Runtime of the SA-$(1, \lambda)$-EA on Dynamic Monotone Functions when the Mutation Rate is $1/n$

Supervisors: Maxime Larcher, Marc Kaufmann
Supervising Professor: Prof. Dr. Angelika Steger
Type of Project: Bachelor's Thesis, Master's Thesis, Research Project

If you are interested please send an email to Maxime Larcher.

1 Introduction

Evolutionary Algorithms (or EAs for short) are optimisers inspired by biological evolution. There are many such algorithms, and one of their core property is making little to no assumption about the function to optimise, so that one can apply them to many problems 'off-the-shelf', i.e. with little adaptation. For this reason, they find application in a wide range of areas such that image processing, robotics, machine learning, etc. where the landscape of the function to optimise is hard to describe.

In theoretical work, one typically tries to understand the runtime and behaviour of EAs on *binary* functions $f\colon \{0,1\}^n \to \mathbf{R}$. One of the first and simplest EA to be introduced is the $(1+1)$-EA: it starts from a random point $x \in \{0,1\}^n$ and generates an offspring y by mutating every bit with probability $1/n$. If y is fitter than x — that is if $f(y) > f(x)$ — then x is replaced by y. Otherwise x remains unchanged and y is discarded. This reproduction-selection process is performed again and again, and one hopes that the optimum x^* of f is found rapidly. In a series of papers [2, 3, 8, 7], it was notably shown that the $(1+1)$-EA finds the optimum of any monotone[1] function in at most $O(n \log n)$ generations.

In two recent papers, Hevia-Fajardo and Sudholt [4] and Kaufmann, Larcher, Lengler and Zou [5] studied a related algorithm called the *Self-Adjusting* $(1, \lambda)$*-EA* (or SA-$(1, \lambda)$-EA for short). The general dynamic is the same except that:

(i) instead of generating a single child, we generate λ offspring $y_1, \dots, y_\lambda$;

(ii) in the selection phase the parent x is *always* replaced by the fittest child y_{i^*}, even if all children are less fit that the parent;

(iii) whenever a step is successful in improving the fitness, the number of children in the next round is decreased to λ/F, and whenever no improvement is found the number of children in the next round is increased to $\lambda \cdot F^{1/s}$. Here F and s are (constant) meta-parameters.

[1] A function f is monotone if $f(y) > f(x)$ whenever $y_i \geq x_i$ for all $i \in [n]$, with strict inequality for at least one index i. In other words, f is monotone if flipping a bit from 0 to 1 always increases the function.

The complete pseudocode may be found in e.g. [4] Algorithm 1. The intuition behind the 'self-adaptation' in (iii) can be summarised as follows: if it is easy to find improvements then we may reduce the number of children generated at each step to save some computation; on the other hand, if it is hard to find improvements, then we wish to increase λ to reduce the number of generations needed to reach the optimum, as well as reduce the risk of seeing a drop in fitness. Very roughly speaking[2], we expect that λ remains in a range where the probability of success is $\approx \frac{1}{s+1}$, i.e. neither too large, nor too small.

2 Goal of the Thesis

Hevia-Fajardo and Sudholt [4] showed that when the meta-parameter s is chosen small enough, the SA-$(1, \lambda)$-EA finds the optimum of OneMax[3] in an expected $O(n)$ generations and $O(n \log n)$ evaluations (i.e. the total number of children generated). Among various other results, Kaufmann, Larcher, Lengler and Zou [5] generalised this result to *dynamic*[4] monotone functions.

A slight downside of their work is that, in the case precisely stated above, their analysis only gives $O(n \log n)$ generations and $\tilde{O}(n^2)$ evaluations when both s, F are small. That is, not only is the bound worst, but the proof only holds when F is small. In this thesis we propose to continue the work of Kaufmann et al. and improve their results. In particular, combining their techniques and some ideas from the PO-EA of Colin, Doerr and Férey [1] we believe the following conjecture can be proved.

Conjecture 2.1. *For all dynamic monotone functions, the SA-$(1, \lambda)$-EA finds the optimum in an expected $O(n^{3/2})$ evaluations.*

It is not entirely clear whether one can improve this further for all dynamic monotone functions. For instance we believe that the function Adversarial Dynamic BinVal (or ADBV for short) defined as in the conclusion of [5], is a good candidate for a function that is hard to optimise. One other objective of the thesis could be to prove the following conjecture.

Conjecture 2.2. *There exists a dynamic monotone function f on which the SA-$(1, \lambda)$-EA takes an expected $\omega(n)$ generations and $\omega(n \log n)$ evaluations.*

Although some functions like ADBV are probably hard, we believe that the SA-$(1, \lambda)$-EA should still be extremely efficient. Another possible part of the thesis could be dedicated to finding large classes of functions for which the SA-$(1, \lambda)$-EA runs in $O(n)$ generations or $O(n \log n)$ evaluations. For example, one could seek to prove the following statement.

Conjecture 2.3. *Let $\mathcal{F}$ be the family of all* linear *functions, i.e. the set of functions which can be written as $f(x) = \sum_{i=1}^{n} w_i x_i$ for some positive $w_1, \ldots, w_n$. The SA-$(1, \lambda)$-EA optimises any $f \in \mathcal{F}$ in $O(n)$ generations and $O(n \log n)$ evaluations.*

If time permits other related problems may be considered.

[2] Although useful, this intuition is wrong; as λ increases and decreases exponentially, it is easy to deviate from expectation.

[3] OneMax (or OM for short) is the function which counts the number of bits set to1, i.e. $\mathrm{OM}(x) = \sum_i x_i$.

[4] In a slight abuse of notation, we call *dynamic monotone function* a sequence $(f_t)_{t \geq 0}$ in which every f_t is monotone.

3 Tools, Prerequisites and Grading

As is now standard in analysis of EAs, the proofs will (very likely) rely on drift analysis and we refer to e.g. Lengler [6] for a survey on the topic (see notably Section 3 for the main idea). Previous experience with drift theory is appreciated, but not required.

However, we do require you to have a very good understanding of probabilities and in particular we ask you to have an excellent grade in RandAlg resp. AlgoWahr (or equivalent lectures) if you are a Master student resp. a Bachelor student.

As usual in mathematics and theoretical computer science, predicting positive results in a limited (even unlimited) amount of time is a hard, if not impossible, task. Therefore, we do not set strict requirements of what the student is expected to achieve for a particular grade. This project is intended both for Bachelor and Master students, but our expectations for either group evidently differ. The grading criteria of the thesis follows the guidelines on CADMO webpage.

References

[1] S. Colin, B. Doerr, and G. Férey. Monotonic functions in ec: anything but monotone! In *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, pages 753–760, 2014.

[2] B. Doerr, T. Jansen, D. Sudholt, C. Winzen, and C. Zarges. Optimizing monotone functions can be difficult. In *International Conference on Parallel Problem Solving from Nature*, pages 42–51. Springer, 2010.

[3] 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.

[4] M. A. Hevia Fajardo and D. Sudholt. Self-adjusting population sizes for non-elitist evolutionary algorithms: why success rates matter. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1151–1159, 2021.

[5] M. Kaufmann, M. Larcher, J. Lengler, and X. Zou. Self-adjusting population sizes for the $(1, \lambda)$-EA on monotone functions. *arXiv preprint arXiv:2204.00531*, 2022.

[6] J. Lengler. Drift analysis. In *Theory of Evolutionary Computation*, pages 89–131. Springer, 2020.

[7] J. Lengler, A. Martinsson, and A. Steger. When does hillclimbing fail on monotone functions: An entropy compression argument. In *2019 Proceedings of the Sixteenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 94–102. SIAM, 2019.

[8] J. Lengler and A. Steger. Drift analysis and evolutionary algorithms revisited. *Combinatorics, Probability and Computing*, 27(4):643–666, 2018.