Bio-Inspired Artificial Intelligence

Prof. Giovanni Iacca
giovanni.iacca@unitn.it

Module 5–Lab Exercises

Introduction

Goal. The goal of this lab is to familiarize yourself with some of the constraints handling techniques used in Evolutionary Computation.

Getting started. Download the file `05.Exercises.zip` from Moodle and unzip it. This lab continues the use of the *inspyred* framework for the Python programming language seen in the previous labs. If you did not participate in the previous labs, you may want to look those over first and then start this lab's exercises.

Each exercise has a corresponding `.py` file. To solve the exercises, you will have to open, edit, and run these `.py` files.

Note once again that, unless otherwise specified, in this module's exercises we will use real-valued genotypes and that the aim of the algorithms will be to *minimize* the fitness function $f(\mathbf{x})$, i.e. lower values correspond to a better fitness!

Exercise 1

In this exercise we will continue the investigation of the multiple-disk clutch brake design problem we have seen in the previous lab. In this case, we will consider the full problem including a number of constraints $g_i(x)$, as defined in Figure 1. The constraints have been implemented for you in the provided `disk_clutch_brake.py`. Please note that the only difference with respect to the code we have seen in the previous lab is the activation of the constraints, obtained by setting the variable `constrained` to `True` in `exercise_1.py` (equivalent to `exercise_3.py` from the previous lab).

When constraints are enforced the notion of constrained-Pareto-domination comes into play. A solution i now is considered to dominate a solution j if any of the following conditions are true:

1. Solution i is feasible and solution j is not
2. Solutions i and j are both infeasible, but solution i has a smaller overall constraint violation.
3. Solutions i and j are feasible and solution i dominates solution j

$$
\begin{aligned}
\text{Minimize} \quad & f_1(\vec{x}) = \pi({x_2}^2 - {x_1}^2)x_3(x_5+1)\rho, \\
\text{Minimize} \quad & f_2(\vec{x}) = T = \tfrac{I_Z \omega}{M_h + M_f}, \\
\text{Subject to} \quad & g_1(\vec{x}) = x_2 - x_1 - \Delta R \geq 0, \\
& g_2(\vec{x}) = L_{\max} - (x_5+1)(x_3+\delta) \geq 0, \\
& g_3(\vec{x}) = p_{\max} - p_{rz} \geq 0, \\
& g_4(\vec{x}) = p_{\max} V_{sr,\max} - p_{rz} V_{sr} \geq 0, \\
& g_5(\vec{x}) = V_{sr,\max} - V_{sr} \geq 0, \quad g_6(\vec{x}) = M_h - sM_s \geq 0, \\
& g_7(\vec{x}) = T \geq 0, \quad g_8(\vec{x}) = T_{\max} - T \geq 0, \\
& r_{i,\min} \leq x_1 \leq r_{i,\max}, \quad r_{o,\min} \leq x_2 \leq r_{o,\max}, \\
& t_{\min} \leq x_3 \leq t_{\max}, \quad 0 \leq x_4 \leq F_{\max}, \quad 2 \leq x_5 \leq Z_{\max}.
\end{aligned}
$$

The parameters are given below:

$$
\begin{aligned}
& M_h = \tfrac{2}{3}\mu x_4 x_5 \tfrac{{x_2}^3 - {x_1}^3}{{x_2}^2 - {x_1}^2} \text{ N·mm}, \omega = \pi n/30 \text{ rad/s}, \\
& A = \pi({x_2}^2 - {x_1}^2) \text{ mm}^2, p_{rz} = \tfrac{x_4}{A} \text{ N/mm}^2, V_{sr} = \tfrac{\pi R_{sr} n}{30} \text{ mm/s}, \\
& R_{sr} = \tfrac{2}{3}\tfrac{{x_2}^3 - {x_1}^3}{{x_2}^2 - {x_1}^2} \text{ mm}, \Delta R = 20 \text{ mm}, L_{max} = 30 \text{ mm}, \mu = 0.5, \\
& p_{\max} = 1 \text{ MPa}, \rho = 0.0000078 \text{ kg/mm}^3, V_{sr,\max} = 10 \text{ m/s}, \\
& s = 1.5, T_{max} = 15 \text{ s}, n = 250 \text{ rpm}, M_s = 40 \text{ Nm}, \\
& M_f = 3 \text{ Nm}, I_z = 55 \text{ kg·m}^2, \delta = 0.5 \text{ mm}, r_{i,\min} = 60 \text{ mm}, \\
& r_{i,\max} = 80 \text{ mm}, r_{o,\min} = 90 \text{ mm}, r_{o,\min} = 110 \text{ mm}, \\
& t_{\min} = 1.5 \text{ mm}, t_{\max} = 3 \text{ mm}, F_{\max} = 1,000 \text{ N}, Z_{\max} = 9.
\end{aligned}
$$

Figure 1: Complete problem formulation of the multiple-disk clutch brake problem.

To start the experiments, from a command prompt run[1]:

```
$>python exercise_1.py
```

As in the previous lab, the final population and fitness values are saved on a file `exercise_1.csv` $\{r_i, r_o, t, F, Z, mass, time\}$, one line for each solution in the Pareto front. Also in this case, you may want to try plotting these data in different ways to gain further insights.

- How do your results change from the unconstrained version (from the previous lab)?
- Do your previous parameters continue to solve the problem?
- Try to increase the population size and/or the number of generations to see if you can find better solutions.

Exercise 2

In this exercise we will test the Genetic Algorithm we used in Lab 2 for solving a set of constrained optimization benchmark functions. In this case we will consider five benchmark problems from the Wikipedia page on "Test functions for constrained optimization"[2], plus an additional sphere

[1] For all the exercises in this lab you may follow the name of the `.py` file with an integer value, which will serve as the seed for the pseudo-random number generator. This will allow you to reproduce your results. Also, please note that in this document `$>` represents your command prompt, do not re-type these symbols.

[2] `https://en.wikipedia.org/wiki/Test_functions_for_optimization#Test_functions_for_constrained_optimization`

function with a constraint. We will limit the experiments only on two dimensions, to visualize the fitness landscape.

Open `exercise_2.py`. Try at least one or two of the following benchmark functions:

1. `RosenbrockCubicLine`
2. `RosenbrockDisk`
3. `MishraBirdConstrained`
4. `Townsend`
5. `Simionescu`

You can change the problem by changing the parameter `args['problem_class']` in the script `python exercise_2.py`. By default, the constraints are ignored by the GA. In order to set the GA to handle the constraints, set the variable `usePenalty=True` in `constrained_benchmarks.py`.

- Do you see any difference in the GA's behavior (and results) when the penalty is enabled or disabled?
- Try to modify the penalty functions used in the code of each benchmark function (check the code corresponding to `if usePenalty`), and/or change the main parameters of the GA (`max_generations`, `pop_size`, `gaussian_stdev`, `mutation_rate`, `tournament_size`, `num_elites`) in `exercise_2.py`. Are you able to find the optimum on all the benchmark functions you tested?

Now, analyze the benchmark `SphereCircle` (look at the code in `constrained_benchmarks.py`). In this case we are **maximizing** the 2-d sphere function we have already seen in the previous labs ($f(x) = x_1^2 + x_2^2$), subject to the constraint:

$$g_1(x) = x_1^2 + x_2^2 \leq 1 \longrightarrow g_1(x) = x_1^2 + x_2^2 - 1 \leq 0$$

Here, candidates solutions represent ordered pairs and their fitness is simply their distance from the origin. However, the constraint punishes solutions that lie *outside* the unit circle. Such a scenario should produce an optimum that lies on the unit circle. By default, the code penalizes candidate solutions outside the unit circle by assigning them a fitness value equal to -1.

- Is the GA able to find the optimal solution lying on the unit circle? If not, try to change some of the GA's parameters to reach the optimum.
- By default, the sphere function is defined in a domain $[-5.12, 5.12]$ along each dimension. Try to increase the search space[3] to progressively increasing boundaries (e.g. $[-10, 10]$, $[-20, 20]$, etc.). Is the GA still able to explore the feasible region and find the optimum?
- If not, try to think of a way to guide the GA towards the feasible region. How could you change the penalty function to do so? (Hint: look at the `evaluator` method of the class `SphereCircle` and consider that we are maximizing the fitness function, while we want to minimize the violation given by $g_1(x)$.)

Finally, you can create your own constrained optimization problem by modifying the class template `SphereConstrained` you will find in `constrained_benchmarks.py`.

- Try to modify the sphere function problem by adding one or more linear/non-linear constraints, and analyze how the optimum changes depending on the presence of constraints.

[3] To do so, change `self.bounder` and `generator` in the class `SphereCircle`.

Instructions and questions

Concisely note down your observations from the previous exercises (follow the bullet points) and think about the following questions.

- What do you think is the most efficient way to handle constraints in EAs?
- Do you think that the presence of constraints makes the search *always* more difficult? Can you think of cases in which the constraints could actually make the search easier?

BONUS: If you have time, you can try to replicate (part of) the experiments from exercise 2, this time using Evolution Strategies (as seen in Lab 3), instead of Genetic Algorithm. Open `exercise_3.py` and follow the same steps from exercise 2.

- Do you see any difference in performance between GA and ES? Why?