A Rigorous Runtime Analysis of the

$(1 + (\lambda, \lambda))$

GA on Jump Functions

Denis Antipov, Benjamin Doerr, Vitalii Karavaev

A Rigorous Runtime Analysis of the $(1 + (\lambda, \lambda))$ GA on Jump Functions[†]

Denis Antipov[*]
ITMO University
St. Petersburg, Russia
and
Laboratoire d'Informatique (LIX),
CNRS, École Polytechnique,
Institut Polytechnique de Paris
Palaiseau, France
antipovden@yandex.ru

Benjamin Doerr
Laboratoire d'Informatique (LIX),
CNRS, École Polytechnique,
Institut Polytechnique de Paris
Palaiseau, France
lastname@lix.polytechnique.de

Vitalii Karavaev
ITMO University
St. Petersburg, Russia
fkve97@gmail.com

November 24, 2021

Abstract

The $(1 + (\lambda, \lambda))$ genetic algorithm is a younger evolutionary algorithm trying to profit also from inferior solutions. Rigorous runtime analyses on unimodal fitness functions showed that it can indeed be

[†]Extended version of the paper [ADK20] in the proceedings of GECCO 2020. This version contains all proofs and other details that had to be omitted in the conference version for reasons of space. Also, we have added a new section which proves the lower bounds.

[*]Corresponding author

faster than classical evolutionary algorithms, though on these simple problems the gains were only moderate.

In this work, we conduct the first runtime analysis of this algorithm on a multimodal problem class, the jump functions benchmark. We show that with the right parameters, the $(1 + (\lambda, \lambda))$ GA optimizes any jump function with jump size $2 \leq k \leq n/4$ in expected time $O(n^{(k+1)/2} e^{O(k)} k^{-k/2})$, which significantly and already for constant k outperforms standard mutation-based algorithms with their $\Theta(n^k)$ runtime and standard crossover-based algorithms with their $\tilde{O}(n^{k-1})$ runtime guarantee.

For the isolated problem of leaving the local optimum of jump functions, we determine provably optimal parameters that lead to a runtime of $(n/k)^{k/2} e^{\Theta(k)}$. This suggests some general advice on how to set the parameters of the $(1 + (\lambda, \lambda))$ GA, which might ease the further use of this algorithm.

1 Introduction

The $(1 + (\lambda, \lambda))$ genetic algorithm, $(1 + (\lambda, \lambda))$ GA for short, is a relatively new genetic algorithm, first proposed at GECCO 2013 [DDE13], that tries to increase the rate of exploration by a combination of mutation with a high mutation rate, an intermediate selection, and crossover as mechanism to repair the possible negative effects of the aggressive mutation. For this algorithm, moderate runtime advantages over classic algorithms have been proven for unimodal [DDE15, DD18] or close-to-unimodal [BD17] problems; also some positive experimental results exist [GP14, MB17].

In this work, we conduct the first mathematical runtime analysis for the $(1 + (\lambda, \lambda))$ GA optimizing a multimodal optimization problem, namely the classic jump functions benchmark. We observe that the combination of aggressive mutation with crossover as repair mechanism works even better here: The $(1 + (\lambda, \lambda))$ GA can optimize jump functions with gap size $k \leq n/4$ in expected time at most

$$n^{(k+1)/2} e^{O(k)} k^{-k/2},$$

which is almost the square root of the $\Omega(n^k)$ runtime many classic mutation-based algorithms have. To obtain this performance, however, the parameters of the algorithm have to be set differently from what previous works recommend.

1.1 The $(1 + (\lambda, \lambda))$ GA

Noting that many classic evolutionary algorithms do not profit a lot from inferior solution, whereas algorithms witnessing the black-box complexity [DJW06] (see also [DD20] for a recent survey), massively do, Doerr, Doerr, and Ebel [DDE13] proposed an algorithm which tries to gain some insight also from solutions inferior to the current-best solution.

The main working principle of their algorithm, which was called $(1 + (\lambda, \lambda))$ GA, is as follows. From a unique parent individual x, first λ offspring are created using standard bit mutation with a relatively high mutation rate p, but in a way that all offspring have equal Hamming distance to x (this can be realized, for example, by first sampling a number ℓ from a binomial distribution with parameters n and p and then generating all offspring by flipping exactly ℓ random bits in x). When the parent is already close to the optimum, these most likely are all worse than the parent. The hope set into the $(1 + (\lambda, \lambda))$ GA is that nevertheless some mutation offspring, besides all destruction from the aggressive mutation, has also made some progress. To distill such progress the $(1 + (\lambda, \lambda))$ GA selects a mutation offspring x' with maximal fitness and creates from it, λ times independently, an offspring via a biased uniform crossover with the parent x. This biased crossover inherits bits from x' only with some small probability c, so that, hopefully, all the destruction caused by the aggressive mutation is repaired. The best of these crossover offspring in an elitist selection competes with x for becoming the parent of the next iteration. The recommendation in previous works was to use a crossover bias of $c = \frac{1}{pn}$. With this parameterization, a single application of mutation and crossover with the parent, without intermediate selection, would create an offspring distributed as if generated via standard bit mutation with mutation rate $\frac{1}{n}$. Note that $\frac{1}{n}$ is a common recommendation for the mutation rate in standard bit mutation [Bäc93, Müh92, Wit13].

Via a rigorous runtime analysis on the OneMax benchmark function it was shown [DDE15] that the basic idea of the $(1 + (\lambda, \lambda))$ GA indeed can work. When the crossover biased is set to $c = \frac{1}{pn}$ as recommended, then the expected runtime (number of fitness evaluations) of the $(1 + (\lambda, \lambda))$ GA with any mutation rate $p \geq \frac{2}{n}$ and offspring population size $\lambda \geq 2$ is

$$O((\tfrac{1}{pn} + \tfrac{1}{\lambda})n \log n + (pn + \lambda)n).$$

Hence any choice of $p \in \omega(\frac{1}{n}) \cap o(\frac{\log n}{n})$ and $\lambda \in \omega(1) \cap o(\log n)$ yields a runtime asymptotically faster than the runtime $\Omega(n \log n)$ observed by many classic evolutionary algorithms, e.g., by the $(1 + 1)$ EA [Müh92], the $(1 + \lambda)$ EA [JJW05], the $(\mu + 1)$ EA [Wit06], the $(\mu + \lambda)$ EA [AD21], and in fact any unary unbiased black-box algorithm [LW12]. The choice $p = \frac{\sqrt{\log n}}{n}$

and $\lambda = \sqrt{\log n}$ minimizes the runtime guarantee above and shows an expected runtime of $O(n\sqrt{\log n})$. With a fitness-dependent [DDE15], self-adjusting [DD18], or heavy-tailed random parameter choice [ABD20], the runtime further improves to $O(n)$. Clearly, these are not a drastic improvement over, say, the $O(n \log n)$ runtime of the $(1+1)$ EA, but one has to admit that the room for improvement is limited: The unrestricted black-box complexity of the OneMax function class is $\Omega(\frac{n}{\log n})$ [ER63, DJW06], hence no black-box optimizer can optimize all functions isomorphic to OneMax in a time better than $O(\frac{n}{\log n})$.

A runtime analysis [BD17] of the $(1+(\lambda,\lambda))$ GA on the random satisfiability instances regarded in [SN14] showed a similar performance as on OneMax. This is caused by the structure of these random instances, which renders them similar to OneMax to the extent that also the $(1+1)$ EA has an $O(n \log n)$ performance [DNS17]. At the same time, these instances do not have the perfect fitness-distance correlation of the OneMax function, and this indeed needed to be taken into account when setting the parameters of the $(1+(\lambda,\lambda))$ GA in [BD17]. A runtime analysis of the $(1+(\lambda,\lambda))$ GA on LeadingOnes [ADK19] showed that for this problem, the $(1+(\lambda,\lambda))$ GA with any $\lambda \leq \frac{n}{2}$ has asymptotically the same runtime of $\Theta(n^2)$ as many other algorithms.

Empirical studies showed that the $(1+(\lambda,\lambda))$ GA works well (compared to classic EAs) on linear functions and RoyalRoad functions [DDE15], on the MAX-3SAT problem [GP14], and on the problem of hard test generation [MB17].

1.2 Multimodal Problems

Clearly, the usual application of evolutionary algorithms are problems with multimodal landscapes, that is, with non-trivial local optima, and these local optima often present a difficulty for the evolutionary algorithm. In the runtime analysis perspective multimodal problems have displayed very different optimization behaviors. For example, on multimodal landscapes it has been observed that crossover can recombine solutions into significantly better ones [JW02, SW04, Sud05], that mutation rates significantly larger than $\frac{1}{n}$ can be preferable [DLMN17], and that probabilistic model-building algorithms such as estimation-of-distribution algorithms and ant-colony optimizers can significantly outperform classic algorithms [HS18, Doe21, DK20, BBD21b].

In this light, and given that all previous runtime analyses for the $(1+(\lambda,\lambda))$ GA consider unimodal or almost unimodal problems, we feel that it is the right time to now investigate how the $(1+(\lambda,\lambda))$ GA optimizes

multimodal problems. Being the most studied multimodal benchmark in runtime analysis, we regard jump functions. These have a fitness landscape isomorphic to the one of OneMax except that there is a valley of low fitness around the optimum. Consequently, a typical hillclimber and also most evolutionary algorithms quickly run into the local optimum consisting of all points on the edge of the fitness valley, but then find it hard to cross the fitness valley.

More precisely, the jump function class comes with a difficulty parameter k, which is the width of the valley of low fitness. The fitness is essentially the fitness of OneMax except for all search points with Hamming distance between one and $k-1$ from the optimum. Consequently, the only way to leave the local optimum to a strictly better search point is to flip exactly the right k bits and go to the optimum.[1] For this reason, it comes as no surprise that many mutation-based evolutionary algorithms need $\Omega(n^k)$ time to optimize such a jump function [DJW02, Doe20a]. Using a higher or a heavy-tailed random mutation rate [DLMN17] or stagnation detection mechanisms [RW20, RW21b, RW21a] the runtime can be reduced, but not below $\Omega((\frac{n}{k})^k)$. Crossover can be helpful, but the maybe most convincing work [DFK+18] in this direction also only obtains a runtime of $O(n^{k-1}\log n)$ with the standard mutation rate and $O(n^{k-1})$ with a higher mutation rate. With additional techniques, runtimes up to $O(n)$ were obtained [DFK+16, FKK+16, WVHM18, RA19], but the lower the runtimes become, the more these algorithms appear custom-tailored to jump functions (see, e.g., [Wit21]). The extreme end is marked by an $O(\frac{n}{\log n})$ time algorithm [BDK16] designed to witness the black-box complexity of jump functions.

1.3 Our Results

Our main result is a runtime analysis of the $(1+(\lambda,\lambda))$ GA on jump functions for all jump sizes $k \in [2..\frac{n}{4}]$. Since we could not be sure that the parameter suggestions from previous works are still valid for our problem, we consider arbitrary values for the mutation rate p, the crossover bias c, and the offspring population size λ. This turned out to be the right decision as we observed much better runtimes with novel parameter values[2]. We also allowed different

[1] This particular structure of the jump benchmark has been criticized and several variants have been proposed [Jan15, RW21a, BBD21a]. With the overwhelming majority of the runtime analyses on multimodal problems still regarding the classic jump benchmark, for the sake of comparability we prefer to regard this benchmark as well.

[2] In [FS20] it was shown that the $(1+(\lambda,\lambda))$ GA with the standard parameter setting is not effective on Jump even when parameter control mechanisms are applied to λ. This is

offspring population sizes λ_M and λ_C for the mutation and crossover phase, which however did not lead to stronger runtime guarantees.

For all $k \in [2..\frac{n}{4}]$ and for arbitrary values of these four parameters (except for the only constraint $p \geq \frac{2k}{n}$), we prove that the $(1+(\lambda,\lambda))$ GA when started in the local optimum of $\textsc{Jump}_k$ crosses the fitness valley in expected time (number of fitness evaluations) at most

$$E[T] \leq \frac{4(\lambda_M + \lambda_C)}{q_\ell \min\{1, \lambda_M (\frac{p}{2})^k\} \min\{1, \lambda_C c^k (1-c)^{2pn-k}\}},$$

where q_ℓ is a constant in $[0.1, 1]$. When ignoring the hidden constants in the $e^{O(k)}$ factor, this bound is optimized for $p = c = \sqrt{\frac{k}{n}}$ and $\lambda_M = \lambda_C = n^{k/2} k^{-k/2}$ and then gives a runtime of

$$E[T] = n^{k/2} e^{O(k)} k^{-k/2}.$$

This time bound is asymptotically optimal, that is, no other parameter values can obtain a faster expected runtime (apart from the unspecified $e^{O(k)}$ factor).

When not starting in the local optimum, but with an arbitrary initial solution or the usual random initialization, the $(1+(\lambda,\lambda))$ GA reaches the local optimum in an expected time of $ne^{O(k)}$ iterations, if $p = c = \sqrt{\frac{k}{n}}$ and λ_M and λ_C are at least $\frac{n}{k}$. Therefore, large population sizes are not beneficial in this first easy part of the optimization. With slightly smaller values for the population sizes as above, namely $\lambda_M = \lambda_C = n^{(k-1)/2} k^{-k/2}$, the expected runtime is

$$E[T] \leq n^{(k+1)/2} e^{O(k)} k^{-k/2}.$$

Similar as in the previous results on OneMax, a speed-up over classic algorithms is observed for larger ranges of parameters, though these are harder to describe in a compact fashion (see Corollary 10 for the details).

The result above shows that the power of the $(1+(\lambda,\lambda))$ GA becomes much more visible for jump functions than for the problems regarded in previous works. Concerning the optimal parameter values, we observe that they differ significantly from those that were optimal in the previous works. In particular, the relation of mutation rate and crossover bias is different. Whereas in previous works $pcn = 1$ was a good choice, we now have $pcn = k$. A moment's thought, however, shows that this is quite natural, or, being more cautious, at least fits to the previous results. We recall that pcn is the expected Hamming distance of the parent from an individual generated

another reason to step back from the standard parameters and consider a wider parameters space.

from one isolated application of mutation and crossover. The previous works suggested that this number should be one, since one is also the expected distance of an offspring generated the classic way, that is, via standard bit mutation with mutation rate $\frac{1}{n}$.

Now for the optimization of jump functions, where a non-trivial local optimum has to be left, it makes sense to put more weight on larger moves in the search space. More specifically, the work [DLMN17] has shown that the optimal mutation rate for the $(1+1)$ EA optimizing jump functions is $\frac{k}{n}$. Hence for the classic $(1+1)$ EA, the best way of generating offspring is such that they have an expected Hamming distance of k from the parent. Clearly, this remains an intuitive argument, but it shows that also when optimizing multimodal problems, the intuitive approach of previous works, which might help an algorithm designer, gave the right intuition.

Our recommendation when using the $(1+(\lambda,\lambda))$ GA for multimodal optimization problems would therefore be to choose p and c larger than in previous works, and more specifically, in a way that *pcn* is equal to an estimate for the number of bits the algorithm typically should flip. Here "typically" does not mean that there are actually many moves of this size, but that this is the number of bits the algorithm has to flip most often. For example, when the $(1+1)$ EA optimizes a jump function, it will maybe only once move to a search point in distance k, however, it will nevertheless need many offspring in distance k until it finds the right move of this distance.

From our rigorous analysis, we conclude that the $(1+(\lambda,\lambda))$ GA is even better suited for the optimization of multimodal objective functions, and we hope that the just sketched intuitive considerations help algorithm designers to successfully apply this algorithm to their problems.

Research conducted after ours: In [AD20], it was shown that the non-trivial choice of the parameters of the $(1+(\lambda,\lambda))$ GA when optimizing multimodal problems can partially be overcome by using heavy-tailed random parameter values. If we choose λ from a power-law distribution with exponent $\beta_\lambda > 2$ and set $p = c = \sqrt{\frac{s}{n}}$, where s follows another power-law distribution with exponent $\beta_s > 1$, then for all $k \geq 3$ the runtime of the $(1+(\lambda,\lambda))$ GA on $\textsc{Jump}_k$ is $(n/k)^{(1+\varepsilon)k/2} e^{\Theta(k)}$ for any small constant $\varepsilon > 0$, which is only by a $(n/k)^{\varepsilon k/2} n^{-1/2}$ factor larger (and for some k and ε even smaller) than the upper bound for the optimal static parameters (apart from the unspecified $e^{O(k)}$ factors).

In [ABD21] it was further shown that if all three parameters of the $(1+(\lambda,\lambda))$ GA are chosen independently, then the runtime stays the same, namely $(n/k)^{(1+\varepsilon)k/2} e^{\Theta(k)}$. The empirical analysis in [ABD21] also shows that the $(1+(\lambda,\lambda))$ GA with the heavy-tailed choice of parameters significantly

outperforms the $(1+1)$ EA on small jump sizes ($k = 3$ and $k = 5$ were considered).

2 Preliminaries and Notation

2.1 Notation

By $\mathbb{N}$ we understand the set of positive integers. We write $[a..b]$ to denote an integer interval including its borders and $(a..b)$ to denote an integer interval excluding its borders. For $a, b \in \mathbb{R}$ the notion $[a..b]$ means $[\lceil a \rceil .. \lfloor b \rfloor]$. For the real-valued intervals we write $[a, b]$ and (a, b) respectively. For any probability distribution $\mathcal{L}$ and random variable X, we write $X \sim \mathcal{L}$ to indicate that X follows the law $\mathcal{L}$. We denote the binomial law with parameters $n \in \mathbb{N}$ and $p \in [0, 1]$ by $\mathrm{Bin}(n, p)$.

2.2 The $(1 + (\lambda, \lambda))$ GA

The main idea of the $(1 + (\lambda, \lambda))$ GA discussed in Section 1.1 is realized as follows. The $(1 + (\lambda, \lambda))$ GA stores a bit string x that is initialized with a random bit string. After the initialization it performs iterations which consist of a mutation phase and a crossover phase until some stopping criterion is met.

In the mutation phase the algorithm first chooses the mutation strength ℓ from the binomial distribution with parameters n and p. Then it creates λ_M mutants $x^{(1)}, \ldots, x^{(\lambda_M)}$, each of them is a copy of x with exactly ℓ bits flipped. The positions of the flipped bits are chosen uniformly at random, independently for each mutant. The goal of this design of the mutation phase is to generate each of λ_M offspring via standard bit mutation, but conditional on that all offspring have the same distance to their parent x. The mutant x' with the best fitness is chosen as a winner of the mutation phase.

In the crossover phase the algorithm creates λ_C offspring $y^{(1)}, \ldots, y^{(\lambda_C)}$ by applying a biased crossover to x and x'. The crossover operator for each position takes a bit value from x with probability $1 - c$ and it takes a bit value from x' with probability c (independently for each position and each offspring). If the best offspring y is not worse than x then it replaces x. The pseudocode of the $(1 + (\lambda, \lambda))$ GA optimizing a pseudo-Boolean function f is shown in Algorithm 1.

We intentionally do not specify a stopping criterion, which is a common practice in theoretical studies. The goal of our analysis is to determine the expected runtime of the $(1 + (\lambda, \lambda))$ GA until it finds an optimal solution.

Algorithm 1: The $(1+(\lambda,\lambda))$ GA maximizing function $f : \{0,1\}^n \to \mathbb{R}$.

$x \leftarrow$ random bit string of length n;
while *not terminated* **do**
  Mutation phase:
  Choose $\ell \sim \mathrm{Bin}(n,p)$;
  for $i \in [1..\lambda_M]$ **do**
    $x^{(i)} \leftarrow$ a copy of x;
    Flip ℓ bits in $x^{(i)}$ chosen uniformly at random;
  end
  $x' \leftarrow \arg\max_{z \in \{x^{(1)},\dots,x^{(\lambda)}\}} f(z)$;
  Crossover phase:
  for $i \in [1..\lambda_C]$ **do**
    $y^{(i)} \leftarrow$ a copy of x;
    Flip each bit in $y^{(i)}$ that is different in x' with probability c;
  end
  $y \leftarrow \arg\max_{z \in \{y^{(1)},\dots,y^{(\lambda)}\}} f(z)$;
  if $f(y) \geq f(x)$ **then**
    $x \leftarrow y$;
  end
end

By the runtime we understand the number of iterations or fitness evaluations which the algorithm performs. Since each iteration of the algorithm uses exactly $\lambda_M + \lambda_C$ fitness evaluations, the transition between these two measures of runtime is trivial.

2.3 Jump Functions

The class of jump functions is defined through the classic OneMax function, which is defined on the space of bit strings of length n and returns the number of one-bits in its argument. In formal words,

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

This function despite its simplicity has given a birth to many fundamental results, e.g. [Dro02, Dro04, Dro05, JJW05, DHK12, Wit13, RS14, BLS14]. In particular, the analysis of the black-box complexity of OneMax led to the development of the $(1+(\lambda,\lambda))$ GA [DDE15].

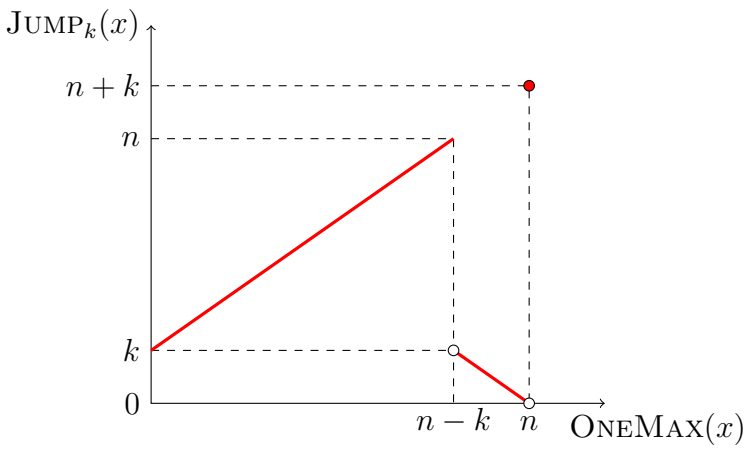

Figure 1: Plot of the JUMP$_k$ function. As a function of unitation, the function value of a search point x depends only on the number of one-bits in x.

The JUMP$_k$ function with parameter $k \in [2..n]$ is then defined as follows.

$$\text{JUMP}_k(x) = \begin{cases} \text{OM}(x) + k, & \text{if } \text{OM}(x) \in [0..n-k] \cup \{n\}, \\ n - \text{OM}(x), & \text{if } \text{OM}(x) \in [n-k+1..n-1]. \end{cases}$$

A plot of JUMP$_k$ is shown in Figure 1.

2.4 Useful Tools

In this section we provide some useful tools which we use in our proofs. We start with the following inequality which we use for multiple times in our proofs.

Lemma 1. *Assume $x \in [0,1]$ and $\lambda > 0$. Then*

$$1 - (1-x)^\lambda \geq \frac{1}{2} \min\{1, \lambda x\}.$$

Proof. By [RS14, Lemma 8] we have $(1-x)^\lambda \leq \frac{1}{1+\lambda x}$. Hence,

$$\begin{aligned} 1 - (1-x)^\lambda &\geq 1 - \frac{1}{1+\lambda x} = \frac{\lambda x}{1+\lambda x} \\ &\geq \frac{\lambda x}{2\max\{1, \lambda x\}} = \frac{1}{2}\min\{\lambda x, 1\}. \end{aligned}$$

□

We also make a use of Chernoff bounds (see Theorem 1.10.1 and 10.10.5 in [Doe20b]) to show the concentration of some random variables involved in our analysis. We use the following lemma, which is a particular case of these bounds for the random variables following a binomial distribution.

Lemma 2 (Chernoff Bounds)**.** *Let X be a random variable following a binomial distribution* $\mathrm{Bin}(n,p)$*. Then for all $\delta \in (0,1)$ the probability that $X \geq (1+\delta)np$ is at most $e^{-\frac{\delta^2 np}{3}}$ and the probability that $X \leq (1-\delta)np$ is at most $e^{-\frac{\delta^2 np}{2}}$. Also for all $\delta \geq 1$ the probability that $X \geq (1+\delta)np$ is at most $e^{-\frac{\delta np}{3}}$.*

The following lemma shows the concentration of the number of the bit flips in the mutation phase by the Chernoff bounds.

Lemma 3. *Let $p \geq \frac{1}{n}$. Then the number ℓ of the bits flipped by the mutation operator of the $(1+(\lambda,\lambda))$ GA is in $[pn, 2pn]$ with at least constant probability $q_\ell \geq 0.1$, if n is at least some sufficiently large constant.*

Proof. Recall that the number ℓ of the flipped bits is chosen according to the binomial distribution $\mathrm{Bin}(n,p)$. We first consider the case when p is small. Assume $pn \in [1,9]$. Then

$$\begin{aligned}\Pr[\ell \in [pn,2pn]] &\geq \Pr[\ell = \lceil pn \rceil] = \binom{n}{\lceil pn \rceil} p^{\lceil pn \rceil}(1-p)^{n-\lceil pn \rceil} \\ &= \frac{n!}{(n-\lceil pn \rceil)!\lceil pn \rceil!} \cdot \frac{(pn)^{\lceil pn \rceil}}{n^{\lceil pn \rceil}} \cdot (1-p)^{n-\lceil pn \rceil}.\end{aligned}$$

Note that

$$\begin{aligned}\frac{n!}{(n-\lceil pn \rceil)!n^{\lceil pn \rceil}} &\geq \frac{(n-\lceil pn \rceil+1)^{\lceil pn \rceil}}{n^{\lceil pn \rceil}} \geq (1-p)^9 = (1-o(1)), \\ (1-p)^{n-\lceil pn \rceil} &\geq (1-p)^{n-np} = (1-p)^{(\frac{1}{p}-1)np} \geq e^{-np}.\end{aligned}$$

To estimate $\frac{(pn)^{\lceil pn \rceil}e^{-np}}{\lceil pn \rceil!}$ we consider function $f(x) = \frac{x^{\lceil x \rceil}e^{-x}}{\lceil x \rceil!}$ in the interval $[1,9]$ (since we now consider only these values of pn). First we note that $f(1) = e^{-1} \geq 0.3$ and further we find its infimum in $(1,9]$. Note that $f(x)$ is increasing in each interval $(i, i+1]$, hence we can bound it from below by $f(x) \geq g(x) = \frac{(\lceil x \rceil - 1)^{\lceil x \rceil}e^{-(\lceil x \rceil - 1)}}{\lceil x \rceil!}$. Note also that $g(x)$ is a non-increasing

function in $(1, 9]$, thus $f(x) \geq g(9) \geq 0.12$ for all $x \in (1, 9]$. Consequently, for n which is large enough we have

$$\Pr[\ell \in [pn, 2pn]] \geq (1 - o(1)) \cdot 0.12 \geq 0.1.$$

Now we consider the case when $pn \geq 9$. Since ℓ follows the binomial distribution with parameters n and p, we have $E[\ell] = pn$. By the Chernoff bounds we have

$$\Pr[\ell \geq 2E[\ell]] \leq \exp\left(-\frac{E[\ell]}{3}\right) = \exp\left(-\frac{pn}{3}\right).$$

By Theorem 10 in [Doe18] we have the following bound on the probability that the binomial distribution exceeds its expectation.

$$\Pr[\ell \geq E[\ell] = pn] \geq \frac{1}{4}.$$

Hence,

$$\Pr[\ell < pn] \leq \frac{3}{4}.$$

Therefore, by the union bound the probability q_ℓ that $\ell \in [pn, 2pn]$ is at least

$$\begin{aligned} q_\ell &= \Pr[\ell \geq pn \cap \ell \leq 2pn] \\ &\geq 1 - \Pr[\ell < pn] - \Pr[\ell > 2pn] \\ &\geq 1 - \frac{3}{4} - \exp\left(-\frac{pn}{3}\right). \end{aligned}$$

Since we assume that $pn \geq 9$, we obtain

$$\exp\left(-\frac{pn}{3}\right) \leq \exp(-3) \leq 0.05$$

and hence,

$$q_\ell \geq 1 - \frac{3}{4} - \exp\left(-\frac{pn}{3}\right) \geq 0.2.$$

Therefore

$$q_\ell \geq 0.1 = \Omega(1).$$

□

We also state a similar lemma for the larger mutation rates (which are of a greater interest when we aim at escaping the local optimum).

Lemma 4. *Assume $p = \omega(\frac{1}{n})$. Then the number ℓ of the bits flipped by the mutation operator is in $[pn, \frac{5}{4}pn]$ with probability $q'_\ell = \frac{1}{4} - o(1)$.*

Proof. By the Chernoff bounds and by Theorem 10 in [Doe18] we have

$$\begin{aligned} q'_\ell &\geq 1 - \Pr[\ell < pn] - \Pr\left[\ell > \tfrac{5}{4}pn\right] \\ &\geq 1 - \tfrac{3}{4} - e^{-\frac{pn}{48}} = 1 - \tfrac{3}{4} - e^{-\omega(1)} = \tfrac{1}{4} - o(1). \end{aligned}$$

□

We also encounter random variables with hypergeometric distribution. A particular example of such random variable is the number ℓ_0 of zero-bits which are flipped by the mutation operator after the total number ℓ of the bits to flip is already chosen. This random variable follows a hypergeometric distribution with parameters n, $n - \mathrm{OM}(x)$ and ℓ. For this random variable the Chernoff bounds are also applicable [Doe20b, Theorem 1.10.25]. We use the following special case of this bound.

Lemma 5. *Let x be a bit string with exactly d zero-bits. Let ℓ_0 be the number of zero-bits of x which are flipped by the mutation operator of the $(1 + (\lambda, \lambda))$ GA after ℓ is chosen. Then the probability that $\ell_0 > (1 + \delta)\frac{\ell d}{n}$ is at most $\exp(-\frac{\delta^2 \ell d}{3n})$.*

3 Upper Bounds

In this section we analyse the $(1 + (\lambda, \lambda))$ GA with general parameters on $\textsc{Jump}_k$ and show upper bounds on its runtime. We recede from the standard parameter setting of the $(1 + (\lambda, \lambda))$ GA, since the intuition behind these parameters values (that is, the intent to have only a single bit flipped if we consequently apply mutation and crossover operators) suggests that they are not efficient to escape local optima.

We observe that the working principles of the $(1 + (\lambda, \lambda))$ GA, which were shown to be efficient on OneMax, are also successful on Jump. That is, we show that in the mutation phase the algorithm detects a beneficial mutation (which flips all missing zero-bits of the current individual to ones) by inspecting the fitness of the offspring and then crossover is capable to repair all wrong bit flips made in the mutation phase. We also show that the optimal mutation rate for performing a jump is $\sqrt{\frac{k}{n}}$, which is very different from the optimal one for optimizing OneMax ($\sqrt{\frac{1}{nd}}$ when we are in distance d from the optimum). At the same time the optimal crossover bias, which is also $\sqrt{\frac{k}{n}}$ is very similar to $\sqrt{\frac{d}{n}}$, which is optimal for OneMax.

We split our analysis into two parts. First we find the expected time the $(1 + (\lambda, \lambda))$ GA needs to perform a jump to the global optimum when it is already in the local optimum. Then we complete the story by considering the runtime until the $(1 + (\lambda, \lambda))$ GA gets to the local optimum starting in a random bit string.

We do not consider the case when $k = 1$, since $\textsc{Jump}_1$ coincides with OneMax, which is already well-studied in the context of the $(1 + (\lambda, \lambda))$ GA (see [DD18] for the full picture). We also omit considering too large values of k (namely, $k > \frac{n}{4}$) since they do not give much new insight about the $(1 + (\lambda, \lambda))$ GA, while they require more complicated arguments for our results to hold.

We also constrain ourselves to the case $p \geq \frac{2k}{n}$ so that once we get to the local optimum we have a decent probability to flip at least $2k$ bits. Since all mutation offspring have the same Hamming distance from the parent, this implies that an individual with k zero-bits flipped will have a better fitness than any other offspring and therefore selected as the winner of the mutation phase x'. Without this assumption an individual with all zero-bits flipped to one might occur in the fitness valley, thus it is not detected as the mutation phase winner. Hence, the jump to the global optimum becomes more challenging for the algorithm, which makes this parameter setting not really promising to be effective on multimodal functions.

3.1 Escaping the Local Optimum

In this section we analyse how the $(1 + (\lambda, \lambda))$ GA leaves the local optimum. Although by runtime we understand the time until the optimum is sampled, it is fair to consider the time until x becomes the optimum for at least two reasons. (i) By disregarding the event that the optimum is sampled in the mutation phase, we still get an upper bound on the runtime. (ii) Since the probability to sample the optimum in the mutation phase is small compared to the probability to sample the optimum in the crossover phase, we expect to lose only a little. Due to the elitist selection the only chance to leave the local optimum is to find the global optimum in one iteration. For this it is sufficient that the following two consecutive events happen.

1. The mutation phase winner x' has all k bits which are zero in the current individual x flipped to one.

2. The crossover winner y takes all k bits which are zero in x from x' and all bits which are zero in x' from x.

We first estimate the probability of the first event and then estimate the probability of the second event conditional on the first one.

We call the mutation phase *successful* if all k zero-bits of x are flipped to one in x' (and possibly some one-bits are flipped to zero) and the number ℓ of the flipped bits is at most $2pn$. We estimate the probability p_M of having a successful mutation phase in the following lemma.

Lemma 6. *Let $k \leq \frac{n}{4}$. If $p \geq \frac{2k}{n}$, then we have*

$$p_M \geq \frac{q_\ell}{2} \min\left\{1, \lambda_M \left(\frac{p}{2}\right)^k\right\},$$

where q_ℓ is as defined in Lemma 3, which is $\Theta(1)$.

Proof. If $\ell \geq 2k$ then we flip at least k one-bits in each mutant, hence the fitness of each mutant is at most $n-k$. Therefore, if there is at least one individual with all k zero-bits flipped, then this individual has a greater value of $\textsc{Jump}_k$ than any other individual which does not have all zero-bits flipped. Hence, such an individual is chosen as the mutation winner x'. Therefore, for a successful mutation phase it suffices that the following two events occur (in this order).

- The number of flipped bits ℓ is in $[pn, 2pn]$.
- The k zero-bits of x are among the ℓ chosen bits in at least one of the λ_M offspring. We call such offspring *good* in this proof.

By Lemma 3 the probability of the first event is $q_\ell \geq 0.1$. We condition on this event in the remainder. The probability $q_M(\ell)$ that one particular offspring is good is $\frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}}$. By the assumption that $p \geq \frac{2k}{n}$ we have

$$\begin{aligned} q_M(\ell) = \frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}} &= \frac{(n-k)!}{(\ell-k)!(n-\ell)!} \cdot \frac{\ell!(n-\ell)!}{n!} \\ &= \frac{\ell(\ell-1)\dots(\ell-k+1)}{n(n-1)\dots(n-k+1)} \geq \left(\frac{\ell-k}{n}\right)^k \geq \left(\frac{pn}{2n}\right)^k = \left(\frac{p}{2}\right)^k. \end{aligned}$$

The probability that at least one offspring is good is $1-(1-q_M(\ell))^{\lambda_M}$. By Lemma 1, we estimate

$$\begin{aligned} 1-(1-q_M(\ell))^{\lambda_M} &\geq \frac{1}{2}\min\{1, \lambda_M q_M(\ell)\} \\ &\geq \frac{1}{2}\min\left\{1, \lambda_M \left(\frac{p}{2}\right)^k\right\}. \end{aligned}$$

Therefore, we conclude

$$\begin{aligned} p_M &\geq \Pr[\ell \in [pn, 2pn]] \cdot \frac{1}{2} \min\left\{1, \lambda_M \left(\frac{p}{2}\right)^k\right\} \\ &\geq \frac{q_\ell}{2} \min\left\{1, \lambda_M \left(\frac{p}{2}\right)^k\right\}. \end{aligned}$$

□

Now we proceed with the crossover phase. We call the crossover phase *successful* (conditional on a successful mutation phase) if the winner y takes all bits which are zero in x' from x (where they are ones) and all k bits which are zero in x from x' (where they are ones). We denote the probability of a successful crossover phase by p_C.

Lemma 7. *Assume that $k \leq \frac{n}{4}$ and the mutation phase was successful. Then*

$$p_C \geq \frac{1}{2} \min\left\{1, \lambda_C c^k (1-c)^{2pn-k}\right\}.$$

Proof. To generate an optimal solution in one application of the crossover operator we need to take k particular bits from x' and $\ell - k$ particular bits from x. The probability q_C to generate such a crossover offspring is

$$q_C = c^k (1-c)^{\ell-k} \geq c^k (1-c)^{2pn-k},$$

since a successful mutation implies that $\ell \leq 2pn$. The probability to generate at least one such offspring is

$$\begin{aligned} p_C &= 1 - (1-q_C)^{\lambda_C} \geq 1 - \left(1 - c^k (1-c)^{2pn-k}\right)^{\lambda_C} \\ &\geq \frac{1}{2} \min\left\{1, \lambda_C c^k (1-c)^{2pn-k}\right\}, \end{aligned}$$

where the last inequality follows from Lemma 1. □

With Lemmas 6 and 7 we are capable of proving the upper bounds on the expected runtime until the $(1 + (\lambda, \lambda))$ GA escapes the local optimum. We estimate the runtime both in terms of the number fitness evaluations and the number of iterations, denoted by T_F and T_I respectively.

Theorem 8. *Let $k \leq \frac{n}{4}$. Assume that $p \geq \frac{2k}{n}$ and q_ℓ is as defined in Lemma 3. Then the expected runtime of $(1 + (\lambda, \lambda))$ GA on $\textsc{Jump}_k$ is*

$$E[T_I] \leq \frac{4}{q_\ell \min\left\{1, \lambda_M \left(\frac{p}{2}\right)^k\right\} \min\left\{1, \lambda_C c^k (1-c)^{2pn-k}\right\}}$$

iterations and

$$E[T_F] \leq \frac{4(\lambda_M + \lambda_C)}{q_\ell \min\left\{1, \lambda_M \left(\frac{p}{2}\right)^k\right\} \min\left\{1, \lambda_C c^k (1-c)^{2pn-k}\right\}}$$

fitness evaluations if the algorithm starts in the local optimum.

Proof. When the algorithm is in the local optimum it stays there until it moves to the optimum. During this time in each iteration it has the same probability P to move into the global optimum, which is the probability that a successful mutation phase is followed by a successful crossover phase:

$$P = p_M p_C \geq \frac{q_\ell}{2} \lambda_M \left(\frac{p}{2}\right)^k \cdot \frac{1}{2} \lambda_C c^k (1-c)^{2pn-k}.$$

Hence we obtain an expected optimization time in terms of iterations of

$$E[T_I] = \frac{1}{P} \leq \frac{4}{q_\ell \lambda_M \lambda_C \left(\frac{pc}{2}\right)^k (1-c)^{2pn-k}}.$$

In each iteration the $(1 + (\lambda, \lambda))$ GA performs exactly $\lambda_M + \lambda_C$ fitness evaluations, which gives us an expected number of

$$E[T_F] \leq \frac{4(\lambda_M + \lambda_C)}{q_\ell \lambda_M \lambda_C \left(\frac{pc}{2}\right)^k (1-c)^{2pn-k}}$$

fitness evaluations in total. □

With help of Theorem 8 we deliver good values for the parameters, namely $p = c = \sqrt{\frac{k}{n}}$ and $\lambda_C = \lambda_M = \sqrt{\frac{n}{k}}^k$. We omit the proof that these parameters yield the lowest upper bound (apart from optimizing the $e^{O(k)}$ factor), since it is just a routine work with complicated derivatives, but we state the runtime bounds resulting from these settings in the following corollary. In order to use this result in Section 3.2 we also formulate this theorem for general population sizes.

Corollary 9. *Let $k \in [2..\lfloor \frac{n}{4} \rfloor]$. Assume that $p = c = \sqrt{\frac{k}{n}}$ and $\lambda_M = \lambda_C = \lambda \leq 2^k \sqrt{\frac{n}{k}}^k$. Then the expected runtime of $(1 + (\lambda, \lambda))$ GA on $\textsc{Jump}_k$ is $E[T_F] \leq n^k k^{-k} e^{O(k)} \lambda^{-1}$ fitness evaluations and $E[T_I] \leq n^k k^{-k} e^{O(k)} \lambda^{-2}$ iterations, if it starts in a local optimum of $\textsc{Jump}_k$. For $\lambda = \sqrt{\frac{n}{k}}^k$ these bounds are $E[T_I] \leq e^{O(k)}$ and $E[T_F] \leq \sqrt{\frac{n}{k}}^k e^{O(k)}$.*

Proof. With $\lambda \leq 2^k \sqrt{\frac{n}{k}}^k$ we have $\lambda(\frac{p}{2})^k \leq 1$ and $\lambda c^k (1-c)^{2pn-k} \leq 1$. Consequently, by Theorem 8 we have

$$\begin{aligned}
E[T_I] &\leq \frac{4}{q_\ell \lambda^2 \left(\frac{k}{2n}\right)^k \left(1-\sqrt{\frac{k}{n}}\right)^{2\sqrt{kn}-k}} \\
&= \frac{2^{k+2}\left(\frac{n}{k}\right)^k}{q_\ell \lambda^2 \left(1-\sqrt{\frac{k}{n}}\right)^{2\sqrt{kn}-k}} \\
&\leq \frac{2^{k+2}}{q_\ell \lambda^2}\left(\frac{n}{k}\right)^k \left(1-\sqrt{\frac{k}{n}}\right)^{-2\sqrt{kn}} \\
&\leq \frac{2^{k+2}}{q_\ell \lambda^2}\left(\frac{n}{k}\right)^k \left(1-\sqrt{\frac{k}{n}}\right)^{-\sqrt{\frac{n}{k}}2k},
\end{aligned}$$

Where $q_\ell \in [0.1, 1]$ is a constant defined in Lemma 3. By the estimate $(1-x)^{-\frac{1}{x}} \leq 4$ which holds for all $x \in (0, \frac{1}{2}]$ and by $\sqrt{\frac{k}{n}} \leq \sqrt{\frac{n}{4n}} = \frac{1}{2}$ we have

$$\begin{aligned}
E[T_I] &\leq \frac{2^{k+2}}{q_\ell \lambda^2}\left(\frac{n}{k}\right)^k \left(1-\sqrt{\frac{k}{n}}\right)^{-\sqrt{\frac{n}{k}}2k} \\
&\leq \frac{2^{k+2}}{q_\ell \lambda^2}\left(\frac{n}{k}\right)^k 4^{2k} \qquad (1) \\
&= \left(\frac{n}{k}\right)^k \frac{e^{O(k)}}{\lambda^2}.
\end{aligned}$$

The expected number of fitness evaluations is $\lambda_M + \lambda_C = 2\lambda$ times greater, hence we have

$$E[T_F] \leq \left(\frac{n}{k}\right)^k \frac{e^{O(k)}}{\lambda}. \qquad (2)$$

Putting $\lambda = \sqrt{\frac{n}{k}}^k$ into (1) and (2) we have $E[T_I] \leq e^{O(k)}$ and $E[T_F] \leq \sqrt{\frac{n}{k}}^k e^{O(k)}$. □

In the following corollary we show a wide range of the parameters, which yield a better upper bound than the mutation-based algorithm (apart from the $e^{O(k)}$ factor) for the sub-linear jump sizes. We do not show it for $k = \Theta(1)$, since in this case the upper bound given by Corollary 9 is $e^{O(k)}$, which is not better than the runtime of best mutation-based EAs.

Corollary 10. *Let $k \geq 2$ and $k = o(n)$. Assume that $p = \omega(\frac{k}{n})$, $c = \omega(\frac{k}{n})$ and $pc = O(\frac{k}{n})$. Define $\alpha := \lambda_M(\frac{p}{2})^k$ and $\beta := \lambda_C c^k(1-c)^{2pn-k}$. If α and β are at most one and $\alpha = \omega((\frac{k}{nc})^k)$ and $\beta = \omega((\frac{2k}{pn})^k)$, then the expected number of fitness evaluations until the $(1+(\lambda,\lambda))$ GA reaches the global optimum starting from the local optimum of* $\textsc{Jump}_k$ *is*

$$E[T_F] = o\left(\left(\frac{n}{k}\right)^k\right)e^{O(k)}.$$

Before we prove the corollary we shortly discuss how one can choose the parameters that give us $o\left(\left(\frac{n}{k}\right)^k\right)e^{O(k)}$ runtime with Corollary 10. First we should choose p. It can be any value which is $\omega(\frac{k}{n})$ and which is $o(1)$. Then with the chosen value of p we can choose any c which is on the one hand $\omega(\frac{k}{n})$, but on the other hand $O(\frac{k}{n}p^{-1})$. Note that the closer p is to $\Theta(1)$, the smaller the range for c (thus we could not choose $p = \Theta(1)$, since in this case we cannot simultaneously satisfy $c = \omega(\frac{k}{n})$ and $pc = O(\frac{k}{n})$).

After we determine p and c, we can choose λ_M and λ_C. For λ_M the upper bound for the possible range is $(\frac{p}{2})^{-k}$, which follows from condition $\alpha = \lambda_M(\frac{p}{2})^k \leq 1$. The lower bound for λ_M is $\omega((\frac{2k}{pcn})^k)$, which follows from the condition $\alpha = \omega((\frac{k}{nc})^k)$. For λ_C we have similarly obtained bounds, which are $c^{-k}(1-c)^{-(2pn-k)}$ and $\omega((\frac{2k}{pcn})^k)(1-c)^{-(2pn-k)}$.

Generally, the choice of the λ_M and λ_C should be made in such way that they were as close as possible to the inverse probabilities of creating a good offsprings in the mutation and crossover phases respectively. By Lemma 1 this choice yields a $\Theta(1)$ probability of a successful iteration. Any smaller population size reduces this probability (usually greater than it reduces the cost of one iteration), while any greater population size only increases the cost of each iteration without significantly increasing the success probability.

Proof of Corollary 10. Since α and β are at most one, the runtime given by Theorem 8 is simplified to

$$\begin{aligned} E[T_F] &\leq \frac{4(\lambda_M + \lambda_C)}{q_\ell \lambda_M \lambda_C \left(\frac{pc}{2}\right)^k (1-c)^{2pn-k}} \\ &= \frac{4}{q_\ell \alpha c^k (1-c)^{2pn-k}} + \frac{4}{q_\ell \beta \left(\frac{p}{2}\right)^k}. \end{aligned}$$

We want both terms to be $o((\frac{n}{k})^k)e^{O(k)}$. For the first term it is sufficient if the three following conditions hold. (i) $c^k = \omega((\frac{k}{n})^k)$, which holds if $c = \omega(\frac{k}{n})$. (ii) $(1-c)^{2pn-k} = e^{-O(k)}$. For this it is sufficient to have $pc = O(\frac{k}{n})$, since

then we have

$$(1-c)^{2pn-k} = (1-c)^{\frac{1}{c}(2pcn-kc)} \geq \left(\frac{1}{4}\right)^{2pcn} = \left(\frac{1}{4}\right)^{O(k)} = e^{-O(k)}.$$

(iii) $\alpha = \omega((\frac{k}{nc})^k)$, since this implies

$$\frac{4}{q_\ell \alpha c^k (1-c)^{2pn-k}} = \frac{o\Big(\big(\frac{nc}{k}\big)^k\Big)}{c^k e^{-O(k)}} = o\bigg(\Big(\frac{n}{k}\Big)^k\bigg) e^{O(k)}.$$

For the second term it is enough that the following two conditions hold. (i) $(\frac{p}{2})^k = \omega((\frac{k}{n})^k)$, for which it is sufficient to have $p = \omega(\frac{k}{n})$. (ii) β should not be too small, namely, $\beta = \omega((\frac{2k}{pn})^k)$, since it implies

$$\frac{4}{q_\ell \beta \big(\frac{p}{2}\big)^k} = o\bigg(\Big(\frac{pn}{2k}\Big)^k\bigg)\Big(\frac{p}{2}\Big)^{-k} = o\bigg(\Big(\frac{n}{k}\Big)^k\bigg). \qquad \square$$

We find it interesting to show that the standard parameter setting does not give us such a good upper bound. Note, however, that our lower bound given in Theorem 16 allows the actual runtime of the $(1+(\lambda,\lambda))$ GA with standard parameters to be better.

Theorem 11. *Let $k \in [2..\lfloor \frac{n}{4} \rfloor]$. Assume that $p = \frac{\lambda}{n}$, $c = \frac{1}{\lambda}$ and $\lambda_M = \lambda_C = \lambda$ for some $\lambda \in [2k..n]$.*

If $\lambda \leq (2n)^{\frac{k}{k+1}}$, then the expected runtime of $(1+(\lambda,\lambda))$ GA on $\textsc{Jump}_k$ is $E[T_I] = O(2^k n^k \lambda^{-2})$ iterations and $E[T_F] = O(2^k n^k \lambda^{-1})$ fitness evaluations.

If $\lambda \geq (2n)^{\frac{k}{k+1}}$, then the expected runtime of $(1+(\lambda,\lambda))$ GA on $\textsc{Jump}_k$ is $E[T_I] = O(\lambda^{k-1})$ iterations and $E[T_F] = O(\lambda^k)$ fitness evaluations.

Proof. Since the standard parameter setting with $\lambda \geq 2k$ satisfies the conditions of Theorem 8, we obtain

$$E[T_I] \leq \frac{4}{q_\ell \min\Big\{1, \lambda\big(\frac{\lambda}{2n}\big)^k\Big\} \min\Big\{1, \lambda \frac{1}{\lambda^k}\big(1-\frac{1}{\lambda}\big)^{2\lambda-k}\Big\}}.$$

Note that the second minimum in the denominator is always equal to its second argument. To understand the first minimum we consider two cases.

Case 1. When $\lambda \leq (2n)^{\frac{k}{k+1}}$, the first minimum is equal to its second argument. Therefore, we have

$$E[T_I] \leq \frac{4}{q_\ell \lambda^2 \left(\frac{\lambda}{2n\lambda}\right)^k \left(1-\frac{1}{\lambda}\right)^{2\lambda-k}}$$
$$\leq \frac{4(2n)^k}{q_\ell \lambda^2 \left(1-\frac{1}{\lambda}\right)^{2\lambda}} = O\left(\frac{(2n)^k}{\lambda^2}\right).$$

In each iteration the $(1+(\lambda,\lambda))$ GA performs 2λ fitness evaluations, thus we have

$$E[T_F] = 2\lambda E[T_I] = O\left(\frac{(2n)^k}{\lambda}\right).$$

Case 2. When $\lambda \geq (2n)^{\frac{k}{k+1}}$, first minimum is equal to its first argument. Therefore, we have

$$E[T_I] \leq \frac{4}{q_\ell \lambda \frac{1}{\lambda^k} \left(1-\frac{1}{\lambda}\right)^{2\lambda-k}}$$
$$\leq \frac{4\lambda^{k-1}}{q_\ell \left(1-\frac{1}{\lambda}\right)^{2\lambda}} = O\left(\lambda^{k-1}\right).$$

In each iteration the $(1+(\lambda,\lambda))$ GA performs 2λ fitness evaluations, thus we have

$$E[T_F] = 2\lambda E[T_I] = O\left(\lambda^k\right).$$

Note that this upper bound is minimized when $\lambda = (2n)^{\frac{k}{k+1}}$ (rounded up or down), in this case the expected runtime is

$$E[T_F] = O\left((2n)^{\frac{k^2}{k+1}}\right) = O\left((2n)^{k-1+\frac{1}{k+1}}\right).$$

□

3.2 Reaching the Local Optimum

In the previous section we showed that the $(1+(\lambda,\lambda))$ GA with non-standard parameters setting can find the global optimum of $\textsc{Jump}_k$ much faster than any standard mutation-based algorithms if the algorithms are started in the local optimum. However, the non-standard parameter setting includes an unnaturally large population size, which makes each iteration costly. At the

same time, there is no guarantee that we increase the fitness by much in one iteration, which makes us pay with many fitness evaluations before we reach the local optimum. Hence we question how much the runtime with this parameter setting increases when we start at a random bit string. In this section we show that slightly changing the parameters we can obtain the runtime which is only by a $\sqrt{n}$ factor greater than the runtime when we start in the local optimum. The main result of this section is the following theorem.

Theorem 12. *Let $k \leq \frac{n}{4}$. If $\lambda_M = \lambda_C = \frac{1}{\sqrt{n}}\sqrt{\frac{n}{k}}^k$ and $p = c = \sqrt{\frac{k}{n}}$, then the expected runtime of the $(1 + (\lambda, \lambda))$ GA with any initialization on the $\textsc{Jump}_k$ function is at most $\sqrt{n}\sqrt{\frac{n}{k}}^k e^{O(k)}$ fitness evaluations.*

To prove Theorem 12 we first analyse the runtime until the $(1 + (\lambda, \lambda))$ GA reaches the local optimum of $\textsc{Jump}_k$.

Theorem 13. *Let $\lambda_M = \lambda_C = \lambda \geq \frac{n}{k}$ and $p = c = \sqrt{\frac{k}{n}}$. Then the expected time until the $(1 + (\lambda, \lambda))$ GA reaches the local optimum of $\textsc{Jump}_k$ with $k \leq \frac{n}{4}$ is at most $E[T_I] = ne^{O(k)}$ iterations.*

The main challenge in the proof of this theorem is that an offspring close to the optimum in the mutation phase can lie in the fitness valley and thus it is not selected as x'. In this section we call the mutation phase *successful* if the winner has at least one zero-bit of x flipped to one. If such a bit exists, we call it *critical* and we call a mutation phase offspring which has such a bit and does not lie in the fitness valley *good*. We write p_M to denote the probability of a successful mutation phase. We call the crossover phase *successful* if the winner of the crossover phase has inherited the critical bit from x' and all other bits are not changed compared to x (hence, the name "critical", since we need such bit to be taken from x' for a successful iteration). We write p_C to denote the probability of this event.

To prove Theorem 13 we show two auxiliary lemmas for the mutation and crossover phases respectively.

Lemma 14. *Let $k \leq \frac{n}{4}$. If $\lambda_M \geq \frac{\sqrt{n}}{k\sqrt{k}}$ and $\ell \in \left[\sqrt{nk}, \frac{5\sqrt{nk}}{4}\right]$, then $p_M = \Theta(1)$.*

Proof. We denote the current distance to the global optimum by d. In order to create a good mutant we need to flip at least one zero-bit, but we also need not to flip too many zero-bits (namely, not more than $\frac{\ell+d-k}{2}$) not to reach the fitness valley. Let ℓ_0 be the number of the flipped zero-bits in a fixed mutant. Then the probability q_M that the mutant is good is

$$q_M = 1 - \Pr[\ell_0 = 0] - \Pr\left[\ell_0 > \frac{\ell + d - k}{2}\right]$$

We estimate the probability that we have not flipped a single zero-bit using Lemma 1.

$$\begin{aligned}\Pr[\ell_0 = 0] = \frac{\binom{\mathrm{OM}(x)}{\ell}}{\binom{n}{\ell}} &\leq \left(\frac{n-k}{n}\right)^{\ell} \leq \left(1-\frac{k}{n}\right)^{\sqrt{nk}} \\ &= 1 - \left(1 - \left(1-\frac{k}{n}\right)^{\sqrt{nk}}\right) \leq 1 - \frac{1}{2}\min\left\{1, \frac{k\sqrt{k}}{\sqrt{n}}\right\}\end{aligned}$$

To estimate the probability that we end up in the fitness valley, we use a Chernoff bound for the hyper-geometric distribution (Lemma 5). Note that the expected value of ℓ_0 is $\frac{\ell d}{n}$. Hence, we have

$$\begin{aligned}\Pr\left[\ell_0 > \frac{\ell+d-k}{2}\right] &= \Pr\left[\ell_0 > \left(1 + \left(\frac{\ell+d-k}{2}\cdot\frac{n}{\ell d} - 1\right)\right)\frac{\ell d}{n}\right] \\ &\leq \exp\left(-\frac{1}{3}\left(\frac{\ell+d-k}{2}\cdot\frac{n}{\ell d} - 1\right)^2 \frac{\ell d}{n}\right).\end{aligned}$$

Considering the argument of the exponential as a function of d and computing its derivative (we omit tedious details), one can see that its value is maximized either at $d = \frac{(\ell-k)n}{n-2\ell}$, if this value lies in $[k+1, n]$, or at the bounds of this interval.

For $d = \frac{(\ell-k)n}{n-2\ell}$ we have

$$\begin{aligned}\Pr\left[\ell_0 > \frac{\ell+d-k}{2}\right] &\leq \exp\left(-\frac{1}{3}\left(\frac{\ell-k+\frac{(\ell-k)n}{n-2\ell}}{2}\cdot\frac{n}{\ell\frac{(\ell-k)n}{n-2\ell}} - 1\right)^2 \frac{\ell\frac{(\ell-k)n}{n-2\ell}}{n}\right) \\ &= \exp\left(-\frac{1}{3}\left(\frac{\left(1+\frac{n}{n-2\ell}\right)n}{2\ell\frac{n}{n-2\ell}} - 1\right)^2 \frac{\ell(\ell-k)}{n-2\ell}\right) \\ &= \exp\left(-\frac{1}{3}\left(\frac{n-\ell}{\ell} - 1\right)^2 \frac{\ell(\ell-k)}{n-2\ell}\right) \\ &= \exp\left(-\frac{1}{3}\cdot\frac{(n-2\ell)(\ell-k)}{\ell}\right)\end{aligned}$$

Since $n \geq 4k$ and $\ell \geq \sqrt{nk} \geq 2k$, we have $(\ell-k) \geq \frac{\ell}{2}$. Hence, if $\ell \leq \frac{n}{3}$, we have

$$\Pr\left[\ell_0 > \frac{\ell+d-k}{2}\right] \leq \exp\left(-\frac{(n-2\ell)\frac{\ell}{2}}{3\ell}\right) \leq \exp\left(-\frac{n-\frac{2n}{3}}{6}\right) \leq \exp\left(-\frac{n}{18}\right).$$

Otherwise, if $\ell > \frac{n}{3}$, then $d = \frac{(\ell-k)n}{n-2\ell} \le n$ only if $(n-2\ell) \ge (\ell-k)$. Therefore,

$$\Pr\left[\ell_0 > \frac{\ell+d-k}{2}\right] \le \exp\left(-\frac{1}{3}\cdot\frac{(\ell-k)^2}{\ell}\right) \le \exp\left(-\frac{(\ell/2)^2}{3\ell}\right)$$
$$= \exp\left(-\frac{\ell}{12}\right) < \exp\left(-\frac{n}{36}\right).$$

For $d = k+1$ we have

$$\Pr\left[\ell_0 > \frac{\ell+d-k}{2}\right] \le \exp\left(-\frac{1}{3}\left(\frac{(\ell+1)n}{2\ell(k+1)} - 1\right)^2 \frac{\ell(k+1)}{n}\right)$$
$$(\text{since } \tfrac{\ell+1}{\ell} > 1 \text{ and } \tfrac{n}{2(k+1)} > 1) \le \exp\left(-\frac{1}{3}\left(\frac{n}{2(k+1)} - 1\right)^2 \frac{\ell(k+1)}{n}\right)$$
$$= \exp\left(-\frac{\ell(n-2(k+1))^2}{12(k+1)n}\right)$$
$$(\text{since } \ell \ge 2k \text{ and } k \le \frac{n}{4}) \le \exp\left(-\frac{2k}{12(k+1)}\cdot\frac{\left(n-\frac{n}{2}-2\right)^2}{n}\right)$$
$$\le \exp\left(-\frac{1}{9}\cdot\left(\frac{n}{4}-2\right)\right)$$
$$\le \exp\left(-\frac{n}{36}+\frac{2}{9}\right),$$

if n is large enough.

For $d = n$ we have

$$\Pr\left[\ell_0 > \frac{\ell+d-k}{2}\right] \le \exp\left(-\frac{1}{3}\left(\frac{(\ell+n-k)n}{2\ell n} - 1\right)^2 \ell\right)$$
$$\le \exp\left(-\frac{1}{3}\left(\frac{\ell+n-k-2\ell}{2\ell}\right)^2 \ell\right)$$
$$= \exp\left(-\frac{(n-k-\ell)^2}{12\ell}\right)$$
$$(\text{since } k \le \frac{n}{4} \text{ and } \ell \le \tfrac{5}{4}\sqrt{kn} \le \tfrac{5}{8}n) \le \exp\left(-\frac{(n/8)^2}{12\cdot(5n/8)}\right) = \exp\left(-\frac{n}{480}\right).$$

Therefore, we estimate the probability to create a good mutant as

$$\begin{aligned} q_M &\geq 1 - \Pr[\ell_0 = 0] - \Pr\left[\ell_0 > \frac{\ell + d - k}{2}\right] \\ &\geq \frac{1}{2}\min\left\{1, \frac{k\sqrt{k}}{\sqrt{n}}\right\} - e^{-\frac{n}{480}} \geq \frac{1}{4}\min\left\{1, \frac{k\sqrt{k}}{\sqrt{n}}\right\}, \end{aligned}$$

where the last inequality holds when n is at least some sufficiently large constant. If $\frac{k\sqrt{k}}{n} > 1$, this probability is already $\Omega(1)$ and hence $p_M = \Omega(1)$.

Otherwise, by Lemma 1 we compute

$$p_M = 1 - (1 - q_M)^{\lambda}_M \geq \frac{1}{2}\min\{1, \lambda_M q_M\}.$$

Since $\lambda_M \geq \frac{\sqrt{n}}{k\sqrt{k}}$, we have $\lambda_M q_M \geq 1$ and therefore, $p_M = \Omega(1)$.

Finally, we note that for constant n the probability q_M is still positive, and hence $\Omega(1)$. □

We proceed with a lemma for the crossover phase.

Lemma 15. *Let $k \leq \frac{n}{2}$. Assume that $c = \sqrt{\frac{k}{n}}$, $\lambda_C \geq \sqrt{\frac{n}{k}}$ and $\ell \in \left[\sqrt{nk}, \frac{5\sqrt{nk}}{4}\right]$, and there is at least one critical bit in x'. Then $p_C = e^{-O(k)}$.*

Proof. To have a successful crossover offspring it is sufficient to take one critical bit from x' and all other different bits from x. Thus the probability q_C of generating one superior crossover offspring is

$$\begin{aligned} q_C = c(1-c)^{\ell-1} &\geq \sqrt{\frac{k}{n}}\left(1 - \sqrt{\frac{k}{n}}\right)^{\frac{5\sqrt{nk}}{4}-1} \\ &\geq \sqrt{\frac{k}{n}}\left(1 - \sqrt{\frac{k}{n}}\right)^{\sqrt{\frac{n}{k}}\frac{5k}{4}} = \sqrt{\frac{k}{n}}e^{-\Theta(k)}. \end{aligned}$$

Since we need only one of the $\lambda_C \geq \sqrt{\frac{n}{k}}$ offspring to be superior, by Lemma 1 we have

$$p_C = 1 - (1 - q_C)^{\lambda_C} \geq \frac{1}{2}\min\left\{1, \lambda_C\sqrt{\frac{k}{n}}e^{-\Theta(k)}\right\} = e^{-\Theta(k)}. \qquad \square$$

Now we are in position to prove Theorem 13

Proof of Theorem 13. We denote the probability to increase fitness in one iteration by P and we estimate this probability as follows.

$$P \geq \Pr\left[\ell \in \left[pn, \frac{5pn}{4}\right]\right] \cdot p_M \cdot p_C.$$

By Lemmas 4, 14, and 15 we have

$$P \geq \left(\tfrac{1}{4} - o(1)\right) \cdot \Omega(1) \cdot e^{-O(k)} = e^{-O(k)}.$$

Therefore the expected runtime (in terms of iterations) until the $(1 + (\lambda, \lambda))$ GA reaches the local optimum of $\textsc{Jump}_k$ is

$$E[T_I] \leq \sum_{i=0}^{n-k} \frac{1}{P} \leq ne^{O(k)}. \qquad \square$$

Finally, we prove the main result of this section, Theorem 12.

Proof of Theorem 12. By Theorems 9 and 13 the upper bound on the total number of fitness evaluations of the $(1 + (\lambda, \lambda))$ GA with random initialization is

$$E[T_F] \leq \lambda n e^{O(k)} + \sqrt{\frac{n}{k}}^k \frac{e^{O(k)}}{\lambda}.$$

With $\lambda = \frac{1}{\sqrt{n}}\sqrt{\frac{n}{k}}^k$, we have

$$\begin{aligned} E[T_F] &\leq \frac{1}{\sqrt{n}}\sqrt{\frac{n}{k}}^k \, n e^{\Theta(k)} + \sqrt{\frac{n}{k}}^k \frac{e^{\Theta(k)}}{\frac{1}{\sqrt{n}}\sqrt{\frac{n}{k}}^k} \\ &\leq \sqrt{n}\sqrt{\frac{n}{k}}^k \, e^{\Theta(k)} + \sqrt{n}\sqrt{\frac{n}{k}}^k \, e^{\Theta(k)} = \sqrt{n}\sqrt{\frac{n}{k}}^k \, e^{\Theta(k)}. \end{aligned} \qquad \square$$

We note that $\lambda = \frac{1}{\sqrt{n}}\sqrt{\frac{n}{k}}^k$ is the value which minimizes our upper bound apart from the $e^{\Theta(k)}$ factor. We omit the proof of this fact, since it trivially follows from the minimization of a function $f(x) = ax + \frac{b}{x}$ via analysis of its derivative.

4 Lower Bounds

In this section, we show that the parameters we chose in the previous section for the optimization starting in the local optimum are asymptotically optimal

(apart from $e^{O(k)}$ factors), that is, that no choice of the parameters λ_M, λ_C, p, and c leads to an asymptotically better runtime (apart from $e^{O(k)}$ factors). This in particular shows that our analysis in the previous section is tight, that is, that the runtime proven in Corollary 9 is of the right asymptotic order of magnitude (apart from $e^{O(k)}$ factors).

Theorem 16. *The probability P that the $(1+(\lambda,\lambda))$ GA finds the optimum of $\textsc{Jump}_k$ with $k \leq \frac{n}{4}$ in one iteration if the current individual is in the local optimum is at most $\lambda_M(\lambda_C+1)\left(\frac{k}{2n}\right)^k$.*

Before we prove Theorem 16 we introduce and prove the following auxiliary lemma. A similar (but less precise) result can be distilled from the proofs of Theorem 4.1 and Corollary 4.2 in [DLMN17], but we give a short proof which also delivers a more precise estimate.

Lemma 17. *For all $n \in \mathbb{N}$, all $k \in [1..n-1]$ and all $x \in [0,1]$ we have $x^k(1-x)^{n-k} \leq (\frac{k}{n})^k(1-\frac{k}{n})^{n-k}$. If $n \geq 2k$, then we also have $x^k(1-x)^{n-k} \leq (\frac{k}{2n})^k$.*

Proof. Consider the function $f(x) = x^k(1-x)^{n-k}$. It is smooth in $[0,1]$, therefore its maxima are in the boundaries of the interval or in the roots of its derivative. Since $f(0) = f(1) = 0$ and $f(x) > 0$ for all $0 < x < 1$, the boundary values cannot be maxima. We compute the derivative as follows.

$$\begin{aligned} f'(x) &= kx^{k-1}(1-x)^{n-k} - (n-k)x^k(1-x)^{n-k-1} \\ &= x^{k-1}(1-x)^{n-k-1}(k-nx). \end{aligned}$$

The roots of the derivative are in $x = 0$, $x = 1$ and $x = \frac{k}{n}$. Hence the maximum can be reached only in $x = \frac{k}{n}$, and the value of f at this point is $f(\frac{k}{n}) = (\frac{k}{n})^k(1-\frac{k}{n})^{n-k}$. If we also have $2k \leq n$, then furthermore

$$\begin{aligned} f(x) \leq f\left(\frac{k}{n}\right) = \left(\frac{k}{n}\right)^k\left(1-\frac{k}{n}\right)^{n-k} &= \left(\frac{k}{n}\right)^k\left(\left(1-\frac{k}{n}\right)^{\frac{n}{k}-1}\right)^k \\ \leq \left(\frac{k}{n}\right)^k 2^{-k} &= \left(\frac{k}{2n}\right)^k. \end{aligned}$$

□

Now we are in position to prove Theorem 16.

Proof of Theorem 16. We use the precise expression of the probability P to go from the local to the global optimum in one iteration, which is

$$P = \sum_{\ell=0}^{n} p_\ell p_M(\ell) p_C(\ell), \tag{3}$$

where p_ℓ is the probability to choose ℓ bits to flip, $p_M(\ell)$ is the probability of a successful mutation phase conditional on the chosen ℓ, and $p_C(\ell)$ is the probability of a successful crossover phase conditional on the chosen ℓ and on the mutation phase being successful. In contrast to the upper bounds, where we have shown a lower bound on the sum of the terms for $\ell \in [np..2np]$, now we aim at giving the upper bound on the whole sum.

Since $\ell \sim \text{Bin}(n,p)$, we have $p_\ell = \binom{n}{\ell} p^\ell (1-p)^{n-\ell}$. The probability of a successful mutation phase depends on the chosen ℓ. If $\ell < k$, then it is impossible to flip all k zero-bits, hence $p_M(\ell) = 0$. For larger ℓ the probability to create a good offspring in a single application of the mutation operator is $q_M(\ell) = \binom{n-k}{\ell-k} / \binom{n}{\ell}$. If $\ell \in [k+1..2k-1]$, then any good offspring occurs in the fitness valley and has worse fitness than any other offspring that is not good. Hence, in order to have a successful mutation phase we need all λ_M offspring to be good. Therefore, the probability of a successful mutation phase is $(q_M(\ell))^{\lambda_M}$. For $\ell = k$ and $\ell \geq 2k$ we are guaranteed to choose a good offspring as the winner of the mutation phase if there is at least one. Therefore, the mutation phase is successful with probability $p_M(\ell) = 1 - (1 - q_M(\ell))^{\lambda_M}$.

If $\ell = k$ and the mutation phase is successful, it implies that the optimum is already found and hence we assume $p_C(k) = 1$. Otherwise, we can create a good offspring in the crossover phase only if $\ell > k$. For this we need to take all k bits which are zero in x from x', and then take all $\ell - k$ one-bits which were flipped from x. The probability to do so in the creation of one offspring is $q_C(\ell) = c^k (1-c)^{\ell-k}$. Since we create λ_C offspring and at least one of them must have a fitness better than the fitness of x, the probability of a successful crossover phase is $p_C(\ell) = 1 - (1 - c^k(1-c)^{\ell-k})^{\lambda_C}$.

Putting these probabilities into (3) we obtain

$$
\begin{aligned}
P = &\binom{n}{k} p^k (1-p)^{n-k} \left(1 - \left(1 - \binom{n}{k}^{-1}\right)^{\lambda_M}\right) \\
&+ \sum_{\ell=k+1}^{2k-1} \binom{n}{\ell} p^\ell (1-p)^{n-\ell} \left(\frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}}\right)^{\lambda_M} \left(1 - (1 - c^k(1-c)^{\ell-k})^{\lambda_C}\right) \\
&+ \sum_{\ell=2k}^{n} \binom{n}{\ell} p^\ell (1-p)^{n-\ell} \left(1 - \left(1 - \frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}}\right)^{\lambda_M}\right) \left(1 - (1 - c^k(1-c)^{\ell-k})^{\lambda_C}\right).
\end{aligned} \tag{4}
$$

Using Bernoulli's inequality $(1+x)^\lambda \geq 1 + \lambda x$ valid for all $x \geq -1$ and $\lambda \geq 1$, we estimate the two terms of type $1 - (1-p)^\lambda$ by $1 - (1-p)^\lambda \leq 1 - (1 - \lambda p) = \lambda p$. Note that an equivalent estimate could have been obtained by applying a union bound in the probabilistic setting that gave rise to these two expressions.

We then note that, trivially, we have

$$
\left(\frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}}\right)^{\lambda_M} \leq \lambda_M \frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}},
$$

which allows to uniformly estimate the terms for $\ell \in [k+1..2k-1]$ and $\ell \geq 2k$.

With these two estimates, we obtain

$$
\begin{aligned}
P \leq &\binom{n}{k} p^k (1-p)^{n-k} \lambda_M \binom{n}{k}^{-1} \\
&+ \sum_{\ell=k+1}^{n} \binom{n}{\ell} p^\ell (1-p)^{n-\ell} \lambda_M \frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}} \lambda_C c^k (1-c)^{\ell-k}.
\end{aligned}
$$

First, we consider the first term which corresponds to $\ell = k$. By Lemma 17 we have

$$
\binom{n}{k} p^k (1-p)^{n-k} \lambda_M \binom{n}{k}^{-1} = \lambda_M p^k (1-p)^{n-k} \leq \lambda_M \left(\frac{k}{2n}\right)^k. \tag{5}
$$

For the rest of the expression we argue as follows.

$$\begin{aligned}
&\sum_{\ell=k+1}^{n} \binom{n}{\ell} p^\ell (1-p)^{n-\ell} \lambda_M \frac{\binom{n-k}{\ell-k}}{\binom{n}{\ell}} \lambda_C c^k (1-c)^{\ell-k} \\
&\quad = \sum_{\ell=k+1}^{n} \lambda_M \lambda_C \binom{n-k}{\ell-k} p^\ell (1-p)^{n-\ell} c^k (1-c)^{\ell-k} \\
&\quad = \lambda_M \lambda_C (1-p)^{n-k} (pc)^k \sum_{\ell=k+1}^{n} \binom{n-k}{\ell-k} \left(\frac{p(1-c)}{1-p}\right)^{\ell-k} \qquad (6) \\
&\quad = \lambda_M \lambda_C (1-p)^{n-k} (pc)^k \sum_{i=1}^{n-k} \binom{n-k}{i} \left(\frac{p(1-c)}{1-p}\right)^{i} \\
&\quad \leq \lambda_M \lambda_C (1-p)^{n-k} (pc)^k \left(1 + \frac{p(1-c)}{1-p}\right)^{n-k} \\
&\quad = \lambda_M \lambda_C (pc)^k (1-pc)^{n-k}.
\end{aligned}$$

Since $pc \in [0,1]$ and $2k \leq n$, by Lemma 17 this expression is at most $\lambda_M \lambda_C (\frac{k}{2n})^k$. Therefore, we conclude that

$$P \leq \lambda_M \left(\frac{k}{2n}\right)^k + \lambda_M \lambda_C \left(\frac{k}{2n}\right)^k = \lambda_M (\lambda_C + 1) \left(\frac{k}{2n}\right)^k.$$

□

Theorem 16 lets us show in the following corollary that the parameters stated in Corollary 9 ($\lambda_M = \lambda_C = (\frac{n}{k})^{k/2}$ and $p = c = \sqrt{\frac{k}{n}}$) which minimize the upper bound give us an optimal runtime (apart from an $e^{\theta(k)}$ factor).

Corollary 18. *The expected runtime of the $(1 + (\lambda, \lambda))$ GA with any parameters on $\textsc{Jump}_k$ is at least $2(\frac{2n}{k})^{k/2} - 1$, if it starts in the local optimum of $\textsc{Jump}_k$. This is of the same asymptotic order (apart from an $e^{\theta(k)}$ factor) as the upper bound shown in Corollary 9 for the parameters $\lambda_M = \lambda_C = (\frac{n}{k})^{k/2}$ and $p = c = \sqrt{\frac{k}{n}}$*

Proof. Since the probability P to find the global optimum in one iteration is at most 1, by Theorem 16 we have

$$E[T_F] \geq \frac{\lambda_M + \lambda_C}{P} \geq \max\left\{ \lambda_M + \lambda_C, \frac{\lambda_M + \lambda_C}{\lambda_M (\lambda_C + 1) \left(\frac{k}{2n}\right)^k} \right\}. \qquad (7)$$

Consider the arguments of the maximum as functions of λ_M (and fix all other parameters). Then $(\lambda_M + \lambda_C)$ is strictly increasing in λ_M, while

$$\frac{\lambda_M + \lambda_C}{\lambda_M(\lambda_C+1)\left(\frac{k}{2n}\right)^k} = \frac{\lambda_C}{(\lambda_C+1)\left(\frac{k}{2n}\right)^k} + \frac{\lambda_C}{\lambda_M(\lambda_C+1)\left(\frac{k}{2n}\right)^k}$$

is strictly decreasing. Therefore, the maximum in (7) is minimized when both its arguments are equal. This condition is satisfied only when $\lambda_M = (\frac{2n}{k})^k(\lambda_C+1)^{-1}$, which yields the following lower bound on the runtime.

$$E[T_F] \geq \lambda_M + \lambda_C \geq \lambda_C + \frac{\left(\frac{2n}{k}\right)^k}{\lambda_C+1}.$$

Considering this as a function of λ_C and studying its derivative one can see that this lower bound is minimized when $\lambda_C = (\frac{2n}{k})^{k/2} - 1$. This implies that the minimal lower bound on the runtime is reached with $\lambda_M = (\frac{2n}{k})^k(\lambda_C + 1)^{-1} = (\frac{2n}{k})^{k/2}$ and is equal to

$$\begin{aligned} E[T_F] \geq \lambda_M + \lambda_C &= \left(\frac{2n}{k}\right)^{k/2} + \left(\frac{2n}{k}\right)^{k/2} - 1 \\ &= 2\left(\frac{2n}{k}\right)^{k/2} - 1 = \left(\frac{n}{k}\right)^{k/2} e^{\Theta(k)}. \end{aligned}$$

This is of the same asymptotical order (apart from an $e^{\theta(k)}$ factor) as the upper bound $(\frac{n}{k})^{k/2}e^{\Theta(k)}$ for parameters $\lambda_M = \lambda_C = (\frac{n}{k})^{k/2}$ and $p = c = \sqrt{\frac{k}{n}}$ shown in Corollary 9.

□

In this section, we have shown that we determined in Corollary 9 an asymptotically optimal parameter setting for leaving the local optimum of jump functions. We leave open the question if we also used the best parameters in Theorem 12, that is, when starting with a random solution. This is also an interesting question, but it is harder to answer due to the trade-off between optimizing the OneMax-type part of the optimization process and the part leaving the local optimum. In addition, we believe that the question we did answer, how to optimally leave the local optimum, is more interesting from the application point of view. We could imagine that when solving several similar instances of a difficult optimization problem, just by analyzing the optimization processes, one can obtain a rough estimate on the number of bits that need to be flipped to leave the hardest local optima. With this information, the parameters determined in Corollary 9 would be a reasonable

starting point for optimizing the parameters of the algorithm. In constrast to this, we doubt that in a practical problem one is able to understand both the structure of the local optima and the easy parts of the fitness landscape sufficiently well that then a trade-off as done in Section 3.2 could reasonably be obtained.

5 Conclusion

In this first runtime analysis of the $(1 + (\lambda, \lambda))$ GA on a multimodal problem, we observed that this algorithm also has a runtime advantage over classic algorithms on multimodal objective functions, and a much more pronounced one. Whereas the advantage in the previous results on unimodal problems was a gain of a logarithmic factor, we have shown here a runtime that is almost the square root of the runtime of classic algorithms.

For the $(1 + (\lambda, \lambda))$ GA to show such a good performance, its parameters have to be chosen differently from what was suggested in previous works, in particular, the mutation rate and crossover bias have to be larger. We developed some general suggestions (at the end of Section 3.1) that might ease the future use of this algorithm.

Being the first runtime analysis on a multimodal problem, this work leaves a number of questions unanswered. To highlight one of them, we note that we have not proven a matching lower bound for our runtime result. Lower bounds for algorithms with several parameters can be technically demanding as the corresponding analysis [DD18, Section 5] of the $(1 + (\lambda, \lambda))$ GA on OneMax shows. Hence such a result, despite desirable and possibly also indicating better upper bounds, is beyond the scope of this paper.

Acknowledgements

This work was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH and by RFBR and CNRS, project number 20-51-15009.

References

[ABD20] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. Fast mutation in crossover-based algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1268–1276. ACM, 2020.

[ABD21] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. Lazy parameter tuning and control: choosing all parameters randomly from a power-law distribution. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 1115–1123. ACM, 2021.

[AD20] Denis Antipov and Benjamin Doerr. Runtime analysis of a heavy-tailed $(1 + (\lambda, \lambda))$ genetic algorithm on jump functions. In *Parallel Problem Solving From Nature, PPSN 2020, Part II*, pages 545–559. Springer, 2020.

[AD21] Denis Antipov and Benjamin Doerr. A tight runtime analysis for the $(\mu + \lambda)$ EA. *Algorithmica*, 83:1054–1095, 2021.

[ADK19] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. A tight runtime analysis for the $(1 + (\lambda, \lambda))$ GA on LeadingOnes. In *Foundations of Genetic Algorithms, FOGA 2019*, pages 169–182. ACM, 2019.

[ADK20] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. The $(1 + (\lambda, \lambda))$ GA is even faster on multimodal problems. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1259–1267. ACM, 2020.

[Bäc93] Thomas Bäck. Optimal mutation rates in genetic search. In *International Conference on Genetic Algorithms, ICGA 1993*, pages 2–8. Morgan Kaufmann, 1993.

[BBD21a] Henry Bambury, Antoine Bultel, and Benjamin Doerr. Generalized jump functions. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 1124–1132. ACM, 2021.

[BBD21b] Riade Benbaki, Ziyad Benomar, and Benjamin Doerr. A rigorous runtime analysis of the 2-MMAS_{ib} on jump functions: ant colony optimizers can cope well with local optima. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 4–13. ACM, 2021.

[BD17] Maxim Buzdalov and Benjamin Doerr. Runtime analysis of the $(1 + (\lambda, \lambda))$ genetic algorithm on random satisfiable 3-CNF formulas. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 1343–1350. ACM, 2017.

[BDK16] Maxim Buzdalov, Benjamin Doerr, and Mikhail Kever. The unrestricted black-box complexity of jump functions. *Evolutionary Computation*, 24:719–744, 2016.

[BLS14] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. Unbiased black-box complexity of parallel search. In *Parallel Problem Solving from Nature, PPSN 2014*, pages 892–901. Springer, 2014.

[DD18] Benjamin Doerr and Carola Doerr. Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm. *Algorithmica*, 80:1658–1709, 2018.

[DD20] Benjamin Doerr and Carola Doerr. Theory of parameter control for discrete black-box optimization: provable performance gains through dynamic parameter choices. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 271–321. Springer, 2020. Also available at `https://arxiv.org/abs/1804.05650`.

[DDE13] Benjamin Doerr, Carola Doerr, and Franziska Ebel. Lessons from the black-box: Fast crossover-based genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2013*, pages 781–788. ACM, 2013.

[DDE15] Benjamin Doerr, Carola Doerr, and Franziska Ebel. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87–104, 2015.

[DFK+16] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima with diversity mechanisms and crossover. In *Genetic and Evolutionary Computation Conference, GECCO 2016*, pages 645–652. ACM, 2016.

[DFK+18] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation*, 22:484–497, 2018.

[DHK12] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425:17–33, 2012.

[DJW02] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[DJW06] Stefan Droste, Thomas Jansen, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39:525–544, 2006.

[DK20] Benjamin Doerr and Martin S. Krejca. The univariate marginal distribution algorithm copes well with deception and epistasis. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP 2020*, pages 51–66. Springer, 2020.

[DLMN17] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 777–784. ACM, 2017.

[DNS17] Benjamin Doerr, Frank Neumann, and Andrew M. Sutton. Time complexity analysis of evolutionary algorithms on random satisfiable k-CNF formulas. *Algorithmica*, 78:561–586, 2017.

[Doe18] Benjamin Doerr. An elementary analysis of the probability that a binomial random variable exceeds its expectation. *Statistics and Probability Letters*, 139:67–74, 2018.

[Doe20a] Benjamin Doerr. Does comma selection help to cope with local optima? In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1304–1313. ACM, 2020.

[Doe20b] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 1–87. Springer, 2020. Also available at `https://arxiv.org/abs/1801.06733`.

[Doe21] Benjamin Doerr. The runtime of the compact genetic algorithm on Jump functions. *Algorithmica*, 83:3059–3107, 2021.

[Dro02] Stefan Droste. Analysis of the (1+1) EA for a dynamically changing OneMax-variant. In *Congress on Evolutionary Computation, CEC 2002*, pages 55–60. IEEE, 2002.

[Dro04] Stefan Droste. Analysis of the (1+1) EA for a noisy OneMax. In *Genetic and Evolutionary Computation Conference, GECCO 2004*, pages 1088–1099. Springer, 2004.

[Dro05] Stefan Droste. Not all linear functions are equally difficult for the compact genetic algorithm. In *Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 679–686. ACM, 2005.

[ER63] Paul Erdős and Alfréd Rényi. On two problems of information theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézet Közleményei*, 8:229–243, 1963.

[FKK+16] Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Samadhi Nallaperuma, Frank Neumann, and Martin Schirneck. Fast building block assembly by majority vote crossover. In *Genetic and Evolutionary Computation Conference, GECCO 2016*, pages 661–668. ACM, 2016.

[FS20] Mario Alejandro Hevia Fajardo and Dirk Sudholt. On the choice of the parameter control mechanism in the $(1+(\lambda,\lambda))$ genetic algorithm. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 832–840. ACM, 2020.

[GP14] Brian W. Goldman and William F. Punch. Parameter-less population pyramid. In *Genetic and Evolutionary Computation Conference, GECCO 2014*, pages 785–792. ACM, 2014.

[HS18] Václav Hasenöhrl and Andrew M. Sutton. On the runtime dynamics of the compact genetic algorithm on jump functions. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 967–974. ACM, 2018.

[Jan15] Thomas Jansen. On the black-box complexity of example functions: the real jump function. In *Foundations of Genetic Algorithms, FOGA 2015*, pages 16–24. ACM, 2015.

[JJW05] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13:413–440, 2005.

[JW02] Thomas Jansen and Ingo Wegener. The analysis of evolutionary algorithms – a proof that crossover really can help. *Algorithmica*, 34:47–66, 2002.

[LW12] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. *Algorithmica*, 64:623–642, 2012.

[MB17] Vladimir Mironovich and Maxim Buzdalov. Evaluation of heavy-tailed mutation operator on maximum flow test generation problem. In *Genetic and Evolutionary Computation Conference, GECCO 2017, Companion Material*, pages 1423–1426. ACM, 2017.

[Müh92] Heinz Mühlenbein. How genetic algorithms really work: mutation and hillclimbing. In *Parallel Problem Solving from Nature, PPSN 1992*, pages 15–26. Elsevier, 1992.

[RA19] Jonathan E. Rowe and Aishwaryaprajna. The benefits and limitations of voting mechanisms in evolutionary optimisation. In *Foundations of Genetic Algorithms, FOGA 2019*, pages 34–42. ACM, 2019.

[RS14] Jonathan E. Rowe and Dirk Sudholt. The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm. *Theoretical Computer Science*, 545:20–38, 2014.

[RW20] Amirhossein Rajabi and Carsten Witt. Self-adjusting evolutionary algorithms for multimodal optimization. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1314–1322. ACM, 2020.

[RW21a] Amirhossein Rajabi and Carsten Witt. Stagnation detection in highly multimodal fitness landscapes. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 1178–1186. ACM, 2021.

[RW21b] Amirhossein Rajabi and Carsten Witt. Stagnation detection with randomized local search. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP 2021*, pages 152–168. Springer, 2021.

[SN14] Andrew M. Sutton and Frank Neumann. Runtime analysis of evolutionary algorithms on randomly constructed high-density

satisfiable 3-CNF formulas. In *Parallel Problem Solving from Nature, PPSN 2014*, pages 942–951. Springer, 2014.

[Sud05] Dirk Sudholt. Crossover is provably essential for the Ising model on trees. In *Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 1161–1167. ACM, 2005.

[SW04] Tobias Storch and Ingo Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320:123–134, 2004.

[Wit06] Carsten Witt. Runtime analysis of the (μ + 1) EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14:65–86, 2006.

[Wit13] Carsten Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing*, 22:294–318, 2013.

[Wit21] Carsten Witt. On crossing fitness valleys with majority-vote crossover and estimation-of-distribution algorithms. In *Foundations of Genetic Algorithms, FOGA 2021*, pages 2:1–2:15. ACM, 2021.

[WVHM18] Darrell Whitley, Swetha Varadarajan, Rachel Hirsch, and Anirban Mukhopadhyay. Exploration and exploitation without mutation: solving the jump function in $\Theta(n)$ time. In *Parallel Problem Solving from Nature, PPSN 2018, Part II*, pages 55–66. Springer, 2018.