Real-coded GA and constraint handling

Shan He

School for Computational Science
University of Birmingham

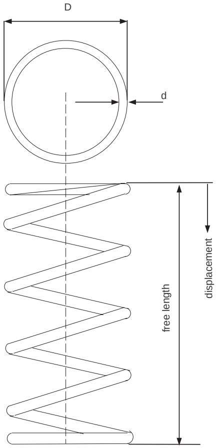
Module 06-27818 and 27819: Advanced Aspects of Nature-Inspired Search and Optimisation (Ext)

Outline of Topics

1. Continuous Optimisation problems
2. The Real-coded Genetic Algorithm
3. Constrained Engineering Optimisation Problems
4. Constrained Optimisation Techniques
    - Penalty Function
    - Stochastic Ranking
    - Feasibility Rules

Motivating example 1: engineering optimisation

- We aim to design a tension/compression spring with minimum weight by optimising the following continuous design variables: Wire diameter d , mean coil diameter D and number of active coils N

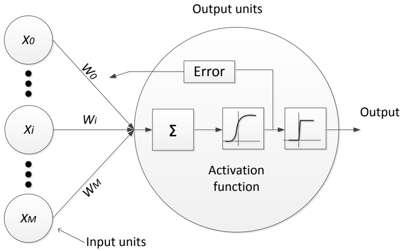


Motivating example 2: Honda Car design

- **Aim**: to optimise the structural and aerodynamic properties of a car.
- The variables are usually continuous
- Link to Honda's page
- Honda also used evolutionary algorithm to design gas turbine stator blades

Motivating example 3: neural network training

- **Aim**: to minimise a cost function (such as Sum Squared Error) by optimising the weights w_{ij} based on training data.
- The weights w_{ij} are usually continuous



Questions

We have implemented the Binary GA with binary point mutation and one point crossover to solve some combinatorial optimisation problems.

- **Question**: can you apply this BGA to solve continuous optimisation problems?

Solving continouous optimisation using GAs

- **A suitable representation of solutions to the problem: each solution is called an individual**
- A way to explore the space of solutions: variation operators
- A way to guide the algorithm to find better solutions (exploitation): selection and reproduction

Genetic Representation

- The selection of representation depends on the problem
- We have the following choices:
  - **Binary representation**
  - **Real number representation**
  - **Random key representation**
  - Permutation representation
  - Other problem specific representations

Generic Genetic Algorithm

Generic Genetic Algorithm

```
$\mathbf{X}_0$ := generate initial population of solutions
terminationflag := false
t := 0
Evaluate the fitness of each individual in $\mathbf{X}_0$.
while (terminationflag != true)
        Select parents from $\mathbf{X}_t$ based on their fitness.
        Breed new individuals by applying crossover and mutation to parents
        Evaluate the fitness of new individuals.
        Generate population $\mathbf{X}_{t+1}$ by replacing least-fit individuals
        t := t + 1
        If a termination criterion is met: terminationflag := true
Output $x_{best}$
```

Gaussian Mutation

Gaussian Mutation

```
Input: An individual $\mathbf{x} = \{x_1, x_2, \cdots, x_n\} \in \mathbb{R}^n$
$L_i$ and $U_i$ be the lower and upper bounds of the variable $x_i$
For $i \in 1, 2, \cdots, n$ do
        $y_i = \min(\max(\mathcal{N}(x_i, \sigma), L_i), U_i)$
Output: $\mathbf{y} = \{y_1, y_2, \cdots, y_n\} \in \mathbb{R}^n$ as the mutated individual
```

Note: $\sigma = (U_i - L_i) \times m$, where $m \in (0, 1]$ and a typical value is set to 0.01.

Arithmetic crossover

Arithmetic crossover

Input: two parents $\mathbf{x}_1$ and $\mathbf{x}_2$ and 2 uniformly distributed random numbers α_1 and $\alpha_2 = 1 - \alpha_1$

Output: two new individuals:

$$\mathbf{y}_1 = \alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2$$
$$\mathbf{y}_2 = \alpha_2\mathbf{x}_1 + \alpha_1\mathbf{x}_2$$

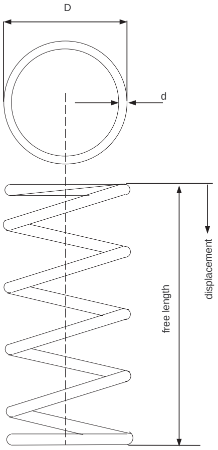
Note: Individual $\mathbf{x} = \{x_1, x_2, \cdots, x_n\} \in \mathbb{R}^n$ is a n dimensional real vector

Code example and exercise 1: Real-coded GA (20 mins)

- Open `RealGA1.m` and I will explain the code
- Task 1: Comment out the code for visualisation. Then execute the Real-coded GA to solve the 2-D Ackley function. The population is set to 50 and maximum iteration is set to 100. Execute your GA for 30 runs. Record the results and calculate the mean and standard deviation of your results.
- Task 2: Implement the arithmetic crossover operator and try Task 1 again.
- Fun staff: you can try some competition such as Black Box Optimization Competition

Constrained Optimisation Example: Spring design

- **Objective**: We aim to design a tension/compression spring with minimum weight by optimising some continuous design variables. However we need to satisfy a set of constraints:
- **Constraints**:
    - Minimum deflection
    - Shear stress
    - Surge frequency
    - Diameters



What Is Constrained Optimisation?

- The general constrained optimisation problem:

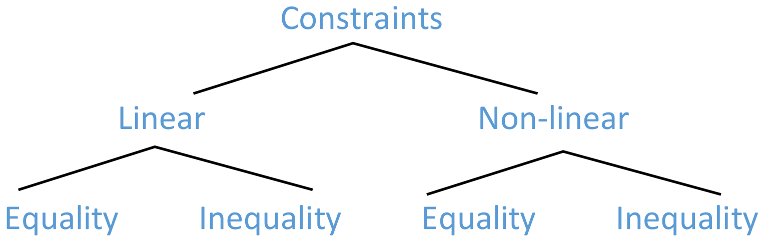
$$\min_{\mathbf{x}}\{f(\mathbf{x})\}$$

  subject to

$$g_i(\mathbf{x}) \leq 0,\ i = 1, \cdots, m$$
$$h_j(\mathbf{x}) = 0,\ j = 1, \cdots, p$$

  where $\mathbf{x}$ is the n dimensional vector, $x = (x_1, x_2, \cdots, x_n)$; $f(\mathbf{x})$ is the objective function; $g_i(\mathbf{x})$ is the inequality constraint; and $h_j(\mathbf{x})$ is the equality constraint.
- Denote the whole search space as $\mathcal{S}$ and the feasible space as $\mathcal{F}$, $\mathcal{F} \in \mathcal{S}$.
- Note: the global optimum in $\mathcal{F}$ might not be the same as that in $\mathcal{S}$.

Different Types of Constraints

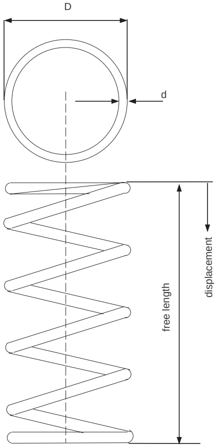


- Linear constraints are relatively easy to deal with
- Non-linear constraints can be hard to deal with

Constrained Optimisation Example: Spring design

- **Objective**: minimise the weight of a tension/compression spring. All design variables are continuous.
- **Constraints**:
  - Minimum deflection
  - Shear stress
  - Surge frequency
  - Diameters
- **Design variables**: Wire diameter d , mean coil diameter D and number of active coils N
- There are boundaries of design variables:

$$0.05 \leq d \leq 2,\ 0.25 \leq D \leq 1.3,\ 2 \leq N \leq 15$$



Constrained Optimisation Example: Spring design

- Minimize

$$f(X) = (N+2)Dd^2 \tag{1}$$

subject to:

$$\begin{aligned} g_1(X) &= 1 - \frac{D^3N}{71785d^4} \leq 0 && (2) \\ g_2(X) &= \frac{4D^2 - dD}{12566(Dd^3 - d^4)} + \frac{1}{5108d^2} - 1 \leq 0 && (3) \\ g_3(X) &= 1 - \frac{140.45d}{D^2N} \leq 0 && (4) \\ g_4(X) &= \frac{D+d}{1.5} - 1 \leq 0 && (5) \end{aligned}$$

Code example and exercise 2: implementing the spring desgin problem

- Open (`SpringDesign.m`) and I will explain the code
- Test this function function with $d = 0.05169040$, $D = 0.35674999$ and $N = 11.28712599$. The output should be consistent with the results in my paper, i.e., the objective function value is 0.0126652812 and all constraints are not larger than 0.

How to solve the problem: Constraint Handling Techniques

- **The purist approach**: rejects all infeasible solutions in search. Known as dead penalty method.
- **The separatist approach**: considers the objective function and constraints separately.
- **The penalty function approach**: converts a constrained problem into an unconstrained one by introducing a penalty function into the objective function.
- **The repair approach**: maps (repairs) an infeasible solution into a feasible one.
- **The hybrid approach** mixes two or more different constraint handling techniques.

Penalty Function Approach: Introduction

- $f'(\mathbf{x})$ (New Objective Function) $=$ $f(x)$ (Original objective function) $+$ Penalty factor * Degree of constraint violation
- The general form of the penalty function approach:

$$f'(\mathbf{x}) = f(\mathbf{x}) + \left(\sum_{i=1}^{m} r_i G_i(\mathbf{x}) + \sum_{j=1}^{p} c_j H_j(\mathbf{x}) \right)$$

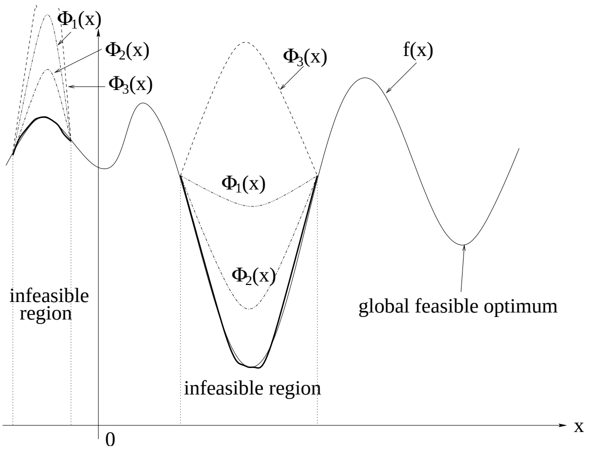
where $f'(\mathbf{x})$ is the new objective function to be minimised, $f(\mathbf{x})$ is the original objective function, r_i and c_j are penalty factors (coefficients), and G_i and H_j are penalty functions:

$$G_i(\mathbf{x}) = \max(0, g_i(\mathbf{x}))^{\beta},\ H_j(\mathbf{x}) = \max(0, |h_j(\mathbf{x})|^{\gamma})),$$

where β and γ are usually chosen as 1 or 2.

Penalties and Fitness Landscape Transformation

- Different penalty functions lead to different new objective functions.



Exercise 3: Constrained genetic algorithm with penalty function (15 mins)

- Use the real-coded genetic algorithm in the downloaded folder (`ConstrainedRealGA.m`) to solve the spring design problem for 30 independent runs with 50 individuals and 200 generations. Try different values of penalty factor, e.g., 0.1 and 0.01.

Stochastic Ranking

- Proposed by Dr Runarsson and Prof. Xin Yao in our school in 2000. Paper is here.
- A **special rank-based selection scheme** that handles constraints
- There is no need to use any penalty functions
- It's self-adaptive since there are few parameters to set
- Became the one of the popular constraint handling techniques due to its effectiveness and simplicity

Ranking Selection

- Sort population size of μ from **best** to **worst** according to their fitness values:

$$x'_{(\mu-1)}, x'_{(\mu-2)}, x'_{(\mu-3)}, \cdots, x'_{(0)},$$

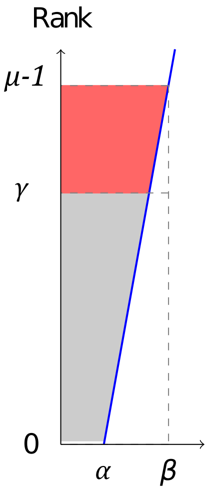
- Select the top γ-ranked individual x'_γ with probability $p(\gamma)$, where $p(\gamma)$ is a ranking function, e.g.
  - linear ranking
  - exponential ranking
  - power ranking
  - geometric ranking

Linear ranking function

- **Linear ranking function:**

$$p(\gamma) = \frac{\alpha + (\beta - \alpha) \cdot \frac{\gamma}{\mu - 1}}{\mu}$$

  where $\sum_{\gamma=0}^{\mu-1} p(\gamma) = 1$ implies $\alpha + \beta = 2$ and $1 \leq \beta \leq 2$
- In expectation
  - best individual reproduced β times
  - worst individual reproduced α times



Penalty Functions Demystified

- Penalty function essentially performs:
  - Fitness (objective) function transformation
  - Rank change $\longrightarrow$ selection change
- Why not change the **rank** directly in an EA?
- Ranking $=$ sorting: we can modify sorting algorithm in order to take constraint violation into consideration.

Stochastic Ranking

Stochastic Ranking Algorithm (based on bubble sort)

```
for $j := 1$ to $\mu$ do
    for $i := 2$ to $\mu$ do
            $u := U(0;1)$, // $u$ is a uniformly distributed random number;
            if $G(x_i) = G(x_{i-1}) = 0$ OR $u \leq P_f$ then
            // We then swap them so that the better (smaller) one is before
            // the worse one
                    if $f(x_i) < f(x_{i-1})$ then
                        swap($I_i$, $I_{i-1}$);
                        swap($f(x_i)$, $f(x_{i-1})$);
                        swap($G(x_i)$, $G(x_{i-1})$);
            else // which means $G(x_{i-1}) > 0$ and $u > P_f$
                    if $G(x_i) < G(x_{i-1})$ then //only compare constraint violations
                        swap($I_i$; $I_{i-1}$);
                        swap($f(x_i)$, $f(x_{i-1})$);
                        swap($G(x_i)$, $G(x_{i-1})$);
```

μ is the number of individuals, $I = 1, \cdots, \mu$ is the indices of the individuals, $G(\cdot)$ is the sum of constraint violation and P_f is a constant that indicates the probability of using the objective function for comparison in ranking.

The role of P_f

- $P_f > 0.5$:
    - Most comparisons are based on $f(x)$ only
    - Infeasible solutions are likely to occur
- $P_f < 0.5$:
    - Most comparisons are based on $G(x)$ only
    - Infeasible solutions are less like to occur, but the solutions might be poor
- As recommended in the paper, P_f is often set between 0.45 and 0.5

Code example: implementing the stochastic ranking

- Open (`bubblesort_standard.m`) and I will explain the code
- Open (`stochastic_ranking_sort_template.m`) and I will explain what you need to do.

Penalty Functions Demystified

- Penalty function essentially performs:
  - Fitness transformation
  - Rank change $\longrightarrow$ selection change
- Why not change **selection** directly in an EA?

Feasibility rule

- Proposed by Deb in 2000
- Based on (binary) tournament selection
- After randomly choose k $(k = 2)$ individuals to form a tournament, apply the following rules:
    - Between 2 feasible solutions, the one with better fitness value wins
    - Between a feasible and an infeasible solutions, the feasible one wins
    - Between 2 infeasible solutions, the one with lowest G wins
- Pros: simple and parameter free
- Cons: premature convergence