Evolutionary Algorithms

INF 421 Project

Contact: Martin (martin.krejca@polytechnique.edu)

General remarks. The source code for this project, the report, as well as the final defense need to be in **English.** You can use **any programming language** as long as the source code is formatted *and annotated* such that it is evident what it does even if one is not familiar with the language.

In general, you can **use any libraries** that you want. However, since some languages might provide shortcuts for certain tasks deemed important in this project, **you need to implement certain algorithms yourself.** This is stated explicitly in this document.

If you do not know what programming language to use, consider D or Nim, not because they are best suited for this task but just because they are fun.

1 Context

Evolutionary algorithms (EAs) are general-purpose optimization heuristics. That is, they are applicable to a wide range of optimization problems, but they do not promise to yield an optimal solution. Nonetheless, EAs usually show promising results, especially in domains where no problem-specific solver exists. Most prominently, EAs are applied with great success in black-box optimization scenarios, that is, settings where the user has no analytical representation of the exact problem or optimizing such a problem would be very hard. In such settings, a solution s is evaluated by running a simulation (or a real experiment) and thus assessing the quality of s. This quality measure is enough for EAs to be guided through the search space.

In general, an EA follows roughly the idea of maintaining a multi-set of solutions (the *population*) that is updated iteratively by creating new random solutions (the *offspring*) from the population and selecting the best from them with respect to the objective function (the *fitness function*). In order to fully define an EA, it is necessary to specify how solutions look like and how to create new solutions. Algorithm 1 shows an easy but quite general EA for a specific optimization setting known as *pseudo-Boolean* optimization, that is, given a problem size $n \in \mathbb{N}_{\geq 1}$, the optimization of functions $f \colon \{0,1\}^n \to \mathbb{R}$. In this project, we only consider pseudo-Boolean optimization, and we always implicitly assume that the problem size n is given.

Algorithm 1: The $(\mu + \lambda)$ evolutionary algorithm ($(\mu + \lambda)$ EA). Given the parent population size $\mu \in \mathbb{N}_{\geq 1}$, the offspring population size $\lambda \in \mathbb{N}_{\geq 1}$, the problem size $n \in \mathbb{N}_{\geq 1}$, and the pseudo-Boolean objective function $f\colon \{0,1\}^n \to \mathbb{R}$, the algorithm generates better solutions with respect to f over time. It stops after a user-defined termination criterion. Sets in the following always denote multi-sets. The operation in line 6 is known as *standard bit mutation*.

1 $t \leftarrow 0$;
2 $P^{(t)} \leftarrow \mu$ solutions chosen uniformly at random (u.a.r.) from $\{0,1\}^n$;
3 **while** termination criterion not met **do**
4 **for** $i \in [1..\lambda]$ **do**
5 $\boldsymbol{x}^{(t,i)} \leftarrow$ individual chosen u.a.r. from $P^{(t)}$;
6 $\boldsymbol{y}^{(t,i)} \leftarrow$ copy of $\boldsymbol{x}^{(t,i)}$, flipping each bit independently with probability $1/n$;
7 $Q^{(t)} \leftarrow$ sort $P^{(t)} \cup \{\boldsymbol{y}^{(t,i)} \mid i \in [1..\lambda]\}$ with respect to f, preferring individuals from $\{\boldsymbol{y}^{(t,i)} \mid i \in [1..\lambda]\}$ over those from $P^{(t)}$, breaking remaining ties u.a.r.;
8 $P^{(t+1)} \leftarrow$ the best μ solutions from $Q^{(t)}$ with respect to f;

2 Terminology and Benchmark Functions

In this project, we investigate the *run time* behavior of different EAs on different fitness functions that are often utilized as benchmarks in the theoretical analysis of EAs. In the following, we make all of these choices more precise.

For a given EA A and a function f, let the *run time of A on f* be defined as the (random) number of fitness function evaluations until the population contains a globally optimal solution of f for the first time. We say that the fitness function f is evaluated for a specific individual $\boldsymbol{x} \in \{0,1\}^n$ if and only if we determine $f(\boldsymbol{x})$ for the first time. In other words, we evaluate the fitness of an individual only once, but if we have identical individuals, we evaluate the fitness of *each* of them (as we consider multi-sets and thus make a distinction between equal solutions).

For the $(\mu + \lambda)$ EA, we initially have μ fitness function evaluations. Afterward, we add exactly λ fitness function evaluations each iteration. Hence, we can think of the run time of the $(\mu + \lambda)$ EA on a function f as μ plus the first iteration (the variable t) such that $P^{(t)}$ contains a global optimum of f time λ. Thus, we define that our termination criterion is that the current population contains a global optimum.

We consider the following three pseudo-Boolean benchmark functions (all with implicit dimension size $n \in \mathbb{N}_{\geq 1}$), which we all aim to *maximize*. We define, where $k \in [1..n]$,

$$\textsc{OneMax}\colon \boldsymbol{x} \mapsto \sum_{i \in [1..n]} \boldsymbol{x}_i, \qquad \textsc{LeadingOnes}\colon \boldsymbol{x} \mapsto \sum_{i \in [1..n]} \prod_{j \in [1..i]} \boldsymbol{x}_j, \text{ and}$$

$$\textsc{Jump}_k\colon \boldsymbol{x} \mapsto \begin{cases} k + \textsc{OneMax}(\boldsymbol{x}) & \text{if } \textsc{OneMax}(\boldsymbol{x}) \in [0..n-k] \cup \{n\}, \\ n - \textsc{OneMax}(\boldsymbol{x}) & \text{else.} \end{cases}$$

In other words, OneMax returns the number of 1s of the input, LeadingOnes returns the length

of the longest consecutive prefix of 1s, and $\textsc{Jump}_k$ is like OneMax except that solutions whose number of 1s is within $[n-k+1..n-1]$ have a worse fitness than any other solution, making it especially hard to get to the global maximum. Further note that the all-1s bit string is the unique global optimum of all three functions.

Task 1. Write code that allows to use individuals as well as the three functions OneMax, LeadingOnes, and $\textsc{Jump}_k$. For individuals, do not use libraries but implement a data type that fully utilizes the memory. That is, do not store each bit value of an individual in a byte but in an actual bit.

3 First Analyses

We first consider the $(1+1)$ EA—a special variant of the $(\mu+\lambda)$ EA for $\mu = \lambda = 1$. The $(1+1)$ EA is the most studied algorithm in the theory domain of EAs. Our aim is to analyze its expected run time on OneMax and on LeadingOnes both theoretically and empirically.

For the theoretical analysis, you are not allowed to cite existing results. Instead, please rely on elementary tools from probability theory. If you are unsure, please ask. The aim is to formally prove *upper* bounds on the *expected* run time of the algorithm (with leading constants).

For the empirical analysis, we aim to see how tight our theoretical bounds are. To this end, please run the algorithm on each of the two fitness functions for various problem sizes and each time for a reasonable number of times. Choose yourself the respective values and briefly explain your choice in the report. For the algorithm, use the tools you created in Task 1. Save the results in a file of a suitable format. Furthermore, please visualize your results nicely and explain how this compares to your theoretical upper bound. The code for running the experiments needs to be handed in together with the report. However, the code for the visualization does not need to be submitted. For the visualization, feel free to use any tool that you deem fit.

Task 2. Prove mathematically (preferably rather tight) upper bounds on the expected run time of the $(1+1)$ EA on OneMax and on LeadingOnes. Complement your theoretical bounds with empirical results and compare them.

Furthermore, run empirical tests for the $(\mu+1)$ EA on OneMax with various, self-chosen values of μ. Visualize the expected run time. What do you see? What μ would you recommend?

In your report, do not forget to add a brief discussion about the parameter choices you made yourself, especially the number of tries for each value of n you chose.

4 Population Dynamics

We now focus on the $\textsc{Jump}_k$ benchmark. In the search space $\{0,1\}^n$ of this function, we call the set $\{x \mid \textsc{OneMax}(x) = n-k\}$ the *plateau*, as it is an area with many local optima. Usually, the population of an EA gets stuck on the plateau for some time before one solution is changed into

the global optimum. Before we waste computing resources in order to get an idea how long this takes, we rather compute the expected time until this happens ourselves.

Task 3. Let $k \in [1..n]$. Assume that you start the $(1+1)$ EA on the plateau of $\textsc{Jump}_k$. Prove mathematically the expected number of iterations until the global optimum is created for the first time via standard bit mutation (line 6).

The problem with the $(\mu+\lambda)$ EA on $\textsc{Jump}_k$ is that once the population is on the plateau, we need to hope that one solution makes the jump to the global optimum. However, to this end, the population is not really useful, as it just results in multiple tries per iteration. While this reduces the number of iterations, it does not reduce the run time.[1] This is unsatisfactory and not the intention of a population. We should make better use of having potentially many different solutions at our disposal.

4.1 Incorporating Crossover

Any operator that combines multiple solutions into a new solution is called *crossover*.[2] The idea behind crossover is that if we use different solutions, a clever combination of them can result in a vastly different and, with a bit of luck, in a vastly better solution. For our setting, we consider a binary crossover operator, known as *uniform crossover*. Given two individuals $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^n$, the uniform crossover of $\boldsymbol{x}$ and $\boldsymbol{y}$ is a *new* individual $\boldsymbol{z}$ where, at each position $i \in [1..n]$, the bit value $\boldsymbol{z}_i$ is chosen uniformly at random from $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$.

Evolutionary algorithms that apply crossover operators are also often called *genetic algorithms*[3]. We are interested in the $(\mu+\lambda)$ genetic algorithm ($(\mu+\lambda)$ GA), which is a modification of the $(\mu+\lambda)$ EA as follows: The $(\mu+\lambda)$ GA requires an extra parameter $p_c \in [0,1]$ called the *crossover rate*. In each iteration $t \in \mathbb{N}$, when creating offspring, for each each new offspring, with probability $1-p_c$, we do exactly what the $(\mu+\lambda)$ EA does. However, with probability p_c, we choose instead *two* individuals from $P^{(t)}$ (with replacement) and let the offspring be the result of a uniform crossover of these two individuals (and only then perform standard bit mutation on the offspring). The rest remains unchanged.

Task 4. Implement the $(\mu+\lambda)$ GA such that it can be run on pseudo-Boolean functions.

When run on $\textsc{Jump}_k$, we hope that the population of the $(\mu+\lambda)$ GA on the plateau gets diverse enough such that it is easier to create the global optimum. However, it is not immediately clear whether uniform crossover is helpful for *creating* diversity. This has been proven to some extent some time ago [DFK+18] and recently improved [DEJ+23] (actually by students from Ecole Polytechnique!). Nonetheless, these theoretical guarantees are still far off from what empirical

1 In fact, as we check for an optimum only after each iteration, that is, in increments of λ, increasing the population size potentially only increases the run time.

2 This is in contrast to operators that change a *single* solution into a new one, which are called *mutation*, such as the standard bit mutation (line 6).

3 Although many people also like to call *any* evolutionary algorithm a genetic algorithm.

results suggest. In other words, studying the expected run time of genetic algorithms on $\textsc{Jump}_k$ is actually still subject to state-of-the art research in the EA domain. Thus, we aim to get at least some empirical insights into this setting.

Our main object of interest is the *diversity* of the $(\mu + \lambda)$ GA. If the population contains sufficiently many individuals that mutually do not share many (or, ideally, any) 0s, the run time improves drastically. Hence, for a population P and a value $d \in [0..k]$, we define the diversity of P at distance d as the number of pairs of individuals in P that have Hamming distance exactly $2d$, where the Hamming distance of two individuals is the number of positions where they differ.[4] Looking at how the diversity changes over the number of iterations for different distances gives us a good idea about how much the probability of reaching the global optimum changes. We are happy once a sufficient number of pairs (for example, a fourth of them) are at maximum distance $2k$.

Task 5. For at least three values of n (at least 100, preferably far larger), of k (do not go larger than 6 or 7 here, but larger than 1), of μ (at least $\lfloor \ln(n) \rfloor$), of λ (containing the value 1), and of p_c (containing the value 1), measure the diversity of the $(\mu + \lambda)$ GA on $\textsc{Jump}_k$, initialized such that the initial population $P^{(0)}$ only contains individuals on the plateau, chosen uniformly at random. Stop the algorithm once the diversity for the maximum distance is sufficiently high (or after a maximum number of iterations).

For each parameter combination, pick one of the runs and visualize the diversity of the population over time. Please also mark the iteration when the optimum was found for the first time.[a]

Please do not forget to briefly discuss your parameter choices, especially how many times you ran each setup. Furthermore, state whether plots for identical setups (of which you only show one in the report) are qualitatively the same. What do you see? Is there a correlation between some of the parameters and the number of iterations required in order to get to a certain level of diversity?

a This shows us what level of diversity is already sufficient.

References

[DEJ+23] Benjamin Doerr, Aymen Echarghaoui, Mohammed Jamal, and Martin S. Krejca. "Lasting Diversity and Superior Runtime Guarantees for the (μ+1) Genetic Algorithm". In: *CoRR* abs/2302.12570 (2023). DOI: 10.48550/ARXIV.2302.12570 (see page 4).

[DFK+18] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. "Escaping Local Optima Using Crossover With Emergent Diversity". In: *IEEE Transaction on Evolutionary Computation* 22.3 (2018), pp. 484–497. DOI: 10.1109/TEVC.2017.2724201 (see page 4).

4 Note that if all individuals are on the plateau, their Hamming distance needs to be an even number.