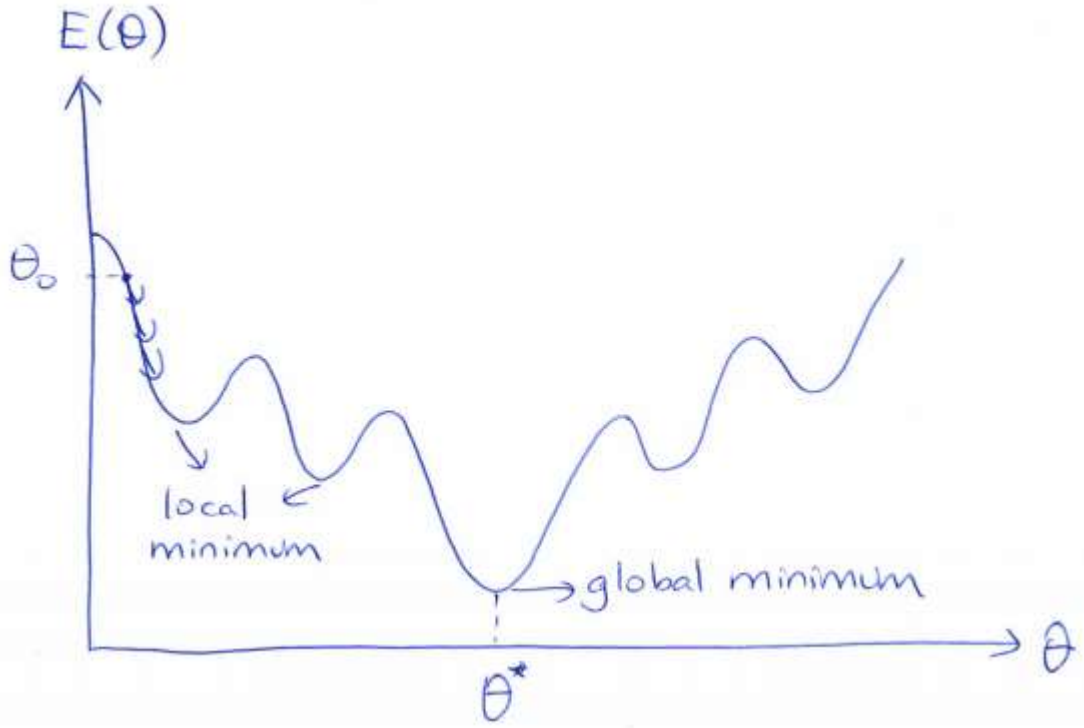
KON 426E
INTELLIGENT CONTROL SYSTEMS

LECTURE 13

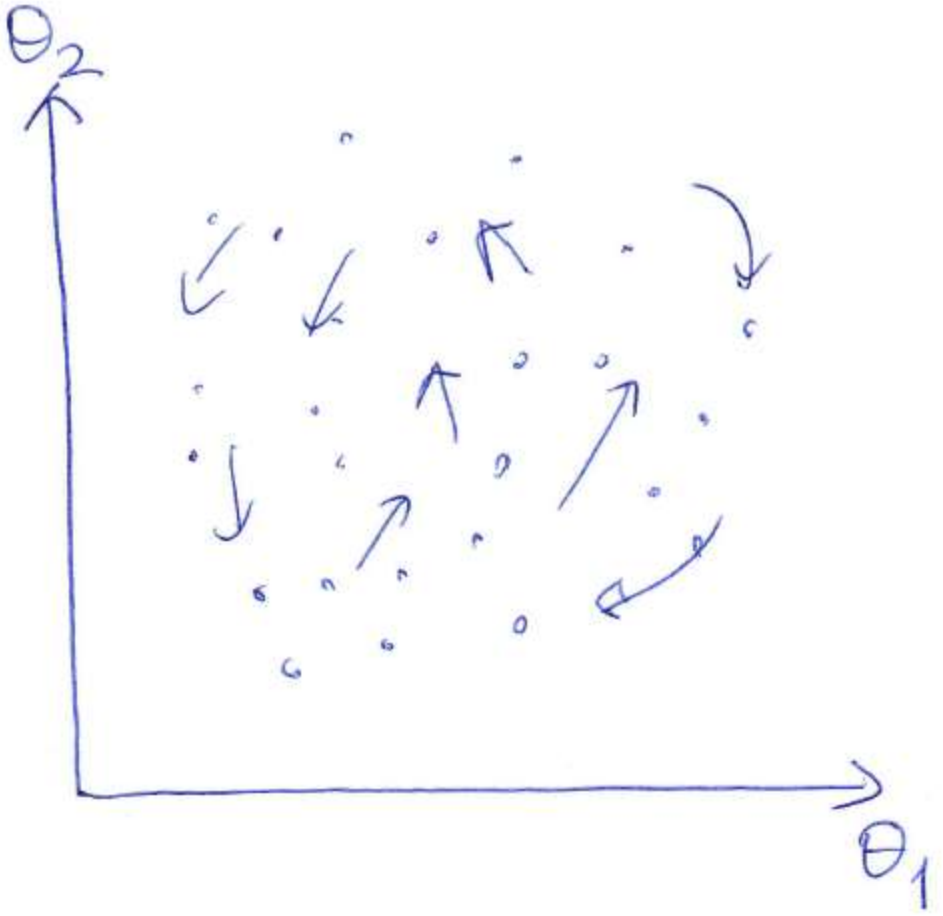
30/06/2022

An Important Property of Derivative-Based Optimization Methods



Derivative-based optimization methods assure you only the **local minimum**, they do not gurantee the **global minimum**.

- In derivative-free optimization algorithms, the aim is to search all of the parameter space.
- This is accomplished by stochastic features utilized in these types of algorithms.



Some Examples of Derivative-Free Optimization Algorithms

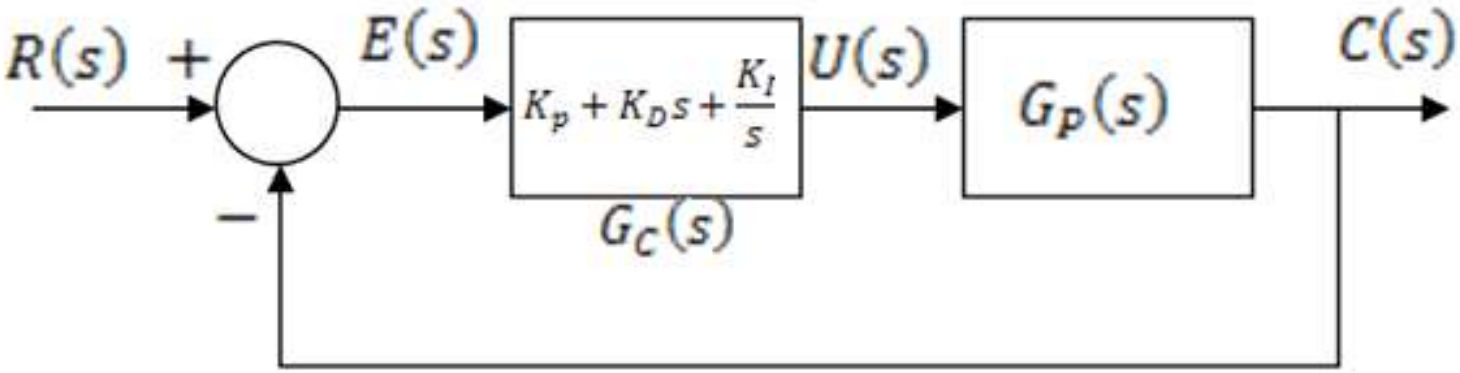
- Hooke and Jeeves Algorithm
- Simplex based Algorithms
- Nelder-Mead Method
- Trust-Region Quadratic based Models
- Genetic Algorithms
- Hit-and-Run Algorithms
- Simulated Annealing
- Particle Swarm Optimization Algorithm
- Firefly Algorithm
- …….

Some Features of Derivative-Free Optimization

- **Derivative Freeness** They don't need functional derivative information to search for a set of parameters that minimize (or maximize) a given objective function.
- **Intuitive Guidelines** Motivated by nature's wisdom
- **Slowness** Generally slower than derivative-based optimization methods
- **Flexibility** Relieves the requirement for differentiable objective functions, so we can use complex objective functions (Even NN's and fuzzy models can be included in the objective function)

- **Randomness** These methods are stochastic, they use random number generators in determining subsequent search directions. The probability of finding a global optimal solution is non-zero over a fixed amount of computation time.
- **Analytic opacity** Difficult to do analytic studies of these methods because of their randomness and problem-specific nature.
- **Iterative nature** They are iterative algorithms.

Use of derivative-free optimization algorithms in control design



Due to stability issues, they can only be used **offline**.

GENETIC ALGORITHMS

- Derivative-free stochastic optimization methods based loosely on concepts of natural selection and evolutionary processes.
- First proposed by John Holland at the University of Michigan in 1975.
- GA's are parallel search procedures that can be implemented on parallel-processing machines for speeding up their operations.
- GA's are applicable to both continuous and discrete optimization problems.
- GA's are stochastic and less likely to get trapped in local minima.

GA's encode each point in a parameter space into a binary bit string called a **chromosome** and each point is associated with a **fitness value** (for maximization fitness value is usually equal to the objective function evaluated at the point)

GA's usually keep a set of points as a **population (or gene pool)** which is **evolved** repeatedly towards a better overall fitness value.

In each **generation**, GA constructs a new population using **genetic operators** (such as **crossover** and **mutation**)

Members with higher fitness values are more likely to survive and participate in mating (crossover) operations. After a number of generations, the population contains members with better fitness values.

Components of GA's

Encoding Schemes

They transform points in parameter space into bit string representations.

Example: (11, 6, 9) in a 3-dim. parameter space can be represented as

- Each coordinate value is encoded as a **gene** composed of four binary bits using binary coding.
- Different coding schemes can be used (e.g. Gray coding)
- This is a way of translating problem-specific knowledge directly into the GA framework.

Fitness Evaluation

- After a **generation** is created, the fitness value of each member in the population is calculated.
- For a maximization problem, the fitness value f_i of the ***ith*** member is usually the objective function evaluated at this member.
- Also, it is possible to use rankings of members in a population as the fitness value (So, you don't need accurate objective function.)
- Usually fitness values have to be positive.

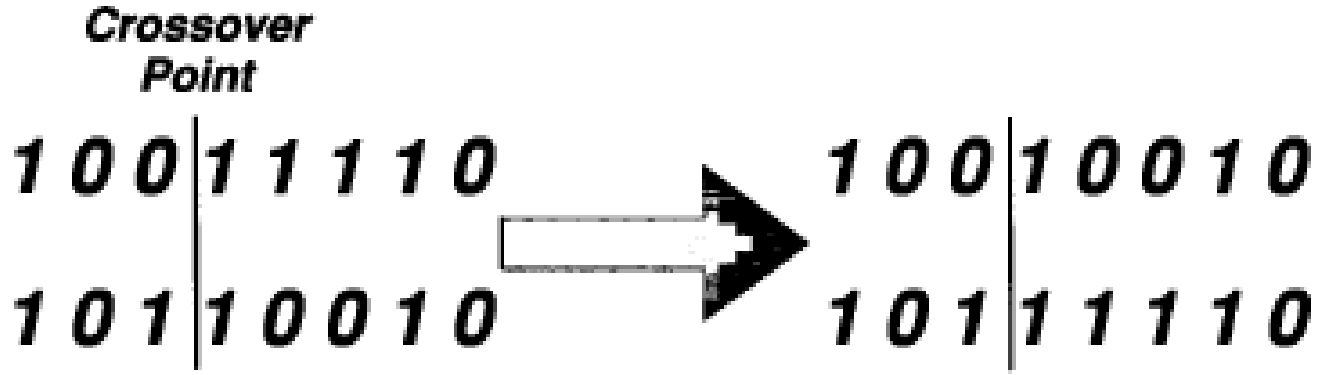
Selection

- After fitness evaluation, we have to create a new population from the current generation.
- The selection operation determines which parents participate in producing off-spring for the next generation --- which is analogous to survival of the fittest in natural selection.
- Usually members are selected for mating with a selection probability proportional to their fitness values.
- E.g. Selection probability can be equal to $\frac{f_i}{\sum_{k=1}^{k=n} f_k}$ where n is the population size.

Crossover

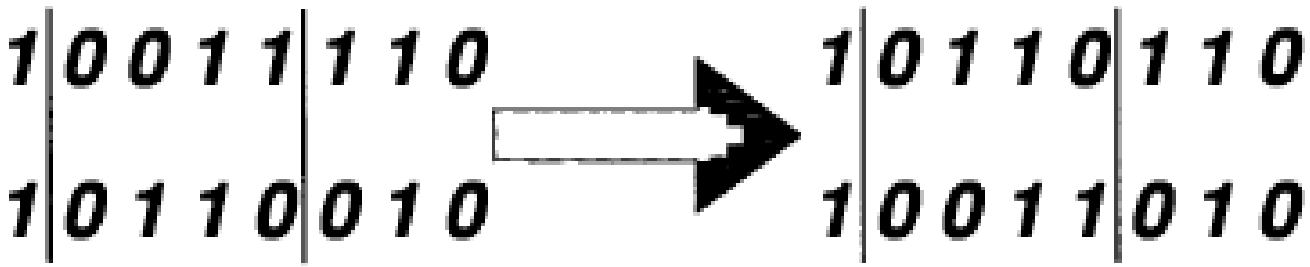
This is analogous to mating in natural evolutionary process. It is usually applied to selected pairs of parents with a probability equal to a given **cross-over rate**.

One-point crossover:



Cross-over point is selected at random.

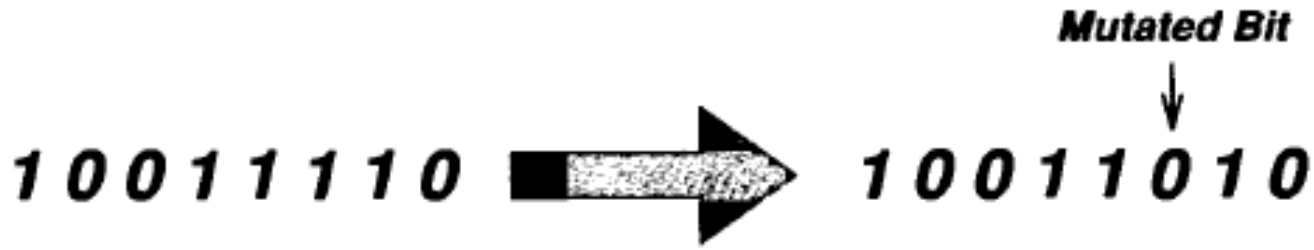
Two-point crossover:



It is possible to define *n*-point crossover.

Mutation

- If the population **does not contain all the encoded information** to solve a particular problem, a satisfactory solution cannot be reached.
- With **mutation operator** new chromosomes are generated.
- With a probability equal to a (very low) **mutation rate** a bit is flipped.
- It prevents any single bit to converge to a value.
- It also prevents the population to converge to a local optimum.
- Mutation rate is generally low so good chromosomes will not be lost (generally below 0.1).



There are two issues about mutation that are determined randomly and therefore add stochasticity.

- Whether there will be a mutation or not.
- At which bit the mutation will take place.

Sources of stochasticity in genetic algorithms:

- Selection of initial individuals
- Crossover
  - Whether there will be crossover or not.
  - Which individuals will mate.
  - At which bit(s) will the crossover take place.
- Mutation
  - Whether there will be mutation or not.
  - At which bit (s) will the mutation take place.

A SIMPLE GENETIC ALGORITHM (for maximization)

Step 1: Initialize a population with randomly generated individuals.

Step 2: Evaluate the fitness value of each individual.

Step 3:

a) Select two members from the population with probabilities proportional to their fitness values.

b) Apply crossover with a probability equal to the crossover rate.

c) Apply mutation with a probability equal to the mutation rate.

d) Repeat (a) to (d) until enough members are generated to form the next generation.

Step 4: Repeat steps 2-4 until a stopping criteria is met.

Stopping Criteria:

- **Computation time:** A designated amount of computation time, or number of iteration counts is reached.
- **Optimization goal:** f_k (best objective function obtained at count k), is less than a certain preset goal value.
- **Minimal improvement**: $f_k - f_{k-1}$ is less than a preset value.
- **Minimal relative improvement**: $\frac{f_k - f_{k-1}}{f_{k-1}}$ is less than a preset value.

Elitism

Always keeping a certain number of best members when each new population is generated.

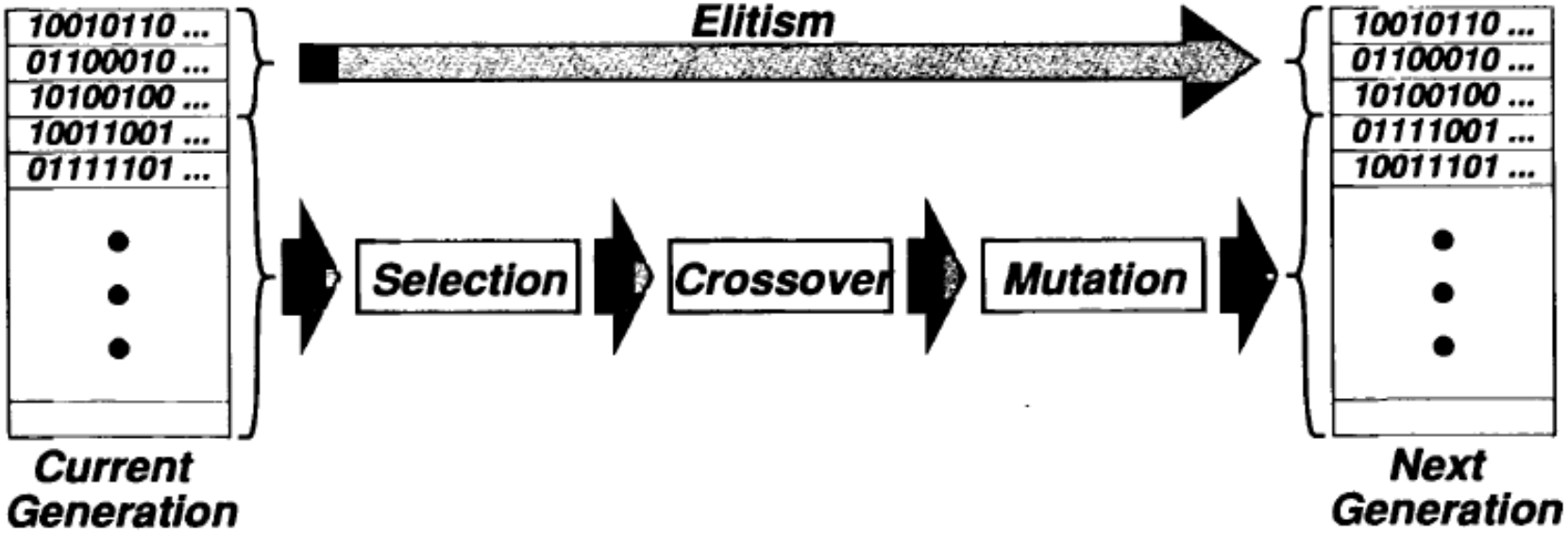


Figure 7.3. *Producing the next generation in GAs.*

PARTICLE SWARM OPTIMIZATION (PSO)

- The particle swarm optimization (PSO) algorithm is a population-based search algorithm based on the simulation of the social behavior of birds within a flock.
- The aim is to graphically simulate the graceful and unpredictable choreography of a bird flock and to discover patterns that govern the ability of birds to fly synchronously, and to suddenly change direction with a regrouping in an optimal formation.

- Individuals are referred to as **particles**, they are “flown” through the hyperdimensional search space.
- Changes to the position of particles in the search space are done based on the social-psychological tendency of individuals to emulate the success of other individuals.
- The changes to a particle within the swarm are influenced by the experience, or knowledge, of its neighbors.
- The behavior of a particle is affected by that of other particles within the swarm (PSO is therefore a kind of symbiotic cooperative algorithm).
- The consequence of modeling this social behavior is that the search process is such that particles stochastically return toward previously successful regions in the search space.

- In PSO, each particle in the swarm represents a potential solution.
- A swarm is similar to a population, while a particle is similar to an individual.
- Particles are “flown” through a multidimensional search space, where the position of each particle is adjusted according to its own experience and that of its neighbours.

$\mathbf{x}_i(t)$ position of particle *i* in the search space at time step *t*

$\mathbf{v}_i(t)$ velocity component for *i* at time step *t*

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1)$$

where $\mathbf{x}_i(0) \sim U(\mathbf{x}_{min}, \mathbf{x}_{max})$

$$v_{ij}(t+1) = v_{ij}(t) + \underbrace{c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)]}_{\text{cognitive component}} + \underbrace{c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)]}_{\text{social component}}$$

Cognitive component is proportional to the distance of the particle from its own best position (particle's personal best position)
Social component is the socially exchanged information.

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)]$$

$v_{ij}(t)$ velocity of particle *i* in dimension *j* at time step *t*

$x_{ij}(t)$ position of particle *i* in dimension *j* at time step *t*

c_1 and c_2 positive acceleration constants used to scale the contribution of the cognitive and social components

$r_{1j}(t), r_{2j}(t) \sim U(0,1)$ random values in the range [0, 1] sampled from a uniform distribution.

These random values introduce a stochastic element to the algorithm.

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)]$$

$\mathbf{y}_i$ The **personal best position** associated with particle *i* the particle has visited since the first time step.

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases}$$

$f : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ is the fitness function

$\hat{\mathbf{y}}(t)$ The **global best position** at time step *t*

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), \ldots, \mathbf{y}_{n_s}(t)\} | f(\hat{\mathbf{y}}(t)) = \min\{f(\mathbf{y}_0(t)), \ldots, f(\mathbf{y}_{n_s}(t))\}$$

$\hat{\mathbf{y}}(t)$ is the best position discovered by any of the particles so far.

Algorithm 16.1 *gbest* PSO

Create and initialize an n_x-dimensional swarm;
repeat
    for *each particle* $i = 1, \ldots, n_s$ **do**
        //set the personal best position
        if $f(\mathbf{x}_i) < f(\mathbf{y}_i)$ **then**
            $\mathbf{y}_i = \mathbf{x}_i$;
        end
        //set the global best position **if** $f(\mathbf{y}_i) < f(\hat{\mathbf{y}})$ **then**
            $\hat{\mathbf{y}} = \mathbf{y}_i$;
        end
    end
    for *each particle* $i = 1, \ldots, n_s$ **do**
        update the velocity
        update the position
    end
until *stopping condition is true*;

FIREFLY ALGORITHM

- Firefly algorithm (FA) was first developed by Yang in 2007 which was based on the flashing patterns and behavior of fireflies.
- FA uses the following three idealized rules:

1.Fireflies are unisexual so that one firefly will be attracted to other fireflies regardless of their sex.

2. The attractiveness is proportional to the brightness and they both decrease as their distance increases. Thus, for any two flashing fireflies, the less brighter one will move toward the more brighter one. If there is no brighter one than a particular firefly, it will move randomly.

3.The brightness of a firefly is determined by the landscape of the objective function.

(1)The absolute luminance I_i of the firefly *i* set at $\overrightarrow{x_i}$ = (x_{i1}, x_{i2}, ... x_{id}) is equal to the objective function value at $\overrightarrow{x_i}$, i.e., $I_i = f\left(\overrightarrow{x_i}\right)$

- Assuming there are two fireflies *i* and *j*, the positions are $\overrightarrow{x_i}$, $\overrightarrow{x_j}$, respectively. When the absolute brightness value of *i* is greater than *j*, then *j* will be attracted and move towards *i*
- The brightness of the firefly *i* is constrained by the absorption of air and the size of the distance, the relative brightness value of the firefly *i* to *j* is defined as: $I_{ij}(r_{ij}) = I_j e^{-\gamma r_{ij}^2}$

γ is the light absorption coefficient, which is generally constant; and r_{ij} is the distance between i and j.

$$r_{ij} = \overrightarrow{x_i} - \overrightarrow{x_j} = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2}$$

γ is the light absorption coefficient, and its size is very important to the convergence speed and optimization effect of the algorithm. Generally take $\gamma \in$ [0.01, 100]

The magnitude of the attraction of the firefly j is proportional to the relative brightness value of i to j, the magnitude of the attraction of i to j is defined by their relative luminance values $\beta_{ij}(r_{ij})$.

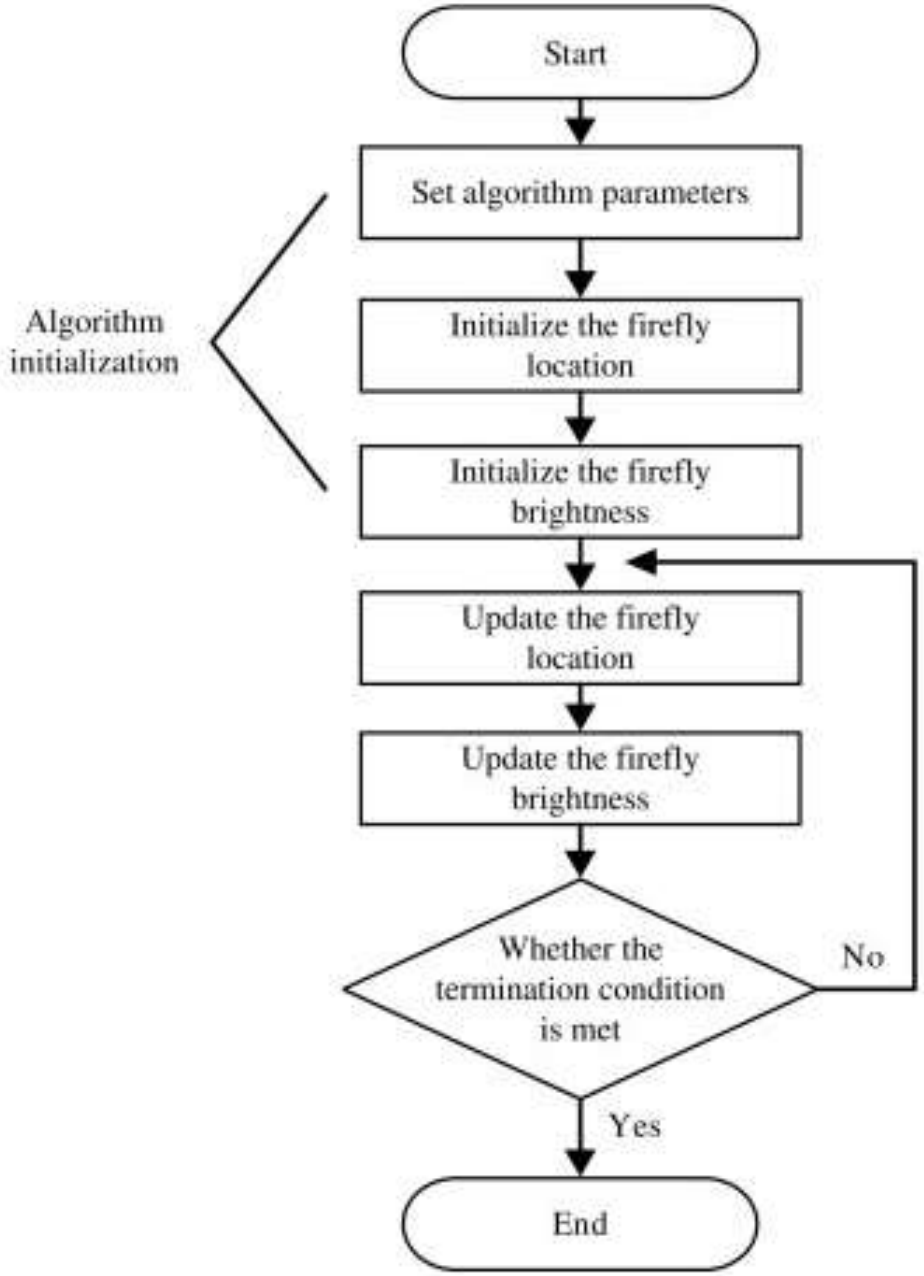
$$\beta_{ij}(r_{ij}) = \beta_0 e^{-\gamma r_{ij}^2}$$

β_0 is the maximum attraction. Generally take $\beta_0 = 1$

The firefly j is attracted by i and moves and updates its position

$$\overrightarrow{x_j}(t+1) = \overrightarrow{x_j}(t) + \beta_{ij}(r_{ij})\left(\overrightarrow{x_i}(t) - \overrightarrow{x_j}(t)\right) + \alpha\overrightarrow{\varepsilon_j}$$

α is a constant, generally $\alpha \in [0, 1]$; $\overrightarrow{\varepsilon_j}$ is a vector of random numbers drawn from a Gaussian distribution or uniform distribution at time. The second term in the position update formula depends on the size of the attraction, and the third term is the randomly generated displacement vector.


Start
Set algorithm parameters
Algorithm initialization
Initialize the firefly location
Initialize the firefly brightness
Update the firefly location
Update the firefly brightness
Whether the termination condition is met
No
Yes
End

In pseudocode the algorithm can be stated as:

```
Begin
    1) Objective function: f(x),   x = (x_1, x_2, ..., x_d);
    2) Generate an initial population of fireflies x_i   (i = 1, 2, ..., n);.
    3) Formulate light intensity I so that it is associated with f(x)
       (for example, for maximization problems, I ∝ f(x) or simply I = f(x);)
    4) Define absorption coefficient γ

    While (t < MaxGeneration)
        for i = 1 : n (all n fireflies)
            for j = 1 : i (n fireflies)
                if (I_j > I_i),
                    Vary attractiveness with distance r via exp(−γ r);
                    move firefly i towards j;
                    Evaluate new solutions and update light intensity;
                end if
            end for j
        end for i
        Rank fireflies and find the current best;
    end while

    Post-processing the results and visualization;

end
```