How Crossover Speeds up Building Block Assembly in Genetic Algorithms

Dirk Sudholt d.sudholt@sheffield.ac.uk
Department of Computer Science, University of Sheffield, UK

doi:10.1162/EVCO_a_00171

Abstract

We reinvestigate a fundamental question: How effective is crossover in genetic algorithms in combining building blocks of good solutions? Although this has been discussed controversially for decades, we are still lacking a rigorous and intuitive answer. We provide such answers for royal road functions and OneMax, where every bit is a building block. For the latter, we show that using crossover makes *every* (μ+λ) genetic algorithm at least twice as fast as the fastest evolutionary algorithm using only standard bit mutation, up to small-order terms and for moderate μ and λ. Crossover is beneficial because it can capitalize on mutations that have both beneficial and disruptive effects on building blocks: crossover is able to repair the disruptive effects of mutation in later generations. Compared to mutation-based evolutionary algorithms, this makes multibit mutations more useful. Introducing crossover changes the optimal mutation rate on OneMax from $1/n$ to $(1+\sqrt{5})/2 \cdot 1/n \approx 1.618/n$. This holds both for uniform crossover and k-point crossover. Experiments and statistical tests confirm that our findings apply to a broad class of building block functions.

Keywords

Genetic algorithms, crossover, recombination, mutation rate, runtime analysis, theory.

1 Introduction

Ever since the early days of genetic algorithms (GAs), researchers have wondered when and why crossover is an effective search operator. In evolutionary biology it has been folklore that crossover can speed up adaptation by bringing together multiple beneficial changes that resulted from independent mutation events, famously illustrated by Muller (1932, diagram 1). The same view was taken in evolutionary computation, where building blocks were regarded as schemata of high fitness (see, e.g., Davis, 1991, p. 18; Mitchell, Forrest, and Holland, 1992; and De Jong and Spears, 1992). But as Watson and Jansen (2007) put it, *there has been a considerable difficulty in demonstrating this rigorously and intuitively*.

Many attempts at understanding crossover have been made in the past. Mitchell et al. (1992) presented so-called *royal road* functions as an example where supposedly genetic algorithms outperform other search algorithms due to the use of crossover. Royal roads divide a bit string into disjoint blocks. Each block makes a positive contribution to the fitness if all bits therein are set to 1. Blocks thus represent schemata, and all-1s configurations are building blocks of optimal solutions. However, the same authors later concluded that simple randomized hill climbers performed better than GAs (Forrest and Mitchell, 1993; Mitchell, Holland, and Forrest, 1994).

Manuscript received: February 26, 2014; revised: November 18, 2014 and October 7, 2015; accepted: October 29, 2015.

The role of crossover has been studied from multiple angles, including algebra (Rowe, Vose, and Wright, 2002), Markov chain models (Vose, 1999), infinite population models and dynamical systems (see De Jong, 2006, ch. 6, for an overview), and statistical mechanics (see, e.g., Prügel-Bennett and Rogers, 2001; Shapiro, 2001, and the references therein).

Also in biology the role of crossover is far from settled. In population genetics exploring the advantages of recombination, or sexual reproduction, is a famous open question (Barton and Charlesworth, 1998) and has been called "the queen of problems in evolutionary biology" by Bell (1982) and others. Evolutionary processes were found to be harder to analyze than those using only asexual reproduction, as they represent quadratic dynamical systems (Arora, Rabani, and Vazirani, 1994; Rabani, Rabinovich, and Sinclair, 1998).

Recent work in population genetics has focused on studying the "speed of adaptation," which describes the efficiency of evolution, in a similar vein to research in evolutionary computation (Weissman and Barton, 2012; Weissman, Feldman, and Fisher, 2010). We refer the interested reader to Paixão, Badkobeh, Barton, Corus, Dang, Friedrich, Lehre, Sudholt, Sutton, and Trubenová (2015) and Paixao, Pérez Heredia, Sudholt, and Trubenova (2015) for steps toward unifying research in both fields. Furthermore, a new theory of mixability has been proposed from the perspective of theoretical computer science (Livnat, Papadimitriou, Dushoff, and Feldman, 2008; Livnat, Papadimitriou, Pippenger, and Feldman, 2010), arguing that recombination favors individuals that are good mixers, that is, individuals that create good offspring when being recombined with others.

Several researchers independently reported empirical observations that using crossover improves the performance of evolutionary algorithms (EAs) on the simple function $\text{OneMax}(x) = \sum_{i=1}^{n} x_i$ (Lässig, 2009; Rowe, 2015) but were unable to explain why. The fact that even settings as simple as OneMax are not well understood demonstrates the need for a solid theory and serves as motivation for this work.

Runtime analysis has become a major area of research that can give rigorous evidence and proven theorems (Neumann and Witt, 2010; Auger and Doerr, 2011; Jansen, 2013). However, studies so far have eluded the most fundamental setting of building block functions. Crossover was proven to be superior to mutation only on constructed artificial examples like Jump_k (Jansen and Wegener, 2002; Kötzing, Sudholt, and Theile, 2011) and "real royal road" functions (Jansen and Wegener, 2005; Storch and Wegener, 2004), the H-IFF problem (Dietzfelbinger, Naudts, Van Hoyweghen, and Wegener, 2003), coloring problems inspired by the Ising model from physics (Fischer and Wegener, 2005; Sudholt, 2005),[1] computing unique input-output sequences for finite state machines (Lehre and Yao, 2011), selected problems from multiobjective optimization (Qian, Yu, and Zhou, 2013), and the all-pairs shortest path problem (Doerr, Happ, and Klein, 2012a; Sudholt and Thyssen, 2012; Neumann and Theile, 2010). H-IFF (Dietzfelbinger et al., 2003) and the Ising model on trees (Sudholt, 2005) consist of hierarchical building blocks. But none of these papers addressed single-level building blocks in a setting as simple as royal roads.

Watson and Jansen (2007) presented a constructed building block function and proved exponential performance gaps between EAs using only mutation and a GA. However, the definition of the internal structure of building blocks is complicated and

[1] For bipartite graphs, the problem is equivalent to the classical graph coloring problem with two colors.

artificial, and they used a tailored multideme GA to get the necessary diversity. With regard to how GAs combine building blocks, their approach does not give the intuitive explanation one is hoping for.

This paper presents such an intuitive explanation, supported by rigorous analyses. We consider royal roads and other functions composed of building blocks such as monotone polynomials. $\text{OneMax}(x) = \sum_{i=1}^{n} x_i$ is a special case where every bit is a building block. We give rigorous proofs for OneMax and show how the main proof arguments transfer to broader classes of building block functions. Experiments support the latter.

Our main results are as follows.

- We show in Section 3 that on OneMax *every* $(\mu + \lambda)$ GA with uniform crossover and standard bit mutation is at least twice as fast as *every* evolutionary algorithm (EA) that only uses standard bit mutations (up to small-order terms). More precisely, the dominating term in the expected number of function evaluations decreases from $e \cdot n \ln n$ to $e/2 \cdot n \ln n$. This holds provided that the parent population and offspring population sizes μ and λ are moderate, so that the inertia of a large population does not slow down exploitation. The reason for this speedup is that the GA can store a neutral mutation (a mutation not altering the parent's fitness) in the population, along with the respective parent. It can then use crossover to combine the good building blocks between these two individuals, improving the current best fitness. In other words, crossover can capitalize on mutations that have both beneficial and disruptive effects on building blocks, as crossover is able to repair the disruptive effects of mutation in later generations.

- The use of uniform crossover leads to a shift in the optimal mutation rate on OneMax. Section 4 demonstrates this for a simple greedy $(2+1)$ GA that always selects parents among the current best individuals. While for mutation-based EAs $1/n$ is the optimal mutation rate (Witt, 2013), the greedy $(2+1)$ GA has an optimal mutation rate of $(1+\sqrt{5})/2 \cdot 1/n \approx 1.618/n$ (ignoring small-order terms). This is because introducing crossover makes neutral mutations more useful and larger mutation rates increase the chance of a neutral mutation. Optimality is proved by means of a matching lower bound on the expected optimization time of the greedy $(2+1)$ GA that applies to *all* mask-based crossover operators (where each bit value is taken from either parent). Using the optimal mutation rate, the expected number of function evaluations is $1.19n \ln n \pm O(n \log \log n)$.

- These results are not limited to uniform crossover or the absence of linkage. Section 5 shows that the same results hold for GAs using k-point crossover, for arbitrary k, under slightly stronger conditions on μ and λ, if the crossover probability p_c is set to an appropriately small value.

- The reasoning for OneMax carries over to other functions with a clear building block structure. Experiments in Section 6 reveal similar performance differences as on OneMax for royal road functions and random polynomials with unweighted, positive coefficients. This is largely confirmed by statistical tests. There is evidence that findings also transfer to weighted building block functions like linear functions, provided that the population can store solutions with different fitness values and different building blocks until crossover is able to combine them. This is not the case for the greedy $(2+1)$ GA, but a simple $(5+1)$ GA is significantly faster on random linear functions than the optimal mutation-based EA for this class of functions, the $(1+1)$ EA (Witt, 2013).

The first result, the analysis for uniform crossover, is remarkably simple and intuitive. It gives direct insight into the working principles of GAs. Its simplicity also makes it very well suited for teaching purposes.

This work extends a preliminary conference paper (Sudholt, 2012) with parts of the results, where results were restricted to one particular GA, the greedy $(2+1)$ GA. This extended version presents a general analytical framework that applies to all $(\mu+\lambda)$ GAs, subject to mild conditions, and includes the greedy $(2+1)$ GA as a special case. To this end, we provide tools for analyzing parent and offspring populations in $(\mu+\lambda)$ GAs, which we believe are of independent interest.

Moreover, results for k-point crossover have been improved. The leading constant in the upper bound for k-point crossover in Sudholt (2012) was by an additive term of $\frac{2c}{3+3c}$ larger than that for uniform crossover, for mutation rates of c/n. This left open the question whether k-point crossover is as effective as uniform crossover for assembling building blocks in OneMax. Here we provide a new and refined analysis, which gives an affirmative answer, under mild conditions on the crossover probability.

1.1 Related Work

The literature on recombination is too vast to be reviewed comprehensively. Sastry et al. (2005) reviewed early literature and gave recommendations on the design of competent genetic algorithms based on building blocks.

In more recent work, Prügel-Bennett (2010) presented five mechanisms that advantage populations with crossover, based on empirical evidence and nonrigorous theory:

- Putting together building blocks from different solutions
- Focusing search by crossover on variables where parents differ
- The ability of a population to act as a low-pass filter of the landscape
- Hedging against bad luck in the initialization and other decisions made
- The opportunity of learning useful parameter values to balance exploration against exploitation

This work explicitly addresses the first mechanism, for which Prügel-Bennett (2010, sec. IIIA) notes "it is nontrivial to construct a toy problem which demonstrated how the building block hypothesis would work." It is shown here that the best known toy problem, OneMax, serves this purpose. We also implicitly address the second benefit, focusing search, as our analysis reveals that crossover very quickly exploits diversity in the population to create improvements on OneMax.

In terms of rigorous runtime analysis, Kötzing et al. (2011) considered the search behavior of an idealized GA on OneMax to highlight the potential benefits of crossover under ideal circumstances. If a GA is able to recombine two individuals with equal fitness that result from independent evolutionary lineages, the fitness gain can be of order $\Omega(\sqrt{n})$. The idealized GA would therefore be able to optimize OneMax in expected time $O(\sqrt{n})$ (Kötzing et al., 2011). However, this idealization cannot reasonably be achieved in realistic EAs with common search operators; hence the result should be regarded as an academic study on the *potential* benefit of crossover.

A related strand of research deals with the analysis of the Simple GA on OneMax. The Simple GA is one of the best known and best researched GAs in the field. It uses a generational model where parents are selected using fitness-proportional selection and the generated offspring form the next population. Neumann, Oliveto, and Witt (2009) showed that the Simple GA without crossover with high probability cannot optimize OneMax in less than exponential time. The reason is that the population typically contains individuals of similar fitness, and then fitness-proportional selection is similar to uniform selection. Oliveto and Witt (2014) extended this result to uniform crossover: the Simple GA with uniform crossover and population size $\mu \le n^{1/8-\varepsilon}$, $\varepsilon > 0$, still needs exponential time on OneMax. It even needs exponential time to reach a solution of fitness larger than $(1 + c) \cdot n/2$ for an arbitrary constant $c > 0$. Oliveto and Witt (2013) relaxed their condition on the population size to $\mu \le n^{1/4-\varepsilon}$. Their work does not exclude that crossover is advantageous, particularly since under the right circumstances crossover may lead to a large increase in fitness (Kötzing et al., 2011). But if there is an advantage, it is not noticeable, as the Simple GA with crossover still fails badly on OneMax (for the stated moderate population sizes).

One year after Sudholt (2012) was published, Doerr, Doerr, and Ebel (2013a) presented a groundbreaking result: they designed an EA that was proven to optimize OneMax (and any simple transformation thereof) in time $O(n\sqrt{\log n})$. This is a spectacular result, as all black-box search algorithms using only unbiased unary operators—operators modifying one individual only, and not exhibiting any inherent search bias—need time $\Omega(n \log n)$, as shown by Lehre and Witt (2012). So their EA shows that crossover can lower the expected running time by more than a constant factor. They call their algorithm a (1+(λ, λ)) EA: starting with one parent, it first creates λ offspring by mutation, with a random and potentially high mutation rate. Then it selects the best mutant and crosses it λ times with the original parent, using parameterized uniform crossover (the probability of taking a bit from the first parent is not always 1/2, but a parameter of the algorithm). This leads to a number of $O(n\sqrt{\log n})$ expected function evaluations. This bound was recently tightened to $O(n\sqrt{\log(n) \cdot \log\log\log(n)/\log\log(n)})$ (Doerr and Doerr, 2015b) and can be further decreased to $O(n)$ by self-adjusting λ (Doerr and Doerr, 2015a).

The (1+(λ, λ)) EA from Doerr et al. (2013a) is very cleverly designed to work efficiently on OneMax and similar functions. It uses a nonstandard EA design because of its two phases of environmental selection. Other differences are that mutation is performed before crossover, and mutation is not fully independent for all offspring: the number of flipping bits is a random variable determined as for standard bit mutations, but the same number of flipping bits is then used in all offspring. The focus of this work is different, as our goal is to understand how standard EAs operate and how crossover can be used to speed up building block assembly in commonly used $(\mu + \lambda)$ EAs.

2 Preliminaries

We measure the performance of the algorithm with respect to the number of function evaluations performed until an optimum is found, referred to as *optimization time*. For steady-state algorithms this equals the number of generations (apart from the initialization), and for EAs with offspring populations such as $(\mu + \lambda)$ EAs or $(\mu + \lambda)$ GAs the optimization time is by a factor of λ larger than the number of generations. Note that the number of generations needed to optimize a fitness function can often be easily decreased by using offspring populations or parallel evolutionary algorithms (Lässig and Sudholt, 2014). But this significantly increases the computational effort within one generation, so the number of function evaluations is a more fair and widely used measure.

Looking at function evaluations is often motivated by the fact that this operation dominates the execution time of the algorithm. Then the number of function evaluations is a reliable measure for wall clock time. However, the wall clock time might increase when introducing crossover as an additional search operator. Also, when increasing the mutation rate, more pseudorandom numbers might be required. Jansen and Zarges (2011) point out a case where this effect leads to a discrepancy between the number of function evaluations and wall clock time. This concern must be taken seriously when aiming at reducing wall clock time. However, each implementation must be checked individually in this respect (Jansen and Zarges, 2011). Therefore, we keep this concern in mind but still use the number of function evaluations in the following.

3 Uniform Crossover Makes $(\mu + \lambda)$ EAs Twice as Fast

We show that, under mild conditions, every $(\mu + \lambda)$ GA is at least twice as fast as its counterpart without crossover. For the latter, that is, evolutionary algorithms using only standard bit mutation, the author proved the following lower bound on the running time of a very broad class of mutation-based EAs (Sudholt, 2013). It covers all possible selection mechanisms, parent or offspring populations, and even parallel evolutionary algorithms. We slightly rephrase this result.

THEOREM 1 (Sudholt, 2013): *Let $n \geq 2$. Every EA that uses only standard bit mutation with mutation rate p to create new solutions has expected optimization time at least*

$$\frac{\min\{\ln n, \ln(1/(p^2 n))\} - \ln\ln n - 3}{p(1-p)^n}$$

on OneMax and every other function with a unique optimum, if $2^{-n/3} \leq p \leq \frac{1}{\sqrt{n}\log n}$. If $p = c/n$, $c > 0$ constant, this is at least

$$\frac{e^c}{c} \cdot n \ln n \cdot (1 - o(1)).$$

In fact, for OneMax the author proved that among all evolutionary algorithms that start with one random solution and only use standard bit mutations, the expected number of function evaluations is minimized by the simple $(1+1)$ EA (Sudholt, 2013, Theorem 13). Also the mutation rate $p = 1/n$ is the best possible choice for OneMax, leading to a lower bound of

$$en \ln n - en \ln\ln n - 3en.$$

For the special case of $p = 1/n$, Doerr, Fouz, and Witt (2011) improved this bound toward $en \ln n - O(n)$.

We show that for a range of $(\mu + \lambda)$ EAs, as defined in the following, introducing uniform crossover can cut the dominant term of the running time in half, for the standard mutation rate $p = 1/n$.

The only requirement on the parent selection mechanism is that selection does not favor inferior solutions over fitter ones. Formally, for maximizing a fitness function f,

$$\forall x, y : f(x) \geq f(y) \Rightarrow \text{Prob(select } x) \geq \text{Prob(select } y). \quad (1)$$

Algorithm 1 Scheme of a (μ+λ) GA with mutation rate p and uniform crossover with crossover probability p_c for maximizing $f\colon \{0,1\}^n \to \mathbb{R}$

```
Initialize population $\mathcal{P}$ of size $\mu \in \mathbb{N}$ uniformly at random.
while true do
    Let $\mathcal{P}' = \emptyset$.
    for $i = 1, \ldots, \lambda$ do
        With probability $p_c$ do
            Select $x_1, x_2$ with an operator respecting (1).
            Let $y :=$ uniform crossover$(x_1, x_2)$.
        otherwise do
            Select $y$ with an operator respecting (1).
        end
        Flip each bit in $y$ independently with probability $p$.
        Add $y$ to $\mathcal{P}'$.
    end
    Let $\mathcal{P}$ contain the $\mu$ best individuals from $\mathcal{P} \cup \mathcal{P}'$; break ties toward
    including individuals with the fewest duplicates in $\mathcal{P} \cup \mathcal{P}'$.
end
```

This in particular implies that equally fit solutions are selected with the same probability. Condition (1) is satisfied for all common selection mechanisms: uniform selection, fitness-proportional selection, tournament selection, cut selection, and rank-based mechanisms.

The class of $(\mu + \lambda)$ EAs covered in this work is defined in Algorithm 1. All $(\mu + \lambda)$ EA s therein create λ offspring through crossover and mutation, or just mutation, and then pick the best out of the μ previous search points and the λ new offspring.

In the case of ties, we pick solutions that have the fewest duplicates among the considered search points. This strategy was used by Jansen and Wegener (2005) in their groundbreaking work on real royal roads; it ensures a sufficient degree of diversity whenever the population contains different search points of the same fitness.

Before stating the main result of this section, we provide two lemmas showing how to analyze population dynamics. Both lemmas are of independent interest and may prove useful in other studies of population-based EAs.

The following lemma estimates the expected time until individuals with fitness at least i take over the whole population. It generalizes Lemma 3 in Sudholt (2009), which in turn goes back to Witt's (2006) analysis of the $(\mu + 1)$ EA. Note that the lemma applies to arbitrary fitness functions, arbitrary values for μ and λ, and arbitrary crossover operators; it merely relies on fundamental and universal properties of cut selection and standard bit mutations.

LEMMA 2: *Consider any $(\mu + \lambda)$ GA implementing Algorithm 1, with any crossover operator, on any n-bit fitness function. Assume the current population contains at least one individual of fitness i. The expected number of function evaluations needed for the $(\mu + \lambda)$ GA before all individuals in its current population have fitness at least i is at most*

$$\frac{O((\mu + \lambda) \log \mu)}{(1 - p_c)(1 - p)^n}.$$

This holds for any tie-breaking rule used in the environmental selection.

Proof: Call an individual *fit* if it has fitness at least i. Now estimate the expected number of generations until the population is taken over by fit individuals, called the *expected takeover time*. As fit individuals are always preferred to nonfit individuals in the environmental selection, the expected takeover time equals the expected number of generations until μ fit individuals have been created, starting with one fit individual.

For each offspring being created, there is a chance that the $(\mu+\lambda)$ GA will simply create a clone of a fit individual. This happens if during the creation of an offspring the $(\mu+\lambda)$ GA decides not to perform crossover, it selects a fit individual as parent to be mutated, and mutation does not flip any bit. The probability for this event is at least

$$(1-p_c)\cdot(1-p)^n\cdot\frac{\text{number of fit individuals in population}}{\mu},$$

since each fit individual is selected as parent with probability at least $1/\mu$.

Now divide the run of the $(\mu+\lambda)$ GA into phases in order to get a lower bound on the number of fit individuals at certain time steps. The jth phase, $0\le j\le\lceil\log_5\mu\rceil-1$, starts with the first offspring creation in the first generation, where the number of fit individuals is at least 5^j. It ends in the first generation where this number is increased to $\min\{5^{j+1},\mu\}$. Let T_j describe the random number of generations spent in the jth phase. Starting with a new generation with $\mu\ge 5^j$ fit individuals in the parent population, consider a phase of $8\mu/((1-p_c)(1-p)^n)$ offspring creations, disregarding generation bounds.

Let N_i denote the random number of new fit offspring created in the phase. Then

$$\mathrm{E}(N_i)\ge\frac{8\mu}{(1-p_c)(1-p)^n}\cdot(1-p_c)(1-p)^n\cdot\frac{5^i}{\mu}=8\cdot 5^i,$$

and by classical Chernoff bounds (see, e.g., Mitzenmacher and Upfal, 2005, ch. 4)

$$\mathrm{Prob}(N_i<4\cdot 5^i)\le e^{-\mathrm{E}(N_i)/8}\le e^{-5^i}\le e^{-1}.$$

If $N_i<4\cdot 5^i$, the phase is called unsuccessful and we consider another phase of $8\mu/((1-p_c)(1-p)^n)$ offspring creations. The expected waiting time for a successful phase is at most $1/(1-e^{-1})$, and the expected number of offspring creations until $N_i\ge 4\cdot 5^i$ is at most $8\mu/((1-p_c)(1-p)^n(1-e^{-1}))$.

Since phases start at generation bounds, we may need to account for up to $\lambda-1$ further offspring creations in between phases. This implies

$$\mathrm{E}(T_i)\le\frac{8\mu}{(1-p_c)(1-p)^n(1-e^{-1})}+\lambda,$$

and the expected takeover time is at most

$$\begin{aligned}\sum_{i=0}^{\lceil\log_5\mu\rceil-1}\mathrm{E}(T_i)&\le\lceil\log_5\mu\rceil\cdot\left(\frac{8\mu}{(1-p_c)(1-p)^n(1-e^{-1})}+\lambda\right)\\&=\frac{O((\mu+\lambda)\log\mu)}{(1-p_c)(1-p)^n}.\end{aligned}$$

□

The following simple but handy lemma relates success probabilities for created offspring to the expected number of function evaluations needed to complete a generation where such an event has first happened.

LEMMA 3: *Consider any $(\mu+\lambda)$ GA implementing Algorithm 1, and assume that in each offspring creation there is a probability at least q that some specific event occurs. Then the expected number of function evaluations to complete a generation where this event first occurs is at most*

$$\lambda - 1 + \frac{1}{q}.$$

PROOF: The expected number of trials for an event with probability q to occur is $1/q$. To complete the generation, at most $\lambda - 1$ further function evaluations are required. □

Now we are able to prove the main result of this section.

THEOREM 4: *The expected optimization time of every $(\mu+\lambda)$ GA implementing Algorithm 1 with $0 < p_c < 1$ constant, mutation probability $0 < p < 1$ and $\mu \geq 2$ on OneMax is at most*

$$\frac{\ln(n^2 p + n) + 1 + p}{p(1-p)^{n-1} \cdot (1+np)} + \frac{O((\mu+\lambda) n \log \mu)}{(1-p)^n}. \tag{2}$$

If $p = c/n$, $c > 0$ constant, and $\mu, \lambda = o((\log n)/(\log \log n))$, this bound simplifies to

$$\frac{e^c}{c \cdot (1+c)} \cdot n \ln n \cdot (1 + o(1)). \tag{3}$$

Both statements hold for arbitrary initial populations.

The main difference between the upper bound for $(\mu+\lambda)$ GAs and the lower bound for all mutation-based EAs is an additional factor of $1 + pn$ in the denominator of the upper bound. This is a factor of 2 for $p = 1/n$ and an even larger gain for larger mutation rates.

For the default value of $p = 1/n$, this shows that introducing crossover makes EAs at least twice as fast as the fastest EA using only standard bit mutation. It also implies that introducing crossover makes EAs at least twice as fast as their counterparts without crossover (i.e., where $p_c = 0$).

PROOF OF THEOREM 4: Bound (3) can be derived from (2) using $(1 - 1/x)^{x-1} \geq 1/e$ for $x > 1$ to estimate

$$\left(1 - \frac{c}{n}\right)^{n-1} = \left(1 - \frac{c}{n}\right)^{(n/c-1)\cdot c} \cdot \left(1 - \frac{c}{n}\right)^{c-1} \geq \frac{1}{e^c} \cdot \left(1 - \frac{c^2}{n}\right) = e^{-c} - O(1/n)$$

as well as $\ln(cn + n) + 1 + c/n = (\ln n) + O(1)$. Note that $(\mu+\lambda) n \log \mu = o(n \log n)$ by conditions on μ, λ; hence this and all other small-order terms are absorbed in the term $o(1)$.

In order to prove the general bound (2), we consider canonical fitness levels, that is, the ith fitness level contains all search points with fitness i. We estimate the time spent on each level i, that is, when the best fitness in the current population is i. For each fitness level we consider three cases. The first case applies when the population contains individuals on fitness levels less than i. The second case is when the population only contains copies of a single individual on level i. The third case occurs when the population contains more than one individual on level i; then the population contains different building blocks that can be recombined effectively by crossover.

All these cases capture the typical behavior of a $(\mu+\lambda)$ GA, albeit some of these cases, and even whole fitness levels, may be skipped. We obtain an upper bound on its expected optimization time by summing up expected times the $(\mu+\lambda)$ GA may spend in all cases and on all fitness levels.

Case i.1. The population contains an individual on level i and at least one individual on a lower fitness level.

A sufficient condition for leaving this case is that all individuals in the population obtain fitness at least i. Since the $(\mu + \lambda)$ GA never accepts worsenings, the case is left for good.

The time for all individuals reaching fitness at least i has already been estimated in Lemma 2. Applying this lemma to all fitness levels i, the overall time spent in all cases $i.1$ is at most

$$\frac{O((\mu+\lambda)n\log\mu)}{(1-p_c)(1-p)^n} = \frac{O((\mu+\lambda)n\log\mu)}{(1-p)^n}.$$

Case i.2. The population contains μ copies of the same individual x on level i.

In this case, each offspring created by the $(\mu + \lambda)$ GA will be a standard mutation of x. This is obvious for offspring where the $(\mu + \lambda)$ GA decides not to use crossover. If crossover is used, the $(\mu + \lambda)$ GA will pick $x_1, x_2 = x$, create $y = x$ by crossover, and hence perform a mutation on x.

The $(\mu + \lambda)$ GA leaves this case for good if either a better search point is created or if it creates another search point with i ones. In the latter case, we will create a population with two different individuals on level i. Note that due to the choice of the tie-breaking rule in the environmental selection, the $(\mu + \lambda)$ GA will always maintain at least two individuals on level i, unless an improvement with larger fitness is found.

The probability of creating a better search point in one mutation is at least $(n-i)\cdot p(1-p)^{n-1}$, as there are $n-i$ suitable 1-bit flips. The probability of creating a different search point on level i is at least $i(n-i)\cdot p^2(1-p)^{n-2}$, as it is sufficient to flip one of i 1-bits, to flip one of $n-i$ 0-bits, and not to flip any other bit. The probability of either event happening in one offspring creation is thus at least

$$\begin{aligned}&(n-i)\cdot p(1-p)^{n-1} + i(n-i)\cdot p^2(1-p)^{n-2}\\ \geq\ & p(1-p)^{n-1}\cdot(n-i)(1+ip).\end{aligned}$$

By Lemma 3, the expected number of function evaluations in case $i.2$ is at most

$$\lambda + \frac{1}{p(1-p)^{n-1}\cdot(n-i)(1+ip)}.$$

The expected number of functions evaluations made in all cases $i.2$ is hence at most

$$\begin{aligned}&\lambda n + \sum_{i=0}^{n-1}\frac{1}{p(1-p)^{n-1}\cdot(n-i)(1+ip)}\\ =\ & \lambda n + \frac{1}{p(1-p)^{n-1}}\cdot\sum_{i=0}^{n-1}\frac{1}{(n-i)(1+ip)}.\end{aligned} \tag{4}$$

The last sum can be estimated as follows. Separating the summand for $i = n-1$,

$$\begin{aligned}&\sum_{i=0}^{n-2}\frac{1}{(n-i)(1+ip)} + \frac{1}{1+(n-1)p}\\ \leq\ & \int_{i=0}^{n-1}\frac{1}{(n-i)(1+ip)}\,\mathrm{d}i + \frac{1+p}{1+np}.\end{aligned}$$

We use Equation 3.3.20 in Abramowitz and Stegun (1964) to simplify the integral and get

$$\begin{aligned}
&\left[\frac{1}{1+np}\cdot\ln\left(\frac{1+ip}{n-i}\right)\right]_0^{n-1}+\frac{1+p}{1+np}\\
&=\frac{\ln(np+1-p)+\ln(n)}{1+np}+\frac{1+p}{1+np}\\
&\le\frac{\ln(n^2p+n)+1+p}{1+np}.
\end{aligned}$$

Plugging this into (4) yields that the expected time in all cases $i.2$ is at most

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

Case i.3. The population only contains individuals on level i, not all of which are identical.

In this case we can rely on crossover recombining two different individuals on level i. As they both have different building blocks, namely, different bits are set to 1, there is a good chance that crossover will generate an offspring with a higher number of 1-bits.

The probability of performing a crossover with two different parents in one offspring creation is at least

$$p_c\cdot\frac{\mu-1}{\mu^2},$$

as in the worst case the population contains $\mu-1$ copies of one particular individual.

Assuming two different parents are selected for crossover, let these have Hamming distance $2d$, and let X denote the number of 1-bits among these positions in the offspring. Note that X is binomially distributed with parameters $2d$ and $1/2$ and its expectation is d. We estimate the probability of getting a surplus of 1-bits, as this leads to an improvement in fitness. This estimate holds for any $d\in\mathbb{N}$. Since $\text{Prob}(X<d)=\text{Prob}(X>d)$,

$$\text{Prob}(X>d)=\frac{1}{2}(1-\text{Prob}(X=d))=\frac{1}{2}\left(1-2^{-2d}\binom{2d}{d}\right)\ge\frac{1}{4}.$$

Mutation keeps all 1-bits with probability at least $(1-p)^n$. Together, the probability of increasing the current best fitness in one offspring creation is at least

$$p_c\cdot\frac{\mu-1}{\mu^2}\cdot\frac{(1-p)^n}{4}.$$

By Lemma 3, the expected number of function evaluations in case $i.3$ is at most

$$\lambda+\frac{4\mu^2}{p_c\cdot(\mu-1)\cdot(1-p)^n}.$$

The total expected time spent in all cases $i.3$ is hence at most

$$\lambda n+\frac{4\mu^2 n}{p_c\cdot(\mu-1)\cdot(1-p)^n}=\lambda n+\frac{O(\mu n)}{(1-p)^n},$$

as $p_c=\Omega(1)$.

Summing up all expected times yields a total time bound of

$$\begin{aligned}&\frac{\ln(n^2 p + n) + 1 + p}{p(1-p)^{n-1} \cdot (1+np)} + 2\lambda n + \frac{O(\mu n) + O((\mu+\lambda) n \log \mu)}{(1-p)^n}\\ =\ &\frac{\ln(n^2 p + n) + 1 + p}{p(1-p)^{n-1} \cdot (1+np)} + \frac{O((\mu+\lambda) n \log \mu)}{(1-p)^n}.\end{aligned}$$

□

The conditions on μ and λ are fairly tight; see Remark 1 in the appendix. The conditions on p_c can be relaxed to include $p_c = 1$; see Remark 2 in the appendix.

It is remarkable that the waiting time for successful crossovers in cases $i.3$ is only of order $O((\mu + \lambda)n)$. For small values of μ and λ, for instance, $\mu, \lambda = O(1)$, the time spent in all cases $i.3$ is $O(n)$, which is negligible compared to the overall time bound of order $\Theta(n \log n)$. This shows how effective crossover is in recombining building blocks.

Also note that the proof of Theorem 4 is relatively simple, as it uses only elementary arguments and, along with Lemmas 2 and 3, it is fully self-contained. The analysis therefore lends itself for teaching purposes on the behavior of evolutionary algorithms and the benefits of crossover.

The analysis has revealed that fitness-neutral mutations, that is, mutations creating a different search point of the same fitness, can help to escape from the case of a population with identical individuals. Even though these mutations do not immediately yield an improvement in terms of fitness, they increase the diversity in the population. Crossover is very efficient in exploiting this gained diversity by combining two different search points at a later stage. From Prügel-Bennett's (2010) perspective, this corresponds to crossover focusing search on bits that differ between parents.

This means that crossover can capitalize on mutations that have both beneficial and disruptive effects on building blocks: crossover is able to repair the disruptive effects of mutation in later generations.

An interesting consequence is that this affects the optimal mutation rate on OneMax. For EAs using only standard bit mutations, Witt (2013) proved that $1/n$ is the optimal mutation rate for the $(1+1)$ EA on all linear functions. Recall that the $(1+1)$ EA is the optimal mutation-based EA (in the sense of Theorem 1) on OneMax (Sudholt, 2013).

For mutation-based EAs on OneMax, neutral mutations are neither helpful nor detrimental. With crossover acting as repair mechanism, neutral mutations now become helpful. Increasing the mutation rate increases the likelihood of neutral mutations. In fact, we can easily derive better upper bounds from Theorem 4 for slightly larger mutation rates, thanks to the additional term $1 + np$ in the denominator of the upper bound.

The dominant term in (3),

$$\frac{e^c}{c \cdot (1+c)} \cdot n \ln n,$$

is minimized for c being the golden ratio $c = (\sqrt{5}+1)/2 \approx 1.618$. This leads to the following.

Corollary 5: *The asymptotically best running time bound from Theorem 4 is obtained for $p = (1+\sqrt{5})/(2n)$. For this choice the dominant term in (3) becomes*

$$\frac{e^{(\sqrt{5}+1)/2}}{\sqrt{5}+2} \cdot n \ln n \approx 1.19 n \ln n.$$

Algorithm 2 Greedy (2+1) GA with mutation rate p for maximizing $f\colon \{0,1\}^n \to \mathbb{R}$

```
Initialize population $\mathcal{P}$ of size 2 uniformly at random.
while true do
    Select $x_1, x_2$ uniformly at random from $\{x \in \mathcal{P} \mid \forall y \in \mathcal{P}\colon f(x) \geq f(y)\}$.
    Let $y :=$ crossover$(x_1, x_2)$.
    Flip each bit in $y$ independently with probability $p$.
    Let $\mathcal{P}$ contain the 2 best individuals from $\mathcal{P} \cup \{y\}$; break ties toward
    including individuals with the fewest duplicates in $\mathcal{P} \cup \{y\}$.
end
```

4 The Optimal Mutation Rate

Corollary 5 gives the mutation rate that yields the best upper bound on the running time that can be obtained with the proof of Theorem 4. However, it does not establish that this mutation rate is indeed optimal for any GA. After all, another mutation rate leads to a smaller expected optimization time.

In the following, we show for a simple $(2+1)$ GA (Algorithm 2) that the upper bound from Theorem 4 is indeed tight up to small-order terms, which establishes $p = (1+\sqrt{5})/(2n)$ as the optimal mutation rate for that $(2+1)$ GA. Proving lower bounds on expected optimization times is often a notoriously hard task, hence we restrict ourselves to a simple bare-bones GA that captures the characteristics of GAs covered by Theorem 4 and is easy to analyze. The latter is achieved by fixing as many parameters as possible.

As the upper bound from Theorem 4 grows with μ and λ, we pick the smallest possible values: $\mu = 2$ and $\lambda = 1$. The parent selection is made as simple as possible: we select parents uniformly at random from the current best individuals in the population. In other words, if we define the parent population as the set of individuals that have a positive probability to be chosen as parents, the parent population only contains individuals of the current best fitness. We call this parent selection “greedy” because it is a greedy strategy to choose the current best search points as parents.

In the context of the proof of Theorem 4, greedy parent selection implies that cases $i.1$ are never reached, as the parent population never spans more than one fitness level. So the time spent in these cases is 0. This also allows us to eliminate one further parameter by setting $p_c = 1$, as lower values for p_c were only beneficial in cases $i.1$. Setting $p_c = 1$ minimizes our estimate for the time spent in cases $i.3$. So Theorem 4 extends toward this GA (see also Remark 2 in the appendix).

We call the resulting GA a “greedy $(2+1)$ GA” because its main characteristic is the greedy parent selection. The greedy $(2+1)$ GA is defined in Algorithm 2.[2]

The following result applies to the greedy $(2+1)$ GA using any kind of mask-based crossover. A mask-based crossover is a recombination operator where each bit value is taken from either parent; that is, it is not possible to introduce a bit value that is not represented in any parent. All common crossovers are mask-based crossovers: uniform crossover, including parameterized uniform crossover, as well as k-point crossovers for

[2] Note that in Sudholt (2012) the greedy $(2+1)$ GA was defined slightly differently: duplicate genotypes are always rejected. Algorithm 2 is equivalent to the greedy $(2+1)$ GA from Sudholt (2012) for the following reasons. If the current population contains two different individuals of equal fitness and a duplicate of one of the parents is created, both algorithms reject a duplicate genotype. If the population contains two individuals of different fitness, both behave like the population only contained the fitter individual.

any k. The following result even includes biased operators like a bitwise OR, which induces a tendency to increase the number of 1-bits.

Theorem 6: *Consider the greedy (2 + 1) GA with mutation rate* $0 < p \leq 1/(\sqrt{n} \log n)$ *using an arbitrary mask-based crossover operator. Its expected optimization time on OneMax is at least*

$$\frac{\min\{\ln n, \ln(1/(p^2 n))\} - O(\log \log n)}{(1 + \max_k\{\frac{(pn)^k}{k!k!}\}) \cdot p(1-p)^n}.$$

Before giving the proof, we note that for $p = c/n$ with $0 < c \leq 4$ constant, $\max_k\{\frac{(pn)^k}{k!k!}\} = pn$ as for $0 < pn \leq 4$ and $i \in \mathbb{N} \frac{(pn)^{i+1}}{(i+1)!(i+1)!} = \frac{pn}{(i+1)^2} \cdot \frac{(pn)^i}{i!i!} \leq \frac{(pn)^i}{i!i!}$; hence a maximum is attained for $k = 1$. Then the lower bound from Theorem 6 is

$$\frac{e^c}{c \cdot (1+c)} \cdot n \ln n - O(n \log \log n).$$

This matches the upper bound (3) up to small-order terms, showing for the greedy (2 + 1) GA that the new term $1 + c$ in the denominator of the bound from Theorem 4 was not a coincidence. For $p > 4/n$, the lower bound is at least

$$(e + \Omega(1)) \cdot n \ln n.$$

Together, this establishes the optimal mutation rate for the greedy (2 + 1) GA on OneMax.

Theorem 7: *For the greedy (2 + 1) GA with uniform crossover on OneMax, mutation rate* $p = (1 + \sqrt{5})/(2n)$ *minimizes the expected number of function evaluations, up to small-order terms.*

For the proof of Theorem 6 we use the following lower-bound technique based on fitness levels by the author.

Theorem 8 (Sudholt, 2013): *Consider a partition of the search space into nonempty sets* $A_1, \ldots, A_m$. *A search algorithm* $\mathcal{A}$ *is in* A_i *or on level* i *if the best individual created so far is in* A_i. *If there are* $\chi, u_i, \gamma_{i,j}$ *for* $i < j$ *where*

(1) *the probability of traversing from level* i *to level* j *in one step is at most* $u_i \gamma_{i,j}$ *for all* $i < j$,

(2) $\sum_{j=i+1}^{m} \gamma_{i,j} = 1$ *for all* i, *and*

(3) $\gamma_{i,j} \geq \chi \sum_{k=j}^{m} \gamma_{i,k}$ *for all* $i < j$ *and some* $0 \leq \chi \leq 1$,

then the expected hitting time of A_m *is at least*

$$\sum_{i=1}^{m-1} \mathrm{Prob}(\mathcal{A} \text{ starts in } A_i) \cdot \chi \sum_{j=i}^{m-1} \frac{1}{u_j}. \quad (5)$$

Proof of Theorem 6: We prove a lower bound for the following sped-up GA instead of the original greedy (2 + 1) GA. Whenever it creates a new offspring with the same fitness, but a different bit string as the current best individual, we assume the following. The algorithm automatically performs a crossover between the two. Also, we assume that this crossover leads to the best possible offspring in a sense that all bits where both parents differ are set to 1 (i.e., the algorithm performs a bitwise OR). That is, if both search points have i 1-bits and Hamming distance $2k$, then the resulting offspring has $i + k$ 1-bits.

Because of these assumptions, at the end of each generation there is always a single best individual. For this reason we can model the algorithm by a Markov chain representing the current best fitness.

The analysis follows a lower bound for EAs on OneMax (Sudholt, 2013, Theorem 9). As in Sudholt (2013) we consider the following fitness-level partition that focuses only on the very last fitness values. Let $\ell = \lceil n - \min\{n/\log n, 1/(p^2 n \log n)\} \rceil$. Let $A_i = \{x \mid |x|_1 = i\}$ for $i > \ell$ and A_ℓ contain all remaining search points. We know from Sudholt (2013) that the GA is initialized in A_ℓ with probability at least $1 - 1/\log n$ if n is large enough.

The probability $p_{i,i+k}$ that the sped-up GA makes a transition from fitness i to fitness $i + k$ equals

$$\begin{aligned} p_{i,i+k} &= \text{Prob}(k \text{ more 0-bits than 1-bits flip}) \\ &\quad + \text{Prob}(k \text{ 0-bits and } k \text{ 1-bits flip}). \end{aligned}$$

According to Sudholt (2013, Lemma 2), for the considered fitness levels $i > \ell$ the former probability is bounded by

$$p^k (1-p)^{n-k} \cdot \frac{(n-i)^k}{k!} \cdot \left(1 + \frac{3}{5} \cdot \frac{i(n-i)p^2}{(1-p)^2}\right).$$

The latter probability is bounded by

$$\begin{aligned} &\text{Prob}(k \text{ 0-bits flip}) \cdot \text{Prob}(k \text{ 1-bits flip}) \\ &\le \frac{(n-i)^k}{k!} \cdot p^k (1-p)^{n-i-k} \cdot \frac{i^k}{k!} \cdot p^k (1-p)^{i-k} \\ &\le \frac{(n-i)^k}{k!} \cdot p^k (1-p)^n \cdot \frac{(pn)^k}{(1-p)^{2k} \cdot k!}. \end{aligned}$$

Together, $p_{i,i+k}$ is at most

$$(p(n-i))^k (1-p)^n \left(1 + \frac{3}{5} \cdot \frac{i(n-i)p^2}{(1-p)^{2+k}} + \frac{(pn)^k}{(1-p)^{2k} \cdot k!k!}\right).$$

We need to find variables u_i and $\gamma_{i,i+k}$ along with some $0 \le \chi \le 1$ such that all conditions of Theorem 8 are fulfilled. Define

$$u_i' := p(1-p)^n (n-i) \left(1 + \frac{3}{5} \cdot \frac{i(n-i)p^2}{(1-p)^3} + \frac{1}{(1-p)^2} \cdot \max_k \left(\frac{(pn)^k}{k!k!}\right)\right)$$

and

$$\gamma_{i,i+k}' := \left(\frac{p(n-i)}{(1-p)^2}\right)^{k-1}.$$

Observe that, for every $k \in \mathbb{N}$,

$$\begin{aligned} u_i' \gamma_{i,i+k}' &\ge p^k (1-p)^n (n-i)^k \left(1 + \frac{3}{5} \cdot \frac{i(n-i)p^2}{(1-p)^{1+2k}} + \frac{1}{(1-p)^{2k}} \cdot \frac{(pn)^k}{k!k!}\right) \\ &\ge p^k (1-p)^n (n-i)^k \left(1 + \frac{3}{5} \cdot \frac{i(n-i)p^2}{(1-p)^{2+k}} + \frac{(pn)^k}{(1-p)^{2k} \cdot k!k!}\right) \\ &\ge p_{i,i+k}. \end{aligned}$$

In order to fulfill the second condition in Theorem 8, we consider the following normalized variables: $u_i := u'_i \cdot \sum_{j=i+1}^{n} \gamma'_{i,j}$ and $\gamma_{i,j} := \frac{\gamma'_{i,j}}{\sum_{j=i+1}^{n} \gamma'_{i,j}}$. As $u_i \gamma_{i,j} = u'_i \gamma'_{i,j} \geq p_{i,j}$, this proves the first condition of Theorem 8.

Following the proof of Theorem 9 in Sudholt (2013), it is easy to show that for $\chi := 1 - \frac{1}{(1-p)^2 \log n}$ we get $\gamma_{i,j} \geq \chi \sum_{k=j}^{m} \gamma_{i,k}$ for all i, j with $j > i$ [the calculations in Sudholt (2013, pp. 427–428) carry over by replacing $(1-p)$ with $(1-p)^2$]. This establishes the third and last condition.

As $\gamma_{i,j} \geq \chi \sum_{k=j}^{m} \gamma_{i,k}$ is equivalent to $\gamma'_{i,j} \geq \chi \sum_{k=j}^{m} \gamma'_{i,k}$, we get

$$\sum_{j=i+1}^{n} \gamma'_{i,j} \leq \frac{\gamma'_{i,i+1}}{\chi} \leq \frac{1}{\chi},$$

which implies, using $i(n-i)p^2 \leq n(n-\ell)p^2 \leq \frac{1}{\log n}$ (Sudholt, 2013, Eq. 12) as well as $1 + x \leq 1/(1-x)$ for $x < 1$,

$$\begin{aligned}
u_i &\leq p(1-p)^n \cdot (n-i) \cdot \frac{1}{\chi} \cdot \left(1 + \frac{3}{5} \cdot \frac{i(n-i)p^2}{(1-p)^3} + \frac{1}{(1-p)^2} \cdot \max_k \left(\frac{(pn)^k}{k!k!}\right)\right) \\
&\leq p(1-p)^{n-3} \cdot (n-i) \cdot \frac{1}{\chi} \cdot \left(1 + \frac{3}{5 \log n} + \max_k \left(\frac{(pn)^k}{k!k!}\right)\right) \\
&\leq p(1-p)^{n-3} \cdot (n-i) \cdot \frac{1}{\chi} \cdot \left(\frac{1}{1 - \frac{3}{5 \log n}} + \max_k \left(\frac{(pn)^k}{k!k!}\right)\right) \\
&\leq p(1-p)^{n-3} \cdot (n-i) \cdot \frac{1}{\chi} \cdot \frac{1 + \max_k \left(\frac{(pn)^k}{k!k!}\right)}{1 - \frac{3}{5 \log n}}.
\end{aligned}$$

Invoking Theorem 8 and recalling that the first fitness level is reached with probability at least $1 - 1/\log n$, we get a lower bound of

$$\begin{aligned}
&\left(1 - \frac{1}{\log n}\right) \chi \sum_{i=\ell}^{n-1} \frac{1}{u_i} \\
&\geq \left(1 - \frac{1}{\log n}\right) \chi^2 \cdot \frac{1 - \frac{3}{5 \log n}}{1 + \max_k \left(\frac{(pn)^k}{k!k!}\right)} \cdot \frac{(1-p)^3}{p(1-p)^n} \sum_{i=\ell}^{n-1} \frac{1}{n-i} \\
&\geq \left(1 - O\left(\frac{1}{\log n}\right)\right) \cdot \frac{1}{1 + \max_k \left(\frac{(pn)^k}{k!k!}\right)} \cdot \frac{1}{p(1-p)^n} \sum_{i=\ell}^{n-1} \frac{1}{n-i},
\end{aligned}$$

where the last step used that all factors $\chi, 1 - \frac{3}{5 \log n}$, and $1 - p$ are $1 - O\left(\frac{1}{\log n}\right)$, and $\left(1 - \frac{c}{\log n}\right)^d \geq 1 - \frac{cd}{\log n}$ for any positive constants c, d. Bounding $\sum_{i=\ell}^{n-1} \frac{1}{n-i} \leq \ln(\min\{n, 1/(p^2 n)\}) - \ln(\log n)$ as in Sudholt (2013) and absorbing all small-order terms in the $-O(\log \log n)$ term from the statement gives the claimed bound. □

We also ran experiments to see whether the outcome matches our inspection of the dominating terms in the running time bounds for realistic problem dimensions. We chose $n = 1{,}000$ bits and recorded the average optimization time over 1,000 runs. The mutation rate p was set to c/n with $c \in \{0.1, 0.2, \ldots, 4\}$. The result is shown in Figure 1.

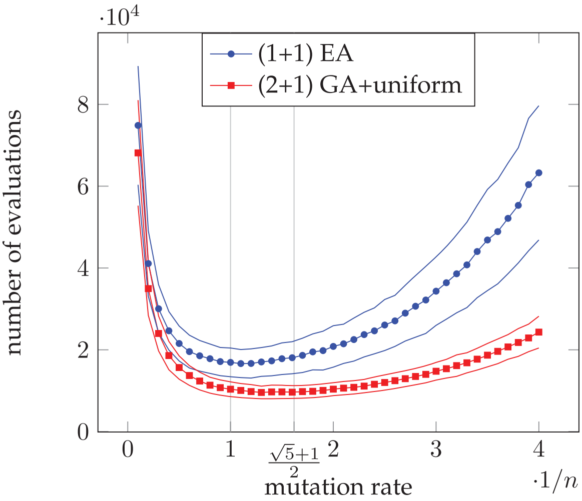


Figure 1: Average optimization times for the $(1+1)$ EA and the greedy $(2+1)$ GA with uniform crossover on OneMax with $n = 1{,}000$ bits. The mutation rate p is set to c/n with $c \in \{0.1, 0.2, \ldots, 4\}$. The thin lines show mean ± standard deviation.

One can see that for every mutation rate the greedy $(2+1)$ GA has a lower average optimization time. As predicted, the performance difference becomes larger as the mutation rate increases. The optimal mutation rates for both algorithms match minimal average optimization times. Note also that the variance/standard deviation was much lower for the GA for higher mutation rates. Preliminary runs for $n = 100$ and $n = 10{,}000$ bits gave very similar results. More experiments and statistical tests are given in Section 6.1.

5 *k*-Point Crossover

The k-point crossover operator picks k cutting points from $\{1, \ldots, n-1\}$ uniformly at random without replacement. These cutting points divide both parents into segments that are then assembled from alternating parents. That is, for parents x, y and cutting points $1 \leq \ell_1 < \ell_2 < \cdots < \ell_k \leq n-1$ the offspring will be

$$x_1 \ldots x_{\ell_1}\ y_{\ell_1+1} \ldots y_{\ell_2}\ x_{\ell_2+1} \ldots x_{\ell_3}\ y_{\ell_3+1} \ldots y_{\ell_4} \quad \ldots$$

the suffix being $y_{\ell_k+1} \ldots y_n$ if k is odd and $x_{\ell_k+1} \ldots x_n$ if k is even.

For uniform crossover we have seen that populations containing different search points of equal fitness are beneficial, as uniform crossover can easily combine the good "building blocks." This holds regardless of the Hamming distance between these different individuals and the position of bits where individuals differ.

The $(\mu + \lambda)$ GA with k-point crossover is harder to analyze, as there the probability of crossover creating an improvement depends on the Hamming distance of parents and the position of differing bits.

Consider parents that differ in two bits, where these bit positions are quite close. Then 1-point crossover has a high probability of taking both bits from the same parent. In order to recombine those building blocks, the cutting point has to be chosen between the two bit positions. A similar effect occurs for 2-point crossover if the two bit positions are on opposite ends of the bit string.

The following lemma gives a lower bound on the probability that k-point crossover combines the right building blocks on OneMax if two parents are equally fit and differ in two bits. The lemma and its proof may be of independent interest.

LEMMA 9: *Consider two search points x, y with $x_i = 1, x_{i+d} = 0, y_i = 0, y_{i+d} = 1$ for $1 \leq i < i + d \leq n$ and $x_s = y_s$ for $s \notin \{i, i + d\}$. The probability of k-point crossover of x and y, for any $1 \leq k \leq N - 1$, where $N := n - 1 \geq 4$ is the number of possible cutting points, creating an offspring with a larger number of 1-bits is at least*

$$\frac{d(N-d)}{N(N-1)}$$

and exactly d/N for $k = 1$.

PROOF: We identify cutting points with bits such that cutting point a results in two strings $x_1 \ldots x_a$ and $x_{a+1} \ldots x_n$. We say that a cutting point a separates i and $i + d$ if $a \in \{i, \ldots, i + d - 1\}$. Note that the prefix is always taken from x. The claim now follows from showing that the number of separating cutting points is odd with the claimed probability.

Let $X_{N,d,k}$ be the random variable that describes the number of cutting points separating i and $i + d$. This variable follows a hypergeometric distribution Hyp(N, d, k), illustrated by the following urn model with red and white balls. The urn contains N balls, d of which are red. We draw k balls uniformly at random without replacement. Then $X_{N,d,k}$ describes the number of red balls drawn. We define the probability of $X_{N,d,k}$ being odd, for $1 \leq d \leq N - 1$ and $1 \leq k \leq N - 1$, as

$$P(N, d, k) := \sum_{x=1,\ x \text{ odd}}^{k} \text{Prob}(X_{N,d,k} = x) = \sum_{x=1,\ x \text{ odd}}^{k} \frac{\binom{d}{x}\binom{N-d}{k-x}}{\binom{N}{k}}.$$

Note that for $k = 1$

$$P(N, d, 1) = \frac{\binom{d}{1}\binom{N-d}{0}}{\binom{N}{1}} = \frac{d}{N},$$

and for $k = 2$

$$P(N, d, 2) = \frac{\binom{d}{1}\binom{N-d}{1}}{\binom{N}{2}} = \frac{2d(N-d)}{N(N-1)}.$$

For all $1 \leq d \leq N - 1$ and all $1 \leq k \leq N - 1$ the following recurrence holds. Imagine drawing the first cutting point separately. With probability d/N, the cutting point is a separating cutting point, and then an even number of further separating cutting points is needed among the remaining $k - 1$ cutting points, drawn from a random variable $X_{N-1,d-1,k-1}$. With the remaining probability $(N - d)/N$, the number of remaining cutting points must be even, and this number is drawn from a random variable $X_{N-1,d,k-1}$. Hence

$$P(N, d, k) = \frac{d}{N} \cdot (1 - P(N - 1, d - 1, k - 1)) + \frac{N-d}{N} \cdot P(N - 1, d, k - 1). \tag{6}$$

Assume for an induction that for all $2 \leq k' < k$,

$$\frac{d(N-d)}{N(N-1)} \leq P(N, d, k') \leq 1 - \frac{d(N-d)}{N(N-1)}. \tag{7}$$

This is true for $k' = 2$ as, using $3d(N - d) \leq 3 \cdot (N/2)^2 \leq N(N - 1)$ for $N \geq 4$,

$$P(N, d, 2) = \frac{2d(N-d)}{N(N-1)} = \frac{3d(N-d) - d(N-d)}{N(N-1)} \leq 1 - \frac{d(N-d)}{N(N-1)}.$$

For $k > 2$, combining (6) and (7) yields

$$\begin{aligned}
P(N,d,k) &= \frac{d}{N} \cdot (1 - P(N-1,d-1,k-1)) + \frac{N-d}{N} \cdot P(N-1,d,k-1)\\
&\geq \frac{d}{N} \cdot \frac{(d-1)(N-d)}{(N-1)(N-2)} + \frac{N-d}{N} \cdot \frac{d(N-d-1)}{(N-1)(N-2)}\\
&= \frac{d(N-d)(d-1+N-d-1)}{N(N-1)(N-2)}\\
&= \frac{d(N-d)}{N(N-1)}.
\end{aligned}$$

The upper bound follows similarly:

$$\begin{aligned}
P(N,d,k) &\leq \frac{d}{N} \cdot \left(1 - \frac{(d-1)(N-d)}{(N-1)(N-2)}\right) + \frac{N-d}{N} \cdot \left(1 - \frac{d(N-d-1)}{(N-1)(N-2)}\right)\\
&= 1 - \frac{d(N-d)(d-1+N-d-1)}{N(N-1)(N-2)}\\
&= 1 - \frac{d(N-d)}{N(N-1)}.
\end{aligned}$$

By induction, the claim follows. □

In the setting of Lemma 9, the probability of k-point crossover creating an improvement depends on the distance between the two differing bits. Fortunately, for search points that result from a mutation of one another, this distance has a favorable distribution. This is made precise in the following lemma.

LEMMA 10: *Let x' result from x by a mutation flipping one 1-bit and one 0-bit, where the positions i, j of these bits are chosen uniformly among all 1-bits and 0-bits, respectively. Define a random variable $d := |i - j|$ and consider min$\{d, n-d\}$. Then for all $1 \leq z \leq n/2$,*

$$\textit{Prob}\,(\textit{min}\{d, n-d\} = z) \leq \frac{4}{n}.$$

PROOF: We first show the following. For any fixed index i and any integer $1 \leq z < n/2$, there are exactly two positions j such that $\min\{d, n-d\} = z$. If $i \in \{1, \ldots, n\}$ and $z \in \mathbb{N}$ are fixed, the only values for j that result in either $|i-j| = z$ or $n - |i-j| = z$ are $i+z, i-z, i+z-n$, and $i-z+n$. Note that at most two of these values are in $\{1, \ldots, n\}$. Hence, there are at most two feasible values for j for every $d \in \mathbb{N}$. Similarly, for $z = n/2$ there is just one position such that $\min\{d, n-d\} = z$.

Let ℓ denote the number of 1-bits in x. If $\ell \geq n/2$, assume that first the 0-bit is chosen uniformly at random, and then consider the uniform random choice of a corresponding 1-bit. As each bit has a probability of $1/\ell$ of being selected, and at most two choices lead to a particular value of $\min\{d, n-d\}$, we have

$$\text{Prob}\,(\min\{d, n-d\} = z) \leq \frac{2}{\ell} \leq \frac{4}{n}.$$

The case $\ell < n/2$ follows symmetrically by considering the uniform choice of the 0-bit among $n - \ell \geq n/2$ choices. □

Taken together, Lemma 9 and Lemma 10 indicate that k-point crossover has a good chance of finding improvements through recombining the right "building blocks."

Algorithm 3 Refined tie-breaking rule "dup-old"

```
Let $\mathcal{P}$ contain the $\mu$ best individuals from $\mathcal{P} \cup \mathcal{P}'$; break ties toward
including individuals with the fewest duplicates in $\mathcal{P} \cup \mathcal{P}'$. If there are still
ties, break them toward including older individuals.
```

However, this is based on the population containing potential parents of equal fitness that only differ in two bits.

The following analysis shows that the population is likely to contain such a favorable pair of parents. However, such a pair might get lost again if other individuals of the same fitness are being created, after all duplicates have been removed from the population. For parents that differ in more than 2 bits, Lemma 9 does not apply; hence we do not have an estimate of how likely such a crossover will find an improvement.

In order to avoid this problem, we consider a more detailed tie-breaking rule. As before, individuals with fewer duplicates are preferred. In case there are still ties after considering the number of duplicates, the $(\mu + \lambda)$ GA will retain older individuals. This refined tie-breaking rule is shown in Algorithm 3. As shown in the remainder, it implies that once a favorable pair of parents with Hamming distance 2 has been created, this pair will never get lost.

This tie-breaking rule, called "dup-old," differs from the one used for the experiments in Figure 1 and those in Section 6. There, we broke ties uniformly at random in case individuals are tied with respect to both fitness and the number of duplicates. We call the latter rule "dup-rnd." Experiments for the greedy $(2+1)$ GA comparing tie-breaking rules dup-old and dup-rnd over 1,000 runs indicate that performance differences are very small (see Figure 2).[3]

Note, however, that on functions with plateaus, like royal road functions, retaining the older individuals prevents the $(\mu + \lambda)$ GA from performing random walks on the plateau, once the population has spread such that there are no duplicates of any individual. In this case performance is expected to deteriorate when breaking ties toward older individuals.

With the refined tie-breaking rule, the performance of $(\mu + \lambda)$ GAs is as follows.

THEOREM 11: *The expected optimization time of every $(\mu + \lambda)$ GA implementing Algorithm 1 with tie-breaking rule dup-old from Algorithm 3, $2 \leq \mu = O(1)$, $\lambda < \mu$, $p_c = o(1)$ and $p_c = \omega(1/\log n)$, $p = c/n$ for some constant $c > 0$, and k-point crossover with any $1 \leq k \leq n-2$, on OneMax is at most*

$$\frac{e^c}{c \cdot (1+c)} \cdot n \ln n \cdot (1 + o(1)).$$

This bound equals the upper bound (3) for $(\mu + \lambda)$ GAs with uniform crossover. It improves upon the previous upper bound for the greedy $(2+1)$ GA (Sudholt, 2012, Theorem 8), whose dominant term was by an additive term of $\frac{2c}{3+3c} \cdot n \ln n$ larger. The reason is that for the $(2+1)$ GA favorable parents could get lost, which is now prevented by the dup-old tie-breaking rule and conditions on p_c.

[3] Even though differences are small, one-sided Mann–Whitney U tests reveal some statistically significant differences: for 1-point crossover dup-rnd is significantly faster than dup-old on a significance level of .001 for mutation rates at least $2.4/n$ (with two exceptions, $2.8/n$ and $3.6/n$, with p values still below 0.003). Contrarily, dup-old was significantly faster for 2-point crossover for mutation rates in the range of $0.8/n$ to $3/n$.

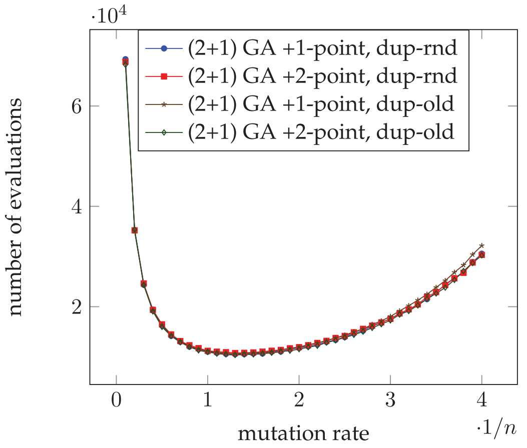


Figure 2: Average optimization times on OneMax with $n = 1{,}000$ bits over 1,000 runs for the greedy $(2+1)$ GA with 1- and 2-point crossover using different tie-breaking rules if individuals are tied with regard to fitness and the number of duplicates. “dup-rnd” breaks these ties randomly, whereas “dup-old” (Algorithm 3) prefers older individuals. The mutation rate p is set to c/n with $c \in \{0.1, 0.2, \ldots, 4\}$.

The conditions $p_c = o(1)$ as well as $\mu, \lambda = O(1)$ are useful because they allow us to estimate the probability that a single good individual takes over the whole population with copies of itself.

In the remainder of this section we work toward proving Theorem 11 and assume that $n \geq n_0$ for some n_0 chosen such that all asymptotic statements that require a large enough value of n hold true. For $n < n_0$ there is nothing to prove, as the statement holds trivially for bounded n.

We again estimate the time spent on each fitness level i, that is, when the best fitness in the current population is i. To this end, the focus is on the higher fitness levels $i \geq n - n/\log n$ where the probability of creating an offspring on the same level can be estimated nicely. The time for reaching these higher fitness levels only constitutes a small-order term, compared to the claimed running time bound. The following lemma proves this claim in a more general setting than needed for the proof of Theorem 11. In particular, it holds for arbitrary tie-breaking rules and crossover operators.

Lemma 12: *For every $(\mu + \lambda)$ GA implementing Algorithm 1 with $\mu, \lambda = O(1)$, $p_c = 1 - \Omega(1)$, and $p = c/n$ for a constant $c > 0$, using any initialization and any crossover operator, the expected time until a fitness level $i \geq n - n/\log n$ is reached for the first time is $o(n \log n)$.*

A proof is given in the appendix.

In the remainder of the section we focus on higher fitness levels $i \geq n - n/\log n$ and specify the different cases on each such fitness level. The cases $i.1$, $i.2$, and $i.3$ are similar to the ones for uniform crossover, with additional conditions on the similarity of individuals in cases $i.2$ and $i.3$. We also have an additional error state that accounts for undesirable and unexpected behavior. We pessimistically assume that the error state cannot be left toward other cases on level i.

Case i.1. The population contains an individual on level i and at least one individual on a lower fitness level.

Case i.2. The population contains μ copies of an individual x on level i.

Case i.3. The population contains two search points x, y with current best fitness i, where y resulted from a mutation of x and the Hamming distance of x and y is 2.

Case i.error. An error state reached from any case $i.\cdot$ when the best fitness is i and none of the prior cases applies.

The difference from the analysis of uniform crossover is that in case $i.2$ we rely on the population collapsing to copies of a single individual. This helps to estimate the probability of creating a favorable parent-offspring pair in case $i.3$, as the $(\mu+\lambda)$ GA effectively only performs mutations of x while being in case $i.2$.

LEMMA 13: *Consider any $(\mu+\lambda)$ GA as defined in Theorem 11, with parameters $2 \leq \mu = O(1)$, $\lambda < \mu$, $p_c = o(1)$ and $p_c = \omega(1/\log n)$, $p = c/n$ for some constant $c > 0$. The total expected time spent in all cases $i.1$, $i.2$, and $i.3$ across all $i \geq n - n/\log n$ is at most*

$$\frac{e^c}{c \cdot (1+c)} \cdot n \ln n + o(n \log n).$$

PROOF: We have already analyzed the expected time in cases $i.1$ and $i.2$ across all fitness levels. As in the proof of Theorem 4, we use Lemma 2 and get that the expected time spent in all cases $i.1$ is at most

$$\frac{O((\mu+\lambda)n \log \mu)}{(1-p_c)(1-p)^n} = O(n).$$

In case $i.2$ the algorithm behaves like the one using uniform crossover described in Theorem 4, as both crossover operators are working on identical individuals. As before, case $i.2$ is left if either a better offspring is created or a different offspring with i ones is created. In the latter case, either case $i.3$ or the error state i.error is reached. By the proof of Theorem 4, we know that the expected time spent in cases $i.2$ across all levels i is bounded by

$$\lambda n + \frac{\ln(n^2 p + n) + 1 + p}{p(1-p)^{n-1} \cdot (1+np)} = \frac{e^c}{c \cdot (1+c)} \cdot n \ln n + O(n).$$

Now we estimate the total time spent in all cases $i.3$. As this time turns out to be comparably small, the fact that not all these cases are actually reached can be ignored.

Case $i.3$ implies that the population contains a parent-offspring pair x, y with Hamming distance 2. Consider the mutation that has created this offspring, and note that this mutation flips each 1-bit and each 0-bit with the same probability. If a, b with $a < b$ denote the bit positions where x and y differ, then $D := b - a$ is a random variable with support $\{1, \ldots, n-1\}$. By the law of total expectation,

$$\mathrm{E}(T_{i.3}) = \sum_{d=1}^{n-1} \mathrm{E}(T_{i.3} \mid D = d) \cdot \mathrm{Prob}(D = d). \tag{8}$$

We first bound the conditional expectation by considering probabilities for improvements. If $D = d$ then crossover is successful if crossover is performed (probability p_c), if the search point where bit a is 1 is selected as first parent (probability at least $1/\mu$), if the remaining search point in $\{x, y\}$ is selected as second parent (probability at least $1/\mu$), and if cutting points are chosen that lead to a fitness improvement. The latter event has probability at least $d(N-d)/(N(N-1))$ by Lemma 9, with $N := n - 1$. Finally, we need to assume that the following mutation does not destroy any fitness improvements (probability at least $(1-p)^n$). The probability of a successful crossover is then at least, using $d(N-d) = \min(d, N-d) \cdot \max(d, N-d) \geq \min(d, N-d) \cdot N/2$,

$$\frac{p_c(1-p)^n}{\mu^2} \cdot \frac{d(N-d)}{N(N-1)} \geq \frac{p_c(1-p)^n}{\mu^2} \cdot \frac{\min(d,(N-d))}{2(N-1)} \geq \frac{p_c(1-p)^n}{\mu^2} \cdot \frac{\min(d, n-d)-1}{2n}.$$

Another means of escaping from case $i.3$ is by not using crossover but having mutation create an improvement. The probability for this is at least

$$(1-p_c) \cdot (n-i)p(1-p)^{n-1} \geq \gamma \cdot \frac{n-i}{n} \tag{9}$$

for a constant $\gamma > 0$. Applying Lemma 3,

$$\mathrm{E}(T_{i.3} \mid D = d) \leq \lambda + \frac{1}{\frac{p_c(1-p)^n}{\mu^2} \cdot \frac{\min(d,n-d)-1}{2n} + \gamma \cdot \frac{n-i}{n}}. \tag{10}$$

Note that this upper bound is nonincreasing with $\min(d, n-d)$. We are therefore pessimistic when replacing $\min(d, n-d)$ by the pessimistic probability estimations from Lemma 10. Combining this with (8) and (10) yields

$$\begin{aligned}\mathrm{E}(T_{i.3}) &\leq \lambda + \sum_{z=1}^{n/4} \frac{4}{n} \cdot \frac{1}{\frac{p_c(1-p)^n}{\mu^2} \cdot \frac{z-1}{2n} + \gamma \cdot \frac{n-i}{n}} \\ &\leq \lambda + O(\mu^2/p_c) \cdot \sum_{z=1}^{n/4} \frac{1}{z-1+n-i}.\end{aligned}$$

The last sum is estimated as follows.

$$\begin{aligned}\sum_{z=1}^{n/4} \frac{1}{z-1+n-i} = \sum_{z=0}^{n/4-1} \frac{1}{z+n-i} &= \frac{1}{n-i} + \sum_{z=1}^{n/4-1} \frac{1}{z+n-i} \\ &\leq 1 + \int_{z=0}^{n/4} \frac{1}{z+n-i} \,\mathrm{d}z \\ &= 1 + \ln\left(1 + \frac{n/4}{n-i}\right).\end{aligned}$$

Along with $\lambda = O(1)$, $n/4 \leq n$, and $O(\mu^2/p_c) = o(\log n)$, we get

$$\mathrm{E}(T_{i.3}) \leq o(\log n) \cdot \left(1 + \ln\left(1 + \frac{n}{n-i}\right)\right).$$

For the sum $T_{\cdot,3} = \sum_{i=0}^{n-1} T_{i,3}$ we then have the following:

$$\begin{aligned}\mathrm{E}(T_{\cdot 3}) &\leq o(n \log n) + o(\log n) \cdot \sum_{i=0}^{n-1} \ln\left(1 + \frac{n}{n-i}\right) \\ &= o(n \log n) + o(\log n) \cdot \sum_{i=1}^{n} \ln\left(1 + \frac{n}{i}\right) \\ &\leq o(n \log n) + o(\log n) \cdot \int_{i=0}^{n} \ln\left(1 + \frac{n}{i}\right) \,\mathrm{d}i \\ &= o(n \log n)\end{aligned}$$

as the integral is $2\ln(2)n$. This completes the proof. □

The remainder of the proof is devoted to estimating the expected time spent in the error state. To this end we need to consider events that take the $(\mu + \lambda)$ GA "off course," that is, deviating from situations described in cases $i.1$, $i.2$, and $i.3$.

Since case $i.3$ is based on offspring with Hamming distance 2 to their parents, one potential failure is that an offspring with fitness i but Hamming distance greater than 2 to its parent is being created. This probability is estimated in the following lemma.

LEMMA 14: *For $i \in \{1, \dots, n-1\}$ let $p^{(i)}$ denote the probability that standard bit mutation with mutation rate $0 < p \leq 1/2$ of a search point with i 1-bits creates a different offspring with i 1-bits. If $i(n-i)p^2(1-p)^{-2} \leq 1/2$, then*

$$i(n-i)p^2(1-p)^{n-2} \;\leq\; p^{(i)} \;\leq\; i(n-i)p^2(1-p)^{n-2} \cdot \left(1 + \frac{2i(n-i)p^2}{(1-p)^2}\right).$$

The probability that additionally the offspring has Hamming distance larger than 2 to its parent is at most

$$2i^2(n-i)^2p^4(1-p)^{n-4}.$$

The proof is found in the appendix.

Another potential failure occurs if the population does not collapse to copies of a single search point, that is, the transition from case $i.1$ to case $i.2$ is not made. First estimate the probability of mutation unexpectedly creating an individual with fitness i.

LEMMA 15: *The probability that a standard bit mutation with mutation probability $0 < p < 1$ creates a search point with i ones out of a parent with fewer than i ones is at most*

$$p(n-i+1) \cdot e^{(pn)^2/4+1}.$$

Note that for the special case $p = 1/n$, Doerr, Johannsen, and Winzen (2012b, Lemma 13) give an upper bound of $(n-i+1)/n$. This is because the highest probability for a jump to fitness level i is attained when the parent is on level $i-1$. However, for larger mutation probabilities this is no longer true in general; there are cases where the probability of jumping to level i is maximized for parents on lower fitness levels. Hence, a closer inspection of transition probabilities between different fitness levels is required; see the proof in the appendix.

Using Lemma 15, we can now estimate the probability of the $(\mu + \lambda)$ GA not collapsing to copies of a single search point as described in case $i.2$.

LEMMA 16: *Consider any $(\mu + \lambda)$ GA as defined in Theorem 11, with parameters $2 \leq \mu = O(1)$, $\lambda < \mu$, $p_c = o(1)$ and $p_c = \omega(1/\log n)$, $p = c/n$ for some constant $c > 0$, and fix a fitness level $i < n$. The probability that the $(\mu + \lambda)$ GA will reach a population containing different individuals with fitness i before either reaching a population containing only copies of the same individual on level i or reaching a higher fitness level is at most*

$$O(\mu \log \mu) \cdot \left(p_c + \frac{n-i}{n}\right).$$

PROOF: We show that there is a good probability of repeatedly creating clones of individuals with fitness i (or finding an improvement) and avoiding the following *bad* event. A bad event happens if an individual on fitness level i is created in one offspring creation by means other than through cloning an existing individual on level i.

The probability of a bad event is bounded as follows. In case crossover is being used, which happens with probability p_c, bound the probability of a bad event by the trivial bound 1. Otherwise, such an individual needs to be created through mutation from either a worst fitness level or by mutating a parent on level i. The probability for the former is bounded from above by Lemma 15. The probability for the latter is at most $p(n-i)$, as it is necessary to flip one out of $n{-}i$ 0-bits. Using $n-i+1 \leq 2(n-i)$, the probability of a bad event on level i is hence bounded from above by

$$\begin{aligned} & p_c + (1-p_c) \cdot \left(p(n-i+1) \cdot e^{(pn)^2/4+1} + p(n-i)\right) \\ \leq\ & p_c + \left(\frac{2c(n-i)}{n} \cdot e^{c^2/4+1} + \frac{c(n-i)}{n}\right) = p_c + \kappa \cdot \frac{n-i}{n}, \end{aligned}$$

where $\kappa := 2c \cdot e^{c^2/4+1} + c$ is a constant. The $(\mu+\lambda)$ GA will only reach a population containing different individuals with fitness i as stated if a bad event happens before the population has collapsed to copies of a single search point or moved on to a higher fitness level.

Consider the first generation where an individual of fitness i is reached for the first time. Since it might be possible to create several such individuals in one generation, we consider all offspring creations being executed sequentially and consider the possibility of bad events for all offspring creations following the first offspring on level i. Let X be the number of function evaluations following this generation before all individuals in the population have fitness at least i. By Lemma 2, we have

$$\mathrm{E}(X) = O\left(\frac{(\mu+\lambda)\log\mu}{(1-p_c)(1-p)^n}\right) = O(\mu\log\mu).$$

Considering up to λ further offspring creations in the first generation leading to level i, and completing the generation at the end of the X function evaluations, we have fewer than $X + 2\lambda$ trials for bad events. The probability that one of these is bad is bounded by

$$\begin{aligned} \left(\sum_{t=1}^{X} t \cdot \mathrm{Prob}(X=t) + 2\lambda\right) \cdot \left(p_c + \kappa \cdot \frac{n-i}{n}\right) &= (\mathrm{E}(X) + 2\lambda) \cdot \left(p_c + \kappa \cdot \frac{n-i}{n}\right) \\ &= O(\mu\log\mu) \cdot \left(p_c + \kappa \cdot \frac{n-i}{n}\right). \end{aligned}$$

Absorbing κ in the O-term yields the claimed result. □

Now we can estimate the expected time spent in all error states i.error for $i \geq n - n/\log n$.

Lemma 17: *Consider any $(\mu+\lambda)$ GA as defined in Theorem 11, with parameters $2 \leq \mu = O(1)$, $\lambda < \mu$, $p_c = o(1)$ and $p_c = \omega(1/\log n)$, $p = c/n$ for some constant $c > 0$. The expected time spent in all states i.error, for $i \geq n - n/\log n$, is at most*

$$O(n + p_c \cdot n \ln n) = o(n \log n).$$

Proof: The $(\mu+\lambda)$ GA only spends time in an error state if it is actually reached. So first calculate the probability that state i.error is reached from case i.1, i.2, or i.3.

Lemma 16 states that the probability of reaching a population with different individuals on level i before reaching case i.2 or a better fitness level is

$$O(\mu\log\mu) \cdot \left(p_c + \frac{n-i}{n}\right) = O\left(p_c + \frac{n-i}{n}\right).$$

We pessimistically ignore the possibility that case $i.3$ might be reached if this happens; thus the above is an upper bound for the probability of reaching i.error from case $i.1$.

Recall that in case $i.2$ all individuals are identical, so crossover has no effect and the $(\mu+\lambda)$ GA only performs mutations. First consider the case $\lambda = 1$. Note that $i \geq n - n/\log n$, along with $p = c/n$, implies that $i(n-i)p^2(1-p)^{-2} \leq 1/2$; hence Lemma 14 is in force. According to Lemma 14 the probability of leaving case $i.2$ by creating a different individual with fitness i is at least $i(n-i)p^2(1-p)^{n-2}$. The probability of doing this with an offspring of Hamming distance greater than 2 to its parent is at most $2i^2(n-i)^2p^4(1-p)^{n-4}$ (second statement of Lemma 14). So the conditional probability of reaching the error state when leaving case $i.2$ toward another case on level i is at most

$$\frac{2i^2(n-i)^2p^4(1-p)^{n-4}}{i(n-i)p^2(1-p)^{n-2}} = 2i(n-i)p^2(1-p)^{-2}. \tag{11}$$

In case $\lambda > 1$ note that case $i.3$ is reached if there is a single offspring with fitness i and Hamming distance 2 to its parent. Such an offspring is guaranteed to survive, as we assume $\lambda < \mu$ and offspring with many duplicates are removed first. Thus in case several offspring with fitness i and differing from their parent are created, *all* of them need to have Hamming distance larger than 2 in order to reach i.error from case $i.2$. This probability decreases with increasing λ; hence the probability bound (11) also holds for $\lambda > 1$.

Finally, case $i.3$ implies that there exists a parent-offspring pair x, y with Hamming distance 2. In a new generation these two offspring, or at least one copy of each, will always survive: individuals with multiple duplicates are removed first, and if among current parents and offspring more than μ individuals exist with no duplicates, x and y will be preferred over newly created offspring. So the probability of reaching the error state from case $i.3$ is 0.

If the error state is reached, according to (9) we have a probability of at least $\gamma \cdot \frac{n-i}{n}$ of finding a better individual in one offspring creation, for a constant $\gamma > 0$. Using Lemma 3 as before, this translates to at most $\lambda + \frac{1}{\gamma} \cdot \frac{n}{n-i}$ expected function evaluations. So the expected time spent in case i.error is at most

$$\begin{aligned}
&\lambda + \left(2i(n-i)p^2(1-p)^{-2} + O\left(p_c + \frac{n-i}{n}\right)\right) \cdot \frac{1}{\gamma} \cdot \frac{n}{n-i}\\
&= \lambda + \left(\frac{2}{\gamma} \cdot inp^2(1-p)^{-2} + O\left(p_c \cdot \frac{n}{n-i} + 1\right)\right)\\
&= O\left(p_c \cdot \frac{n}{n-i}\right) + O(1),
\end{aligned}$$

as both $\lambda = O(1)$ and $inp^2(1-p)^{-2} \leq (pn)^2 \cdot (1-p)^{-2} = O(1)$. The total expected time across all error states is at most

$$O\left(n + p_c \cdot n \cdot \sum_{i=0}^{n-1} \frac{1}{n-i}\right) = O(n + p_c \cdot n \ln n).$$

□

Now Theorem 11 follows from all previous lemmas.

Proof of Theorem 11: The claimed upper bound now follows from adding the upper bounds on the expected time on the smaller fitness levels (Lemma 12) to the expected times spent in all considered cases (Lemma 13 and Lemma 17). □

Some of the technical conditions from Theorem 11 involving μ, λ, and p_c could be relaxed if it is possible to generalize Lemmas 9 and 10 toward more than 2 differing bits between individuals of equal fitness.

Figure 3, discussed in the following Section 6, presents further experiments and statistical tests. Figure 3 includes a comparison of uniform crossover and k-point crossover in the greedy $(2+1)$ GA.

6 Extensions to Other Building Block Functions

6.1 Royal Roads and Monotone Polynomials

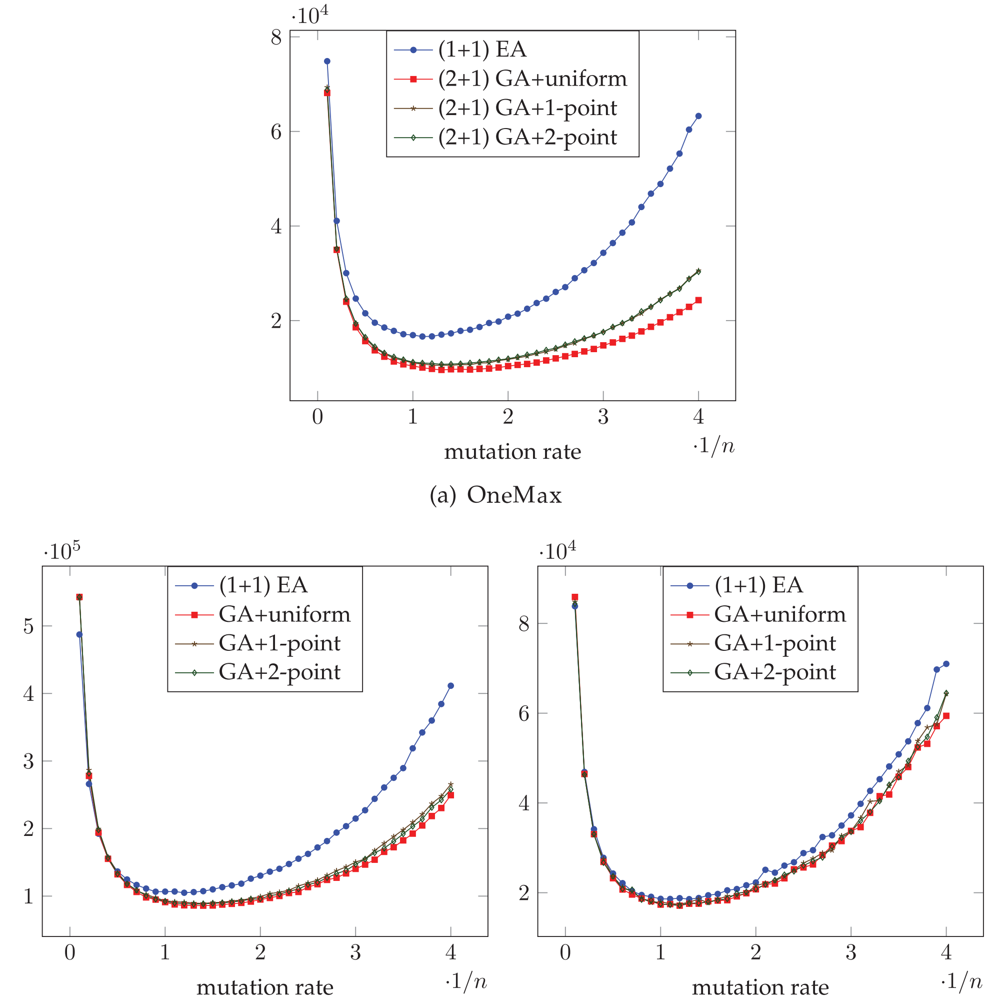
So far, our theorems and proofs have been focused on OneMax only. This is because we do have very strong results about the performance of EAs on OneMax at hand. However, the insights gained stretch far beyond OneMax. Royal road functions generally consist of larger blocks of bits. All bits in a block need to be set to 1 in order to contribute to the fitness; otherwise the contribution is 0. All blocks contribute the same amount to the fitness, and the fitness is just the sum of all contributions.

The fundamental insight we have gained for neutral mutations also applies to royal road functions. If there is a mutation that completes one block but destroys another block, this is a neutral mutation and the offspring will be stored in the population of a $(\mu+\lambda)$ GA. Then crossover can recombine all finished blocks in the same way as for OneMax. The only difference is that the destroyed block may evolve further. More neutral mutations can occur that only alter bits in the destroyed block. Then the population can be dominated by many similar solutions, and it becomes harder for crossover to find a good pair for recombination. However, as crossover has generally a very high probability of finding improvements, the last effect probably plays only a minor role.

A theoretical analysis of general royal roads up to the same level of detail as for OneMax is harder but not impossible. So far, results on royal roads and monotone polynomials have been mostly asymptotic (Wegener and Witt, 2005; Doerr, Sudholt, and Witt, 2013b). Only recently, Doerr and Künnemann (2013) presented a tighter runtime analysis of offspring populations for royal road functions, which may lend itself to a generalization of our results on OneMax in future work.

For now, we use experiments to see whether the performance is similar to that on OneMax. We use royal roads with $n = 1{,}000$ bits and block size 5, that is, we have 200 pairwise disjoint blocks of 5 bits each. We also consider random monotone polynomials. Instead of using disjoint blocks, we use 1,000 monomials of degree 5 (conjunctions of 5 bits): each monomial is made up of 5 bit positions chosen uniformly at random, without replacement. This leads to a function similar to royal roads, but "blocks" are broken up and can share bits; bit positions are completely random. Figure 3 shows the average optimization times in 1,000 runs on all these functions, for the $(1+1)$ EA and the greedy $(2+1)$ GA with uniform, 1-point, and 2-point crossover. We chose the last two because k-point crossovers for odd k treat ends of bit strings differently from those for even k: for odd k two bits close to opposite ends of a bitstring have a high probability to be taken from different parents, whereas for even k there is a high chance that both will be taken from the same parent (see Lemma 9 for $k = 2$ and the special case of $k = 1$).

For consistency and simplicity, use $p_c = 1$ and the tie-breaking rule dup-rnd in all settings, that is, ties in fitness are broken toward minimum numbers of duplicates and any remaining ties are broken uniformly at random. For OneMax this does not perfectly match the conditions of Theorem 11, as they require a lower crossover probability,



(a) OneMax

(b) Royal road

(c) Random polynomials

Figure 3: Average optimization times over 1,000 runs for $(1+1)$ EA and the greedy $(2+1)$ GA with various crossover operators on functions with $n = 1{,}000$ bits: OneMax, a royal road function with block size 5, and random polynomials with 1,000 unweighted monomials of degree 5. The mutation rate is c/n with $c \in \{0.1, 0.2, \ldots, 4\}$.

$p_c = o(1)$, and tie-breaking rule dup-old. But the experiments show that k-point crossover is still effective when these conditions are not met.

On OneMax both k-point crossovers are better than the $(1+1)$ EA but slightly worse than uniform crossover. This is in accordance with the observation from our analyses that improvements with k-point crossover might be harder to find if the differing bits are in close proximity.

For royal roads the curves are very similar. The difference between the $(1+1)$ EA and the greedy $(2+1)$ GA is just a bit smaller. For random polynomials there are visible differences, albeit smaller. Mann–Whitney U tests confirm that wherever there

Table 1: Summary of the results of two-sided Mann–Whitney U tests on the data from Figure 3. For each function the table shows pairwise comparisons between the $(1+1)$ EA and the greedy GA with uniform, 1-point, and 2-point crossover, respectively. Here p is the p value output by the statistics package R (version 2.8.1), and c is the constant in the mutation rate c/n. Each cell describes a rule for p subject to a minimum value of c and gives the number of exceptions (ex.) from this rule where applicable.

		$(1+1)$ EA	Uniform	1-point
OneMax	uniform	$p < 10^{-3}$		
	1-point	$p < 10^{-3}$	$p < 10^{-3}$ for $c \geq 0.4$	
	2-point	$p < 10^{-3}$	$p < 10^{-3}$ for $c \geq 0.3$	$p > 10^{-3}$ (11 ex.)
Royal road	uniform	$p < 10^{-3}$ for $c \geq 0.6$		
	1-point	$p < 10^{-3}$ for $c \geq 0.6$	$p < 10^{-3}$ for $c \geq 0.8$ (1 ex.)	
	2-point	$p < 10^{-3}$ for $c \geq 0.6$	$p < 10^{-3}$ for $c \geq 1.4$ (5 ex.)	$p > 10^{-3}$ (6 ex.)
Random polynomial	uniform	$p < 10^{-3}$ (1 ex.)		
	1-point	$p < 10^{-3}$ for $c \geq 0.3$ (3 ex.)	$p > 10^{-3}$ (13 ex.)	
	2-point	$p < 10^{-3}$ for $c \geq 0.4$ (1 ex.)	$p > 10^{-3}$ (6 ex.)	$p > 10^{-3}$ (6 ex.)

is a noticeable gap between the curves, there is a statistically significant difference on a significance level of .001. The outcome of Mann–Whitney U tests is summarized in Table 1.

For very small mutation rates c/n the tests were not significant. For mutation rates no less than $0.6/n$ all differences between the $(1+1)$ EA and all greedy $(2+1)$ GAs were statistically significant, apart from a few exceptions on random polynomials. For OneMax the difference between uniform crossover and k-point crossover was significant for $c \geq 0.4$. For royal roads the majority of such comparisons showed statistical significance, with a number of exceptions. However, for random polynomials the majority of comparisons were not statistically significant. Most comparisons between 1-point and 2-point crossover did not show statistical significance.

These findings give strong evidence that the insights drawn from the analysis on OneMax transfer to broader classes of functions where building blocks need to be assembled.

6.2 Linear Functions

Another interesting question is how far the theoretical analyses in this work extend to cases where building blocks have different weights. The simplest such case is the class of linear functions, defined as

$$f(x) = \sum_{i=1}^{n} w_i x_i,$$

where $w_i > 0$ are positive real-valued weights.

Doerr et al. (2013a) provided empirical evidence that their (1+(λ, λ)) EA is faster than the $(1+1)$ EA on linear functions with weights drawn uniformly at random from $[1, 2]$.

It is an open question whether this also holds for more common GAs, that is, those implementing Algorithm 1. Experiments in Doerr et al. (2013a) on the greedy $(2+1)$ GA found that on random linear functions "no advantage of the $(2+1)$ GA over the $(1+1)$ EA is visible." We provide an explanation for this observation and reveal why the $(2+1)$ GA is not well suited for weighted building blocks, whereas other GAs might be.

The reason the $(2+1)$ GA behaves like the $(1+1)$ EA in the presence of weights is that in case the current population of the $(2+1)$ GA contains two members with different fitness, the $(2+1)$ GA ignores the inferior one. So it behaves as if the population only contained the fitter individual. Since the $(2+1)$ GA will select the fitter individual twice for crossover, followed by mutation, it essentially just mutates the fitter individual. This behavior of the $(2+1)$ GA then equals that of a $(1+1)$ EA working on the fitter individual.

The $(2+1)$ GA is more efficient than the $(1+1)$ EA on OneMax (and other building-block functions where all building blocks are equally important) as it can easily generate and store individuals with equal fitness in the population and recombine their different building blocks. However, in the presence of weights, chances of creating individuals of equal fitness might be very slim, and then the $(2+1)$ GA behaves like the $(1+1)$ EA.

Theorem 18: *As long as the population of the $(2+1)$ GA does not contain two different individuals with the same fitness, the $(2+1)$ GA is equivalent to the $(1+1)$ EA.*

On functions where all search points have different fitness values, the $(2+1)$ GA is equivalent to the $(1+1)$ EA. This includes linear functions with extreme weights like

$$\mathit{BinVal}(x) := \sum_{i=1}^{n} 2^{n-i} x_i$$

and, more generally, functions where $w^{(i)} > \sum_{j=i+1}^{n} w^{(j)}$ for all $1 \leq i \leq n$, where $w^{(i)}$ denotes the ith largest weight. It also includes, almost surely, random linear functions with weights being drawn from some real-valued interval $[a, b]$ with $a < b$.

Proof: The first two statements have been established in the preceding discussion.

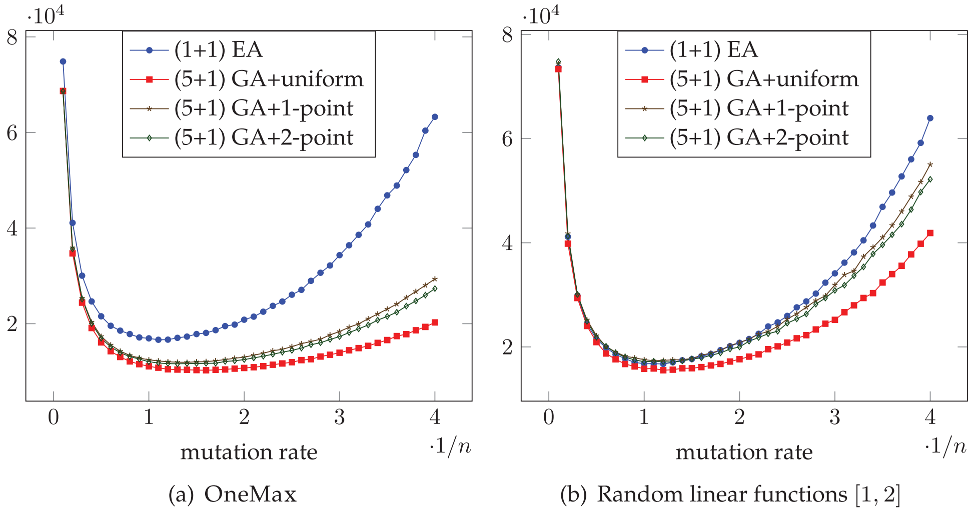
For functions where $w^{(i)} > \sum_{j=i+1}^{n} w^{(j)}$ for all $1 \leq i \leq n$, all search points with a 1 on the bit of weight $w^{(i)}$ have a higher fitness than all search points where this bit is 0, provided that all bits with larger weights are being fixed. It follows inductively that all search points have different fitness values.

For random linear functions, consider the function being constructed sequentially by adding new bits with randomly drawn weights. Assume that after adding i bits, all 2^i bit patterns have different fitness values. This is trivially true for 0 bits. When adding a new bit $i+1$, a fitness value can only be duplicated with these $i+1$ bits if the $i+1$-st weight is equal to any selection of weights from the first i bits. Since there are at most 2^i selections, which is finite, the $i+1$-st weight will almost surely be different from all of these. The statement then follows by induction. □

In a sense, the $(2+1)$ GA is not able to benefit from crossover in the settings from Theorem 18 since its greedy parent selection suppresses diversity in the population.

So, in order for a GA to benefit from crossover, the population needs to be able to maintain and select individuals with different building blocks and slightly different fitness values for long enough so that crossover has a good chance of combining those building blocks. The (1+(λ, λ)) EA (Doerr et al., 2013a) achieves this using a cleverly designed two-stage offspring creation process: mutation first creates diversity and the best among λ mutants is retained and recombined with its parent λ times. However, this does not explain why crossover is beneficial in common GA designs.

A promising common GA design does not need to be sophisticated. Figure 4 shows that a simple $(5+1)$ GA with uniform parent selection performs significantly better than



(a) OneMax

(b) Random linear functions $[1, 2]$

Figure 4: Average optimization times over 1,000 runs for $(1+1)$ EA and a $(5+1)$ GA with uniform parent selection and various crossover operators on functions with $n =$ 1,000 bits: OneMax and random linear functions with weights drawn independently, uniformly at random from [1, 2], and anew for each run.

the $(1+1)$ EA (and hence the greedy $(2+1)$ GA). The benefit of crossover is smaller than that on OneMax, but the main qualitative observations are the same: the average optimization time is smaller with crossover, and mutation rates slightly larger than $1/n$ further improve performance.

One-sided Mann–Whitney U tests on a significance level of .001 showed that the $(5+1)$ GA with uniform crossover was significantly faster than the $(1+1)$ EA on random linear functions, for mutation rates no less than $0.6/n$. Both k-point crossovers gave mixed results: they were slower than the $(1+1)$ EA for low mutation rates ($0.4 \le c \le 1.2$, except for $c = 0.6$, for 1-point crossover and $0.9 \le c \le 1.1$ for 2-point crossover) but faster for high mutation rates ($c = 2.3$ and $c \ge 2.6$ for 1-point crossover, $c = 2.0$ and $c \ge 2.2$ for 2-point crossover).

This shows that uniform crossover can speed up building block assembly for weighted building blocks, albeit not for all $(\mu + \lambda)$ GAs and in particular not for the greedy $(2+1)$ GA. Proving this rigorously for random or arbitrary linear functions remains a challenging open problem, and so is identifying characteristics of $(\mu + \lambda)$ GAs for which crossover is beneficial in these cases.

7 Conclusions and Future Work

We have demonstrated rigorously and intuitively that crossover can speed up building block assembly on OneMax, with evidence that the same holds for a broad class of functions. The basic insight is that crossover can capitalize on mutations that have both beneficial and disruptive effects on building blocks: mutants can be stored in the population and crossover is able to repair the detrimental effects of mutation in a later generation. This effect makes every $(\mu + \lambda)$ GA with cut selection and moderate population sizes twice as fast as every mutation-based EA on OneMax. In other words, adding crossover to any such $(\mu + \lambda)$ EA halves the expected optimization time (up to small-order terms). This applies to uniform crossover and to k-point crossover, for arbitrary values of k.

Furthermore, we have demonstrated how to analyze parent and offspring populations as in $(\mu+\lambda)$ EA s and $(\mu+\lambda)$ GAs. As long as both μ and λ are moderate, so that exploitation is not slowed down, we obtained essentially the same results for arbitrary $(\mu+\lambda)$ GAs as for the simple greedy $(2+1)$ GA analyzed in Sudholt (2012). This work provides novel techniques for the analysis of $(\mu$+$\lambda)$-type algorithms, including Lemmas 2 and 3, which may prove useful in further studies of EAs.

Another intriguing conclusion following naturally from our analysis is that the optimal mutation rate for GAs such as the greedy $(2+1)$ GA changes from $1/n$ to $(1+\sqrt{5})/2 \cdot 1/n \approx 1.618/n$ when using uniform crossover. This is simply because neutral mutations and hence multibit mutations become more useful. Experiments are in perfect accordance with the theoretical results for OneMax. For other functions like royal roads and random polynomials, they indicate that the performance differences also hold in a much more general sense. We have empirical evidence that this might also extend to linear functions, and weighted building blocks in general, albeit this does not apply to the greedy $(2+1)$ GA. The discussion in Section 6.2 showed that the population must be able to store individuals with different building blocks for long enough so that crossover can combine them, even though some individuals might have inferior fitness values and be subject to replacement.

Our results give novel, intuitive, and rigorous answers to a question that has been discussed controversially for decades.

There are plenty of avenues for future work. We would like to extend the theoretical analysis of $(\mu+\lambda)$ GAs to royal road functions and monotone polynomials. Also investigating weighted building blocks, as in linear functions, is an interesting and challenging topic for future work.

Our $(\mu+\lambda)$ GAs benefit from crossover and an increased mutation rate because cut selection removes offspring with inferior fitness. As such, cut selection counteracts disruptive effects of crossover and an increase of the mutation rate. The situation is entirely different in generational GAs, where Ochoa, Harvey, and Buxton (1999) reported that introducing crossover can *decrease* the optimal mutation rate. Future work could deal with complementing these different settings and investigating the balance between selection pressure for replacement selection and the optimal mutation rate.

Acknowledgments

This work was partially supported by EPSRC grant EP/D052785/1 while the author was a member of CERCIA, University of Birmingham, UK. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 618091 (SAGE). The author thanks the reviewers for their detailed and constructive comments that helped to improve the manuscript and Nick Barton, Lee Altenberg, and Joshua Knowles for insightful discussions at Dagstuhl seminar 15211 “Theory of Evolutionary Algorithms” as well as references to early related work.

References

Abramowitz, M., and Stegun, I. A. (Eds.) (1964). *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. New York: Dover.

Arora, S., Rabani, Y., and Vazirani, U. V. (1994). Simulating quadratic dynamical systems is PSPACE-complete. In *Proceedings of the Symposium on the Theory of Computing*, pp. 459–467.

Auger, A., and Doerr, B. (Eds.) (2011). *Theory of randomized search heuristics: Foundations and recent developments*. Singapore: World Scientific.

Badkobeh, G., Lehre, P. K., and Sudholt, D. (2014). Unbiased black-box complexity of parallel search. In *Parallel Problem Solving from Nature*, pp. 892–901.

Barton, N. H., and Charlesworth, B. (1998). Why sex and recombination? *Science*, 281(5385): 1986–1990.

Bell, G. (1982). *The masterpiece of nature: The evolution and genetics of sexuality*. Berkeley: Univ. of California Press.

Davis, L. (Ed.) (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.

De Jong, K. A. (2006). *Evolutionary computation: A unified approach*. Cambridge, MA: MIT Press.

De Jong, K. A., and Spears, W. M. (1992). A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 5(1): 1–26.

Dietzfelbinger, M., Naudts, B., Van Hoyweghen, C., and Wegener, I. (2003). The analysis of a recombinative hill-climber on H-IFF. *IEEE Transactions on Evolutionary Computation*, 7(5): 417–423.

Doerr, B., and Doerr, C. (2015a). Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1335–1342.

Doerr, B., and Doerr, C. (2015b). A tight runtime analysis of the (1+(λ, λ)) genetic algorithm on OneMax. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1423–1430.

Doerr, B., Doerr, C., and Ebel, F. (2013a). Lessons from the black-box: Fast crossover-based genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 781–788.

Doerr, B., Fouz, M., and Witt, C. (2011). Sharp bounds by probability-generating functions and variable drift. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 2083–2090.

Doerr, B., Happ, E., and Klein, C. (2012a). Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425:17–33.

Doerr, B., Johannsen, D., and Winzen, C. (2012b). Multiplicative drift analysis. *Algorithmica*, 64:673–697.

Doerr, B., and Künnemann, M. (2013). Royal road functions and the $(1+\lambda)$ evolutionary algorithm: Almost no speed-up from larger offspring populations. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 424–431.

Doerr, B., Sudholt, D., and Witt, C. (2013b). When do evolutionary algorithms optimize separable functions in parallel? In *Proceedings of the Workshop on Foundations of Genetic Algorithms*, pp. 51–64.

Fischer, S., and Wegener, I. (2005). The one-dimensional Ising model: Mutation versus recombination. *Theoretical Computer Science*, 344(2–3): 208–225.

Forrest, S., and Mitchell, M. (1993). Relative building block fitness and the building block hypotheses. In *Proceedings of the Workshop on Foundations of Genetic Algorithms*, pp. 198–226.

Jansen, T. (2013). *Analyzing evolutionary algorithms: The computer science perspective.* Berlin: Springer.

Jansen, T., DeJong, K. A., and Wegener, I. (2005). On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13:413–440.

Jansen, T., and Wegener, I. (2002). On the analysis of evolutionary algorithms: A proof that crossover really can help. *Algorithmica*, 34(1): 47–66.

Jansen, T., and Wegener, I. (2005). Real royal road functions: Where crossover provably is essential. *Discrete Applied Mathematics*, 149:111–125.

Jansen, T., and Zarges, C. (2011). Analysis of evolutionary algorithms: From computational complexity analysis to algorithm engineering. In *Proceedings of the Workshop on Foundations of Genetic Algorithms*, pp. 1–14.

Kötzing, T., Sudholt, D., and Theile, M. (2011). How crossover helps in pseudo-Boolean optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 989–996.

Lässig, J. (2009). Personal communication.

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

Lehre, P. K., and Witt, C. (2012). Black-box search by unbiased variation. *Algorithmica*, 64(4): 623–642.

Lehre, P. K., and Yao, X. (2011). Crossover can be constructive when computing unique input-output sequences. *Soft Computing*, 15(9): 1675–1687.

Livnat, A., Papadimitriou, C., Dushoff, J., and Feldman, M. W. (2008). A mixability theory for the role of sex in evolution. *Proceedings of the National Academy of Sciences*, 105(50): 19803–19808.

Livnat, A., Papadimitriou, C., Pippenger, N., and Feldman, M. W. (2010). Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences*, 107(4): 1452–1457.

Mitchell, M., Forrest, S., and Holland, J. H. (1992). The royal road function for genetic algorithms: Fitness landscapes and GA performance. In *Proceedings of the European Conference on Artificial Life*, pp. 245–254.

Mitchell, M., Holland, J. H., and Forrest, S. (1994). When will a genetic algorithm outperform hill climbing? In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, pp. 51–58.

Mitzenmacher, M., and Upfal, E. (2005). *Probability and computing*. Cambridge: Cambridge University Press.

Muller, H. J. (1932). Some genetic aspects of sex. *American Naturalist*, 66(703): 118–138.

Neumann, F., Oliveto, P. S., and Witt, C. (2009). Theoretical analysis of fitness-proportional selection: Landscapes and efficiency. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 835–842.

Neumann, F., and Theile, M. (2010). How crossover speeds up evolutionary algorithms for the multi-criteria all-pairs-shortest-path problem. In *Parallel Problem Solving from Nature*, pp. 667–676.

Neumann, F., and Witt, C. (2010). *Bioinspired computation in combinatorial optimization: Algorithms and their computational complexity*. Berlin: Springer.

Ochoa, G., Harvey, I., and Buxton, H. (1999). Error thresholds and their relation to optimal mutation rates. In D. Floreano, J.-D. Nicoud, and F. Mondada (Eds.), *Advances in artificial life*, pp. 54–63. Berlin: Springer.

Oliveto, P. S., and Witt, C. (2013). Improved runtime analysis of the simple genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1621–1628.

Oliveto, P. S., and Witt, C. (2014). On the runtime analysis of the simple genetic algorithm. *Theoretical Computer Science*, 545:2–19.

Paixão, T., Badkobeh, G., Barton, N., Corus, D., Dang, D.-C., Friedrich, T., Lehre, P. K., Sudholt, D., Sutton, A. M., and Trubenová, B. (2015). Toward a unifying framework for evolutionary processes. *Journal of Theoretical Biology*, 383:28–43.

Paixao, T., Pérez Heredia, J., Sudholt, D., and Trubenova, B. (2015). First steps towards a runtime comparison of natural and artificial evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1455–1462.

Prügel-Bennett, A. (2010). Benefits of a population: Five mechanisms that advantage population-based algorithms. *IEEE Transactions on Evolutionary Computation*, 14(4): 500–517.

Prügel-Bennett, A., and Rogers, A. (2001). Modelling genetic algorithm dynamics. In L. Kallel, B. Naudts, and A. Rogers (Eds.), *Theoretical aspects of evolutionary computing*, pp. 59–85. Berlin: Springer.

Qian, C., Yu, Y., and Zhou, Z.-H. (2013). An analysis on recombination in multi-objective evolutionary optimization. *Artificial Intelligence*, 204:99–119.

Rabani, Y., Rabinovich, Y., and Sinclair, A. (1998). A computational view of population genetics. *Random Structures and Algorithms*, 12(4): 313–334.

Rowe, J. E. (2015). Genetic algorithms. In J. Kacprzyk and W. Pedrycz (Eds.), *Handbook of computational intelligence*, pp. 825–844. Berlin: Springer.

Rowe, J. E., Vose, M. D., and Wright, A. H. (2002). Group properties of crossover and mutation. *Evolutionary Computation*, 10(2): 151–184.

Sastry, K., Goldberg, D., and Kendall, G. (2005). Genetic algorithms. In E. K. Burke and G. Kendall (Eds.), *Search methodologies*, pp. 97–125. New York: Springer.

Shapiro, J. L. (2001). Statistical mechanics theory of genetic algorithms. In L. Kallel, B. Naudts, and A. Rogers (Eds.), *Theoretical aspects of evolutionary computing*, pp. 87–108. Berlin: Springer.

Storch, T., and Wegener, I. (2004). Real royal road functions for constant population size. *Theoretical Computer Science*, 320:123–134.

Sudholt, D. (2005). Crossover is provably essential for the Ising model on trees. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1161–1167.

Sudholt, D. (2009). The impact of parametrization in memetic evolutionary algorithms. *Theoretical Computer Science*, 410(26): 2511–2528.

Sudholt, D. (2012). Crossover speeds up building-block assembly. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 689–696.

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

Sudholt, D., and Thyssen, C. (2012). Running time analysis of ant colony optimization for shortest path problems. *Journal of Discrete Algorithms*, 10:165–180.

Vose, M. D. (1999). *The simple genetic algorithm: Foundations and theory*. Cambridge, MA: MIT Press.

Watson, R. A., and Jansen, T. (2007). A building-block royal road where crossover is provably essential. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1452–1459.

Wegener, I., and Witt, C. (2005). On the optimization of monotone polynomials by simple randomized search heuristics. *Combinatorics, Probability and Computing*, 14:225–247.

Weissman, D. B., and Barton, N. H. (2012). Limits to the rate of adaptive substitution in sexual populations. *PLoS Genetics*, 8(6): e1002740.

Weissman, D. B., Feldman, M. W., and Fisher, D. S. (2010). The rate of fitness-valley crossing in sexual populations. *Genetics*, 186(4): 1389–1410.

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

Witt, C. (2013). Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing*, 22:294–318.

Appendix

The appendix contains two technical remarks on Theorem 4 and proofs of lemmas omitted from the main part.

REMARK 1 (CONDITIONS FOR μ AND λ): The second statement of Theorem 4 requires $\mu, \lambda = o((\log n)/(\log\log n))$ in order to establish the upper bound in (3). This condition seems necessary, as for larger values of μ and λ, the inertia of a large population slows down exploitation, at least in the absence of crossover. Note not all EAs covered by Theorem 4 (after removing crossover) optimize OneMax in time $O(n \log n)$.

Witt (2006) showed that a $(\mu+1)$ EA with uniform parent selection has an expected optimization time of $\Omega(\mu n + n \log n)$ on OneMax. For $\mu = \omega(\log n)$, this lower bound is $\omega(n \log n)$. Jansen, DeJong, and Wegener (2005) showed that a (1+λ) EA needs time $\omega(n \log n)$ on OneMax if $\lambda = \omega((\log n)(\log\log n)/(\log\log\log n))$. Badkobeh, Lehre, and Sudholt (2014) showed that every black box algorithm creating λ offspring, using only standard bit mutation or other unary unbiased operators, needs time $\omega(n \log n)$ on OneMax for $\lambda = \omega((\log n)(\log\log n))$. This indicates that the threshold in the condition $\mu, \lambda = o((\log n)/(\log\log n))$ is tight up to polynomials of $\log\log n$.

REMARK 2 (CONDITIONS FOR p_c): Theorem 4 assumes $0 < p_c < 1$ constant, which reflects the most common choices in applications of EAs. The theorem can be extended toward smaller or larger values as follows. If $p_c = o(1)$, the upper bound on the time spent in cases $i.3$ increases, as it contains a factor of $1/p_c$. The other cases remain unaffected, and if $((\mu+\lambda)\log\mu)/p_c = o(\log n)$, we still get the upper bound from (3).

For high crossover probabilities, that is, $p_c = 1 - o(1)$ or $p_c = 1$, only cases $i.1$ need to be revisited. The time in those cases was derived from Lemma 2, which can be adapted as follows: the probability for increasing the number of fit individuals is at least

$$p_c \cdot (1-p)^n \cdot \frac{(\text{number of fit individuals in population})^2}{2\mu^2},$$

as it suffices to select two fit individuals and generate an average or above-average number of 1-bits in the offspring, which happens with probability at least 1/2. The time bound from Lemma 2 then becomes

$$\frac{O(\mu^2 + \lambda \log \mu)}{(1-p)^n},$$

and the time bound in Theorem 4 becomes

$$\frac{\ln(n^2 p + n) + 1 + p}{p(1-p)^{n-1} \cdot (1+np)} + \frac{O(n(\mu^2 + \lambda \log \mu))}{(1-p)^n}.$$

For $p = c/n$, $c > 0$ constant, and $\mu, \lambda = o(\sqrt{\log n})$, this also establishes the upper bound from (3).

Proof of Lemma 12: If the current population has a best individual of fitness $j < i$, by Lemma 2 after an expected number of $O((\mu + \lambda)\log\mu) = O(1)$ function evaluations, all individuals will have fitness at least j. Then one offspring creation results in an improvement if no crossover is being used, and mutation flips exactly one out of $n-j$ 0-bits. The probability for this event is

$$(1 - p_c) \cdot (n - j)p(1 - p)^{n-1} \geq \gamma \cdot \frac{n - j}{n}$$

for some constant $\gamma > 0$, due to our conditions for p and p_c.

Using Lemma 3, the expected time until a fitness level $i \geq n - n/\log n$ is reached for the first time is therefore at most

$$\begin{aligned}\sum_{j=0}^{n-(n/\log n)-1} \left(O(1) + \lambda + \frac{n}{\gamma(n-j)}\right) &= O(n) + \frac{n}{\gamma} \cdot \sum_{j=(n/\log n)+1}^{n} \frac{1}{j} \\ &\leq O(n) + \frac{n}{\gamma} \cdot \int_{j=n/\log n}^{n} \frac{1}{j}\,\mathrm{d}j \\ &= O(n) + \frac{n}{\gamma} \cdot \big(\ln n - \ln(n/\log n)\big) \\ &= O(n) + \frac{n}{\gamma} \cdot \ln(\log n) \\ &= o(n \log n). \qquad \square\end{aligned}$$

Proof of Lemma 14: In order to create a different search point on the same fitness level, there must be some integer $\ell \in \{1, \dots, \min\{i, n-i\}\}$ such that ℓ 1-bits flip to 0 and ℓ 0-bits flip to 1. This is a necessary and sufficient condition, so

$$p^{(i)} = \sum_{\ell=1}^{\min\{i,n-i\}} \binom{i}{\ell}\binom{n-i}{\ell} p^{2\ell}(1-p)^{n-2\ell}. \qquad (12)$$

The case $\ell = 1$ yields the claimed lower bound. For the upper bound we bound the above term using $\binom{n}{k} \leq n^k/(k!)$ to bound both binomial coefficients:

$$\begin{aligned}p^{(i)} &\leq (1-p)^n \sum_{\ell=1}^{\min\{i,n-i\}} \frac{i^\ell (n-i)^\ell}{\ell!\ell!} \cdot p^{2\ell}(1-p)^{-2\ell} \\ &\leq (1-p)^n \sum_{\ell=1}^{\infty} \left(\frac{i(n-i)p^2}{(1-p)^2}\right)^\ell = (1-p)^n \frac{\frac{i(n-i)p^2}{(1-p)^2}}{1 - \frac{i(n-i)p^2}{(1-p)^2}},\end{aligned}$$

where in the last step we used $\frac{i(n-i)p^2}{(1-p)^2} \leq 1/2 < 1$, implying that the series converges. Applying $\frac{1}{1-x} = 1 + \frac{x}{1-x} \leq 1 + 2x$ for $x \leq 1/2$ to $x := \frac{i(n-i)p^2}{(1-p)^2} \leq 1/2$ in the above formula yields

$$(1-p)^n \cdot \frac{i(n-i)p^2}{(1-p)^2} \cdot \left(1 + \frac{2i(n-i)p^2}{(1-p)^2}\right)$$

and hence the claimed upper bound.

The second statement follows from the upper bound and the fact that the offspring has Hamming distance 2 in the case $\ell = 1$, that is, with probability $i(n-i)p^2(1-p)^{n-2}$.

Proof of Lemma 15: A search point with i ones is created from a parent with $i - d < i$ ones if there is a value ℓ such that $d + \ell$ 0-bits flip to 1 and ℓ 1-bits flip to 0. The sought probability therefore is

$$\begin{aligned}
&\max_{d \geq 1} \left(\sum_{\ell=0}^{\infty} \binom{n-i+d}{d+\ell} \binom{i-d}{\ell} p^{d+2\ell} (1-p)^{n-d-2\ell} \right) \\
\leq\ &\max_{d \geq 1} \left(\sum_{\ell=0}^{\infty} \frac{(n-i+d)^{d+\ell}}{(d+\ell)!} \cdot \frac{(i-d)^{\ell}}{\ell!} \cdot p^{d+2\ell} \right) \\
=\ &\max_{d \geq 1} \left(\sum_{\ell=0}^{\infty} \frac{(p(n-i+d))^{d}}{(d+\ell)!} \cdot \frac{(p^2 \cdot (i-d)(n-i+d))^{\ell}}{\ell!} \right) \\
\leq\ &\max_{d \geq 1} \left(\frac{(p(n-i+d))^{d}}{d!} \right) \cdot \sum_{\ell=0}^{\infty} \frac{((pn)^2/4)^{\ell}}{\ell!} \\
=\ &\max_{d \geq 1} \left(\frac{(p(n-i+d))^{d}}{d!} \right) \cdot e^{(pn)^2/4}.
\end{aligned}$$

Using $1/(d!) \leq (e/d)^d$, we bound the max term as

$$\begin{aligned}
\max_{d \geq 1} \left(\frac{(p(n-i+d))^{d}}{d!} \right) &\leq \max_{d \geq 1} \left(\frac{(ep(n-i+d))}{d} \right)^{d} \\
&\leq \max_{d \geq 1} \left(ep(n-i+1) \right)^{d}.
\end{aligned}$$

Now, if $ep(n-i+1) \leq 1$, the maximum is attained for $d = 1$, in which case we got a probability bound of $ep(n-i+1) \cdot e^{(pn)^2/4}$ as claimed. If $ep(n-i+1) > 1$, we trivially bound the sought probability by

$$1 < ep(n-i+1) \leq ep(n-i+1) \cdot e^{(pn)^2/4}. \qquad \square$$