OUTLINE

- WHAT ARE EVOLUTIONARY ALGORITHMS
- HOW CLASSICAL GENETIC ALGORITHMS WORK
- FROM GA TO DIFFERENTIAL EVOLUTION
- PARTICLE SWARM OPTIMIZATION

Darwin's core statement in his own language (1859)

"As many more individuals are produced than can possibly survive, there must in every case be a struggle for existence, either one individual with another of the same species, or with the individuals of distinct species, or with the physical conditions of life… . Can it, then, be thought improbable, seeing that variations useful to man have undoubtedly occurred, that other variations, useful in some way to each being in the great and complex battle of life, should sometimes occur in the course of thousands of generations? If such do occur, can we doubt (remembering that many more individuals are born than can possibly survive) that individuals having any advantage, however slight, over others, would have the best chance of surviving and of procreating their kind? On the other hand, we may feel sure that any variation in the least degree injurious would be rigidly destroyed. This preservation of favorable variations and the rejection of injurious variations, I call Natural Selection."

WHAT ARE EVOLUTIONARY ALGORITHMS?

- Application of the principle of Natural Selection – *The Survival of the Fittest* – to the problem of Optimization.
- HOW?

Darwin was unaware of Genetics

- Darwin was unaware of the existence of genes, chromosomes or DNA
  - However, he with his team was aware that parents somehow pass on their physical characteristics to children
- Gregory Mendel, Austrian monk (1866), postulated the theory that small elements in the cells of parents codify these physical characteristics and pass them on to children as pairs – one from each parent
- His theory became popular only after 1900 when scientists actively starting researching into and developing this field
- Now Gregor Mendel is known as the father of Genetics.

From Natural Selection of Species – through Natural Selection of Chromosomes – to Natural Selection of Optimal Solutions

- It was not until the 1960's that people started thinking –
- Can not the genetic techniques that enable the natural evolution of biological species …..
- ….. be applied to the natural evolution of optimal solutions ….
- if these artificial optimal solutions can somehow be expressed as chromosomes and genes?
- In the late 1960's and early 70's , Ingo Rechenberg in Germany and John Holland in USA independently came up with methods for the artificial evolution of optimization problems
- Rechenberg came up with Evolutionary Strategies and Holland with Genetic Algorithms
- Other techniques were also invented that broadly come under the umbrella of Evolutionary Algorithms.

Biologically Inspired Algorithms

- There is an important class of Algorithms in Artificial Intelligence that are known as *Biologically Inspired Algorithms*
- They Include:
  - Evolutionary Algorithms
  - Swarm Algorithms (also known as *Socially Inspired Algorithms*)
  - Fuzzy Logic
  - Artificial Neural Networks
- They comprise an important subset of AI called *Computational Intelligence* and also *Soft Computing*.

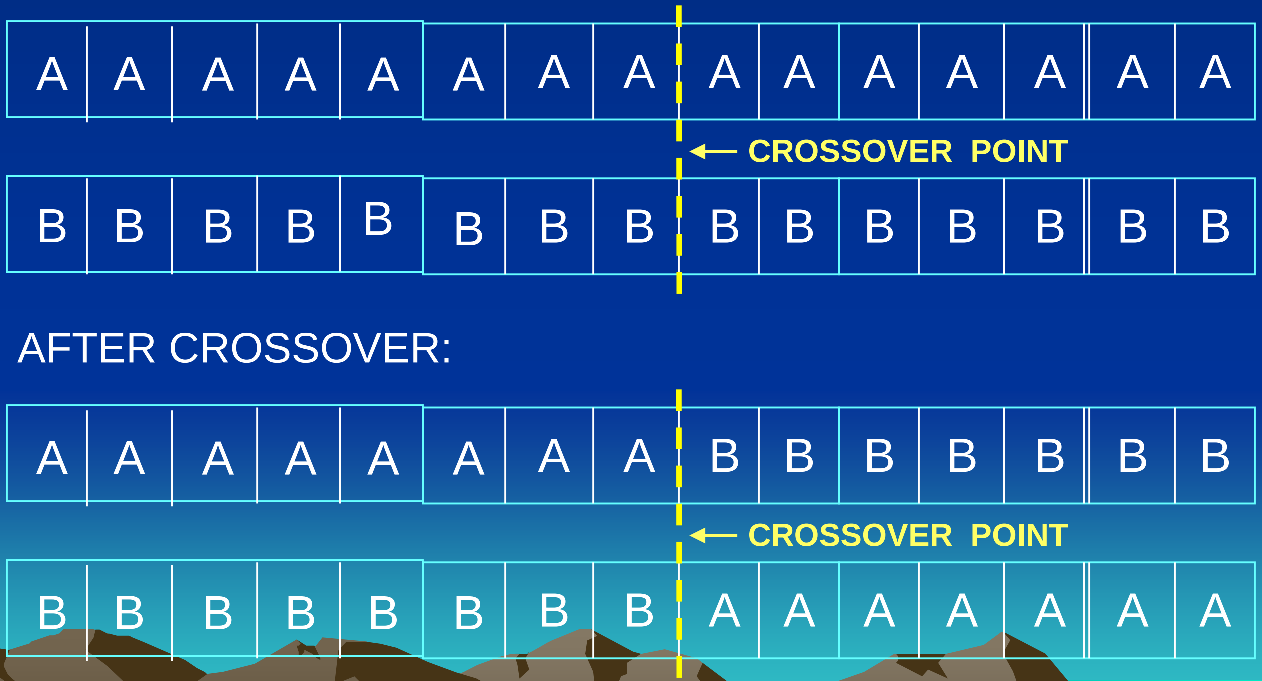
FROM GENETIC ALGORITHMS TO DIFFERENTIAL EVOLUTION

- In the N- dimensional (assuming N- parameters) solution space there exists one point that is the optimum – *the global optimum*
- At this point the objective function which we seek to optimize reaches a global maximum or minimum value
- Every single point in this space is associated with a "fitness", i.e. where it stands relative to this global optimum; there is a one-to-one mapping between the location of each point and its fitness
- A number of candidate solutions – the "population" – are created that concurrently traverse the total solution space across "generations" – striving to "evolve" to higher and higher levels of fitness
- When a candidate moves from one point to another (evolves), what essentially changes is its state, i.e. the values of its component variables – very often coded as bits called the *genes* that compose the *chromosome*
- When moving from one generation to the next, the fitter solutions are reproduced more, while the weaker ones are gradually eliminated

CLASSICAL GENETIC ALGORITHMS

SELECTION: More the fitness, more the probability of being reproduced into the next generation

CROSSOVER: A pair of candidate solutions flip their genes within a crossover zone


A A A A A A A A A A A A A A A
CROSSOVER POINT
B B B B B B B B B B B B B B B
AFTER CROSSOVER:
A A A A A A A A B B B B B B B
CROSSOVER POINT
B B B B B B B B A A A A A A A

CLASSICAL GENETIC ALGORITHMS

- Note that Stochastics (Randomness) plays a crucial role in ALL Evolutionary Algorithms.
- Thus which pairs are crossed over are selected randomly
- Importantly, there is a probability of crossover (say 0.7), and this is effected by generating a random number for each (randomly selected) pair and checking if that number is within the CP
- the point of crossover in the chromosome is also selected randomly.

CLASSICAL GENETIC ALGORITHMS

MUTATION: An “Arbitrary” bit in an “Arbitrarily” selected Chromosome is flipped – role of stochastics and probability of mutation

SELECTION: More the fitness, more the probability of being reproduced into the next generation. Roulette wheel selection and the role of stochastics.

ELITISM: In a new generation, after crossover & mutation, if the fitness of the best solution has deteriorated from the best fitness of previous generation, then the worst solution of current generation is replaced by that best solution (of previous gen.)

CROSSOVER – MATHEMATICAL EXPRESSION

Formally, if the dimensionality of the solution space is denoted as *D* and the number of candidate solutions is *N*, then the elements of the ith solution vector $X_{i,G}$ at generation *G* may be denoted as

$$X_{i,G} = (x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, \ldots\ldots\ldots, x_{D,i,G}) \qquad \forall \mathrm{i} \in \{1, 2, \ldots, \mathrm{N}\}$$

Upon crossover between two candidates *i* and *k*, an element $x_{j,i,G}^{new}$, i.e. element j of the *new candidate i* is given as

$$x_{j,i,G}^{new} = \begin{cases} x_{j,k,G} & if \ CrP \le j \le D \\ x_{j,i,G} & otherwise \end{cases}$$

where *CrP* denotes random Crossover point; $1 < CrP < D$

while an element $x_{j,k,G}^{new}$, i.e. element j of the new candidate *k* is given as

$$x_{j,k,G}^{new} = \begin{cases} x_{j,i,G} & if \ CrP \le j \le D \\ x_{j,k,G} & otherwise \end{cases}$$

FROM GENETIC ALGORITHMS TO DIFFERENTIAL EVOLUTION

- Evolutionary Algorithms have two advantages over gradient-based methods:
  - They are unlikely to get stuck in local optima
  - They can be applied to problems that are not analytic, and even those that are not mathematically tractable – as long as you can measure the fitness of a candidate

- They differ from one another essentially in the mechanism by which a candidate solution evolves from one state (i.e. one point in the solution space) to the next

CLASSICAL GENETIC ALGORITHMS

A SHORTCOMING: The GA brings the solution within a close neighborhood of the global optimum, and then, instead of moving straight towards that point , it tends to “beat around the bush”

ACCELERATION SCHEMES: Various techniques have been developed to accelerate the solution in this neighborhood zone, and one class of these techniques plays with the mutation

ALTERNATE APPROACHES: Keeping in mind the pros and cons of classical GA, alternate approaches founded on the evolutionary principle have been developed. One such approach is *Differential Evolution*

DIFFERENTIAL EVOLUTION RAINER STORN, 1995, Univ. of Calif, Berkeley

STEP 1: CREATE A NEW SOLUTION CALLED *MUTANT VECTOR* AS

$$\overline{x}_{i,g} = \overline{x}_{i,g} + K*(\overline{x}_{r1,g} - \overline{x}_{i,g}) + F*(\overline{x}_{r2,g} - \overline{x}_{r3,g})$$

where i, r1, r2, r3 are all different, randomly selected members of the chromosome pool (i.e. population) in generation “g”,
K and F are relaxation parameters

- in effect a mutant vector is being created for member “i” by superposing on the old solution suitably scaled differences between randomly selected alternate members of the population

DIFFERENTIAL EVOLUTION: *CROSSOVER*

STEP 2: Use the Mutant Vector as a *crossover* partner for the old solution to generate the new solution; i.e. some randomly selected variables composing the old solution are replaced by corresponding variables from the mutant vector to create the new solution – called the *TRIAL VECTOR*

Crossover is performed between the ‘mutant’ vector $V_{i,G}$ and the existing (parent) vector $X_{i,G}$ to generate a ‘trial’ vector $Z_{i,G}$ according to

$$z_{j,i,G} = \begin{cases} v_{j,i,G} & \text{if } \mathrm{rand}_{\mathrm{j}}(0,1) \le Cr \\ x_{j,i,G} & \text{otherwise} \end{cases}$$

where $z_{j,i,G}$ is the element *j* of the trial vector $Z_{i,G}$, $rand_j(0, 1)$ denotes a random number between *0* & *1* applied to the element *j*, *Cr* is the crossover threshold (probability).

DIFFERENTIAL EVOLUTION: *ELITISM*

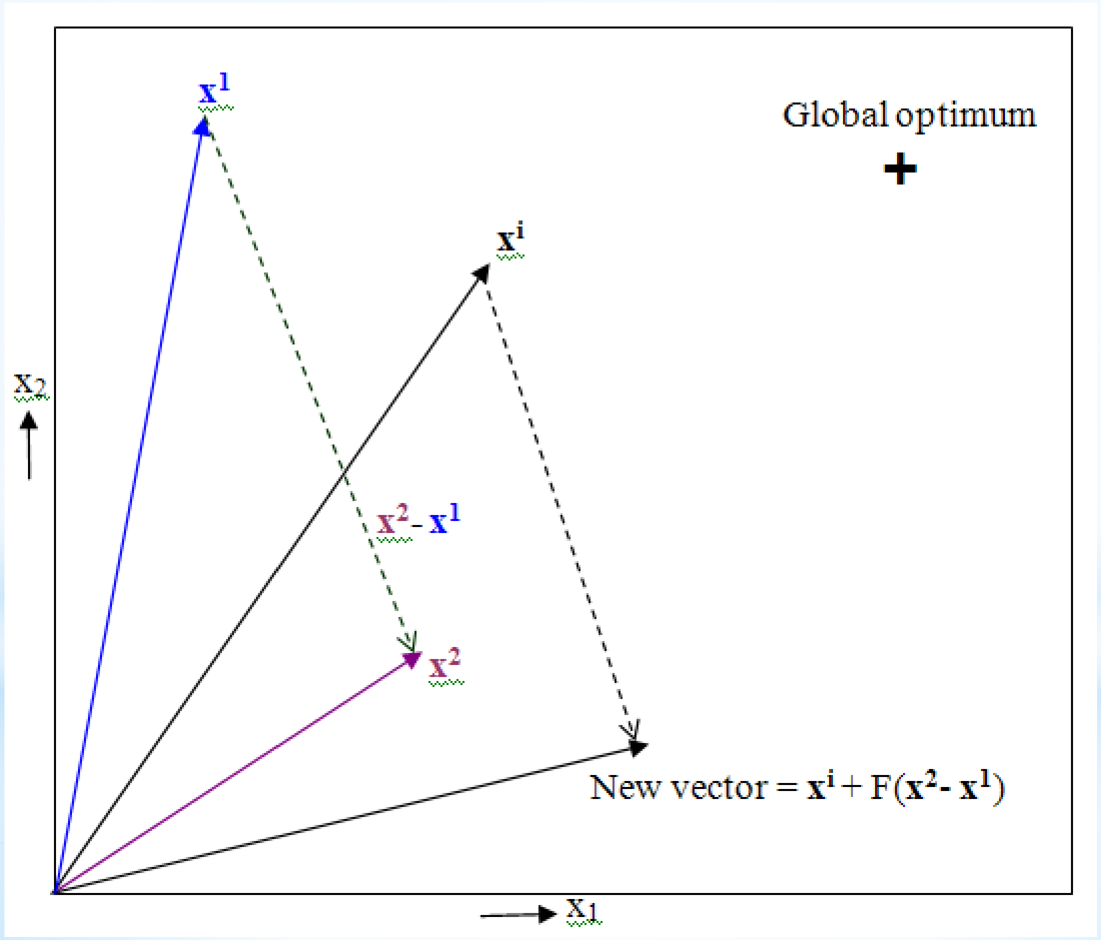
STEP 3: Check if the new solution member "i" has higher fitness than the previous one; if yes, only then replace: this is Elitism (vertical).

Question related to Step 3: Where is the difference with GA and what are the possible implications?

IT IS SEEN THAT DE's GENERALLY LEAD TO FASTER CONVERGENCE THAN CLASSICAL GA's – SEE ILLUSTRATION IN NEXT SLIDE.

DIFFERENTIAL EVOLUTION RAINER STORN, second-half-90'S

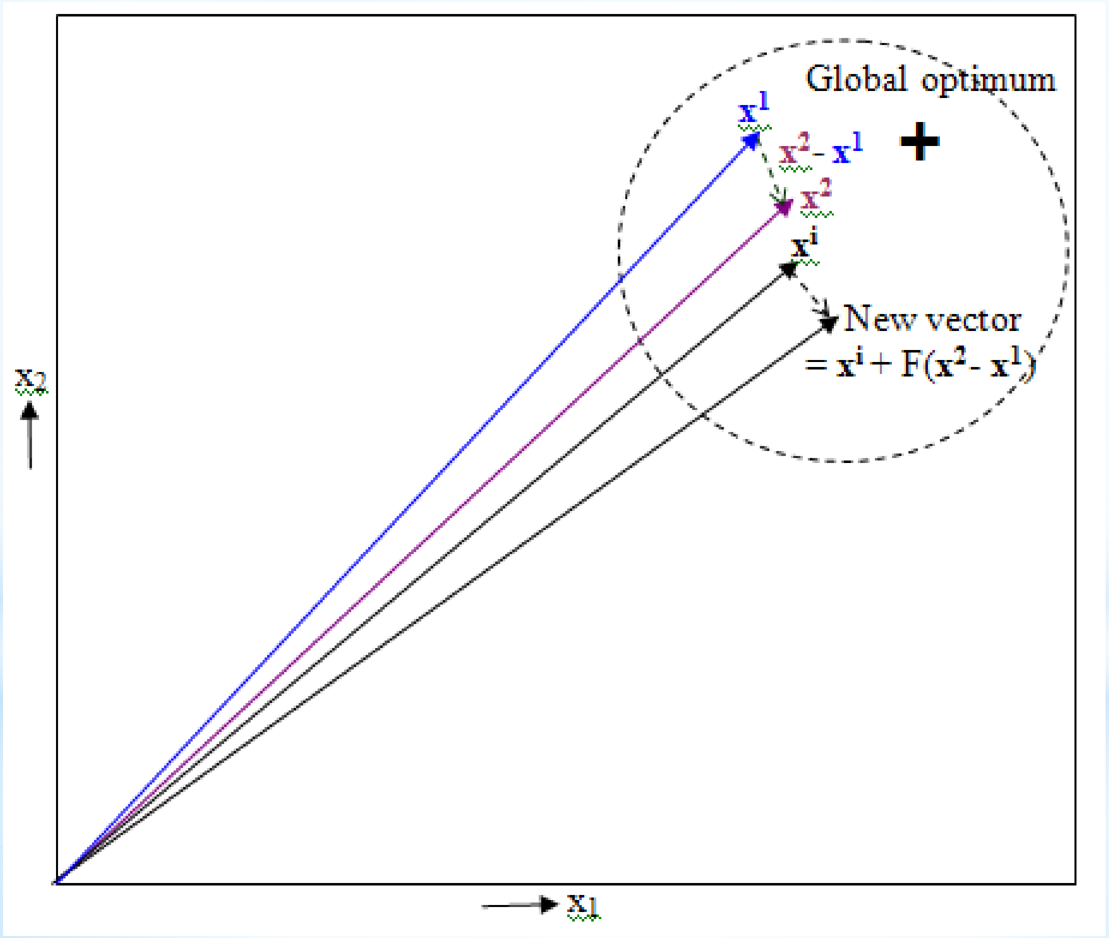
$$\overline{x}_{i,g} = \overline{x}_{i,g} + K*(\overline{x}_{r1,g} - \overline{x}_{i,g}) + F*(\overline{x}_{r2,g} - \overline{x}_{r3,g})$$



Close after Start

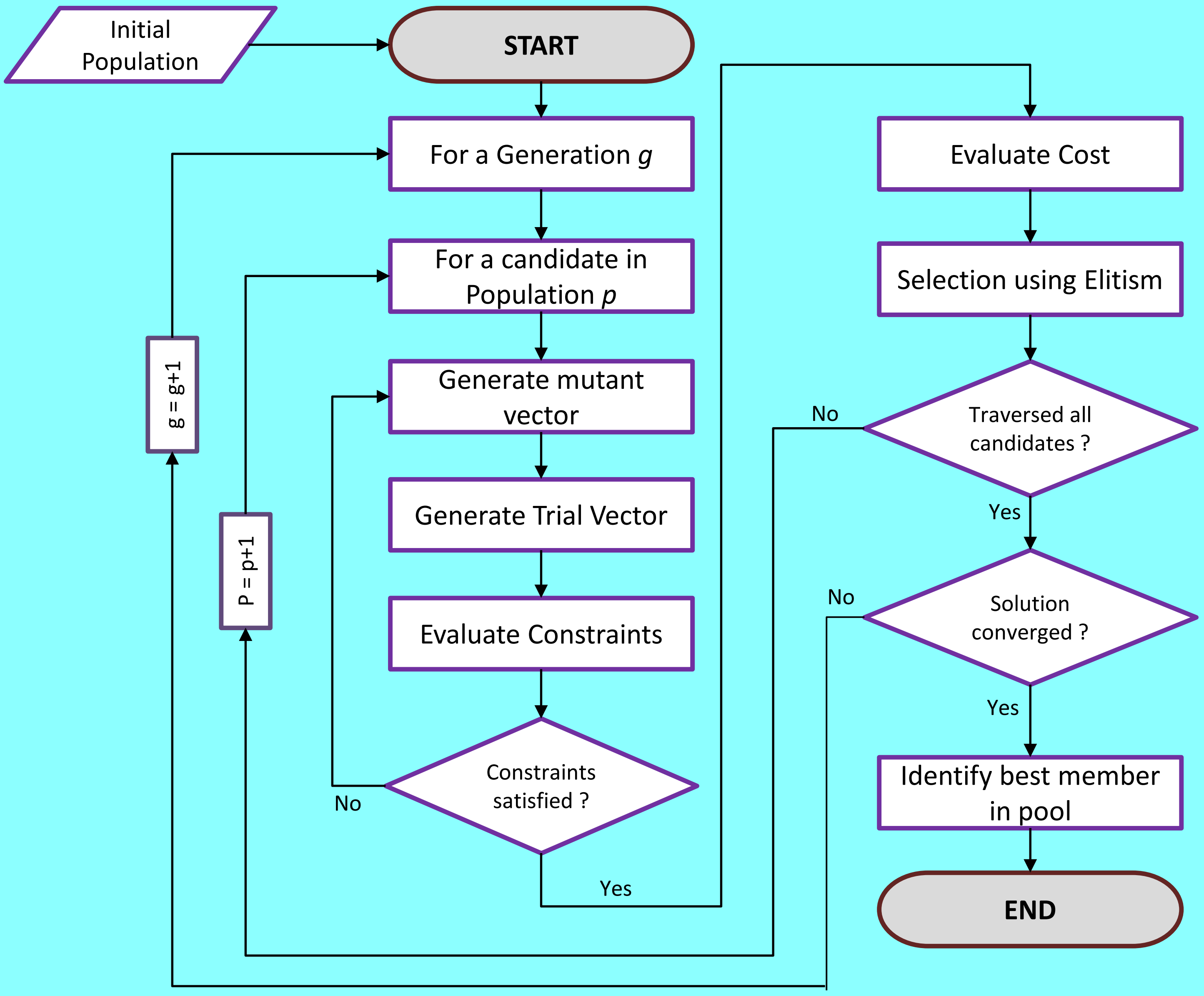
DIFFERENTIAL EVOLUTION RAINER STORN, second-half-90'S

$$\overline{x}_{i,g} = \overline{x}_{i,g} + K*(\overline{x}_{r1,g} - \overline{x}_{i,g}) + F*(\overline{x}_{r2,g} - \overline{x}_{r3,g})$$



Close to convergence

Implementation of DE Solution



Algorithm for Design and Implementation of a DE Solution

- Step 0: Get the Inputs: the number of design variables for your system, the range of each variable, the objective function, the constraints
- Step 0A: Normalize the input variables *only if* the nature of the objective function does not get altered by the normalization
- Step 1: Decide on the meta-parameters: population size, crossover probability, value of two mutation parameters K and F, max. number of generations
- Step 2: Initialize all candidates of the population. Simplest approach: take all random. Better approach: ensure that they are spread over the entire solution space
- Step 2A: Confirm that all constraints are satisfied by each initialized candidate, if not, regenerate each till constraints are satisfied

Algorithm for Design and Implementation of a DE Solution

- Step 3: Go over each generation
- Step 4: Go over each member of population (i.e. candidate)
- Step 5: Randomly select the candidates to mutate with this candidate
- Step 6: Perform mutation to generate the Mutant Vector
- Step 7: Perform crossover to generate the Trial Vector
- Step 8: Check the Trial Vector for satisfaction of all constraints, if it fails, repeat steps 5-7 till it passes constraint checks
- Step 9: Compare fitness of Trial Vector with Parent Vector and select the fitter one to get into the next generation
- Step 10: End population loop
- Step 11: Check if solution converged (your convergence criteria?) or max. no. of generations reached, in either case, End Evolution.

Particle Swarm Optimization

Particle Swarm Optimization

- Particle Swarm Optimization (PSO) is the foremost among the *Swarm Optimization Algorithms*, which can be called Cousins of *Evolutionary Optimization Algorithms*
- The similarity is that both work upon a population of candidate solutions that improve in parallel across a large sequence of iterative steps
- Dissimilarity: while Evolutionary Algorithms create new solutions from old ones by combining them through crossover- and mutation- like (Genetic Evolution) operations, in Swarm Algorithms the candidates maintain their individual identities and travel over the parameter space by exchanging information with each other (like Swarms)(no “Selection” step)
- Strictly speaking, Swarm Algorithms can be more appropriately said to be *Socially Inspired* rather than *Biologically Inspired*.

Particle Swarm Optimization

Particle Swarm Optimization

- Let us take N candidates of a population in a multi-dimensional (M) solution space, we will call these candidates as “particles”
- At any iteration G – strictly we no longer call them *generations* but instead *iterations*, however, for similarity with earlier formalisms we still call it G – we have these N particles at various positions p_i and moving with corresponding velocities v_i, for $i \in \{1, 2, \ldots, N\}$
- Importantly, each particle retains memory of its own best ever position $p_{i,best}$ – where best-ever implies that position which had the best fitness when mapped into objective space – and has information also of the g_{best}, i.e. the global best ever attained by any particle

Particle Swarm Optimization

- Then the velocity and position update rules of particle *i*, in the classical PSO Algorithm, is

$$v_i^{G+1} = v_i^G + c_1 \times rand \times (p_{i,best} - p_i^G) + c_2 \times rand \times (g_{best} - p_i^G)$$

and

$$p_i^{G+1} = p_i^G + v_i^{G+1}$$

- where, apart from the earlier defined notations,
  - c_1, c_2 are the weights of local and global information
  - rand is a random number in [0, 1], different for each use
  - it is being assumed that velocity represents the change in position from one iteration to the next
  - c_1 and c_2 are empirical parameters (meta-parameters) but these are selected by rule of thumb such that $c_1 + c_2 = 4$
  - higher velocities lead to faster movements but with higher chances of instability, hence a max. vel. v_{max} is set for each dimension
  - remember that *v, p and g* are all real-valued vectors in *M-dimensional* solution space.

Particle Swarm Optimization: Algorithm

```
for each particle
    initialize particle  # Try to ensure maximum spread across solution space
    if constraints not satisfied, re-initialize
    obtain fitness
    fbest, pbest ← fitness, position
    broadcast fbest, pbest
    gbest ← pbest of best(all fbest)
    v      ← 0
end
do       # for iteration or “generation” loop
    for each particle
        evaluate new v  # Check and implement bound
        evaluate new p
        evaluate fitness
        if fitness < fbest  # for minimization problem, else reverse
            fbest, pbest ← fitness, p
        broadcast fbest, pbest
        gbest ← pbest of best(all fbest)
    end
while preset fbest not achieved or within max. num. of gens.
```

WATCH NPTEL VIDEO

THANK YOU