Escaping Large Deceptive Basins of Attraction with Heavy-Tailed Mutation Operators

Tobias Friedrich
Hasso Plattner Institute
Potsdam, Germany

Francesco Quinzan
Hasso Plattner Institute
Potsdam, Germany

Markus Wagner
University of Adelaide
Adelaide, Australia

ABSTRACT

In many evolutionary algorithms (EAs), a parameter that needs to be tuned is that of the mutation rate, which determines the probability for each decision variable to be mutated. Typically, this rate is set to $1/n$ for the duration of the optimization, where n is the number of decision variables. This setting has the appeal that the expected number of mutated variables per iteration is one.

In a recent theoretical study, it was proposed to sample the number of mutated variables from a power-law distribution. This results in a significantly higher probability on larger numbers of mutations, so that escaping local optima becomes more probable.

In this paper, we propose another class of non-uniform mutation rates. We study the benefits of this operator in terms of average-case black-box complexity analysis and experimental comparison. We consider both pseudo-Boolean artificial landscapes and combinatorial problems (the Minimum Vertex Cover and the Maximum Cut).

We observe that our non-uniform mutation rates significantly outperform the standard choices, when dealing with landscapes that exhibit large deceptive basins of attraction.

CCS CONCEPTS

• **Mathematics of computing** → **Combinatorial optimization**; • **Theory of computation** → *Theory of randomized search heuristics*;

KEYWORDS

Heavy-tailed Mutation, Combinatorial Optimization, Single-objective Optimization

ACM Reference Format:
Tobias Friedrich, Francesco Quinzan, and Markus Wagner. 2018. Escaping Large Deceptive Basins of Attraction with Heavy-Tailed Mutation Operators. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205515

1 INTRODUCTION

One of the building blocks of many evolutionary algorithms and bio-inspired search heuristics is the *mutation operator*, that has the potential to randomly introduce diversity in each generation. This operator have been studied trough the years, both from a theoretical and experimental point of view. Sometimes, the mutation is defined in terms of a fixed probability distribution over the set of decision variables; sometimes in terms of dynamic schedules and self-adaptive schemes (cf. [8, 9]).

Particularly studied in the theory community is the static *uniform mutation*. This operator consists of mutating each decision variable in the input string independently, with fixed probability p — the *mutation rate*. The parameter p is given at the beginning of the optimization process and never changed afterwards. Typically, the mutation rate is set as $p = 1/n$, with n the number of decision variables. The advantage of this default setting is that the expected number of mutated variables per iteration is 1.

However, this choice is not necessarily optimal, and wrong tuning can lead to substantial slow-down. In fact, Doerr et al. [4] prove that in some cases a constant-factor change of the mutation rate results in a larger variation of the overall run time.

Recently, Doerr et al. [5] proposed a mutation rate of the form α/n, with $\alpha \leq 1/2$ drawn from a power-law distribution in each iteration. The authors analyze the optimization time of their "fast genetic algorithm" on $\mathsf{Jump}(m, n)$, and prove that the resulting setting is indeed significantly faster compared to the standard choice.

The idea of using non-uniform mutation probabilities is not new in the literature. Jansen et al. [17] propose a mutation rate that at time step t flips each bit independently with probability (w.p.) $2^{(t-1) \mod (\lceil log_2 n \rceil - 1)}/n$. Doerr et al. [5] notice that this mutation rate is equivalent to a mutation rate of the form α/n, with α drawn uniformly at random u.a.r.) over the powers of two described above.

The benefit of non-standard mutation probabilities has also been noted in connection with the study of the $\mathsf{LeadingOnes}$ function (cf. Böttcher et al. [3]), and in connection with the analysis of the $(1 + \lambda)$EA (cf. Gießen and Witt [14]).

In this study, we propose a new non-standard mutation operator, and compare it against uniform and power-law mutation rates. We study both artificial landscapes and combinatorial optimization problems. In some cases, we use an automated algorithm configurator to search for the *optimal* mutation, and compare the output with the proposed operator. With this combination of empirical observation and theoretical analysis we conclude that our non-uniform mutation rate is particularly well-suited to optimize fitness functions that exhibit local optima with large deceptive basins of attraction.

Algorithm 1: General framework for the (1+1) EA

```
Choose initial solution $x \in \{0,1\}^n$ u.a.r.;
while convergence criterion not met do
    $y \leftarrow$ Mutation($x$) for given mutation operator;
    if $f(y) \geq f(x)$ then
        $x \leftarrow y$;
return x;
```

This paper is structured as follows. In Section 2 we present a general framework for the (1+1) EA, as well as the hereby considered mutation rates. In Section 3 we give the theoretical analysis on some artificial pseudo-Boolean landscapes. We then study the benefit of using heavy tails in connection with the Minimum Vertex Cover problem and the Maximum Cut (cf. Section 4). We conclude with an experimental comparison in Section 5.

2 ALGORITHMS AND SETTING

In this paper, we look at the run time of the simple (1+1) EA, with various mutation rates. This algorithm requires as input an individual of fixed length. An offspring is generated with an operator that resembles asexual reproduction — the *mutation* operator. The fitness is then computed, and the less desirable result is discarded. Note that this algorithm is *elitist*, in that the solution quality never decreases throughout the process. Pseudo-code for this method is given in Algorithm 1. The (1+1) EA is one of the simplest examples of an *evolutionary* algorithm, and it has been theoretically extensively studied (cf. Droste et al. [6]). A straightforward generalization of it is the $(\mu + 1)$EA, that applies the operators discussed above to a population of objectives of size μ. The simple (1+1) EA has the advantage that many results related to its run time extend to more complex heuristics.

The standard choice for the mutation operator Mutation(−) is to flip each bit independently w.p. (with probability) $1/n$. In a slightly more general setting, we let each bit flip w.p. p for some $0 < p < 1/2$. We refer to this class of mutation operators as *uniform* mutations. We also consider the *power-law* mutation rate fmut$_\beta$, for a user-defined parameter $\beta > 1$ (cf. Doerr et al. [5]). Intuitively, this kind of mutation is such that the probability of performing a k-bit flip in one iteration is roughly $\sim k^{-\beta}$. More formally, we say that a random variable X follows the discrete power-law distribution $D^{\beta}_{n/2}$ if it holds $\Pr[X = k] = k^{-\beta}/H^{\beta}_{n/2}$, where we have used the notation

$$H^{\beta}_{\ell} := \sum_{j=1}^{\ell} \frac{1}{j^{\beta}}.$$

The H^{β}_{ℓ} are known in the literature as generalized harmonic numbers. Interestingly, generalized harmonic numbers can be approximated with the Riemann Zeta function as

$$\lim_{\ell \to +\infty} H^{\beta}_{\ell} = \zeta(\beta),$$

with $\zeta(\alpha)$ the Riemann Zeta function. In particular, harmonic numbers $H^{\beta}_{n/2}$ are always upper-bounded by a constant, for increasing problem size and for a fixed $\beta > 1$. Pseudo-code for this kind of mutation operator is given in Algorithm 2. We propose a new mutation operator, which we refer to as cMut$_p$, as given in Algorithm 3. This mutation operator allows for any kind of k bit-flip, for $k = 0, \ldots, n$ with n the problem size. It performs a single bit flip w.p. p, and a k -bit flip w.p. $(1-p)/(n-1)$. cMut$_p$ is similar to the power-law mutation, although the corresponding probability mass does not yield exponential decay.

Algorithm 2: The mutation operator fmut$_\beta(x)$

```
$y \leftarrow x$;
choose $k \in [1, \ldots, n/2]$ according to distribution $D^{\beta}_{n/2}$;
for $j = 1, \ldots, n$ do
    if random([0, 1])$n \leq k$ then
        $y[j] \leftarrow 1 - y[j]$;
return $y$;
```

Algorithm 3: The mutation operator cMut$_p(x)$

```
$y \leftarrow x, k \leftarrow 1$;
if random([0, 1]) $> p$ then
    choose $k \in \{2, \ldots, n\}$ u.a.r.;
flip $k$-bits of $y$ chosen u.a.r.;
return $y$;
```

3 ARTIFICIAL LANDSCAPES

3.1 The OneMax and TwoMax Functions

We analyze the run time of the (1+1) EA on the OneMax, defined as $\text{OneMax}(x_1, \ldots, x_n) = |x|_1 = \sum_{j=1}^{n} x_j$. This simple linear function of unitation returns the number of ones in a pseudo-Boolean input string. It is well-known that the (1+1) EA with uniform and fmut$_\beta$ mutations finds the global optimum after $O(n \log n)$ fitness evaluations (cf. Mühlenbein [18]). It can be easily shown that the (1+1) EA with mutation cMut$_p$ also achieves similar performance on this instance. More formally,

Lemma 3.1. *The (1+1) EA with mutation cMut$_p$ finds the global optimum of the* OneMax *after expected* $O\left(\frac{n}{p} \log n\right)$ *fitness evaluations, for any constant* $0 < p < 1$.

The lemma above can be proven with the observation that the (1+1) EA with mutation cMut$_p$ perform single bit-flips w.p. at least p/n, together the fitness level method (see Wegener [20]). This method consists partitioning the set of f-values into nonempty levels, and looking at the expected time a jump from one level to the other is performed.

We also analyze the run time of the (1+1) EA with cMut$_p$ mutation on the following fitness function

$$\text{TwoMax}(x_1, \ldots, x_n) = \max\left\{|x|_1, n - |x|_1\right\} + \prod_{i=1}^{n} x_i,$$

Among all search points with more than $\frac{n}{2}$ 1-bits, this function increases with the number of ones. Among all search points with

less than $\frac{n}{2}$ 1-bits, it increases with the number of zeros.
The TwoMax has two branches and it is symmetric w.r.t. the underlying hypercube. The point 0^n is a *local* optimum, while the point 1^n is a *global* optimum. The leftmost branch is a basin of attraction for the local optimum. This function was first investigated in the context of Genetic Algorithms by Pelikan and Goldberg [19] and Hoyweghen et al. [15]). It is well-known that the (1+1) EA using uniform mutation has expected run time $\Omega(n^n)$ (cf. Friedrich et al. [13]). The following lemma gives an estimate of the expected run time for the (1+1) EA using mutation fmut_β.

Lemma 3.2. *The (1+1) EA using the* fmut_β *mutation finds the global maximum of* TwoMax *after expected* $2^{\Omega(n)}$ *fitness evaluations, for any constant* $\beta > 1$.

This result is intuitively motivated by the fact that 0^n is a deceptive attractor: if the algorithm reaches 0^n, then the probability of hitting the global optimum afterwards is very low, both for the (1+1) EA using uniform mutation, and the (1+1) EA following mutation fmut_β. We show that $(1+1)$ *EA* with mutation cMut_p yields polynomial run time on this instance. Useful for the analysis is the following lemma.

Lemma 3.3. *The (1+1) EA with mutation cMut$_p$ reaches a* local *optimum of the* TwoMax *after expected* $O\left(\frac{n}{p}\log n\right)$ *fitness evaluations, for any constant* $0 < p < 1$.

Again, this lemma can be proven using the fitness level method, together with the observation that the $(1+1)$ *EA* with mutation cMut_p performs single bit-flips w.p. at least p/n. We can readily use the lemma above to prove an upper-bound on the run time of the $(1+1)$ *EA* with mutation cMut_p on the instance TwoMax in polynomial time. More formally,

Theorem 3.4. *The (1+1) EA with mutation cMut$_p$ finds the global maximum of the* TwoMax *after expected* $O\left(\frac{n}{p}\log n + \frac{n}{1-p}\right)$ *fitness evaluations, for any constant* $0 < p < 1$.

3.2 The Jump function

We consider the problem of maximizing the n-dimensional *jump* function, first introduced by Droste et al. [7]. This function is defined as

$$\text{Jump}(m,n)(x) = \begin{cases} m + |x|_1 & \text{if } |x|_1 \le n-m \text{ or } |x|_1 = n; \\ n - |x|_1 & \text{otherwise;} \end{cases}$$

This function depends on the parameters n and m. The first parameter denotes the problem size, while the second parameter denotes the size of the jump. For $1 < m < n$ this function exhibits a single *local* maximum and a single *global* maximum. The jump function $\text{Jump}(m,n)$ with $n = 50$ and $m = 20$ is displayed in Figure 1. It is well-known that the (1+1) EA with uniform mutation has expected run time $\Theta(n^m + n\log n)$ (cf. Droste et al. [7]). Interestingly, Doerr et al. [5] derive detailed upper-bounds for the run time $(1+1)$ *EA* using any kind of uniform mutation, as well as the fmut_β on any jump function with $m \le n/2$. We first present a general upper-bound on the run time of the (1+1) EA with mutation operator cMut_p. We then perform a theoretical comparison of the cMut_p and fmut_β on the $\text{Jump}(m,n)$, depending on how the parameter m evolves for increasing problem size. The following lemma holds.

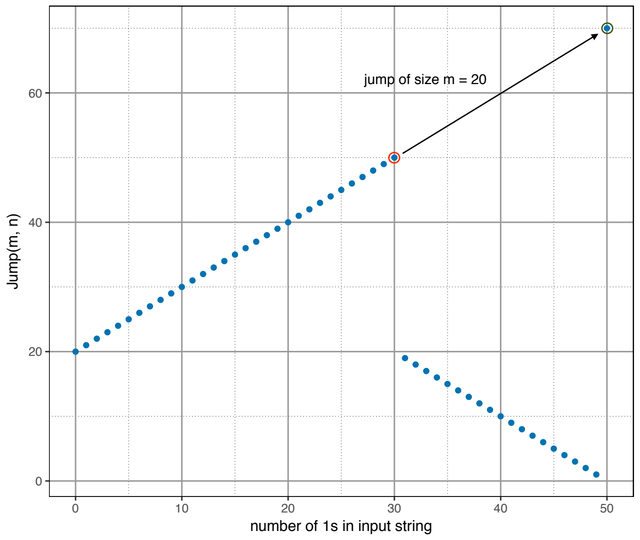

Figure 1: The jump function $\text{Jump}(m,n)$ **with** $n = 50$ **and** $m = 20$**. The local maximum is highlighted in red, and the global maximum in green. We observe that if an algorithm reaches the local optimum, a** 20**-bit flip operation is necessary in order to reach a solution with higher fitness.**

Lemma 3.5. *Consider a jump function* $f = \text{Jump}(m,n)$ *for* $m = \Theta(1)$. *Denote with* $T_p(f)$ *the run time of the (1+1) EA using the mutation cMut$_p$ on the function* f*, and for all* $p < 1$ *constant. Then it holds*

$$T_p(f) \le \binom{n}{m}\frac{n-1}{1-p} + \frac{2}{p}\log\frac{n}{m}.$$

In the remaining part of this section, we compare the (1+1) EA using mutation fmut_β with the (1+1) EA using mutation cMut_p, for evolving jump size. We first show that on $\text{Jump}(m,n)$ functions, with m constant for increasing problem size, the fmut_β outperforms the cMut_p by a factor of $\Theta(n)$. We then identify boundaries on the parameter m s.t. the (1+1) EA with mutation cMut_p at least outperforms the mutation fmut_β, up to a multiplicative constant. We conclude with the observation that for larger jumps the (1+1) EA with mutation fmut_β outperforms the fmut_β. The following lemma holds.

Lemma 3.6. *Consider a jump function* $f = \text{Jump}(m,n)$*, with* m *constant for increasing problem size. Let* $T_\beta(f)$ *be the run time of the (1+1) EA on* f *using the mutation operator* fmut_β*, and denote with* $T_p(f)$ *the run time of the (1+1) EA on* f *using the mutation operator cMut$_p$. Then it holds* $T_p(f) = \Theta(nT_\beta(f))$*, for* $0 < p < 1$ *and* $\beta > 1$ *constant.*

The lemma above intuitively shows that the mutation cMut_p is a linear factor worse than the mutation fmut_β, on jump functions with m constant for increasing problem size. We now prove an intermediate result, to show that on some jump functions $\text{Jump}(m,n)$ with $m \le n/2$ non-constant, the (1+1) EA with mutation operator cMut_p has run time as least as good as the run time fmut_β, up to a multiplicative constant.

Lemma 3.7. *Let $f = \text{Jump}(m, n)$ be a jump function. Let $T_\beta(f)$ be the run time of the (1+1) EA on f using the mutation operator* fmut_β. *Similarly, denote with $T_p(f)$ the run time of the (1+1) EA on f using the mutation operator cMut$_p$. Then it holds*

$$T_p(f) \leq \frac{c}{H^{(\beta)}_{n/2}} T_\beta(f)$$

with c a constant independent of m, n, β and p, for all functions $f =$ $\text{Jump}(m, n)$ *with $\sqrt[\beta]{(n-1)/(1-p)} \leq m \leq n/2$.*

We conclude by showing that if $n - m$ is constant for increasing problem size, then the cMut$_p$ significantly outperforms the fmut$_\beta$.

Lemma 3.8. *Consider a jump function $f = \text{Jump}(m, n)$, and suppose that $n - m$ is constant for increasing problem size. Let $T_\beta(f)$ be the run time of the (1+1) EA on f using the mutation operator* fmut_β, *and denote with $T_p(f)$ the run time of the (1+1) EA on f using the mutation operator cMut$_p$. Then it holds $T_\beta(f) = 2^{\Omega(n)}$ and $T_p(f) = n^{\Theta(1)}$.*

Lemma 3.6, Lemma 3.7 and Lemma 3.8 suggest that for each problem size n, there exists a *sweet-spot* $0 < m^* < n$ s.t. the cMut$_p$ outperforms the fmut$_\beta$ on all functions $\text{Jump}(m, n)$ with $m \geq m^*$.

3.3 Experimental Comparison on the Jump

We complement the theoretical results above by locating experimentally the minimal size of the smallest m^* s.t. the (1+1) EA with mutation cMut$_p$ outperforms the (1+1) EA with mutation fmut$_\beta$, on all functions $\text{Jump}(m, n)$ with $m \geq m^*$. We let the (1+1) EA using different mutation rates run on the function $\text{Jump}(m, n)$ and take the sample mean of 100 runs, for various choices of n, and for all $2 \leq m \leq n - 1$. We let the algorithm run until the *local* optimum is reached. We then artificially compute the expected run time to perform the appropriate jump in order reach the global optimum, and add it to the previously found run time. The results are displayed in Figure 2. We observe that in the worst case, fmut$_\beta$ with $\beta = 1.5$ vs. cMut$_p$ with $p = 0.1$, the sweet-spot m^* exhibits linear growth in the problem size. In all other cases, the value m^* is significantly lower. Note that in the case of uniform mutation vs. cMut$_p$ it holds $m^* \leq 3$ for jumps with problem size between 100 and 1000.

4 COMBINATORIAL OPTIMIZATION

4.1 The Minimum Vertex Cover

In this section, we study the *Minimum Vertex Cover* problem (MVC): Given a graph $G = (V, E)$ of order n find a minimal subset $U \subseteq V$ s.t. each edge in E is adjacent to at least one vertex. For a given indexing on the vertices of G, each subset $U \subseteq V$ is represented as a pseudo-boolean array $(x_1, \ldots, x_n)$ with $x_i = 1$ iff. the i-th vertex is in U. Thus, in this context the problem size is the order of the graph. We approach the MVC by minimizing the functions $(u(x), |x|_1)$ in lexicographical order, with $u(x)$ the function that returns the number of uncovered edges. We restrict the analysis on complete bipartite graphs, defined as follows.

Definition 4.1. We say that a graph $G = (V, E)$ is *complete bipartite* if there exists a partition $\{V_1, V_2\}$ of V such that V_1 and V_2 are independent sets, and every pair of vertices $(u, v) \in V_1 \times V_2$ is adjacent. We denote any such graph as $G = (V_1, V_2, E)$.

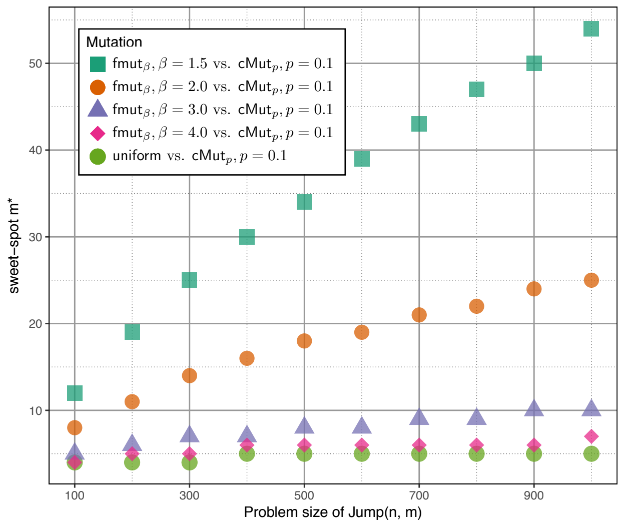

Figure 2: We consider the run time of the (1+1) EA with uniform mutation, fmut$_\beta$ **with** $\beta = 1.5, 2, 3, 4$ **and cMut$_p$ with** $p = 0.1$. **For each function** $\text{Jump}(m, n)$ **with** $n \in [100, \ldots, 1000]$ **and** $2 \leq m \leq n-1$, **we let each algorithm run for** 100 **times and compute the sample mean. We then display the first** m^* **s.t. the mutation cMut$_p$ outperforms the** fmut$_\beta$ **or the uniform mutation. We observe that for increasing problem size, the sweet-spot** m^* **exhibits a slow linear increase. Note that in the case of uniform mutation it holds** $m^* \leq 3$ **for the hereby considered problem size.**

It is well-known that on complete bipartite graphs the $(1 + 1)$ *EA* using uniform mutation performs poorly. More specifically, consider any such graph $G = (V_1, V_2, E)$ with partitions of size $|V_1| = m$ and $|V_2| = n - m$ for a constant $0 < m < n/2$. Then the expected run time of the $(1 + 1)$ *EA* with uniform mutation to search for a minimum vertex cover on this instance is at least $\Omega\left(mn^{m-1} + n \log n\right)$. For $m \leq n/3$ the $(1 + 1)$ *EA* using mutation fmut$_\beta$ finds the global optimum after expected $O\left(H^\beta_{n/2} n^\beta 2^m\right)$, and for $m \geq n/3$ the upper-bound is $O\left(H^\beta_{n/2} n^\beta 2^n\right)$. For a discussion on these run time bounds see Doerr et al. [5] and Friedrich et al. [11]. We have that the following theorem holds.

Theorem 4.2. *Let $G = (V_1, V_2, E)$ be a complete bipartite graph. Then the run time of the (1+1) EA using mutation cMut(p) finds a solution to the MVC after expected $O\left(\frac{n}{p} \log n + \frac{n}{1-p}\right)$ fitness evaluations.*

Proof. We assume that $|V_1| \leq |V_2|$. We divide the run time analysis in three phases. In *Phase* 1 the algorithm finds a (non-minimum) vertex cover; in *Phase* 2 the (1+1) EA finds a minimum (non optimal) vertex cover, given that a cover has been reached; in *Phase* 3 the algorithm finds the smallest vertex cover, given that a minimum cover has been reached.

(Phase 1*)* Following the same argument as given in Theorem 1 in Friedrich et al. [11], it can be shown with the multiplicative increase method that the expected number of single bit-flips to obtain a vertex cover is $O(n \log n)$. Their proof is based on the observation that by adding a single chosen bit-flip to the solution yields a fitness decrease of $(1 - 1/n)u(x)$, with $u(x)$ the number of uncovered edges in the current solution. Since the probability of performing a single bit-flip is p, then the expected optimization time to perform a single bit-flip is $1/p$. It follows that after an initial phase of length $O(n/p \log n)$ the (1+1) EA with mutation operator cMut_p finds a vertex cover.

(Phase 2*)* Once the $(1+1)$ *EA* finds a vertex cover, then all unnecessary nodes are removed from the solution, until a minimum (not necessarily optimal) vertex cover is reached. In this phase, no bit-flip that removes points in V_1 *and* V_2 is accepted, since the resulting set is no longer a cover. To estimate the run time in this phase, we simply count the 1-bit flips necessary to obtain a Minimum Vertex Cover. Our analysis only counts the number of single bit-flips, as larger jumps only result in faster convergence. Every solution reached after (Phase 1) contains either V_1 or V_2, or both. Since $|V_2| \geq |V_1|$, we give an upper-bound on the run time until all bit-flips of V_2 are removed. The probability of selecting a single chosen bit-flip for mutation is at least p/n, and the expected run time of such event to occur is at most n/p. The number of steps necessary to remove all unnecessary nodes with single bit-flips is

$$\sum_{i=1}^{|V_2|} \frac{ni}{p} = O\left(\frac{n}{p} \log n\right)$$

where we have used $|V_2| \leq n$.

(Phase 3*)* If the solution reached after Phase 2 is optimal, then there is nothing to prove. Otherwise, it is possible to reach the optimal solution by flipping all bits in one iteration. This event occurs w.p. at least $(1-p)/(n-1)$. The thesis follows. □

Note that for $p = \Theta(1)$ from Theorem 4.2 yields an expected run time of $O(n \log n)$ fitness evaluations.

4.2 The Maximum Cut

In this section we consider the following problem: Given a (directed) graph $G = (V, E)$ find a subset of vertices $U \subseteq V$ s.t. the sum of the weights edges leaving U is maximal. This problem is known as the *Maximum Cut*, and we denote it with maxCut. It is well-known that in the case of an undirected graph G with positively weighted edges one can find a $1/(3(1+\epsilon))$-approximation in time $O\left(\frac{1}{\epsilon}n^7 \log n\right)$ with a multi-objective EA (cf. Friedrich and Neumann [12]). In this section, we study the maxCut on directed unweighted graphs. Note that in this case, the maxCut consists of finding a subset of vertices $U \subseteq V$ s.t. the sum of the edges leaving U is maximal. We perform an experimental study on a class of instances G with directed weighted edges in Section 5. As in the case of the MVC, for a given graph $G = (V, E)$ of order n, fix an indexing on the vertices. Then any subset $U \subseteq V$ is represented as a pseudo-boolean array $(x_1, \ldots, x_n)$, with $x_i = 1$ iff. the i-th vertex is in U. The maxCut problem can be approached by maximizing the flow function, as defined below.

Definition 4.3. Let $G = (V, E)$ be a (directed) graph with unweighted edges. For each subset $U \subseteq V$ consider the set $\Delta(U) := \{(e_1, e_2) \in E : e_1 \in U \text{ and } e_2 \notin U\}$. We define the flow function $f : 2^V \longrightarrow \mathbb{R}_{\geq 0}$ as

$$f(U) := |\Delta(U)|.$$

It is well-known that for any (directed) graph $G = (V, E)$, flow functions f are always *submodular* (cf. Feige et al. [10] and Friedrich and Neumann [12]). In fact, it can be shown that for any two subsets $U, W \subseteq V$ it holds $f(U \cup W) + f(U \cap W) \leq f(U) + f(W)$. Using this lemma it can be easily proven that f is sub-additive, and the following lemma holds.

Lemma 4.4. *For any directed graph* $G = (V, E)$*, denote with* f *the corresponding flow function, and let* $\Delta := \max_{v \in V} |\Delta(v)|$ *its outer node degree. Then it holds* $f(U) \leq \Delta n$*, for all subsets* $U \subseteq V$*.*

We omit a formal proof of Lemma 4.4, simply follow from the simple observation that any submodular function is *sub-additive*. This fact is widely acknowledged in the related literature. In its general form the maxCut is NP-complete. However, it can be approached by searching for approximations of *local* maxima of the flow function, as defined below.

Definition 4.5. Let $G = (V, E)$ be a graph with positively weighted edges, and denote with f its flow function. A local maximum to the maxCut is any subset $S \subseteq V$ s.t. $f(S) \geq f(S \setminus \{u\})$ for all $u \in S$, and $f(S) \geq f(S \cup \{v\})$ for all $v \in V \setminus S$.

In other words, local maxima are optimal solutions w.r.t. f up to single bit-flips. When dealing with flow functions f it can be proven that either local maxima or their complement always yield a good approximation of the global maximum. The following theorem holds.

Theorem 4.6 (Theorem 3.4 in Feige et al. [10]). *Consider a graph* $G = (V, E)$ *and let* f *be the corresponding flow function. Let* S *be a* $(1 + \epsilon/n^2)$*-approximation of a local maximum of* f*. Then either* S *or* $V \setminus S$ *is a* $(1/3 - \epsilon/n)$*-approximation of the global optimum.*

We remark that in Feige et al. [10] the theorem above is given for any submodular function. Also, the statement as we present it is implicit in the proof of their Theorem 3.4. We can use the Theorem 4.6 to estimate the run time of the (1+1) EA using mutation cMut_p to maximize a given flow function. Intuitively, it is always possible to find a $(1 + \epsilon/n^2)$-approximation of a local optimum in polynomial time using single bit-flips. It is then possible to compare the approximate local solution S with its complement $V \setminus S$, by flipping all bits in one iteration, which occurs w.p. $(1-p)/(n-1)$. We have that the following theorem holds.

Theorem 4.7. *On any directed graph* $G = (V, E)$ *with unweighted edges, the (1+1) EA with mutation* cMut_p *is a* $(1/3 - \epsilon/n)$*-approximation algorithm for the* maxCut*, for all* $0 < p < 1$*. Its expected optimization time is* $O\left(\frac{1}{\epsilon}\frac{n^3}{p}\log(n\Delta) + \frac{n}{1-p}\right)$*, with* Δ *the maximum outer degree.*

Proof. We divide the proof in two phases. In an initial phase, the algorithm finds a $(1 + \epsilon/n^2)$-approximation of a local optimum, and in a second phase it finds a $(1/3 - \epsilon/n)$-approximation using the heavy-tailed mutation.

(Phase 1*)* We estimate the run time in the first part of the statement

using the multiplicative increase method. We first observe that if the initial solution x_0 has fitness $f(x_0) = 0$, then it is always possible to obtain a new solution with f-value strictly greater than 0, by adding or removing a single chosen bit-flip. This event occurs w.p. at least p/n, and the expected run time of this event to occur is $O(n/p)$. Thus, after an initial phase linear in the problem size, we can assume that the solution has f-value of at least 1 - which is a lower-bound on all positive values of the flow function. For any solution x_t found at time step t, it is always possible to make an improvement of $(1 + \epsilon/n^2)f(x_t)$ by adding or removing a single vertex, unless x_t is already the desired approximation of a local maximum. Using the multiplicative increase method, we find the minimum number of steps k in order to reach the desired approximation, by solving the following equation

$$\left(1 + \frac{\epsilon}{n^2}\right)^k \leq \Delta n$$

where we have used the upper-bound on the f-values given in Lemma 4.4. It follows that the algorithm reaches a $(1 + \epsilon/n^2)$-approximation of a local maximum after expected $\frac{1}{\epsilon}O\left(n^2 \log(n\Delta)\right)$. Since the probability of performing a single chosen bit-flip is at least p/n, then we can upper-bound the run time in this initial phase as $\frac{1}{\epsilon}O\left(n^3/p \log(n\Delta)\right)$.
(Phase 2) Assume that a desired approximation of a local optimum S has been found. Then from Theorem 4.6 it follows that either this solution or its complement is a desired approximation of the global optimum. Thus, if in (Phase 1) the desired solution has not been fund, than a n-bit flip is sufficient to find the optimal solution. The probability of performing an n-bit flip is $(1 - p)/(n - 1)$, and the run time in this phase is upper-bounded as $O(n/(1 - p))$. □

Note that if we set p to be constant for increasing problem size, then it follows that the (1+1) EA with mutation cMut_p finds a $(1/3 - \epsilon/n)$-approximation of the maxCut, within expected $O\left(\frac{1}{\epsilon}n^3 \log(n\Delta)\right)$ fitness evaluations, on all directed graphs $G = (V, E)$ with unweighted edges.

5 EXPERIMENTS

5.1 Evolving the Probability Distribution

In this first set of experiments, we consider the following setting: given a bipartite graph $G = (V_1, V_2, E)$ with randomly directed edges[1], find the optimal mutation rate for the (1+1) EA (cf. Algorithm 1) to solve the maxCut. In this setting, the probability distribution of the mutation is a parameter of the algorithm: it is given once initially and then it remains fixed. We employ so-called "automated algorithm configuration" (AAC) in order to evolve it to best performance. We remark that an alternative to this approach would be to change the distribution dynamically, either with or without feedback from the search. We leave it as a future goal to bring together full self-adaptiveness of non-uniform mutation probabilities. For now, our goal is to further investigate the benefits of non-uniform mutation probabilities.

In recent years, a number of AAC methods have been developed. General purpose approaches include SMAC [16], GGA [1]), and the iterated f-race procedure called irace [2]. The aim of these methods is to allow a wide range of parameters to be efficiently tested in a systematic way. For example, irace's procedure begins with a large set of possible configurations, and tests these on a succession of given problem instances. As soon as there is sufficiently strong statistical evidence that a particular setting is sub-optimal, it is removed from consideration (the particular statistical test used in the f-race is the rank-based Friedman test). In practice, a large number of settings will typically be eliminated after just a few iterations, making this a relatively efficient process. irace allows for a fair comparison between different methods, by allocating to each one an evaluation or time budget.

```
# ACDT: Elite configurations (first number is the configuration ID):
          p1      p2      p3      p4      p5      p6      p7      p8      p9     p10
5599    0.70    0.03    0.03    0.02    0.02    0.02    0.04    0.04    0.02    0.06
8176    0.69    0.07    0.04    0.02    0.01    0.01    0.02    0.07    0.02    0.06
6578    0.70    0.02    0.02    0.02    0.04    0.04    0.06    0.01    0.02    0.07
8991    0.71    0.04    0.03    0.01    0.06    0.04    0.02    0.02    0.01    0.05
9143    0.75    0.02    0.00    0.01    0.02    0.00    0.04    0.04    0.03    0.08
```

Table 1: irace results of optimising the probability distribution for mutations on $\mathsf{Jump}(m, n)$ with $n = 10$ and $m = 3$. The upper block lists the outcomes of running each configuration 10,000 times. We observe that on the best runs we get 70% probability on the 1 bit-flips, and then the remaining probability is roughly distributed evenly among the $p2, \ldots, p10$.

For our experiments, we use irace 2.3[2]. Our (1+1) EA is implemented in Java, and all code and results are available at *https://cs.adelaide.edu.au/ optlog/research/combinatorial.php*. We give irace a budget of 100,000 (1+1) EA runs. The (1+1) EA performance is measured in the number of iterations it takes to find the global optimum, capped at 1000 iterations per run.[3] In the final testing phase, the best configurations (as determined by irace) are run 10,000 times to achieve stable average performance values with a standard error of the mean of 1%.

In Table 1 we show the final output of irace when optimizing the mutation rate distribution. As one would expect, the distribution quickly evolves to one where the largest probability mass is on "perform one bit-flip (p1)", and the remaining probability mass is distributed approximately evenly among the remaining probabilities (p2) . . . (p10). Note that the general intuition behind the mutation cMut_p is in line with the output of this set of experiments, for the hereby considered parameters. This set of experiments is performed on Lenovo compute cluster with Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz and 128GB RAM.

5.2 Run Time Comparison

We further compare experimentally the run time of the (1+1) EA using uniform mutation, fmut_β and cMut_p on directed graphs with positively weighted edges. We consider the following class of instances.

[1]We consider a complete bipartite graph $G = (V, E)$ of order $n = 100$, and s.t. $|V_1| = |V_2|$; each edge has probability 1/2 of being directed from V_1 to V_2, and probability 1/2 to be directed from V_2 to V_1.

[2]The irace Package, http://iridia.ulb.ac.be/irace, last accessed on 9 November 2017.

[3]*Implementation Note.* Due to the time overhead in starting a Java VM, we actually run (1+1) EA 100 times (resulting in a less noisy performance landscape) and then return the average of these 100 runs. For irace, these 100 repetitions are hidden, and it calls this (1+1) EA 100,000 times during its iterated f-race.

Definition 5.1. Consider a complete bipartite graph $G = (V_1, V_2, E)$ of even order n s.t. $|V_1| = |V_2|$, and with bi-directed edges. For any two integers $a, b > 0$ we set the weight of all edges from V_1 to V_2 equal to a, and all weights on the edges V_2 to V_1 to be equal to b. We denote with $W_{a,b}$ such a graph.

We consider a $W_{a,b}$ of order $n = 100$, with $a = 1$ and $b = (n+1)/n = 1.01$. We compare the $(1+1)$ *EA* using uniform mutation, fmut_β with $\beta = 1.5, 2, 3, 4$, and cMut_p with $p = 0.1, 0.5, 0.9$. We let the algorithm run for a fixed time budget (100 fitness evaluations, 1000 fitness evaluations, and 10000 fitness evaluations), and look at the solution quality. We compare performance by considering the sample mean of 100 such runs. The results are displayed in Table 2. We observe that for low time budget the (1+1) EA with fmut_β mutation tends to outperform cMut_p, whereas for sufficiently large time budget the situation reverses. Note also that the $(1 + 1)$ *EA* with fmut_β is the only setting that reaches the global maximum within 10000 fitness evaluations. We consider the same model as described above, but with reduced number of edges. The goal of the following experiments is to perform a comparison of the uniform, fmut_β and cMut_p on a graph that is slightly more irregular than the complete bipartite case.

mutation	100 steps	1000 steps	10000 steps
uniform	1609.5	2263.7	2500.0
$\text{fmut}_{1.5}$	1705.4	2503.4	2513.8
$\text{fmut}_{2.0}$	1761.4	2513.1	2514.6
$\text{fmut}_{3.0}$	1761.5	2514.1	2514.6
$\text{fmut}_{4.0}$	1749.3	2166.7	2514.1
$\text{cMut}_{0.1}$	1467.4	2511.5	2525.0
$\text{cMut}_{0.5}$	1618.3	2521.5	2525.0
$\text{cMut}_{0.9}$	1618.3	2521.5	2525.0

Table 2: We consider the run time of the (1+1) EA with uniform mutation, fmut_β with $\beta = 1.5, 2, 3, 4$ and cMut_p with $p = 0.1, 0.5, 0.9$ on the instance $W_{a,b}$. For a given time budget, we consider 100 independent runs and display the sample mean of the solution quality found in each run. Note that for increasing number of runs the (1+1) EA with mutation cMut_p yields best performance.

Definition 5.2. For a directed simple graph $G = (V, E)$ we define the density as

$$\rho = \frac{|E|}{|V|\,(|V| - 1)}.$$

Throughout the remaining part of this section, we denote with $\hat{\rho}$ the density of $W_{a,b}$. Note that it holds $\hat{\rho} \sim 0.5$. In a second set of experiments we consider again a $W_{a,b}$ of order $n = 100$, with $a = 1$ and $b = (n + 1)/n = 1.01$, and we remove edges u.a.r. until the new resulting density is $\rho = 0.5\hat{\rho} \sim 0.25$. Again, we let the algorithm run for a fixed time budget (100 evaluations, 1000 evaluations, and 10000 evaluations), and look at the solution quality. We compare performance by considering 100 independent runs, and looking at the sample mean over all resulting solution qualities. The results are displayed in Table 3. Again, we observe that for increasing time budget the $(1 + 1)$ *EA* with mutation cMut_p tends to give better approximation of the optimal solution, then with uniform and fmut_β mutation rates. We conclude this set of experiments by further lowering the density of the graph $W_{a,b}$. Again, we randomly remove edges to the instance $W_{a,b}$ with $n = 100$, with $a = 1$, $b = (n + 1)/n = 1.01$, and lower the density to $\rho = 0.1\hat{\rho}$. We compare performance by considering 100 such run, and looking at the sample mean over all resulting solution qualities. The results are displayed in Table 4. Again, we observe that for increasing problem size the mutation cMut_p gives the highest solution quality. In this case, however, the results are less clear. This set of experiments is performed on MacBook Pro wit Intel Core i7, 3.1 GHz, 16 GB RAM.

mutation	100 steps	1000 steps	10000 steps
uniform	801.6	1141.7	1247.4
$\text{fmut}_{1.5}$	840.4	1255.8	1261.2
$\text{fmut}_{2.0}$	844.4	1257.1	1258.3
$\text{fmut}_{3.0}$	869.2	1259.3	1259.7
$\text{fmut}_{4.0}$	873.8	1261.2	1261.3
$\text{cMut}_{0.1}$	724.4	1042.4	1269.9
$\text{cMut}_{0.5}$	722.8	1267.6	1273.2
$\text{cMut}_{0.9}$	855.4	1269.4	1271.2

Table 3: We consider the run time of the (1+1) EA with uniform mutation, fmut_β with $\beta = 1.5, 2, 3, 4$ and cMut_p with $p = 0.1, 0.5, 0.9$ on the instance $W_{a,b}$ with density $\rho \sim 0.25$. For a given time budget, we consider 100 independent runs and display the sample mean of the solution quality found in each run. Note that for increasing number of runs the (1+1) EA with mutation cMut_p yields best performance.

mutation	100 steps	1000 steps	10000 steps
uniform	174.5	226.4	255.2
$\text{fmut}_{1.5}$	178.1	251.9	255.9
$\text{fmut}_{2.0}$	177.8	249.9	253.0
$\text{fmut}_{3.0}$	179.2	249.7	231.3
$\text{fmut}_{4.0}$	177.8	251.5	252.3
$\text{cMut}_{0.1}$	158.7	202.4	256.3
$\text{cMut}_{0.5}$	170.5	246.4	255.8
$\text{cMut}_{0.9}$	177.3	251.8	255.3

Table 4: We consider the run time of the (1+1) EA with uniform mutation, fmut_β with $\beta = 1.5, 2, 3, 4$ and cMut_p with $p = 0.1$ on the instance $W_{a,b}$ with density $\rho \sim 0.05$. The experimental setting is as given in Figure 4. For a given time budget, we consider 100 independent runs and display the sample mean of the solution quality found in each run. Again, we observe that the $(1+1)$ *EA* with mutation cMut_p gives best performance, although the $(1 + 1)$ *EA* using mutation fmut_β with $\beta = 0.1$ gives similar performance.

6 SUMMARY

In this paper we investigate the benefit of using heavy mutation operators to escape large deceptive basins of attractions, with the $(1+1)$ *EA* (cf. Algorithm 1). We propose a new mutation operator cMut_p (cf. Algorithm 3), with $p \in (0,1)$ a parameter that gives the probability of performing a single bit-flip. The probability of performing a k-bit flip with $k > 1$ is always $(n-1)/(p-1)$, with n the problem size.

We consider the run time of the $(1+1)$ *EA* with cMut_p on commonly studied pseudo-boolean landscapes. We prove that the $(1+1)$ *EA* with mutation cMut_p optimizes the OneMax within expected $O\left(\frac{n}{p}\log n\right)$ fitness evaluations, and the TwoMax after expected $O\left(\frac{n}{p}\log n + \frac{n}{1-p}\right)$ fitness evaluations (cf. Lemma 3.1 and Lemma 3.4). In both cases, if p is constant for increasing problem size, the (1+1) EA with mutation cMut_p yields an upper bound of $O(n\log n)$ on the expected run time.

We then study the $\mathrm{Jump}(m,n)$ function. We first derive a general upper-bound for the (1+1) EA using mutation cMut_p (cf. Lemma 3.5). We then perform a comparison on the run time of the (1+1) EA with mutation cMut_p ($T_p(f)$) and fmut_β ($T_\beta(f)$), on functions $f = \mathrm{Jump}(m,n)$ with the parameter m evolving in different ways. We first observe that if m is constant for increasing problem size, then it holds $T_p(f) = \Theta(nT_\beta(f))$ (cf. Lemma 3.6); we then observe that $T_p(f) = O\left(T_\beta(f)\right)$ for all $\sqrt[\beta]{(n-1)/(1-p)} \le m \le n$ (cf. Lemma 3.7); we further show that for $m-n$ constant for increasing problem size it holds $T_p(f) = n^{\Theta(1)}$ and $T_\beta(f) = 2^{\Omega(n)}$ (cf. Lemma 3.8). We then experimentally locate the point m^* by which the cMut_p outperforms the fmut_β and uniform mutation (cf. Figure 1).

We also study the benefits of heavy-tailed mutation operators on two combinatorial optimization problems. We first observe that on complete bipartite graphs the $(1+1)$ *EA* with mutation cMut_p finds a Minimum Vertex Cover after expected $O(n\log n)$ fitness evaluations (cf. Theorem 4.2). We then consider the maxCut problem on directed graphs with unweighted edges. We find that the (1+1) EA using mutation cMut_p achieves a $(1/3-\epsilon/n^2)$-approximation of the optimal solution after expected $O\left(\frac{1}{\epsilon}n^3\log(n\Delta)\right)$ fitness evaluations (cf. Theorem 4.7), with Δ the outer degree of the graph.

We conclude by experimentally comparing the run time of the (1+1) EA with different mutation rates on some instances of the maxCut. We consider a complete bipartite graph $G = (V,E)$ of order $n = 100$, and s.t. $|V_1| = |V_2|$; each edge has probability 1/2 of being directed from V_1 to V_2, and probability 1/2 to be directed from V_2 to V_1. We use a parameter tuner to evolve the mutation rate of the (1+1) EA to best performance, on the instance described above. The results are displayed in Table 1. We observe that the evolving distribution is in line with the general intuition behind the operator cMut_p.

We investigate the performance of the (1+1) EA with various mutation rates on the maxCut. We consider the instance $W_{a,b}$, as given in Definition 5.1. We first consider the run time of the (1+1) EA with uniform mutation, fmut_β with $\beta = 1.5, 2, 3, 4$ and cMut_p with $p = 0.1$ on this instance (cf. Table 2). We observe that the mutation cMut_p is the only operator that always finds the optimal solution, within our considered time budget. We (randomly) lower the density of $W_{a,b}$ and compare the (1+1) EA with uniform mutation, fmut_β with $\beta = 1.5, 2, 3, 4$ and cMut_p with $p = 0.1$ on the resulting instances (cf. Table 3 and Table 4). Again, we observe that for increasing problem size the cMut_p gives the highest solution quality, although the results become less strong as the density lowers.

Overall, we observe that there are benefits of using non-standard mutation probabilities. Since the $\mathrm{Jump}(m,n)$ and TwoMax functions are useful functions to understand how well algorithms escape local optima, we expect our operator to yield good run time on landscapes that exhibit deceptive basins of attractions. We plan to further explore the benefits of heavy-tailed distribution in combinatorial optimization in the future.

ACKNOWLEDGEMENTS

M. Wagner was supported by the Australian Research Council project DE160100850.

REFERENCES

[1] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. 2009. *A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms.* 142–157.

[2] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. 2002. A Racing Algorithm for Configuring Metaheuristics. In *Proc. of GECCO.* 11–18.

[3] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. 2010. Optimal Fixed and Adaptive Mutation Rates for the LeadingOnes Problem. In *Proc. of PPSN.* 1–10.

[4] Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. 2013. Mutation Rate Matters Even When Optimizing Monotonic Functions. *Evolutionary Computation* 21, 1 (2013), 1–27.

[5] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. 2017. Fast genetic algorithms. In *Proc. of GECCO.* 777–784.

[6] Stefan Droste, Thomas Jansen, and Ingo Wegener. 2002. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science* 276, 1-2 (2002), 51–81.

[7] Stefan Droste, Thomas Jansen, and Ingo Wegener. 2002. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science* 276, 1-2 (2002), 51–81.

[8] A. E. Eiben, R. Hinterding, and Z. Michalewicz. 1999. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 3, 2 (1999), 124–141.

[9] A. E. Eiben and J. E. Smith. 2003. *Introduction to evolutionary computation.* Springer.

[10] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. 2011. Maximizing Non-monotone Submodular Functions. *SIAM Journal of Computing* 40, 4 (2011), 1133–1153.

[11] Tobias Friedrich, Jun He, Nils Hebbinghaus, Frank Neumann, and Carsten Witt. 2010. Approximating Covering Problems by Randomized Search Heuristics Using Multi-Objective Models. *Evolutionary Computation* 18, 4 (2010), 617–633.

[12] Tobias Friedrich and Frank Neumann. 2015. Maximizing Submodular Functions under Matroid Constraints by Evolutionary Algorithms. *Evolutionary Computation* 23, 4 (2015), 543–558.

[13] Tobias Friedrich, Pietro S. Oliveto, Dirk Sudholt, and Carsten Witt. 2009. Analysis of Diversity-preserving Mechanisms for Global Exploration. *Evolutionary Computation* 17, 4 (2009), 455–476.

[14] Christian Gießen and Carsten Witt. 2015. Population Size vs. Mutation Strength for the (1+λ)EA on OneMax. In *Proc. of GECCO.* 1439–1446.

[15] Clarissa Van Hoyweghen, David E. Goldberg, and Bart Naudts. 2002. From Twomax To The Ising Model: Easy And Hard Symmetrical Problems. In *Proc. of GECCO.* 626–633.

[16] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Proc. of LION.* 507–523.

[17] Thomas Jansen and Ingo Wegener. 2005. Real royal road functions–where crossover provably is essential. *Discrete Applied Mathematics* 149, 1-3 (2005), 111–125.

[18] Heinz Mühlenbein. 1992. How Genetic Algorithms Really Work: Mutation and Hillclimbing. In *Proc. of PPSN.* 15–26.

[19] Martin Pelikan and David E. Goldberg. 2000. Genetic Algorithms, Clustering, and the Breaking of Symmetry. In *Proc. of PPSN.* 385–394.

[20] Ingo Wegener. 2001. Theoretical Aspects of Evolutionary Algorithms. In *Proc. of ICALP.* 64–78.