Nature Inspired Optimisation – 06-26949
Nature-Inspired Optimisation (Extended) – 06-26948

Course Work I (10%): Programming Assignment
Continuous Function Optimisation

Due date: 27th February 2014, 12:00 noon

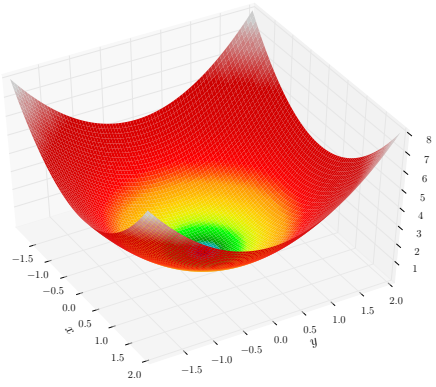
Introduction

This assignment aims to assess how you implement and experimentally compare different nature-inspired algorithms. It aims at giving you some first hands-on experience of applying a nature-inspired algorithm in a given context and drawing conclusions about some operators from a set of experimental results. We consider different mutation operators in the context of continuous function optimisation.

Task

Consider the following continuous benchmark function SPHERE:

$$\text{SPHERE}(x) = \sum_{i=1}^{n} x_i^2 \text{ with } -\infty < x_i < \infty$$



The SPHERE function has a unique global minimum in

$$\text{SPHERE}(x_1, \ldots, x_n) = \text{SPHERE}(0, \ldots, 0) = 0$$

and can be visualised for $n = 2$ as shown above. It is a suitable benchmark to investigate if and how an algorithm starting from some arbitrary point in the search space is able to converge to a local optimum, which is an important property for nature-inspired algorithms.

In the lecture you have learned about two mutation operators in the context of continuous function optimisation, uniform and non-uniform mutation. Here, we consider these and two additional operators, Gaussian mutation with and without 1/5-rule. You can find explanations and pseudocode for all four operators in the appendix of this assignment. Your task is to implement and compare the performance of these four operators on the SPHERE function.

Detailed Instructions

The assignment requires you to implement the above four operators (Algorithms 2, 3, 4 and 5 in the appendix) within a simple (1+1) evolution strategy only using mutation (see Algorithm 1 in the appendix) on the SPHERE function in a programming language of your choice (e. g., Java, C, C++, Matlab). You will then run a set of experiments as detailed below to assess the performance of the mutation operators.

Additionally, you are required to describe and discuss your implementation and your experimental findings in a report. The report should in particular discuss the following aspects (amongst any other points which you might consider important):

- How did you implement the algorithm and the fitness function (e. g., programming language, design decisions, ...)? What data structures, algorithms or libraries did you use, e. g., for a random number generator? What other software/scripts did you use to facilitate your experiments/analysis?
- How does the performance of the different operators compare? How do parameters of the operators/the fitness function influence the performance?
- Try to explain why the operators behave the way you observe in your experiments. What properties, strengths or weaknesses can you observe? Can you name reasons for them?

As a minimum you need to run experiments using the following parameter settings. You are free to test additional settings if you feel these will help you with the discussion. Make sure to include these in your report as well. Note that uniform mutation does not have any parameters.

- A 10-dimensional SPHERE function (i. e., $n = 10$) with search space restricted to $[-100, 100]^{10}$.
- Non-uniform mutation with parameter $b \in \{0.05, 1.0, 10.0, 20.0\}$.
- Gaussian mutation without 1/5-rule and parameter $\sigma \in \{0.05, 0.5, 1.0, 10.0\}$
- Gaussian mutation with 1/5-rule and initial σ uniformly distributed over $[1, 100]$.

You need to perform 30 independent runs for each of the above settings. Each run should have an upper limit of 5,000 iterations. For each of the runs you need to record the fitness value for each iteration of the run. Visualise and present your data as follows:

- Plot the average function value over time (i. e., iteration number on the x axis; average function value over the 30 trails on the y axis). Use logscale for the y axes to improve visibility. You should have such a graph for each of the following scenarios:
  - a figure comparing different parameters for non-uniform mutation
  - a figure comparing different parameters for Gaussian mutation without 1/5 rule
  - a figure comparing uniform mutation, Gaussian mutation with 1/5 rule and the 'best' (i. e., best fitness value after 5,000 iterations) parameter setting for non-uniform and Gaussian mutation without 1/5 rule
- Create a table stating the average fitness value and its standard deviation (over the 30 runs) after 5,000 iterations for each of the considered settings. You can determine these values with a statistics or spreadsheet software of your choice (e. g., R, Matlab, Microsoft Excel, LibreOffice).

Requirements and Submission

Submit your coursework through the Canvas system. In case you encounter any problems with the Canvas system, please email support.

Your submission should contain

- a **single pdf file** with your report and
- a **single zip file** containing your source code and raw data.

No late submissions are allowed.

1000–2000 words are appropriate for the report (excluding any figures, tables and the list of references), however, note that this is not a hard limit. You should add a declaration that the report and source code is entirely written by yourself, unless it is noted otherwise.

The implementation and report should be completed independently and any sources used need to be clearly cited in your report. You can use a reference system of your choice, but you must be consistent. Please refer to the University's guide to referencing: `https://intranet.birmingham.ac.uk/as/libraryservices/library/referencing/icite/referencing/index.aspx`

Please also refer to the School's guidance notes on plagiarism:

`http://www.cs.bham.ac.uk/internal/students/handbook/current/#PLAG`

Assignment Marks

Marking will be based on the quality of the implementation and the report in its content, structure and language. It will be split according to the following parts of the assignment:

- Implementation (40%, based on source code and report)
- Experiments and their presentation (40%, based on raw data and report)
- Discussion of findings (20%, based on report)

Mark descriptors will roughly follow those used in project marking:

`http://www.cs.bham.ac.uk/internal/staff/handbook/ProjectGradeDescriptors.html`

Appendix

This appendix gives you a summary of the algorithms you need to implement. The different mutation operators involve different types of random number generation. Uniform and non-uniform mutation use random numbers drawn from a uniform distribution over a given interval of real numbers. Gaussian mutation makes use of standard-normally distributed variables, i. e., normally distributed variables with mean 0 and standard deviation 1. There are libraries you can use to generate these random numbers for most programming languages. Please let us know in case there are any problems with this. Make sure you explain your choices in your report.

In this assignment we restrict the range of x_i to $[-100, 100]$ for all $1 \leq i \leq n$. For the sake of simplicity the following algorithms are formulated for this special case, only.

Algorithm

The (1+1) evolution strategy initialises an individual uniformly at random in the search space. It uses a population of size 1, creates 1 offspring in each iteration using mutation, only, and replaces the parent by the offspring if the offspring's fitness is strictly better, i. e., the offspring is an improvement. This is repeated for a given number T of iterations.

Algorithm 1: (1+1) evolution strategy

Input: Total number of iterations T

1 Create initial search point $x = (x_1, \ldots, x_n)$ by picking x_i uniformly at random from $[-100, 100]$ for all $i \in \{1, 2, \ldots, n\}$.
2 **repeat**
3 $\quad y :=$mutate(x)
4 $\quad$ **If** $f(y) < f(x)$ **then** $x := y$.
5 **until** T *times*;

Output: $x, f(x)$

Mutation Operators

Uniform mutation replaces a random component x_i in the search point by a new random value.

Algorithm 2: Uniform Mutation

Input: Search point $x = (x_1, \ldots, x_n)$

1 Select $i \in \{1, \ldots, n\}$ uniformly at random.
2 Replace x_i with a number x_i' generated uniformly at random in the range of $[-100, 100]$.

In non-uniform mutation a random component x_i is modified by adding or subtracting a random value. The mutation strength, i. e., the size of this change, decreases over time.

Gaussian mutation adds a normally-distributed random value to each component of the search point. The 'size' of this value is controlled by a parameter σ, which is usually called *step size*.

1/5-rule

The 1/5-rule is a method to adapt the parameter σ in the Gaussian mutation operator depending on the current state of the search process. The idea is to observe the success rate of mutations over some period of time. If the success rate is larger than 1/5 the step size σ is increased to allow for larger changes; if it is smaller than 1/5 the step size is decreased to restrict the search to smaller mutations.

Algorithm 3: Non-Uniform Mutation

Input: Search point $x = (x_1, \ldots, x_n)$; Current iteration number t; Total number of iterations T; Parameter b

Select $i \in \{1, \ldots, n\}$ uniformly at random.

Replace x_i by $x'_i := \begin{cases} x_i + \Delta(t, 100 - x_i) & \text{if } \tau \geq 0.5 \\ x_i - \Delta(t, x_i + 100) & \text{otherwise} \end{cases}$

with $\Delta(t, y) = y \cdot \left(1 - r^{1 - \frac{t}{T}}\right)^b$; $r \in [0, 1]$ uniformly at random

Algorithm 4: Gaussian Mutation

Input: Search point $x = (x_1, \ldots, x_n)$; Step size σ

for $i \in \{1, 2, \ldots, n\}$ **do**
    $x_i := x_i + \sigma \cdot m_i$ where $m_i \sim \mathcal{N}(0, 1)$, i. e., m_i is standard-normally distributed
end

Here, we use a period of n iterations and count the number of successful and unsuccessful mutations in this time frame. After n iterations the measured success rate is calculated by dividing the number of successful mutations by n. If this is larger than $1/5$ we double σ; if it is smaller σ is halved. Both counters are reset to zero afterwards.

Algorithm 5: (1+1) evolution strategy with 1/5-rule

Input: Total number of iterations T; Step size σ

Create initial search point $x = (x_1, \ldots, x_n)$ by picking x_i uniformly at random from $[-100, 100]$.
Initialise σ uniformly at random in $[1, 100]$.
Set $g := 0$ and $b := 0$. (# number of good/bad mutations, respectively)
repeat
    $y :=$mutate(x), Algorithm 4 in this case
    If $f(y) < f(x)$ **then** $x := y$; $g := g + 1$ **else** $b := b + 1$
    if $b + g = n$ **then**
        If $g/n > 1/5$ **then** $\sigma := \sigma \cdot 2$
        If $g/n < 1/5$ **then** $\sigma := \sigma/2$
        $b := 0$; $g := 0$
    end
until T *times*;
Output: $x, f(x)$