A Simple Proof for the Usefulness of Crossover in Black-Box Optimization

Eduardo Carvalho Pinto[1] and Carola Doerr[2]

[1] DreamQuark, Paris, France
[2] Sorbonne Université, CNRS, LIP6, Paris, France

Abstract. The idea to recombine two or more search points into a new solution is one of the main design principles of evolutionary computation (EC). Its usefulness in the combinatorial optimization context, however, is subject to a highly controversial discussion between EC practitioners and the broader Computer Science research community. While the former, naturally, report significant speedups procured by crossover, the belief that sexual reproduction cannot advance the search for high-quality solutions seems common, for example, amongst theoretical computer scientists. Examples that help understand the role of crossover in combinatorial optimization are needed to promote an intensified discussion on this subject.

We contribute with this work an example of a crossover-based genetic algorithm (GA) that provably outperforms any mutation-based black-box heuristic on a classic benchmark problem. The appeal of our examples lies in its simplicity: the proof of the result uses standard mathematical techniques and can be taught in a basic algorithms lecture.

Our theoretical result is complemented by an empirical evaluation, which demonstrates that the superiority of the GA holds already for quite small problem instances.

Keywords: Evolutionary Computation · Crossover · Recombination · Runtime Analysis.

1 Introduction

Evolutionary Computation (EC) borrows inspiration from phenomena observed in biological evolution processes. One of the fundamental design principles of EC is *crossover;* i.e., the recombination of two or more candidate solutions into one or several *offspring.* EC practitioners frequently report that crossover (which is also referred to as *sexual reproduction*) brings significant performance gains. This belief, however, is often challenged in the broader Computer Science (CS) community, and in particular in the subarea of Theoretical CS, yielding to very generally formulated statements that crossover cannot be beneficial in combinatorial optimization. As an example we mention a quote by Christos Papadimitriou and colleagues, who formulated the claim that "Simulated annealing tends to work quite well, but genetic algorithms do not".[3]

In light of this discrepancy, it has been one of the main focus question in the theory of EC community to contribute to a better understanding of when and why crossover-based algorithms can perform better than purely mutation-based ones. It seems quite notable that only very few examples exist where such an effect can be rigorously proven.

1.1 Selected Theoretical Results on the Benefits of Crossover

We summarize a few selected results that prove an advantage of crossover in the discrete black-box optimization context, and refer the interested reader to [15] for an extended discussion. The first work observing a benefit of crossover-based GAs over a standard evolutionary algorithm (EA) dates back to [10], where so-called Jump functions are considered, in which algorithms are required to "jump" a gap between local and global optima. As discussed in [2,15], several follow-up works introduced similarly artificial problems to demonstrate an advantage of crossover. The first classical combinatorial example for which recombination

[3] See, for example, here: https://www.simonsfoundation.org/2010/05/18/why-sex/.

could be shown to be beneficial was presented in [7]. In this work, a problem-tailored crossover operation was shown to be advantageous for the all-pairs-shortest-path problem.

These theoretical results, albeit being very appealing, do not answer the question of how beneficial the use of crossover is in standard EAs, or for standard benchmark problems. Starting with the work [13], the quest to prove advantages of crossover for simple hill-climbing tasks has recently taken considerable momentum. Sudholt proved that a greedy $(\mu + 1)$ GA with a diversity mechanism that avoids duplicates needs $(1 + o(1))\frac{e}{2} n \ln n \approx 1.359n \ln n + o(n \ln n)$ function evaluations, on average, to optimize OneMax; the combinatorial optimization problem asking to minimize the Hamming distance to an unknown bit string $z \in \{0, 1\}^n$. This runtime is better by a factor of two than the expected $(1+o(1))en \ln n$ optimization time of any evolutionary algorithm using only standard bit mutation [14, 16]. Sudholt also proved that the expected runtime of the algorithm can be further reduced to approximately $1.19n \ln n + o(n \ln n)$ by increasing the mutation rate from $1/n$ to $(1 + \sqrt{5})/(2n)$.

The results of [13] were generalized to less greedy $(\mu + 1)$ GAs in [15] and to GAs avoiding the diversity mechanism in [2]. All these works show an advantage of crossover-based $(\mu + 1)$ GAs over evolutionary algorithms using standard bit mutation. They do not, however, beat the average performance of another very common randomized optimization heuristic, Randomized Local Search (RLS). The expected optimization time of RLS on OneMax is $n \ln(n/2) + \gamma n \pm o(1)$, with $\gamma \approx 0.5772156649$ being the EulerMascheroni constant [3]. For a more convincing argument in favor of crossover, one would like to have an example for a crossover-based heuristic that outperforms not only mutation-based EAs but also RLS as well as any other so-called *unary unbiased black-box algorithm*. The notion of a unary unbiased black-box algorithm was introduced in [12] as a model for purely mutation-based algorithms. While it was already proven in [12] that any unary unbiased black-box algorithm has an expected optimization time on OneMax of order at least $n \log n$, a precise lower bound, which is $n \ln(n) - cn \pm o(n)$ for a constant c between 0.2539 and 0.2665, could be shown only recently [6]. The expected optimization times proven in [2, 13, 15] are all by a multiplicative factor of at least 1.19 larger than this bound.

In [8] it was shown that *binary* unbiased black-box algorithms exist that achieve a linear expected optimization time on OneMax. While this can be seen as a proof in favor of recombination, the algorithm is highly problem-tailored. A more appealing example rigorously proving an advantage of crossover over any unary unbiased black-box algorithm has been presented in [5]. The $(1 + (\lambda, \lambda))$ GA uses only well-known and widely applied building blocks from the EC literature (standard bit mutation, (biased) uniform crossover, and elitist selection), but recombines them in a new way: by first mutating a best so-far solution through standard bit mutation, the crossover operator becomes a "repair mechanism". For suitable parameter settings, the $(1 + (\lambda, \lambda))$ GA can achieve linear expected optimization time on OneMax [4, 5], and, by the lower bounds of [6, 12], therefore scales much more favorably with the problem dimension than any unary unbiased black-box algorithm.

1.2 Our Results

In this work, we revisit the analysis of the greedy $(\mu + 1)$ GA with diversity mechanism presented in [13]. Following the suggestion made in [1] we take a more implementation-aware perspective on this algorithm, in that we do not charge function evaluations for search points that are identical to one of their direct ancestors. Put differently, we try to avoid creating such offspring, as they do not provide any new information about the problem instance at hand. In the absence of noise, this is how one would implement the $(\mu + \lambda)$ GA for all practical purposes, cf. [1] for a discussion. We note that for the two variation operators employed by the $(\mu + \lambda)$ GA, standard bit mutation and uniform crossover, tracking whether or not an offspring equals one of its parents is very simple and comes at almost no cost.

Quite surprisingly, we show that this simple modification yields performance bounds that are strictly better than the above-mentioned $n \ln(n) - cn \pm o(n)$ lower bound valid for all unary unbiased black-box algorithms. More precisely, we show that, for a suitably chosen mutation rate p, the modified greedy $(\mu+1)$ GA with diversity mechanism achieves an $0.851..n \ln(n) + o(n \log n)$ expected optimization time on OneMax. The proof of this result is surprisingly simple, and can be taught in an undergraduate course.

2 The Greedy $(\mu + 1)$ GA

We present the crossover-based genetic algorithm (GA) for which we will prove in Section 3 that it outperforms any mutation-based algorithm on the Hamming distance problem OneMax. The algorithm is a (mild) modification of an algorithm previously suggested for the study of the effectiveness of crossover: the greedy $(\mu + 1)$ GA presented in [13]. We present the original algorithm in Section 2.1, motivate our modifications in Section 2.2, and describe the modified greedy $(\mu + 1)$ GA in Section 2.3.

2.1 The Original Greedy $(\mu + 1)$ GA

The greedy $(\mu + 1)$ GA proposed by Sudholt in [13] is Algorithm 1 with lines 5 to 8 replaced by "Sample ℓ from $\mathrm{Bin}(n, p)$". It maintains a population $\mathcal{P}$ of μ individuals. $\mathcal{P}$ is initialized by sampling μ search points independently and uniformly at random. Each iteration consists of two steps; a crossover step and a mutation step. In the *crossover step* two parents x and y are selected uniformly at random (with replacement) from those individuals $u \in \mathcal{P}$ for which $f(u) = \max_{v \in \mathcal{P}} f(v)$ holds. From these two search points an offspring z' is created by *uniform crossover* $\mathrm{cross}(x, y)$, which samples a new search point by choosing, independently for every position $i \in [n]$ and uniformly at random, whether to copy the entry of the first or the second argument. In the *mutation phase* this offspring z' is modified by *standard bit mutation*, which flips each bit independently with some probability $p \in (0, 1)$. The so-created offspring z is evaluated. If $z \notin \mathcal{P}$ and its fitness is at least as good as $\min_{v \in \mathcal{P}} f(v)$, it replaces the worst individual in the population, ties broken uniformly at random. The requirement $z \notin \mathcal{P}$ is a so-called *diversity mechanism*.

From this description, we easily observe that from the whole population only those with a best-so-far fitness value are relevant, the others are never selected for reproduction, hence the attribute "greedy" in the name of this algorithm. When there is only one search point of best-so-far function value, the crossover simply creates a copy of this search point, and progress has to be made by mutation, while in the case that at least two different search points with best-so-far fitness exist, there is positive probability that crossover recombines these into a strictly better solution. Sudholt proved that this probability is large enough for the greedy $(\mu+1)$ GA to outperform its mutation-only analog, the $(1+1)$ EA. More precisely, it is shown in [13] that, for $\mu \geq 2$ and $n \geq 2$, the expected optimization time of the greedy $(\mu + 1)$ GA on OneMax, i.e., the expected number of function evaluations that the algorithm performs until it evaluates for the first time an optimal solution, is at most

$$\frac{\ln(n^2 p + n) + 1 + p}{p(1-p)^{n-1}(1+np)} + \frac{8n}{(1-p)^n}. \tag{1}$$

As mentioned in the introduction, this bound was later generalized to a less greedy variant of the $(\mu+1)$ GA in [15] and to one avoiding the diversity mechanism in [2]. These generalizations are not relevant to this present work.

The bound in (1) is by a multiplicative factor of about $1/(1+np)$ smaller than the expected optimization time of the $(1 + 1)$ EA. For $p = 1/n$ this factor evaluates to $1/2$, showing that for this choice of p the greedy $(\mu + 1)$ GA is about a factor of two faster than the $(1 + 1)$ EA. This advantage can be boosted by choosing larger mutation rates. In fact, the expression in (1) is minimized for $p = (1 + \sqrt{5})/(2n)$. With this mutation rate, the expected optimization time of the greedy $(\mu + 1)$ GA on OneMax is at most $1.19n \ln n + 35n$. This is better than the expected optimization time of the $(1 + 1)$ EA, but worse than the $nH_{n/2} - 1/2 \approx n \ln(n) - 0.1159n + O(1)$ expected optimization time of RLS [3].

2.2 Standard Bit Mutation: Theory vs. Practice

When the offspring created in the crossover phase of the greedy $(\mu + 1)$ GA equals one of its parents, the only source for a successful iteration is the standard bit mutation operator applied in the mutation step. Standard bit mutation is probably *the* most frequently used variation operator in evolutionary approaches for the optimization of pseudo-Boolean problems $f : \{0, 1\}^n \to \mathbb{R}$. We discuss in this section that most EA

Algorithm 1: The greedy $(\mu+1)$ $\mathrm{GA}_{\mathrm{mod}}$ with mutation probability p for the maximization of a given function $f : \{0,1\}^n \to \mathbb{R}$.

```
Choose x^(1), ..., x^(μ) from {0,1}^n independently and u.a.r. and evaluate them;
for t = 1, 2, 3, ... do
    Choose x, y ∈ arg max_{w∈P} f(w) u.a.r. (with replacement);
    if x ≠ y then z' ← cross(x, y); else z' ← x;
    if z' ∉ {x, y} then
        Sample ℓ from Bin(n, p);
    else
        Sample ℓ from Bin_{>0}(n, p);
    Sample z ← mut_ℓ(z') and evaluate f(z);
    if (z ∉ P and f(z) ≥ min_{w∈P} f(w)) then
        Choose v ∈ arg min_{w∈P} f(w) u.a.r. and replace v by z;
```

practitioners do not take the literal definition of standard bit mutation provided above too seriously, and implement a slightly different variation operator instead.

We start our discussion by observing that for every mutation rate $p \in (0,1)$ the probability that standard bit mutation merely creates a copy of the parent individual is strictly positive. More precisely, the number ℓ of bits that are flipped by the standard bit mutation operator follows the binomial distribution $\mathrm{Bin}(n,p)$. That is, for all $k \in [0..n] := \{0,1,\dots,n\}$ the probability to flip exactly k bits equals $\binom{n}{k}p^k(1-p)^{n-k}$. For $k=0$ this expression evaluates to $(1-p)^n$. The evaluation of copies, however, does not provide any new information about the problem instance f, unless f is a dynamic function or its evaluation is noisy.

The question how to deal with these offspring disunites theoretical and empirical research in evolutionary computation. While almost all theoretical runtime results for evolutionary algorithms charge the algorithms for evaluating such copies, the practitioner would typically not call the function evaluation for such offspring. Two strategies are commonly used in practice. The first one, which is the most common one for +-selection strategies, avoids to generate copies in the first place, by sampling from a conditional distribution that assigns probability 0 to sampling the parent individual. An alternative strategy, that is more reasonable for comma-selection, does include sampled copies of the parent individual in the offspring population, but does not evaluate these as their function values are already known. When the performance measure is based on counting function evaluations, both aforementioned strategies coincide for the $(\mu+1)$-type algorithms considered in this work.

We now describe how the creation of copies can be avoided. To this end, we first observe that a reasonable implementation of standard bit mutation would first sample the number ℓ of bits to flip, and then apply the mut_ℓ variation operator that flips ℓ pairwise different, uniformly selected bits. As discussed above, in the literal interpretation of standard bit mutation the number ℓ is distributed according to $\mathrm{Bin}(n,p)$. If we do not want to create copies, we only need to change the distribution that we sample from. The most common implementation of standard bit mutation uses a resampling strategy in which ℓ is sampled from $\mathrm{Bin}(n,p)$ until a strictly positive value is sampled for the first time. Thus, effectively, in this resampling approach, the *mutation strength* ℓ is sampled from the conditional binomial distribution $\mathrm{Bin}_{>0}(n,p)$, which assigns to every $k \in [n]$ a probability of $\mathrm{Bin}(n,p)(k)/\sum_{i=1}^{\infty}\mathrm{Bin}(n,p)(i) = \binom{n}{k}p^k(1-p)^{n-k}/(1-(1-p)^n)$.

2.3 The Modified Greedy $(\mu+1)$ GA

We apply the resampling idea to the greedy $(\mu+1)$ GA. To motivate this, we briefly discuss the circumstances under which the solution created in the crossover phase is identical to one of its parents. Note that only in this case we need to enforce that at least one bit is flipped by the standard bit mutation, since in the other case, the crossover may have successfully created a new solution that is at least as good as its parents. When the population contains only one search point of current-best function value, this one is (deterministically)

selected twice for the crossover step, so that the crossover operator cannot create diversity. However, even in the presence of $k > 1$ different search points of best-so far fitness, the probability to choose the same one twice equals $1/k$.[4] Furthermore, it can happen that the parents are not identical, but the offspring copies one of them. This event is also not unlikely: if we denote by d the Hamming distance of the two selected parents x and y, the probability that the offspring created by the uniform crossover cross equals either x or y is $1/2^{d-1}$. The situation $d = H(x, y) = 2$ occurs quite frequently, resulting in a $1/2$ probability that the crossover reproduces one of the two parents. In all these cases, the chances to make progress rely exclusively on the mutation phase.

Tracking whether or not the offspring created by crossover equals one of its parents is very simple, and can be done efficiently while creating it. As argued above, in such an event we would like to avoid that the mutation operator chooses mutation strength $\ell = 0$. In line with common implementations of standard bit mutation, we use the re-sampling strategy described in Section 2.2. With this re-sampling strategy, the greedy $(\mu + 1)$ GA becomes Algorithm 1, which we refer to as the greedy $(\mu + 1)$ GA_{mod}.

3 Theoretical Investigation

Following very closely the proof of Theorem 2 in [13], it is not difficult to obtain the following runtime statement, which is the main result of this paper.

Theorem 1. *For $n \geq 2$ and $\mu \geq 2$, the expected optimization time of the greedy $(\mu+1)$ GA_{mod} with mutation rate p on any* OneMax *function $\text{OM}_u : \{0,1\}^n \to [0..n], x \mapsto |\{i \in [n] \mid x_i = u_i\}|$ is at most*

$$\frac{(1-(1-p)^n)(\ln(n^2p+n)+1+p)}{p(1-p)^{n-1}(1+np)} + \frac{8n}{(1-p)^n}. \tag{2}$$

Before presenting the proof for Theorem 1, we first discuss its consequences, and why it shows that the greedy $(\mu + 1)$ GA_{mod} can hillclimb faster than any unary unbiased black-box algorithm.

For mutation rate $p = c/n$, the upper bound (2) evaluates to

$$\frac{1-(1-c/n)^n}{c(1-c/n)^{n-1}(1+c)} n\ln(n) + \Theta(n).$$

For large n, we can approximate the factor $B(c, n) := \frac{1-(1-c/n)^n}{c(1-c/n)^{n-1}(1+c)}$ in this expression by $A(c) := \frac{1-\exp(-c)}{c\exp(-c)(1+c)}$. Evaluating $B(1, n)$ and minimizing $A(c)$ with respect to c gives the following result.

Corollary 1. *For $\mu \geq 2$ the expected optimization time of the greedy $(\mu + 1)$ GA_{mod} with mutation rate $p = 1/n$ on* OneMax *is at most $(1+o(1))\frac{e-1}{2} n\ln(n) \approx 0.859140914n\ln(n)+o(n\ln n)$ and for $p = 0.773581/n$ it is at most $(1 + o(1))0.850953n\ln(n)$.*

By the result of [6], these two bounds are about 14 to 15 % smaller than the expected optimization time of *any* unary unbiased black-box algorithm. As far as we know this is the first time that a "classic" GA is shown to outperform RLS on OneMax—the only other evolutionary algorithm that we are aware of is the $(1 + (\lambda, \lambda))$ GA with fitness-based [5] and self-adjusting [4] population size.

To study the convergence towards the mutation rate used in Corollary 1, we summarize in the following table how the value of c that minimizes $B(c, n)$ changes with the problem dimension n. We also provide a numerical evaluation of the factor $B(1, n)$, the multiplicative factor of the $n\ln n$ term for the greedy $(\mu + 1)$ GA_{mod} with mutation rate $p = 1/n$.

[4] See Section 3 for a discussion of the fact that sampling the parent without replacement improves the expected optimization time of this algorithm on OneMax. We do not apply this modified parent selection rule in the greedy $(\mu + 1)$ GA_{mod} to highlight that the main improvement stems from the modified mutation step.

n	10	100	500	1 000	5 000
c	0.783953	0.774577	0.773778	0.773679	0.773599
$B(c,n)$	0.831839	0.859091	0.850581	0.850766	0.850915
$B(1,n)$	0.840587	0.857340	0.858782	0.858961	0.859105

Proof (of Theorem 1). Following [13], we say that the algorithm is on fitness level i if the best individual in the population has function value i. Like Sudholt, for each i, we distinguish two cases.

Case i.1: there is exactly one search point $x \in \mathcal{P}$ with $f(x) = i$ and for all $y \in \mathcal{P} \setminus \{x\}$ it holds that $f(y) < i$. In this situation, the offspring z is the outcome of standard bit mutation on x. The algorithm leaves this situation when (a) $f(z) > i$ or (b) $f(z) = f(x)$ and $z \neq x$. The probability for (a) to happen is at least $(n-i)p(1-p)^{n-1}/(1-(1-p)^n)$, since this is the probability that exactly one of the zero bits is flipped in the mutation phase. Likewise, the probability of event (b) is $i(n-i)p^2(1-p)^{n-2}/(1-(1-p)^n) \geq i(n-i)p^2(1-p)^{n-1}/(1-(1-p)^n)$. Once the algorithm has left case i.1 it never returns to it. This is ensured by the diversity mechanism, which allows to include z in the population only if it isn't there yet (line 10 of Algorithm 1). The total expected time spent in the cases i.1, $i = 0, \dots, n-1$ is therefore at most

$$\frac{1-(1-p)^n}{p(1-p)^{n-1}} \sum_{i=0}^{n-1} \frac{1}{(n-i)(1+ip)}.$$

The same algebraic computations as in [13] show that this expression can be bounded from above by

$$\frac{(1-(1-p)^n)(\ln(pn^2+n)+1+p)}{p(1-p)^{n-1}(1+np)}.$$

Case i.2: there are at least two different search points x and y with $f(x) = f(y) = i$ and, for all $w \in \mathcal{P}$, $f(w) \leq i$ holds. For this case we can use exactly the same arguments as Sudholt does for the original greedy $(\mu+1)$ GA: the probability to sample two different parents $x \neq y$ in the crossover step is at least $1/2$. Assuming that we are in this situation, it is not difficult to show that the probability that the intermediate offspring z' satisfies $f(z') > i$ is at least $1/4$, cf. [13] for an explicit proof. Conditioning on this event, we certainly have $z' \notin \{x, y\}$ so that the mutation strength ℓ is therefore sampled from the unconditional binomial distribution $\mathrm{Bin}(n,p)$. The probability to sample $\ell = 0$ equals $(1-p)^n$. Putting everything together, we see that, starting in case i.2, the total probability to leave fitness level i is at least $(1-p)^n/8$, so that the total expected time spent in the cases i.2, $i = 0, \dots, n-1$ is at most $8n/(1-p)^n$. □

The reader familiar with the notion of k-ary unbiased black-box algorithms may wonder if the greedy $(\mu+1)$ $\mathrm{GA}_{\mathrm{mod}}$ is unbiased, and of which arity it is. We note without proof that it is unbiased, but that care has to be taken when computing the arity of this algorithm. Line 10 of Algorithm 1 seems to suggest that the arity of this algorithm is $\mu+1$. Note however, that in particular for the case $\mu = 2$, only a mild modification of Algorithm 1 is needed to obtain a binary unbiased algorithm whose expected optimization time on OneMax also satisfies the bound stated in Theorem 1. This not being the main focus of the present work (rather are we interested in a simple example of a "classic" GA that can be proven to outperform any unary unbiased black-box optimization algorithm), we defer the details of this alternative to an extended journal version of this work.

Additional Performance Gains. It is beyond the scope of this work to analyze the tightness of the upper bounds proven in Theorem 1, and additional gains may be possible by choosing different values for p. We also remark that $\mathrm{RLS}_{\mathrm{opt}}$ (described below), the RLS-variant from [6] achieving the (up to lower order term) optimal runtime among all unary unbiased black-box algorithms on OneMax, uses fitness-dependent mutation rates. It is possible (and likely) that the greedy $(\mu+1)$ $\mathrm{GA}_{\mathrm{mod}}$, as well, could profit further from choosing its mutation rate in such an adaptive way. We have to leave this question for future work.

One may wonder why we have not abbreviated line 4 as "$z' \leftarrow \mathrm{cross}(x,y)$", regardless of whether or not $x = y$. This would of course give the same algorithm. Our variant, however, makes it more explicit that it may happen that $x = y$ is sampled in line 3. As discussed above, when $k := |\arg\max_{w \in \mathcal{P}} f(w)| = 1$, this is always the case. But also for $k > 1$ this situation can occur, because the sampling in line 3 uses replacement.

If we focus, for a moment on the situation $\mu = 2$, then one might argue that it is more "natural" to do a crossover of both parents in line 4, provided that they have the same function value. More generally, one would want to enforce $x \neq y$ whenever $k > 1$. This modification does not affect the cases $i.1$ in the proof of Theorem 1, but it does increase the success probability of the cases $i.2$ by a multiplicative factor of 2. With this observation, we easily see that the additive $\frac{8n}{(1-p)^n}$ term in the runtime bounds for the $(\mu+1)$ GA and the greedy $(\mu+1)$ GA_{mod} with mutation rate p can be replaced by $\frac{4n}{(1-p)^n}$.

4 Empirical Evaluation

Complementing the theoretical results above, we now investigate the performance of the greedy (2+1) GA_{mod} on OneMax by empirical means, to shed light on its behavior for small dimensions. As we shall see, our experiments confirm a considerable advantage of the greedy (2+1) GA_{mod} over RLS already for small problem dimensions. We use this section also to compare the greedy $(2+1)$ GA_{mod} with another crossover-based genetic algorithm, the self-adjusting $(1+(\lambda,\lambda))$ GA suggested in [5]. For a fair comparison, we modify the $(1+(\lambda,\lambda))$ GA in the same spirit in which we have modified the greedy $(\mu+1)$ GA. Finally, we also provide a comparison with RLS_{opt}, the RLS variant that in each iteration chooses the drift-maximizing mutation strength. We briefly describe these two algorithms before we present our empirical findings.

Modifying the $(1+(\lambda,\lambda))$ GA. It was shown in [4] that the $(1+(\lambda,\lambda))$ GA achieves a linear optimization time on OneMax when equipped with a self-adjusting choice of the offspring population size. No static parameter choice can achieve this performance [4] and experimental results presented in [5] suggest that already for $n \geq 1\,500$ the self-adjusting choice of the population size outperforms any static one.

For reasons of space, we cannot discuss the algorithm in great detail and refer the reader to [4] for a discussion of the self-adjusting $(1+(\lambda,\lambda))$ GA. In line with our modifications of the greedy $(\mu+1)$ GA, we change the original $(1+(\lambda,\lambda))$ GA by choosing the mutation strength ℓ from the conditional $\text{Bin}_{>0}(n,p)$ distribution (instead of sampling from $\text{Bin}(n,p)$) and by not evaluating those offspring created in the crossover phase that are identical to one of their two direct parents.

As in the original self-adjusting $(1+(\lambda,\lambda))$ GA we use a mutation rate of $p = \lambda/n$, a crossover bias $c = 1/\lambda$, and update strength $F = 3/2$. With this parametrization, the probability of the original $(1+(\lambda,\lambda))$ GA to sample a mutation strength $\ell = 0$ equals $(1-\lambda/n)^n \approx \exp(-\lambda)$. A choice of $\ell = 0$ results in an entirely useless iteration that costs 2λ function evaluations. Note further that particularly in the beginning (λ is close to one) but also in the last steps of the optimization process (λ approaches $\sqrt{n}$), the probability that an offspring created from $\text{cross}_{1/\lambda}(x,y)$ equals x or y is fairly large. It is therefore not surprising that our modified $(1+(\lambda,\lambda))$ GA_{mod} indeed corresponds to how practitioners have implemented the $(1+(\lambda,\lambda))$ GA for an empirical evaluation [9].

None of our modifications can influence the *asymptotic order* of the optimization time, since the linear performance of the original $(1+(\lambda,\lambda))$ GA is already asymptotically optimal [4]. What we do observe, however, is that our modifications have a non-trivial impact on the leading constant.

RLS with Fitness-Dependent Mutation Strengths. RLS_{opt} is the (1+1)-type heuristic which in every iteration creates one offspring y from the parent x by flipping a number of bits that is chosen to maximize the expected progress towards the optimum. y replaces x if it is at least good; i.e., if $f(y) \geq f(x)$ holds.

It was proven in [6] that this *drift maximizer* is (almost) optimal among all unary unbiased black-box algorithms. More precisely, it is shown that the performance of any unary unbiased algorithm can be better by at most an additive $o(n)$ term.

To run RLS_{opt} in our experiments, we have computed, for every tested dimension n and every fitness value $v \in [0..n-1]$ the value $\ell^*_{n,v}$ that maximizes the expected drift

$$\begin{aligned} B(n,v,\ell) &:= \mathbb{E}[\max\{\text{OM}(y) - \text{OM}(x), 0\} \mid \text{OM}(x) = v, y = \text{mut}_\ell(x)] \\ &= \sum_{i=\lceil \ell/2 \rceil}^{\ell} \frac{\binom{n-v}{i}\binom{v}{\ell-i}(2i-\ell)}{\binom{n}{\ell}}, \end{aligned} \tag{3}$$

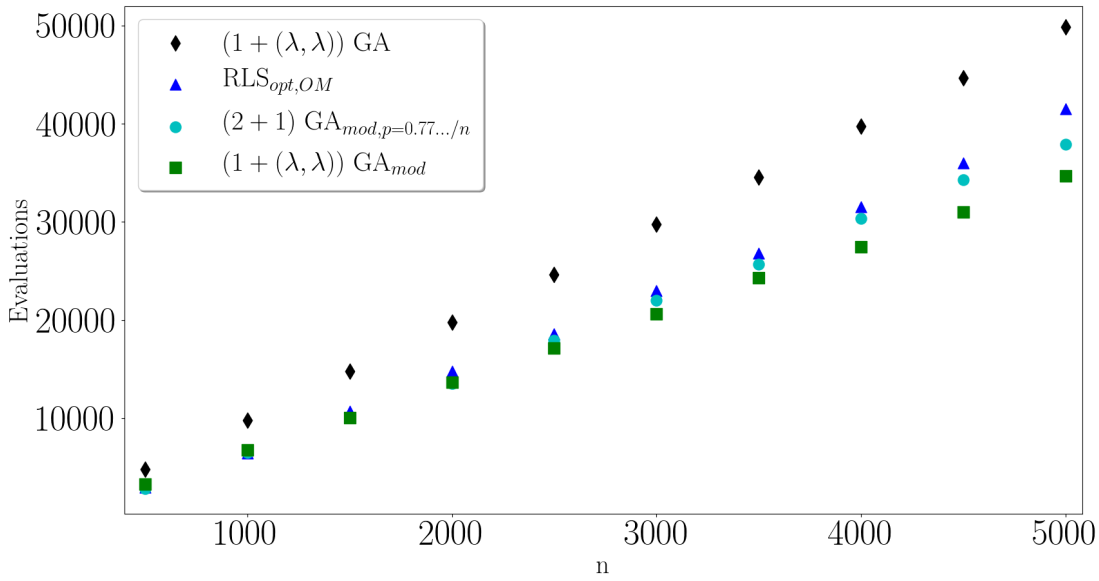

Fig. 1: Average runtimes for 100 independent runs of the respective algorithms on OneMax for different problem sizes n.

i.e., we do not work with the approximation proposed in [6] but the original drift maximizer.

Experimental Results. Figure 1 shows experimental data for the performance of the aforementioned algorithms on OneMax, for n ranging from 500 to 5 000. The $(1+(\lambda,\lambda))$ GA and the $(1+(\lambda,\lambda))$ $\mathrm{GA}_{\mathrm{mod}}$ use self-adjusting λ values, and for the greedy $(2+1)$ $\mathrm{GA}_{\mathrm{mod}}$ we use mutation rate $0.773581/n$ and the variant that recombines both parents if their function values are identical. In the reported ranges, the expected performance of the original greedy $(2+1)$ GA from [13] with mutation rate $p = (1+\sqrt{5})/(2n)$ is very similar to that of the self-adjusting $(1+(\lambda,\lambda))$ GA (cf. Figure 8 in [5]); we do not plot these data points to avoid an overloaded plot. Detailed statistical information for Figure 1 can be found in Table 1. We observe that both the $(1+(\lambda,\lambda))$ $\mathrm{GA}_{\mathrm{mod}}$ as well as the greedy $(2+1)$ $\mathrm{GA}_{\mathrm{mod}}$ are better than $\mathrm{RLS}_{\mathrm{opt}}$ already for quite small problem sizes. We also observe that, in line with the theoretical bounds, the advantage of the $(1+(\lambda,\lambda))$ $\mathrm{GA}_{\mathrm{mod}}$ over the greedy $(2+1)$ $\mathrm{GA}_{\mathrm{mod}}$ and over $\mathrm{RLS}_{\mathrm{opt}}$ increases with the problem size.

5 Conclusions

We have presented a simple example of a crossover-based heuristic that performs better than any unary unbiased black-box algorithm on the OneMax benchmark function. The mathematical proof is surprisingly easy, and raises the question why the result has been previously overlooked, despite the considerable attention that the *usefulness of crossover* question has received in the runtime analysis community.

The main idea behind our proof is a more careful performance evaluation. We therefore believe that the discussion how to measure the efficiency of an evolutionary algorithm, which had previously been suggested in [11], should be taken more seriously, in particular in light of the significant increase in the precision of state-of-the-art runtime results. We believe this question to be particularly relevant for the comparison of evolutionary algorithms with other standard optimization approaches like local search.

The proof of Theorem 1 does not invoke any involved mathematical machinery, and can be taught to undergraduate students. We hope that this makes it an appealing example for the discussion on the role of sexual reproduction in combinatorial optimization.

Acknowledgments. We thank the anonymous reviewers of this paper for their constructive feedback, which has helped us to improve the presentation of our main result. This research benefited from the support of the FMJH Program Gaspard Monge in optimization and operation research, and from the support to this program from EDF.

References

1. Carvalho Pinto, E., Doerr, C.: Discussion of a more practice-aware runtime analysis for evolutionary algorithms. In: EA'17. pp. 298–305 (2017)
2. Corus, D., Oliveto, P.S.: Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. IEEE Transactions on Evolutionary Computation (2018), to appear.
3. Doerr, B., Doerr, C.: The impact of random initialization on the runtime of randomized search heuristics. Algorithmica **75**, 529–553 (2016)
4. Doerr, B., Doerr, C.: Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm. Algorithmica **80**, 1658–1709 (2018)
5. Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing new genetic algorithms. Theoretical Computer Science **567**, 87–104 (2015)
6. Doerr, B., Doerr, C., Yang, J.: Optimal parameter choices via precise black-box analysis. In: GECCO'16. pp. 1123–1130. ACM (2016)
7. Doerr, B., Happ, E., Klein, C.: Crossover can provably be useful in evolutionary computation. Theoretical Computer Science **425**, 17–33 (2012)
8. Doerr, B., Johannsen, D., Kötzing, T., Lehre, P.K., Wagner, M., Winzen, C.: Faster black-box algorithms through higher arity operators. In: FOGA'11. pp. 163–172. ACM (2011)
9. Goldman, B.W., Punch, W.F.: Fast and efficient black box optimization using the parameter-less population pyramid. Evolutionary Computation **23**, 451–479 (2015), implementations available on https://github.com/brianwgoldman?tab=repositories
10. Jansen, T., Wegener, I.: The analysis of evolutionary algorithms - a proof that crossover really can help. Algorithmica **34**, 47–66 (2002)
11. Jansen, T., Zarges, C.: Analysis of evolutionary algorithms: from computational complexity analysis to algorithm engineering. In: FOGA'11. pp. 1–14. ACM (2011)
12. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**, 623–642 (2012)
13. Sudholt, D.: Crossover speeds up building-block assembly. In: GECCO'12. pp. 689–702. ACM (2012)
14. Sudholt, D.: A new method for lower bounds on the running time of evolutionary algorithms. IEEE Transactions on Evolutionary Computation **17**, 418–435 (2013)
15. Sudholt, D.: How crossover speeds up building block assembly in genetic algorithms. Evolutionary Computation **25**, 237–274 (2017)
16. Witt, C.: Tight bounds on the optimization time of a randomized search heuristic on linear functions. Combinatorics, Probability & Computing **22**, 294–318 (2013)

Appendix: Statistical Details for Figure 1

n	**Algorithm**	**Percentile** 2	25	50	75	98	**Mean**	**StdDev/ Mean**
500	$(1+(\lambda,\lambda))$ GA	4082	4532	4746	4986	5748	4791	8.8%
500	RLS_{opt}	1885	2481	2842	3309	4725	2964	21.6%
500	greedy $(2+1)$ GA_{mod}	1951	2422	2816	3343	4317	2928	21.4%
500	$(1+(\lambda,\lambda))$ GA_{mod}	2752	3065	3280	3499	3929	3300	9.4%
1000	$(1+(\lambda,\lambda))$ GA	8238	9206	9684	10222	11120	9754	8.1%
1000	RLS_{opt}	4575	5616	6187	7071	9160	6398	18.5%
1000	greedy $(2+1)$ GA_{mod}	4252	5503	6090	6700	8753	6200	17.0%
1000	$(1+(\lambda,\lambda))$ GA_{mod}	5855	6378	6716	6993	8060	6771	8.3%
1500	$(1+(\lambda,\lambda))$ GA	13134	14162	14604	15234	16816	14767	6.2%
1500	RLS_{opt}	7965	9621	10397	11220	14980	10678	16.1%
1500	greedy $(2+1)$ GA_{mod}	7219	8872	9554	10422	11911	9642	12.0%
1500	$(1+(\lambda,\lambda))$ GA_{mod}	8985	9522	10058	10446	11742	10054	6.9%
2000	$(1+(\lambda,\lambda))$ GA	17502	18960	19604	20384	22240	19749	5.7%
2000	RLS_{opt}	10993	12958	14424	15796	20085	14749	15.7%
2000	greedy $(2+1)$ GA_{mod}	10791	12179	13393	14963	20410	13856	16.4%
2000	$(1+(\lambda,\lambda))$ GA_{mod}	12146	13139	13511	13940	15477	13638	6.4%
2500	$(1+(\lambda,\lambda))$ GA	21828	24024	24558	25234	26874	24614	4.2%
2500	RLS_{opt}	13553	16655	18658	19803	23815	18596	13.2%
2500	greedy $(2+1)$ GA_{mod}	13511	16081	17212	18755	23870	17703	13.7%
2500	$(1+(\lambda,\lambda))$ GA_{mod}	15289	16549	17076	17673	19275	17134	5.8%
3000	$(1+(\lambda,\lambda))$ GA	27300	28716	29596	30430	32788	29736	4.9%
3000	RLS_{opt}	16569	20386	22534	25156	31789	22968	15.8%
3000	greedy $(2+1)$ GA_{mod}	17493	19601	21447	23292	30849	22081	15.4%
3000	$(1+(\lambda,\lambda))$ GA_{mod}	18860	19906	20549	21108	23088	20641	5.1%
3500	$(1+(\lambda,\lambda))$ GA	31888	33516	34624	35264	37190	34544	3.8%
3500	RLS_{opt}	19860	23569	26262	28687	36960	26801	16.1%
3500	greedy $(2+1)$ GA_{mod}	19598	22805	25340	27811	36918	25925	15.6%
3500	$(1+(\lambda,\lambda))$ GA_{mod}	21858	23468	24119	24933	27131	24276	5.0%
4000	$(1+(\lambda,\lambda))$ GA	36648	38758	39848	40390	43014	39733	3.8%
4000	RLS_{opt}	25114	28487	30448	33653	43626	31512	16.3%
4000	greedy $(2+1)$ GA_{mod}	23175	26752	28673	31389	36222	29372	12.3%
4000	$(1+(\lambda,\lambda))$ GA_{mod}	25106	26564	27466	28139	30187	27496	4.7%
4500	$(1+(\lambda,\lambda))$ GA	41698	43520	44434	45298	48550	44664	3.9%
4500	RLS_{opt}	28047	32449	34814	38433	52178	36040	15.9%
4500	greedy $(2+1)$ GA_{mod}	27140	30578	32795	35492	43957	33682	12.7%
4500	$(1+(\lambda,\lambda))$ GA_{mod}	28326	29927	30900	31551	33991	30988	4.8%
5000	$(1+(\lambda,\lambda))$ GA	46568	48378	49468	50912	54402	49857	4.9%
5000	RLS_{opt}	31760	36541	39523	44214	57874	41537	16.3%
5000	greedy $(2+1)$ GA_{mod}	30441	34279	38186	40793	50552	38437	13.3%
5000	$(1+(\lambda,\lambda))$ GA_{mod}	32234	33514	34348	35436	38117	34666	4.4%

Table 1: Statistical Details for Figure 1