Domino convergence: Why one should hill-climb on linear functions

Witt, Carsten

Published in:
Proceedings of the 2018 Genetic and Evolutionary Computation Conference

Link to article, DOI:
10.1145/3205455.3205581

Publication date:
2018

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Witt, C. (2018). Domino convergence: Why one should hill-climb on linear functions. In *Proceedings of the 2018 Genetic and Evolutionary Computation Conference* (pp. 1539-1546). Association for Computing Machinery. https://doi.org/10.1145/3205455.3205581

Domino Convergence: Why One Should Hill-Climb on Linear Functions

Carsten Witt
DTU Compute, Technical University of Denmark
2800 Kgs. Lyngby, Denmark

ABSTRACT

In the theory community of evolutionary computation, linear pseudo-boolean functions are often regarded as easy problems since all of them can be optimized in expected time $O(n \log n)$ by simple unbiased algorithms. However, results from genetic algorithms and estimation-of-distribution algorithms indicate that these algorithms treat different linear functions differently. More precisely, an effect called "domino convergence" is described in the literature, which means that bits of large weight in the linear function are optimized earlier than bits of low weight. Hence, different linear functions may lead to rather different expected optimization times.

The present paper conducts a study of domino convergence. By rigorous runtime analyses, it is shown that domino convergence is mostly a consequence of the crossover underlying genetic algorithms and EDAs. Here a performance gap of order $\Omega(n/\log n)$ between different linear functions is proved. In simple mutation-only EAs the effect of domino convergence is much less pronounced, with the typical performance gap being logarithmic in the population size. The effect disappears when population size 1 is used and the algorithm is reduced to hillclimbing. Different selection mechanisms, including cut and tournament selection are investigated and their impact on domino convergence is analyzed.

CCS CONCEPTS

• **Theory of computation → Optimization with randomized search heuristics**;

KEYWORDS

Theory, Estimation of Distribution Algorithms, Genetic Algorithms, Runtime Analysis

ACM Reference Format:
Carsten Witt. 2018. Domino Convergence: Why One Should Hill-Climb on Linear Functions. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205581

ACM ISBN 978-1-4503-5618-3/18/07…$15.00
https://doi.org/10.1145/3205455.3205581

1 INTRODUCTION

The optimization of linear pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$, i. e., functions that can be written as $f(x_1,\dots,x_n) = w_1x_1 + \dots + w_nx_n$ for coefficients (weights) $w_i \in \mathbb{R}$, is an intensively studied problem in the research community dealing with runtime analysis of evolutionary algorithms (e. g. [17] and references therein). This class of functions includes famous benchmark functions like $\text{OneMax}(x_1,\dots,x_n) = x_1 + \dots + x_n$ that are very well understood due to their simple structure but also somewhat more involved examples like $\text{BinVal}(x_1,\dots,x_n) = \sum_{i=1}^{n} 2^{n-i}x_i$ where the weights differ heavily and the weight at position i even outweighs all smaller weights together. Despite these structural differences within the class of linear functions, all of them are usually considered easy problems in the runtime analysis community. It has been known since 2002 [5] that a simple (1+1) EA, a mutation-based hillclimber, optimizes any linear function in expected time $\Theta(n \log n)$. More recently [17], it was proven that the (1+1) EA optimizes any linear function in expected time $en \ln n \pm O(n)$, i. e., the runtime behavior is essentially the same on all linear functions, up to lower-order terms. Hence, one can have the impression that the optimization of BinVal proceeds in a very similar way to the one of OneMax.

This impression is correct with respect to the (1+1) EA, although a proof of the time bound $en \ln n \pm O(n)$ cited above is highly challenging for linear functions whose weights differ heavily (like in BinVal). A hillclimber like the (1+1) EA may exchange many correct light-weight bits for a heavy-weight bit and even a step mutating the string $(0,1,\dots,1)$ into $(1,0,\dots,0)$ would be accepted while maximizing BinVal since it leads to an increase in fitness despite the huge increase in Hamming distance to the optimum. Runtime analysis reveals that such extreme steps are rare enough in order not to impede the progress to the optimum too frequently. For OneMax, which is a function of unitation, the proof idea is simpler. The number of one-bits in the current search point is monotone increasing over time. The general runtime bound $en \ln n \pm O(n)$ intuitively means that steps trading low-weight for heavy-weight bits do not influence the optimization behavior significantly such that the optimization process behaves similar to OneMax on all linear functions. In fact, only steps flipping more than one bit may lead to different behavior than on OneMax. The randomized local search heuristic (RLS), an even simpler hillclimber that flips only one uniformly selected bit per iteration, only selects new search points based on that their Hamming distance to the optimum has increased or decreased – since this is equivalent to an increase or decrease of fitness if only one bit is flipped. Hence, RLS does not really see the weights of the linear function and treats all of them in the same way. In fact, linear functions can be optimized in time $O(n)$ by local search without replacement, but we exclude such local search operators from this study since we do not consider them typical for the basic evolutionary algorithms investigated here.

Researchers stemming from the community of genetic algorithms (GAs) may dispute the claim that all linear functions are equally difficult to optimize. Already in 1992, Rudnick [12] pointed out that GAs where crossover is the only search operator exhibit a widely different behavior on different linear functions. He also coined the term *domino convergence* to describe that heavy-weight bits (called more salient bits) are earlier optimized by the GA in the sense that the fraction of individuals having the correct bit entry raises fastest for the heavy bits and more slowly for the light bits. In fact, in extreme cases it is necessary for all heavier bits to converge before the light bits can converge, in analogy to a falling row of domino stones.

Domino convergence was also considered from a theoretical angle by Thierens et al. [15] and Lobo et al. [10]. Increasingly refined models of the convergence behavior of GAs without mutation on both BinVal (at that time called BinInt) and OneMax were proposed and predictions about the runtime of the GAs were given based on the model. Roughly speaking and adapting notation to standards in runtime analysis, a runtime behavior of $O(\mu n)$, with μ being the population size, was predicted w. r. t. BinVal, whereas $O(\mu\sqrt{n})$ was obtained w. r. t. OneMax. Both bounds require a minimum feasible value for μ to counteract so-called genetic drift. If μ is too small, the fraction of individuals having the correct value at some bit position may approach 0 and optimization becomes impossible. Also this minimum value depends on the problem since genetic drift should never occur within the whole optimization process, which takes longer for BinVal. For OneMax, the minimum population size predicted by the model is roughly $\sqrt{n}$, whereas for BinVal is $\Omega(n)$. Altogether, the best possible runtime is $\Theta(n)$ for OneMax and $\Theta(n^2)$ for BinVal. Hence, domino convergence is responsible for a factor of $\Theta(n)$ in the predicted runtime. Essentially, we will see that these claims can be proven rigorously for specific algorithms, except for $\log n$-factors that the models did not capture.

Domino convergence was also observed empirically in a wide range of algorithms [13] but rarely considered in a rigorous runtime analysis. An exception is the work by Droste [4] who studies the Compact Genetic Algorithm (cGA) and explains a very different runtime behavior on BinVal and OneMax. Roughly speaking, with the presumed best possible setting of its main parameter K, the cGA optimizes BinVal in time $O(nK) = O(n^{2+\epsilon})$ for small constant $\epsilon > 0$ and OneMax in time $O(\sqrt{n}K) = O(n^{1+\epsilon})$. Also, a lower bound $\Omega(nK)$ for BinVal is shown in [4]. Although strong indications are given, this lower bound neither shows that the bound $O(n^{2+\epsilon})$ is tight nor that the performance difference in the upper bounds w. r. t. OneMax and BinVal is real. Droste's paper (the conference version of which was called "Not all linear functions are equally difficult for the compact genetic algorithm") also gives bounds for all linear functions and bases the analysis on the fact that the cGA optimizes heavier weighted positions before lighter ones, in accordance with the behavior predicted by domino convergence.

Jägersküpper [8] proved that also the (1+1) EA will optimize the heavy positions of a given linear function no later than the light ones (in a stochastic sense); however, as mentioned above, the degree to which heavy positions are favored is so minor that the expected optimization time is at most by a lower-order term larger than on OneMax. We are only aware of one further runtime analysis paper that observes effects similar to domino convergence on linear functions: the analysis of a (1+λ) EA on linear functions [3]; however, the expected runtime on the two functions OneMax and BinVal differs only by a factor $o(\log\lambda)$, so also here domino convergence is much less pronounced than in the models of GAs mentioned above. Altogether, we do not have a clear characterization of algorithms and effects leading to pronounced domino convergence and different optimization times on linear functions.

The aim of this paper is to give a rigorous view on domino convergence and the reasons for its appearance, using runtime analyses as our main tool. We consider the cGA and two algorithms derived from it to understand which of the underlying operators and working principles is responsible for domino convergence. In this sequence of algorithms, we move from crossover-based algorithms to mutation-only EAs using tournament selection and discover that domino convergence almost disappears when removing crossover. Finally, the (μ+1) EA, which uses cut selection, is studied and it is shown that the effect of domino convergence is also very limited here, if present at all. Along the way, existing runtime results are revisited and partly improved, including a proof that Droste's upper bound for BinVal is almost tight. Moreover, important techniques for the analysis of EDAs, GAs and EAs are presented, including tools to bound the effect of genetic drift in a rigorous way.

After having studied the four algorithms, our conclusion is that domino convergence is likely to occur in algorithms that maintain very diverse populations. Then a comparison of two individuals with respect to their BinVal-value can lead to large changes in Hamming distance to the optimum, such that only the most significant different bit tends to stabilize first. This large Hamming distance is mostly due to the high variance of the crossover operator. In mutation-based EAs, individuals are often similar such that also progress in bits of low weight is likely to be maintained to some extent. To optimize BinVal in expected time $O(n \log n)$, it seems best to compare two search points of Hamming distance 1, which is exactly the behavior observed in RLS and most of the time in the (1+1) EA. Stating it in a different way, on BinVal the simple EAs behave efficiently since they have properties of hillclimbers and mostly select on Hamming distance and not on fitness. From this perspective, it is not longer natural to claim that all linear functions are optimized in expected time $O(n \log n)$: this property heavily depends on the philosophy behind the algorithm used.

This paper is structured as follows: in Section 2, we introduce the algorithms we are going to analyze and provide elementary definitions. In Section 3, we give detailed analyses of the different algorithms and establish on the one hand cases of heavy domino convergence and on the other hand cases where domino convergence is very limited, if it exists at all. We finish with some conclusions.

2 PRELIMINARIES

We consider different algorithms, the first three of which are the cGA and modifications thereof, while the last is a classical (μ+1) EA. The cGA, introduced by [7] and described in Algorithm 1 evolves a so-called frequency vector $(p_{1,t}, \dots, p_{n,t})$ over time to find the maximum of a pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$. At iteration t, two bit strings, so-called individuals, are sampled by letting frequency $p_{i,t}$ be the marginal probability of setting bit i to 1. Afterwards, the two individuals are ranked by fitness and a frequency is changed if the two individuals differ in the bit; more specifically, if the bit is 1 in the better offspring, the frequency is increased by $1/K$, otherwise decreased by $1/K$. Here K, the so-called population size, is a parameter to the algorithm that determines the strength of updates of frequencies and crucially determines its runtime [4, 6, 14].

Algorithm 1: Compact Genetic Algorithm (cGA)

```
$t \leftarrow 0$; $p_{1,t} \leftarrow p_{2,t} \leftarrow \cdots \leftarrow p_{n,t} \leftarrow 1/2$;
while termination criterion not met do
    for $i \in \{1, \dots, n\}$ do
        $x_i \leftarrow 1$ with prob. $p_{i,t}$, $x_i \leftarrow 0$ with prob. $1 - p_{i,t}$;
    for $i \in \{1, \dots, n\}$ do
        $y_i \leftarrow 1$ with prob. $p_{i,t}$, $y_i \leftarrow 0$ with prob. $1 - p_{i,t}$;
    if $f(x) < f(y)$ then swap $x$ and $y$;
    for $i \in \{1, \dots, n\}$ do
        if $x_i > y_i$ then $p_{i,t+1} \leftarrow p_{i,t} + 1/K$;
        if $x_i < y_i$ then $p_{i,t+1} \leftarrow p_{i,t} - 1/K$;
        if $x_i = y_i$ then $p_{i,t+1} \leftarrow p_{i,t}$;
    $t \leftarrow t + 1$;
```

The cGA was called "compact genetic algorithm" since it models the behavior of a specific GA without using an explicit population. More specifically, imagine a population consisting of K bitstrings of length n. A generation chooses two slots of the population, i. e., indices in an arbitrary but fixed enumeration of the K individuals, uniformly at random, creates two offspring from the whole population by applying gene pool crossover (see below) twice and independently, and puts the better of the two offspring into both slots. Essentially, the selection is a steady-state tournament selection with a tournament of size 2, but the two individuals chosen for the tournament are obtained by recombining the whole population through gene pool crossover. This operator sets each bit independently by choosing an individual from the population uniformly at random and taking its value at the considered bit. Hence, the probability of setting a bit to 1 is exactly the fraction j/K, where j is the number of individuals having a 1 at the bit, equivalent to the sampling procedure of the cGA. Note also that the tournament selection will update the frequencies in exactly the same way as the cGA. If the two chosen individuals coincide at the considered bit, the frequency does not change. If they differ, the frequency changes by $1/K$ according to the value of the better individual. Note that a frequency is allowed to reach the extremal values 0 or 1 to maintain the analogy with the GA just described. Theoretical analyses often limit the frequencies to a smaller interval like $[1/n, 1 - 1/n]$ [14].

The formulation of the cGA as an equivalent GA is somewhat artificial. A more common GA is obtained by keeping the population and the steady-state tournament selection but replacing the gene pool crossover by uniform crossover, see Algorithm 2. We call the resulting algorithm StSt $\binom{\mu}{2}$ GA as it chooses 2 individuals from the population of size μ uniformly at random. We use the common term μ from EAs for population sizes from now on.

Algorithm 2: StSt $\binom{\mu}{2}$ GA

```
$t \leftarrow 0$; $P_0$ consists of $\mu$ individuals chosen u. a. r.;
while termination criterion not met do
    Choose $x$ and $y$ from $P_t$ uniformly at random;
    Apply uniform crossover to $x$ and $y$, resulting in the two
      offspring $x'$ and $y'$;
    if $f(x') < f(y')$ then swap $x'$ and $y'$;
    Create $P_{t+1}$ from $P_t$ by replacing both $x$ and $y$ with $x'$;
    $t \leftarrow t + 1$;
```

It is interesting to study whether the StSt $\binom{\mu}{2}$ GA behaves similarly to the cGA, in particular w. r. t. domino convergence. To investigate whether domino convergence is caused by crossover, we further modify the StSt $\binom{\mu}{2}$ GA to use mutation, in this case standard bit mutation with mutation probability $1/n$, instead of crossover. Steady-state tournament selection is maintained, however, we will allow tournaments of general size $\lambda \geq 2$. The resulting algorithm is called StSt $\binom{\mu}{\lambda}$ EA, see Algorithm 3. As it will turn out, the algorithm for typical functions only works if $\lambda = \Omega(\log n)$. This has similar reasons as in (1,λ) EAs [11] since both algorithms tend to drift away from the optimum if λ is too small; in fact the (1,λ) EA is more or less a special case if $\lambda = \mu$, as detailed in Section 3.3. Basically, setting the parameter λ to $c \log n$ for some big enough constant c makes it very likely that at least one offspring copies the parent so that the algorithm usually behaves in an elitist way.

Algorithm 3: StSt $\binom{\mu}{\lambda}$ EA

```
$t \leftarrow 0$; $P_0$ consists of $\mu$ individuals chosen u. a. r.;
while termination criterion not met do
    Choose $x_1, \dots, x_\lambda$ from $P_t$ u. a. r. without replacement;
    for $i \in \{1, \dots, \lambda\}$ do
        Create $y_i$ by flipping each bit in $x_i$ independently
          with probability $1/n$;
    Choose $y^*$ from $\arg\max_{y \in \{y_1, \dots, y_\lambda\}} f(y)$ u. a. r.;
    Create $P_{t+1}$ from $P_t$ by replacing $x_1, \dots, x_\lambda$ all by $y^*$;
    $t \leftarrow t + 1$;
```

Finally, to analyze whether domino convergence can occur with other operators than tournament selection, we consider a mutation-only, steady state EA that uses cut selection: the (μ+1) EA, see Algorithm 4. This algorithm has been studied with methods from runtime analysis before [16]. It is also of particular interest since it includes the (1+1) EA as a special case, where domino convergence on linear functions provably does not matter (more precisely, can only influence the expected runtime by a lower-order term).

The *runtime* (synonymously, optimization time) is the number of function evaluations of the algorithm until an optimal solution is sampled. As the cGA, StSt $\binom{\mu}{2}$ GA and (μ+1) EA only evaluate 1 or 2 new solutions per iteration, their runtime is asymptotically identical to the number of iterations until sampling the optimum. With respect to StSt $\binom{\mu}{\lambda}$ EA, the runtime is by a factor of λ larger than the number of iterations.

In our theorems, we often say that an event occurs *with high probability* (w. h. p.). This means that the event occurs with probability at least $1 - n^{-c}$, where the reader may choose the constant c.

Algorithm 4: (μ+1) EA

```
$t \leftarrow 0$; $P_0$ consists of $\mu$ individuals chosen u. a. r.;
while termination criterion not met do
    Choose $x$ from $P_t$ u. a. r.;
    Create $y$ by flipping each bit in $x$ independently with probability $1/n$;
    Add $y$ to $P_t$, resulting in $P^*$ of size $\mu + 1$;
    Create $P_{t+1}$ by deleting from $P^*$ an individual of smallest $f$-value, breaking ties u. a. r.;
    $t \leftarrow t + 1$;
```

3 RESULTS

In this section, we consider the four algorithms introduced before and investigate whether they are prone to domino convergence, and, if so, to what extent.

3.1 Gene-Pool Crossover and Tournament Selection: the cGA

As mentioned in the introduction, Droste [4] has proved that the cGA may suffer from domino convergence resulting in a performance gap between OneMax and BinVal. We now make Droste's results more precise, e. g., by stating a general dependency on K in the upper bound, obtaining high-probability results, and improving the bound on BinVal by a factor of n^{ϵ}. At the same time, we answer an open question by Droste and give a shorter and more modern proof. However, in this context it must be noted that Droste's analysis is also more complex since he considers all linear functions.

Theorem 3.1. *Let $K \geq cn \log n$ for some sufficiently large constant $c > 0$ and $K = n^{O(1)}$. Then the optimization time of the cGA on BinVal is $O(Kn)$ with high probability.*

To show the theorem, we use the following structural result dealing with genetic drift. Similar analyses were proposed in [14] w. r. t. OneMax.

Lemma 3.2. *Consider cGA on BinVal and pick an arbitrary but fixed entry p of its frequency vector. The probability that p ever reaches a value of $1/3$ or less within a phase of t steps is at most $e^{-K^2/(36t)}$.*

Proof. The probability of increasing the frequency in an iteration of the cGA is at least as large as the probability of decreasing it. Namely, if the underlying bit value in the two sampled individuals is different and decides which of them has higher fitness (since it is the most significant different bit) then the individual having a 1 will have higher fitness. By ignoring the steps where the frequency does not change at all, we only overestimate the probability of reaching the minimum value within t steps. Moreover, we pessimistically assume that the probability of increasing is the same as of decreasing, resulting in a fair random walk.

The expected number of decreasing steps within t steps is $t/2$. To reach $1/3$ from the initial frequency $1/2$, it is necessary to have at least $K/6$ more decreasing than increasing steps within t steps. By the Hoeffding bound with random variables $X_i \in \{-1, 1\}$, the probability of this is at most $e^{-2(K/6)^2/(2t)} = e^{-K^2/(36t)}$. □

Proof of Theorem 3.1. We consider a phase of $t^* := c'Kn$ iterations for some sufficiently large constant $c' > 0$ and show that the optimum is found in the phase with high probability. To obtain a failure probability of $n^{-\gamma}$ for a given $\gamma > 0$, the constants c and c' and further constants appearing later in this proof have to be chosen appropriately.

The main idea of the proof is similar to [4]: we analyze the stochastic process of the potential $\Phi_t = \sum_{i=1}^{n}(1 - p_{i,t})$, which measures the accumulated distance of the frequencies from their optimal setting. As soon as the potential function has reached the value 0, the cGA will surely sample the optimum. In fact, this is already likely when the potential has become $O(1)$.

To show the claim, we work under the following assumption: within t iterations, no frequency drops to a value of less than $1/3$. According to Lemma 3.2 and applying a union bound over all n bits, this assumption holds with probability at least $1 - ne^{-K^2/36t} = 1 - ne^{-K^2/(36c'Kn)} = 1 - ne^{-K/(36c'n)}$, which, by plugging in our lower bound on K, is at least $1 - ne^{-cn\log n/(36c'n)} = 1 - e^{-\Omega((c/c')\log n)}$, choosing c large enough.

Next we analyze the one-step change $\Phi_t - \Phi_{t+1}$ and prove that its drift satisfies $E(\Phi_t - \Phi_{t+1} \mid \Phi_t) = \Omega(1/K)$ as long as Φ_t is at least a constant. Since the maximum Φ-value is n, this suggests the time bound $O(nK)$ in expectation by additive drift analysis. The details of the drift argument, including a high-probability statement, will be shown after deriving the drift.

Consider the two individuals x and y sampled in an iteration $t \leq t^*$ of the cGA based on the current frequency vector $\mathbf{p}_t$. With probability $2p_{i,t}(1 - p_{i,t}) \geq (2/3)(1 - p_{i,t})$ it happens that $x_i \neq y_i$. Here we use the assumption that $p_{i,t} \geq 1/3$ within the phase. The most significant bit sampled differently determines which individual has higher BinVal-value and the frequency of the considered bit will increase by $1/K$. The frequencies of the more significant bits will not change by definition, and the frequencies of the less significant bits are expected to remain the same since they do not influence the ranking of x and y. Hence, sampling x and y differently is sufficient for an expected decrease of the Φ-value by $1/K$. We obtain $E(\Phi_t - \Phi_{t+1} \mid \Phi_t) \geq \Pr(x \neq y)/K$.

The probability $\Pr(x \neq y)$ that x and y differ in a least one bit is

$$1 - \prod_{i=1}^{n}(1 - 2p_{i,t}(1 - p_{i,t})) \geq 1 - \left(1 - \frac{\sum_{i=1}^{n} 2p_{i,t}(1 - p_{i,t})}{n}\right)^n,$$

which is at least $1 - \left(1 - \frac{(2/3)\Phi_t}{n}\right)^n$, where the first inequality used that the arithmetic mean is at least the geometric mean.

The last bound is $\Omega(1)$ if $\Phi_t \geq 1$, where the 1 has been chosen for convenience and can be replaced by any other positive constant. Hence, conditioned on $\Phi_t \geq 1$, we obtain the drift $E(\Phi_t - \Phi_{t+1} \mid \Phi_t) = \Omega(1/K)$. If $\Phi_t \leq 1$, the probability of sampling the optimal all-ones string is at least $(1/3)(2/3) = 2/9$ since all frequencies are at least $1/3$ and the probability of sampling the all-ones string is minimized if as many frequencies as possible take their extremal values ($1/3$ and 1). The probability of not sampling the optimum within the next $c_2 \log n$, c_2 an arbitrary constant, iterations is $e^{-\Omega(c_2 \log n)}$, observing that the potential can increase by at most $(n/K)c_2 \log n = O(1)$ in this number of iterations.

We still have to show concentration of the time T until the potential is reduced from at most n to at most 1 under an additive drift of $\Omega(1/K)$. To apply tail bounds for additive drift [9], we have to bound the maximum change of Φ-value. By Hoeffding's inequality, for any $c_3 > 0$ it holds that $|\Phi_t - \Phi_{t+1}| \leq c_3\sqrt{n \log n}/K$ with probability $1 - e^{-\Omega(c_3 \log n)}$, pessimistically assuming the maximum sampling variance according to $p_{t,i} = 1/2$ for all i. By a union bound, this holds throughout the phase of $t^* = c'Kn = n^{O(1)}$ steps still with probability $1 - n^{-c_4}$ for any given constant $c_4 > 0$ if c_3 is chosen large enough. Under this assumption, Theorem 2 in [9] applied with (according to the notation of the theorem) $\epsilon = \Theta(1/K)$ and $c = \Theta(\sqrt{n \log n}/K)$ yields that

$$\Pr(T > t^*) \leq e^{-t^*\epsilon^2/8c^2} = e^{-(c'Kn)\Theta(1/K^2)/\Theta((n \log n)/K^2)},$$

which is $e^{-\Omega(n)}$ using our assumption $K = \Omega(n \log n)$. Hence, the total failure probability is clearly less than $n^{-\gamma}$. □

Also the lower bound $\Omega(Kn)$ follows.

Theorem 3.3. *Let $K \geq cn \log n$ for a suffciently large constant $c > 0$ and $K = n^{O(1)}$. Then the optimization time of the cGA on BinVal is $\Omega(Kn)$ with high probability and in expectation.*

Proof. We again consider the potential $\Phi_t = \sum_{i=1}^{n}(1 - p_{i,t})$. By the initialization procedure of the cGA, we have $\Phi_0 = n/2$. We note that as long as $\Phi_t \geq n/4$, the probability of sampling the all-ones string is exponentially unlikely. Namely, if $\Phi_t \geq n/4$, there must be at least $n/8$ bits of frequency of at most $7/8$. Otherwise, if there were more than $7n/8$ bits of frequency larger than $7/8$, the potential would be less than $n \cdot (1/8) + n/8 \cdot 1 = n/4$. The probability of sampling ones only at all the bits of frequency at most $7/8$ is at most $(7/8)^{n/8} = 2^{-\Omega(n)}$.

We claim that with high probability $t^* := c'Kn$ steps are not sufficient to lower the potential to less than $n/4$ if c' is a sufficiently small constant and c is chosen large enough. The theorem then follows from the claim since the probability of sampling the optimum within cKn steps is still $2^{-\Omega(K)} = n^{-\Omega(c)}$ by the union bound and our assumptions on K.

To show the claim, we analyze the one-step change $\Delta_t := \Phi_t - \Phi_{t+1}$ and derive that the drift satisfies $\mathrm{E}(\Delta_t \mid \Phi_t) \leq 1/K$. More precisely, we will prove that Δ_t is the sum of $k \leq n - 1$ independent random variables with support $[-1/K, 1/K]$ and zero mean, and a random variable that is stochastically dominated by $+1/K$. By applying Hoeffding's inequality on t^*n random variables in $[-1/K, 1/K]$, we bound the quantity $\sum_{i=0}^{t^*-1} \Delta_i - t^*/K$ and obtain that $\Pr(\Phi_0 - \Phi_{t^*} \geq 2t^*/K) \leq e^{-\Omega(t^*/n)} = e^{-\Omega(K)}$. Hence, the potential is not reduced to less than $n/4$ in t^* steps with probability $1 - e^{-\Omega(K)}$.

To see the claim about the distribution of Δ_t, consider the two individuals $x, y \in \{0,1\}^n$ sampled at iteration t. Let i be the leftmost (i. e., most signficant) bit where x and y differ; w. l. o. g., $x_i = 1$ and $y_i = 0$ so that $\text{BinVal}(x) > \text{BinVal}(y)$. Then $p_{j,t+1} = p_{j,t}$ for $j < i$ since the bits left of i coincide. Moreover, $p_{i,t+1} \leq p_{i,t} + 1/K$. Finally, for $j > i$, $p_{i,t} - p_{i,t+1}$ takes values in $\{-1/K, 0, 1/K\}$ with equal probability for the non-zero values since the outcome of these bits is independent of the ranking $\text{BinVal}(x) > \text{BinVal}(y)$. □

Theorem 3.1 demands that $K \geq cn \ln n$ for a runtime of $O(Kn)$ to hold, whereas Theorem 3.3 only shows a lower bound $\Omega(Kn)$ on the runtime. In principle, it could still be the case that the condition on K from the upper bound is way too strong and that the runtime bound $\Theta(Kn)$ could hold for values of K that are, e. g., $\Theta(\log n)$. An expected runtime of $\Theta(n \log n)$ has not been excluded yet – and would not contradict Droste's [4] results either. Therefore, we now show that the condition on K from Theorem 3.1 is almost tight since lower values lead to genetic drift and will irreversibly fix frequencies to 0. In fact, a statement in this vein is also demanded in the conclusions of [4].

Theorem 3.4. *The cGA will fail to optimize BinVal with probability $\Omega(1)$ if $K \leq \epsilon n$ for a sufficiently small constant $\epsilon > 0$.*

Proof. Consider a phase of $t := \gamma Kn$ iterations for some small enough constant $\gamma > 0$. Reusing the argument from the proof of Theorem 3.3, we have $\Phi_s \geq n/2 - \epsilon n$ for all $s \leq t$ with probability $1 - 2^{-\Omega(n)}$. We consider the $n' := n - \epsilon n$ most significant bits. By Lemma 3.2, every frequency from these bits stays above $1/3$ within $t = \gamma Kn$ steps with probability $1 - e^{-K^2/(36t)} \geq 1 - e^{-(\epsilon n)^2/(36\gamma\epsilon n^2)} = 1 - e^{-\epsilon/(36\gamma)}$. By choosing γ small enough, the failure probability is at most ϵ. Applying Chernoff bounds (which is possible since Lemma 3.2 estimates the processes belonging to different frequencies by independent random walks), we have at least $n' - 2\epsilon n'$ bits of frequency at least $1/3$ from the n' considered bits with probability $1 - 2^{-\Omega(n)}$. The other bits contribute at most 1 to Φ_s, $s \leq t$. Since $\Phi_s \geq n/2 - \epsilon n$, this implies at least $n'/3$ many of these must have a frequency of at most $2/3$ (for ϵ small enough).

We are in the situation that $\Omega(n)$ from the first n' frequencies are in the interval $[1/3, 2/3]$. Consider the two individuals sampled in such a situation. The probability that they coincide in the first n' bits is $2^{-\Omega(n)}$, and also with probability $1 - t2^{-\Omega(n)} = 1 - 2^{-\Omega(n)}$ it never happens in t iterations that the two offspring coincide in all n' bits. Hence, the $n - n' = \epsilon n$ least significant bits have never been relevant for the selection in the cGA and perform therefore unbiased random walks. These random walks have been analyzed in [14, Lemma 7]. Applying this lemma (with 0 so-called b-steps) twice, first to exclude that $5/6$ is exceeded and then, under this condition, to analyze the event of hitting the lowest possible value 0), we obtain for each of these random walks that it hits 0 within $\alpha K^2 \leq t$ iterations, where α can be chosen as a constant, with probability $\Omega(1)$. □

We think that the assumption from Theorem 3.4 can be relaxed to $K \leq \epsilon n \log n$. In any case, Theorems 3.3 and 3.4 together, we obtain that a parametrization of K is necessary that leads to an expected runtime of $\Omega(n^2)$.

Corollary 3.5. *The expected runtime of cGA on BinVal is $\Omega(n^2)$.*

For comparison, we present the runtime bound that holds for the cGA on OneMax and is basically known from the literature:

Theorem 3.6 (cf. [14]). *The expected optimization time of the cGA on OneMax with $K \geq c\sqrt{n} \log n$ for a sufficiently large $c > 0$ and $K = n^{O(1)}$ is $O(\sqrt{n}K)$ with high probability.*

The proof of this theorem is almost the same as in the paper [14]. However, as their variant imposes borders $\{1/n, 1 - 1/n\}$ on the frequencies, we cannot get finite expected time, in contrast to [14]. As their proof uses that no frequency hits the lower border with high probability, it can also be used for our variant of the cGA.

The runtime in Theorem 3.6 is $O(n \log n)$ for the smallest possible choice of K. This is in sharp contrast to the $\Omega(n^2)$ bound derived above and shows that domino convergence makes cGA lose a factor $\Omega(n/\log n)$ in runtime when optimizing BinVal instead of OneMax.

3.2 Uniform Crossover and Tournament Selection

We now turn to the StSt $\binom{\mu}{2}$ GA, which is a real GA and uses uniform crossover instead of the gene-pool crossover implicit in the cGA. As a consequence, even though we still can define frequency values for every bit, corresponding to the fraction of individuals having a one, the bit values of selected individuals are not obtained by sampling independently according to the frequency vector. In fact, the bit values are heavily dependent, at least in the early phases of optimization. Despite these differences, we will see that the StSt $\binom{\mu}{2}$ GA behaves essentially in the same way as the cGA on BinVal and OneMax. In particular, under the assumption that an upper bound is tight, strong domino convergence occurs on BinVal.

As a preparation, we show a lemma about the effects of the uniform crossover on frequencies. Essentially, every frequency shows the same stochastic process as in the cGA, but the processes are not necessarily independent, not even for constant fitness functions.

Lemma 3.7. *For the StSt $\binom{\mu}{2}$ GA on BinVal, consider an arbitrary but fixed bit and let X_t, $t \geq 0$, denote its frequency value at iteration t. Then μX_t stochastically dominates a walk on $\{0, 1, \ldots, \mu - 1, \mu\}$ with transition probabilities $\Pr(X_{t+1} = i + 1 \mid X_t = i) = (i/\mu)(1 - i/\mu)$, $\Pr(X_{t+1} = i - 1 \mid X_t = i) = (i/\mu)(1 - i/\mu)$, $\Pr(X_{t+1} = i \mid X_t = i) = 1 - 2(i/\mu)(1 - i/\mu)$ for $i \in \{0, \ldots, \mu\}$. The random walks w. r. t. different bits are not necessarily independent.*

The probability that the X_t-value becomes $1/3$ or less within s steps is at most $e^{-\mu^2/(36s)}$.

Proof. We denote the index of the considered bit by k. If there are i individuals having a one at position k, then uniform selection will select an individual with a one at bit k with probability $p = i/\mu$. Since the two individuals used for tournament selection (before applying crossover) are chosen independently, the probability that bit k coincides in the tournament equals $1-2p(1-p)$. Since crossover will not have any effect in this case, this already proves the claimed probability of the event $X_{t+1} = X_t$.

The event that the first chosen individual has a one at bit k and the second individual a zero has probability $p(1 - p)$, as does the opposite event of a zero in the first and a one in the second individual. Note that with probability $1/2$ the entries at position k are swapped by crossover. Still, after crossover the two offspring satisfy these two cases each with the same probability $p(1 - p)$. The crucial observation is that due to crossover, the value of the two selected individuals are swapped with probability $1/2$, and swapping occurs independently of all other bit values.

Hence, if the ranking of the two offspring does not depend on bit k then the selected offspring will have a one with probability exactly $p(1 - p)$. If it depends on bit k then the offspring with a one at bit k has a higher probability of being selected due to the structure of BinVal, so $\Pr(X_{t+1} = i + 1 \mid X_t = i) \geq p(1 - p)$ in any case. This proves the first statement of the lemma.

Since a frequency value in the StSt $\binom{\mu}{2}$ GA and cGA dominates the same fair random walk, the proof of Lemma 3.2 carries over to StSt $\binom{\mu}{2}$ GA, which proves the second statement of the lemma. □

We now show an upper bound similar to Theorem 3.1.

Theorem 3.8. *Let $\mu \geq cn \log^2 n$ for some sufficiently large constant $c > 0$ and $\mu = n^{O(1)}$. Then the runtime of the StSt $\binom{\mu}{2}$ GA on BinVal is $O(\mu n \log \mu)$ w. h. p.*

Proof. In the same way as in the proof of Theorem 3.1, we show that no frequency drops below $1/3$ within a time span of $c'\mu n \log \mu$ iterations for some sufficiently large constant $c' > 0$. Instead of Lemma 3.2, Lemma 3.7 is used. The fact that the random walks are not independent does not harm since a union bound is applied.

Next we consider the process in phases, where phase i ends when the population has converged for bit i, i. e., the corresponding frequency has reached 1. Note that some phases may be empty and all search points in phase i must have the $i - 1$ most significant bits set to i. We investigate the frequency p_i and prove that the expected time to reach its maximum 1 is bounded by $O(\mu \log \mu)$. Then the lemma follows by summing over all phases.

The expected time until p_i has reached is maximum can be bounded as follows: if the two individuals chosen for tournament selection differ in bit $n - i$, then p_i increases and stays unchanged otherwise. The probability of differing in bit i is at least $p_i(1-p_i) \geq (1 - p_i)/3$. Since $1 - p_i \geq 1/\mu$, multiplicative drift analysis of the process $X_i := (1-p_i)/3$ yields the expected time $O(\mu \log \mu)$ to reach $X_i = 0$, equivalent to $p_i = 1$. Using the tail bounds for multiplicative drift analysis [2], the bound holds with high probability for a single phase and by a union bound also for all phases together. □

The bound from Theorem 3.8 is by a factor of $\Theta(\log n)$ weaker than the one from Theorem 3.1. We think that the same bound also holds for the StSt $\binom{\mu}{2}$ GA; however, our techniques for the analyses are weaker since independence of bit values cannot be exploited in the proof of Theorem 3.8.

We also conjecture strongly that a lower bound of $\Omega(\mu n)$ holds on BinVal. Similarly to the proof of Theorem 3.3, we can show that with probability $\Omega(1)$, the potential Φ_t still is $\Omega(n)$ after $c\mu n$ steps for a small constant $c > 0$. However, since bits are not sampled independently, this does not yet imply that it is unlikely to sample the optimum. Additional arguments showing that frequencies are quickly decoupled by crossover will be needed.

The proof of Theorem 3.8 essentially pessimistically assumes that the bits are optimized in order of decreasing significance, i. e., according to domino convergence. We now show a much better bound w. r. t. OneMax, resembling Theorem 3.6 up to a $\log n$-factor. In this proof, it is exploited that progress may come from all bits, not only the most significant ones.

Theorem 3.9. *Let $\mu \geq c\sqrt{n} \log n$ for some sufficiently large constant $c > 0$ and $\mu = n^{O(1)}$. Then the runtime of the StSt $\binom{\mu}{2}$ GA on OneMax is $O(\mu\sqrt{n} \log n)$ w. h. p..*

Proof. We consider a phase of $t := c'\mu\sqrt{n}$ iterations for a sufficiently large constant $c' > 0$. Applying the second statement of Lemma 3.7, we obtain that no frequency drops below $1/3$ with high probability if c is chosen as a sufficiently large constant.

The remainder of the proof carries out a drift analysis on the potential $\Phi_t = \sum_{i=1}^{n}(1 - p_{i,t})$, which was already used in the proof of Theorem 3.1. The overall approach resembles mathematically the proof of Theorem 3.6; however, many mathematically similar relations are obtained for different reasons compared to the cGA. We will show a drift $E(\Phi_t - \Phi_{t+1} \mid \Phi_t) = \Omega(V_t/(\mu\sqrt{n}))$, where $V_t := \sum_{i=1}^n p_i(1-p_{i,t})$. Note that the latter quantity would be the sampling variance of the distribution of the cGA, however, here we do not sample bit values independently.

An iteration of StSt $\binom{\mu}{2}$ GA draws two individuals x and y uniformly at random. Each of these has an expected OneMax-value of $\sum_{i=1}^n p_{i,t}$ using linearity of expectation. Also by linearity of expectation, the expected value of the Hamming distance D of x and y is $E(D) := \sum_{i=1}^n 2p_i(1-p_i)$, which we will use below.

Crossing x and y over creates the two offspring x' and y'. The bits where x and y coincide also coincide in the offspring, so selection will not change their frequencies. The bits where they differ contain together D ones and contribute D/μ to the Φ_t-value. We therefore consider an experiment where D independent trials with success probability $1/2$ are executed on the bits different in x and y, identifying a success with crossover putting the one-bit from the considered position into offspring x'. By the properties of the binomial distribution, the number of successes is $D/2 + \Omega(\sqrt{D})$ with constant probability. Hence, the probability that x' has $\Omega(\sqrt{D})$ more ones than the average $D/2$ w. r. t. the different bits is $\Omega(1)$.

To determine the drift of Φ_t, we are interested in the expected value of $\sqrt{D}$, knowing $E(D)$. To this end, we use the following converse of Jensen's inequality (Ineq. (2) in [1]): for convex f and a discrete random variable X with support $\subseteq [0,n]$ it holds that $E(f(X)) \leq E(X)\frac{f(n)}{n}$. Using this with $f(x) = -\sqrt{x}$, we obtain $E(\sqrt{D}) \geq \frac{E(D)}{\sqrt{n}}$. Altogether, we obtain an expected surplus of one-bits in x' over the average $D/2$ that is bounded from below by

$$\Omega(E(\sqrt{D})) = \Omega\left(\sum_{i=1}^n 2p_i(1-p_i)/\sqrt{n}\right) = \Omega\left(\sum_{i=1}^n (2/3)(1-p_i)/\sqrt{n}\right),$$

using that $p_{i,t} \geq 1/3$. Every one-bit in the better offspring contributes $2/\mu$ to Φ_{t+1}. Hence, replacing both x and y by the better offspring decreases the Φ_t-value in expectation by $(c_2/(\mu\sqrt{n}))\sum_{i=1}^n(1-p_i) = (c_2/(\mu\sqrt{n}))\Phi_t$ for some constant $c_2 > 0$. Using the multiplicative drift theorem [2], the expected number of iterations until the potential is reduced to at most $x_{\min} := (n-1)/\mu$ is at most $\frac{\log(x_{\min})+1}{c_2/(\mu\sqrt{n})} = O(\mu\sqrt{n}\log\mu) = O(\mu\sqrt{n}\log n)$, and, using tail bounds for multiplicative drift, the time is $O(\mu\sqrt{n}\log n)$ with high probability. The theorem now follows by observing that at potential $(n-1)/\mu$ the optimum must be in the population. □

3.3 Tournament Selection and Mutation Only

We now look into the StSt $\binom{\mu}{\lambda}$ EA, which is obtained from the StSt $\binom{\mu}{2}$ GA by replacing the crossover by standard bit mutation and using a possibly larger tournament of size λ. In fact, if $\mu = \lambda$, the algorithm after one iteration collapses to the (1,λ) EA which needs population size $\Omega(\log n)$ on all functions with a unique optimum [11]. Intuitively, this minimum size is required to add a sufficient degree of elitism to the algorithm, making it resemble the well-known (1+λ) EA [3]. Hence, we assume throughout this section that $\lambda \geq c\log n$ for a sufficiently large constant $c > 0$.

Interestingly, also the StSt $\binom{\mu}{\lambda}$ EA can suffer from domino convergence but to a much lower extent than the algorithms discussed before. We show this by establishing a relation to the just-mentioned (1+λ) EA, which from 1 parent creates λ offspring by standard bit mutation and replaces the parent by the fittest offspring.

Lemma 3.10. *Given a phase of length $t = n^{c_1}$ and a given failure probability $1/n^{c_2}$ for constants $c_1, c_2 > 0$, there is a choice $\lambda = c\log n$ for a sufficiently large constant $c > 0$, depending on c_1 and c_2, such that the following two properties hold: (1) in all tournaments chosen within the phase there is at least one mutation that does not flip any bit; (2) in addition, if $\mu = \lambda$, the algorithm after the first iteration and until iteration t is identical to the (1+λ) EA with λ reduced by* 1.

Proof. The probability of a mutation not flipping any bit is at least $(1-1/n)^n = (1-o(1))e^{-1}$. The probability that such a mutation does not occur at least one in λ trials it at most $(1-(1-o(1))e^{-1})^\lambda \leq e^{-\lambda} \leq n^{-c}$ if n is large enough. We choose c large enough such that a union bound over n^{c_1} iterations still yields a failure probability of at most n^{-c_2}. This proves the first statement.

For the second statement, observe that the best offspring from the tournament overtakes the whole population. Hence, starting after the first tournament all populations will contain μ identical individuals, so it makes sense to speak of "the" parent in all subsequent iterations. Moreover, note that the condition of reproducing at least once the parent can be modelled equivalently as that the first offspring from the tournament equals the parent while the remaining offspring still are obtained by applying standard bit mutation to the parent. Hence, we obtain the $(1 + (\lambda - 1))$ EA. □

The equivalence established in Lemma 3.10 allows us to transfer known results about domino convergence from [3] to the StSt $\binom{\mu}{\lambda}$ EA. The following two results are given in somewhat simplified and unified form; the paper considers also different settings of λ.

Theorem 3.11 (cf. [3], Th. 8 and 19). *For the (1+λ) EA with $\lambda = O(n)$, the expected number of iterations to optimize BinVal is $\Theta(n + (n\log n)/\lambda)$. The upper bound holds with high probability if $\lambda = O(n^{1-\epsilon})$ for some $\epsilon > 0$.*

Theorem 3.12 (cf. [3], Th. 24). *Let $\epsilon > 0$ be constant and $\lambda = O(n^{1-\epsilon})$. Then for the (1+$\lambda$) EA the number of iterations to optimize OneMax is upper bounded by $O(n\log\log\lambda/\log\lambda + (n\log n)/\lambda)$ in expectation and with high probability.*

Hence, the performance gap of the (1+λ) EA between OneMax and BinVal is $\Omega(\log\lambda/\log\log\lambda)$. Roughly speaking, it stems from the fact that one out of λ offspring in expectation flips this number of bits. This progress is immediately exploited on OneMax but on BinVal only the most significant flipping bit is relevant for selection so that the offspring with many flipping bits does not necessarily have best fitness. This is again a scenario that can be considered as domino convergence.

The same performance gap applies with high probability also to the StSt $\binom{\mu}{\lambda}$ EA with $\mu = \lambda = c\log n$ for a sufficiently large constant $c > 0$. Here we use Lemma 3.10 and exploit that the runtime bounds cited before apply with high probability.

Finally, we should investigate the case $\mu > \lambda$ in the StSt $\binom{\mu}{\lambda}$ EA. It is tempting to conjecture by setting $\mu = s\lambda$, where still $\lambda = c\log n$ for large enough c, results in an algorithm that is at most by a factor of s slower than the setting of $\mu = \lambda$. In general, however, this does not seem to be true. There are constructed example functions where increasing μ in an (μ+1) EA by a polynomial factor makes the expected optimization time grow by an exponential factor [16]. It is very likely that similar examples exist for the StSt $\binom{\mu}{\lambda}$ EA.

Nevertheless, on the simple monotone functions OneMax and BinVal we conjecture that increasing μ by a factor of s changes the expected optimization time also by a factor of $O(s)$. Intuitively, this holds since the next tournament shares an expected $1/s$-fraction of the previous tournament and that neglecting inferior solutions does not harm. So if $s = o(\log \lambda / \log\log \lambda)$, we still expect the StSt $\binom{\mu}{\lambda}$ EA with $\mu = s\lambda$ to be more efficient on OneMax than on BinVal.

3.4 Cut Selection and Mutation Only: the (μ+1) EA

We finally turn to the (μ+1) EA whose runtime on pseudo-Boolean functions, in particular OneMax, is relatively well understood. We therefore start with a presentation of a classical result.

Theorem 3.13 ([16]). *The expected optimization time of the (μ+1) EA on OneMax is* $\Theta(\mu n + n \log n)$.

The lower bound from Theorem 3.13 applies to all functions with a unique optimum. For BinVal, we obtain a larger upper bound.

Theorem 3.14. *The expected optimization time of the (μ+1) EA on BinVal is* $O(\mu n \log \mu + n^2)$.

Sketch of proof. We can basically re-use the analysis of the (μ+1) EA on $\text{LeadingOnes}(x_1, \dots, x_n) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$ from [16], which represents a pessimistic model for the optimization process. For the population at iteration t, we define the potential Φ_t as the number of leading one-bits in the individual having maximal BinVal-value. By the structure of BinVal and the elitist selection of (μ+1) EA, Φ_t is monotone increasing in t. The following event is sufficient to increase the potential: after the first point in time $t' \geq t$ where the whole population consists of individuals with Φ_t leading ones, the zero-bit at position $\Phi_t + 1$ flips. For space reasons, the analysis of the corresponding waiting times is omitted. □

Comparing Theorem 3.13 and 3.14 for $\mu = \Omega(n \log n)$, the lower bound for OneMax and the upper bound for BinVal are only by a factor $O(\log \mu)$ apart. Hence, if the (μ+1) EA exhibits domino convergence (in this range of μ), it is much less pronounced than in the cGA and StSt $\binom{\mu}{2}$ GA. We know in the special case of $\mu = 1$, that the expected optimization time of the (μ+1) EA, in this case the (1+1) EA, is $(1 \pm o(1))en \ln n$ on all linear functions. This is due to the fact that the simple (1+1) EA usually compares search points of small Hamming distance. In the (μ+1) EA, an individual is selected uniformly at random and accepted for the next iteration if is is at least as good as the worst from the population, which may be very different. Hence, we conjecture that the upper bound of Theorem 3.14 is tight for large μ, more precisely we think that a lower bound of $\Omega(\mu n \log \mu + n \log n)$ holds and that domino convergence in fact occurs to the extent of a performance gap of $\Theta(\log \mu)$.

CONCLUSIONS

We have presented a rigorous study of so-called domino convergence in EDAs, genetic algorithms using crossover, and mutation-only evolutionary algorithms. For the first two algorithms we observe a performance gap of almost linear order between the two linear functions OneMax and BinVal, answering an open problem formulated by Droste [4]. Intuitively, this performance gap is due to the effect of crossover, which on BinVal makes the algorithm compare individuals of typically large Hamming distance. Then frequencies belonging to bits of high weight tend to stabilize much earlier than bits of low weight, as predicted by earlier models of domino convergence and observed in experimental studies.

We have also investigated two algorithms that use mutation as only variation operator. In a variant using tournament selection, also a performance gap between BinVal and OneMax stemming from domino convergence is observed, but this gap is only logarithmic in the population size. In the (μ+1) EA, the performance gap is also at most logarithmic in the population size μ, but in fact it may even be smaller. Altogether, our results show that the optimization of linear functions by randomized search heuristics leads to a much richer behavior than the $\Theta(n \log n)$ bound for the (1+1) EA suggests. It is an open problem to study more complex and common GAs involving both mutation and crossover in this context.

Acknowledgments. The author would like to thank Fernando Lobo for pointing him to [10].

REFERENCES

[1] M. K. Bakula, J. Pečarić, and J. Perić. On the converse Jensen inequality. *Applied Mathematics and Computation*, 218:6566–6575, 2012.
[2] B. Doerr, D. Johannsen, and C. Winzen. Multiplicative drift analysis. *Algorithmica*, 64:673–697, 2012.
[3] B. Doerr and M. Künnemann. Optimizing linear functions with the (1+λ) evolutionary algorithm – different asymptotic runtimes for different instances. *Theoretical Computer Science*, 561:3–23, 2015.
[4] S. Droste. A rigorous analysis of the compact genetic algorithm for linear functions. *Natural Computing*, 5:257–283, 2006.
[5] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.
[6] T. Friedrich, T. Kötzing, M. S. Krejca, and A. M. Sutton. The benefit of recombination in noisy evolutionary search. In *Proc. of ISAAC '15*, pages 140–150. Springer, 2015.
[7] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3:287–297, 1999.
[8] J. Jägersküpper. Combining markov-chain analysis and drift analysis – the (1+1) evolutionary algorithm on linear functions reloaded. *Algorithmica*, 59:409–424, 2011.
[9] T. Kötzing. Concentration of first hitting times under additive drift. *Algorithmica*, 75:490–506, 2016.
[10] F. G. Lobo, D. E. Goldberg, and M. Pelikan. Time complexity of genetic algorithms on exponentially scaled problems. In *Proc. of GECCO '00*, pages 151–158. Morgan Kaufmann, 2000.
[11] J. E. Rowe and D. Sudholt. The choice of the offspring population size in the (1, λ) evolutionary algorithm. *Theoretical Computer Science*, 545:20–38, 2014.
[12] W. M. Rudnick. *Genetic algorithms and fitness variance with an application to the automated design of artificial neural networks*. PhD thesis, Oregon Graduate Institute of Science & Technology, Klamath Falls Publication, 1992.
[13] H. Stringer and A. S. Wu. A simple method for detecting domino convergence and identifying salient genes within a genetic algorithm. In *Proc. of GECCO '02*, pages 594–601. Morgan Kaufmann, 2002.
[14] D. Sudholt and C. Witt. Update strength in EDAs and ACO: How to avoid genetic drift. In *Proc. of GECCO '16*, pages 61–68. ACM Press, 2016.
[15] D. Thierens, D. Goldberg, and A. Pereira. Domino convergence, drift, and the temporal-salience structure of problems. In *Proceedings of CEC '98*, pages 535–540. IEEE Press, 1998.
[16] C. Witt. Runtime analysis of the (μ + 1) EA on simple pseudo-boolean functions. *Evolutionary Computation*, 14:65–86, 2006.
[17] C. Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing*, 22:294–318, 2013.