optimization problems is that they contain pathological features such as multiple local optima, fitness valleys, deceptive regions, neutrality, etc. These structural features are problem-dependent and impact the performance of optimization algorithms. In a fitness landscape, a locally optimal solution is a solution where its neighbors do not contain improving solutions. Local optima bend the fitness landscape and create valleys of less fit candidate solutions. Once an algorithm hits a local optimum, it needs to take steps to escape fitness valleys in order to find fitter solutions.

Escaping local optima can often require considerable changes to the local optimal solution. For example, in the iterated local search algorithm [15], a perturbation mechanism coupled with a local search algorithm are used to escape locally optimal solutions. Evolutionary algorithms employ several strategies for navigating fitness valleys. Populations allow the algorithm to simultaneously maintain different optimization paths, and non-elitist replacement mechanisms allow valleys of lower fitness to be traversed. Mutation and crossover can also be leveraged for crossing fitness valleys [4, 6]. On one hand, a large mutation rate makes significantly large jumps in Hamming distance, whereas a comparatively small mutation rate (e.g., the commonly recommended rate of $1/n$ for bit strings of length $n$) do not often make large jumps, which can make them easily stuck at local optima. On the other hand, a smaller mutation rate allows an algorithm to hill-climb efficiently, but does not easily jump over fitness valleys, especially in the context of elitist algorithms.

In this paper, we consider the question of designing mutation operators that can, at the same time, hill-climb and jump over fitness valleys efficiently. Our idea is to have two components, one that enables hill-climbing efficiently and another one that would enable an optimization algorithm to escape from local optima fast. Concretely, we design our mutation operator by mixing the binomial distribution using the mutation rate $\frac{1}{n}$, with another binomial distribution that uses a mutation rate proportional to the jump size. The motivation is the simple observation that in a multi-modal fitness landscape with local optima and valleys, an optimization algorithm would hit a local optima and get stuck until fitter solutions are sampled. If an algorithm is equipped with the classical $\frac{1}{n}$ mutation rate, it is shown that it will hill-climb efficiently. But this operator becomes inefficient to escape local optima. In an elitist setting, the probability that mutation with a $1/n$ rate finds the exact and correct bits to flip in order to sample fitter candidate solutions diminishes exponentially as the fitness valley grows.

If nothing is known about the landscape, then most likely the best strategy is to use *heavy-tailed* mutation or *self-adaptation* [1, 12]. However, if a small set of good candidates for different mutation rates are available *a priori* (due to problem analysis or probing), a mutation rate distribution consisting of a mixture of binomials might be more efficient. Considering mixtures of distributions is especially interesting from the perspective of automated algorithm configuration as it would enable tuning packages like i-race [14] to search for the best mixture of distributions to use.


*GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia*

ACM ISBN 979-8-4007-0494-9/24/07…$15.00
https://doi.org/10.1145/3638529.3654010

## 2 RELATED WORK

The challenges posed by multimodal fitness landscapes have been addressed with a number of different approaches. Most of these approaches focus on designing EAs that can escape local optima efficiently.

When no *a priori* information is available about the best mutation rate for escaping local optima, one of the best approaches is to use so-called *heavy-tailed* mutation [6]. In this scenario, a random mutation rate $d/n$ is used each time where $d$ is drawn from a power-law distribution. In the context of $\textsc{Jump}_m$ functions, this approach results in a runtime that is only a polynomial factor in $m$ slower than the optimal mutation rate for jumping over the gap from the local optimum to the optimal solution [6].

A drawback to the canonical Jump benchmark function is that it is fairly well structured in the sense to leave a local optimum, it is necessary to jump directly to the global optimum. This behavior may not generalize well to realistic functions, and to address this, Bambury, Bultel and Doerr [2] introduced the *generalized* jump functions in which valleys of low fitness with tunable width can lie at an arbitrary distance from the global optimum. They extend the results of heavy-tailed mutation to this class, but also point out that efficiencies on randomized local search equipped with stagnation detection [17] do not translate to the generalized scenario.

Friedrich et al. [10] investigate heavy-tailed mutation (both power law mutation and a probability that does not exponentially decay) on Jump as well as the combinatorial optimization problems minimum vertex cover and maximum cut.

Rajabi and Witt [16] introduced a *stagnation detection* mechanism that operates by systematically increasing the mutation rate as long as an improving move has not been generated. They proved that the expected runtime on Jump is asymptotically equivalent to the optimal mutation rate and outperforms heavy-tailed mutation approaches. The same authors developed this idea further to add the concept of radius memory that incorporates past successful mutation strength values into the calculation of the budget [17]. Recently, Doerr and Rajabi [7] have also investigated combining stagnation detection with heavy-tailed mutation.

Witt [19] further investigated variants of the Jump function including the generalized jump function mentioned above, as well as a version in which the optimum appears within the fitness gap. For the former, he showed that techniques (such as the cGA and a majority voting algorithm) that estimate the relative benefit of setting bits to one or zero can cross the gap of the offset jump even with comparatively large gaps. In contrast, the majority vote approach fails when the optimum is moved into the gap whereas the the cGA remains efficient.

Lehre and Qin [13] also study settings in which a static mutation rate fails. In particular, they analyze self-adaptation of two mutation rates in the context of non-elitist EAs under a prior noise model. They also prove that a uniform mixture of two mutation rates can optimize LeadingOnes in quadratic time, but fails in certain noisy envrionments. In [8], Doerr et al. studied the (1+1) EA on linear functions in which mutation is an arbitrary unbiased distribution over bit flips. They provided runtime bounds for linear functions taking into account only the mean of the distribution and the single bit-flip probability.

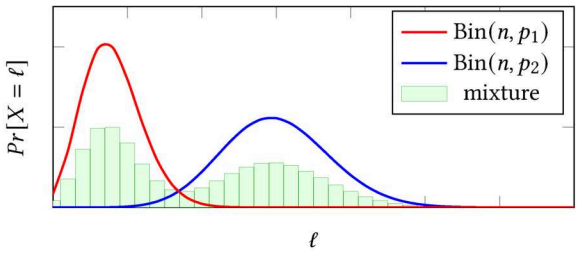



**Figure 1: Mixture of two binomial distributions. $\ell$ is the number of bits to flip.**

## 3 ALGORITHMS

We consider the (1+1) EA detailed in Algorithm 1 as a simple evolutionary algorithm framework. The (1+1) EA maintains a single candidate solution, a bit string of length $n$, that is mutated using a mutation operator in each generation. The produced solution replaces the current solution if it has a fitness superior or equal to the current solution's fitness. This process of mutation, fitness evaluation and replacement is repeated until a stopping condition is met. The mutation is carried out by sampling the number $\ell$ of bits to flip from a distribution $\mathcal{D}$. In the *common* setting of EAs, mutation is carried out by flipping each bit with probability $\frac{1}{n}$. This is equivalent to sampling the number $\ell$ bits to flip from the binomial distribution $\text{Bin}\left(n, \frac{1}{n}\right)$ and then choosing uniformly which $\ell$ bits of the $n$ to flip. As we are interested in different probability distributions to design mutation operators, we define our (1+1) EA independently of the distribution used. Once the number of bits to flip is sampled, the $\texttt{flip}(x, \ell)$ function is used to flip $\ell$ different positions in the offspring, and these locations are sampled uniformly at random.

Throughout this paper, we focus on distributions defined over binary search spaces $\mathcal{X} = \{0,1\}^n$. For a binary search space, we can use the Hamming distance between two solutions which counts the number of disagreements between two strings. Formally, let $x, y \in \mathcal{X}$ be two candidate solutions; the Hamming distance between $x$ and $y$ is $H(x, y) = |\{i \in [n] \mid x_i \neq y_i\}|$.

**Algorithm 1:** (1+1) EA maximizing a function $f : \mathcal{X} \to \mathbb{R}$.

```
x ← initialize solution u.a.r;
while stop condition not met do
    sample ℓ ~ D;
    y ← flip(x, ℓ);
    if f(y) ≥ f(x) then
        x ← y;
return x;
```

### 3.1 Binomial distribution based mutation operator

The binomial distribution $\text{Bin}(n, p)$ is a probability distribution on $\{0, 1, \ldots, n\}$, which translates to a distribution over the search space via uniform bit flips. Concretely, when using the binomial

**Algorithm 2:** `flip`$(x, \ell)$

select $\ell$ different positions $i_1, \dots, i_\ell \in [n]$ u.a.r.;
$y \leftarrow x$;
**foreach** $i_1, \dots, i_\ell$ **do**
  $y[i_1] \leftarrow 1 - y[i_1]$;
**return** $y$;

distribution the probability of obtaining solutions at Hamming distance $H(x, y)$ is

$$\Pr[H(x,y) = \ell] = \binom{n}{\ell} p^\ell (1-p)^{n-\ell}.$$

A mutation operator based on this distribution can be implemented by setting the parameters $n$ and $p$. As noted above, the classical setting for EAs is $p = \frac{1}{n}$, and when using this mutation probability, there is a probability of $(1 - \frac{1}{n})^n$ to not flip any bit. Unless the optimization scenario at hand for which evaluating the same solution multiple times can be beneficial, as when optimizing a noisy problem, it does not make sense to evaluate the fitness of the solution again. To mitigate this, one could condition the distribution on nonzero value, e.g., by resampling from the distribution until a nonzero value is obtained. As we do not consider noise in this paper, repeatedly evaluating the same solution is a waste of computational budget. Thus, when the sampled number of bits to flip $\ell$ equals zero, we resample from the distribution until $\ell \geq 1$. As noted by [9], this resampling results in a slightly modified probability distribution

$$\Pr[H(x,y) = \ell] = \frac{\binom{n}{\ell} p^\ell (1-p)^{n-\ell}}{1 - (1-p)^n}.$$

Throughout the rest of the paper, except where otherwise noted, the binomial distributions we consider use this resampling technique.

## 3.2 Mixed binomial based mutation operator

The mixed binomial distribution is a probability distribution that mixes multiple binomial distributions into a single distribution. Consider a mixed binomial distribution with $\kappa$ components. Each component $i \in [\kappa]$ is described by a binomial distribution $\mathrm{Bin}(n_i, p_i)$, with its own parameters $n_i$ and $p_i$. We associate a mixing coefficient $\alpha_i$ to each distribution, such that $0 \leq \alpha_i \leq 1$ and $\sum_{i=1}^{\kappa} \alpha_i = 1$. The probability mass function of the mixed binomial distribution is given by the weighted sum of the probability mass functions of the individual binomial components:

$$\Pr[H(x,y) = \ell] = \sum_{i=1}^{\kappa} \alpha_i \binom{n_i}{\ell} p_i^\ell (1-p_i)^{n_i-\ell} / (1 - (1-p_i)^{n_i}).$$

In this paper, we focus on the simplest nontrivial setting of $\kappa = 2$. Let $p_1$ and $p_2$ be the distributional parameter binomial distribution respectively, and $\alpha$ the mixing coefficient. We will study the distribution formally stated below.

$$\mathcal{D} := \alpha \,\mathrm{Bin}(n, p_1) + (1-\alpha)\,\mathrm{Bin}(n, p_2).$$

In the case of two components, the probability mass function becomes $\Pr[H(x,y) = \ell] = \alpha \binom{n}{\ell} p_1^\ell (1-p_1)^{n-\ell} / (1 - (1-p_1)^n) + (1-\alpha) \binom{n}{\ell} p_2^\ell (1-p_2)^{n-\ell} / (1 - (1-p_2)^n)$.

Using this distribution, we define the mixed binomial mutation operator as shown in Algorithm 3 (we refer to this operator as **m-bin**). The mixed binomial mutation operator proceeds by selecting the component to use according to the mixing coefficient. The selection of the component is done as follows. We sample $r$ uniformly at random and compute the cumulative sums of mixing coefficients. For the first mixing coefficient with a cumulative sum greater than or equal to $r$, we sample the variate $\ell$ from the corresponding binomial distribution and return $\ell$. A subsequent call to `flip`$(x, \ell)$ flips $\ell$ bits uniformly at random.

**Algorithm 3:** Mixed binomial distribution **m-bin**

$r \leftarrow rand([0,1])$;
**for** $i \in [\kappa]$ **do**
  $C \leftarrow \sum_{j=1}^{i} \alpha_j$;
  **if** $r \leq C$ **then**
    sample $\ell \sim \mathrm{Bin}(n, p_i)$;
    **return** $\ell$;

## 3.3 Heavy-tailed mutation operator

The heavy-tailed mutation operator was proposed in [6]. In a heavy-tailed operator, the number of bits to flip is chosen according to the power-law distribution $D^\beta_{n/2}$ for a fixed $\beta$. The main idea behind using a discrete power-law distribution $D^\beta_{n/2}$ is to increase the chances of flipping a large number of bits. More specifically, the distribution $D^\beta_{n/2}$ is defined as follow. Let $\beta > 1$, if a random variable $X$ follows $D^\beta_{n/2}$ then $Pr[X = \alpha] = (C^\beta_{n/2})^{-1} \alpha^{-\beta}$, where $C^\beta_{n/2} := \sum_{i=1}^{n/2} i^{-\beta}$. Using this distribution, Doerr et al. [6] defined the fast mutation operator $\mathbf{fmut}_\beta$. The mutation operator proceeds by sampling the mutation rate $\alpha$ from $[1, n/2]$, and flipping each bit in a binary string with a probability of $\alpha/n$.

# 4 ANALYSIS

## 4.1 Pseudo-Boolean Optimization Problems

We consider optimization problems defined over binary strings of size $n$, $\{0,1\}^n$.

*4.1.1 OneMax function.* We use the OneMax function as a simple model for assessing the hill-climbing performances of using mixed binomial distributions. We focus on the impact of the mixing coefficient $\alpha$ on the leading constant in the $n \log n$ optimization time. Indeed, as far the $\frac{1}{n}$ mutation rate is used with constant probability, then we trivially still have an $O(n \log n)$ upper bound. The interesting point is how the mixing with of $\frac{1}{n}$ with other distributions might impact this bound.

$$\textsc{OneMax}(x) = \sum_{i=1}^{n} x_i. \tag{1}$$

*4.1.2 Generalized Jump.* The *generalized jump* class of functions was recently introduced independently by Bambury et al. [2] and

Rajabi and Witt [17]. We consider this function class as a simple model to study the capacity of (1+1) EA to escape locally optimal solutions. For $x \in \{0,1\}^n$, the generalized jump function is defined as

$$\textsc{Jump}_{k,\delta}(x) = \begin{cases} |x| & \text{if } |x| \le n-k \text{ or } |x| \ge n-k+\delta. \\ -|x| & \text{otherwise.} \end{cases} \tag{2}$$

where $|x| = |\{i : x_i = 1\}| = \sum_{i=1}^{n} x_i$ is the number of ones in a string, $\delta$ is the gap size, and $k$ tunes the location of the gap size on the $\textsc{Jump}_{k,\delta}$ (i.e., the larger $k$, the further the gap is from the global optimum.).

*4.1.3 MAX-SAT.* One of the most studied problems in optimization and artificial intelligence is propositional satisfiability (SAT). Indeed, this problem is NP-complete, and many hard combinatorial problems can be naturally reduced to SAT. It can be used to represent many crucial problems, for instance in systems verification. SAT is a decision problem that can be stated as follows. A SAT formula is defined as a conjunction of $m$ clauses $c_1 \wedge \cdots \wedge c_m$. Let $x_1, \ldots, x_n$ be Boolean variables. A clause is defined as a disjunction of $k$ literals $l_1 \vee \cdots \vee l_k$, where a literal is a Boolean variable $x_i$ or its negation $\neg x_i$. A formula is satisfiable if there is an assignment to the Boolean variables such that the formula is true.

The optimization problem that corresponds to SAT is the MAX-SAT problem. In the MAX-SAT problem, the objective is to find an assignment to the boolean variables that maximize the number of true clauses in a propositional formula. Formally we can state the MAX-SAT as:

$$\begin{aligned} &\text{maximize } f_{\text{maxsat}} \\ &\text{where } f_{\text{maxsat}}(x) := |\{c \in F : \text{c is satisfied by x}\}|. \end{aligned} \tag{3}$$

where $F$ is the set of clauses and $x$ is an assignment to the $\{x_1, \ldots, x_n\}$ Boolean variables.

## 4.2 Theoretical Analysis

Runtime bounds for static mutation rates can usually be easily be translated to the mixture mutation setting as we show in the following lemma.

Lemma 4.1. *Let $T^* := T^*(n)$ be the runtime of the (1+1) EA on some problem class using a static mutation probability $p^* := p^*(n)$. Suppose $\mathrm{E}[T^*] \le g(n)$ where $g(n)$ is a bound obtained using the fitness level method [18].*

*Let $\mathcal{D}$ be a mixture of binomial distributions with $\{p_1, p_2, \ldots, p_\kappa\}$. If $p^* \in \{p_1, p_2, \ldots, p_\kappa\}$, then the expected runtime of the (1+1) EA using binomial mixture mutation $\mathcal{D}$ is bounded above by $\max_i\{\alpha_i^{-1}\}g(n)$. If $\mathcal{D}$ is a uniform mixture, then the expected runtime is at most $\kappa g(n)$.*

Proof. Let $\{A_1, \ldots, A_m\}$ be a partition of the search space into nonempty sets such that for all $1 \le i < j \le m$ and for all $x \in A_i$, $y \in A_j$ it holds that $f(x) < f(y)$ and $A_m$ contain only global optima.

Let $s_j$ be a lower bound on the probability of the (1+1) EA using a static mutation rate $p^*$ creating an offspring in $A_{j+1} \cup \cdots \cup A_m$ given that the current point is in $A_j$, Then by the fitness-level method [18, Theorem 2], $\mathrm{E}[T^*] \le \sum_{j=1}^{m-1} \frac{1}{s_j} =: g(n)$.

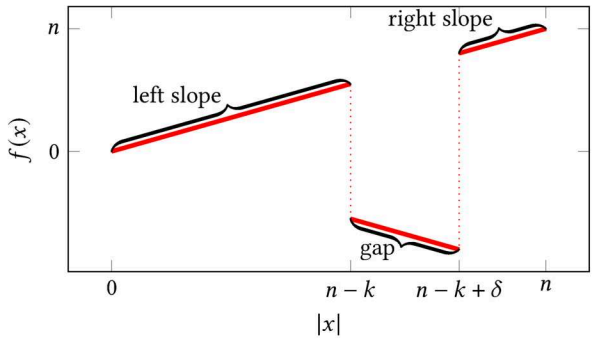



**Figure 2: The landscape of generalized $\textsc{Jump}_{k,\delta}$.**

Now consider the runtime of the (1+1) EA on the same problem class with binomial mixture mutation $\mathcal{D}$ with parameters $\{p_1, \ldots, p_\kappa\}$, and suppose there is some $1 \le i \le \kappa$ such that $p_i = p^*$. It follows that level $A_j$ can be escaped in one step by choosing mutation rate $p_i$ (which happens with probability $\alpha_i$) and then mutating the point with mutation rate $p^*$. Thus the escape probability for $A_j$ in this setting is at least $\alpha_i s_j$. Again, by the fitness level method, the expected runtime is bounded above by

$$\sum_{j=1}^{m-1} \frac{1}{\alpha_i s_j} = \frac{1}{\alpha_i} \sum_{j=1}^{m-1} \frac{1}{s_j} \le \max_i\{\alpha_i^{-1}\} g(n).$$

In the case of a uniform mixture, $\alpha_i = 1/\kappa$ for all $1 \le i \le \kappa$, so we would have $\max_i\{\alpha_i^{-1}\} = \kappa$. □

For generalized jump function with parameters $k$ and $\delta$, the asymptotically optimal static mutation rate was proved by Bambury et al. [2].

Theorem 4.2 (Theorem 8 of [2]). *Let $\delta \le k = o(n^{1/3})$. For the (1+1) EA with static mutation rate on $\textsc{Jump}_{k,\delta}$, the mutation rate in $[0, \frac{1}{2}]$ that asymptotically minimizes the expected runtime is $\delta/n$ and results in a runtime of*

$$(1 + o(1))\binom{k}{\delta}^{-1} \left(\frac{en}{\delta}\right)^{\delta}.$$

*Moreover, for any constant $\epsilon \in (0,1)$, a mutation rate of $(1 \pm \epsilon)\delta/n$ results in an increase in the runtime by a factor exponential in $\delta$.*

Theorem 4.2 provides the optimal static mutation rate for the (1+1) EA, which is required for the process to jump over the size $\delta$ gap. However, in the case where $\delta$ is large, once the process clears the gap, the $\delta/n$ mutation rate could be too strong to hill-climb efficiently to the global optimum.

To illustrate this, we consider two subsets of the landscape of $\textsc{Jump}_{k,\delta}$: the *left slope*, which is the set $L = \{x \in \{0,1\}^n : f(x) \le n-k\}$, and the *right slope*, which is the set $R = \{x \in \{0,1\}^n : n-k+\delta \le f(x) < n\}$. This is illustrated in Figure 2. The optimal static mutation rate for generalized jump is too high to efficiently hill-climb on the right slope. This is captured by the following theorem.

Theorem 4.3. *Let $n-k+\delta > n/2$ with $\delta = \omega(\log n)$. Consider the (1+1) EA with on $\textsc{Jump}_{k,\delta}$ using the optimal static mutation rate of $\delta/n$. After the first time the EA generates some point on the right slope,*

*it requires in expectation at least $e^{\delta/2}$ additional steps to generate the global optimum, which is superpolynomial in $n$.*

Proof. We consider the point in time after the (1+1) EA using static mutation rate of $\delta/n$ generates for the first time a point $x \in R$ on the right slope of the $\textsc{Jump}_{k,\delta}$ landscape. Since the (1+1) EA is elitist, after this time, every subsequent point accepted must have fitness at least $n-k+\delta$, and thus must have at least $n-k+\delta$ ones. A necessary condition for generating the global optimum is that mutation must not flip any of these ones to zero. For a $\delta/n$ static rate, the probability that mutation does not flip any ones to zero is thus at most

$$\left(1-\frac{\delta}{n}\right)^{n-k+\delta} \leq \exp\left(-\frac{\delta(n-k+\delta)}{n}\right) \leq e^{-\delta/2},$$

since $(k-\delta)/n \leq 1/2$. The waiting time until this event occurs is distributed geometrically with success probability bounded as above, and the expected waiting time until the global optimum is generated is at least $e^{\delta/2}$ as claimed. □

A simple binomial mixture mutation that includes both the optimal rate and the $1/n$ rate alleviates the problem of hill-climbing the right slope, which we show in the following theorem. Furthermore, it would only contribute at most an extra $\alpha$ factor to the waiting time to reach the right slope from the left slope.

Theorem 4.4. *Let $\alpha \in (0,1)$ and let $\mathcal{D}$ be a mixture of two binomial distributions $\alpha \operatorname{Bin}(n,1/n)$ and $(1-\alpha)\operatorname{Bin}(n,\delta/n)$. Consider the (1+1) EA with on $\textsc{Jump}_{k,\delta}$ using binomial mixture distribution $\mathcal{D}$. After the first time the EA generates some point on the right slope, it requires in expectation at most $(en/\alpha)\log n + en/\alpha$ additional steps to generate the global optimum.*

Proof. Consider the point in time after the process has generated a point $x \in R$ on the right slope. Hereafter, it is always possible to generate an improving move by flipping a single zero bit to a one. If $x$ is the current point, this event occurs with probability

$$\alpha \cdot \left(\frac{n-|x|}{n}\right) \cdot \left(1-\frac{1}{n}\right)^{n-1} \geq \frac{\alpha(n-|x|)}{en}.$$

We may thus consider the potential function $\varphi := x \mapsto n-|x|$. Starting from the first time after a point on the right slope is generated, let $(X_t)_{t\in\mathbb{N}}$ be the stochastic process corresponding to the value of the potential function in iteration $t$ of the EA. Then the drift of the potential is bounded as

$$\mathrm{E}[X_t - X_{t+1} \mid X_t = s] \geq \frac{\alpha s}{en}.$$

Let $T = \inf\{t : X_t = 0\}$ be the number of iterations until the potential is zero (and thus the optimum has been generated). By the multiplicative drift theorem [5],

$$\mathrm{E}[T] \leq \frac{en\log(n-k+\delta)}{\alpha} + \frac{en}{\alpha}.$$

□

Note that the runtime bounds of Theorem 4.3 and 4.4 assume standard mutation without the resampling procedure outlined in Section 3.1. Using the resampling procedure speeds up the runtime by a constant factor that lies between $1/(2e)$ and $1/e$, but this does not impact the bounds.

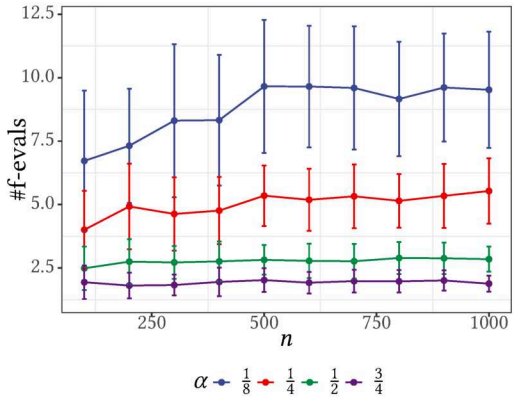



**Figure 3: Mean optimization time, and standard deviation, divided by $n\log(n)$ of (1+1) EA with mixed binomials on OneMax as a function of $n$ for different $\alpha$.**

## 4.3 Experimental Analysis

In this section, we detail our experimental analysis. We run EAs using mutation operators based on various distributions on different pseudo-boolean benchmarks. We first present our experiment design in section 4.3.1, then report our results and analysis in section 4.3.2.

### 4.3.1 *Experiment Design.*

***Optimization time analysis.*** In this first empirical analysis, we focus on the growth of the expected optimization time as a function of problem size. We compare mixed binomial mutation operations against using a single binomial distribution, specifically the classical $\operatorname{Bin}(n,1/n)$ mutation operator. We also considered the heavy-tailed mutation operator. As benchmark problems, we consider the OneMax function, the generalized $\textsc{Jump}_{k,\delta}$ function, and the MaxSat problem.

Our motivation for choosing the OneMax function is to characterize how well the (1+1) EA equipped with mixed binomial mutation can perform simple hill-climbing. We consider OneMax instances with dimension $n \in \{100,\ldots,1000\}$. We consider a mixed binomial distribution with two components, where $p_1 = \frac{1}{n}$ and $p_2 = \frac{\lceil\log(n)\rceil}{n}$. For the mixing coefficient $\alpha$, we consider the following settings $\{\frac{1}{8},\frac{1}{4},\frac{1}{2},\frac{3}{4}\}$.

With the generalized Jump function, our goal is to consider a simple multimodal model to characterize the capacity of mixed binomial mutations on fitness landscape with valleys, local optima, and global optima solutions. We consider the following parameters for $\textsc{Jump}_{k,\delta}$. For the problem size $n \in \{50,\ldots,150\}$ and $k \in \{i\lceil\log(n)\rceil \mid i \in \{2,\ldots,9\}\}$. The gap size $\delta$ is considered as a function of the parameter $k$. That is, for a fixed $k$ we consider $\delta \in \{\frac{k}{i} \mid i \in \{4,\ldots,10,\}\}$. The shown statistics[1], the mean and the standard deviation, are computed over 100 independent runs of the algorithm for each considered instance. For each run, we fixed a budget of $10n^3$ function evaluations, after which the algorithm is stopped.

[1]We use $\log_{10}$ base to plot the number of fitness evaluations.

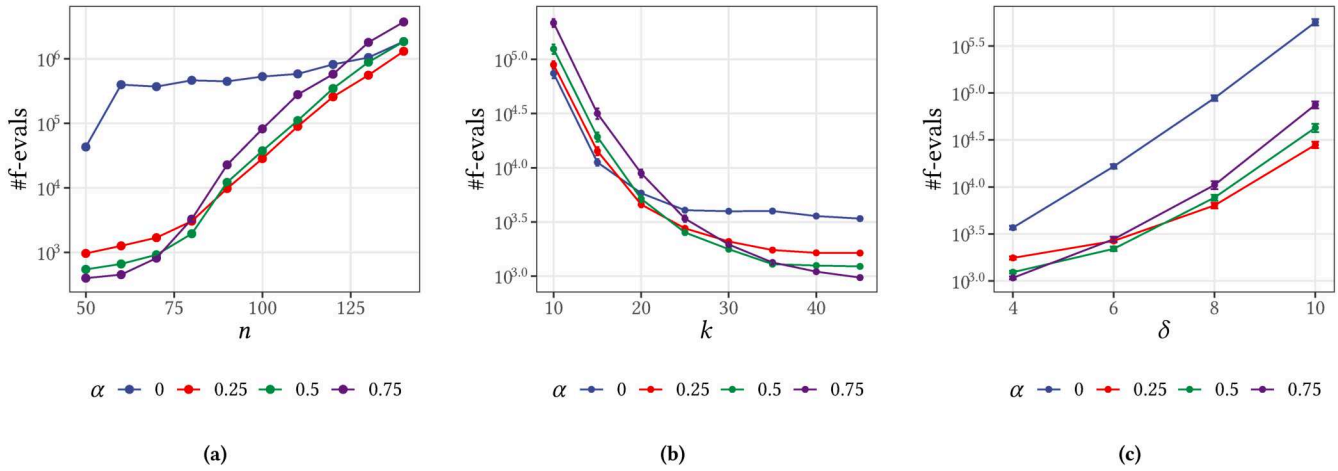



**Figure 4: Mean optimization time of (1+1) EA with mixed binomials on $\text{Jump}_{k,\delta}$ as a function of $n, k, \delta$ for different $\alpha$. (a) $k = 8\lceil\log(n)\rceil, \delta = k/4$. (b) $n = 100, \delta = \lceil\log(n)\rceil$. (c) $n = 100, k = 40$.**

***Mixing coefficient and mutation rates.*** For the second empirical analysis, we focus more on understanding mixing of binomial distributions. We study the interaction between the mixing coefficient and the choice of mutation rates for each binomial distribution.

***MaxSat instances.*** For the MaxSat problem we consider instances from SATLIB [11] benchmark which are uniform random instances and real-like instances in which variable frequency follows a power law [3]. From SATLIB, we selected randomly three instances with 50 variables. These instances are `uf50-062`, `uf50-077`, and `uf50-084`. The number of clauses for all three instances is 218. All three instances are satisfiable. For the real-like instances, we used the generator of [3] to generate three instances with 50 variables. For these instances, the number of clauses is 150, 165, and 175 respectively. We consider these clause counts so that the ratio of clauses to variables (i.e., 3.0, 3.3, and 3.6 for the consider instances) is in the region of difficult instances as shown in [3]. The power law's parameter is set to $\beta = 2.5$. For both uniform and power law instances, every clause was length three.

#### 4.3.2 *Experimental results.*

***Hill-climbing behavior.*** Figure 3 reports the mean optimization time of (1+1) EA using mixed binomial operator as a function of problem dimension $n$ on OneMax, for different mixing coefficient $\alpha$. We notice that (1+1) EA with m-bin hill climbs efficiently, especially for higher $\alpha$ values. Notice also that for the considered range of mixing coefficients, the (1+1) EA optimizes OneMax in $n\log(n)$. The mixing coefficient impacts the leading constant and induces a slowdown factor proportionate to $\alpha$. The lower $\alpha$, the higher the leading constant. This behavior is expected as small $\alpha$ allows more trails to the second component, which becomes inefficient as the algorithm gets close to the optimum especially when the mutation probability of this component is high.

***Analysis of (1+1) EA on $\text{Jump}_{k,\delta}$.*** In figure 4, we plot the mean optimization time of (1+1) EA with mixed binomial mutation operator as a function of problem size $n$ and parameters $k$ and $\delta$. All the experiments reported in this figure are carried out with the distribution $\alpha\,\text{Bin}(n, 1/n) + (1-\alpha)Bin(n, \delta/n)$. Note that $\alpha = 0$ means that we only use the $Bin(n, \delta/n)$ as a probability distribution. From figure 4a where $k = 8\log(n)$ and $\delta = k/4$, we can see for the considered problem sizes that the (1+1) EA with a mixed binomial distribution can solve the $\text{Jump}_{k,\delta}$ efficiently compared with only using the optimal static $Bin(n, \delta/n)$ for mutation.

Figure 4b shows the optimization time as a function of parameter $k$. In this experiment, we fix $n = 100$ and $\delta = \log(n)$. We can see that when $k$ is set to high values ($k \geq 20$), the (1+1) EA with mixed binomial distributions performs better compared to only using $\delta/n$ as a mutation rate, especially for small mixing coefficient values. This can be explained by the fact that once (1+1) EA jumps over the gap, using the mixing can find improving solutions on the right slope faster compared to only using $\delta/n$ as it can flip single bits more often.

To understand the impact of the gap size in $\text{Jump}_{k,\delta}$, we plot in figure 4c the optimization time of (1+1) EA as a function of $\delta$. Here we set $k = 40$ and $n = 100$. We can see that (1+1) EA with mixed binomials performs better compared with only using $\delta/n$. We also notice that for large $\delta$, taking a small mixing coefficient further improves the optimization time. This is expected, as a larger gap size requires more time to jump over, thus giving more trials to the second component (i.e., taking small $\alpha$) would improve the optimization time. At the same time, large $\alpha$ would increase the optimization time as $\text{Bin}(n, 1/n)$ probability is unlikely to sample solutions at some Hamming distance greater than $\delta$.

***Mixed binomial mutation operator analysis.*** After characterizing the optimization time of the (1+1) EA using mixed binomials on $\text{Jump}_{k,\delta}$, now we aim to better understand the behavior of the mixed binomial mutation operator by considering the interaction between the binomial distribution and the mixing coefficient. We focus on the interaction of the second component $\text{Bin}(n, p_2)$ and $\alpha$. For all the experiments we report here we fix $p_1 = \frac{1}{n}$. In figure

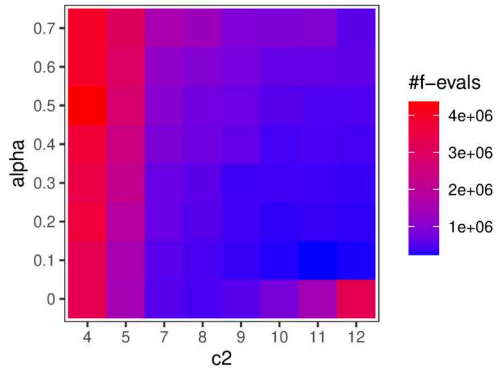



**Figure 5: Heat-map of mean optimization time of (1+1) EA with mixed binomial on $\text{Jump}_{25,8}$ as a function of $\alpha$ and $p_2$ ($p_2 = c_2/n$). $n = 100$.**

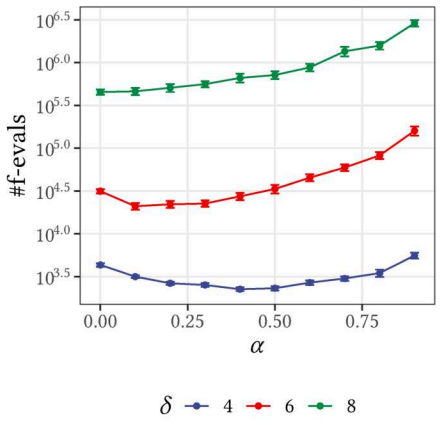


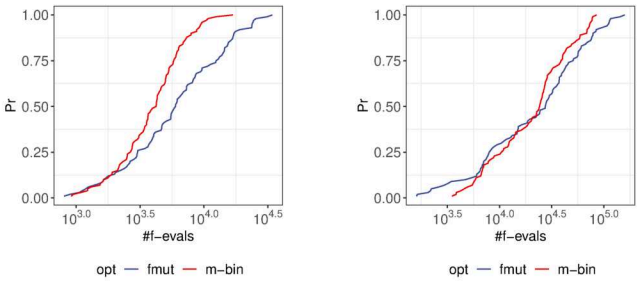

**Figure 6: Mean optimization time of (1+1) EA with mixed binomials on $\text{Jump}_{25,\delta}$ as a function of $\alpha$, for $\delta = 4, 6, 8$. $n = 100$.**

5, we plot the heat-map of mean optimization time for the interaction of $\alpha$ and $p_2$. We fix $n = 100, k = 25, \delta = 8$, and consider $p_2 \in \{\delta - 4, \ldots, \delta + 4\}$ and $\alpha \in \{0.0, 0.1, \ldots, 0.7\}$.

In the heat-map figure 5, we can observe how varying the setting of the second component (i.e., $p_2$) and $\alpha$ impacts the optimization time of (1+1) EA. We see that for $p_2$ small than $\delta$ quickly increases the optimization time, compared to when $p_2$ is set to values greater than $\delta$. Surprisingly mixing $p_1 = 1/n$ with $p_2 = (\delta + 3)/n$ seems to perform better than mixing $1/n$ with $\delta/n$. We think this can be explained by the fact that large $\delta$ increases the chances of flipping more than $\delta$ bits, compared to setting $p_2 = \delta/n$. This also tells us that we do not need to the exact gap size, and that we can allow some uncertainty in the setting of $p_2$. We expect this to be a strength for mutation operators especially for complex landscapes in which valleys are of different sizes.

To complete our analysis of the mixed binomial parameters, we plot the impact of the mixing coefficient $\alpha$ in figure 6 on the performance of the (1+1) EA. This figure indicates that lower $\alpha$ performs better, i.e., 0.4, 0.1 for $\delta = 4, 6$, respectively. For the case of $\delta = 8$ and $k = 25$, taking $\alpha$ between 0.0 and 0.1 performs better. This is expected for large gap sizes, as applying $\delta/n$ mutation rate more often would be faster than giving more trials for the hill-climbing component, also because, for the considered gap size, the time to jump dominates the time to hill climb.

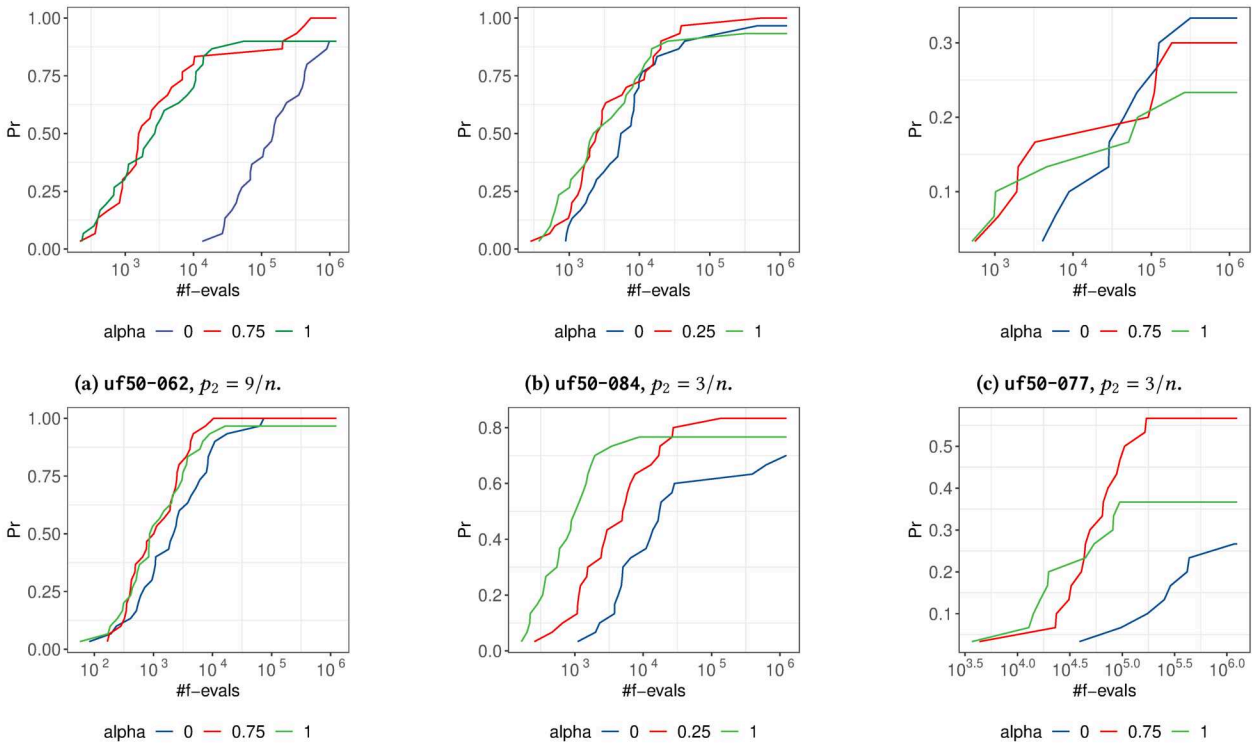
***Mixed binomial mutation vs. fast mutation.*** Here we compare the optimization time of (1+1) EA using mixed binomial mutation against fast mutation [2] on $\text{Jump}_{k,\delta}$. For this comparison, we fix $n = 100$ and consider $k = 20, 40$ and $\delta = 4, 10$. For the mixed binomial mutation, we mix $1/n$ with $\delta/n$ using a mixing coefficient $\alpha = 1/4$. For the fast mutation operator, we set its parameter $\beta = 1.5$ as this is the setting used in [2]. For each $k$, we plot in Figure 7 the empirical cumulative distribution of the success rates of the (1+1) EA using mixed binomial mutation and fast mutation over 100 independent runs. We note of course that the superior performance of mixed binomial mutation in this setting depends entirely on knowing the optimal rate beforehand.




**(a)** $k = 20, \delta = 4$ **(b)** $k = 40, \delta = 10$

**Figure 7: Empirical cumulative distribution function of (1+1) EA using mixed binomial (m-bin) vs. fast mutation (fmut). $n = 100$.**

***Mixed binomial mutation on MaxSat.*** The third benchmark function we consider is the MaxSat problem. We compare the (1+1) EA using mixed binomial mutation operators against the $\frac{1}{n}$ static mutation rate. We experimented with different mixing coefficients $\alpha$ and a mixed binomial distribution with $p_1 = 1/n$ and different $p_2 \in \{c/n \mid c \in \{3, 5, 7, 9\}\}$. Figure 8 reports the empirical cumulative distribution function (ECDF) for the considered instances. For each instance, the ECDFs are computed over 30 independent runs of each algorithm, with a maximum number of fitness evaluations $10n^3$ ($n$ here is the number of variables which is 50 for the considered instances.). For each instance we plot the configuration that performed the best. When $\alpha = 0, 1$ this means running (1+1) EA with a single binomial distribution with $p = p_2$, and $p = 1/n$ respectively. In figure 8, the first row plots the ECDFs for uniform random instance, and the power law instance are plotted on the second row. For the two first uniform instances (`uf50-062` and `uf50-084`), the (1+1) EA with mixed binomial distribution has a higher success rate (100%) compared to (1+1) EA using a static mutation rate. Notice that while allocating more computational budget for the (1+1) EA with mixed binomial mutation improves its success




**(a) uf50-062,** $p_2 = 9/n$. **(b) uf50-084,** $p_2 = 3/n$. **(c) uf50-077,** $p_2 = 3/n$.

**(d) pl50-150,** $p_2 = 5/n$. **(e) pl50-175,** $p_2 = 7/n$. **(f) pl50-165,** $p_2 = 7/n$.

**Figure 8: Empirical Cumulative Distribution Function of (1+1) EA with mixed binomial mutation on the considered MaxSat instances. For each instance we give the setting of $\alpha$ and $p_2$ that performed the best.**

rate until it reaches 100%, the (1+1) EA with a static rate, though its success rate increases with more computation budget, does not attain 100%. For the instance `uf50-084`, all three algorithms have a low success rate around 30%. For the power-law instance, we observe similar behavior on the first two instances. We note that the (1+1) EA was always successful on the first instance, and has a lower optimization time compared to the two other settings. For the second instance, the success rate is a little above 80%, while using a single mutation has a lower success rate. The third instance seems to be challenging, and mixed binomial mutation could only achieve a 53% success rate. Note that the $1/n$ mutation has less than 30% success rate. Overall, on the considered instances, the mixed binomial mutation does improve the optimization performance of (1+1) EA.

## 5 CONCLUSION

In this paper we have investigated a technique for designing mutation operators over binary strings comprised of a mixture of binomial distributions. This approach may be useful in settings where a collection of promising mutation strengths are already known due to domain knowledge or landscape probing.

On generalized jump functions, we proved that mixing the standard $1/n$ rate with the optimal rate can obtain a speed-up in some regions of the search space over the optimal static mutation rate alone. This is due to the fact that after the process has jumped over the gap, the strong mutation rate becomes detrimental to the remaining hill-climbing process. We also presented a number of experimental results that investigate different settings for which the mixed mutation rate may be beneficial. Our empirical results provide concrete running time comparisons for different benchmark parameters as well as a detailed investigation of the influence of the mixing coefficient on the success rate distribution for solving propositional satisfiability instances.

There are several directions for future work. First, studies on the mixtures of more than two distributions along with a sensitivity analysis of the mixing coefficients are needed. It would be also straightforward to apply mixed binomial mutation to more sophisticated evolutionary algorithms such as ones that use a population. Another important research direction would be to design other benchmarks that require significantly different mutation rates in different search space regions. This likely would require moving beyond simple functions of unitation in order to introduce more detailed landscape structures. Also using tuning methods to automate the search for mixtures of distributions would be an interesting question to investigate. Finally, the compelling effect we observed on maximum satisfiability problems suggests that further exploration of the approach on complicated combinatorial landscapes is crucial.

## ACKNOWLEDGMENTS

A.M. Sutton was supported by NSF grant 2144080.

## REFERENCES

[1] A. Aleti and I. Moser. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys (CSUR)*, 49(3):1–35, 2016.

[2] H. Bambury, A. Bultel, and B. Doerr. Generalized jump functions. In F. Chicano and K. Krawiec, editors, *GECCO '21: Genetic and Evolutionary Computation Conference, Lille, France, July 10-14, 2021*, pages 1124–1132. ACM, 2021.

[3] T. Bläsius, T. Friedrich, and A. M. Sutton. On the empirical time complexity of scale-free 3-sat at the phase transition. In T. Vojnar and L. Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 2019.

[4] D. Dang, T. Friedrich, T. Kötzing, M. S. Krejca, P. K. Lehre, P. S. Oliveto, D. Sudholt, and A. M. Sutton. Escaping local optima using crossover with emergent diversity. *IEEE Trans. Evol. Comput.*, 22(3):484–497, 2018.

[5] B. Doerr, D. Johannsen, and C. Winzen. Multiplicative drift analysis. *Algorithmica*, 64:673–697, 2012.

[6] B. Doerr, H. P. Le, R. Makhmara, and T. D. Nguyen. Fast genetic algorithms. In P. A. N. Bosman, editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*, pages 777–784. ACM, 2017.

[7] B. Doerr and A. Rajabi. Stagnation detection meets fast mutation. *Theor. Comput. Sci.*, 946:113670, 2023.

[8] C. Doerr, D. A. Janett, and J. Lengler. Tight runtime bounds for static unary unbiased evolutionary algorithms on linear functions. In S. Silva and L. Paquete, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*, pages 1565–1574. ACM, 2023.

[9] C. Doerr and M. Wagner. Simple on-the-fly parameter selection mechanisms for two classical discrete black-box optimization benchmark problems. In H. E. Aguirre and K. Takadama, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*, pages 943–950. ACM, 2018.

[10] T. Friedrich, F. Quinzan, and M. Wagner. Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 293–300, 2018.

[11] H. H. Hoos and T. Stützle. Satlib: An online resource for research on sat. *Sat*, 2000:283–292, 2000.

[12] G. Karafotias, M. Hoogendoorn, and Á. E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187, 2014.

[13] P. K. Lehre and X. Qin. Self-adaptation can improve the noise-tolerance of evolutionary algorithms. In *Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, pages 105–116, 2023.

[14] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[15] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. *Handbook of metaheuristics*, pages 129–168, 2019.

[16] A. Rajabi and C. Witt. Self-adjusting evolutionary algorithms for multimodal optimization. In C. A. C. Coello, editor, *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*, pages 1314–1322. ACM, 2020.

[17] A. Rajabi and C. Witt. Stagnation detection in highly multimodal fitness landscapes. In F. Chicano and K. Krawiec, editors, *GECCO '21: Genetic and Evolutionary Computation Conference, Lille, France, July 10-14, 2021*, pages 1178–1186. ACM, 2021.

[18] D. Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17(3):418–435, 2013.

[19] C. Witt. How majority-vote crossover and estimation-of-distribution algorithms cope with fitness valleys. *Theor. Comput. Sci.*, 940(Part):18–42, 2023.