Genetic Algorithm

Issues in Design Optimization:

The *potential objective functions for many engineering* systems are based on:
cost, weight, volume, mass, stress, performance, reliability of a system, among others.

The constraints can be placed on:
resources, stresses, forces, pressure, temperature, strains, displacements, velocity, current flow, voltage, frequencies, manufacturing limitations, and other performance criteria.

Genetic Algorithm

Issues in Design Optimization:

The size of the problem:
Number of decision variables, constraints and complexity of the model.

In case the number of equality constraints is greater than the number of design variables in a problem, *no solution will exist* unless some of the constraints are dependent.

Genetic Algorithm

Issues in Design Optimization:

Design variables:
To start with, all the possible parameters or unknowns should be viewed as potential design variables which should be independent of each other as far as possible.

As we gain more knowledge about the problem, redundant or unnecessary design variables can be fixed or eliminated from the model.

Genetic Algorithm

Issues in Design Optimization:

The idea of *design variable linking is useful to reduce the number of design* variables in an optimization model.

If one of the design variables can be expressed in terms of others, then that variable can be eliminated from the model.

In engineering design problems, *lower and upper limits on the design variables are* often imposed as a result of practical limitations.

Genetic Algorithm

Issues in Design Optimization:

Sensitivity of design variables is important factor

In general, *it is desirable to normalize all the constraints with respect to the objective function or their limit values*

Penalty and Barrier Methods

General classical constrained minimization problem

$$\begin{aligned} & \text{minimize } f(\boldsymbol{x}) \\ \text{subject to} \quad & g(\boldsymbol{x}) \leq 0 \\ & h(\boldsymbol{x}) = 0 \end{aligned}$$

Penalty methods are motivated by the desire to use unconstrained optimization techniques to solve constrained problems.

This is achieved by either

- adding a penalty for infeasibility and forcing the solution to feasibility and subsequent optimum, or
- adding a barrier to ensure that a feasible solution never becomes infeasible.

Genetic Algorithm

Penalty and Barrier Methods:

- minimize objective as unconstrained function
- provide penalty to limit constraint violations
- magnitude of penalty varies throughout optimization
- sequential unconstrained minimization techniques

$$F(\boldsymbol{x}, r_p) = f(\boldsymbol{x}) + r_p\, P(\boldsymbol{x})$$

$f(\boldsymbol{x})$: original objective function

$P(\boldsymbol{x})$: imposed penalty function

r_p : scalar multiplier to determine penalty magnitude

p : unconstrained minimization number

Genetic Algorithm

Penalty and Barrier Methods:

$$F(\boldsymbol{x}, r_p) = f(\boldsymbol{x}) + r_p\, P(\boldsymbol{x})$$

$$P(\boldsymbol{x}) = \sum_{j=1}^{n} \left(\max\left[0, g_j(\boldsymbol{x})\right]\right)^2 + \sum_{k=1}^{m} [h_k(\boldsymbol{x})]^2$$

Notes:

- if all constraints are satisfied, then $P(\boldsymbol{x}) = 0$
- penalty parameter, r_p starts as a small number but can increase with number of iterations.
- if r_p is small $F(\boldsymbol{x}, r_p)$ is easy to minimize but yields large constraint violations
- if r_p is large, constraints are all nearly satisfied but the function is numerically ill-conditioned

Optimization summary

1. *Problem definition or statement*
2. *Information and data collection*
3. *Definition of design or decision variables*
4. *Definition and formulation: Objective function*
5. *Definition and formulation: Constraints*
6. *Decision: Optimization method*
7. *Solution: Decision, Objective function, Constraints*

Optimization summary

1. *Problem definition or statement*
2. *Information and data collection*
3. *Definition of design or decision variables*
4. *Definition and formulation: Objective function*
5. *Definition and formulation: Constraints*
6. *Decision: Optimization method*
7. *Solution: Decision, Objective function, Constraints*

Optimization summary: Assignment 5

1. *Problem definition or statement*
   How much production of: *Shortening*
   Salad oil
   Margarine
   Maximize the profit
2. *Information and data collection*
3. *Definition of design or decision variables*
4. *Definition and formulation: Objective function*
5. *Definition and formulation: Constraints*
6. *Decision: Optimization method*
7. *Solution: Decision, Objective function, Constraints*

Material:

250,000 kg of soybean oil,
110,000 kg of cottonseed oil, and
2,000 kg of milk base substances.

Losses

10 percent for shortening,
5 percent for salad oil, and
no loss for margarine.

Min prod:

100,000 kg of shortening,
50,000 kg of salad oil, and
10,000 kg of margarine.

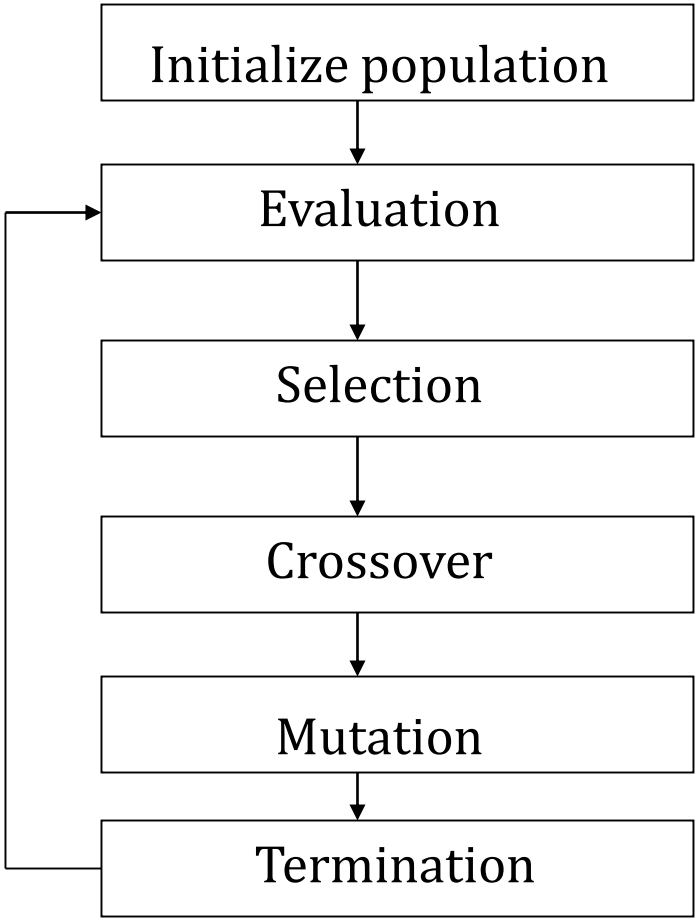
Income, requirements

Product	Profit per kg USD	Parts per kg of base stock requirements Soybean	Cottonseed	Milk base
Shortening	0,10	2	1	0
Salad oil	0,08	0	1	0
Margarine	0,05	3	1	1

Optimization summary: Assignment 5

```
from scipy.optimize import linprog
# Objective function
maxf = [-1.0, -0.8, -0.5]
# Constraints
A = [[2/3/0.9, 0., 3/5], [0.0, 0.0, 1/5], [1/3/0.9, 1/0.95, 1/5]]
b = [250000, 2000, 110000]
x1_b = (100000, None)
x2_b = (50000, None)
x3_b = (10000, None)
boundvalues = [x1_b, x2_b, x3_b]
res = linprog(maxf, A_ub=A, b_ub=b, bounds=boundvalues)
print("Profit: ",-1.0*res.fun)
print("Production: ", res.x)
print(res)
```

Genetic Algorithm



Encoding

*The process of representing the solution in the form of a **string** that conveys the necessary information.*

Just as in a chromosome, each gene controls a particular characteristic of the individual, similarly, each bit in the string represents a characteristic of the solution.

Representation

- When applying a GA to a problem one of the decisions we have to make is how to represent the problem
- The classic approach is to use bit strings and there are still some people who argue that unless you use bit strings then you have moved away from a GA
- Bit strings are useful as
  - How do you represent and define a neighbourhood for real numbers?
  - How do you cope with invalid solutions?
- Bit strings seem like a good coding scheme ***if*** we can represent our problem using this notation

Encoding Methods: Variables - Chromosomes / bits:

- All kind of alphabets can be used for a chromosome (numbers, characters….), but often a binary alphabet is used.
  Binary – Gray – Real – Integer – Permutation (Sequence)

- Decision variables (Genes) are concatenated to form a string (chromosome).
- Order of genes on chromosome can be important
- Coding is one of the most important factor for the performance of a GA.
- Code for feasible solutions and global optimum solution

Genetic Algorithm

Initialize population - Binary and Gray coding

Number of bits:

```
nbi = (np.ceil(np.log((xmax-xmin)/xres+1)/np.log(2))).astype(int)
```

M

```
nbi = ceil( log((xmax - xmin)/xres+1)/log(2));
```

First population:

```
Bpop = np.zeros((npop,nbi))
Bpop = np.around(np.random.random(size=(Bpop.shape)))
```

M

```
Bpop(npop,nch) = round(rand(npop,nch));
```

Genetic Algorithm

Initialize population - Binary and Gray coding

```
import numpy as np
xmax = 9
xmin = 0
xres = 0.1
npop = 10
nbi = (np.ceil(np.log((xmax-xmin)/xres+1)/np.log(2))).astype(int)
Bpop = np.zeros((npop,nbi))
Bpop = np.around(np.random.random(size=(Bpop.shape)))
print(Bpop)
```

```
[[0 1 1 0 0 0 1]
 [1 0 1 0 0 0 1]
 [1 1 1 0 1 0 1]
 [1 1 1 1 0 0 0]
 [0 1 1 1 0 0 1]
 [1 0 0 1 0 0 0]
 [0 1 0 0 1 0 1]
 [0 1 0 0 1 0 0]
 [1 1 0 0 1 0 0]
 [0 0 0 0 1 1 0]]
```

Genetic Algorithm

Encoding Methods: Variables - Chromosomes / bits:

Number of variables: n_v

Variables: $x_1, x_2, x_3, \ldots . x_{nv}$

Each variable: $x_{imin} \leq xi \leq x_{imax} \quad and\ x_{ires}$

Binary

Number of chromoshomes(bits):

$$\text{xmax} = [9, 9]$$
$$\text{xmin} = [0, 0]$$
$$\text{xres} = [0.1, 0.1]$$
$$\text{npop} = 10$$

$$\text{nbi} = (\text{np.ceil}(\text{np.log}(\text{np.subtract}(\text{xmax}, \text{xmin})/\text{xres} + 1)/\text{np.log}(2))).\text{astype}(\text{int})$$

Total number of bits: $nbi_t = \sum_1^{nv} nbi_i$

Encoding Methods: Gray to Binary to Real (Integer)

Binary number B: 1001110110 Gray number: G: 110101101

Gray to Binary

$$B_{nbi} = G_{nbi}$$

$$B_{nbi-1} = XOR(B_{nbi}, G_{nbi-1})$$

$$B_{nbi-2} = XOR(B_{nbi-1}, G_{nbi-2})$$

$$G_i = XOR(B_{i+1}, B_i)$$

	B	G
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111

Number $k_i = \sum_{j=0} B_{(nchi-j)}\ 2^j$

Real $x_i = x_{imin} + (x_{imax} - x_{imin})\ k_i / (2^{nchi} - 1)$

Genetic Algorithm

Coding – Initialize population

Integer variable representation 5

```
Ipop = np.array([Imin + (Imax-Imin)*np.random.random(size=(Imax.shape))]).astype(int)
```

Real variable representation 3.567

```
Rpop = np.array([Rmin + (Rmax-Rmin)*np.random.random(size=(Rmax.shape))])
```

Permutation – order representation 412563

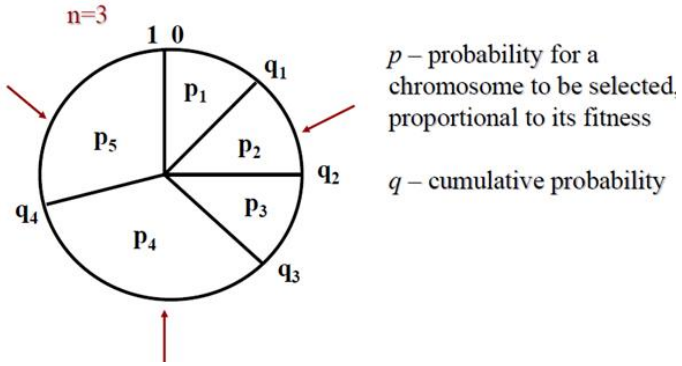
```
Ppop = np.zeros((npop,npvar))
Ppop = np.argsort(np.random.random(size=(Ppop.shape)))
```

Genetic Algorithm

GA Parent Sampling and Selection

Sampling space:

- Size : npop
- Parents
- Parents + offsprings



Selection mechanism:

- Roulette wheel selection (Selection proportional to fitness)
- Stochastic Universal Sampling. “
- Truncation selection (All individuals are sorted)
- Ranking selection
- Tournament selection (The fittest one in group is selected)

Genetic Algorithm

Roulette Wheel selection:

The most common fitness-proportionate selection technique is called *Roulette Wheel Selection*. Conceptually, each member of the population is allocated a section of an imaginary roulette wheel. Unlike a real roulette wheel the sections are different sizes, proportional to the individual's fitness, such that the fittest candidate has the biggest slice of the wheel and the weakest candidate has the smallest. The wheel is then spun and the individual associated with the winning section is selected. The wheel is spun as many times as is necessary to select the full set of parents for the next generation.

Using this technique it is possible (probable) that one or more individuals is selected multiple times. That's OK, it's what we want to happen. Remember that we are not selecting the members of the next generation, we are selecting their parents and it is possible for an individual to be a parent multiple times. If there is a particularly fit member of the population we would expect it to be more successful at producing offspring than a weaker rival.

Genetic Algorithm

Stochastic Universal Sampling:

Stochastic Universal Sampling is an elaborately-named variation of roulette wheel selection. Stochastic Universal Sampling ensures that the observed selection frequencies of each individual are in line with the expected frequencies. So if we have an individual that occupies 4.5% of the wheel and we select 100 individuals, we would expect on average for that individual to be selected between four and five times. Stochastic Universal Sampling guarantees this. The individual will be selected either four times or five times, not three times, not zero times and not 100 times. Standard roulette wheel selection does not make this guarantee.

Stochastic Universal Sampling works by making a single spin of the roulette wheel. This provides a starting position and the first selected individual. The selection process then proceeds by advancing all the way around the wheel in equal sized steps, where the step size is determined by the number of individuals to be selected. So if we are selecting 30 individuals we will advance by 1/30 x 360 degrees for each selection. Note that this does not mean that every candidate on the wheel will be selected. Some weak individuals will have very thin slices of the wheel and these might be stepped over completely depending on the random starting position.

Genetic Algorithm

Truncation selection:

Truncation selection is the simplest and arguably least useful selection strategy. Truncation selection simply retains the fittest x% of the population. These fittest individuals are duplicated so that the population size is maintained.

For example, we might select the fittest 25% from a population of 100 individuals. In this case we would create four copies of each of the 25 candidates in order to maintain a population of 100 individuals.

This is an easy selection strategy to implement but it can result in premature convergence as less fit candidates are ruthlessly culled without being given the opportunity to evolve into something better. Nevertheless, truncation selection can be an effective strategy for certain problems.

Genetic Algorithm

Rank selection:

Rank Selection is similar to fitness-proportionate selection except that selection probability is proportional to relative fitness rather than absolute fitness. In other words, it doesn't make any difference whether the fittest candidate is ten times fitter than the next fittest or 0.001% fitter. In both cases the selection probabilities would be the same; all that matters is the ranking relative to other individuals.

Rank selection will tend to avoid premature convergence by tempering selection pressure for large fitness differentials that occur in early generations. Conversely, by amplifying small fitness differences in later generations, selection pressure is increased compared to alternative selection strategies.

Genetic Algorithm

Turnament selection:

Tournament Selection is among the most widely used selection strategies in evolutionary algorithms. It works well for a wide range of problems, it can be implemented efficiently, and it is amenable to parallelisation.

At its simplest tournament selection involves randomly picking two individuals from the population and staging a tournament to determine which one gets selected.

The "tournament" isn't much of a tournament at all, it just involves generating a random value between zero and one and comparing it to a pre-determined selection probability. If the random value is less than or equal to the selection probability, the fitter candidate is selected, otherwise the weaker candidate is chosen. The probability parameter provides a convenient mechanism for adjusting the selection pressure. In practise it is always set to be greater than 0.5 in order to favour fitter candidates. The tournament can be extended to involve more than two individuals if desired.