The $(1+1)$-EA with mutation rate $(1+\varepsilon)/n$ is efficient on monotone functions: an entropy compression argument. (Hot-off-the-Press Track at GECCO 2019)

Johannes Lengler
Department of Computer Science,
ETH Zürich, Zürich, Switzerland

Anders Martinsson
Department of Computer Science
ETH Zürich, Zürich, Switzerland

Angelika Steger
Department of Computer Science
ETH Zürich, Zürich, Switzerland

ABSTRACT

An important benchmark for evolutionary algorithms are (strictly) monotone functions. For the (1 + 1)-EA with mutation rate c/n, it was known that it can optimize any monotone function on n bits in time $O(n \log n)$ if $c < 1$. However, it was also known that there are monotone functions on which the (1 + 1)-EA needs exponential time if $c > 2.2$. For $c = 1$ it was known that the runtime is always polynomial, but it was unclear whether it is quasilinear, and it was unclear whether $c = 1$ is the threshold at which the runtime jumps from polynomial to superpolynomial.

We show there exists a $c_0 > 1$ such that for all $0 < c \leq c_0$ the (1 + 1)-Evolutionary Algorithm with rate c/n finds the optimum in $O(n \log^2 n)$ steps in expectation. The proof is based on an adaptation of Moser's entropy compression argument. That is, we show that a long runtime would allow us to encode the random steps of the algorithm with less bits than their entropy.

This short note summarizes the results that have been published as "When Does Hillclimbing Fail on Monotone Functions: An entropy compression argument" in the proceedings of the 16th Workshop on Analytic Algorithmics and Combinatorics (ANALCO), SIAM, 2019 [9].

SUMMARY OF OUR RESULTS

One of the most basic evolutionary algorithms (EAs) is the (1 + 1)-*Evolutionary Algorithm* or (1 + 1)-EA for maximizing an objective function $f : \{0,1\}^n \to \mathbb{R}$. It uses a population size of one, and in each round it creates an offspring by flipping each bit of the parent independently with probability c/n, where $c > 0$ is the *mutation parameter*. Then it greedily selects the fitter one, breaking ties towards the offspring.

Here we study the impact of the mutation parameter c on the performance of the (1 + 1)-EA on (strictly) monotone functions. A function $f : \{0,1\}^n \to \mathbb{R}$ is said to be *monotone* if $f(x) < f(y)$ whenever $x \neq y$ and $x^i \leq y^i$ for all $i \in [n]$, where x^i denote the i-th coordinate of x. For any such function, the unique global maximum is the all-ones string. Monotone functions are an important class of benchmark functions for EAs because of their discriminative power. On the one hand, there is good reason to consider monotone functions as easy benchmarks, and there is a large variety of EAs that optimize all monotone functions efficiently. On the other hand, a surprising number of EAs need exponential time on some monotone functions if the mutation is too aggressive, including the (1 + 1)-EA, the $(1 + \lambda)$-EA, the $(\mu + 1)$-EA, their so-called "fast" versions and the $(1 + (\lambda, \lambda))$-GA, while the genetic algorithms $(\mu + 1)$-GA and its "fast" version seem to be more robust, see [8] for an overview. Monotone functions also have surprising links to optimization in dynamic environments [10].

The (1 + 1)-EA is a prototypical example for the behavior of many EAs. While for any constant mutation parameter $c < 1$ it is easy to see that the algorithm needs time $O(n \log n)$ to find the optimum of any monotone function [4], it was shown in a sequence of papers [4, 5, 11] that for $c > 2.13\ldots$ there are monotone functions (dubbed HotTopic functions in [8]) on which the algorithm needs exponential time. The standard proof techniques for upper runtime bounds fail precisely at $c = 1$, and there were split opinions in the community on whether there should be a phase transition from polynomial to exponential at $c = 1$ [3]. On the presumed threshold $c = 1$ a more general model of Jansen [7] shows that the runtime is $O(n^{3/2})$, but it was unclear whether the runtime is quasilinear.

The value $c = 1$ is of special interest, for several reasons. From a practical perspective, it is considered the standard choice and explicitly recommended by textbooks [1, 2]. From a theoretical perspective, $c = 1$ is known to be the optimal parameter choice for linear functions, i.e., for functions of the form $f(X) = \sum_{i=1}^{n} w_i X^i$, where the w_i are fixed weights. More precisely, the choice $c = 1$ gives runtime $(1 + o(1))en \log n$ on any linear function, while any other (constant) choice of c gives a strictly worse leading constant on any linear function [14].

In the new result, we use an entropy compression argument to show that there is an $\varepsilon > 0$ such that for $c \leq 1 + \varepsilon$ the runtime remains quasilinear for all monotone functions. More precisely, we show that a long runtime would allow us to encode the random trajectory of the algorithm with fewer bits than its entropy, which is an information theoretic contradiction. This type of argument is attributed to Moser, who used the technique in his celebrated algorithmic proof of the Lovász local lemma [6, 12]. Since then, the method has been used in other computer science contexts, see [9] for a summary. Nevertheless, and despite popular blogposts [6, 13], the method still does not seem to be widely known.

We use this technique to prove that no phase transition occurs at $c = 1$. More precisely, we show the following.

Theorem 0.1. *There exists an $\varepsilon > 0$ such that, for any (strictly) monotone function f and any constant $0 < c \leq 1 + \varepsilon$, the $(1+1)$-EA with mutation parameter c requires an expected number of $O(n \log^2 n)$ steps until it finds the maximum of f, and it visits an expected number*

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07.
https://doi.org/10.1145/3319619.3326767

of $O(n)$ search points. The same remains true if the initial search point of the algorithm is chosen by an adversary.

To be more precise, we show that there are constants $\varepsilon, C > 0$ such that for all $0 < c \leq 1 + \varepsilon$ the runtime is at most $C/c \cdot n \log^2 n$, and the number of search points is at most $C \cdot n$.

To apply an entropy compression argument, it turns out to be natural to not measure performance in the number of time steps, but in the number of *updates*, i.e. the number of times $X_{t+1} \neq X_t$. We show that the expected number of updates until the algorithm finds the maximum of any monotone f is $O(n)$. Towards the end of the proof, we link the expected number of updates to the expected number of time steps and show that they only differ by a polylogarithmic factor.

We first give some intuition on the behaviour of the algorithm and on our proof. For a general monotone function f, it is natural to measure the progress of the $(1 + 1)$-EA is terms of number of one-bits in the current search point. In order to make an update, it is necessary to flip at least one zero-bit into a one-bit, since otherwise the offspring would be rejected due to monotonicity (unless it is identical to the parent, in which case there is no update either). Thus, if the average update does not flip too many ones to zeros, the number of ones in the current search point will tend to n efficiently. For a small mutation parameter (specifically for $c < 1$), this is indeed true as the average number of ones flipped to zeros is at most c, and this remains true for update steps. For larger c, one might expect this to still hold as, intuitively, any offspring with more ones flipped to zeros than zeros flipped to ones should be unlikely to be fitter than its parent. To see why this intuition fails for large c, consider the following situation for a linear objective function. If a zero-bit of high weight is flipped to a one, then the algorithm may accept the offspring even if many low weight one-bits are flipped at the same time. The larger c is, the more such one-bits are flipped in expectation. While such bad steps do not occur too often for linear functions, it has been shown that there exist monotone (locally linear) functions, such as HotTopic functions [8], where this effect keeps the algorithm away from the optimum for exponentially long time.

Based on this intuition, we define *good* and *bad* updates. The bad updates capture cases in which a zero-bit with a disproportionally high weight is flipped. Then we study the entropy of the update steps of the algorithms. On the one hand, we will analyze the algorithm in a forward manner to give a lower bound on the entropy of each update. On the other hand, we give a backwards encoding (from last step to first) of the update steps, and this encoding saves some bits in bad update steps. Since the expected number of bits needed for the encoding is lower bounded by the entropy, we get an upper bound for the expected number of bad update steps. This, in turn, gives us a linear upper bound for the expected number of update steps. Finally, the runtime bound follows by a slight refinement of the calculation, in which we compute how many steps we need to decrease the number of zero-bits from 2^m to 2^{m-1}, for $m = \log n, \dots, 1$.

It remains open whether the technique transfers to other EAs. It was shown in [8] that also other EAs than the $(1 + 1)$-EA optimize every monotone function efficiently if their parameters are set conservatively. E.g., this was shown for the $(1+\lambda)$-EA with constant $\lambda \in \mathbb{N}$, if the mutation parameter satisfies $c < 1$. It remains an open question whether an entropy compression technique can also be used to show that the runtime of the $(1+\lambda)$-EA is quasilinear for $c \leq 1+\varepsilon$. Moreover, in [8] analogous results were shown for the so-called "fast $(1+1)$-EA" and "fast $(1+\lambda)$-EA", and for the $(1+(\lambda, \lambda))$-GA, but the condition $c < 1$ needs to be replaced by analogous conditions on the parameters of the respective algorithms.[1] Also in these cases it remains open whether the entropy compression method can be used to push the condition under which the algorithm is provably fast for all monotone functions.

REFERENCES

[1] T. Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford university press, 1996.

[2] T. Bäck, D. B. Fogel, and Z. Michalewicz. *Handbook of evolutionary computation.* CRC Press, 1997.

[3] B. Doerr, C. Doerr, and T. Kötzing. Personal communication.

[4] B. Doerr, T. Jansen, D. Sudholt, C. Winzen, and C. Zarges. Optimizing monotone functions can be difficult. In *Parallel Problem Solving from Nature (PPSN)*, 2010.

[5] B. Doerr, T. Jansen, D. Sudholt, C. Winzen, and C. Zarges. Mutation rate matters even when optimizing monotonic functions. *Evolutionary computation*, 21(1):1–27, 2013.

[6] L. Fortnow. A Kolmogorov Complexity Proof of the Lovász Local Lemma. Blogpost, 2009. https://blog.computationalcomplexity.org/2009/06/kolmogorov-complexity-proof-of-lov.html.

[7] T. Jansen. On the brittleness of evolutionary algorithms. In *Foundations of Genetic Algorithms (FOGA)*. Springer, 2007.

[8] J. Lengler. A general dichotomy of evolutionary algorithms on monotone functions. In *Parallel Problem Solving from Nature (PPSN), full version at arxiv.org/abs/1803.09227*, 2018.

[9] J. Lengler, A. Martinsson, and A. Steger. When does hillclimbing fail on monotone functions: An entropy compression argument. In *Analytic Algorithmics and Combinatorics (ANALCO)*. SIAM, 2019.

[10] J. Lengler and U. Schaller. The (1+ 1)-EA on noisy linear functions with random positive weights. In *Foundations of Computational Intelligence (FOCI)*. IEEE, 2018.

[11] J. Lengler and A. Steger. Drift Analysis and Evolutionary Algorithms Revisited. *Combinatorics, Probability and Computing*, 27(4):643–666, 2018.

[12] R. Moser. A constructive proof of the Lovász Local Lemma. In *Symposium on Theory of Computing (STOC)*, 2009.

[13] T. Tao. Moser's entropy compression argument. Blogpost, 2009. https://terrytao.wordpress.com/2009/08/05/mosers-entropy-compression-argument/.

[14] C. Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing*, 22(2):294–318, 2013.

[1] In the case of the "fast $(1 + \lambda)$-EA", the result was only proven if the algorithm starts sufficiently close to the optimum.