Theoretical and Empirical Analysis of Parameter Control Mechanisms in the (1 + (λ, λ)) Genetic Algorithm

MARIO ALEJANDRO HEVIA FAJARDO, Department of Computer Science, University of Sheffield, Sheffield, United Kingdom
DIRK SUDHOLT, Department of Computer Science, University of Sheffield, Sheffield, United Kingdom and Chair of Algorithms for Intelligent Systems, University of Passau, Germany

The self-adjusting (1 + (λ, λ)) GA is the best known genetic algorithm for problems with a good fitness-distance correlation as in OneMax. It uses a parameter control mechanism for the parameter λ that governs the mutation strength and the number of offspring. However, on multimodal problems, the parameter control mechanism tends to increase λ uncontrollably.

We study this problem for the standard Jump_k benchmark problem class using runtime analysis. The self-adjusting (1 + (λ, λ)) GA behaves like a (1 + n) EA whenever the maximum value for λ is reached. This is ineffective for problems where large jumps are required. Capping λ at smaller values is beneficial for such problems. Finally, resetting λ to 1 allows the parameter to cycle through the parameter space. We show that resets are effective for all Jump_k problems: the self-adjusting (1 + (λ, λ)) GA performs as well as the (1 + 1) EA with the optimal mutation rate and evolutionary algorithms with heavy-tailed mutation, apart from a small polynomial overhead.

Along the way, we present new general methods for translating existing runtime bounds from the (1 + 1) EA to the self-adjusting (1 + (λ, λ)) GA. We also show that the algorithm presents a bimodal parameter landscape with respect to λ on Jump_k. For appropriate n and k, the landscape features a local optimum in a wide basin of attraction and a global optimum in a narrow basin of attraction. To our knowledge this is the first proof of a bimodal parameter landscape for the runtime of an evolutionary algorithm on a multimodal problem.

CCS Concepts: • **Theory of computation** → *Theory of randomized search heuristics;*

Additional Key Words and Phrases: Parameter control, parameter landscape, evolutionary algorithms, runtime analysis, theory

ACM Reference format:
Mario Alejandro Hevia Fajardo and Dirk Sudholt. 2022. Theoretical and Empirical Analysis of Parameter Control Mechanisms in the (1 + (λ, λ)) Genetic Algorithm. *ACM Trans. Evol. Learn. Optim* 2, 4, Article 13 (December 2022), 39 pages.
https://doi.org/10.1145/3564755

This research has been supported by CONACYT under the grant no. 739621 and registration no. 843375.
Authors' addresses: M. A. Hevia Fajardo, Chair of Algorithms for Intelligent Systems, University of Passau, Innstraße 43, 94032, Passau, Germany; D. Sudholt, Department of Computer Science, University of Sheffield, Regent Court, 211 Portobello, Sheffield, S1 4DP, United Kingdom.

2688-3007/2022/12-ART13 $15.00
https://doi.org/10.1145/3564755

1 INTRODUCTION

1.1 Motivation

Evolutionary Algorithms as well as other Randomised Search Heuristics are parametrised algorithms, that is, they have parameters such as the mutation rate, population size, and so on, that the user needs to choose. The behaviour of Evolutionary Algorithms largely depends on these parameter settings and it is well known that the efficiency of the optimisation process may depend drastically on its parameters and the problem in hand [Doerr and Doerr 2020; Lengler 2020; Lissovoi et al. 2020a, 2019]. Hence, parameter selection is an important field of study.

There are two main approaches to the parameter selection problem. The first approach, commonly known as *parameter tuning*, is to use static parameter values that have been deemed as *good parameter values* by either theoretical analyses or by an initial set of experiments that aim to manually or automatically identify good parameter values for the problem settings in hand. The second approach is known as *parameter control*; it uses non-static and *self-adjusting* parameter choices aiming to identify parameter values that are optimal for the current state of the optimisation process. The former approach has one disadvantage: during the optimisation process the optimal parameter values may change, hence any static choice may have sub-optimal performance [Doerr and Doerr 2020]. Parameter control mechanisms aim to identify such values *on the fly* without the need of parameter tuning.

In continuous optimisation, parameter control such as step-size adaptation is vital to ensure convergence to the optimum. In the discrete domain, parameter control is much less common and we are just beginning to understand the benefits that parameter control can provide. Providing a solid theoretical foundation for discrete self-adjusting algorithms was highlighted as a major challenge and a hot topic in evolutionary computation by leading researchers in the field [Doerr and Neumann 2021; Friedrich and Neumann 2017].

There has been an increasing interest in the theoretical study of parameter control mechanisms in the latest years in order to understand how they work and when they perform better than static parameter settings. One of the most successful implementations of parameter control mechanisms is the self-adjusting $(1 + (\lambda, \lambda))$ Genetic Algorithm (GA) [Doerr and Doerr 2018; Doerr et al. 2015], which is the fastest known unbiased genetic algorithm on the popular test function $\textsc{OneMax}(x) := \sum_{i=1}^{n} x_i$ and has shown excellent performance on NP-hard problems like MaxSat in both empirical [Goldman and Punch 2015] and theoretical studies [Buzdalov and Doerr 2017]. Despite its success on these problems it has been shown via theoretical [Antipov et al. 2019; Buzdalov and Doerr 2017] and empirical studies [Doerr et al. 2019; Foster et al. 2020; Hevia Fajardo 2019] that the self-adjusting $(1 + (\lambda, \lambda))$ GA does not behave well on instances with low fitness-distance correlation, that is, instances where the distance of search points to the optimum has a low correlation to their fitness values.

Several of the above works identified that the issue lies in the parameter control mechanism used to control the main parameter of the algorithm: the offspring population size λ. On functions with low fitness-distance correlation, the algorithm can get stuck in situations where λ diverges to its maximum value $\lambda = n$, and then performance deteriorates.

Goldman and Punch [2015] suggested to reset λ to 1 when $\lambda = n$ but also restart the search from a random individual. Buzdalov and Doerr [2017] proposed capping the value of λ depending on the fitness-distance correlation of the problem in hand. Lastly, Bassin et al. [2021] proposed a modification where the growth of λ is slowed down for long unsuccessful runs, while still letting the algorithm increase the parameter indefinitely. It achieves this by resetting λ to the parameters of its last successful generation after a certain number of unsuccessful generations and letting the algorithm increase λ a bit more every time it is reset.

At the moment, it is not clear which of these modifications is the best choice. Previous research is fragmented and most of the modifications proposed have only been studied empirically.

We make a step towards understanding the effects that different approaches for parameter control in the self-adjusting $(1 + (\lambda, \lambda))$ GA have, by providing a comprehensive theoretical analysis on the standard Jump_k and Trap benchmark functions (formally defined in Section 2.3). Our main contribution is a comprehensive analysis of three variations of the original parameter control mechanism on the self-adjusting $(1 + (\lambda, \lambda))$ GA showing theoretically and empirically that all the variations studied can improve the performance of the original mechanism on multimodal problems whilst not (or slightly) affecting its performance on simple problems such as OneMax.

1.2 Contributions

We consider the standard Jump_k benchmark problem class, a class of multimodal problems on which evolutionary algorithms typically have to make a jump to the optimum at a Hamming distance of k. The parameter k means that Jump_k has an adjustable difficulty and thus represents a whole range from easy to difficult multimodal and even deceptive problems.

We first present a general method for obtaining upper bounds on the expected optimisation time (the expected number of fitness evaluations to find a global optimum) of the original self-adjusting $(1 + (\lambda, \lambda))$ GA, based on the fitness-level method, in Section 3. With it we obtain novel bounds including bounds for unimodal functions and the Jump_k function class. Despite its simplicity, we show that it gives tight bounds, up to lower-order terms, on Jump_k functions. Tightness is established through lower bounds in Section 4. We show that the original self-adjusting $(1 + (\lambda, \lambda))$ GA does not benefit from crossover on Jump_k functions given that it has the same asymptotic runtime as the $(1 + 1)$ Evolutionary Algorithm (EA) with standard parameters $p = 1/n$.

Subsequently, in Section 5 we analyse the performance change when λ is capped to a value less than n. We suggest a generic choice for $\lambda_{\max} := n/2$ that can be advantageous on very hard and deceptive functions without affecting its performance on simple problems. We also show that capping λ can improve the performance of the algorithm on Jump_k, but its behaviour is highly dependent on the choice of $\lambda_{\max}$, defying the point of using a parameter control mechanism. Additionally, with the generic choice of $\lambda_{\max} := n/2$ the self-adjusting $(1 + (\lambda, \lambda))$ GA is faster than the original self-adjusting $(1 + (\lambda, \lambda))$ GA and the $(1 + 1)$ EA with standard parameters for jump sizes $k \leq \log n$ and $k > n/\log n$.

Our analysis in Section 6 also provides insights into the *parameter landscape*, which describes how parameter values relate to performance and which has recently emerged as a hot topic. The celebrated work by Pushak and Hoos [2018] provided evidence that parameter landscapes of prominent algorithms on well-known NP-hard problems like Satisfiability Problem (SAT), Mixed Integer Programming (MIP) and Travelling Salesman Problem (TSP), are surprisingly benign, and parameters often have a unimodal response. Hall et al. [2019, 2020a] rigorously proved that parameters of simple evolutionary algorithms on pseudo-Boolean test problems exhibit a unimodal landscape. These works led to algorithm configurators that exploit unimodal structures [Hall et al. 2020b; Pushak and Hoos 2020]. In sharp contrast to the above, we show that the parameter landscape concerning the choice of $\lambda_{\max}$ in the self-adjusting $(1 + (\lambda, \lambda))$ GA on Jump_k is bimodal. More precisely, we regard the time to reach the global optimum from a local optimum with λ at its maximum value as this setting dominates the optimisation time. We give a rigorous but complex formula for this expected time and use semi-rigorous arguments to identify optimal parameters. For appropriate choices of n and k, the landscape features a local optimum located in a wide basin of attraction and a global optimum hidden in a narrow basin of attraction. To the best of our knowledge this is the first proof of a bimodal parameter landscape for the runtime of an evolutionary algorithm on a multimodal benchmark problem. The closest related work and only other such proof we are aware of is for a *unimodal* problem: Lengler et al. [2021] showed that the parameter landscape for the compact Genetic Algorithm on OneMax has a bimodal structure.

The above results are proven for a small and natural modification of the self-adjusting $(1 + (\lambda, \lambda))$ GA in which the offspring created in the mutation step are included in the replacement selection. The advantage of this small change is that mutation creates search points at a possibly much larger distance, and then the $(1 + (\lambda, \lambda))$ GA searches within a wide neighbourhood (in the mutation step) and in a more local neighbourhood (after crossing the best mutant with its parent). Our analysis of the parameter landscape in Section 6 emphasizes the possible performance benefits. For jump sizes of k not too small, the best performance is obtained by choosing the parameter $\lambda_{\max}$ as to maximise the probability of finding the optimum during the mutation phase.

We also analyse the benefits of resetting λ in Section 7. With this strategy the algorithm is able to traverse the parameter space, instead of getting stuck with a certain parameter, benefiting the algorithm's behaviour when encountering local optima. In particular, with a clever selection of the update factor F we show that for Jump$_k$ the runtime of the algorithm is only by a factor of $O(n^2/k)$ slower than the runtime of the (1 + 1) EA with optimal parameter choice of $O((en/k)^k)$ [Doerr et al. 2017]. We remark that our choice of F does not depend on k, whereas the optimal mutation rate for the (1 + 1) EA requires k to be known in advance.

Finally, in Section 8 we provide an experimental analysis to test how our theoretical results translate to other fitness landscapes. The experimental results agree with our theoretical results on Jump$_k$ and show that the original self-adjusting $(1 + (\lambda, \lambda))$ GA (from Doerr and Doerr [2018]) has a poor performance on most problems tested compared against the other parameter control variations. This demonstrates that our results translate to other common problem settings.

Parts of our theoretical results appeared in a preliminary version [Hevia Fajardo and Sudholt 2020]. This improved and extended manuscript contains new theoretical analyses on the deceptive benchmark function Trap and significant new material has been added in Sections 6 (proof of a bimodal parameter landscape) and 8 (empirical analysis).

1.3 Related Work

We give a brief review of existing theoretical studies of parameter control mechanisms, followed by a review of theoretical studies on the $(1 + (\lambda, \lambda))$ GA and Jump$_k$.

Parameter control. There have been several examples where parameter control mechanisms were proposed, along with proven performance guarantees.

Böttcher et al. [2010] considered the test function LeadingOnes$(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_i$ that counts the number of consecutive ones at the start of the bit string. They showed that fitness-dependent mutation rates can improve performance of the (1 + 1) EA on LeadingOnes by a constant factor. Badkobeh et al. [2014] presented an adaptive strategy for the mutation rate in the $(1 + \lambda)$ EA that, for all values of λ, leads to provably optimal performance on OneMax. Lässig and Sudholt [2011] presented adaptive schemes for choosing the offspring population size in $(1 + \lambda)$ EAs and the number of islands in an island model. Doerr et al. [2019] showed that a success-based parameter control mechanism is able to identify and track the optimal mutation rate in the $(1 + \lambda)$ EA on OneMax, matching the performance of the best known fitness-dependent parameter [Badkobeh et al. 2014]. Doerr et al. [2020] presented a self-adaptive mechanism for the mutation rate in the $(1,\lambda)$ EA that matches the asymptotic expected runtime on OneMax as in Badkobeh et al. [2014]. Mambrini and Sudholt [2015] adapted the migration interval in island models and showed that adaptation can reduce the communication effort beyond the best possible fixed parameter. Doerr et al. [2021] proved that a success-based parameter control mechanism based on the 1/5th rule is able to achieve an asymptotically optimal runtime on LeadingOnes. Lissovoi et al. [2020b] proposed a

Generalised Random Gradient Hyper-Heuristic that uses a learning period of τ steps that can learn to adapt the neighbourhood size of Random Local Search optimally during the run on LeadingOnes. They proved that it has the best possible runtime achievable by any algorithm that uses the same low level heuristics. This result required the correct selection of the learning period τ; this was later solved using a self-adjusting mechanism adapting the learning period having an optimal asymptotic expected runtime on LeadingOnes [Doerr et al. 2018], OneMax and Ridge [Lissovoi et al. 2020a]. In a similar manner, Lissovoi et al. [2019] proposed a hyper-heuristic that chooses between elitist and non-elitist heuristics that achieves the best known expected runtime for general purpose randomised search heuristics on the problem class Cliff_k. Rajabi and Witt [2020a] used a self-adjusting asymmetric mutation that can provide a constant-factor speed-up on OneMax and obtained the same asymptotic performance on the generalised OneMax function. Rajabi and Witt [2020b] proposed a stagnation detection mechanism that raises the mutation rate when the algorithm is likely to have encountered a local optimum. The mechanism can be added to any existing EA; when added to the (1 + 1) EA, the SD−(1 + 1) EA has the same asymptotic runtime on Jump_k as the optimal parameter setting. Subsequently, Rajabi and Witt [2021a, 2021b] added the stagnation detection mechanism to the Randomised Local Search (RLS) obtaining a constant factor speed-up over the SD−(1 + 1) EA on Jump_k. Doerr and Doerr give a comprehensive survey of theoretical results on parameter control [Doerr and Doerr 2020].

The standard Jump_k benchmark problem class. The Jump_k function (formally defined in Section 2.3) was designed as a multimodal benchmark function with adjustable difficulty [Droste et al. 2002] allowing to test the ability of an algorithm to jump over a fitness valley of size k. Crossing this valley can be difficult for evolutionary algorithms. For the (1 + 1) EA using standard bit mutation with the default mutation rate of $p = 1/n$ it takes in expectation $\Theta(n^k)$ evaluations, for $k \leq n/2$. Because of this it has been commonly used in the theory of randomised search heuristics to evaluate their performance on multimodal problems.

For the (1 + 1) EA [Doerr et al. 2017] showed that for Jump_k with jump size k any mutation rate of $p \in [2/n, k/n]$ is exponentially faster (in k) than the standard choice of $p = 1/n$, with $p = k/n$ being the optimal choice. This showed that the traditional choice of mutation probability is not ideal and as an alternative the authors proposed the mutation operator fmut_β that chooses the mutation rate at random from a heavy-tailed distribution with exponent $\beta > 1$. This new algorithm achieved an asymptotic runtime that is only by a factor $k^{\beta-1}$ larger than the optimal mutation rate. A similar mutation operator has been proposed in Friedrich et al. [2018] that can outperform the fmut_β on Jump_k, but only for large jump sizes.

The Jump_k function has also been used as an example where crossover can be beneficial. The first work showing a benefit was Jansen and Wegener [2002] showing that a (μ + 1) *GA* applying uniform crossover only with an unnaturally small crossover probability of $p_c = 1/(kn)$ optimises Jump_k in expected time $O(\mu n^2 k^3 + 4^k/p_c)$. For more natural crossover probabilities, Dang et al. [2018] showed a runtime bound of $O(n^{k-1} \log n)$. This can be improved to $O(n \log n + 4^k)$ using additional diversity mechanisms [Dang et al. 2016] and to $O(n \log n)$ using an island model with majority vote [Friedrich et al. 2016]. Tailored hybrid genetic algorithms using a voting mechanism can optimise Jump_k in expected time $\Theta(n)$ [Whitley et al. 2018] and tailored black-box algorithms using a voting mechanism can even optimise (a slight variation of) Jump_k in expected time $\Theta(n/\log n)$ [Buzdalov et al. 2015].

The (1 + (λ, λ)) GA. The (1 + (λ, λ)) GA was first proposed in Doerr et al. [2015] and studied on OneMax. The (1 + (λ, λ)) GA using the recommended parameters from Doerr et al. [2015] (derived from studying OneMax) first creates $\lambda \leq n$ mutants by a process similar to a standard bit mutation with mutation rate λ/n (the only difference to standard GAs being that all mutants flip *the same* number of bits). Then it picks the best mutant and performs λ crossovers with the original parent. A biased uniform crossover is used that independently picks each bit from the mutant with probability $1/\lambda$. If the best search point created by crossover is at least as good as the parent, the former replaces the latter. Doerr et al. [2015] showed that for $\lambda \in [\omega(1), o(\log n)]$ the $(1 + (\lambda, \lambda))$ GA optimises OneMax in $o(n \log n)$ expected evaluations, breaking the $\Theta(n \log n)$ barrier that applies to all mutation-only algorithms [Lehre and Witt 2012]. Additionally, a fitness-dependent choice of $\lambda = \lceil \sqrt{n/(n - f(x))} \rceil$ leads to an expected optimisation time of $O(n)$ on OneMax. This highlights that the parameter λ is key to the performance of the $(1 + (\lambda, \lambda))$ GA as it governs the number of offspring, the mutation rate and the crossover bias.

To aid the complicated task of finding an optimal fitness-dependent choice, Doerr et al. [2015] proposed the self-adjusting $(1 + (\lambda, \lambda))$ GA that uses the 1/5-th success rule to adjust λ. Doerr and Doerr [2018] proved that the algorithm only needs $O(n)$ expected function evaluations on OneMax. The time of $O(n)$ is asymptotically optimal among all possible parameter choices for the $(1 + (\lambda, \lambda))$ GA. This was the first success-based parameter control mechanism proven to reduce the optimisation time of an algorithm by more than a constant factor, compared to optimal static parameter settings.

Goldman and Punch [2015] reported excellent performance for the self-adjusting $(1 + (\lambda, \lambda))$ GA on random instances of the maximum satisfiability problem, and a similar setting was studied by Buzdalov and Doerr [2017]. In the latter analysis it was shown that the algorithm is effective on instances with good fitness-distance correlation. However, on instances with low fitness-distance correlation the algorithm's performance decays. Antipov et al. [2019] analysed the $(1 + (\lambda, \lambda))$ GA on LeadingOnes to better understand the behaviour on functions with low fitness-distance correlation. They showed that the self-adjusting $(1 + (\lambda, \lambda))$ GA has the same asymptotic runtime as the (1 + 1) EA, but the hidden constants seem to be very large.

In an attempt to avoid the risk of λ diverging to its maximum that comes when using the parameter control mechanism of the self-adjusting $(1 + (\lambda, \lambda))$ GA, Antipov et al. [2020] modified the $(1 + (\lambda, \lambda))$ GA to sample λ from a heavy-tailed distribution each step of the optimisation instead of being self-adjusted. For the correct selection of the exponent $\beta \in (2, 3)$ and an upper bound for λ of $u > \ln^{\frac{1}{3-\beta}}(n)$ in a power law distribution $\text{pow}(\beta, u)$ it was shown that the algorithm can obtain the same asymptotic runtime of $O(n)$ on OneMax.

All studies mentioned so far used the standard parameter choice in the $(1 + (\lambda, \lambda))$ GA of $\lambda \leq n$, $p = \frac{\lambda}{n}$ and $c = \frac{1}{\lambda}$ but the mutation rate and crossover bias can take any value in $[0, 1]$ and the population size can be any integer number. Recently, Antipov et al. [2020] proved that a non-standard parameter setting of $\lambda = \frac{1}{n}\sqrt{\frac{n}{k}}^{k}$ and $p = c = \sqrt{\frac{k}{n}}$ can be useful on the multimodal function Jump$_k$, achieving a runtime of $O(n^{(k+1)/2} e^{O(k)} k^{-k/2})$. However, this result can only be achieved by knowing the jump size k beforehand. To overcome this, Antipov and Doerr [2020] and Antipov et al. [2021] proposed parameter settings that sample the parameters λ, p and c from power-law distributions. With a non-trivial selection of the distributions these instance-independent algorithms can obtain expected optimisation times that are only a small polynomial factor worse than the aforementioned bound.

Note that the $(1 + (\lambda, \lambda))$ GA with non-standard parameters from Antipov et al. [2021] and Antipov and Doerr [2020] does not use self-adjustment. Since we are interested in understanding self-adjusting mechanisms, in our theoretical work we restrict our attention to the standard parameterisation of the $(1 + (\lambda, \lambda))$ GA, that is, we fix the relationship $p = \frac{\lambda}{n}$ and $c = \frac{1}{\lambda}$ and aim to self-adjust λ.

We are confident that the strategies studied here can be translated to other self-adjusting EAs. Indeed, the resetting mechanism have been recently used by Hevia Fajardo and Sudholt [2021] for a $(1, \lambda)$ EA with self-adjusting offspring population size λ, where the resetting mechanism was essential for the algorithm to optimise the multimodal problem Cliff in $O(n \log n)$ evaluations, which is faster than any static λ by at least a polynomial factor. Additionally, most self-adjusting mechanisms proposed and analysed in the literature are mainly tailored towards optimising unimodal problems. Hevia Fajardo and Sudholt [2021] and Rajabi and Witt [2021a, 2021b], warn that these mechanisms tend to be problematic on multimodal problems because once a local optimum is reached the success of previous generations does not give a good indication of what parameters are needed to escape the local optimum. But, as we show in this work both the capping and resetting mechanisms can help self-adjusting mechanisms deal with local optima by avoiding divergent parameter values that can become harmful or let the mechanism cycle through the possible parameter values.

2 PRELIMINARIES

We use runtime analysis to analyse the performance of the self-adjusting $(1 + (\lambda, \lambda))$ GA on n-dimensional pseudo-Boolean functions $f \colon \{0,1\}^n \to \mathbb{R}$. We consider maximisation problems only, but it is straightforward to translate any maximisation problem into a minimisation problem by multiplying the function f by -1. We consider both the random number of fitness evaluations T^{eval} until a global optimum is found (also called optimisation time or runtime) as well as the random number of generations T^{gen}. The former reflects the total computational effort, whereas the latter is more relevant when the execution and the generation of offspring can be parallelised efficiently.

In the following, we write $H(x, x^*)$ to denote the Hamming distance between bit strings x and x^*. By $\mathcal{B}(n, p)$ we denote the binomial distribution with parameters $n \in \mathbb{N}$ and $p \in [0, 1]$. By log we denote the logarithm of base 2. We say an event happens with overwhelming probability (w.o.p.) if $\Pr(\text{event}) = 1 - 2^{-\Omega(n)}$.

2.1 Self-Adjusting $(1 + (\lambda, \lambda))$ GA

The self-adjusting $(1 + (\lambda, \lambda))$ GA is a crossover-based evolutionary algorithm that uses a mutation phase with a mutation rate higher than usual to assist exploration and a crossover phase as a repair mechanism. During the mutation phase the parent is mutated λ times, using a mutation operator called $\text{flip}_\ell(x)$. It chooses ℓ different bit positions in x uniformly at random and then it flips the values in those bits to create the mutated bit string. Each mutation phase the variable ℓ is sampled only once from a binomial distribution $\mathcal{B}(n, \frac{\lambda}{n})$, causing all mutation offspring to have the same Hamming distance to the parent. Afterwards, during the crossover phase the algorithm creates λ offspring by applying a biased uniform crossover called $\text{cross}_c(x, x')$ between the parent x and the best offspring from the mutation phase x'. The crossover operator works as follows: for each bit position, it selects the bit value from x' with probability $c = \frac{1}{\lambda}$ and from x otherwise. After the crossover phase, the algorithm performs an elitist selection using only the offspring from the crossover phase and the parent.

The algorithm adjusts λ every generation with a multiplicative update rule, where λ is multiplied by a factor $F^{1/4}$ if there is no improvement in fitness and divided by F otherwise. We consider $F > 1$ as a constant independent of n unless mentioned otherwise. The parameter λ has an upper limit $\lambda_{\max}$ which is commonly set to n. We note that in lines 5 and 8 of Algorithm 1, following [Doerr and Doerr 2018], we round λ to its closest integer ($\lfloor\lambda\rceil$); that is, $\lambda = \lfloor\lambda\rfloor$ if the decimal part of λ is less than $1/2$ and $\lambda = \lceil\lambda\rceil$ otherwise.

In this work we use a small variation of the algorithm, shown in Line 10 of Algorithm 1, where during the selection step the best offspring from the mutation phase is also considered. This

ALGORITHM 1: The self-adjusting (1 + (λ, λ)) GA with maximum offspring population size $\lambda_{\max} \leq n$.

```
1  Initialization: Sample $x \in \{0,1\}^n$ uniformly at random (u.a.r.);
2  Initialize $\lambda \leftarrow 1$, $p \leftarrow \lambda/n$, $c \leftarrow 1/\lambda$;
3  Optimization: for $t = 1, 2, \ldots$ do
       Mutation phase:
4      Sample $\ell$ from $\mathcal{B}(n, p)$;
5      for $i = 1, \ldots, \lfloor\lambda\rceil$ do
6          Sample $x^{(i)} \leftarrow \text{flip}_\ell(x)$ and query $f(x^{(i)})$;
7      Choose $x' \in \{x^{(1)}, \ldots, x^{(\lambda)}\}$ with $f(x') = \max\{f(x^{(1)}), \ldots, f(x^{(\lambda)})\}$ u.a.r.;
       Crossover phase:
8      for $i = 1, \ldots, \lfloor\lambda\rceil$ do
9          Sample $y^{(i)} \leftarrow \text{cross}_c(x, x')$ and query $f(y^{(i)})$;
10     If $\{x', y^{(1)}, \ldots, y^{(\lambda)}\} \setminus \{x\} \neq \emptyset$, choose $y \in \{x', y^{(1)}, \ldots, y^{(\lambda)}\} \setminus \{x\}$ with
       $f(y) = \max\{f(x'), f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ u.a.r.;
11     otherwise, set $y := x$;
       Selection and update step:
12     if $f(y) > f(x)$ then $x \leftarrow y$; $\lambda \leftarrow \max\{\lambda/F, 1\}$;
13     if $f(y) = f(x)$ then $x \leftarrow y$; $\lambda \leftarrow \min\{\lambda F^{1/4}, \lambda_{\max}\}$;
14     if $f(y) < f(x)$ then $\lambda \leftarrow \min\{\lambda F^{1/4}, \lambda_{\max}\}$;
```

modification has been suggested before in Goldman and Punch [2015] and Pinto and Doerr [2018] as a way to improve the performance of the (1 + (λ, λ)) GA. We believe (and tacitly take for granted) that this change does not invalidate previous theoretical runtime guarantees on problems such as OneMax [Doerr and Doerr 2018] and LeadingOnes [Antipov et al. 2019]. For OneMax, Doerr and Doerr [2018] confirm this fact without proof. Analyses on OneMax and LeadingOnes were based on drift analysis, and the drift can only increase if additional opportunities for improvements are considered. Comparing our empirical results on OneMax to the ones from Doerr et al. [2015] and empirical tests on LeadingOnes show that this is the case: the modification improves the runtime on OneMax and LeadingOnes by a small constant factor on all problem sizes tested.

Pinto and Doerr [2018] presented additional refinements of the (1 + (λ, λ)) GA that they call *implementation-aware*; these can save unnecessary evaluations and decrease some runtime results by constant factors. We only consider the aforementioned change with respect to the selection step for simplicity, and since we are interested in larger performance differences.

Considering the best offspring from the mutation phase is particularly helpful when the algorithm needs to make large jumps, as when encountering local optima. In fact, in Section 4 and 6 we show that for large jumps the crossover phase is not very helpful for reaching a higher fitness level, because the crossover phase tends to search near the current parent while the large mutations during the mutation phase can more easily jump out of local optima.

2.2 The (1 + 1) EA and (1 + n) EA

The (1 + λ) EA (Algorithm 2) starts by sampling a random solution $x \in \{0,1\}^n$. In each generation the algorithm creates λ offspring from the parent x using standard bit mutation with probability $p \in [0,1]$ and selects the offspring with the highest fitness (breaking ties u.a.r.), which replaces the parent if it has the same or better fitness. If $\lambda = 1$ the algorithm becomes the simplest EA, that is, the (1 + 1) EA. Another special case is the (1 + n) EA, that uses an offspring population size $\lambda = n$

ALGORITHM 2: $(1 + \lambda)$ EA.

```
1 Initialization: Choose $x \in \{0,1\}^n$ uniformly at random (u.a.r.);
2 Optimization: for $t \in \{1, 2, \dots\}$ do
3     Mutation: for $i \in \{1, \dots, \lambda\}$ do
4         Create $y'_i \in \{0,1\}^n$ by flipping each bit in $x$ independently with probability $p$.;
5     Selection: Choose $y \in \{y'_1, \dots, y'_\lambda\}$ with $f(y) = \max\{f(y'_1), \dots, f(y'_\lambda)\}$ u.a.r.;
6     if $f(y) \geq f(x)$ then $x \leftarrow y$;
```

2.3 Fitness Functions

The Jump_k benchmark problem class is a class of unitation functions, that is, functions that only depend on the number of 1-bits in a bit string, denoted as $|x|$:

$$\text{Jump}_k(x) := \begin{cases} n - |x| & \text{if } n - k < |x| < n, \\ k + |x| & \text{otherwise.} \end{cases}$$

We restrict ourselves to parameters $k \geq 2$ as otherwise the function essentially becomes OneMax. For $k \geq 2$ the Jump_k function increases its fitness value with the first $n - k$ 1-bits in the bit string, reaching a set of local optima that all have $n - k$ 1-bits. Larger numbers of 1-bits leads to a *valley* that contains the lowest fitness values in all the search space. The fitness value in the *valley* decreases when adding 1-bits, with the minimum fitness neighboring the global optimum. The optimum is the bit string 1^n.

The Trap function is described by:

$$\text{Trap}(x) := \begin{cases} |x| & \text{if } |x| \neq 0, \\ n + 1 & \text{otherwise.} \end{cases}$$

Similar to the Jump_k function, in the Trap function, the fitness value increases with the number of 1-bits, guiding hill climbing algorithms to the local optimum 1^n. However, the global optimum 0^n is at the maximum Hamming distance from the local optimum. For comparison-based algorithms the Trap function is equivalent to Jump_n.

2.4 Fitness-Level Method

The fitness-level method [Wegener 2002] is a general analysis method for upper bounds on the expected optimisation time. We describe it in the context of the $(1 + 1)$ EA. The method uses so-called f-based partitions, which is a partition of the search space $\{0,1\}^n$ into sets $A_1, \dots, A_{m+1}$ for some $m \in \mathbb{N}$ where all search points A_i are strictly worse than all search points in A_{i+1}, and A_{m+1} exclusively contains all global optima. Each of these sets needs to be left at most once. Suppose that we know that for every search point $x \in A_i$, the probability of creating a search point in a higher fitness-level set is at least s_i, for some expression $s_i > 0$. Then the expected time for leaving set A_i is at most $1/s_i$. Since every fitness level has to be left at most once before reaching A_{m+1} and once A_{m+1} is reached a global optima is found due to its definition, the expected optimisation time of the $(1 + 1)$ EA is at most $\sum_{i=1}^{m} 1/s_i$.

The fitness-level method is a simple and versatile method in its own right, and it allows researchers to translate bounds on the runtime of the simple $(1 + 1)$ EA to other elitist algorithms. This has been achieved for parallel evolutionary algorithms [Lässig and Sudholt 2014b], ant colony optimisation [Gutjahr 2008; Neumann et al. 2009], and particle swarm optimisation [Sudholt and Witt 2010]. It further gives rise to tail bounds [Witt 2014] and lower bounds [Doerr and Kötzing

2021; Sudholt 2013], and the principles extend to non-elitist algorithms as well [Corus et al. 2018; Doerr and Kötzing 2021].

Fitness levels may contain search points of different fitness. In the special case where each set A_i contains search points with only one fitness value the partition is called a *canonical* partition.

2.5 Useful Inequalities

In this section, for the readers' convenience, we compile well-known inequalities that we will use.

Lemma 2.1.

(a) *For all* $x \geq -1$ *and* $r \geq 1$, $(1+x)^r \geq 1 + rx$.
(b) *For all* $x \in [0,1]$ *and* $r \geq 1$, $(1-x)^r \leq \frac{1}{1+rx}$.

Part (a) is well known as Bernoulli's inequality. Part (b) was shown in Rowe and Sudholt [2014]. Also, we use the following inequalities, taken from Doerr [2020, Corollary 1.4.6].

Lemma 2.2. *For all* $r \geq 1$ *and* $0 \leq x \leq r$, $(1 - \frac{x}{r})^r \leq e^{-x} \leq (1 - \frac{x}{r})^{r-x}$.

3 FITNESS-LEVEL UPPER BOUNDS FOR THE SELF-ADJUSTING (1 + (λ, λ)) GA

The self-adjusting $(1 + (\lambda, \lambda))$ GA has only been analysed theoretically on easy unimodal functions like OneMax [Doerr and Doerr 2018] and LeadingOnes [Antipov et al. 2019] as well as random satisfiability instances [Buzdalov and Doerr 2017].[1] Despite these analyses we do not have a clear understanding of its behaviour on other problem settings, especially when it encounters local optima.

In this section we give a new general method to find upper bounds for the runtime of the self-adjusting $(1 + (\lambda, \lambda))$ GA using previously known runtime bounds from the (1 + 1) EA. It is based on the observation that, when λ hits its maximum value of $\lambda = n$, the algorithm temporarily performs n standard bit mutations and thus simulates a generation of a $(1 + n)$ EA.

The following theorem gives a fitness-level upper bound tailored to the self-adjusting $(1 + (\lambda, \lambda))$ GA.

Theorem 3.1. *Given an arbitrary* f*-based partition* $A_1, \ldots, A_{m+1}$. *Let* d *be the number of non-optimal fitness values,* $F > 1$ *constant, and* s_i *be a lower bound for the probability that the (1 + 1) EA with mutation probability* $1/n$ *creates a search point in a higher fitness-level set from any search point in fitness level* A_i. *Then for the self-adjusting (1 + (λ, λ)) GA we have*

$$E(T^{\text{eval}}) \leq O(dn) + 2\sum_{i=1}^{m} \frac{1}{s_i};$$

$$E(T^{\text{gen}}) \leq \lceil 4 \log_F(n) \rceil + 6d + \frac{1}{n}\sum_{i=1}^{m} \frac{1}{s_i}.$$

To prove Theorem 3.1, we analyse the time the algorithm spends in generations with $\lambda < n$ and those in generations with $\lambda = n$.

In the following, we refer to a generation that improves the current best fitness as *successful* and otherwise as *unsuccessful*. We show that a logarithmic number of unsuccessful generations is sufficient to reach the maximum λ value.

Lemma 3.2. *Let* x *and* y *be the parent and offspring, respectively, as in Algorithm 1. For all values of* $F := F(n) > 1$, *initial* λ *value* $\lambda_{\text{init}} \geq 1$ *and target value* $\lambda_{\text{new}} \leq \lambda_{\max}$, *the following holds. If in every*

[1]The (1 + (λ, λ)) GA with static parameter choices has also been studied on multimodal functions but only using non-standard parameter choices that do not translate to the self-adjusting mechanism.

generation $f(y) \leq f(x)$, the self-adjusting (1 + (λ, λ)) GA needs at most $\lceil 4\log_F(\frac{\lambda_{\text{new}}}{\lambda_{\text{init}}})\rceil$ generations to grow λ from λ_{init} to at least λ_{new}. During these generations the algorithm makes at most $\frac{4F^{1/4}}{F^{1/4}-1} \cdot \lambda_{\text{new}}$ evaluations.

For constant $F > 1$ the number of generations is $O(\log \lambda_{\text{new}})$ and the number of evaluations is $O(\lambda_{\text{new}})$.

Proof. Starting with an offspring population size of λ_{init}, after i unsuccessful generations the offspring population size is $\lambda_{\text{init}} \cdot F^{i/4}$. For $i := \lceil 4\log_F(\frac{\lambda_{\text{new}}}{\lambda_{\text{init}}})\rceil$, we get an offspring population size of

$$\lambda_{\text{init}} \cdot F^{\lceil 4\log_F(\frac{\lambda_{\text{new}}}{\lambda_{\text{init}}})\rceil/4} \geq \lambda_{\text{init}} \cdot F^{\log_F(\frac{\lambda_{\text{new}}}{\lambda_{\text{init}}})} = \lambda_{\text{new}}.$$

The number of unsuccessful generations needed is thus at most

$$\left\lceil 4\log_F\left(\frac{\lambda_{\text{new}}}{\lambda_{\text{init}}}\right)\right\rceil.$$

Using $\lceil \lambda_{\text{init}} \cdot F^{i/4}\rceil \leq 2\lambda_{\text{init}} \cdot F^{i/4}$, during these generations, the number of evaluations is at most

$$\begin{aligned}\sum_{i=0}^{\lceil 4\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})\rceil-1} 2\left\lceil \lambda_{\text{init}} \cdot F^{i/4}\right\rceil &\leq 4\lambda_{\text{init}} \sum_{i=0}^{\lceil 4\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})\rceil-1} \left(F^{1/4}\right)^i \\ &= 4\lambda_{\text{init}} \cdot \frac{(F^{1/4})^{\lceil 4\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})\rceil} - 1}{F^{1/4}-1} \\ &\leq 4\lambda_{\text{init}} \cdot \frac{(F^{1/4})^{4\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})+1}}{F^{1/4}-1} \\ &= 4\lambda_{\text{init}} \cdot \frac{F^{1/4} \cdot \lambda_{\text{new}}/\lambda_{\text{init}}}{F^{1/4}-1} = \frac{4F^{1/4}}{F^{1/4}-1} \cdot \lambda_{\text{new}}.\end{aligned}$$

For constant $F > 1$, the number of generations simplifies to $\lceil 4\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})\rceil \leq \lceil 4\log_F \lambda_{\text{new}}\rceil = \lceil 4\log(\lambda_{\text{new}})/\log(F)\rceil = O(\log \lambda_{\text{new}})$ and the number of evaluations simplifies to $\frac{4F^{1/4}}{F^{1/4}-1} \cdot \lambda_{\text{new}} = O(\lambda_{\text{new}})$. □

Now we bound the number of generations in which the self-adjusting (1 + (λ, λ)) GA operates with $\lambda < n$. To do so, we take into account that the algorithm does not need to increase from λ_{init} every time since we only decrease λ by a factor F each time we find an improvement.

Lemma 3.3. *Let $F := F(n) > 1$, $\lambda_{\max} \leq n$, and d be the number of non-optimal fitness values of an arbitrary fitness function. The maximum number of generations in which the self-adjusting (1 + (λ, λ)) GA uses $\lambda < \lambda_{\max}$ is at most $\lceil 4\log_F(\lambda_{\max})\rceil + 5d$. These generations lead to at most $2d\lambda_{\max} + \frac{4F^{1/4}\lambda_{\max}}{F^{1/4}-1}$ evaluations. For constant $F > 1$ the number of evaluations is $O(d\lambda_{\max})$.*

Proof. In every successful generation, λ is decreased to $\max\{1, \lambda/F\}$ and otherwise it is increased to $\min\{\lambda_{\max}, \lambda \cdot F^{i/4}\}$.

We use the *accounting method* [Cormen et al. 2009, Chapter 17] to account for all generations with $\lambda < \lambda_{\max}$. The basic idea is to create a fictional bank account to which operations are being charged. Some operations are allowed to pay excess amounts, while others can take money from such accounts to pay for their costs. Provided that no fictional account gets overdrawn the total amount of money paid bounds the total cost of all operations.

We start with a fictional bank account and pay costs of $\lceil 4\log_F(\lambda_{\max})\rceil$, since that is the maximum number of consecutive unsuccessful generations before reaching $\lambda = \lambda_{\max}$ (Lemma 3.2).

In a successful generation, we pay costs of 1 to cover the cost of the generation, and deposit an additional amount of 4 to the fictional bank account, which will be used to pay for 4 unsuccessful generations needed to increase λ to its original value. Unsuccessful generations that increase λ may withdraw 1 from the fictional account and pay for the cost of this generation. Unsuccessful generations where $\lambda = \lambda_{\max}$ are not charged since they are not counted.

We now need to prove that the fictional bank account is never overdrawn. For any point in time, the number of generations where λ increases is bounded by $T^{\text{inc}} \leq \lceil 4\log_F(\lambda_{\max})\rceil + 4T^{\text{dec}}$ where T^{inc} and T^{dec} are the number of generations increasing and decreasing λ, respectively. This holds by Lemma 3.2 and the fact that one successful generation that decreases λ compensates for 4 unsuccessful generations that may increase λ. Considering the initial payment of $\lceil 4\log_F(\lambda_{\max})\rceil$ and transactions for each generation, the current balance is

$$\lceil 4\log_F(\lambda_{\max})\rceil - T^{\text{inc}} + 4T^{\text{dec}} \geq 0,$$

that is, the account is never overdrawn. The number of generations with $\lambda < \lambda_{\max}$ is thus bounded by the sum of all payments. There can only be d successful generations, hence the sum of payments is at most $\lceil 4\log_F(\lambda_{\max})\rceil + 5d$.

It remains to bound the number of evaluations. Since we initialise with $\lambda = 1$, there must be $\lceil 4\log_F(\lambda_{\max})\rceil$ generations $t_1, t_2, \ldots$ such that during generation t_i, we have $\lambda \leq \lceil F^{i/4}\rceil$. From Lemma 3.2, we know that during these $\lceil 4\log_F(\lambda_{\max})\rceil$ generations, the algorithm uses at most $\frac{4F^{1/4}}{F^{1/4}-1} \cdot \lambda_{\max}$ evaluations. Additionally, in the other at most $5d$ possible generations, the maximum number of evaluations per generation is bounded by $2\lambda_{\max}$. Therefore the algorithm uses at most $2d\lambda_{\max} + \frac{4F^{1/4}\lambda_{\max}}{F^{1/4}-1}$ evaluations with an offspring population size $\lambda < \lambda_{\max}$. For constant $F > 1$, the number of evaluations simplifies to $2d\lambda_{\max} + \frac{4F^{1/4}\lambda_{\max}}{F^{1/4}-1} = O(d\lambda_{\max})$. □

With Lemma 3.3 now we have the tools necessary to analyse the runtime of Algorithm 1.

Proof of Theorem 3.1. Owing to Lemma 3.3, we can focus on bounding the time spent in generations with $\lambda = n$. In these generations, the mutation rate is $p = \lambda/n = 1$ and thus all bits are flipped during mutation. When the current search point is x, mutation thus produces its binary complement, $\overline{x}$. The crossover phase uses a crossover bias of $c = 1/\lambda$, which means that each bit is independently taken from the mutant $\overline{x}$ with probability $1/\lambda$ and otherwise it is taken from x. This is equivalent to a standard bit mutation with mutation probability of $1/\lambda$. Given that $\lambda = n$, during the crossover phase the algorithm creates n independent offspring using standard bit mutation. The crossover phase is then equivalent to the output of a $(1 + n)$ EA.

For each fitness level we calculate the number of generations the self-adjusting $(1 + (\lambda, \lambda))$ GA spends on this level while $\lambda = n$ and the self-adjusting $(1 + (\lambda, \lambda))$ GA essentially simulates a $(1 + n)$ EA. (We pessimistically ignore the fact that such a situation may not be reached at all; especially on easy problems, λ may not hit the maximum value before the optimum is found.)

We argue as in Lässig and Sudholt [Lässig and Sudholt 2014a, Theorem 1] to derive a fitness-level bound for the $(1 + n)$ EA. The probability that there is one of n offspring that finds a better fitness level is at least

$$1 - (1 - s_i)^n \geq 1 - \left(\frac{1}{1 + s_i n}\right) = \frac{s_i n}{1 + s_i n},$$

where the inequality holds by Lemma 2.1 (b). The expected number of generations to leave A_i using $\lambda = n$ is at most $\frac{1+s_i n}{s_i n}$. Adding the generations with $\lambda = n$ over all fitness levels and the generations spent with $\lambda < n$, we get

$$\mathrm{E}(T^{\text{gen}}) \leq \lceil 4\log_F(n)\rceil + 5d + \sum_{i=1}^{m}\left(1 + \frac{1}{s_i n}\right) \leq \lceil 4\log_F(n)\rceil + 6d + \frac{1}{n}\sum_{i=1}^{m}\frac{1}{s_i},$$

where the last step used $m \leq d$, that is, the number of fitness levels is bounded by the number of fitness values.

By Lemma 3.3, the number of evaluations used with $\lambda < n$ is $O(dn)$ when $F > 1$ is a constant. Since with $\lambda = n$ each generation leads to $2n$ evaluations, multiplying the above bound yields the claimed bound on the number of evaluations. □

We show how to apply Theorem 3.1 to obtain novel bounds on the expected optimisation time of the self-adjusting $(1 + (\lambda, \lambda))$ GA, including the Jump$_k$ function class.

Theorem 3.4. *The expected optimisation time* $\mathrm{E}(T^{\text{eval}})$ *of the self-adjusting (1 + (λ, λ)) GA with* $F > 1$ *constant is at most*

(a) $\mathrm{E}(T^{\text{eval}}) = O(n^n/|\mathrm{OPT}|)$ *and* $\mathrm{E}(T^{\text{gen}}) = O(n^{n-1}/|\mathrm{OPT}|)$ *on any function with a set* OPT *of global optima,*
(b) $\mathrm{E}(T^{\text{eval}}) = O(dn)$ *and* $\mathrm{E}(T^{\text{gen}}) = O(d + \log n)$ *on unimodal functions (functions for which every non-optimal search point has a strictly better Hamming neighbour) with* $d + 1$ *fitness values,*
(c) $\mathrm{E}(T^{\text{eval}}) \leq (1 + o(1)) \cdot 2n^k(1 - 1/n)^{-n+k}$ *and* $\mathrm{E}(T^{\text{gen}}) \leq (1 + o(1)) \cdot 2n^{k-1}(1 - 1/n)^{-n+k}$ *on* Jump$_k$ *with* $k \geq 3$.

Proof. For the general upper bound, we use a fitness level partition with A_1 containing all non-optimal fitness values and A_2 containing the set OPT, then the number of fitness levels is $m + 1 = 2$. We use the corresponding success probability $s_i \geq |\mathrm{OPT}|/n^n$. With this we bound $\sum_{i=1}^{m} \frac{1}{s_i} = O(n^n/|\mathrm{OPT}|)$. All functions have $d < 2^n$ non-optimal fitness values and $n^n/|\mathrm{OPT}| \geq (n/2)^n$, therefore the term $O(dn)$ can be absorbed.

For unimodal functions, we use a canonical f-based partition and success probabilities of $s_i \geq 1/n \cdot (1 - 1/n)^{n-1} \geq 1/(en)$ where the last inequality holds by Lemma 2.2. This yields $\mathrm{E}(T^{\text{eval}}) \leq O(dn) + 2\sum_{i=1}^{d} en = O(dn)$. For the expected number of generations, we get $\mathrm{E}(T^{\text{gen}}) \leq \lceil 4\log_F(n)\rceil + 6d + \frac{1}{n}\sum_{i=1}^{d} en = O(d + \log n)$.

For Jump$_k$ functions with $k \geq 3$ any individual that is not a local or global optimum can find an improvement by increasing or decreasing the number of 1-bits. This yields success probabilities of at least $(n - i)/(en)$ for all search points with $0 \leq i < n - k$ ones and of at least $i/(en)$ for all search points with $n - k < i < n$ ones. For search points with $n - k$ ones, a standard bit mutation can jump to the optimum by flipping the correct k 0-bits and not flipping any other bit. This has a probability of $s_{n-k} = (1/n)^k(1 - 1/n)^{n-k}$. Hence,

$$\mathrm{E}(T^{\text{eval}}) \leq O(n^2) + 2\sum_{i=0}^{n-k-1} \frac{en}{n-i} + 2\sum_{i=n-k+1}^{n-1} \frac{en}{i} + 2n^k\left(1 - \frac{1}{n}\right)^{-n+k} = O(n^2) + 2n^k\left(1 - \frac{1}{n}\right)^{-n+k}.$$

As $k \geq 3$ the second term is $\Omega(n^3)$ and the first term constitutes a smaller-order term, yielding

$$\mathrm{E}(T^{\text{eval}}) = (1 + o(1)) \cdot 2n^k\left(1 - \frac{1}{n}\right)^{-n+k}.$$ □

In Theorem 3.1 as in most of the theoretical bounds proven in other studies for the $(1 + (\lambda, \lambda))$ GA, we focus only on the offspring from the crossover phase. But as mentioned before we argue that considering also the offspring from the mutation phase can improve the performance of the $(1 + (\lambda, \lambda))$ GA. To exemplify this we consider the Trap function which is the most difficult problem for the standard (1 + 1) EA having an expected optimisation time of $\Theta(n^n)$ [Droste et al. 2002]. In contrast, when considering the offspring from the mutation phase, the self-adjusting $(1 + (\lambda, \lambda))$ GA is able to optimise the Trap function using only $O(n)$ evaluations.

Theorem 3.5. *Let $F > 1$ be a constant. The expected optimisation time (in terms of fitness evaluations) of the self-adjusting $(1 + (\lambda, \lambda))$ GA on the* Trap *function is $O(n)$.*

Proof. We divide the proof in two parts, the OneMax phase, and the *trap* phase. While in the OneMax phase, if the optimum is not found the algorithm behaves as on OneMax. From Theorem 9 in Doerr and Doerr [2018], we know that the OneMax phase takes $O(n)$ evaluations.

Once it reaches the local optimum (*trap* phase), within at most $\lceil 4 \log_F(n) \rceil$ unsuccessful generations λ increases to $\lambda = n$. This then implies $p = 1$ and the algorithm creates the global optimum during the mutation phase with probability 1. Therefore, the *trap* phase is solved in $O(\log n)$ generations with probability 1. During these generations, the algorithm uses $O(n)$ evaluations, as shown in Lemma 3.2. Adding the optimisation times of both phases proves the claim. □

We want to point out that in this example the algorithm benefits from the global optimum being exactly at Hamming distance n from the local optimum. Its purpose is to illustrate the possible benefits of considering the offspring from the mutation phase during selection. This inspired us to consider the strategies in the following sections, where we show more extensively the improvements that considering the offspring from the mutation phase during selection can give to the algorithm.

4 CROSSOVER DOES NOT BENEFIT THE SELF-ADJUSTING $(1 + (\lambda, \lambda))$ GA ON JUMP$_k$

We now show that the bound for the self-adjusting $(1 + (\lambda, \lambda))$ GA with standard parameter settings on Jump$_k$ from Theorem 3.4 (c) is asymptotically tight. This implies that unless k is very small the self-adjusting $(1 + (\lambda, \lambda))$ GA is no more efficient on Jump$_k$ than the $(1 + 1)$ EA and less efficient than other GAs using crossover [Dang et al. 2018, 2016; Jansen and Wegener 2002].

Theorem 4.1. *Let $F > 1$ be a constant. The expected optimisation time (in terms of fitness evaluations) of the self-adjusting $(1 + (\lambda, \lambda))$ GA on the* Jump$_k$ *function with $4 \le k \le (1 - \varepsilon)n/2$, for any constant $\varepsilon > 0$, is at least*

$$(1 - o(1)) \cdot 2n^k \left(1 - \frac{1}{n}\right)^{-n+k}.$$

We first show upper and lower bounds on the probability that the $(1 + (\lambda, \lambda))$ GA finds any particular target search point x^* during one mutation phase. Even though we only need the upper bounds in this section, the lower bounds will be useful later on.

Lemma 4.2. *For every current search point x, every target search point x^* and every current parameter λ, let $p_{\text{mut}}^{\lambda}(x, x^*)$ be the probability that the $(1 + (\lambda, \lambda))$ GA creates x^* during the mutation phase of one generation.*

If $x^* = \overline{x}$ and $\lambda = n$, $p_{\text{mut}}^{\lambda}(x, x^*) = 1$.

If $x^* \neq \overline{x}$ and $\lambda = n$, $p_{\text{mut}}^{\lambda}(x, x^*) = 0$.

If $x^* \in \{x, \overline{x}\}$ and $\lambda < n$,

$$p_{\text{mut}}^{\lambda}(x, x^*) = (\lambda/n)^{H(x,x^*)}(1 - \lambda/n)^{n-H(x,x^*)}.$$

Otherwise,

$$\frac{\lfloor \lambda \rfloor}{2}(\lambda/n)^{H(x,x^*)}(1 - \lambda/n)^{n-H(x,x^*)} \le p_{\text{mut}}^{\lambda}(x, x^*) \le \lceil \lambda \rceil (\lambda/n)^{H(x,x^*)}(1 - \lambda/n)^{n-H(x,x^*)}.$$

If $H(x, x^*) \in \{2, \dots, n-2\}$, the factor $1/2$ in the above expression can be replaced by $1 - O(1/n)$.

From the transition probabilities, the term $(\lambda/n)^{H(x,x^*)}(1 - \lambda/n)^{n-H(x,x^*)}$ equals the probability of a standard bit mutation with mutation probability λ/n creating x^* from x. If $x^* \notin \{x, \overline{x}\}$, the offspring population of λ amplifies this probability by a factor within $[\lfloor \lambda \rfloor/2, \lceil \lambda \rceil]$. If $x^* \in \{x, \overline{x}\}$, the $(1 + (\lambda, \lambda))$ GA does not benefit from its offspring population at all.

Proof of Lemma 4.2. The algorithm needs to sample $\ell = H(x, x^*)$, in order to find x^* during the mutation phase. The probability of this event is

$$\Pr(\ell = H(x, x^*)) = \binom{n}{H(x, x^*)} (\lambda/n)^{H(x,x^*)} (1 - \lambda/n)^{n-H(x,x^*)}. \tag{1}$$

In the special case where $\lambda = n$ then the mutation probability $p = 1$ and $\ell = n$, hence if $x = \overline{x}$ the target search point is created with probability 1 and if $x^* \neq \overline{x}$ the target search point is created with probability 0. If $\lambda < n$ and $x^* \in \{x, \overline{x}\}$, $\binom{n}{H(x,x^*)} = 1$ and the claim for this case follows from (1) as all λ mutants will create x^* for the right choice of ℓ.

Otherwise, the (1 + (λ, λ)) GA also needs to flip the correct bits during the mutation phase. Since there are $\binom{n}{H(x,x^*)}$ possible ways to flip the bits the probability that one offspring flips the correct bits is $\binom{n}{H(x,x^*)}^{-1}$. This gives us the following probability of finding x^* during $\lfloor\lambda\rceil$ mutations (the algorithm creates $\lfloor\lambda\rceil$ offspring), conditional on $\ell = H(x, x^*)$:

$$\Pr(x^* \mid \ell = H(x, x^*)) = 1 - \left(1 - 1/\binom{n}{H(x, x^*)}\right)^{\lfloor\lambda\rceil}.$$

Using the estimate from Lemma 2.1 (a) and $\lfloor\lambda\rceil \le \lceil\lambda\rceil$ this is bounded from above by

$$\lceil\lambda\rceil/\binom{n}{H(x, x^*)}$$

and bounded from below using Lemma 2.1 (b) and $\lfloor\lambda\rceil \ge \lfloor\lambda\rfloor$ by

$$\frac{\lfloor\lambda\rfloor/\binom{n}{H(x,x^*)}}{1 + \lfloor\lambda\rfloor/\binom{n}{H(x,x^*)}} \ge \frac{\lfloor\lambda\rfloor/\binom{n}{H(x,x^*)}}{2}, \tag{2}$$

where the last inequality follows from $x^* \notin \{x, \overline{x}\}$ yielding $\binom{n}{H(x,x^*)} \ge n$ and thus $1 + \lfloor\lambda\rfloor/n \le 2$. Since

$$p^{\lambda}_{\text{mut}}(x, x^*) = \Pr(x^* \mid \ell = H(x, x^*)) \Pr(\ell = H(x, x^*)),$$

multiplying (1) with the above bounds on $\Pr(x^* \mid \ell = H(x, x^*))$ and observing that the binomial coefficients cancel completes the proof.

If $H(x, x^*) \in \{2, \ldots, n-2\}$ we bound the denominator in (2) using $\binom{n}{H(x,x^*)} = \Omega(n^2)$ and thus $1 + \lfloor\lambda\rfloor/\binom{n}{H(x,x^*)} \le 1 + O(1/n)$. This yields a factor of $1/(1 + O(1/n)) = 1 - O(1/n)$. □

The following lemma gives an upper bound on the probability of hitting any specific search point x^* during one crossover phase of the (1 + (λ, λ)) GA. For the original (1 + (λ, λ)) GA that does not consider mutants for selection, Lemma 4.3 bounds the probability of hitting x^* in one generation.

Lemma 4.3. *For every current search point x, every target search point x^* and every $\lambda \in [1, n]$, the probability that the (1 + (λ, λ)) GA creates x^* during the crossover phase of one generation is at most*

$$\lfloor\lambda\rceil^2 \left(\frac{1}{n}\right)^{H(x,x^*)} \left(1 - \frac{1}{n}\right)^{n-H(x,x^*)}.$$

Note that the probability bound may be larger than 1 for very small $H(x, x^*)$ and large λ. Before diving into the proof, we give the main idea here. Recall that in every generation, the (1 + (λ, λ)) GA performs $\lfloor\lambda\rceil$ mutations with a radius of ℓ (drawn from a binomial distribution with parameters λ/n and n) and $\lfloor\lambda\rceil$ crossover operations with the best mutant. All mutants are chosen uniformly at random from the Hamming ball of radius ℓ around x. However, the following selection of the best mutant does not preserve uniformity as some offspring on said Hamming ball may have a higher fitness than others. Hence, the crossover operations will affect particular regions of the search space more than others. While this is a helpful algorithmic concept (in a sense that this makes the (1 + (λ, λ)) GA solve OneMax in expected time $O(n)$ [Doerr et al. 2015]), it makes it hard to analyse what search points will be generated during crossover as it depends on the fitness function in hand.

As a solution, we borrow an idea similar to *non-selective family trees* by Witt [2006] and Lehre and Yao [2009] that was also used in previous analyses of the (1 + (λ, λ)) GA [Doerr and Doerr 2018, Proof of Proposition 1]. We consider a variant of the (1 + (λ, λ)) GA that we call *non-selective (1 + (λ, λ)) GA*: instead of performing $\lfloor \lambda \rceil$ crossovers with the best mutant, it performs $\lfloor \lambda \rceil$ crossovers for *all of the* $\lfloor \lambda \rceil$ *mutants*. This results in $\lfloor \lambda \rceil^2$ offspring generated from crossover, in addition to $\lfloor \lambda \rceil$ mutants. Since the offspring created by the original (1 + (λ, λ)) GA form a subset of the offspring generated by the non-selective (1 + (λ, λ)) GA, the probability of the original (1 + (λ, λ)) GA creating x^* in one crossover phase is bounded by the probability of the non-selective (1 + (λ, λ)) GA creating x^* during crossover. Owing to the absence of selection, the output of the $\lfloor \lambda \rceil^2$ crossover operations is independent of the fitness and we obtain a probability bound that only depends on the Hamming distance $H(x, x^*)$.

Proof of Lemma 4.3. We argue similarly as the proof of Proposition 1 in Doerr and Doerr [2018]. Fix an offspring y created by the non-selective (1 + (λ, λ)) GA. The process for creating y can be described as follows. The algorithm first picks a random value of ℓ according to a binomial distribution with parameters n and λ/n, and then flips ℓ bits chosen at random to create a mutant x'. The creation of x' can alternatively be regarded as a standard bit mutation with a mutation rate of λ/n. To create y, each bit is independently taken from x' with probability $1/\lambda$. Hence, each bit y_i in y attains the value $1 - x_i$ with probability $1/n$, and it attains value x_i with probability $1 - 1/n$, independently from all other bits. Hence, the creation of y can be described as a standard bit mutation with mutation rate $1/n$.

The probability of $y = x^*$ is thus $(1/n)^{H(x,x^*)}(1 - 1/n)^{n-H(x,x^*)}$. Note that different offspring are not independent as they use the same random value of ℓ, and every batch of $\lfloor \lambda \rceil$ crossover operations is derived from the same mutant. A union bound over all $\lfloor \lambda \rceil^2$ offspring allows us to conclude, despite these dependencies, that the probability of one offspring generating x^* is at most

$$\lfloor \lambda \rceil^2 \left(\frac{1}{n}\right)^{H(x,x^*)} \left(1 - \frac{1}{n}\right)^{n-H(x,x^*)}. \qquad \square$$

Now we are in a position to prove Theorem 4.1.

Proof of Theorem 4.1. By standard Chernoff bounds, the probability that the initial search point has at most $(1 + \varepsilon)n/2$ ones is $1 - 2^{-\Omega(n)}$. We assume this to happen and note that then the algorithm will never accept a search point in the fitness valley of $n - k < i < n$ ones.

Let T^{local} be the random number of generations until any search point in the local optimum with $n-k$ ones or the global optimum is reached for the first time. We bound $\mathrm{E}(T^{\text{local}})$ from above as follows[2]. This can be done using Theorem 3.1 as in Theorem 3.4 (c), but with a non-canonical f-based partition where the best fitness level includes the local optimum and the global optimum. This yields $\mathrm{E}(T^{\text{local}}) = O(n)$ generations.

[2]The (1 + (λ, λ)) GA optimises OneMax in expected time $O(n)$, but it is not immediately obvious how to translate the analysis to the OneMax-like parts of Jump$_k$. Note that the (1 + (λ, λ)) GA may overshoot the local optimum and then the analysis on OneMax breaks down. We suspect this can be fixed with small modifications, but for now we show a more obvious bound as this is sufficient for our purposes.

By Lemmas 4.3 and 4.2, before we reach the local optimum a generation with parameter λ reaches the optimum with probability at most

$$\lceil\lambda\rceil(\lambda/n)^k(1-\lambda/n)^{n-k} + \lfloor\lambda\rceil^2 n^{-k} := p$$

since the current search point has Hamming distance at least k from the optimum.

We claim that $p = O(1/n^2)$ by showing that both summands are in $O(1/n^2)$. Recalling $k \geq 4$, we have $\lfloor\lambda\rceil^2 n^{-k} \leq n^{2-k} = O(1/n^2)$. To bound the first summand $\lceil\lambda\rceil(\lambda/n)^k(1-\lambda/n)^{n-k}$, we use $\lceil\lambda\rceil \leq n$ and the fact that the maximum of $x \mapsto x^k(1-x)^{n-k}$ in the interval $[0,1]$ is attained for $x := k/n$. Thus, the first summand is at most $n(k/n)^k(1-k/n)^{n-k}$. The function $k \mapsto (k/n)^k$ is non-increasing in the interval $[4, n/e]$, thus it is bounded by $O(n^{-4})$ for all $k \in [4, n/e]$. For $k > n/e$ we recall $k \leq n/2$ and observe that $(1-k/n)^{n-k} \leq (1-1/e)^{n/2}$ and then the first summand is exponentially small. In all cases, $p = O(1/n^2)$.

By a union bound the probability that the global optimum is found in t steps is at most tp. Then the probability that the global optimum is found during the first T^{local} steps is at most

$$\sum_{t=0}^{\infty} \Pr(T^{\text{local}} = t) \cdot tp = p \cdot \mathrm{E}(T^{\text{local}}) = O(1/n).$$

Now assume that the local optimum has been reached and $\lambda < \lambda_{\max} = n$. Since the optimum is the only search point with a strictly larger fitness, λ will be increased in every generation unless the optimum is found. By Lemma 3.2, there are at most $\lceil 4\log_F(n)\rceil$ generations before λ has increased to $\lambda_{\max}$. By the same arguments as above, the probability that the optimum is found during this time is $O((\log n)/n^2)$.

Once $\lambda = \lambda_{\max} = n$ has been reached, mutation always creates search points with k ones (i.e. mutation will never find the optimum) and crossover boils down to a standard bit mutation with mutation rate $1/n$. Then the probability of one crossover creating the optimum is $(1/n)^k(1-1/n)^{n-k}$ and the expected number of crossover operations for hitting the optimum is thus

$$n^k \cdot \left(1 - \frac{1}{n}\right)^{-n+k}.$$

Since every batch of λ crossover operations is preceded by λ (useless) fitness evaluations during mutation, this adds a factor of 2 to the above lower bound. The proof is completed by noting that, after adding up all failure probabilities, this case is reached with probability at least $1 - O(1/n)$ and $\mathrm{E}(T) \geq \mathrm{E}(T \mid A)\Pr(A)$, where A is the event that we do not reach the global optimum before reaching the local one. □

5 CAPPING λ

A simple solution to prevent λ from growing to large values is to constrain it. Buzdalov and Doerr [2017] first suggested this strategy, showing that it benefits the algorithm when optimising instances of the maximum satisfiability problem with weak fitness-distance correlation. Similarly, in Bassin et al. [2021] the authors showed empirically that capping λ at $\log n$ can improve the performance on linear functions with random weights. Antipov et al. [2019] capped λ to $n/2$, arguing that since $p = \lambda/n$ larger values of λ would use mutation probabilities larger than 1/2 which are considered ill-natured because mutation would create offspring that on average are further away from the parent than the average search point. In this section we explore its benefits.

5.1 Self-Adjusting (1 + (λ, λ)) GA Capping λ

In previous sections we have shown that the self-adjusting (1 + (λ, λ)) GA can increase λ to its maximum in only a logarithmic number of steps. This indicates that the algorithm may increase the parameter in a difficult part of the optimisation and afterwards it could end up with a too large population size, affecting performance. In Algorithm 1 the hyper-parameter $\lambda_{\max}$ prevents such a problem by capping the value of λ to any value $\lambda_{\max} \in \{1, \dots, n\}$, allowing the user to prevent the algorithm from increasing λ towards possible sub-optimal parameters.

5.2 Generic Choice for $\lambda_{\max}$

Since the first time the self-adjusting (1 + (λ, λ)) GA was proposed, the algorithm had a limit on the value of λ of n. This was imposed in order to prevent the mutation probability from exceeding 1. As a secondary effect it gives the algorithm a safety net, allowing it to fall back to the behaviour of the (1 + n) EA in case λ reaches its maximum value, as we explored on Section 3.

We argue that, if the problem is not known, a good generic strategy is to set $\lambda_{\max} = n/2$. This means that, in a situation where no improvements are found quickly and λ increases, the self-adjusting (1 + (λ, λ)) GA is able to simulate random search during the mutation phase whenever $\lambda_{\max}$ is reached. This is a potential advantage when the algorithm is stuck in a very hard local optimum, or on deceptive functions where random search is a viable technique (e.g. Jump_k with large k such as $k > n/\log n$). Note that the self-adjusting (1 + (λ, λ)) GA still retains its exploitation capability even with $\lambda = n/2$ because the offspring from the crossover phase would still have on average only 1 bit different from the parent. Hence, the algorithm performs wide exploration and exploitation at the same time, in different phases of the same generation. In addition, the algorithm is still able to optimise OneMax efficiently; the cap on λ only kicks in when regular exploitation fails.

To justify our choice of $\lambda_{\max} = n/2$ we present a general bound for the self-adjusting (1 + (λ, λ)) GA with this parameter choice in Theorem 5.1. This theorem shows that $\lambda_{\max} = n/2$ only has a worst-case runtime of $O(dn) + \frac{2^{n+4}}{|\text{OPT}|}$ that applies for most functions as opposed to the worst-case expected runtime of $n^{\Theta(n)}$ for the (1 + 1) EA. Note that this bound does not include Trap functions which we consider in Section 5.3.

Theorem 5.1. *Let f be any function with d non-optimal fitness values and a set* OPT *of global optima such that either* $|\,\text{OPT}\,| \geq 2$ *or* $\text{OPT} = \{x^*\}$ *and its complement* $\overline{x^*}$ *does not have the second-best fitness value. Then for the self-adjusting (1 + (λ, λ)) GA with* $\lambda_{\max} := n/2$ *and* $F > 1$ *constant we have*

$$E(T^{\text{eval}}) \leq O(dn) + \frac{2^{n+4}}{|\text{OPT}|};$$
$$E(T^{\text{gen}}) \leq O(d + \log n) + \frac{2^{n+3}}{n|\text{OPT}|}.$$

Proof. By Lemma 3.3, the algorithm spends $O(dn)$ evaluations and $O(d + \log n)$ generations in settings with $\lambda < \lambda_{\max}$. Hence, we can focus on improvement probabilities when $\lambda = \lambda_{\max}$.

If $|\,\text{OPT}\,| \geq 2$, by Lemma 4.2, the probability of one generation hitting any search point in OPT that is not the binary complement of the current search point is at least $\lambda_{\max} \cdot 2^{-n-1} = n \cdot 2^{-n-2}$. Since there are at least $|\,\text{OPT}\,| - 1 \geq |\,\text{OPT}\,|/2$ such search points, the probability for finding the optimum is at least $n \cdot 2^{-n-3} \cdot |\,\text{OPT}\,|$. Taking the reciprocal gives an upper bound on $E(T^{\text{gen}})$, and multiplying by $2\lambda_{\max} = n$ yields a bound on $E(T^{\text{eval}})$.

If $\text{OPT} = \{x^*\}$, we use the same argument to show that within $\frac{2^{n+3}}{n|\text{OPT}|}$ generations we either hit an optimum or a second-best search point. From the latter, the probability of hitting the optimum is bounded in the same way, since by assumption the current search point is different from $\overline{x^*}$. Then we proceed as before. □

5.3 Trap Functions

The conditions from Theorem 5.1 require a fitness function to either contain at least two global optima, or that the complement of the unique global optimum does not have the second-best fitness. Most fitness functions meet this condition. Notable exceptions are Trap functions (defined in Section 2.3) as there the local optimum 1^n with the second-best fitness is precisely the complement of the unique global optimum 0^n. We show that Trap functions were excluded from Theorem 5.1 for a good reason since for Trap functions the runtime is higher than the bound from Theorem 5.1 by a factor of order $\Omega(n)$.

Theorem 5.2. *Let $F > 1$ be a constant and $\lambda_{\max} = n/2$. The expected optimisation time (in terms of fitness evaluations) of the self-adjusting $(1 + (\lambda, \lambda))$ GA on the Trap function is $\Omega(n2^n)$.*

Proof. We divide the proof in three parts, the *initialisation* phase, the OneMax phase, and the *trap* phase.

The expected number of zero bits in the initial individual is $n/2$. By applying Chernoff bounds we can see that, w.o.p. the initial individual has at least $n/3$ one bits. Therefore, with at least the same probability the algorithm does not find the optimum during the *initialisation* phase.

From Theorem 9 in Doerr and Doerr [2018], we know that the OneMax phase takes in expectation $E(T^{\text{OneMax}}) = O(n)$ generations.[3] We now prove that during these generations the algorithm does not find the optimum w.o.p.

By Lemma 4.2 the probability to find the optimum during the mutation phase at a Hamming distance of at least $n/3$ is:

$$\Pr(x' = x^*) \le \lceil\lambda\rceil \left(\frac{\lambda}{n}\right)^{n/3} \left(1 - \frac{\lambda}{n}\right)^{2n/3} \le n \cdot 3^{-\frac{n}{3}} \left(\frac{3}{2}\right)^{-\frac{2n}{3}}. \tag{3}$$

The second inequality comes from bounding $\lceil\lambda\rceil \le n$ and using the parameter $\lambda = n/3$ everywhere else, which maximises the term $(\frac{\lambda}{n})^{n/3}(1-\frac{\lambda}{n})^{2n/3}$. For the crossover phase, to find the global optimum we use Lemma 4.3, obtaining,

$$\Pr(y = x^*) \le \lfloor\lambda\rfloor^2 \left(\frac{1}{n}\right)^{n/3} \left(1 - \frac{1}{n}\right)^{2n/3} \le n^{-\frac{n}{3}+2} \cdot e^{-2/3}. \tag{4}$$

Adding Equations (3) and (4) we get a probability of finding the optimum in one iteration of $p = O(n \cdot 3^{-n} \cdot 2^{2n/3})$. Then the probability to find the optimum during T^{OneMax} iterations is at most

$$\sum_{t=0}^{\infty} \Pr\left(T^{\text{OneMax}} = t\right) \cdot tp = p \cdot E\left(T^{\text{OneMax}}\right) = O(n^2 \cdot 3^{-n} \cdot 2^{2n/3}).$$

Finally, during the *trap* phase since $x = \overline{x^*}$ from Lemma 4.2 we get

$$\Pr(y = x^*) = \left(\frac{\lambda}{n}\right)^n \le 2^{-n}.$$

By Lemma 3.2, the algorithm spends at most $\lceil 4\log_F(n/2)\rceil$ generations to get to the parameter $\lambda = n/2$. The probability of finding the optimum during these iterations is $1-(1-2^{-n})^{\lceil 4\log_F(n/2)\rceil} \le 2^{2-n}\log_F n$. Once the maximum parameter value has been reached, the probability of sampling $\ell = n$ and thus find the optimum is $\Pr(y = x^*) = 2^{-n}$. Hence, in expectation 2^n further iterations are needed to find the optimum and each one of these iterations uses n evaluations. Since this situation is reached w.o.p., the expected number of evaluations is $\Omega(n \cdot 2^n)$. □

[3] As long as $\lambda_{\max}$ is bigger than $C_0\sqrt{n/(n - \text{OneMax}(x))}$ for a constant C_0 large enough to satisfy Lemma 16 in Doerr and Doerr [2018] the bound of $O(n)$ from Theorem 9 in Doerr and Doerr [2018] holds.

5.4 Jump$_k$ Functions

In this section we explore in detail how capping λ affects the performance of the algorithm on Jump$_k$ functions. In the following theorem, we only consider the mutation phase and assume for simplicity to start in the local optimum.

Theorem 5.3. *After reaching the local optimum, the expected number of function evaluations for the self-adjusting (1 + (λ, λ)) GA with $F > 1$ constant and λ capped at $\lambda_{\max} < n$ is at most*

$$O(\lambda_{\max}) + 4\left(\frac{n}{\lambda_{\max}}\right)^k \left(1 - \frac{\lambda_{\max}}{n}\right)^{-n+k}.$$

Proof. By Lemma 3.2, λ reaches $\lambda_{\max}$ or the algorithm finds the global optimum within $O(\lambda_{\max})$ evaluations. Then the probability of jumping to the optimum in one generation is at least $\lambda_{\max}/2 \cdot (\lambda_{\max}/n)^k(1-\lambda_{\max}/n)^{n-k}$ by Lemma 4.2. Taking the reciprocal and multiplying by $2\lambda_{\max}$ yields the claim. □

Note that for $\lambda_{\max} := k$, Theorem 5.3 yields an upper bound of

$$O(k) + 4\left(\frac{n}{k}\right)^k \left(1 - \frac{k}{n}\right)^{-n+k}.$$

For $k \geq 3$, these bounds match the expected time for the (1 + 1) EA with the optimal mutation rate of k/n up to constant factors [Doerr et al. 2017]. However, we would need to know k in advance, which defies the goal of parameter control. As mentioned before, an alternative strategy is to set $\lambda_{\max} := n/2$.

Theorem 5.4. *Let $k < \frac{n}{4}$. After reaching the local optimum, the expected number of function evaluations for the self-adjusting (1 + (λ, λ)) GA with $F > 1$ constant and $\lambda_{\max} := n/2$ on Jump$_k$ is*

$$\mathrm{E}(T^{\mathrm{eval}}) \leq \begin{cases} O\left(\left(\frac{n}{2}\right)^k\right) & \text{if } k \leq \frac{\log(n/2)}{2}, \\ O(\min\{(2n)^{k-1}, 2^n\}) & \text{otherwise.} \end{cases}$$

Note that these upper bounds prove that the self-adjusting (1 + (λ, λ)) GA with $\lambda_{\max} := n/2$ is faster than the (1 + 1) EA with the default mutation rate $1/n$ for all $k \leq \log n$ and $k > n/\log n$ as the latter needs expected time $\Theta(n^k)$ [Droste et al. 2002].

Before proving Theorem 5.4 we need to understand how the crossover phase might help the optimisation process. For the crossover phase to be able to find the optimum, the selected offspring from the mutation phase must flip all 0-bits from the parent. Afterwards the crossover phase is able to repair such offspring to find the global optimum. This was studied by Antipov et al. [2020] for any static parameter choice.

Theorem 5.5 (Theorem 3.3 in Antipov et al. [2020]). *Let $k \leq \frac{n}{4}$. Assume that $p \geq \frac{2k}{n}$ and $q_\ell \geq 0.1$ is the probability that the number of bits flipped $\ell \in [pn, 2pn]$. After reaching the local optimum, the expected runtime of the (1 + (λ, λ)) GA with static parameters on Jump$_k$ is*

$$\mathrm{E}(T^{\mathrm{eval}}) \leq \frac{8\lambda}{q_\ell \min\{1, \lambda\left(\frac{p}{2}\right)^k\} \min\{1, \lambda c^k(1-c)^{2pn-k}\}};$$

$$\mathrm{E}(T^{\mathrm{gen}}) \leq \frac{4}{q_\ell \min\{1, \lambda\left(\frac{p}{2}\right)^k\} \min\{1, \lambda c^k(1-c)^{2pn-k}\}}.$$

An extension of this theorem to the self-adjusting (1 + (λ, λ)) GA can be easily shown as follows.

Lemma 5.6. *Let $k \leq \frac{n}{4}$. After reaching the local optimum the expected runtime of the self-adjusting (1 + (λ, λ)) GA with $p = \lambda/n$, $c = 1/\lambda$, $\lambda_{\max} \geq 2k$ and $F > 1$ constant on* $\textsc{Jump}_k$ *is*

$$\mathrm{E}(T^{\text{eval}}) \leq O(n) + \frac{8}{q_\ell \min\left\{1, \lambda_{\max}\left(\frac{\lambda_{\max}}{2n}\right)^k\right\}\left(\frac{1}{\lambda_{\max}}\right)^k\left(1-\frac{1}{\lambda_{\max}}\right)^{2\lambda_{\max}-k}}.$$

Proof. By Lemma 3.2, λ reaches $\lambda_{\max}$ or the algorithm finds the global optimum within $O(\log n)$ generations and $O(n)$ evaluations. Then the algorithm will always use $\lambda = \lambda_{\max}$, therefore the probability of jumping to the optimum during the crossover phase is given by Theorem 5.5. □

We now have the necessary tools to prove Theorem 5.4.

Proof of Theorem 5.4. For the proof we use Lemma 5.6 as follows:

$$\mathrm{E}(T^{\text{eval}}) \leq O(n) + \frac{8}{q_\ell \min\{1, \frac{n}{2}\left(\frac{1}{4}\right)^k\}(\frac{2}{n})^k(1-\frac{2}{n})^{n-k}}.$$

Note that $k \leq \frac{\log(n/2)}{2}$ is equivalent to $\frac{n}{2}(\frac{1}{4})^k \geq 1$ and this implies $\min\{1, \frac{n}{2}(\frac{1}{4})^k\} = 1$. Hence,

$$\mathrm{E}\left(T^{\text{eval}}\right) \leq O(n) + \frac{8}{q_\ell(\frac{2}{n})^k(1-\frac{2}{n})^{n-k}} = O\left(\left(\frac{n}{2}\right)^k\right).$$

If $\frac{\log(n/2)}{2} < k$ then $\min\{1, \frac{n}{2}(\frac{1}{4})^k\} = \frac{n}{2}(\frac{1}{4})^k$ and

$$\mathrm{E}(T^{\text{eval}}) \leq O(n) + \frac{8}{q_\ell \frac{n}{2}\left(\frac{1}{4}\right)^k(\frac{2}{n})^k(1-\frac{2}{n})^{n-k}} \leq O(n) + \frac{32}{q_\ell(\frac{1}{2n})^{k-1}(1-\frac{2}{n})^{n-k}} = O\left((2n)^{k-1}\right).$$

In addition, we also consider the upper bound from Theorem 5.1, obtaining $\mathrm{E}(T^{\text{eval}}) = O(2^n)$. □

6 PARAMETER LANDSCAPE ON $\textsc{Jump}_k$

In Section 5 it was shown that choosing $\lambda_{\max} = k$ gives an expected runtime only a constant factor worse than the (1 + 1) EA with optimal mutation probability of k/n. However, this runtime guarantee only takes into account the mutation phase of the algorithm. Here we analyse the parameter landscape of the self-adjusting (1 + (λ, λ)) GA with $p = \lambda/n$, $c = 1/\lambda$ and $\lambda_{\max} \in [1, n]$. To analyse the effects of the parameter $\lambda_{\max}$ we use the precise expression of the probability $s_k^{\lambda_{\max}}$ of the (1 + (λ, λ)) GA with $\lambda = \lambda_{\max}$ to jump from the local optimum to the global optimum in one iteration. Hereby we ignore the initial time to climb up to the local optimum and the time for λ to increase to $\lambda_{\max}$. Our previous analyses have shown that these times are negligible anyway. Antipov and Doerr [2020] computed the probability of jumping from a local optimum to the global optimum, but only considered the offspring from the crossover phase for selection. The probability of jumping from the local optimum to the global optimum is

$$s_k^{\lambda_{\max}} = \sum_{j=0}^{n} \Pr\left(\ell = j \mid \lambda = \lambda_{\max}\right) \cdot \Pr\left(y = x^* \mid \ell = j \wedge \lambda = \lambda_{\max}\right).$$

Since $\ell \sim \mathcal{B}(n, p)$ the probability of sampling $\ell = j$ is given by

$$\Pr\left(\ell = j \mid \lambda = \lambda_{\max}\right) = \binom{n}{j}\left(\frac{\lambda_{\max}}{n}\right)^j\left(1-\frac{\lambda_{\max}}{n}\right)^{n-j}.$$

The probability of finding the optimum depends on the number of bits flipped, ℓ. For the algorithm to find the optimum, ℓ needs to be at least the size of the jump k in order to be able to flip all the

0-bits. Thus, for all $\ell < k$, $\Pr(y = x^* \mid \ell = j \wedge \lambda = \lambda_{\max}) = 0$. When $\ell = k$, if the mutation phase flips all the 0-bits, the optimum is found and the crossover phase is not needed. This has a probability of

$$\Pr\left(y = x^* \mid \ell = k \wedge \lambda = \lambda_{\max}\right) = 1 - \left(1 - \frac{1}{\binom{n}{k}}\right)^{\lambda_{\max}}.$$

Following Antipov and Doerr [2020], for $\ell > k$ the global optimum can only be found during the crossover phase because the mutation phase flips more than k bits. In order for the crossover to be able to find the optimum, first the algorithm needs to select a *good offspring* during the mutation phase, that is, an offspring from the set of all possible offspring that flip all the k 0-bits from the parent that we call $X^{(*)}$. Hence, we can calculate the probability of finding the optimum during the crossover phase by taking the probability of selecting an offspring during the mutation phase in $X^{(*)}$ and multiplying it by the conditional probability of finding the optimum given that $x' \in X^{(*)}$. That is, $\Pr(y = x^* \mid \ell = j \wedge \lambda = \lambda_{\max})$ equals

$$\Pr\left(x' \in X^{(*)} \mid \ell = j \wedge \lambda = \lambda_{\max}\right) \cdot \Pr\left(y = x^* \mid x' \in X^{(*)} \wedge \ell = j \wedge \lambda = \lambda_{\max}\right).$$

The probability of the mutation operator flipping all the 0-bits from the parent is $\binom{n-k}{\ell-k}/\binom{n}{\ell}$. If $\ell \in [k+1, 2k-1]$ any offspring from the mutation phase that flips all 0-bits has more than $n-k$ ones and hence falls into the valley of search points with a fitness less than k. This is worse than any offspring that does not flip all 0-bits. Therefore, in order to select an offspring with all 0-bits flipped the algorithm needs to create all offspring with all k 0-bits flipped, that is,

$$\Pr\left(x' \in X^{(*)} \mid \ell = j \wedge \lambda = \lambda_{\max}\right) = \left(\frac{\binom{n-k}{j-k}}{\binom{n}{j}}\right)^{\lambda_{\max}}.$$

In contrast, for $\ell \geq 2k$ the algorithm always selects an offspring with all k 0-bits flipped if one is created. This is because the algorithm flips at least k 1-bits in all mutants, hence the number of 1-bits in each mutant is at most $n-k$, making an offspring with the k 0-bits flipped have the greatest fitness value. Therefore, the probability of this event is

$$\Pr\left(x' \in X^{(*)} \mid \ell = j \wedge \lambda = \lambda_{\max}\right) = 1 - \left(1 - \frac{\binom{n-k}{j-k}}{\binom{n}{j}}\right)^{\lambda_{\max}}.$$

After selecting an offspring $x' \in X^{(*)}$ during the mutation phase, to generate the global optimum in the crossover phase the algorithm needs to take the k bits which are zero in x from x' and take all the $\ell - k$ bits which are zero in x' from x. This has a probability of $c^k(1-c)^{\ell-k}$ for one offspring, and it needs to happen for at least one of the λ offspring. This yields

$$\Pr\left(y = x^* \mid x' \in X^{(*)} \wedge \ell = j \wedge \lambda = \lambda_{\max}\right) = 1 - \left(1 - \left(\frac{1}{\lambda_{\max}}\right)^k \left(1 - \frac{1}{\lambda_{\max}}\right)^{j-k}\right)^{\lambda_{\max}}.$$

Putting all together and using $\lambda' := \lambda_{\max}$ to improve readability we have shown the following.

Theorem 6.1. *Consider the self-adjusting (1 + (λ, λ)) GA capping λ at $\lambda' = \lambda_{\max}$ on* Jump$_k$. *When the algorithm has reached the local optimum and $\lambda = \lambda'$, the probability of creating the optimum in one generation is*

$$
\begin{aligned}
s_k^{\lambda'} = {} & \binom{n}{k}\left(\frac{\lambda'}{n}\right)^k\left(1-\frac{\lambda'}{n}\right)^{n-k}\left(1-\left(1-\frac{1}{\binom{n}{k}}\right)^{\lambda'}\right)\\
& + \sum_{j=k+1}^{2k-1}\binom{n}{j}\left(\frac{\lambda'}{n}\right)^j\left(1-\frac{\lambda'}{n}\right)^{n-j}\left(\frac{\binom{n-k}{j-k}}{\binom{n}{j}}\right)^{\lambda'}\cdot\left(1-\left(1-\left(\frac{1}{\lambda'}\right)^k\left(1-\frac{1}{\lambda'}\right)^{j-k}\right)^{\lambda'}\right)\\
& + \sum_{j=2k}^{n}\binom{n}{j}\left(\frac{\lambda'}{n}\right)^j\left(1-\frac{\lambda'}{n}\right)^{n-j}\left(1-\left(1-\frac{\binom{n-k}{j-k}}{\binom{n}{j}}\right)^{\lambda'}\right)\cdot\left(1-\left(1-\left(\frac{1}{\lambda'}\right)^k\left(1-\frac{1}{\lambda'}\right)^{j-k}\right)^{\lambda'}\right)
\end{aligned}
$$

and the expected number of fitness evaluations to find the global optimum is $2\lambda_{\max}/s_k^{\lambda_{\max}}$.

Note that the first line is precisely the probability $p_{\text{mut}}^{\lambda'}(x, x^*)$ that the optimum is found during the mutation phase. We have already bounded this in Lemma 4.2 from above and below. By the final statement in Lemma 4.2, for $k \geq 2$ the probability equals $\lambda'(\lambda'/n)^k(1-\lambda'/n)^{n-k}$ bar a negative small-order term. The term $(\lambda'/n)^k(1-\lambda'/n)^{n-k}$ (without the leading factor λ') is maximised for $\lambda' := k$. Hence, the first line attains its maximum around $\lambda' = k$.

In the first summation, the term $(\frac{\binom{n-k}{j-k}}{\binom{n}{j}})^{\lambda'}$ simplifies to $(\frac{j(j-1)\cdot\ldots\cdot(j-k+1)}{n(n-1)\cdot\ldots\cdot(n-k+1)})^{\lambda'}$, which is bounded from above by $(2k/(n-k))^{k\lambda'}$. For $k = o(n)$, these summands are all negligibly small, compared to the other terms from Theorem 6.1.

The last summation was bounded from below by Antipov et al. [2020] for all $\lambda \geq 2k$ as:

$$
\frac{1}{40}\min\left\{1, \lambda\left(\frac{\lambda}{2n}\right)^k\right\}\lambda\left(\frac{1}{\lambda}\right)^k\left(1-\frac{1}{\lambda}\right)^{2\lambda-k}.
$$

Since $(1-\frac{1}{\lambda})^{2\lambda-k} = \Theta(1)$ for $\lambda \geq 2k$ and it converges to e^{-2} as λ grows, we ignore this factor, and the constant $1/40$, for the time being, and focus on the term

$$
\min\left\{1, \lambda\left(\frac{\lambda}{2n}\right)^k\right\}\lambda\left(\frac{1}{\lambda}\right)^k = \min\left\{\lambda\left(\frac{1}{\lambda}\right)^k, \lambda^2\left(\frac{1}{2n}\right)^k\right\}
$$

instead. For $\lambda(\frac{\lambda}{2n})^k \leq 1$ the minimum simplifies to $\lambda^2(\frac{1}{2n})^k$, which is non-decreasing in λ. For larger λ, the minimum simplifies to $\lambda(\frac{1}{\lambda})^k$, which is non-increasing in λ. Hence, the above expression is maximised for $\lambda(\frac{\lambda}{2n})^k = 1$ or, equivalently, $\lambda = (2n)^{k/(k+1)}$.

This suggests that the probability of finding the global optimum from the local optimum, $s_k^{\lambda_{\max}}$, as stated in Theorem 6.1 is maximal around $\lambda_{\max} = k$ and around $\lambda_{\max} = (2n)^{k/(k+1)}$. Since the expected time to find the optimum, $2\lambda_{\max}/s_k^{\lambda_{\max}}$, is proportional to the reciprocal of the improvement probability (with an additional factor of $2\lambda_{\max}$), this would imply that the expected optimisation time is locally minimal around these values. Note that, while Theorem 6.1 is rigorous, the above discussion about the possible location of minima is only semi-rigorous and partly based on upper and lower bounds of probabilities. Therefore, to analyse the parameter landscape we compute $2\lambda_{\max}/s_k^{\lambda_{\max}}$ exactly for $n \in \{125, 250, 500\}$, $k \in \{3, 4, 5, 6\}$ and $\lambda \in \{1, \ldots, n\}$ using the exact formula from Theorem 6.1. Figure 1 shows these computations.

As seen in Figure 1, the parameter landscape is a bimodal function for $k \geq 4$ with one minimum at or around $\lambda_{\max} = k$, as predicted by the discussion above. There is another local optimum for much larger values of $\lambda_{\max}$. Our above semi-rigorous arguments predicted a minimum around $\lambda_{\max} = (2n)^{k/(k+1)}$. The real minimum is attained for slightly smaller values of $\lambda_{\max}$. Recall that our prediction was based on a bound of the improvement probability, and we suspect that the bound is not precise enough to predict the exact location of the minimum.

The two local optima are surrounded by a basin of attraction, one narrow and one wide, that changes according to the relation between n and k. For small values of k, the wider basin of attraction contains the optimal parameter value and if k is small enough ($k \leq 3$) both basin of attractions merge, transforming the parameter landscape into a unimodal function. For larger k values the narrower basin of attraction contains the optimal parameter and it gets narrower for bigger problem sizes. We note that in this case the optimal parameter maximises the probability of finding the optimum during the mutation phase, hence we obtain much better performance when considering the offspring from both phases versus only the crossover phase.

This complex parameter landscape further shows that it can be hard to predict the correct parameter values to use for a given problem, especially if the problem is not well understood.

Additionally, since the parameter landscape is bimodal, if gradient descent was used for parameter tuning, it would easily get stuck far away from the optimal parameter value. This is particularly true for values of $k = o(n)$ large enough for $\lambda = k$ to be the best parameter (as seen in Figure 1 when $k = 6$) because we believe the basin of attraction around $\lambda = k$ has a size of $\Theta(k)$, while the other basin of attraction seems to have a size of $\Theta(n)$. Hence, it seems plausible that gradient descent would start in the wrong basin of attraction and only be able to find a locally optimal parameter.

Finally, to complement our runtime predictions, we executed experiments varying $\lambda_{\max}$ with $n = 60$ and $k = 4$ and compared them against the computed $\mathrm{E}(T^{\mathrm{eval}})$. The results of the experiments are shown in Figure 2. We note that although the average function evaluations follow closely our predictions, the standard deviation was in the same order of magnitude as the mean, meaning that there was a large variation from run to run. This large variation also results in the median being smaller than the mean, but both follow the same patterns with respect to $\lambda_{\max}$.

7 RESETTING λ

Any generic choice of a maximum $\lambda_{\max}$ bears the risk that the self-adjusting $(1 + (\lambda, \lambda))$ GA might get stuck with sub-optimal parameters. A solution to avoid this is to reset λ to 1 if $\lambda = \lambda_{\max}$ and there is another unsuccessful generation. This makes the algorithm cycle through the parameter space in unsuccessful generations. A similar modification was made in Goldman and Punch [2015], where the authors restart the parameter to $\lambda = 1$, but also restart the search from a random individual. We do not restart the search because for functions like Jump$_k$, restarts would run into the same set of local optima w.o.p. In Bassin et al. [2021], the authors reset λ, but instead of resetting to 1, they reset to the last successful parameter. We argue that, if the next step of the optimisation needs a lower value of λ than the used in the last successful generation, since λ only increases in unsuccessful generations the algorithm will never use the correct parameter.

7.1 Self-Adjusting $(1 + (\lambda, \lambda)$ GA Resetting λ

We analyse the simple strategy of resetting λ to 1 after an unsuccessful generation at $\lambda_{\max} = n$. This strategy takes advantage of two different behaviours. When hill-climbing, the algorithm uses self-adjustment to regulate λ and maintain its value in a *good* parameter range. Because of this, its optimisation time is not affected for problems like OneMax. However, when the algorithm encounters a local optimum, its behaviour is similar to the dynamic $(1 + 1)$ EA [Jansen and Wegener 2006] that cycles through different parameter regions, like $\lambda \sim n/2, \lambda = n$. This helps the algorithm to simulate random search and the $(1 + n)$ EA in one cycle. Furthermore, it is not affected by the modality of the parameter landscape. In addition, during every generation the crossover phase is still focusing on exploitation, generating offspring concentrated around the parent.

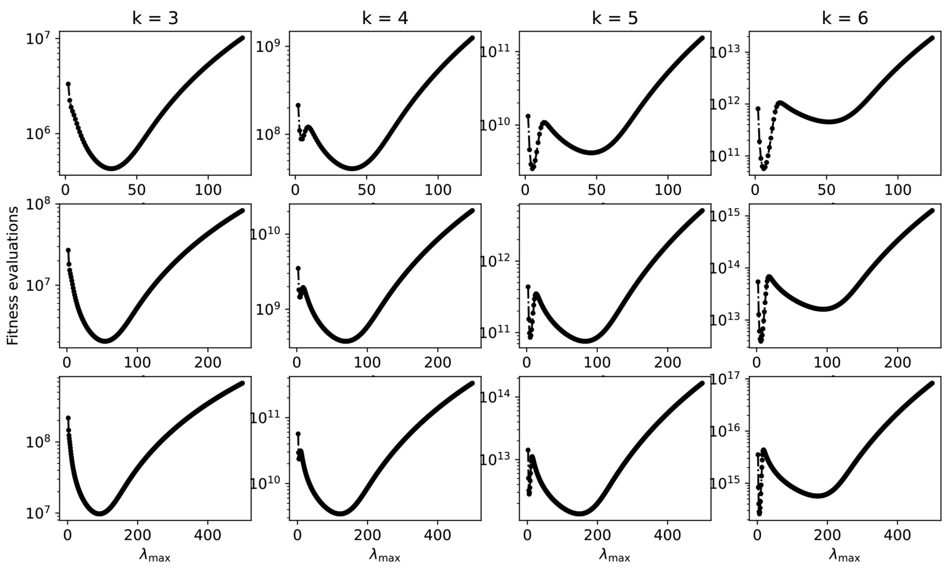

Fig. 1. $E(T^{eval})$ of the self-adjusting $(1 + (\lambda, \lambda))$ GA on Jump$_k$ with $n \in \{125, 250, 500\}$ and $k \in \{3, 4, 5, 6\}$ varying λ_{max}.

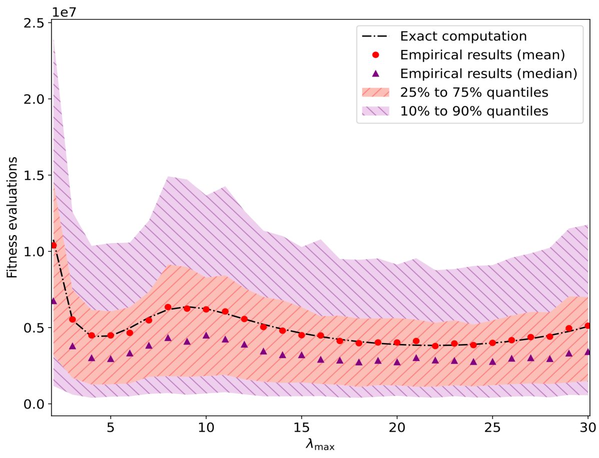

Fig. 2. $E(T^{eval})$, mean and median number of evaluations and 10%, 25%, 75%, 90% quantiles of the self-adjusting $(1 + (\lambda, \lambda))$ GA on Jump$_k$ with $n = 60$ and $k = 4$ varying λ_{max} over 1,000 runs.

This strategy is similar to the stagnation detection described in Rajabi and Witt [2020b, 2021b] in the sense that when $\lambda = n$, the algorithm is likely to be in a local optimum and the behaviour of the algorithm changes to explore different parameters. Once $\lambda = n$, different approaches could be used, such as sampling λ from a power-law distribution as in Antipov et al. [2020]. If a large jump is expected we could even use the non-standard parameters from Antipov and Doerr [2020] and come back to the self-adjusting behaviour once an improvement is found. Although these can be viable solutions we acknowledge that the parameter λ could increase to n without being in a local optimum, for example while optimising a function with low fitness-distance correlation or

ALGORITHM 3: The self-adjusting (1 + (λ, λ)) GA resetting λ.

```
1  Initialization: Sample $x \in \{0,1\}^n$ uniformly at random;
2  Initialize $\lambda \leftarrow 1, p \leftarrow \lambda/n, c \leftarrow 1/\lambda$;
3  Optimization: for $t = 1, 2, \ldots$ do
4      Sample $\ell$ from $\mathcal{B}(n,p)$;
       Mutation phase:
5      for $i = 1, \ldots, \lfloor\lambda\rceil$ do
6          Sample $x^{(i)} \leftarrow \text{flip}_\ell(x)$ and query $f(x^{(i)})$;
7      Choose $x' \in \{x^{(1)}, \ldots, x^{(\lambda)}\}$ with $f(x') = \max\{f(x^{(1)}), \ldots, f(x^{(\lambda)})\}$ u.a.r.;
       Crossover phase:
8      for $i = 1, \ldots, \lfloor\lambda\rceil$ do
9          Sample $y^{(i)} \leftarrow \text{cross}_c(x, x')$ and query $f(y^{(i)})$;
10     If $\{x', y^{(1)}, \ldots, y^{(\lambda)}\} \setminus \{x\} \neq \emptyset$, choose $y \in \{x', y^{(1)}, \ldots, y^{(\lambda)}\} \setminus \{x\}$ with
       $f(y) = \max\{f(x'), f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ u.a.r.;
11     otherwise, set $y := x$;
       Selection and update step:
12     if $f(y) \geq f(x)$ then $x \leftarrow y$;
13     if $f(y) > f(x)$ then $\lambda \leftarrow \max\{\lambda/F, 1\}$;
14     if $f(y) \leq f(x) \wedge \lambda = n$ then $\lambda \leftarrow 1$;
15     if $f(y) \leq f(x) \wedge \lambda \neq n$ then $\lambda \leftarrow \min\{\lambda F^{1/4}, n\}$;
```

by choosing F as a large constant as explored in Doerr and Doerr [2018] where the authors warn that λ can diverge even on simple problems if $F \geq 2.25$. Because of this we decide to study the algorithm without changing its behaviour drastically.

7.2 General Method

We show that the fitness-level method can be applied here as well. In contrast to Theorem 3.1, here improvement probabilities refer to the transitions of the (1 + (λ, λ)) GA, and we consider improvement probabilities across a whole cycle of parameter values.

THEOREM 7.1. *Given a canonical f-based partition $A_1, \ldots, A_{m+1}$, and s_i^{cycle} a lower bound on the probability of finding an improvement on level i during a cycle. Then for the self-adjusting (1 + (λ, λ)) GA resetting λ to 1, using $F > 1$ which may depend on n, we have*

$$E(T^{\text{eval}}) \leq O\left(\frac{F^{1/4} n}{F^{1/4} - 1}\right) \sum_{i=1}^{m} \frac{1}{s_i^{\text{cycle}}};$$

$$E(T^{\text{gen}}) \leq O(\lceil \log_F(n) \rceil) \sum_{i=1}^{m} \frac{1}{s_i^{\text{cycle}}}.$$

PROOF. Since the f-based partition is canonical, the current fitness level is left as soon as we encounter a successful generation. Until this happens, the self-adjusting (1 + (λ, λ)) GA cycles through all parameters for λ. We use Lemma 3.2 to show that for every time the algorithm cycles through all the parameters once, it uses $O(\lceil \log_F(n) \rceil)$ generations and $O(\frac{F^{1/4} n}{F^{1/4}-1})$ evaluations. The probability of leaving the current fitness level during a cycle is at least s_i^{cycle} by assumption. Hence, the expected number of cycles is at most $1/s_i^{\text{cycle}}$. Together, this proves the claimed bounds. □

7.3 Upper Bounds on Jump$_k$

To showcase how this bound can be used we show an upper bound for Jump$_k$. Note that, when applying Theorem 7.1, we may bound s_i^{cycle} from below by only considering the best parameter settings the algorithm can use for the current fitness level, making it easy to use. In our case we focus on values of λ close to k since in Section 6 we showed that for large k it gives the best performance. In addition, we also consider $\lambda = n$ because for small k and steps outside the local optimum this gives a better runtime, decreasing the complexity of the proof.

Theorem 7.2. *Let $F = F(n) > 1$ and $k \geq 2$. The expected number of evaluations of the self-adjusting (1 + (λ, λ)) GA resetting λ to 1 on Jump$_k$ is*

$$\min\left\{O\left(\frac{F^{1/4}n^k}{F^{1/4}-1}\right), O\left(\left(\frac{F^{(k+2)/4}}{F^{1/4}-1}\right)\left(\frac{en}{k}\right)^{k+1}\right)\right\}.$$

Proof. For the fitness levels $A_1 \dots A_{m-1}$ any individual can leave the current fitness level by increasing or decreasing the number of 1-bits. For these fitness levels we bound s_i^{cycle} by only considering generations with $\lambda = n$, therefore similar to Theorem 3.1 $s_i^{\text{cycle}} \geq \frac{s_i n}{1+s_i n}$, where s_i is a lower bound for the probability that the (1 + 1) EA with $p = 1/n$ creates an offspring in $A_{i+1} \dots A_{m+1}$ from a search point in A_i. Using the crude estimate $s_i \geq 1/(en)$ gives us an expected number of generations of $1/s_i^{\text{cycle}} \leq e + 1$ to leave any of these fitness levels.

For the local optimum in fitness level A_m, we use $s_m^{\text{cycle}} \geq \max\{s_m^{k^*}, s_m^n\}$ with $s_m^{k^*}$ and s_m^n being the probability of leaving A_m in one generation with $\lambda \in [\frac{k}{F^{1/4}}, k]$ and $\lambda = n$, respectively, and bound them separately. To compute s_m^n, as before, we use the probability that the (1 + 1) EA with $p = 1/n$ finds an improvement, that is, $s_m = (1/n)^k(1-1/n)^{n-k} \geq 1/(en^k)$ to obtain

$$s_m^n \geq \frac{s_m n}{1+s_m n} \geq \frac{1/(en^{k-1})}{1+1/(en^{k-1})} \geq \frac{1}{en^{k-1}+1}.$$

For $s_m^{k^*}$ we use Lemma 4.2 and the range $\lambda \in [\frac{k}{F^{1/4}}, k]$ as follows,

$$\begin{aligned} s_m^{k^*} &\geq \frac{\lfloor\lambda\rfloor}{2}(\lambda/n)^k(1-\lambda/n)^{n-k} \\ &\geq \frac{\lambda}{4}(\lambda/n)^k(1-\lambda/n)^{n-k} \geq \frac{k}{4F^{1/4}}\left(\frac{k}{F^{1/4}n}\right)^k\left(1-\frac{k}{n}\right)^{n-k} \geq \frac{k}{4F^{1/4}}\left(\frac{k}{F^{1/4}en}\right)^k, \end{aligned}$$

where the last inequality holds by Lemma 2.2. Applying Theorem 7.1 with $s_m^{\text{cycle}} \geq \max\{s_m^{k^*}, s_m^n\}$ and absorbing the expected times for fitness levels $i < m$ in the asymptotic notation proves the claimed bounds. □

Similar to Bassin et al. [2021], we can slow down the growth of λ. We accomplish this by cleverly choosing F in such a way that the algorithm is able to use every $\lambda \leq n$, ensuring that the algorithm uses the best parameter value. Choosing $F = (1+1/n)^4$ in Theorem 7.2 implies that $F^{(k+2)/4} = (1+1/n)^{k+2} \leq (1+1/n)^{n+2} = O(1)$, hence this factor can be dropped. Also note that $\frac{1}{F^{1/4}-1} = \frac{1}{(1+1/n)-1} = n$. Together, we obtain the following.

Corollary 7.3. *Let $F = (1+1/n)^4$ and $k \geq 2$. The expected optimisation time (in terms of fitness evaluations) of the modified self-adjusting (1 + (λ, λ)) GA on the Jump$_k$ function is*

$$\min\left\{O\left(n^{k+1}\right), O\left(n\left(\frac{en}{k}\right)^{k+1}\right)\right\}.$$

Table 1. Example of Runtime Bounds We Obtain for the Self-adjusting $(1 + (\lambda, \lambda))$ GA with Different Self-adjusting Mechanisms

Mechanism	Bounds	
	General bound	$\textsc{Jump}_k$
	Expected number of evaluations	
Vanilla	$O(dn) + 2\sum_{i=1}^{m} \frac{1}{s_i}$	$(1 \pm o(1)) \cdot 2n^k \left(1 - \frac{1}{n}\right)^{-n+k}$
Capping λ at $\lambda_{\max} = k$	–	$O(k) + 4\left(\frac{n}{k}\right)^k \left(1 - \frac{k}{n}\right)^{-n+k}$
Capping λ at $\lambda_{\max} = n/2$	$O(dn) + \frac{2^{n+4}}{\lvert \mathrm{OPT} \rvert}$	$\begin{cases} O\left(\left(\frac{n}{2}\right)^k\right) & \text{if } k \leq \frac{\log(n/2)}{2}, \\ O\left(\min\{(2n)^{k-1}, 2^n\}\right) & \text{otherwise.} \end{cases}$
Resetting λ	$O\left(\frac{F^{1/4}n}{F^{1/4}-1}\right) \sum_{i=1}^{m} \frac{1}{s_i^{\text{cycle}}}$	$\min\left\{O\left(\frac{F^{1/4}n^k}{F^{1/4}-1}\right), O\left(\left(\frac{F^{(k+2)/4}}{F^{1/4}-1}\right)\left(\frac{en}{k}\right)^{k+1}\right)\right\}$
Resetting λ with $F = (1 + 1/n)^4$	–	$\min\left\{O\left(n^{k+1}\right), O\left(n\left(\frac{en}{k}\right)^{k+1}\right)\right\}$
	Expected number of generations	
Vanilla	$\lceil 4 \log_F(n) \rceil + 6d + \frac{1}{n}\sum_{i=1}^{m} \frac{1}{s_i}$	$(1 + o(1)) \cdot 2n^{k-1} \left(1 - \frac{1}{n}\right)^{-n+k}$
Capping λ at $\lambda_{\max} = n/2$	$O(d + \log n) + \frac{2^{n+3}}{n\lvert \mathrm{OPT} \rvert}$	–
Resetting λ	$O(\lceil \log_F(n) \rceil) \sum_{i=1}^{m} \frac{1}{s_i^{\text{cycle}}}$	–

Note that some of the results are subject to further conditions, e.g., $4 \leq k \leq (1 - \varepsilon)n/2$.

Comparing the bound of Corollary 7.3 against the expected optimisation time of the $(1 + 1)$ EA with optimal mutation rate of k/n, which is $T_{\text{opt}} = \Theta((\frac{en}{k})^k)$ [Doerr et al. 2017], our bound is larger than T_{opt} by a factor of $O(n^2/k)$ without the need to know the jump size k in advance. Our bound is only by a factor of $O(n^2/k^2)$ larger than the bound for the $(1 + 1)$ EA with the heavy-tailed ("fast") mutation operators from Doerr et al. [2017] with the recommended parameter $\beta = 1.5$.

8 EXPERIMENTAL RESULTS

In the previous sections we performed asymptotic analyses focusing mainly on $\textsc{Jump}_k$ functions (summarised in Table 1). In this section, we conduct an experimental analysis that aims to accomplish the following goals: (1) to complement our theoretical results with precise runtime results for concrete problem instances and jump sizes, (2) to compare the runtime of all the proposed versions of the self-adjusting $(1 + (\lambda, \lambda))$ GA against related algorithms, and (3) to test how the mathematical results obtained translate to other fitness landscapes.

In the experiments of this section, we ran an implementation[4] of all the different modifications of the self-adjusting $(1 + (\lambda, \lambda))$ GA studied in previous sections on different problems and compared them against:

- The $(1 + 1)$ EA with the standard mutation rate $p = 1/n$.
- The $(1 + 1)$ EA with the optimal choice of $p = k/n$ (on $\textsc{Jump}_k$).

[4]The complete implementation and results can be found on GitHub (https://github.com/mariohevia/Parameter-Control-Mechanisms-Genetic-Algorithm)

- The "fast" (1 + 1) fEA [Doerr et al. 2017] with a heavy-tailed mutation rate: $p = r/n$ and $r \sim \mathrm{pow}(1.5, n)$.
- The $(1 + (\lambda, \lambda))$ GA with a heavy-tailed choice of $\lambda \sim \mathrm{pow}(2.5, n/2)$ [Antipov et al. 2020] that we call the heavy-tailed $(1 + (\lambda, \lambda))$ GA.
- The $(1 + (\lambda, \lambda))$ GA with non-standard parameters and a heavy-tailed choice of $\lambda \sim \mathrm{pow}(2, \min\{5^{n/10}, 10^6\})$, $p = c = \sqrt{s/n}$ and $s \sim \mathrm{pow}(1, n)$ [Antipov and Doerr 2020].

We note that for all versions of the $(1 + (\lambda, \lambda))$ GA including the ones with heavy-tailed parameters we consider the offspring from both mutation and crossover phases in the selection phase.

The parameter selection for the heavy-tailed algorithms was made following the recommendations of each study. In particular, for the $(1 + (\lambda, \lambda))$ GA with non-standard parameters, the upper bound for λ in the distribution is suggested to be exponential in n but because of memory demand from storing the probabilities in the power-law distribution, during implementation we impose a limit of 10^6 independent of n.

All experiments comprise of 500 runs for each algorithm-problem pair, recording the number of fitness evaluations to reach the optimum and the average is reported unless otherwise stated.

In the following we report on results of statistical tests executed as follows. We performed Mann-Whitney U Test for all pairs of algorithms compared in the text. We performed two-sided tests to check whether the two input distributions differ or not, followed by one-sided tests both ways to confirm which algorithm is stochastically faster than the other. We report a comparison as statistically significant if the p-values of the two-sided test and that of the respective one-sided test both satisfied $p \leq 0.01$, that is, a confidence level of 0.01. When comparing one algorithm against several others, we performed a pairwise comparison for each pair as explained before and applied the Bonferroni correction.

8.1 Empirical Analyses on Jump$_k$ Functions

For the class of Jump$_k$ functions we performed two experiments focused on the effects of the problem size and the jump size separately.

In the first experiment, we used a jump size $k = 4$ and n varying from 20 to 160 shown in Figure 3. The $(1 + (\lambda, \lambda))$ GA with non-standard parameters has the best performance for $n \geq 40$; all comparisons are statistically significant. This is expected because the selection of parameters λ, c and p are tailored towards Jump$_k$ functions. The heavy-tailed $(1 + (\lambda, \lambda))$ GA is able to use better λ values than the vanilla self-adjusting $(1 + (\lambda, \lambda))$ GA, performing statistically significantly better for all n although it performs statistically significantly worse than both resetting strategies and capping λ to $n/2$ for $n \geq 80$. It is worth pointing out that the self-adjusting $(1 + (\lambda, \lambda))$ GA capping λ to $n/2$ and both versions resetting λ scale better with n than the (1 + 1) fEA. The self-adjusting $(1 + (\lambda, \lambda))$ GA variants are statistically slower than the (1 + 1) fEA for $n = 20$ but for $n \geq 80$ they are statistically faster. This is most likely because the parameter landscape for instances with large n resemble the ones shown in the second column of Figure 1 in Section 6, where using values of λ that are linear in n help the crossover phase find the optimum even faster than the mutation phase with $\lambda = k$ and by extension faster than any mutation rate for the (1 + 1) EA. Despite this possible advantage, these algorithms perform statistically significantly worse than the (1 + 1) EA with $p = k/n$ on these Jump$_k$ instances because they waste some evaluations with non-optimal parameters. To inquire further, we performed additional experiments on Jump$_k$ with $n = 160$ and $k = 2$ that are not shown in the figures where the self-adjusting $(1 + (\lambda, \lambda))$ GA resetting λ with $F = (1+1/n)^4$ is statistically significantly faster than the (1 + 1) EA with optimal parameter values, showing that for some instances of Jump$_k$ the $(1 + (\lambda, \lambda))$ GA with standard parameter settings can be faster than the (1 + 1) EA with the optimal mutation rate.

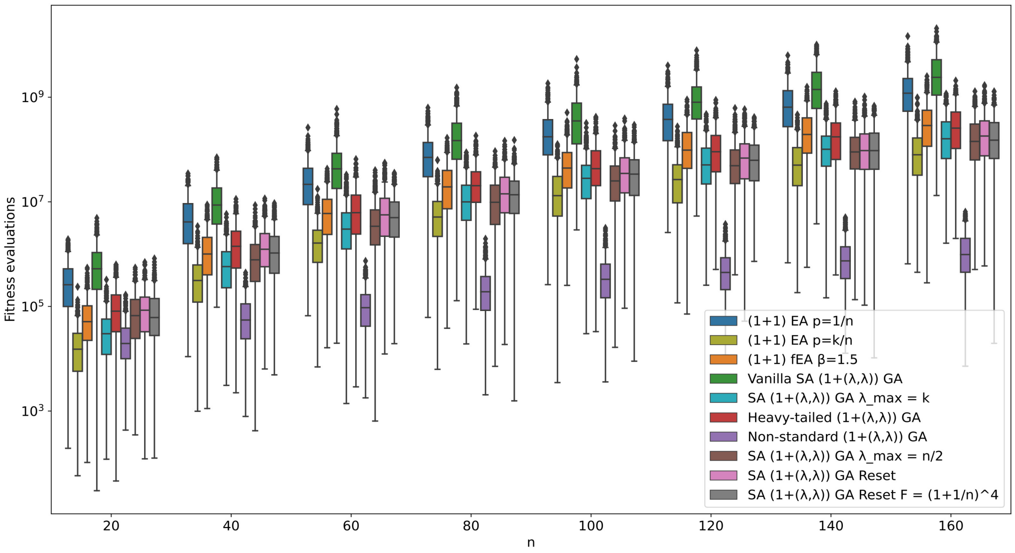

Fig. 3. Box plot of the number of fitness evaluations on Jump$_k$ with $n = \{20, 40, 60, \ldots, 160\}$ and $k = 4$ over 500 runs.

For the second experiment, we aimed to explore the effects of the jump size k. We used a small problem size of $n = 20$, varying the jump size from 2 to 7 shown in Figure 4. In this case we see that for larger values of k the self-adjusting $(1 + (\lambda, \lambda))$ GA capping λ to $n/2$ and resetting λ do not excel but they are still competitive with the $(1 + 1)$ fEA. This in conjunction with the good performance of the self-adjusting $(1 + (\lambda, \lambda))$ GA capping λ to k indicates that the mutation phase is much better at finding the optimum for large k. This is in agreement with our analysis of the parameter landscape from Section 6. Lastly in this experiment the non-standard $(1 + (\lambda, \lambda))$ GA does not excel. We suspect that this is caused by the small problem size $n = 20$.

In both plots shown in Figures 3 and 4, the average number of fitness evaluations for the self-adjusting $(1 + (\lambda, \lambda))$ GA with standard parameters and with $\lambda_{\max} = k$ is around twice the average for the $(1 + 1)$ EA with $p = 1/n$ and $p = k/n$, respectively, as predicted by our theoretical results.

8.2 Empirical Analyses on OneMax

In this section we study the algorithms on OneMax. We consider this problem because the self-adjusting $(1 + (\lambda, \lambda))$ GA is the fastest known unbiased genetic algorithm on OneMax (with an expected number of $O(n)$ fitness evaluations, whereas the $(1 + 1)$ EA needs $\Theta(n \log n)$ expected fitness evaluations) and we want to illustrate the effects of the different parameter control strategies on the performance on OneMax.

Figure 5 shows the average number of fitness evaluations, normalised by the problem size n with the x-axis (problem size) being log-scaled. Figure 5 gives us the following three insights. First, the $(1 + (\lambda, \lambda))$ GA with non-standard parameters stands out as the algorithm with the worst performance by far; all comparisons are statistically significant. This can be attributed to the choice of parameters, mainly the larger offspring population size. Secondly, all the other variants of the $(1 + (\lambda, \lambda))$ GA have a good performance and the fact that the normalised time seems to increase slower than the $(1 + 1)$ EA suggests that they have an asymptotic runtime faster than $n \log n$. Lastly, resetting and capping λ to $n/2$ does not affect the performance of the algorithm on OneMax, compared to the vanilla version.

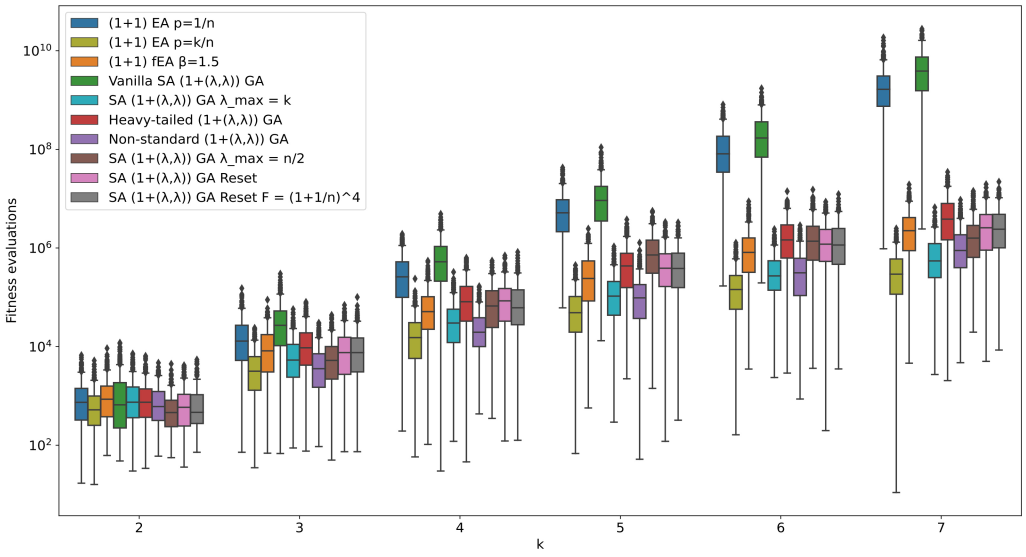

Fig. 4. Box plot of the number of fitness evaluations on Jump_k with $n = 20$ and $k = \{2, 3, 4, 5, 6, 7\}$ over 500 runs.

8.3 Empirical Analyses on Other Benchmark Functions

To see whether the theoretical results extend to other fitness landscapes we study the algorithms with generic parameter choices on five different functions. During these experiments we put a limit of 2.1 billion evaluations which is close to the upper limit of the variable **int** in C++ ($2^{31} - 1$). The mean is computed by assigning the limit of 2.1 billion evaluations to all incomplete runs.

We consider the multimodal functions NearestPeak (NP) and WeightedNearestPeak (WNP) proposed by Jansen and Zarges [2016]. These functions are characterised by a set of peaks that define a fitness landscape. The characteristics of their landscape depends on the number, position, slope and height of these peaks. Let p_i be the bit string representing peak i, then the weight W_i of peak i is $W_i = (n - \mathrm{H}(x, p_i))a_i + b_i$ where x is the current solution, a_i is the slope and b_i is the height of the peak. The difference between NP and WNP is that for NP the fitness is the weight of the nearest peak (in Hamming distance), while for WNP the fitness is the biggest weight with respect to the current search point. Both problems have a basin of attraction around the peaks, creating local and global optima.

The other three functions are well known NP-hard problems: Partition, Ising Spin Glass (ISG), and MAX-3SAT. The Partition optimisation problem is to divide a set of positive weights into two disjoint subsets such that the largest subset sum is minimised. The ISG problem is a problem derived from physics. The Ising model consists of discrete variables that represent magnetic dipole moments of atomic “spins” that can be in one of two states (+1 or −1). The spins are arranged in a graph describing the interactions strengths (edges) between spins (vertices). Neighbouring spins with the same state have a lower interaction than those with a different state. The ISG problem is to set the signs of all spins to minimise interactions. The MAX-3SAT problem is related to the more common 3-SAT problem. A MAX-3SAT instance is a Boolean formula in conjunctive normal form, that is, a logical conjunction of one or more clauses, where a clause is a logical disjunction of three binary variables, some of which can be negated. The search space $S = \{0, 1\}^n$ encodes the choice of truth values of the binary variables in the Boolean formula and the optimisation problem

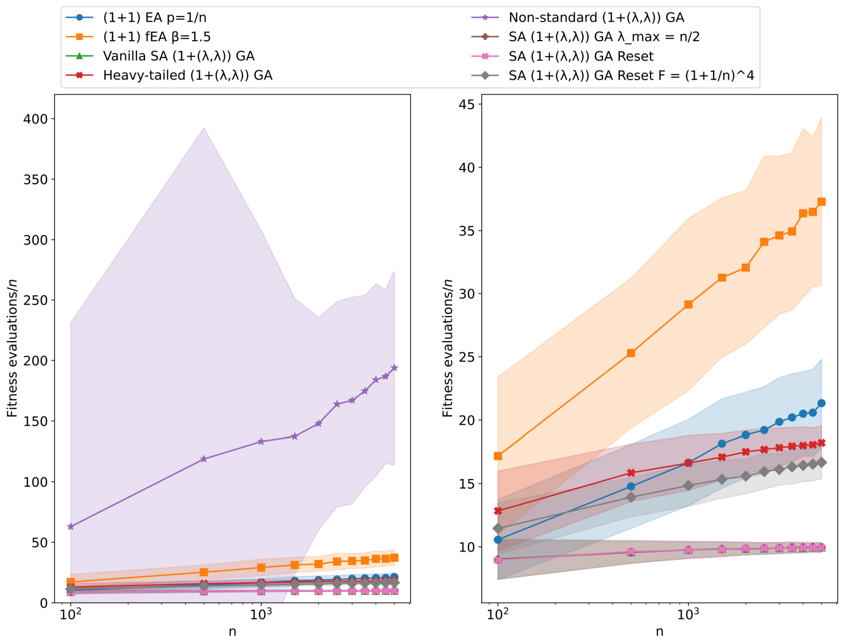

Fig. 5. Average number of fitness evaluations and standard deviation on OneMax with varying n over 500 runs of all tested algorithms (left) and a more detailed view of the best performing algorithms (right).

is to find a solution that satisfies the maximum number of clauses. Since the clauses are disjunctive, they are satisfied if at least one of its literals is satisfied.

For all problems we used the same set of instances (one run per instance) across all algorithms because some instances might be more difficult than others. The instances were generated at random. The NP and WNP instances were generated with a problem size of 50 and different number of peaks with one global optimum at 1^n. The peaks were defined by $[|p_i|_1, a_i, b_i]$ where the values correspond to the number of ones, slope and height of peak i. Every peak was generated sampling uniformly at random a search point with the number of ones specified in p_i. The following peaks (including the global optimum) were used:

- NearestPeak:
  (a) [50, 5, 0], [42, 2, 0], [41, 4, 0], [42, 2, 10]
  (b) [50, 5, 0], [40, 3, 0], [42, 2, 0], [44, 1, 0], [42, 2, 10]
  (c) [50, 10, 0], [40, 9, 0], [40, 9, 0], [40, 9, 0], [40, 9, 0], [40, 9, 0]
- WeightedNearestPeak:
  (a) [50, 10, 0], [40, 9, 0], [41, 9, 0]
  (b) [50, 10, 0], [40, 9, 0], [40, 9, 0], [40, 9, 0], [40, 9, 0], [40, 9, 0]

For Partition the problem size is 500, with weights selected uniformly at random in the range [0, 1]. Given that Partition is NP-hard we use a deterministic approximation algorithm called Longest Processing Time [Neumann and Witt 2010] to set a fitness goal. If an algorithm finds a solution with the same or higher fitness we consider the problem solved. In the case of ISG and MAX-3SAT we use the same 100 instances per problem size used by Goldman and Punch [2015] that were also generated at random in their work. Following Goldman and Punch [2015], we executed one run per instance, resulting in 100 runs per problem size.

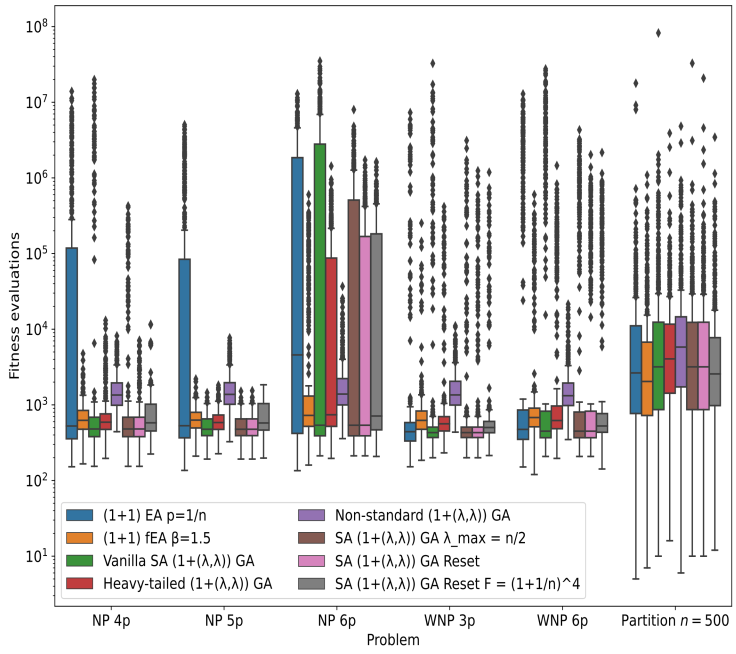

Fig. 6. Box plot of the number of fitness evaluations on NearestPeak (NP), WeightedNearestPeak (WNP) with $n = 50$ and Partition with $n = 500$ over 500 runs.

In Figure 6 we see the results on NP, WNP and Partition instances. For these problems all algorithms found the optimum on all instances within the time budget, but given that there are a high number of outliers we present the data in box plot form. In Figure 6 the number of peaks is mentioned after the problem, where $5p$ means five peaks i.e. four local optima and one global optimum. For these problems we see a common trend: the original self-adjusting $(1 + (\lambda, \lambda))$ GA has the largest outliers in five of the six sets of experiments. This might be caused by the algorithm increasing λ to its maximum when the algorithm gets stuck. The modifications studied in this paper and the heavy-tailed $(1 + (\lambda, \lambda))$ GA seem to reduce the runtime of the outliers while maintaining a similar median, with the capping strategy having a smaller impact in the runtime of the outliers than the other algorithms. We would like to note that for all the $(1 + (\lambda, \lambda))$ GA versions excluding the $(1 + (\lambda, \lambda))$ GA with non-standard parameters the median number of evaluations is similar because the algorithms behave similarly when they do not get stuck in a local optimum as seen in the experiments on OneMax and this happens in most of the runs. In addition, we can see that the $(1 + (\lambda, \lambda))$ GA with non-standard parameters tends to have a slightly larger median runtime in most cases (statistically significantly larger for all comparisons on all problems except for NP 6p). This is most likely because when an easy instance is found the algorithm spends more time, similar to OneMax. On the other hand, the outliers for the $(1 + (\lambda, \lambda))$ GA with non-standard parameters tend to have a smaller runtime compared with the outliers for other algorithms. We attribute this to the ability of the algorithm to perform large jumps when stuck in a local optimum on these *hard* instances.

The next set of experiments were made on the Ising Spin Glass problem. Figure 7 shows a box plot of the number of fitness evaluations, the mean and number of failures. We include the box plot because the mean can be disturbed by failed runs capped at the limit, but the boxplot shows the median that is unaffected by this as long as half of the runs are able to find the optimum. The mean

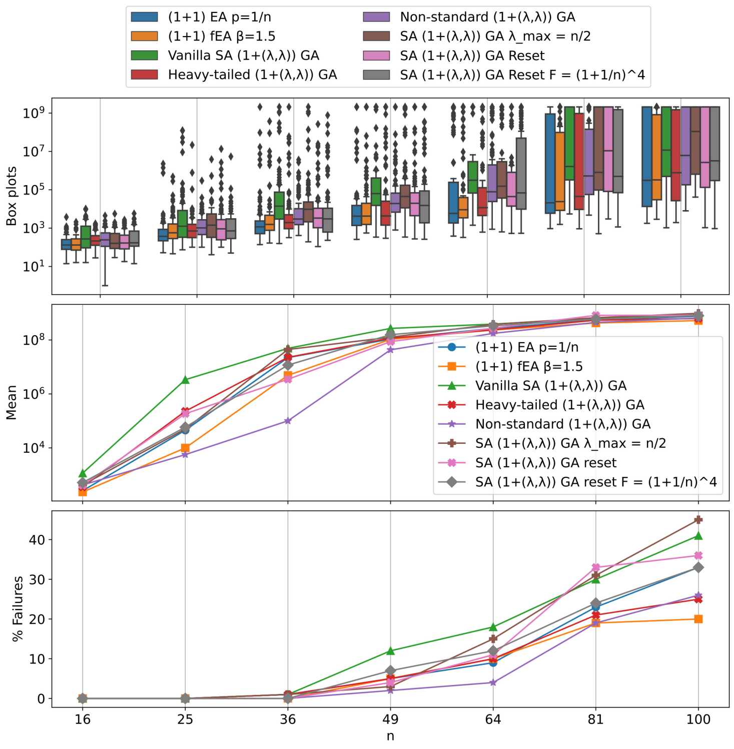

Fig. 7. Box plot of the number of fitness evaluations (top), average number of fitness evaluations (middle), and percentage of failed runs (bottom) on the Ising Spin Glass problem varying the problem size over 200 runs.

is still useful because it shows the effectiveness of an algorithm to escape local optima. A prime example is the $(1 + (\lambda, \lambda))$ GA with non-standard parameters, which has the smallest mean for most problem sizes, but its median is almost an order of magnitude larger than the smallest median.

Similar to other problems, on ISG the original self-adjusting $(1 + (\lambda, \lambda))$ GA tends to have larger outliers than all other algorithms, which is reflected in the larger mean and higher number of failures. This is attenuated by the variations studied here and the heavy-tailed $(1 + (\lambda, \lambda))$ GA. An interesting point is that the $(1 + 1)$ EA is statistically significantly faster than all the $(1 + (\lambda, \lambda))$ GA variations with and without standard parameters (excluding the heavy-tailed $(1 + (\lambda, \lambda))$ GA and the resetting mechanism with $F = (1 + 1/n)^4$) in at least three of the problem sizes ($n = 49, 64, 81$), which might indicate that there is a bad fitness-distance correlation for these instances. The results of the heavy-tailed $(1+(\lambda, \lambda))$ GA and the resetting mechanism with $F = (1+1/n)^4$ can be explained by the λ values used by these algorithms. The heavy-tailed $(1 + (\lambda, \lambda))$ GA samples $\lambda = 1$ in most generations and $F = (1 + 1/n)^4$ maintains the value of $\lfloor \lambda \rceil = 1$ for longer, making both algorithms behave exactly as the $(1 + 1)$ EA in these generations, therefore they show a similar performance.

The results for MAX-3SAT are shown in Figure 8. In this case there are no statistically significant differences between all the algorithms, but still there are some observations we can obtain. Similar to the ISG problem for MAX-3SAT we show the median, mean and number of failed runs. Both the $(1 + 1)$ EA and the original self-adjusting $(1 + (\lambda, \lambda))$ GA run into local optima and it is hard for them to escape, making the number of outliers, mean and number of failures increase. In contrast, all the self-adjusting $(1 + (\lambda, \lambda))$ GA variants have a smaller mean and smaller number of failures. Once

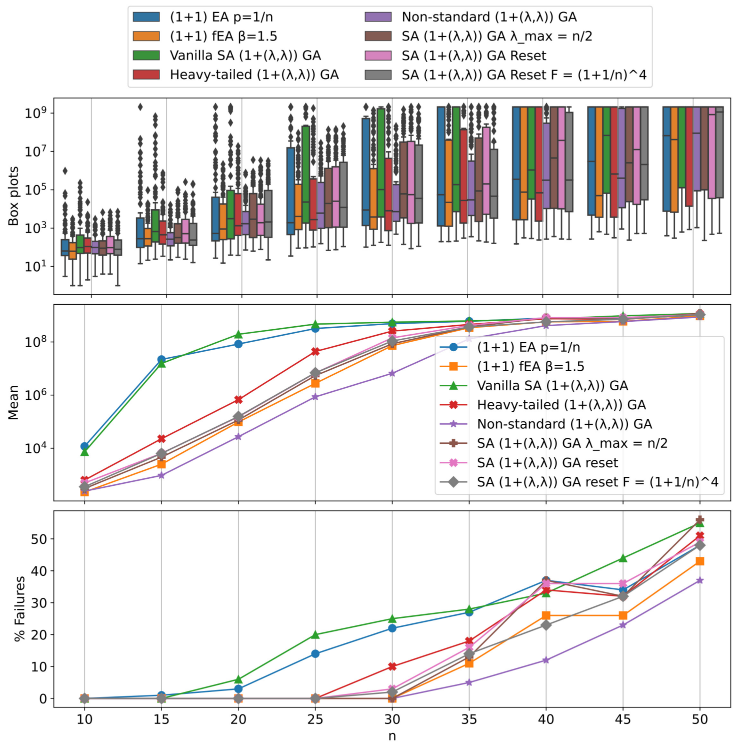

Fig. 8. Box plot of the number of fitness evaluations (top), average number of fitness evaluations (middle), and percentage of failed runs (bottom) on the MAX-3SAT problem varying the problem size over 100 runs.

again the $(1 + (\lambda, \lambda))$ GA with non-standard parameters has the smallest mean and number of failures; we attribute this to its capacity to jump out of local optima faster than all the other algorithms.

8.4 Discussion

We have seen that the original self-adjusting $(1 + (\lambda, \lambda))$ GA has a poor performance compared to all algorithms studied, on most problems. This shows that, as expected from theoretical results in Sections 3 and 4, the self-adjusting $(1 + (\lambda, \lambda))$ GA easily increases λ to its maximum on difficult parts of the optimisation, degrading the performance of the algorithm. Both resetting and capping λ improved the performance on almost all difficult problems and instances without affecting the performance on easy problems. Slowing down the increase of λ with small values of F can be helpful on difficult problems but as a trade-off it can slightly increase the runtime on easy problems.

On difficult problems both the (1 + 1) fEA and the $(1 + (\lambda, \lambda))$ GA with non-standard parameters tend to have fewer outliers and fewer failures than all the versions of the self-adjusting $(1 + (\lambda, \lambda))$ GA. This is largely because both algorithms are tailored to perform large jumps with high probability when stuck in local optima. Because of this, both algorithms have a worse performance on easy problems with the $(1 + (\lambda, \lambda))$ GA, with non-standard parameters being the worst algorithm when easy problems (OneMax, WNP and some instances of NP) are encountered.

9 CONCLUSIONS

We have provided a rigorous runtime analysis of the self-adjusting $(1 + (\lambda, \lambda))$ GA (considering the best offspring from the mutation phase during the selection step) for general function classes by presenting a fitness-level theorem for the self-adjusting (1 + (λ, λ)) GA that is easy to use and enables a transfer of runtime bounds from the (1 + 1) EA to the self-adjusting (1 + (λ, λ)) GA.

The parameter control mechanism in the original self-adjusting (1 + (λ, λ)) GA tends to diverge λ to its maximum on multimodal problems. Then the algorithm effectively simulates a (1 + n) EA with the default mutation rate of $1/n$. For the multimodal benchmark problem class Jump_k, we proved upper and lower runtime bounds that are tight up to lower-order terms, showing that despite using crossover the self-adjusting (1 + (λ, λ)) GA is not as efficient as other crossover-based algorithms.

Imposing a maximum value $\lambda_{\max}$ can improve performance, however then the problem remains of how to set $\lambda_{\max}$ if no problem-specific knowledge is available. The generic choice $\lambda_{\max} = n/2$ makes the self-adjusting (1 + (λ, λ)) GA perform random search steps during the mutation phase in case the algorithm gets stuck. This guards against deceptive problems and the algorithm still retains its original exploitation capabilities in the crossover phase.

We showed that the parameter landscape with respect to the impact of $\lambda_{\max}$ on the runtime of the self-adjusting (1 + (λ, λ)) GA on Jump_k is bimodal for appropriate n and k, and the optimal parameter changes drastically depending on the problem size n and the jump size k. The parameter landscape features two optima, one located in a wide basin of attraction that maximises the probability of finding the global optimum in the crossover phase and the other hidden in a narrow basin of attraction that maximises the probability of finding the global optimum during the mutation phase. For small k the wider basin of attraction leads to the optimal parameter value, but for larger k the optimal parameter value changes to the narrow basin of attraction. For large k, considering the mutation offspring in the selection phase gives a large performance improvement to the (1 + (λ, λ)) GA. This fluctuating parameter landscape combined with it being bimodal makes parameter selection difficult.

Additionally, we investigated resetting λ to 1 after an unsuccessful generation at the maximum value. This makes the self-adjusting (1 + (λ, λ)) GA cycle through the parameter space, approaching optimal or near-optimal parameter values in every cycle in spite of the parameter landscape. We recommend to choose $F = (1 + 1/n)^4$ if a slow growth of λ is desired. For Jump_k, this strategy gives the same expected runtime as that of the (1 + 1) EA with the optimal mutation rate and fast mutation operators, up to small polynomial factors.

Finally, the empirical results show that the original self-adjusting (1 + (λ, λ)) GA tends to increase λ to its maximum, and this yields the worst performance on most multimodal problems tested, while the modifications of the parameter control mechanism improved its performance. Additionally, these modifications do not significantly affect the algorithm's performance on easy problems. This suggests that for common optimisation problems it is better to use the parameter control variants (capping or resetting λ) studied here than the original self-adjusting (1 + (λ, λ)) GA.

REFERENCES

Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. 2020. Fast mutation in crossover-based algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'20).* ACM, 1268–1276.

Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. 2021. Lazy parameter tuning and control: Choosing all parameters randomly from a power-law distribution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'21).* ACM, 1115–1123.

Denis Antipov and Benjamin Doerr. 2020. Runtime analysis of a heavy-tailed (1 + (λ, λ)) genetic algorithm on jump functions. In *Parallel Problem Solving from Nature – PPSN XVI.* Springer, 545–559.

Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. 2019. A tight runtime analysis for the (1 + (λ, λ)) GA on LeadingOnes. In *Proceedings of the 15th ACM-SIGEVO Conference on Foundations of Genetic Algorithms (FOGA'19).* ACM, 169–182.

Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. 2020. The (1 + (λ, λ)) GA is even faster on multimodal problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'20).* ACM, 1259–1267.

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

Anton O. Bassin, Maxim V. Buzdalov, and Anatoly A. Shalyto. 2021. The "one-fifth rule" with rollbacks for self-adjustment of the population size in the $(1 + (\lambda, \lambda))$ genetic algorithm. *Autom. Control. Comput. Sci.* 55, 7 (2021), 885–902.

Süntje Böttcher, Benjamin Doerr, and Frank Neumann. 2010. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Proc. of Parallel Problem Solving from Nature – PPSN XI*, Vol. 6238. Springer, 1–10.

Maxim Buzdalov and Benjamin Doerr. 2017. Runtime analysis of the $(1 + (\lambda, \lambda))$ genetic algorithm on random satisfiable 3-CNF formulas. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'17)*. ACM, 1343–1350.

Maxim Buzdalov, Mikhail Kever, and Benjamin Doerr. 2015. Upper and lower bounds on unrestricted black-box complexity of $\text{Jump}_{n,\ell}$. In *Evolutionary Computation in Combinatorial Optimization*. Springer, 209–221.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (3rd ed.). The MIT Press.

Dogan Corus, Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. 2018. Level-based analysis of genetic algorithms and other search processes. *IEEE Transactions on Evolutionary Computation* 22, 5 (2018), 707–719.

Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. 2018. Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation* 22, 3 (2018), 484–497.

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

Benjamin Doerr. 2020. *Probabilistic Tools for the Analysis of Randomized Optimization Heuristics*. Springer, 1–87.

Benjamin Doerr and Carola Doerr. 2018. Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm. *Algorithmica* 80, 5 (01 May 2018), 1658–1709.

Benjamin Doerr and Carola Doerr. 2020. *Theory of Parameter Control for Discrete Black-box Optimization: Provable Performance Gains through Dynamic Parameter Choices*. Springer, 271–321.

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

Benjamin Doerr, Carola Doerr, and Johannes Lengler. 2021. Self-adjusting mutation rates with provably optimal success rules. *Algorithmica* 83, 10 (2021), 3108–3147.

Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. 2019. The $(1 + \lambda)$ evolutionary algorithm with self-adjusting mutation rate. *Algorithmica* 81 (2019), 593–631.

Benjamin Doerr and Timo Kötzing. 2021. Lower bounds from fitness levels made easy. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'21)*. ACM, 1142–1150.

Benjamin Doerr and Timo Kötzing. 2021. Multiplicative up-drift. *Algorithmica* 83, 10 (2021), 3017–3058.

Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. 2017. Fast genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'17)*. ACM, 777–784.

Benjamin Doerr, Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2018. On the runtime analysis of selection hyper-heuristics with adaptive learning periods. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'18)*. ACM, 1015–1022.

Benjamin Doerr and Frank Neumann. 2021. A survey on recent progress in the theory of evolutionary algorithms for discrete optimization. *ACM Trans. Evol. Learn. Optim.* 1, 4 (2021), 43.

Benjamin Doerr, Carsten Witt, and Jing Yang. 2020. Runtime analysis for self-adaptive mutation rates. *Algorithmica* 83 (2020), 1012–1053.

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

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

Michael Foster, Matthew Hughes, George O'Brien, Pietro S. Oliveto, James Pyle, Dirk Sudholt, and James Williams. 2020. Do sophisticated evolutionary algorithms perform better than simple ones?. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'20)*. ACM, 184–192.

Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Samadhi Nallaperuma, Frank Neumann, and Martin Schirneck. 2016. Fast building block assembly by majority vote crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'16)*. ACM, 661–668.

Tobias Friedrich and Frank Neumann. 2017. What's hot in evolutionary computation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31. 5064–5066.

Tobias Friedrich, Francesco Quinzan, and Markus Wagner. 2018. Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'18)*. ACM, 293–300.

Brian W. Goldman and William F. Punch. 2015. Fast and efficient black box optimization using the parameter-less population pyramid. *Evolutionary Computation* 23, 3 (2015), 451–479.

Walter J. Gutjahr. 2008. First steps to the runtime complexity analysis of ant colony optimization. *Computers and Operations Research* 35, 9 (2008), 2711–2727.

George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. 2019. On the impact of the cutoff time on the performance of algorithm configurators. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'19)*. ACM, 907–915.

George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. 2020a. Analysis of the performance of algorithm configurators for search heuristics with global mutation operators. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'20)*. ACM, 823–831.

George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. 2020b. Fast perturbative algorithm configurators. In *Parallel Problem Solving from Nature – PPSN XVI*. Springer, 19–32.

Mario A. Hevia Fajardo. 2019. An empirical evaluation of success-based parameter control mechanisms for evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO' 19)*. ACM, 787–795.

Mario A. Hevia Fajardo and Dirk Sudholt. 2020. On the choice of the parameter control mechanism in the $(1 + (\lambda, \lambda))$ genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO' 20)*. ACM, 832–840.

Mario Alejandro Hevia Fajardo and Dirk Sudholt. 2021. Self-adjusting offspring population sizes outperform fixed parameters on the Cliff function. In *Proceedings of the 16th Workshop on Foundations of Genetic Algorithms (FOGA'21)*. ACM, 5:1–5:15.

Thomas Jansen and Ingo Wegener. 2002. On the analysis of evolutionary algorithms—a proof that crossover really can help. *Algorithmica* 34, 1 (2002), 47–66.

Thomas Jansen and Ingo Wegener. 2006. On the analysis of a dynamic evolutionary algorithm. *Journal of Discrete Algorithms* 4, 1 (2006), 181–199.

Thomas Jansen and Christine Zarges. 2016. Example landscapes to support analysis of multimodal optimisation. In *Parallel Problem Solving from Nature – PPSN XIV*. Springer, 792–802.

Jörg Lässig and Dirk Sudholt. 2011. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proceedings of the 11th Workshop Proceedings on Foundations of Genetic Algorithms (FOGA'11)*. ACM, 181–192.

Jörg Lässig and Dirk Sudholt. 2014a. Analysis of speedups in parallel evolutionary algorithms and $(1 + \lambda)$ EAs for combinatorial optimization. *Theoretical Computer Science* 551 (2014), 66–83.

Jörg Lässig and Dirk Sudholt. 2014b. General upper bounds on the running time of parallel evolutionary algorithms. *Evolutionary Computation* 22, 3 (2014), 405–437.

Per Kristian Lehre and Carsten Witt. 2012. Black-box search by unbiased variation. *Algorithmica* 64, 4 (2012), 623–642.

Per Kristian Lehre and Xin Yao. 2009. On the impact of the mutation-selection balance on the runtime of evolutionary algorithms. In *Proceedings of the 10th ACM SIGEVO Workshop on Foundations of Genetic Algorithms (FOGA'09)*. ACM, 47–58.

Johannes Lengler. 2020. A general dichotomy of evolutionary algorithms on monotone functions. *IEEE Transactions on Evolutionary Computation* 24, 6 (2020), 995–1009.

Johannes Lengler, Dirk Sudholt, and Carsten Witt. 2021. The complex parameter landscape of the compact genetic algorithm. *Algorithmica* 83, 4 (2021), 1096–1137.

Andrei Lissovoi, Pietro Oliveto, and John Alasdair Warwicker. 2020a. How the duration of the learning period affects the performance of random gradient selection hyper-heuristics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 2376–2383.

Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2019. On the time complexity of algorithm selection hyper-heuristics for multimodal optimisation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 2322–2329.

Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2020b. Simple hyper-heuristics control the neighbourhood size of randomised local search optimally for LeadingOnes. *Evolutionary Computation* 28, 3 (2020), 437–461.

Andrea Mambrini and Dirk Sudholt. 2015. Design and analysis of schemes for adapting migration intervals in parallel evolutionary algorithms. *Evolutionary Computation* 23, 4 (2015), 559–582.

Frank Neumann, Dirk Sudholt, and Carsten Witt. 2009. Analysis of different MMAS ACO algorithms on unimodal functions and plateaus. *Swarm Intelligence* 3, 1 (2009), 35–68.

Frank Neumann and Carsten Witt. 2010. *Bioinspired Computation in Combinatorial Optimization*. Springer.

Eduardo Carvalho Pinto and Carola Doerr. 2018. Towards a More Practice-Aware Runtime Analysis of Evolutionary Algorithms. (2018). arXiv:1812.00493

Yasha Pushak and Holger Hoos. 2018. Algorithm configuration landscapes: More benign than expected?. In *Parallel Problem Solving from Nature – PPSN XV*. Springer, 271–283.

Yasha Pushak and Holger H. Hoos. 2020. Golden parameter search: Exploiting structure to quickly configure parameters in parallel. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'20)*. ACM, 245–253.

Amirhossein Rajabi and Carsten Witt. 2020a. Evolutionary algorithms with self-adjusting asymmetric mutation. In *Parallel Problem Solving from Nature – PPSN XVI*. Springer, 664–677.

Amirhossein Rajabi and Carsten Witt. 2020b. Self-adjusting evolutionary algorithms for multimodal optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'20)*. ACM, 1314–1322.

Amirhossein Rajabi and Carsten Witt. 2021a. Stagnation detection in highly multimodal fitness landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'21)*. ACM, 1178–1186.

Amirhossein Rajabi and Carsten Witt. 2021b. Stagnation detection with randomized local search. In *Evolutionary Computation in Combinatorial Optimization*. Springer, 152–168. Full version available at http://arxiv.org/abs/2101.12054.

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

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

Dirk Sudholt and Carsten Witt. 2010. Runtime analysis of a binary particle swarm optimizer. *Theoretical Computer Science* 411, 21 (2010), 2084–2100.

Ingo Wegener. 2002. Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In *Evolutionary Optimization*. Kluwer, 349–369.

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

Carsten Witt. 2006. Runtime analysis of the $(\mu + 1)$ EA on simple pseudo-Boolean functions. *Evolutionary Computation* 14, 1 (2006), 65–86.

Carsten Witt. 2014. Fitness levels with tail bounds for the analysis of randomized search heuristics. *Inform. Process. Lett.* 114, 1–2 (2014), 38–41.

Received 11 October 2021; revised 5 May 2022; accepted 17 August 2022