Advanced Optimization class

Université Paris-Saclay

Exercise: Pure random search and (1+1)-EA on $\Omega = \{0,1\}^n$

November 20, 2018

Anne Auger, Dimo Brockhoff
firstname.lastname@inria.fr

Pure Random Search (PRS)

The first stochastic optimization algorithm, introduced before any genetic algorithm (GA) or evolution strategy (ES), is the so-called pure random search, or PRS for short[1]. Assuming a bounded search domain, the algorithm consists in sampling independently points *uniformly* distributed in the search space.

We will implement the algorithm PRS in the context of the maximization of functions defined on the space of bit strings of a certain length n, i.e. on the search space $\Omega = \{0,1\}^n$.

Pure Random Search to maximize $f : \{0,1\}^n \to \mathbb{R}$

```
Initialize uniformly at random $\boldsymbol{x} \in \Omega = \{0,1\}^n$
while not terminate
    Sample $\boldsymbol{x}'$ uniformly at random in $\Omega$
    if $f(\boldsymbol{x}') \geq f(\boldsymbol{x})$
        $\boldsymbol{x} = \boldsymbol{x}'$
return $\boldsymbol{x}$
```

Example 1 *For $\boldsymbol{x} \in \{0,1\}^n$, the function* OneMax *is defined as*

$$f_{\text{OneMax}}(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{x}_i$$

1. What is the maximum of the OneMax function? What is the value of f_{OneMax} at the optimum?
2. Write a Python function `onemax` that takes as argument a bitstring $\boldsymbol{x}$ of arbitrary length (as `numpy.array`) and gives back the value $f_{\text{OneMax}}(\boldsymbol{x})$ (useful instruction: `numpy.sum`).
3. Write a Python function `pureRS` that takes as argument the search space size n and returns the number of function evaluations needed to reach the optimum of the OneMax function, as well as a

[1] The historical papers proposing the use of the PRS are:

⋆ S.H. Brooks: "Discussion of random methods for locating surface maxima". Operations Research 6 (1958), pp. 244–251.

⋆ L.A. Rastrigin: "The convergence of the random search method in the extremal control of a many-parameter system". Automation and Remote Control 24 (1963), pp. 1337–1342.

vector `fitness` that contains the sequence of the best-ever objective function values found so far. In other words, the i^{th} coordinate of the vector `fitness` contains the best function value found until iteration i. To sample, you might use the instructions `numpy.random.rand`, `numpy.round`, and `numpy.random.randint`.

4. Plot the evolution of the function value ("fitness") as a function of the number of function evaluations for two independent runs of the algorithm. What do you observe? You might want to use `matplotlib` for the plotting: check in particular the instructions `plot` and `show`.
5. Write the code to plot for $n = 2, \dots, 14$ the empirical expected value of the time needed to reach the optimum of f_{ONEMAX} with the PRS algorithm. Plot as well the standard deviation around the expected value. We advise to compute the empirical expected value and the standard deviation based on 11 independent runs of the algorithm.

Note: in a blackbox scenario, the time to reach the optimum is measured by counting the number of function evaluations needed to reach the optimum and not by the real wall-clock time. Indeed, internal operations are generally negligible compared to the cost of evaluating the objective function.

6. Compute theoretically the expected time to reach the optimum as a function of n. Compare the theoretical and empirical results. Hint: show that the time to reach the optimum follows a geometric distribution with a parameter to determine.

(1+1)-EA

We will now implement a simple evolutionary algorithm, the so-called (1+1)-EA. Its population is reduced to a single individual. The single parent (the first "1" in the notation (1+1)) mutates to give a single offspring (the second "1" in the notation (1+1)). The best among the offspring *and* the parent is kept for the next iteration (symbolized by the "+" in the notation (1+1)).

The mutation used is the so-called bit-flip mutation where each bit of the parent is changed with probability $1/n$ (and thus stays unchanged with a probability $1 - 1/n$).

(1+1)-EA to maximize $f : \{0,1\}^n \to \mathbb{R}$

```
Initialize uniformly at random $\boldsymbol{x} \in \Omega = \{0,1\}^n$
while not terminate
     Create $\boldsymbol{x}'$ by flipping each bit of $\boldsymbol{x}$ with probability $1/n$
     if $f(\boldsymbol{x}') \geq f(\boldsymbol{x})$
          $\boldsymbol{x} = \boldsymbol{x}'$
return $\boldsymbol{x}$
```

7. Follow again the steps 3, 4, and 5 above—now for the (1+1)-EA.
8. The theoretical complexity[2] of the expected time to reach the optimum is $\Theta(n \log n)$. Compare the theoretical and empirical results. Give an idea for the theoretical proof of the upper bound of $cn \log n$.
9. Explain the differences obtained between the PRS and the (1+1)-EA.

Optional: Behavior on Other Functions

We consider now the function `Needle in the Haystack` defined as follows.

[2]Reminder: $f(n) \in \Theta(g(n))$ if $\exists k_1, k_2,\ |g(n)| \cdot k_1 \leq |f(n)| \leq |g(n)| \cdot k_2$.

Example 2 *For $\boldsymbol{x} \in \{0,1\}^n$, the function* Needle *is defined as*

$$f_{\textsc{Needle}}(x) = \begin{cases} 1 & \textit{if } \boldsymbol{x} = (1, \ldots, 1) \\ 0 & \textit{otherwise} \end{cases}$$

10. What will be the performances of PRS and (1+1)-EA on the function $f_{\textsc{Needle}}$? Comment the differences w.r.t. the function $f_{\textsc{Onemax}}$.

11. Can other algorithms outperform the above algorithms on the Needle function?

11. What is the performance of the two investigated algorithms on the Zeromax function that aims at maximizing the number of zeros in the bit string instead of the number of ones?.