Benchmarking a

$(\mu + \lambda)$

Genetic Algorithm with Configurable Crossover Probability

Furong Ye, Hao Wang, Carola Doerr, Thomas Back

Benchmarking a $(\mu + \lambda)$ Genetic Algorithm with Configurable Crossover Probability

Furong Ye[1], Hao Wang[2], Carola Doerr[2], and Thomas Bäck[1]

[1] LIACS, Leiden University, Leiden, The Netherlands
{f.ye,t.h.w.baeck}@liacs.leidenuniv.nl
[2] Sorbonne Université, CNRS, LIP6, Paris, France
{hao.wang,carola.doerr}@lip6.fr

Abstract. We investigate a family of $(\mu+\lambda)$ Genetic Algorithms (GAs) which creates offspring either from mutation or by recombining two randomly chosen parents. By scaling the crossover probability, we can thus interpolate from a fully mutation-only algorithm towards a fully crossover-based GA. We analyze, by empirical means, how the performance depends on the interplay of population size and the crossover probability.

Our comparison on 25 pseudo-Boolean optimization problems reveals an advantage of crossover-based configurations on several easy optimization tasks, whereas the picture for more complex optimization problems is rather mixed. Moreover, we observe that the "fast" mutation scheme with its are power-law distributed mutation strengths outperforms standard bit mutation on complex optimization tasks when it is combined with crossover, but performs worse in the absence of crossover.

We then take a closer look at the surprisingly good performance of the crossover-based $(\mu + \lambda)$ GAs on the well-known LeadingOnes benchmark problem. We observe that the optimal crossover probability increases with increasing population size μ. At the same time, it decreases with increasing problem dimension, indicating that the advantages of the crossover are not visible in the asymptotic view classically applied in runtime analysis. We therefore argue that a mathematical investigation for fixed dimensions might help us observe effects which are not visible when focusing exclusively on asymptotic performance bounds.

Keywords: Genetic Algorithms · Crossover · Fast Mutation

1 Introduction

Classic evolutionary computation methods build on two main variation operators: *mutation* and *crossover*. While the former can be mathematically defined as unary operators (i.e., families of probability distributions that depend on a single argument), crossover operators sample from distributions of higher arity, with the goal to "recombine" information from two or more arguments.

There is a long debate in evolutionary computation about the (dis-)advantages of these operators, and about how they interplay with each

other [32,37]. In lack of generally accepted recommendations, the use of these operators still remains a rather subjective decision, which in practice is mostly driven by users' experience. Little guidance is available on which operator(s) to use for which situation, and how to most efficiently interleave them. The question how crossover can be useful can therefore be seen as far from being solved.

Of course, significant research efforts are spent to shed light on this question, which is one of the most fundamental ones that evolutionary computation has to offer. While in the early years of evolutionary computation (see, for example, the classic works [11,22,2]) crossover seems to have been widely accepted as an integral part of an evolutionary algorithm, we observe today two diverging trends. Local search algorithms such as GSAT [36] for solving Boolean satisfiability problems, or such as the general-purpose Simulated Annealing [27] heuristic, are clearly very popular optimization methods in practice – both in academic and in industrial applications. These purely mutation-based heuristics are nowadays more commonly studied under the term *stochastic local search*, which forms a very active area of research. Opposed to this is a trend to reduce the use of mutation operators, and to fully base the iterative optimization procedure on recombination operators; see [41] and references therein. However, despite the different recommendations, these opposing positions find their roots in the same problem: we hardly know how to successfully dovetail mutation and crossover.

In addition to large bodies of empirical works aiming to identify useful combinations of crossover and mutation [11,33,23,21],

the question how (or whether) crossover can be beneficial has also always been one of the most prominent problems in *runtime analysis*, the research stream aiming at studying evolutionary algorithms by mathematical means [42,28,25,30,26,15,38,14,13,40,7,8,35,44,34,10] , most of these results focus on very particular algorithms or problems, and are not (or at least not easily) generalizable to more complex optimization tasks.

Our Results In this work, we study a simple variant of the $(\mu + \lambda)$ GA which allows us to conveniently scale the relevance of crossover and mutation, respectively, via a single parameter. More precisely, our algorithm is parameterized by a crossover probability p_c, which is the probability that we generate in the reproduction step an offspring by means of crossover. The offspring is generated by mutation otherwise, so that $p_c = 0$ corresponds to the mutation-only $(\mu+\lambda)$ EA, whereas for $p_c = 1$ the algorithm is entirely based on crossover. Note here that we *either* use crossover *or* mutation, so as to better separate the influence of the two operators.

We first study the performance of different configurations of the $(\mu + \lambda)$ GA on 25 pseudo-Boolean problems (the 23 functions suggested in [19], a concatenated trap problem, and random NK landscape instances). We observe that the algorithms using crossover perform significantly better on some simple functions as OneMax (F1) and LeadingOnes (F2), but also on some problems that are considered hard, e.g., the 1-D Ising model (F19).

We then look more closely into the performance of the algorithm on a benchmark problem intensively studied in runtime analysis: LeadingOnes, the prob-

lem of maximizing the function $f : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\}$. We observe some very interesting effects, that we believe may motivate the theory community to look at the question of usefulness of crossover from a different angle. More precisely, we find that, against our intuition that uniform crossover cannot be beneficial on LeadingOnes, the performance of the $(\mu+\lambda)$ GA on LeadingOnes improves when p_c takes values greater than 0 (and smaller than 1), see Fig. 3. The performances are quite consistent, and we can observe clear patterns, such as a tendency for the optimal value of p_c (displayed in Tab. 2) to increase with increasing μ, and to decrease with increasing problem dimension. The latter effect may explain why it is so difficult to observe benefits of crossover in theoretical work: they disappear with the asymptotic view that is generally adopted in runtime analysis.

We have also performed similar experiments on OneMax (see our project data [45]), but the good performance of the $(\mu + \lambda)$ GA configurations using crossover is less surprising for this problem, since this benefit has previously been observed for genetic algorithms that are very similar to the $(\mu + \lambda)$ GA; see [40,35,7,8] for examples and further references. In contrast to a large body of literature on the benefit of crossover for solving OneMax, we are not aware of the existence of such results for LeadingOnes, apart from the highly problem-specific algorithms developed and analyzed in [1,17].

We hope to promote with this work (1) runtime analysis for fixed dimensions, (2) an investigation of the advantages of crossover on LeadingOnes, and (3) the $(\mu + \lambda)$ GA as a simplified model to study the interplay between problem dimension, population sizes, crossover probability, and mutation rates.

2 Algorithms and Benchmarks

We describe in this section our $(\mu + \lambda)$ GA framework (Sec. 2.1) and the benchmark problems (Sec. 2.2). Since in this paper we can only provide a glimpse on our rich data sets, we also summarize in Sec. 2.3 which data the interested reader can find in our repository [45].

2.1 A Family of $(\mu + \lambda)$ Genetic Algorithms

Our main objective is to study the usefulness of crossover for different kinds of problems. To this end, we investigate a meta-model, which allows us to easily transition from a mutation-only to a crossover-only algorithm. Alg. 1 presents this framework, which, for ease of notation, we refer to as the family of the $(\mu + \lambda)$ GA in the following.

The $(\mu + \lambda)$ GA initializes its population uniformly at random (u.a.r., lines 1-2). In each iteration, it creates λ offspring (lines 6–16). For each offspring, we first decide whether to apply crossover (with probability p_c, lines 8–11) or whether to apply mutation (otherwise, lines 12–15). Offspring that differ from their parents are evaluated, whereas offspring identical to one of their parents inherit this fitness value without function evaluation (see [5] for a discussion).

Algorithm 1: A Family of $(\mu + \lambda)$ Genetic Algorithms

```
1  Input: Population sizes μ, λ, crossover probability p_c, mutation rate p;
2  Initialization: for i = 1, ..., μ do sample x^(i) ∈ {0,1}^n uniformly at random
     (u.a.r.), and evaluate f(x^(i));
3  Set P = {x^(1), x^(2), ..., x^(μ)} ;
4  Optimization: for t = 1, 2, 3, ... do
5      P' ← ∅;
6      for i = 1, ..., λ do
7          Sample r ∈ [0, 1] u.a.r. ;
8          if r ≤ p_c then
9              select two individuals x, y from P u.a.r. (w/ replacement);
10             z^(i) ← Crossover(x, y);
11             if z^(i) ∉ {x, y} then evaluate f(z^(i)) else infer f(z^(i)) from parent;
12         else
13             select an individual x from P u.a.r.;
14             z^(i) ← Mutation(x);
15             if z^(i) ≠ x then evaluate f(z^(i)) else infer f(z^(i)) from parent;
16         P' ← P' ∪ {z^(i)};
17     P is updated by the best μ points in P ∪ P' (ties broken u.a.r.);
```

The best μ of parent and offspring individuals form the new parent population of the next generation (line 17).

Note the unconventional use of *either* crossover *or* mutation. As mentioned, we consider this variant to allow for a better attribution of the effects to each of the operators. Moreover, note that in Alg. 1 we decide for each offspring individually which operator to apply. We call this scheme the $(\mu + \lambda)$ **GA with offspring-based variator choice**. We also study the performance of the $(\mu + \lambda)$ **GA with population-based variator choice**, which is the algorithm that we obtain from Alg. 1 by swapping lines 7 and 6.

We study three different crossover operators, *one-point crossover*, *two-point crossover*, and *uniform crossover*, and two different mutation operators, *standard bit mutation* and the *fast mutation* scheme suggested in [16]. These variation operators are briefly described as follows.

- *One-point crossover*: a crossover point is chosen from $[1..n]$ u.a.r. and an offspring is created by copying the bits from one parent until the crossover point and then copying from the other parent for the remaining positions.
- *Two-point crossover*: similarly, two different crossover points are chosen u.a.r. and the copy process alternates between two parents at each crossover point.
- *Uniform crossover* creates an offspring by copying for each position from the first or from the second parent, chosen independently and u.a.r.
- *Standard bit mutation:* a mutation strength ℓ is sampled from the conditional binomial distribution $\mathrm{Bin}_{>0}(n, p_m)$, which assigns to each k a probability of $\binom{n}{k} p^k (1-p)^{n-k} / (1-(1-p)^n)$ [5]. Thereafter, ℓ distinct positions are chosen

u.a.r. and the offspring is created by first copying the parent and then flipping the bits in these ℓ positions. In this work, we restrict our experiments to the standard mutation rate $p = 1/n$. Note, though, that this choice is not necessarily optimal, as in particular the results in [3,40] and follow-up works demonstrate.
- *Fast mutation* [16]: operates similarly to standard bit mutation except that the mutation strength ℓ is drawn from a power-law distribution: $\Pr[L = \ell] = (C^\beta_{n/2})^{-1}\ell^{-\beta}$ with $\beta = 1.5$ and $C^\beta_{n/2} = \sum_{i=1}^{n/2} i^{-\beta}$.

2.2 The IOHprofiler Problem Set

To test different configurations of the $(\mu + \lambda)$ GA, we first perform an extensive benchmarking on the problems suggested in [19], which are available in the IOHprofiler benchmarking environment [18]. This set contains 23 real-valued pseudo-Boolean test problems: **F1 and F4-F10:** OneMax (F1) and W-model extensions (F4-10), **F2 and F11-F17:** LeadingOnes (F2) and W-model extensions (F11-17), **F3:** Linear function $f(\mathbf{x}) = \sum_{i=1}^{n} ix_i$, **F18:** Low Autocorrelation Binary Sequences (LABS), **F19-21:** Ising Models, **F22:** Maximum Independent Vertex Set (MIVS), and **F23:** N-Queens (NQP).

We recall that the W-model, originally suggested in [43] and extended in [19], is a collection of perturbations that can be applied to a base problem in order to calibrate its features, such as its neutrality, its epistasis, and its ruggedness. We add to the list of [19] the following two problems:
F24: Concatenated Trap (CT) is defined by partitioning a bit-string into segments of length k and concatenating $m = n/k$ trap functions that takes each segment as input. The trap function is defined as follows: $f_k^{\text{trap}}(u) = 1$ if the number u of ones satisfies $u = k$ and $f_k^{\text{trap}}(u) = \frac{k-1-u}{k}$ otherwise. We use $k = 5$ in our experiments.
F25: Random NK landscapes (NKL). The function values are defined as the average of n sub-functions $F_i \colon [0..2^{k+1} - 1] \to \mathbb{R}, i \in [1..n]$, where each component F_i only takes as input a set of $k \in [0..n-1]$ bits that are specified by a neighborhood matrix. In this paper, k is set to 1 and entries of the neighbourhood matrix are drawn u.a.r. in $[1..n]$. The function values of F_i's are sampled independently from a uniform distribution on $(0, 1)$.

Note that the IOHprofiler problem set provides for each problem several problem instances, which all have isomorphic fitness landscapes, but different problem representations. In our experiments we only use the first instance of each problem (seed 1). For the mutation-based algorithms and the ones using uniform crossover, the obtained results generalize to all other problem instances. For algorithms involving one- or two-point crossover, however, this is not the case, as these algorithms are not unbiased (in the sense of Lehre and Witt [29]).

2.3 Data Availability

Detailed results for the different configurations of the $(\mu+\lambda)$ GA are available in our data repository at [45]. In particular, we host there data for the IOHprofiler

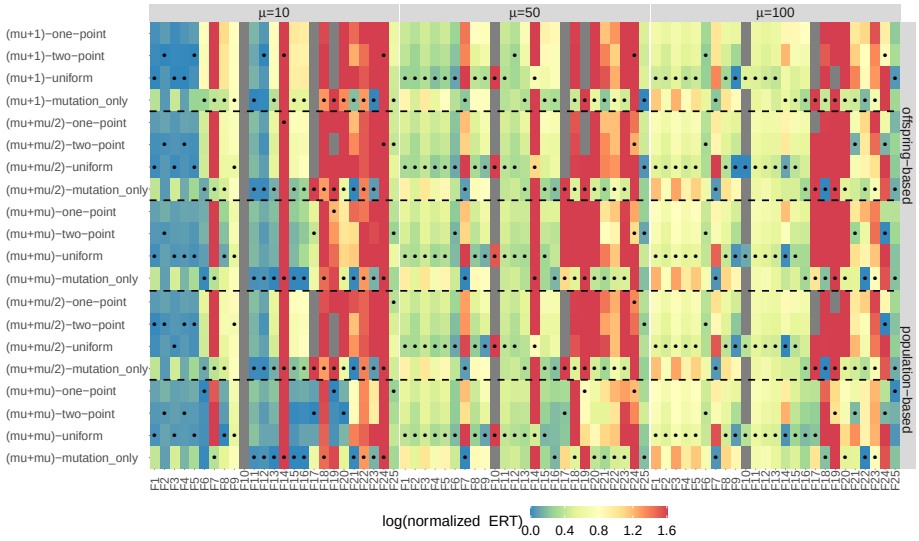

Fig. 1. Heat map of normalized ERT values of the $(\mu + \lambda)$ GA with offspring-based (top part) and population-based (bottom part) variator choice for the 100-dimensional benchmark problems, computed based on the target values specified in Table 1. The crossover probability p_c is set to 0.5 for all algorithms except the mutation-only ones (which use $p_c = 0$). The displayed values are the the quotient of the ERT and ERT_{best}, the ERT achieved by the best of all displayed algorithms. These quotients are capped at 40 to increase interpretability of the color gradient in the most interesting region. The three algorithm groups – the $(\mu + 1)$, the $(\mu + \lceil \mu/2 \rceil)$, and the $(\mu + \mu)$ GAs – are separated by dashed lines. A dot indicates the best algorithm of each group of four. A grey tile indicates that the $(\mu + \lambda)$ GA configuration failed, in all runs, to find the target value within the given budget.

experiments (36 algorithms, 25 functions, 5 dimensions ≤ 250, 100 independent runs) and for the $(\mu + \lambda)$ GA on OneMax and on LeadingOnes for all of the following 5544 parameter combinations: $n \in \{64, 100, 150, 200, 250, 500\}$ (6 values), $\mu \in \{2, 3, 5, 8, 10, 20, 30, ..., 100\}$ (14 values), $\lambda \in \{1, \lceil \mu/2 \rceil, \mu\}$ (3 values), $p_c \in \{0.1k \mid k \in [0..9]\} \cup \{0.95\}$ (11 values), two mutation operators (standard bit mutation and fast mutation). In these experiments on OneMax and LeadingOnes, the crossover operator is fixed to uniform crossover.

A detailed analysis of these results, for example using IOHprofiler or using HiPlot [9] may give additional insights into the dependence of the overall performance on the parameter setting

3 Results for the IOHprofiler Problems

In order to probe into the empirical performance of the $(\mu+\lambda)$ GA, we test it on the 25 problems mentioned in Sec. 2.2, with a total budget of $100n^2$ function evaluations. We perform 100 independent runs of each algorithm on each problem.

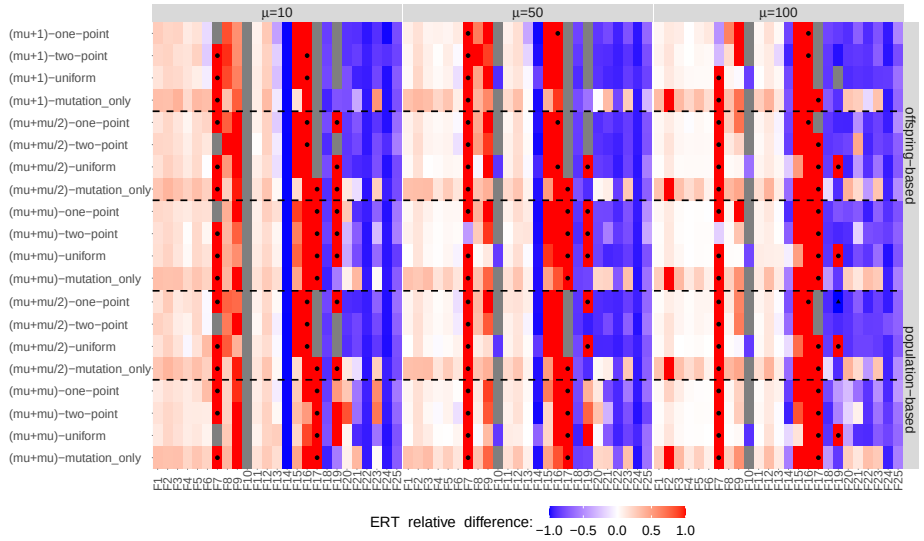

Fig. 2. Heat map comparing standard bit mutation (sbm) with fast mutation on the 25 problems from Sec. 2.2 in dimensions $n = 100$. Plotted values are $(\mathrm{ERT}_{\mathrm{fast}} - \mathrm{ERT}_{\mathrm{sbm}})/\mathrm{ERT}_{\mathrm{sbm}}$, for ERTs computed wrt the target values specified in Table 1. p_c is set to 0.5 for all crossover-based algorithms. Values are bounded in $[-1, 1]$ to increase visibility of the color gradient in the most interesting region. A black dot indicates that the $(\mu + \lambda)$ GA with fast mutation failed in all runs to find the target with the given budget; the black triangle signals failure of standard bit mutation, and a gray tile is chosen for settings in which the $(\mu + \lambda)$ GA failed for both mutation operators.

$F1$	$F2$	$F3$	$F4$	$F5$	$F6$	$F7$	$F8$	$F9$	$F10$	$F11$	$F12$	$F13$	$F14$
100	100	5050	50	90	33	100	51	100	100	50	90	33	7

$F15$	$F16$	$F17$	$F18$	$F19$	$F20$	$F21$	$F22$	$F23$	$F24$	$F25$
51	100	100	4.215852	98	180	260	42	9	17.196	-0.2965711

Table 1. Target values used for computing the ERT value in Fig. 1.

A variety of parameter settings are investigated: (1) all three crossover operators described in Sec. 2 (we use $p_c = 0.5$ for all crossover-based configurations), (2) both mutation variator choices, (3) $\mu \in \{10, 50, 100\}$, and (4) $\lambda \in \{1, \lceil \mu/2 \rceil, \mu\}$.

In Fig. 1, we highlight a few basic results of this experimentation for $n = 100$, where the mutation operator is fixed to the standard bit mutation. More precisely, we plot in this figure the normalized expected running time (ERT), where the normalization is with respect to the best ERT achieved by any of the algorithms for the same problem. Table 1 provides the target values for which we computed the ERT values. For each problem and each algorithm, we first calculated the 2% percentile of the best function values. We then selected the largest of these percentiles (over all algorithms) as target value.

On the OneMax-based problems F1, F4, and F5, the $(\mu+\lambda)$ GA outperforms the mutation-only GA, regardless of the variator choice scheme, the crossover operator, and the setting of λ. When looking at problem F6, we find out that

when $\mu = 10$ the mutation-only GA surpasses most of $(\mu + \lambda)$ GA variants except the population-based $(\mu + \mu)$ GA with one-point crossover. On F8-10, the $(\mu + \lambda)$ GA takes the lead in general, whereas it cannot rival the mutation-only GA on F7. Also, only the configuration with uniform crossover can hit the optimum of F10 within the given budget.

On the linear function F3 we observe a similar behavior as on OneMax. On LeadingOnes (F2), the $(\mu + \lambda)$ GA outperforms the mutation-only GA again for $\mu \in \{50, 100\}$ while for $\mu = 10$ the mutation-only GA becomes superior with one-point and uniform crossovers. On F11-13 and F15-16 (the W-model extensions of LeadingOnes), the mutation-only GA shows a better performance than the $(\mu + \lambda)$ GA with one-point and uniform crossovers and this advantage becomes more significant when $\mu = 10$. On problem F14, that is created from LeadingOnes using the same transformation as in F7, the mutation-only GA is inferior to the $(\mu + \lambda)$ GA with uniform crossover.

On problems F18 and F23, the mutation-only GA outperforms the $(\mu+\lambda)$ GA for most parameter settings. On F21, the $(\mu + \lambda)$ GA with two-point crossover yields a better result when the population size is larger (i.e., $\mu = 100$) while the mutation-only GA takes the lead for $\mu = 10$. On problems F19 and F20, the $(\mu + \mu)$ GA with the population-based variator choice significantly outperforms all other algorithms, whereas it is substantially worse for the other parameter settings. On problem F24, the $(\mu + \mu/2)$ GA with two-point crossover achieves the best ERT value when $\mu = 100$. None of the tested algorithms manages to solve F24 with the given budget. The target value used in Fig. 1 is 17.196, which is below the optimum 20. On problem F25, the mutation-only GA and the $(\mu + \lambda)$ GA are fairly comparable when $\mu \in \{10, 50\}$. Also, we observe that the population-based $(\mu+\mu)$ GA outperforms the mutation-only GA when $\mu = 100$.

In general, we have made the following observations: (1) on problems F1-6, F8-9, and F11-13, all algorithms obtain better ERT values with $\mu = 10$. On problems F7, F14, and F21-25, the $(\mu + \lambda)$ GA benefits from larger population sizes, i.e., $\mu = 100$; (2) The $(\mu+\mu)$ GA with uniform crossover and the mutation-only GA outperform the $(\mu+\lceil\mu/2\rceil)$ GA across all three settings of μ on most of the problems, except F10, F14, F18, and F22. For the population-based variator choice scheme, increasing λ from one to μ improves the performance remarkably on problems F17-24. Such an improvement becomes negligible for the offspring-based scheme; (3) Among all three crossover operators, the uniform crossover often surpasses the other two on OneMax, LeadingOnes, and the W-model extensions thereof.

To investigate the impact of mutation operators on GA, we plot in Fig. 2 the relative ERT difference between the $(\mu + \lambda)$ GA configurations using fast and standard bit mutation, respectively. As expected, fast mutation performs slightly worse on F1-6, F8, and F11-13. On problems F7, F9, and F15-17, however, fast mutation becomes detrimental to the ERT value for most parameter settings. On problems F10, F14, F18, and F21-25, fast mutation outperforms standard bit mutation, suggesting a potential benefit of pairing the fast mutation with crossover operators to solve more difficult problems. Interestingly, with an in-

creasing μ, the relative ERT of the $(\mu + \lambda)$ GA quickly shrinks to zero, most notably on F1-6, F8, F9, F11-13.

Interestingly, in [31], an empirical study has shown that on a randomly generated maximum flow test generation problem, fast mutation is significantly outperformed by standard bit mutation when combined with uniform crossover. Such an observation seems contrary to our findings on F10, F14, F18, and F21-25. However, it is made on a standard $(100 + 70)$ GA in which both crossover and mutation are applied to the parent in order to generate offspring. We are planning to investigate the effects of this inter-chaining in future work, but this topic is beyond the focus of this study.

4 Case-Study: LeadingOnes

The surprisingly good performance of the $(\mu + \lambda)$ GA with $p_c = 0.5$ on LeadingOnes motivates us to investigate this setting in more detail. Before we go into the details of the experimental setup and our results, we recall that for the optimization of LeadingOnes, the fitness values only depend on the first bits, whereas the tail is randomly distributed and has no influence on the selection. More precisely, a search point x with LeadingOnes-value $f(x)$ has the following structure: the first $f(x)$ bits are all 1, the $f(x) + 1$st bit equals 0, and the entries in the tail (i.e., in positions $[f(x) + 2..n]$) did not have any influence on the optimization process so far. For many algorithms, it can be shown that these tail bits are uniformly distributed, see [12] for an extended discussion.

Experimental setup. We fix in this section the variator choice to the *offspring-based setting*. We do so because its performance was seen to be slightly better on LeadingOnes than the population-based choice. We experiment with the parameter settings specified in Sec. 2.3. For each of the settings listed there, we perform 100 independent runs, with a maximal budget of $5n^2$ each.

Overall Running Time. We first investigate the impact of the crossover probability on the average running time, i.e., on the average number of function evaluations that the algorithm performs until it evaluates the optimal solution for the first time. The results for the $(\mu + 1)$ and the $(\mu + \mu)$ GA using uniform crossover and standard bit mutation are summarized in Fig. 3. Since not all algorithms managed to find the optimum within the given time budget, we plot as red bars the ERT values for such algorithms with success ratio strictly smaller than 1, whereas the black bars are reserved for algorithms with 100 successful runs. All values are normalized by n^2, to allow for a better comparison. All patterns described below also apply to the $(\mu + \lceil \mu/2 \rceil)$ GA, whose results we do not display for reasons of space. They are also very similar when we replace the mutation operator by the fast mutation scheme suggested in [16].

As a first observation, we note that the pattern of the results are quite regular. As can be expected, the dispersion of the running times is rather small. For reasons of space, we do not describe this dispersion in detail, but to give an impression for the concentration of the running times, we report that the standard deviation of the $(50 + 1)$ GA on the 100-dimensional LeadingOnes

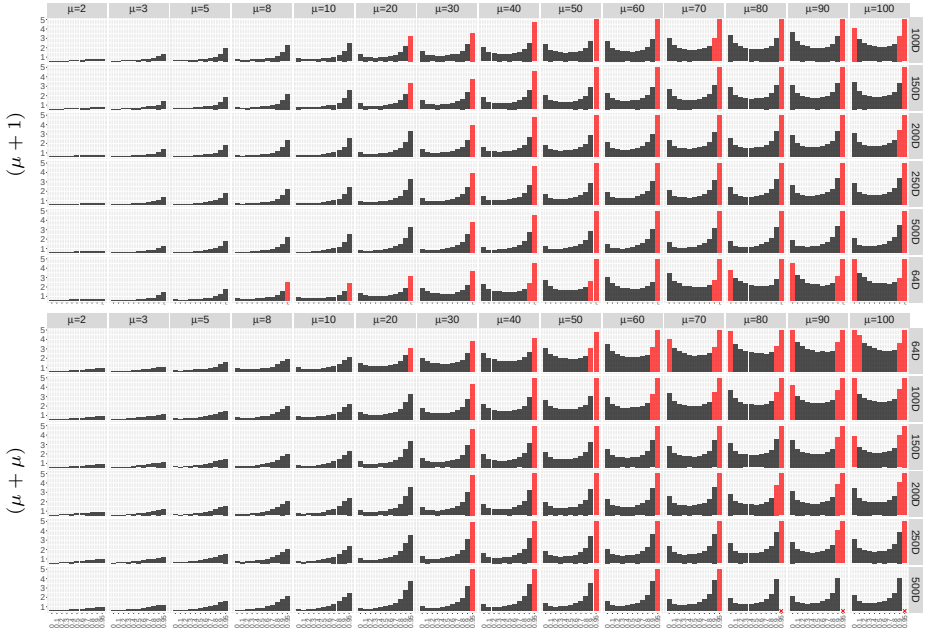

Fig. 3. By n^2 normalized ERT values for the $(\mu + \lambda)$ GA using standard bit mutation and uniform crossover on LeadingOnes, for different values of μ and for $\lambda = 1$ (top) and for $\lambda = \mu$ (bottom). Results are grouped by the value of μ (main columns), by the crossover probability p_c (minor columns), and by the dimension (rows). The ERTs are computed from 100 independent runs for each setting, with a maximal budget of $5n^2$ fitness evaluations. ERTs for algorithms which successfully find the optimum in all 100 runs are depicted as black bars, whereas ERTs for algorithms with success rates in $(0, 1)$ are depicted as red bars. All bars are capped at 5.

function is approximately 14% of the average running time across all values of p_c. As can be expected for a genetic algorithm on LeadingOnes, the average running increases with increasing population size μ, see [39] for a proof of this statement when $p_c = 0$.

Next, we compare the sub-plots in each row, i.e., fixing the dimension. We see that the $(\mu+\lambda)$ GA suffers drastically from large p_c values when μ is smaller, suggesting that the crossover operator hinders performance. But as μ gets larger, the average running time at moderate crossover probabilities (p_c around 0.5) is significantly smaller than that in two extreme cases, $p_c = 0$ (mutation-only GAs), and $p_c = 0.95$. This observation holds for all dimensions and for both algorithm families, the $(\mu + 1)$ and the $(\mu + \mu)$ GA.

Looking at the sub-plots in each column (i.e., fixing the population size), we identify another trend: for those values of μ for which an advantage of $p_c > 0$ is visible for the smallest tested dimension, $n = 64$, the relative advantage of this rate decreases and eventually disappears as the dimension increases.

	n \ μ	2	3	5	8	10	20	30	40	50	60	70	80	90	100
$(\mu+1)$	64	0.0	0.1	0.1	0.1	0.2	0.3	0.5	0.4	0.5	0.5	0.6	0.7	0.6	0.7
	100	0.0	0.1	0.1	0.1	0.1	0.3	0.4	0.4	0.4	0.5	0.5	0.5	0.4	0.6
	150	0.0	0.1	0.1	0.1	0.1	0.2	0.3	0.3	0.4	0.4	0.5	0.4	0.4	0.5
	200	0.0	0.0	0.1	0.1	0.1	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4
	250	0.0	0.0	0.1	0.1	0.1	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.3	0.4
	500	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.2	0.2	0.3	0.3	0.3	0.3	0.3
$(\mu+\mu)$	64	0.0	0.2	0.1	0.1	0.2	0.2	0.4	0.4	0.6	0.5	0.5	0.7	0.5	0.7
	100	0.0	0.0	0.1	0.1	0.2	0.3	0.3	0.3	0.5	0.4	0.5	0.5	0.6	0.5
	150	0.0	0.0	0.1	0.1	0.2	0.2	0.3	0.3	0.5	0.4	0.5	0.5	0.5	0.5
	200	0.0	0.0	0.1	0.1	0.1	0.1	0.3	0.3	0.3	0.3	0.4	0.5	0.4	0.5
	250	0.0	0.0	0.1	0.1	0.1	0.2	0.3	0.3	0.3	0.3	0.3	0.3	0.4	0.4
	500	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.2	0.2	0.3	0.3	0.3	0.3	0.3

Table 2. On LeadingOnes, the optimal value of p_c for the $(\mu + 1)$ and the $(\mu + \mu)$ GA with uniform crossover and standard bit mutation, for various combinations of dimension n (rows) and μ (columns). Values are approximated from 100 independent runs each, probing $p_c \in \{0.1k \mid k \in [0..9]\} \cup \{0.95\}$.

Finally, we compare the results of the $(\mu+1)$ GA with those of the $(\mu+\mu)$ GA. Following [24], it is not surprising that for $p_c = 0$, the results of the $(\mu + 1)$ GA are better than those of the $(\mu+\mu)$ GA (very few exceptions to this rule exist in our data, but in all these cases the differences in average runtime are negligibly small), and following our own theoretical analysis [20, Theorem 1], it is not surprising that the differences between these two algorithmic families are rather small: the typical disadvantage of the $(\mu + \lceil \mu/2 \rceil)$ GA over the $(\mu + 1)$ GA is around 5% and it is around 10% for the $(\mu + \mu)$ GA, but these relative values differ between the different configurations and dimensions.

Optimal Crossover Probabilities. To make our observations on the crossover probability clearer, we present in Table 2 a heatmap of the values p_c^* for which we observed the best average running time (with respect to all tested p_c values). We see the same trends here as mentioned above: as μ increases, the value of p_c^* increases, while, for fixed μ its value decreases with increasing problem dimension n. Here again we omit details for the $(\mu + \lceil \mu/2 \rceil)$ GA and for the fast mutation scheme, but the patterns are identical, with very similar absolute values.

Fixed-Target Running Times. We now study where the advantage of the crossover-based algorithms stems from. We demonstrate this using the example of the $(50 + 50)$ GA in 200 dimensions. We recall from Table 2 that the optimal crossover probability for this setting is $p_c^* = 0.3$. The left plot in Fig. 4 is a fixed-target plot, in which we display for each tested crossover probability p_c (different lines) and each fitness value $i \in [0..200]$ (x-axis) the average time needed until the respective algorithm evaluates for the first time a search point of fitness at least i. The mutation-only configuration ($p_c = 0$) performs on par with the best configurations for the first part of the optimization process, but then loses in performance as the optimization progresses. The plot on the right shows the gradients of the fixed-target curves. The gradient can be used to analyze which configuration performs best at a given target value. We observe an interesting behavior here, namely that the gradient of the configuration $p_c = 0.8$, which has

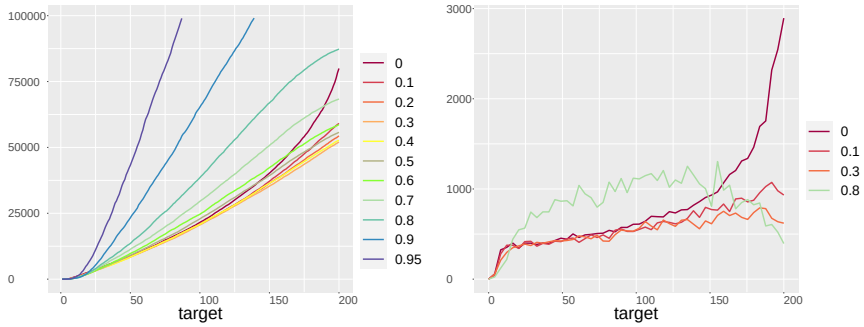

Fig. 4. Left: Average fixed-target running times of the $(50 + 50)$ GA with uniform crossover and standard bit mutation on LeadingOnes in 200 dimensions, for different crossover probabilities p_c. Results are averages of 100 independent runs. **Right:** Gradient of selected fixed-target curves.

a very bad fixed-target performance on all targets (left plot), is among the best in the final parts of the optimization. The plot on the right therefore suggests that an adaptive choice of p_c should be investigated further.

5 Conclusions

In this paper, we have analyzed the performance of a family of $(\mu + \lambda)$ GAs, in which offspring are either generated by crossover (with probability p_c) or by mutation (probability $1 - p_c$). On the IOHprofiler problem set, it has been shown that this random choice mechanism reduces the expecting running time on OneMax, LeadingOnes, and many W-model extensions of those two problems. By varying the value of the crossover probability p_c, we discovered on LeadingOnes that its optimal value p_c^* (with respect to the average running time) increases with the population size μ, whereas for fixed μ it decreases with increasing dimension n.

Our results raise the interesting question of whether a non-asymptotic runtime analysis (i.e., bounds that hold for a fixed dimension rather than in big-Oh notation) could shed new light on our understanding of evolutionary algorithms. We note that a few examples of such analyses can already be found in the literature, e.g., in [6,4]. The regular patterns observed in Fig. 3 suggest the presence of trends that could be turned into formal knowledge.

It would certainly also be interesting to extend our study to a $(\mu + \lambda)$ GA variant using dynamic values for the relevant parameters μ, λ, crossover probability p_c, and mutation rate p. We are also planning to extend the study to more conventional $(\mu + \lambda)$ GA, which apply mutation right after crossover.

Acknowledgments. Our work was supported by the Chinese scholarship council (CSC No. 201706310143), by the Paris Ile-de-France Region, and by COST Action CA15140.

References

1. Afshani, P., Agrawal, M., Doerr, B., Doerr, C., Larsen, K.G., Mehlhorn, K.: The query complexity of finding a hidden permutation. In: Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday. LNCS, vol. 8066, pp. 1–11. Springer (2013)
2. Bäck, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Inc., USA (1996)
3. Böttcher, S., Doerr, B., Neumann, F.: Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In: Proc. of Parallel Problem Solving from Nature (PPSN'10). Lecture Notes in Computer Science, vol. 6238, pp. 1–10. Springer (2010)
4. Buskulic, N., Doerr, C.: Maximizing drift is not optimal for solving onemax. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'19). pp. 425–426. ACM (2019). Full version available at http://arxiv.org/abs/1904.07818
5. Carvalho Pinto, E., Doerr, C.: A simple proof for the usefulness of crossover in black-box optimization. In: Proc. of Parallel Problem Solving from Nature (PPSN'18). LNCS, vol. 11102, pp. 29–41. Springer (2018).
6. Chicano, F., Sutton, A.M., Whitley, L.D., Alba, E.: Fitness probability distribution of bit-flip mutation. Evolutionary Computation **23**(2), 217–248 (2015).
7. Corus, D., Oliveto, P.S.: Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. IEEE Transactions on Evolutionary Computation **22**(5), 720–732 (2018).
8. Corus, D., Oliveto, P.S.: On the benefits of populations for the exploitation speed of standard steady-state genetic algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'19). pp. 1452–1460. ACM (2019).
9. D. Haziza, J. Rapin, G.S.: HiPlot - High dimensional Interactive Plotting. https://github.com/facebookresearch/hiplot (2020)
10. Dang, D.C., Friedrich, T., Kötzing, T., Krejca, M.S., Lehre, P.K., Oliveto, P.S., Sudholt, D., Sutton, A.M.: Escaping local optima using crossover with emergent diversity. IEEE Transactions on Evolutionary Computation **22**(3), 484–497 (2017)
11. De Jong, K.A.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph.D. thesis, University of Michigan, Ann Arbor, MI, USA (1975)
12. Doerr, B.: Analyzing randomized search heuristics via stochastic domination. Theoretical Computer Science **773**, 115–137 (2019)
13. Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing new genetic algorithms. Theoretical Computer Science **567**, 87–104 (2015)
14. Doerr, B., Happ, E., Klein, C.: Crossover can provably be useful in evolutionary computation. Theoretical Computer Science **425**, 17–33 (2012)
15. Doerr, B., Johannsen, D., Kötzing, T., Neumann, F., Theile, M.: More effective crossover operators for the all-pairs shortest path problem. Theoretical Computer Science **471**, 12–26 (2013)
16. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'17). pp. 777–784. ACM (2017)
17. Doerr, B., Winzen, C.: Black-box complexity: Breaking the $O(n \log n)$ barrier of LeadingOnes. In: Artificial Evolution (EA'11), Revised Selected Papers. LNCS, vol. 7401, pp. 205–216. Springer (2012)

18. Doerr, C., Wang, H., Ye, F., van Rijn, S., Bäck, T.: IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics. arXiv e-prints:1810.05281 (Oct 2018), https://arxiv.org/abs/1810.05281
19. Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O.M., Bäck, T.: Benchmarking discrete optimization heuristics with IOHprofiler. Applied Soft Computing **88**, 106027 (2019)
20. Doerr, C., Ye, F., van Rijn, S., Wang, H., Bäck, T.: Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: profiling $(1 + \lambda)$ EA variants on onemax and leadingones. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'18). pp. 951–958. ACM (2018).
21. Elsayed, S.M., Sarker, R.A., Essam, D.L.: Multi-operator based evolutionary algorithms for solving constrained optimization problems. Computers & Operations Research **38**(12), 1877 – 1896 (2011).
22. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edn. (1989)
23. Hyun-Sook Yoon, Byung-Ro Moon: An empirical study on the synergy of multiple crossover operators. IEEE Transactions on Evolutionary Computation **6**(2), 212–223 (April 2002).
24. Jansen, T., De Jong, K.A., Wegener, I.: On the choice of the offspring population size in evolutionary algorithms. Evolutionary Computation **13**, 413–440 (2005)
25. Jansen, T., Wegener, I.: The analysis of evolutionary algorithms—a proof that crossover really can help. Algorithmica **34**, 47–66 (2002)
26. Jansen, T., Wegener, I.: Real royal road functionswhere crossover provably is essential. Discrete applied mathematics **149**(1-3), 111–125 (2005)
27. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**, 671–680 (1983)
28. Kötzing, T., Sudholt, D., Theile, M.: How crossover helps in pseudo-boolean optimization. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'11). pp. 989–996. ACM (2011)
29. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**, 623–642 (2012)
30. Lehre, P.K., Yao, X.: Crossover can be constructive when computing unique input–output sequences. Soft Computing **15**(9), 1675–1687 (2011)
31. Mironovich, V., Buzdalov, M.: Evaluation of heavy-tailed mutation operator on maximum flow test generation problem. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'17), Companion Material. pp. 1423–1426. ACM (2017).
32. Mitchell, M., Holland, J.H., Forrest, S.: When will a genetic algorithm outperform hill climbing? In: Proc. of Neural Information Processing Systems Conference (NIPS'93). Advances in Neural Information Processing Systems, vol. 6, pp. 51–58. Morgan Kaufmann (1993)
33. Murata, T., Ishibuchi, H.: Positive and negative combination effects of crossover and mutation operators in sequencing problems. In: Proc. of Conference on Evolutionary Computation. pp. 170–175 (May 1996).
34. Neumann, F., Oliveto, P.S., Rudolph, G., Sudholt, D.: On the effectiveness of crossover for migration in parallel evolutionary algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'11). pp. 1587–1594. ACM (2011).
35. Pinto, E.C., Doerr, C.: A simple proof for the usefulness of crossover in black-box optimization. In: Parallel Problem Solving from Nature - PPSN XV - 15th

International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part II. LNCS, vol. 11102, pp. 29–41. Springer (2018).
36. Selman, B., Levesque, H.J., Mitchell, D.G.: A new method for solving hard satisfiability problems. In: Proc. of National Conference on Artificial Intelligence. pp. 440–446. AAAI (1992)
37. Spears, W.M.: Crossover or mutation? In: Foundations of genetic algorithms, vol. 2, pp. 221–237. Elsevier (1993)
38. Sudholt, D.: Crossover is provably essential for the Ising model on trees. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'05). pp. 1161–1167. ACM Press (2005)
39. Sudholt, D.: A new method for lower bounds on the running time of evolutionary algorithms. IEEE Transactions on Evolutionary Computation **17**, 418–435 (2013)
40. Sudholt, D.: How crossover speeds up building block assembly in genetic algorithms. Evolutionary computation **25**(2), 237–274 (2017)
41. Varadarajan, S., Whitley, D.: The massively parallel *mixing genetic algorithm* for the traveling salesman problem. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'19). pp. 872–879. ACM (2019).
42. Watson, R.A., Jansen, T.: A building-block royal road where crossover is provably essential. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'07). pp. 1452–1459. ACM (2007).
43. Weise, T., Wu, Z.: Difficult features of combinatorial optimization problems and the tunable w-model benchmark problem for simulating them. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'18, Companion Material), pp. 1769–1776. ACM (2018).
44. Whitley, D., Varadarajan, S., Hirsch, R., Mukhopadhyay, A.: Exploration and exploitation without mutation: Solving the jump function in $\theta(n)$ time. In: Proc. of Parallel Problem Solving from Nature (PPSN'18). LNCS, vol. 11102, pp. 55–66. Springer (2018).
45. Ye, F., Wang, H., Doerr, C., Bäck, T.: Experimental Data Sets for the study "Benchmarking a $(\mu + \lambda)$ Genetic Algorithm with Configurable Crossover Probability" (Apr 2020). https://doi.org/10.5281/zenodo.3753086