Artificial Intelligence

Local Search and Metaheuristics

DR. RAIHAN UL ISLAM

ASSOCIATE PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ROOM NO# AB3-1003
EMAIL: RAIHAN.ISLAM@EWUBD.EDU

MOBILE: +8801992392611

Local Search and Metaheuristics

Local search methods work by starting from some initial configuration (usually random) and making small changes to the configuration until a state is reached from which no better state can be achieved. E.g: Hill Climbing

Local search techniques, used in this way, suffer from the same problems as hill climbing and, in particular, are prone to finding local maxima that are not the best solution possible.

The methods used by local search techniques are known as metaheuristics. Examples of metaheuristics include simulated annealing, tabu search, genetic algorithms, ant colony optimization etc.

This kind of search method is also known as local optimization because it is attempting to optimize a set of values but will often find local maxima rather than a global maximum.

The Basics of Differential Evolution

- Stochastic, population-based optimisation algorithm
- Introduced by Storn and Price in 1996
- Developed to optimise real parameter, real valued functions
- General problem formulation is:

  For an objective function $f : X \subseteq \mathrm{R}^D \rightarrow \mathrm{R}$ where the feasible region $X \neq \emptyset$, the minimisation problem is to find

$$x^* \in X \text{ such that } f(x^*) \leq f(x)\ \forall x \in X$$

  where:

$$f(x^*) \neq -\infty$$

Why use Differential Evolution?

- Global optimisation is necessary in fields such as engineering, statistics and finance
- But many practical problems have objective functions that are non-differentiable, non-continuous, non-linear, noisy, flat, multi-dimensional or have many local minima, constraints or stochasticity
- Such problems are difficult if not impossible to solve analytically
- DE can be used to find approximate solutions to such problems

Evolutionary Algorithms

- DE is an Evolutionary Algorithm
- This class also includes Genetic Algorithms, Evolutionary Strategies and Evolutionary Programming

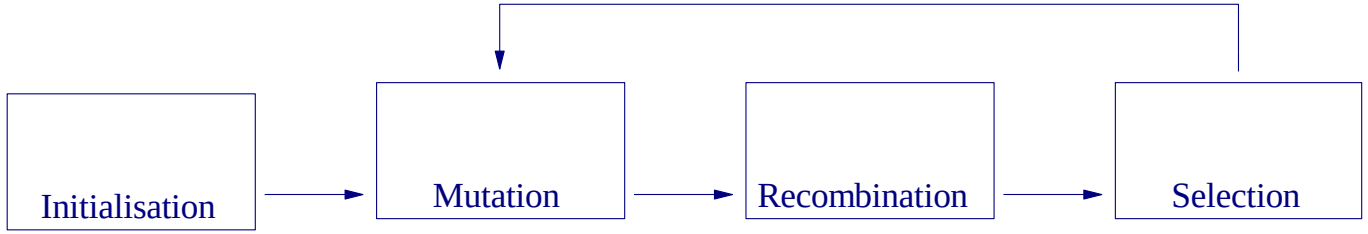


Figure 1: General Evolutionary Algorithm Procedure

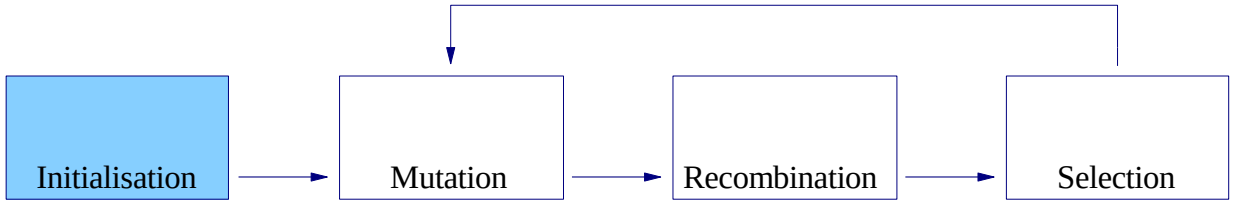
Notation

- Suppose we want to optimise a function with D real parameters
- We must select the size of the population N (it must be at least 4)
- The parameter vectors have the form:

$$x_{i,G} = [x_{1,i,G}, x_{2,i,G}, \ldots x_{D,i,G}] \quad i = 1, 2, \ldots, N.$$

where G is the generation number.

Initialisation

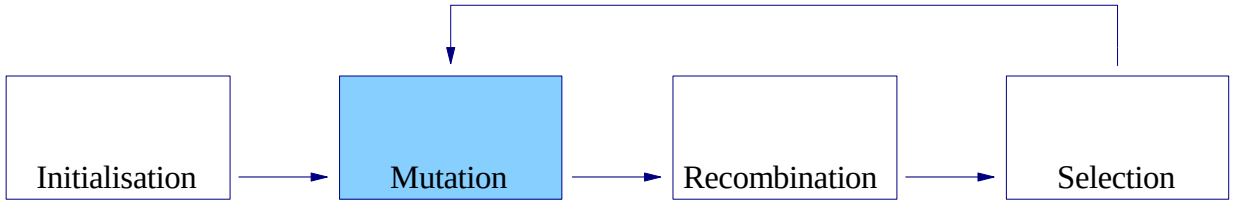


- Define upper and lower bounds for each parameter:

$$x_j^L \leq x_{j,i,1} \leq x_j^U$$

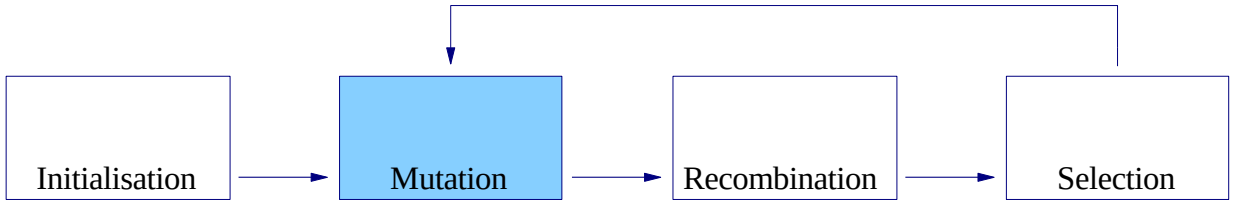
- Randomly select the initial parameter values uniformly on the intervals $[x_j^L, x_j^U]$

Mutation



- Each of the N parameter vectors undergoes mutation, recombination and selection
- Mutation expands the search space
- For a given parameter vector $x_{i,G}$ randomly select three vectors $x_{r1,G}$, $x_{r2,G}$ and $x_{r3,G}$ such that the indices i, $r1$, $r2$ and $r3$ are distinct

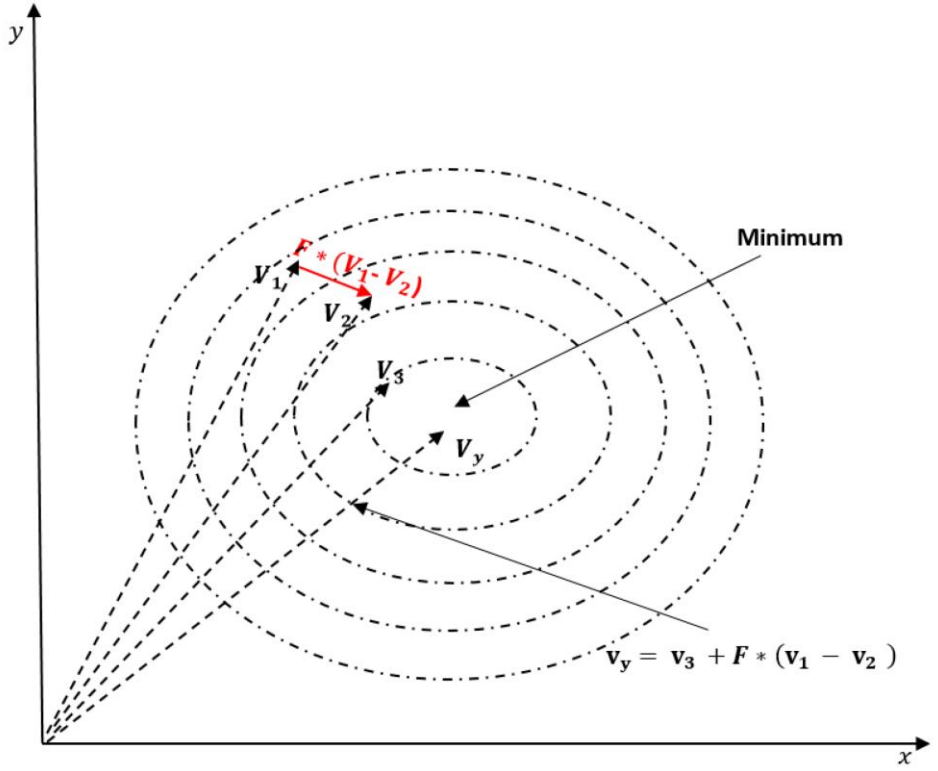
Mutation



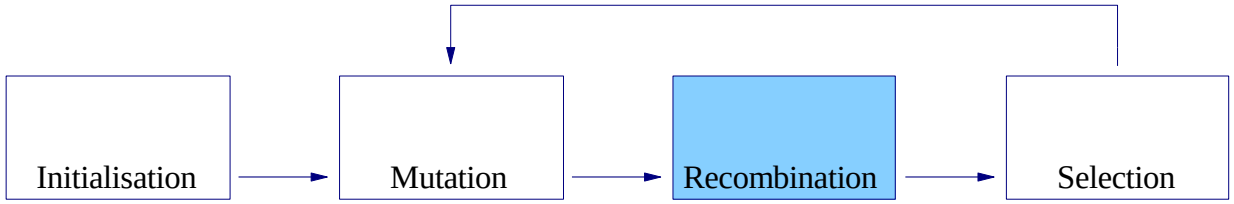
- Add the weighted difference of two of the vectors to the third

$$v_{i,G+1} = x_{r1,G} + F(x_{r2,G} - x_{r3,G})$$

- The mutation factor F is a constant from $[0, 2]$
- $v_{i,G+1}$ is called the donor vector

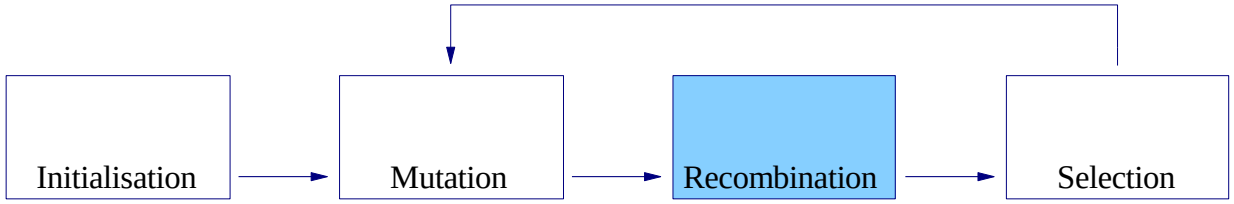


Recombination



- Recombination incorporates successful solutions from the previous generation
- The trial vector $u_{i,G+1}$ is developed from the elements of the target vector, $x_{i,G}$, and the elements of the donor vector, $v_{i,G+1}$
- Elements of the donor vector enter the trial vector with probability CR

Recombination

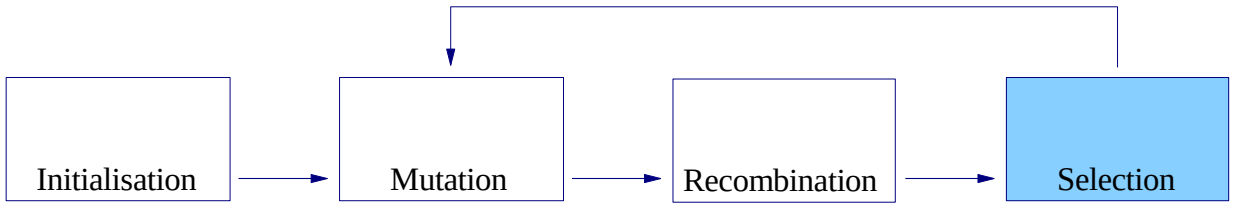


$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1} & \text{if } \mathrm{rand}_{j,i} \leq CR \text{ or } j = \mathrm{I}_{\mathrm{rand}} \\ x_{j,i,G} & \text{if } \mathrm{rand}_{j,i} > CR \text{ and } j \neq \mathrm{I}_{\mathrm{rand}} \end{cases}$$

$$i = 1, 2, \ldots, N; \; j = 1, 2, \ldots, D$$

- $\mathrm{rand}_{j,i} \sim U[0, 1]$, $\mathrm{I}_{\mathrm{rand}}$ is a random integer from $[1, 2, ..., D]$
- $\mathrm{I}_{\mathrm{rand}}$ ensures that $v_{i,G+1} = \mathrm{x}_{\mathrm{i,G}}$

Selection



- The target vector $x_{i,G}$ is compared with the trial vector $v_{i,G+1}$ and the one with the lowest function value is admitted to the next generation

$$x_{i,G+1} = \begin{cases} u_{i,G+1} & \text{if } f(u_{i,G+1}) \leq f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases} \qquad i = 1, 2, \ldots, N$$

- Mutation, recombination and selection continue until some stopping criterion is reached

Genetic Algorithms (GA)

Many practical optimum design problems are characterized by mixed continuous– discrete variables, and discontinuous and nonconvex design spaces.

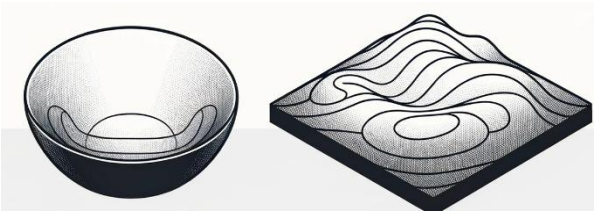
If standard nonlinear programming techniques are used for this type of problem, they will be inefficient, computationally expensive, and, in most cases, find a relative optimum that is closest to the starting point.

Genetic algorithms (GAs) are well suited for solving such problems, and in most cases, they can find the global optimum solution with a high probability.

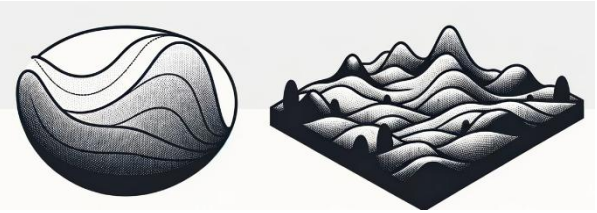
Although GAs were first presented systematically by J . H. Holland, the basic ideas of analysis and design based on the concepts of biological evolution can be found in the work of I. Rechenberg. Philosophically, GAs are based on Darwin's theory of survival of the fittest.

Genetic algorithms are based on the principles of natural genetics and natural selection. The basic elements of natural genetics— reproduction, crossover, and mutation—are used in the genetic search procedure

Convex

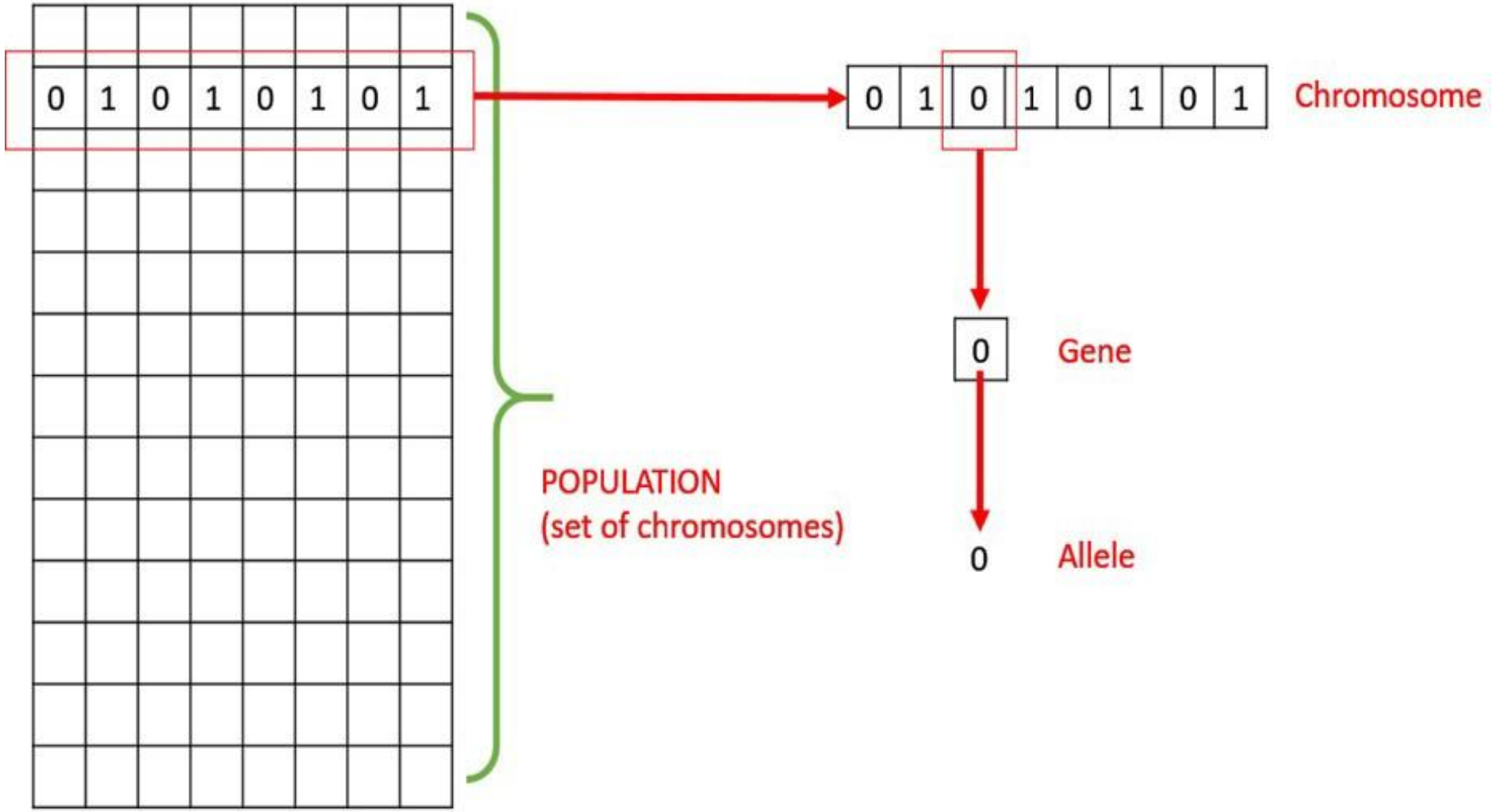


Non-convex



BASIC TERMINOLOGY

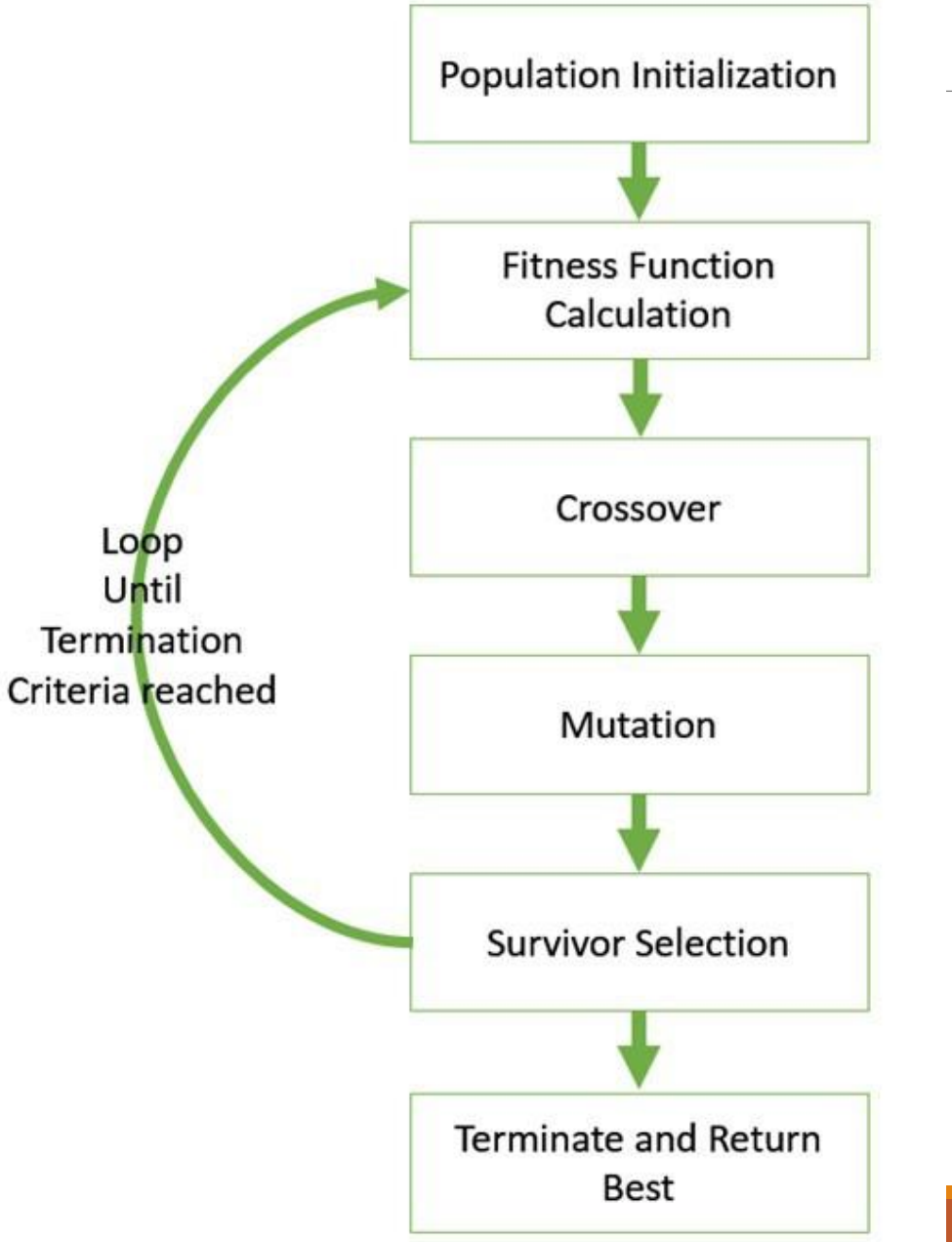
- **Population** – It is a subset of all the possible (encoded) solutions to the given problem.
- **Chromosomes** – A chromosome is one such solution to the given problem.
- **Gene** – A gene is one element position of a chromosome.
- **Allele** – It is the value a gene takes for a particular chromosome.


0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1
Chromosome
0
Gene
POPULATION
(set of chromosomes)
0
Allele

- **Genotype** – Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.
- **Phenotype** – Phenotype is the population in the actual real world solution space in which solutions are represented in a way they are represented in real world situations.
- **Decoding and Encoding** – For simple problems, the phenotype and genotype spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. Decoding is a process of transforming a solution from the genotype to the phenotype space, while encoding is a process of transforming from the phenotype to genotype space.Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.

- **Fitness Function** – A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output. In some cases, the fitness function and the objective function may be the same, while in others it might be different based on the problem.
- **Genetic Operators** – These alter the genetic composition of the offspring. These include crossover, mutation, selection, etc.

BASIC STRUCTURE OF GENETIC ALGORITHM

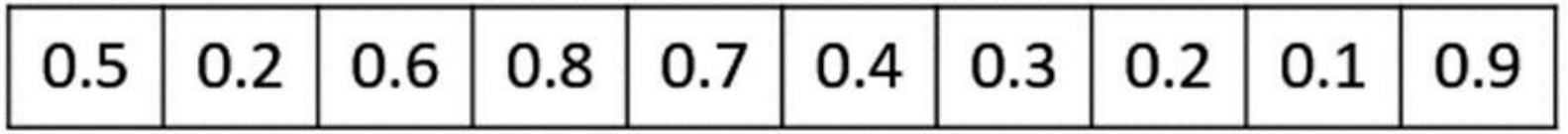


- POPULATION INITIALIZATION
    1. RANDOM INITIALIZATION
    2. HEURISTIC INITIALIZATION

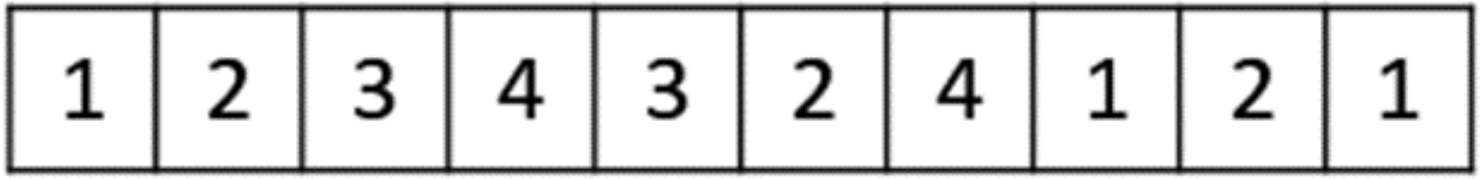
GENOTYPE REPRESENTATION

- BINARY REPRESENTATION

- REAL VALUED REPRESENTATION



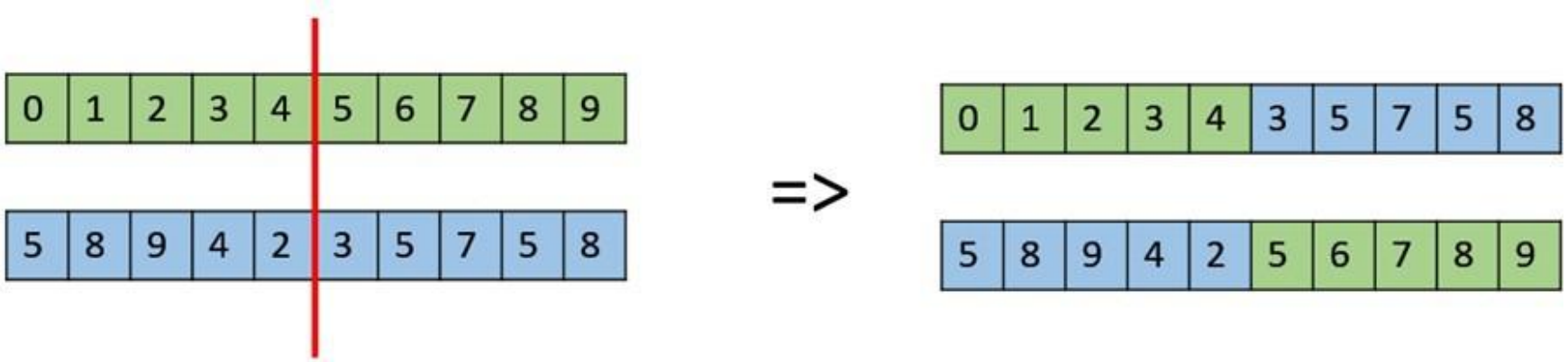
- INTEGER REPRESENTATION



- PERMUTATION REPRESENTATION

GA- CROSSOVER

- one parent is selected and one or more off-springs are produced using the genetic material of the parents
- **One Point Crossover**



- Multi Point Crossover

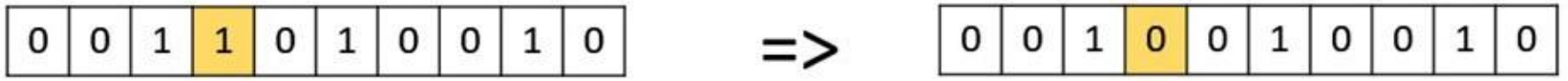
- Uniform Crossover

GA- MUTATION

- used to maintain and introduce diversity in the genetic population

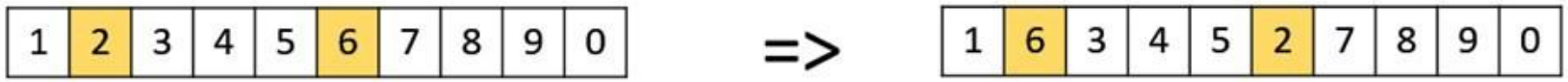
  Mutation Operators:

- **-Bit Flip Mutation**

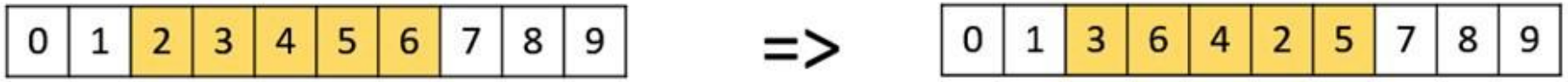


-Random Resetting

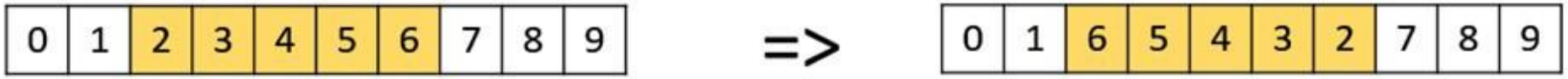
-Swap Mutation



- Scramble Mutation

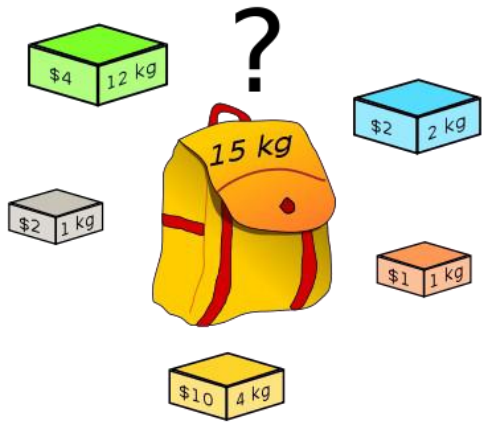


- Inversion Mutation



Knapsack Problem

- The knapsack problem or rucksack problem is a problem in combinatorial optimization:
- Given a set of items, each with a weight and a value
- Determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
- It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.



Knapsack Problem

kg	7	2	1	9
$	5	4	7	2
	A	B	C	D

Max Weight: 15kg

Knapsack Problem

Individual Representation

kg	7	2	1	9
$	5	4	7	2
	A	B	C	D
	1	0	0	1
	A picked	B and C not picked		C picked

Knapsack Problem

Initial Population

0	1	0	0
0	1	0	1
1	1	1	1
0	0	1	0
1	0	0	0
1	0	1	0

Randomly selected initial population.

Knapsack Problem

Fitness Function Calculation

0	1	0	0	4
0	1	0	1	6
1	1	1	1	0
0	0	1	0	7
1	0	0	0	5
1	0	1	0	12

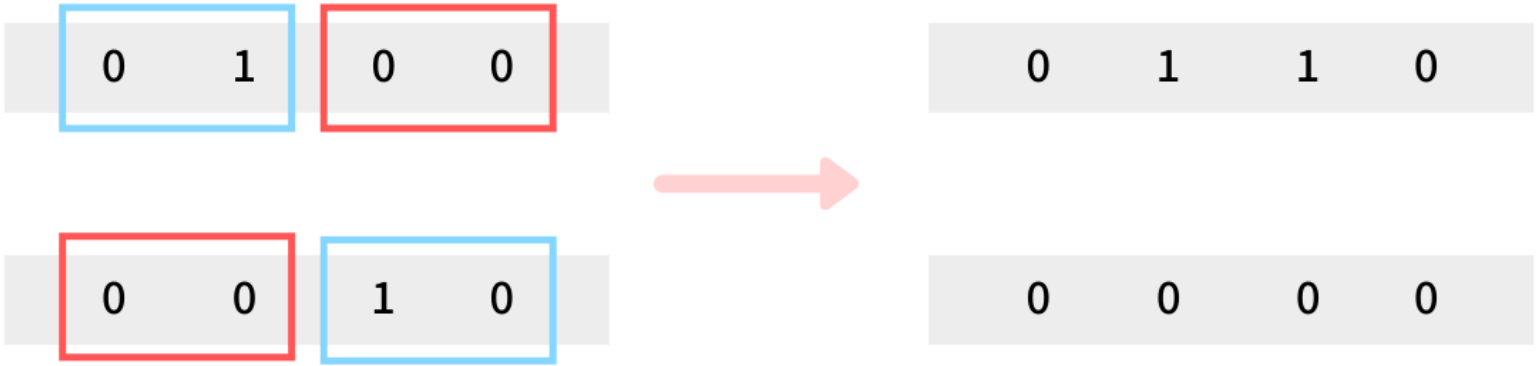
Fitness Coefficient for all individuals

Knapsack Problem

Selection

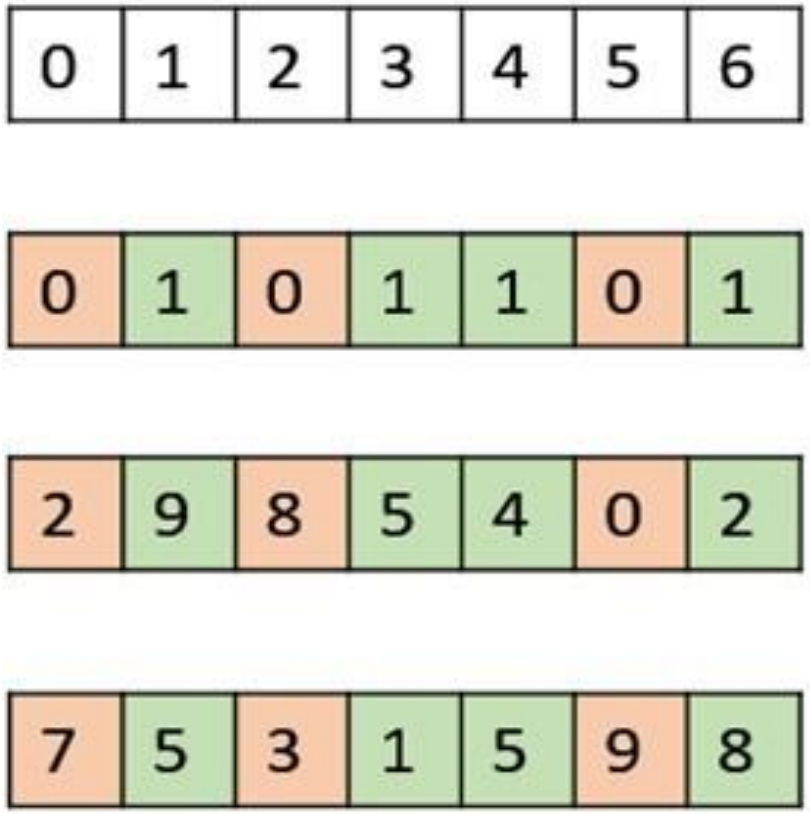
Knapsack Problem

Crossover



Knapsack Problem

Knapsack Problem



Knapsack capacity = 15
Total associated profit = 18
Last item not picked as it exceeds knapsack capacity

Simulated Annealing (SA)

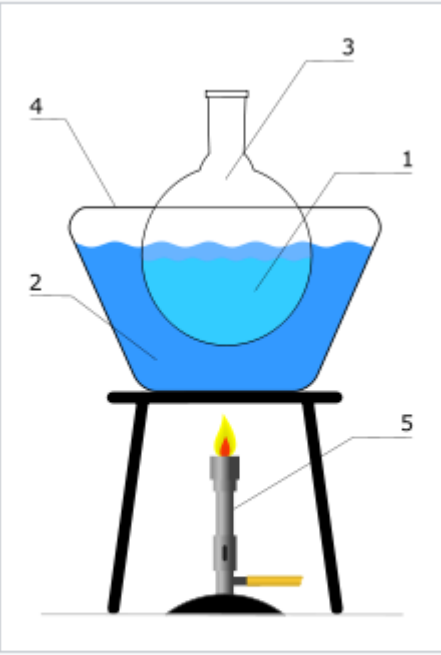
Annealing is a thermal (stochastic) process for obtaining low energy states of a solid in a heat bath. The process contains two steps:

- Increase the temperature of the heat bath to a maximum value at which the solid melts.
- Decrease carefully the temperature of the heat bath until the particles arrange themselves in the ground state of the solid. Ground state is a minimum energy state of the solid.

The ground state of the solid is obtained only if the maximum temperature is high enough and the cooling is done slowly.

Simulated annealing is a local search metaheuristic based on this method and is an extension of a process called *metropolisMonteCarlo* simulation.

Simulated annealing is applied to a multi-value combinatorial problem where values need to be chosen for many variables to produce a particular value for some global function, dependent on all the variables in the system



Heated bath schema. 1: Heated substance. 2: Heating medium. 3: Laboratory flask. 4: Bowl. 5: Gas burner

https://en.wikipedia.org/wiki/Heated_bath#/media/File:Laznia_laboratoryjna.svg

Annealing Process

- Annealing Process
  - **Raising the temperature up** to a very high level (melting temperature, for example), the atoms have a higher energy state and ***a high possibility to re-arrange the crystalline*** structure.
  - **Cooling down slowly**, the atoms have a *lower and lower energy state* and a smaller and smaller possibility to re-arrange the crystalline structure.

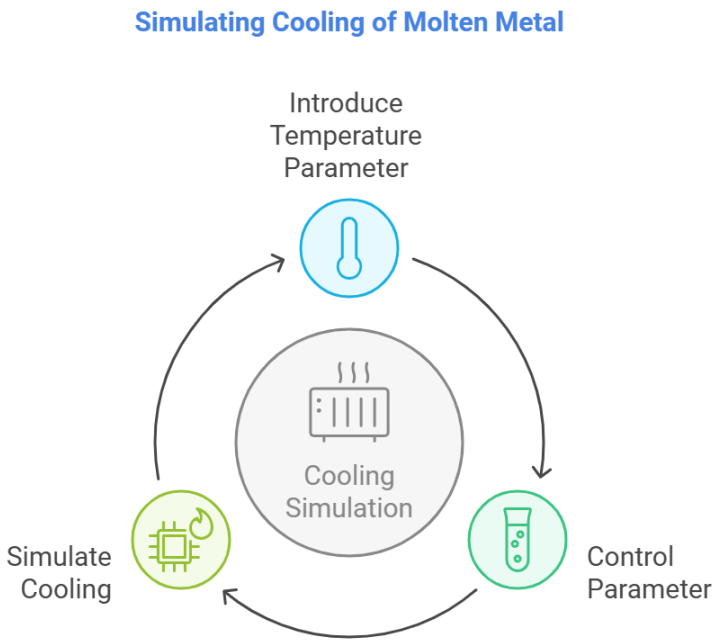
SA Procedure

The cooling phenomenon of the molten metal is simulated by introducing a *temperature-like parameter* and controlling it using the concept of Boltzmann's probability distribution.

The Boltzmann's probability distribution implies that the energy (E) of a system in thermal equilibrium at temperature T is distributed probabilistically according to the relation

$$P(E) = e^{(-E/kT)} \qquad (1)$$

where P(E) denotes the probability of achieving the energy level E, and k is called the Boltzmann's constant.



SA Procedure

$$P(E) = e^{(-E/kT)}$$

where P(E) denotes the probability of achieving the energy level E, and k is called the Boltzmann's constant.

Equation (1) shows that at high temperatures the system has nearly a uniform probability of being at any energy state;

however, at low temperatures, the system has a small probability of being at a high-energy state.

This indicates that when the search process is assumed to follow Boltzmann's probability distribution, the convergence of the simulated annealing algorithm can be controlled by controlling the temperature T.

The method of implementing the Boltzmann's probability distribution in simulated thermodynamic systems can also be used in the context of minimization of functions.

Analogy

Physical System		Optimization Problem
Metal	→	Problem
State (configuration)	→	Solution
Energy State	→	Cost function
Ground State	→	Optimal solution
Temperature	→	Control Parameter
Rapid Quenching	→	Iteration improvement
Careful Annealing	→	Simulated annealing

SA Procedure

In the case of function minimization, let the current design point (state) be X_i, with the corresponding value of the objective function given by $f_i = f(X_i)$.

Similar to the energy state of a thermodynamic system, the energy E_i at state X_i is given by

$$E_i = f_i = f(X_i)$$

Then, according to the Metropolis criterion, the probability of the next design point (state) X_{i+1} depends on the difference in the energy state or function values at the two design points (states) given by

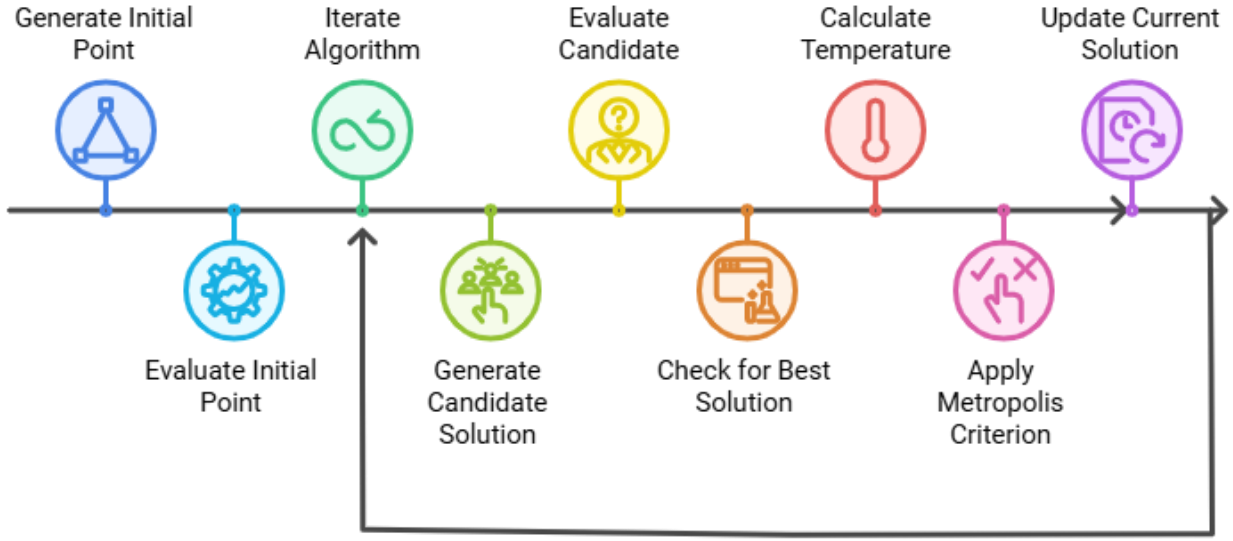
$$\Delta E = E_{i+1} - E_i = \Delta f = f_{i+1} - f_i \equiv f(X_{i+1}) - f(X_i)$$

How to find X_{i+1}?

$$P[E_{i+1}] = \min \begin{Bmatrix} 1 & if\ \Delta E \leq 0 \\ e^{-\frac{\Delta E}{kT}} & if\ \Delta E > 0 \end{Bmatrix}$$

Here Boltzmann's constant k is a scaling factor.

SA Flow Chart

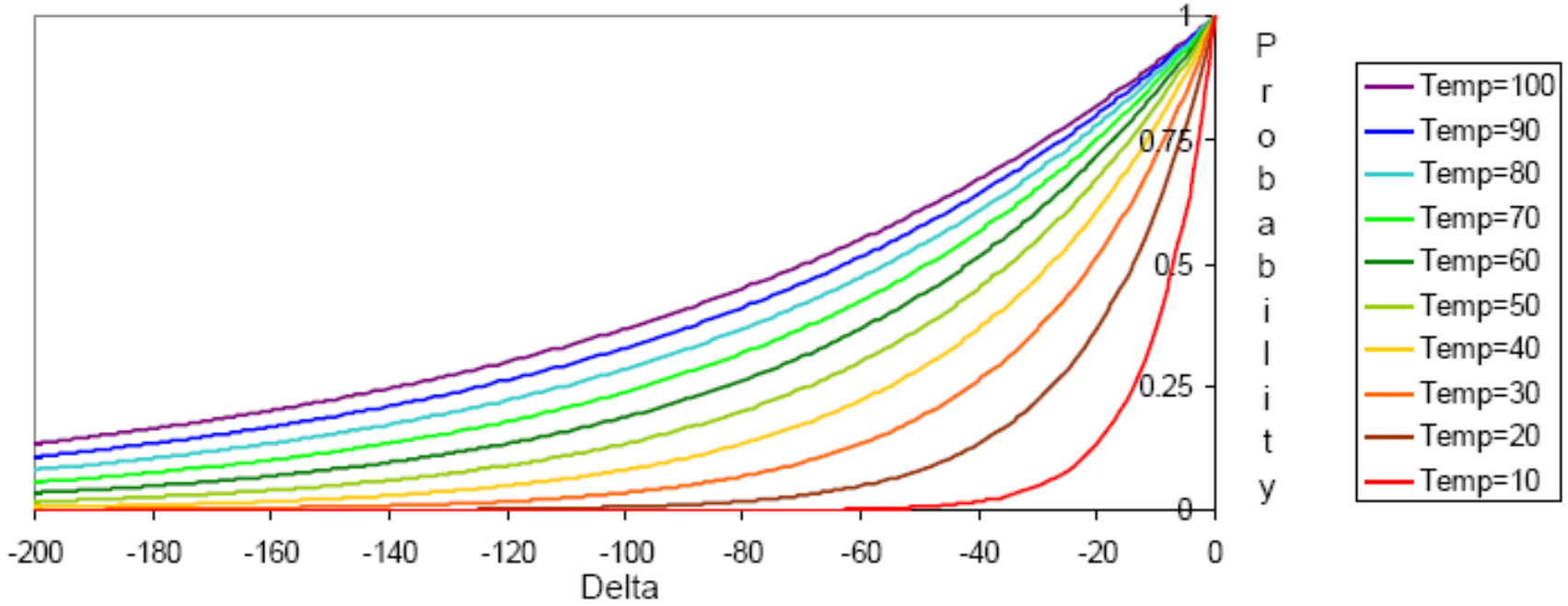



```
T = 100 n = 100
best = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
best_eval = objective(best)
curr, curr_eval = best, best_eval
for i in range(n):
    candidate = curr + randn(len(bounds)) * step_size
    candidate_eval = objective(candidate)
    if candidate_eval < best_eval:
        best, best_eval = candidate, candidate_eval
    diff = candidate_eval - curr_eval
    temp = T / float(i + 1)
    metropolis = exp(-diff / temp)
    t_rand = rand()
    if diff < 0 or t_rand < metropolis:
        curr, curr_eval = candidate, candidate_eval
```

Acceptance criterion and cooling schedule

if (delta>=0) accept

else if ($random < e^{delta / Temp}$) accept, else reject /* 0<=random<=1 */

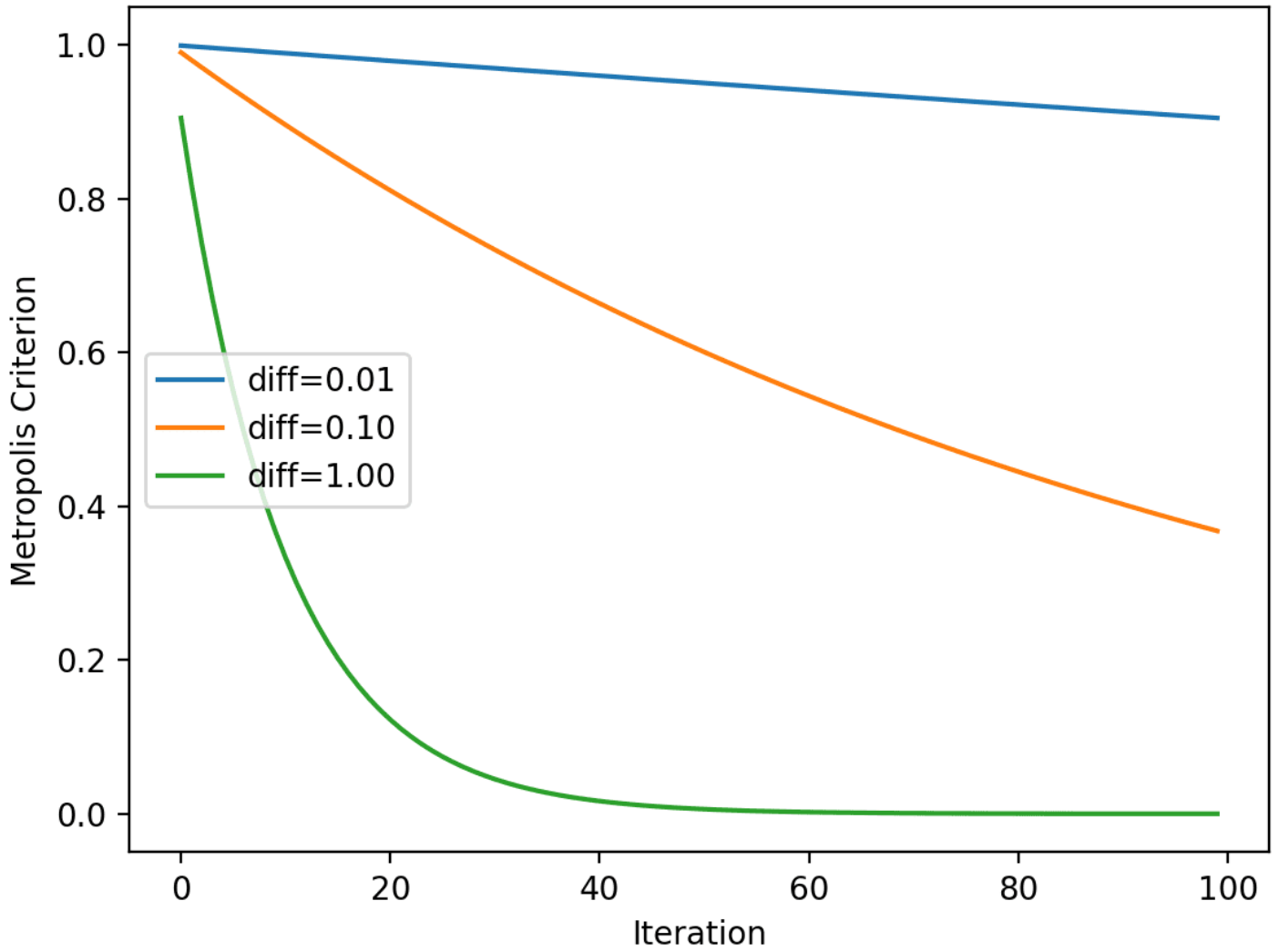


Initially temperature is very high (most bad moves accepted)

Temp slowly goes to 0, with multiple moves attempted at each temperature

Final runs with temp=0 (always reject bad moves) greedily “quench” the system

Acceptance criterion and cooling schedule

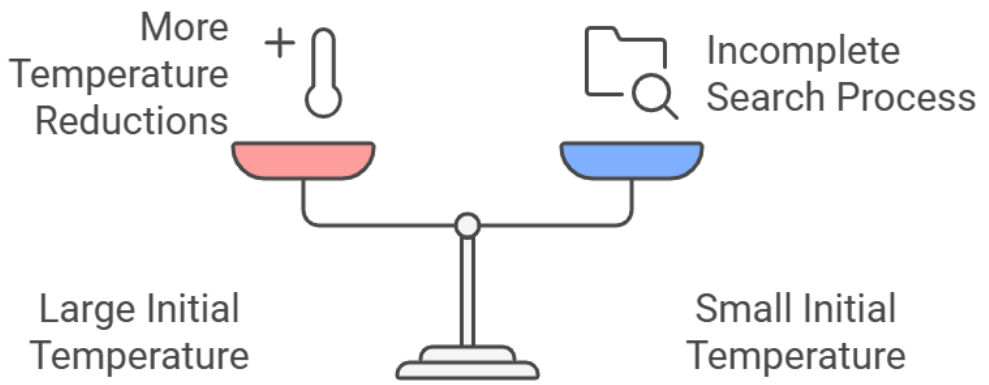


We can also see that in all cases, the likelihood of accepting worse solutions decreases with algorithm iteration.

Choice of T, n, and c

- If the initial temperature T is too large, it requires a larger number of temperature reductions for convergence.

- On the other hand, if the initial temperature is chosen to be too small, the search process may be incomplete in the sense that it might fail to thoroughly investigate the design space in locating the global minimum before convergence.

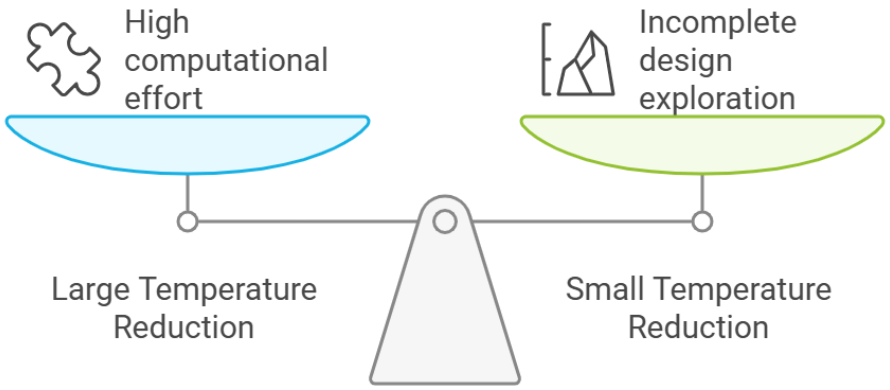


Balance initial temperature for optimal search.

Choice of T, n, and c

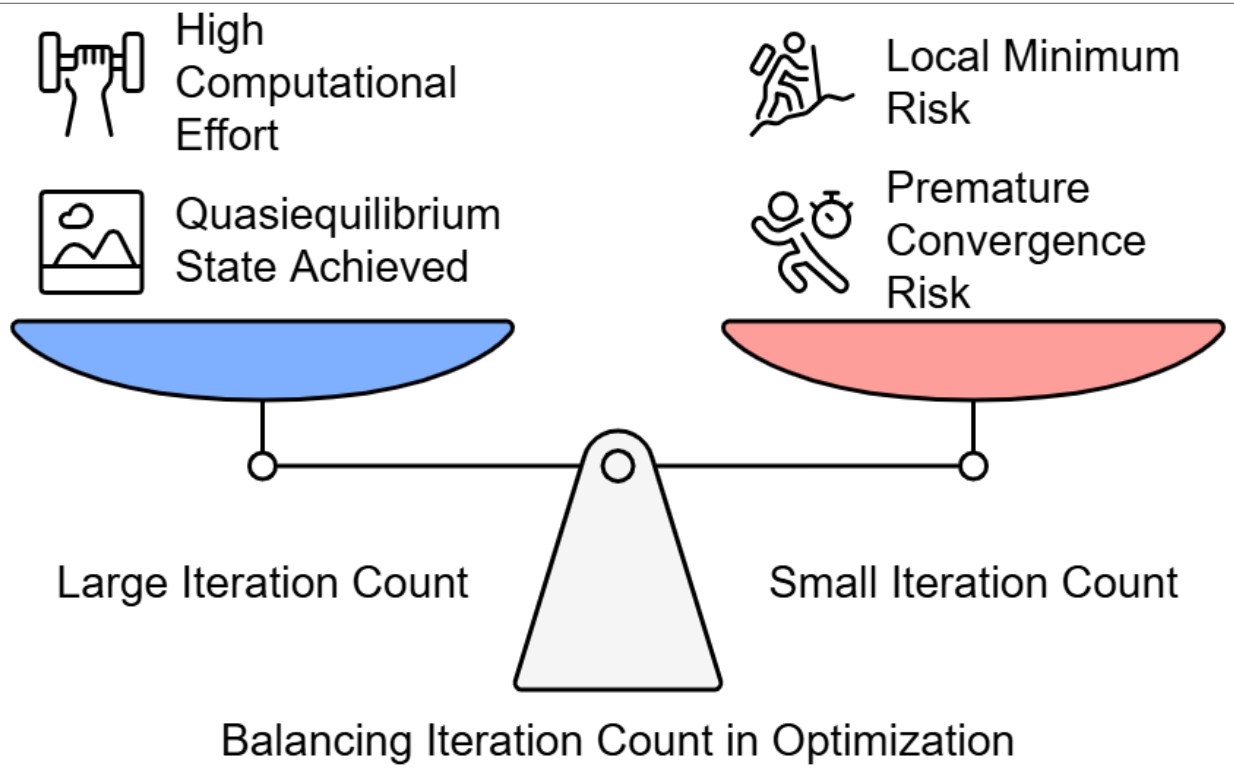
The temperature reduction factor c has a similar effect.

- Too large a value of c (such as 0.8 or 0.9) requires too much computational effort for convergence.
- On the other hand, too small a value of c (such as 0.1 or 0.2) may result in a faster reduction in temperature that might not permit a thorough exploration of the design space for locating the global minimum solution.



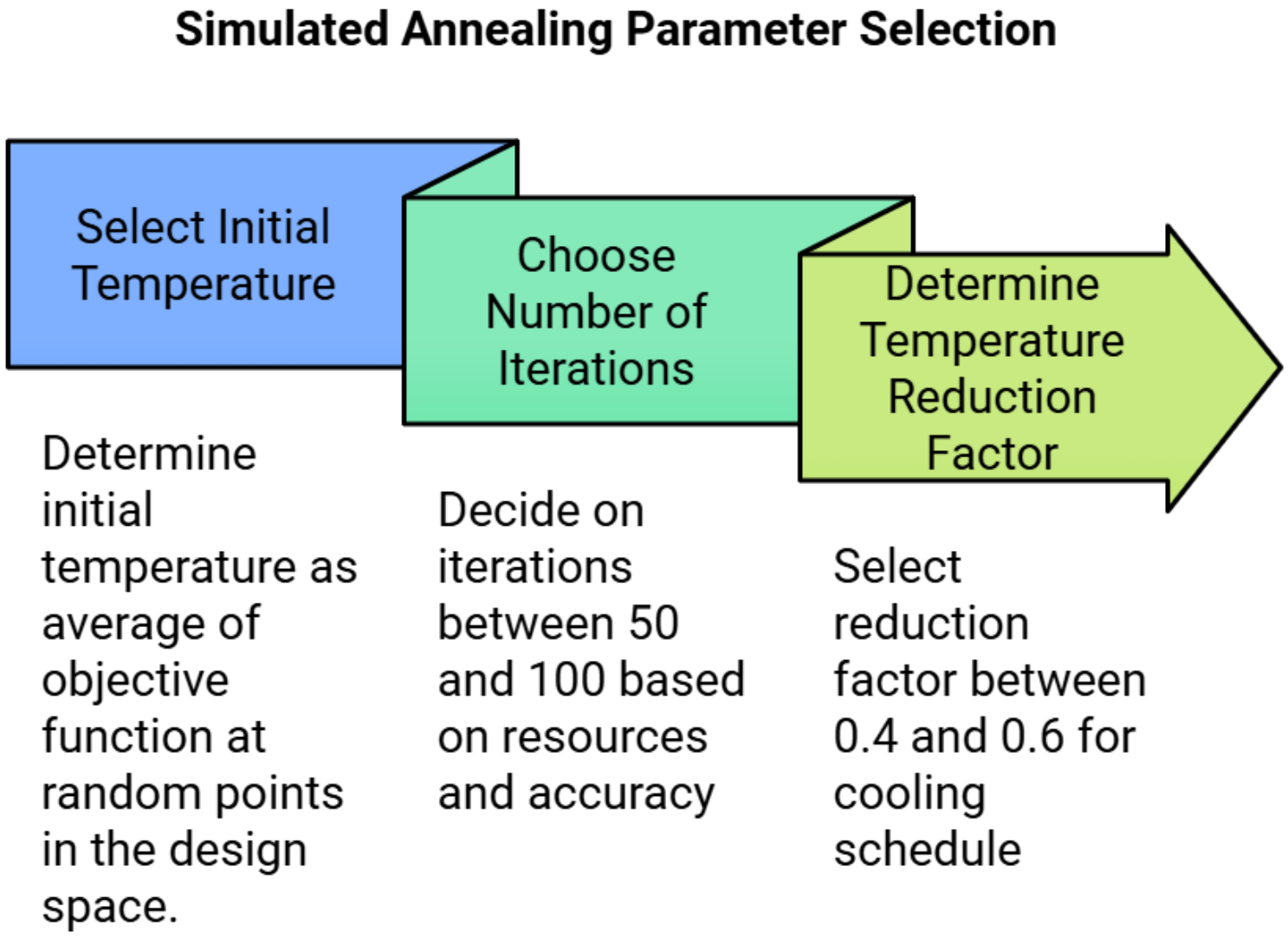
Balancing Computational Effort and Exploration Completeness

Choice of T, n, and c



- Similarly, a large value of the number of iterations n will help in achieving quasi-equilibrium state at each temperature but will result in a larger computational effort.
- A smaller value of n, on the other hand, might result either in a premature convergence or convergence to a local minimum (due to inadequate exploration of the design space for the global minimum).

Choice of T, n, and c



SA Math

Example 13.3 Find the minimum of the following function using simulated annealing:

$$f(\mathbf{X}) = 500 - 20x_1 - 26x_2 - 4x_1x_2 + 4x_1^2 + 3x_2^2$$

SOLUTION We follow the procedure indicated in the flowchart of Fig. 13.2.

Step 1: Choose the parameters of the SA method. The initial temperature is taken as the average value of f evaluated at four randomly selected points in the design space. By selecting the random points as $\mathbf{X}^{(1)} = \begin{Bmatrix} 2 \\ 0 \end{Bmatrix}$, $\mathbf{X}^{(2)} = \begin{Bmatrix} 5 \\ 10 \end{Bmatrix}$, $\mathbf{X}^{(3)} = \begin{Bmatrix} 8 \\ 5 \end{Bmatrix}$, $\mathbf{X}^{(4)} = \begin{Bmatrix} 10 \\ 10 \end{Bmatrix}$, we find the corresponding values of the objective function as $f^{(1)} = 476$, $f^{(2)} = 340$, $f^{(3)} = 381$, $f^{(4)} = 340$, respectively. Noting that the average value of the objective functions $f^{(1)}$, $f^{(2)}$, $f^{(3)}$, and $f^{(4)}$ is 384.25, we assume the initial temperature to be $T = 384.25$. The temperature reduction factor is chosen as $c = 0.5$. To make the computations brief, we choose the maximum permissible number of iterations (at any specific value of temperature) as $n = 2$. We select the initial design point as $\mathbf{X}_1 = \begin{Bmatrix} 4 \\ 5 \end{Bmatrix}$.

Step 2: Evaluate the objective function value at $\mathbf{X}_1$ as $f_1 = 349.0$ and set the iteration number as $i = 1$.

SA Math

Step 3: Generate a new design point in the vicinity of the current design point. For this, we select two uniformly distributed random numbers u_1 and u_2; u_1 for x_1 in the vicinity of 4 and u_2 for x_2 in the vicinity of 5. The numbers u_1 and u_2 are chosen as 0.31 and 0.57, respectively. By choosing the ranges of x_1 and x_2 as $(-2, 10)$ and $(-1, 11)$, which represent ranges of ± 6 about their respective current values, the uniformly distributed random numbers r_1 and r_2 in the ranges of x_1 and x_2, corresponding to u_1 and u_2, can be found as

$$r_1 = -2 + u_1\{10 - (-2)\} = -2 + 0.31(12) = 1.72$$

$$r_2 = -1 + u_2\{11 - (-1)\} = -1 + 0.57(12) = 5.84$$

which gives $\mathbf{X}_2 = \begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \begin{Bmatrix} 1.72 \\ 5.84 \end{Bmatrix}$.
Since the objective function value $f_2 = f(\mathbf{X}_2) = 387.7312$, the value of Δf is given by

$$\Delta f = f_2 - f_1 = 387.7312 - 349.0 = 38.7312$$

SA Math

Step 4: Since the value of Δf is positive, we use the Metropolis criterion to decide whether to accept or reject the current point. For this we choose a random number in the range (0, 1) as $r = 0.83$. Equation (13.18) gives the probability of accepting the new design point $\mathbf{X}_2$ as

$$P[\mathbf{X}_2] = e^{-\Delta f/kT} \tag{E$_1$}$$

By assuming the value of the Boltzmann's constant k to be 1 for simplicity in Eq. (E$_1$), we obtain

$$P[\mathbf{X}_2] = e^{-\Delta f/kT} = e^{-38.7312/384.25} = 0.9041$$

Since $r = 0.83$ is smaller than 0.9041, we accept the point $\mathbf{X}_2 = \begin{Bmatrix} 1.72 \\ 5.84 \end{Bmatrix}$ as the next design point. Note that, although the objective function value f_2 is larger than f_1, we accept $\mathbf{X}_2$ because this is an early stage of simulation and the current temperature is high.

Step 3: Update the iteration number as $i = 2$. Since the iteration number i is less than or equal to n, we proceed to step 3.

SA Math

Step 3: Generate a new design point in the vicinity of the current design point $\mathbf{X}_2 = \begin{Bmatrix} 1.72 \\ 5.84 \end{Bmatrix}$. For this, we choose the range of each design variable as ± 6 about its current value so that the ranges are given by $(-6 + 1.72, 6 + 1.72) = (-4.28, 7.72)$ for x_1 and $(-6 + 5.84, 6 + 5.84) = (-0.16, 11.84)$ for x_2. By selecting two uniformly distributed random numbers in the range (0, 1) as $u_1 = 0.92$ and $u_2 = 0.73$, the corresponding uniformly distributed random numbers in the ranges of x_1 and x_2 become

$$r_1 = -4.28 + u_1\{7.72 - (-4.28)\} = -4.28 + 0.92(12) = 6.76$$

$$r_2 = -0.16 + u_2\{11.84 - (-0.16)\} = -0.16 + 0.73(12) = 8.60$$

which gives $\mathbf{X}_3 = \begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \begin{Bmatrix} 6.76 \\ 8.60 \end{Bmatrix}$ with a function value of $f_3 = 313.3264$. We note that the function value f_3 is better than f_2 with $\Delta f = f_3 - f_2 = 313.3264 - 387.7312 = -74.4048$.

Step 4: Since $\Delta f < 0$, we accept the current point as $\mathbf{X}_3$ and increase the iteration number to $i = 3$. Since $i > n$, we go to step 5.

Step 5: Since a cycle of iterations with the current value of temperature is completed, we reduce the temperature to a new value of $T = 0.5\ (384.25) = 192.125$. Reset the current iteration number as $i = 1$ and go to step 3.

Step 3: Generate a new design point in the vicinity of the current design point $\mathbf{X}_3$ and continue the procedure until the temperature is reduced to a small value (until convergence).

Features of SA

- The quality of the final solution is not affected by the initial guesses, except that the computational effort may increase with worse starting designs.
- Because of the discrete nature of the function and constraint evaluations, the convergence or transition characteristics are not affected by the continuity or differentiability of the functions.
- The convergence is also not influenced by the convexity status of the feasible space.
- The design variables need not be positive.
- The method can be used to solve mixed-integer, discrete, or continuous problems.

Applications of Simulated Annealing

Traveling salesman problem: Finding the shortest route that visits a set of cities once and returns to the starting city.

Scheduling: Finding the optimal schedule for a set of jobs or tasks.

VLSI design: Placing and routing components on a chip.

Image processing: Segmenting images and optimizing image quality.

Machine learning: Training neural networks and other learning algorithms

Thank you