Contents

- Genetic Algorithms (GA)
- Differential Evolution Algorithm (DEA)
- Ant Colony Optimization (ACO) Algorithm
- Particle Swarm Optimization (PSO) Algorithm
- Simulated Annealing (SA)

Probabilistic Search Algorithms

- Disadvantages of most algorithms?
  - Inability to distinguish local and global minima
  - Discrete design variables: discontinuous and disjointed design space
  - Multiple minima

- How to solve?
  - Random search technique / enumerative type algorithm (X)
  - Simulated Annealing / Genetic Algorithms
    - Naturally observed phenomena
    - Use of random selection process guided by probabilistic decisions

Basic Concept (1)

- optimization algorithms inspired by natural phenomena
  - stochastic programming, evolutionary algorithms, genetic programming, swarm intelligence, evolutionary computation, nature-inspired metaheuristics methods
- general class of direct search methods
  - do not require the continuity or differentiability of problem functions
  - evaluate functions at any point within the allowable ranges for the design variables
- use stochastic ideas and random numbers in their calculations to search for the optimum point
  - execute at different times, the algorithms can lead to a different sequence of designs and a different solution even with the same initial conditions
  - tend to converge to a global minimum point for the function, but there is no guarantee of convergence or global optimality

Basic Concept (2)

- can overcome some of the challenges that are due to
  - multiple objectives, mixed design variables, irregular/noisy problem functions, implicit problem functions, expensive and/or unreliable function gradients, and uncertainty in the model and the environment
- very general and can be applied to all kinds of problems— discrete, continuous, and nondifferentiable
- relatively easy to use and program since they do not require the use of gradients of cost or constraint functions
- drawbacks of these algorithms
  - require a large amount of function evaluations → use of massively parallel computers
  - no absolute guarantee that a global solution has been obtained → execute the algorithm several times and allow it to run longer

Genetic Algorithm (GA)

- Search algorithm based on the mechanics of natural selection and natural genetics subject to Darwin's theory of "survival of the fittest" among string structures
  - Basic operations of natural genetics: reproduction, crossover, mutation
  - Mixed continuous-discrete variables, discontinuous and nonconvex design spaces (practical optimum design problems)
  - Global optimum solution with a high probability
    - J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, Mich., 1975
    - D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1983

$$\begin{aligned} &\text{maximize} \quad f(\boldsymbol{x}) \\ &\text{subject to} \quad l_i \le x_i \le u_i, \quad i = 1, \ldots, n \end{aligned}$$

GA: Terminology

- Gene: each design variable (x)
- Chromosome: group of design variables
- Individual: each design point
- Fitness: how good is the individual?
- Population: group of individuals

Individual

Chromosome [0.00,4.00,2.80,19.0] or [0000010011101101]
Fitness 25.8

- Genetic operators: drive the search
  - Selection: select the high fitness individuals, exploit the info
  - Crossover: parents create children, explore the design space
  - Mutation: sudden random changes in chromosomes
  - Inversion: reverse gene sequence (sometimes improve diversity)
- Generation: each cycle of genetic operations

GA: Characteristics

- A population of points is used for starting the procedure instead of a single design point.
  - Size of the population: 2n to 4n (n: # of DVs)
  - Less likely to get trapped at a local optimum
- GAs use only the values of the objective function.
  - No derivatives used in the search procedure
- Design variables are represented as strings of binary variables that correspond to the chromosomes in natural genetics.
  - Naturally applicable for solving discrete and integer programming problems
- The objective function value corresponding to a design vector plays the role of fitness in natural genetics.
- In every new generation, a new set of strings is produced by using randomized parents selection and crossover from the old generation.
  - Efficiently explore the new combinations with the available knowledge

Design Representation: Schema (1)

- it needs to be encoded (ie, defined)
  - binary encoding, real-number coding, integer encoding
- V–string for a binary string
  - represents the value of a variable
  - the component of a design vector (a gene)
- D–string for a binary string
  - represents a design of the system
  - particular combination of n V–strings (n: number of design variables
  - genetic string (or a chromosome)

Design Representation: Schema (2)

V-string $\Leftrightarrow$ discrete value of a variable having N_c allowable discrete values

let m be the smallest integer satisfying $2^m > N_c$

$$j = \sum_{i=1}^{m} ICH(i)\, 2^{(i-1)} + 1 \text{ where } ICH(i) : \text{value of the } i\text{-th digit } (\text{either 0 or 1})$$

$$\text{when } j > N_c,\ j = INT\left(\frac{N_c}{2^m - N_c}\right)(j - N_c)$$

$n = 3,\ N_c = 10 \rightarrow m = 4$

j value for three V-strings $\rightarrow \{7,16,14\} \rightarrow \{7,6,4\}$

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ |0110| & |1111| & |1101| \end{bmatrix} \qquad 3467\ 0254\ 7932\ 7612 \xrightarrow[5\sim9\to1]{0\sim4\to0} 0011\ 0010\ 1100\ 1100$$

Fitness Function: defines the relative importance of a design

$F_i = (1+\varepsilon) f_{\max} - f_i$ for example, $\varepsilon = 2\times10^{-7}$

f_i : cost function (penalty function value for a constrained problems) for the i-th design

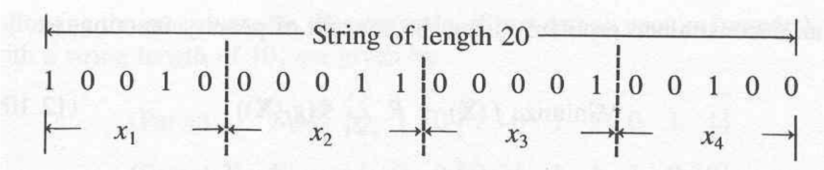
$f_{\max}$: largest recorded cost (penalty) function value

Genetic Operations

- Coding and decoding of design variables

$$\text{binary number}: b_q b_{q-1} \cdots b_2 b_1 b_0 \quad \textit{where} \;\; b_k = 0 \text{ or } 1$$

$$x = x_l + \frac{x_u - x_l}{2^q - 1} \underbrace{\sum_{k=0}^{q} 2^k b_k}_{\textit{decimal}}$$



$$2^q \geq \frac{x_u - x_l}{\Delta x} + 1 \quad (\Delta x : accuracy)$$

- Reproduction procedure
- Creation of a mating pool (selection)
  - The weaker members are replaced by stronger ones based on the fitness values.
  - Eg., Roulette wheel selection

Example

$$\text{Maximize } f(x_1, x_2) = 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2)$$

$$\text{subject to } \quad -3.0 \le x_1 \le 12.1$$

$$4.1 \le x_2 \le 5.8$$

binary coding with 5 decimal digits

$$x_1 : 2^{17} < \left[12.1 - (-3.0)\right] \times 10^5 = 151{,}000 \le 2^{18} \rightarrow q_1 = 18$$

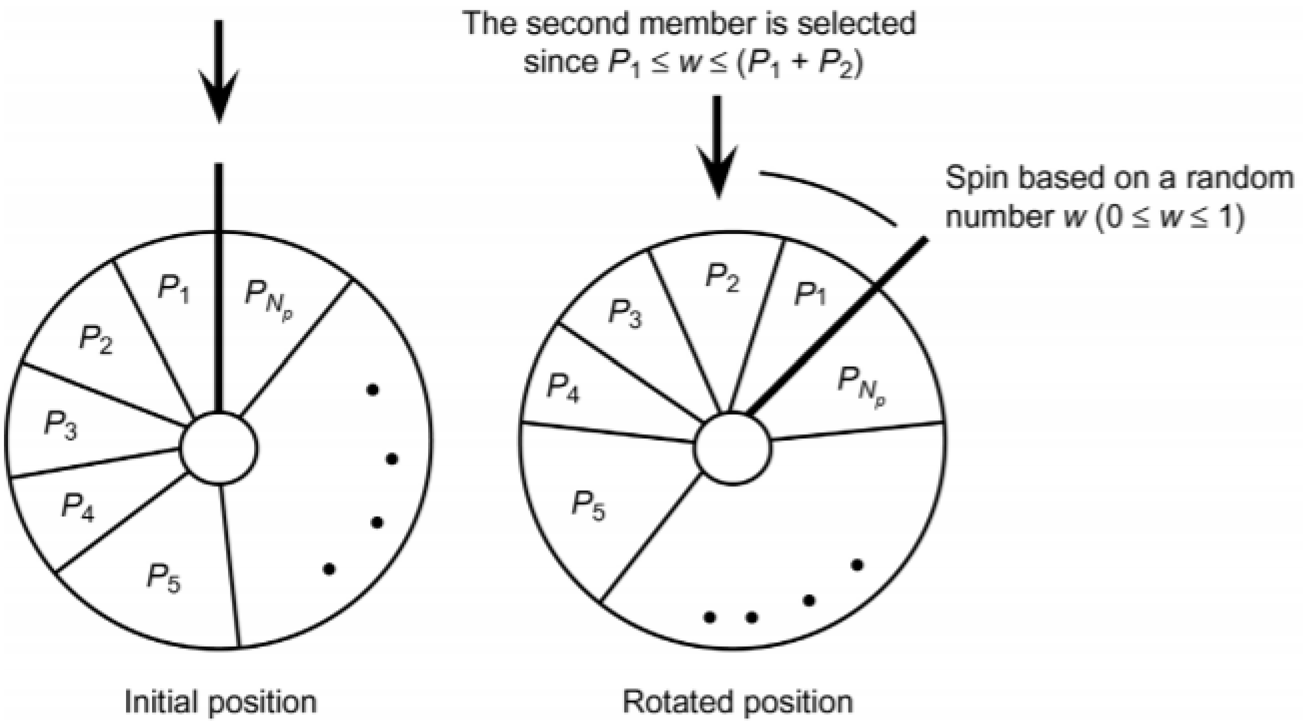
$$x_2 : 2^{14} < \left[5.8 - 4.1\right] \times 10^5 = 17{,}000 \le 2^{15} \rightarrow q_2 = 15$$

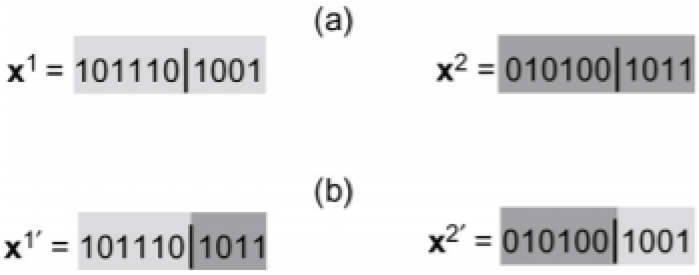
$$\text{chromosome:} \underbrace{000001010100101001}_{x_1 : 18 bit \rightarrow 5417} \underbrace{101110111111110}_{x_2 : 15 bit \rightarrow 24318}$$

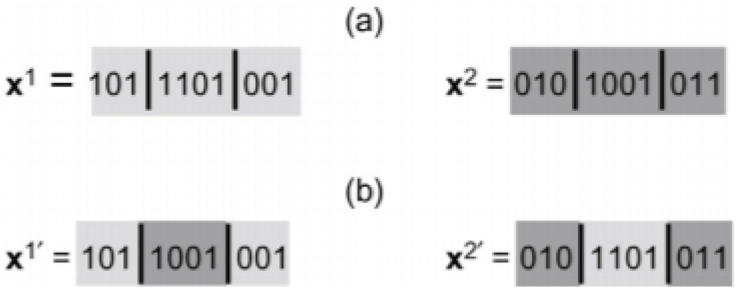
$$\rightarrow \begin{cases} x_1 = -3.0 + 5417 \times \dfrac{12.1 - (-3.0)}{2^{18} - 1} = -2.68797 \\ x_2 = 4.1 + 24318 \times \dfrac{5.8 - 4.1}{2^{15} - 1} = 5.36165 \end{cases}$$

Selection, Crossover

$$P_i = \frac{F_i}{\sum_{j=1}^{N_p} F_j}$$



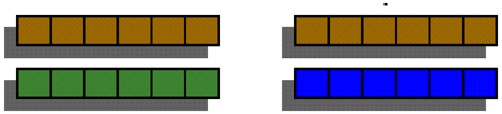




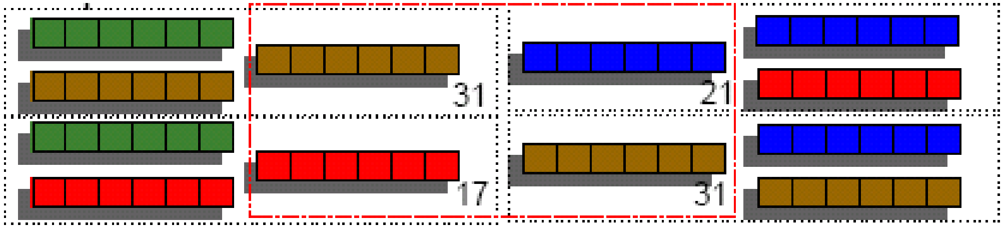
Reproduction / Selection

- Exploit the available information – drive to high fitness region
- Better individuals go to mating pool
- Population of high fitness individuals increased
- Remove low fitness individuals
- Improve the mean fitness
- Roulette wheel
  - Select individuals probabilistically

Individual	Fitness	Probability
🟩	8	0.10
🟦	21	0.27
🟫	31	0.40
🟥	17	0.22
	77	

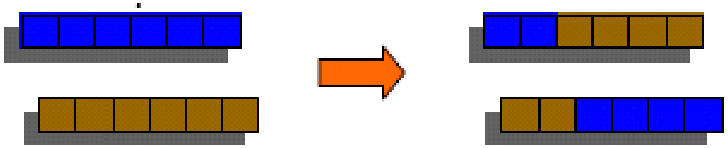


- Tournament selection
  - Compare two individuals at a time



Crossover

- Chromosomes are spliced – genes are shared



- Diversify the population – explore the design space
- Select parents randomly and mate them probabilistically(c_p)
- Binary crossover

$x_1^1 = 10$, $x_2^1 = 11$, 010|101011 → 010|000111 $y_1^1 = 08$, $y_2^1 = 07$

$x_1^2 = 20$, $x_2^2 = 07$, 101|000111 → 101|101011 $y_1^2 = 22$, $y_2^2 = 11$

- Real crossover

$$y_i^1 = \alpha_i x_i^1 + \beta_i x_i^2; \qquad y_i^2 = \beta_i x_i^1 + \alpha_i x_i^2$$

$$x_1^1 = 10, x_2^1 = 11; \qquad x_1^2 = 20, x_2^2 = 07$$

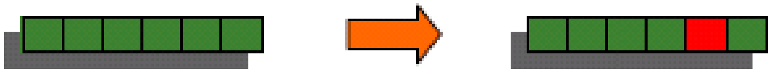
$$\alpha_1 = 0.5, \beta_1 = 1.25; \qquad \alpha_2 = 1.5, \beta_2 = -0.75$$

$$y_1^1 = 0.5x_1^1 + 1.25x_1^2 = 30; \quad y_2^1 = 1.5x_2^1 - 0.75x_2^2 = 11.25$$

$$y_1^2 = 1.25x_1^1 + 0.5x_1^2 = 22.5; \quad y_2^2 = -0.75x_2^1 + 1.5x_2^2 = 2.25$$

Mutation / Inversion

- Some genes change randomly
- Sometimes these changes are favorable
- Select genes(bits) probabilistically for mutation(m_p:0.005~0.1)
- Binary mutation

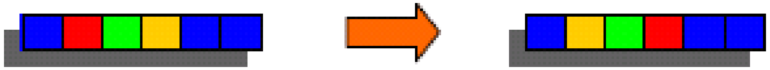


10110111 [x_1=11,x_2=7] → 00110011 [x_1=11,x_2=3]

- Real mutation

$$\boxed{y_i = x_i + \delta\Delta x_i}; x_2 = 11;\ \Delta x = 1, \delta = 0.5;\ y_i = 11 + 0.5 = 11.5$$

- Inversion: seldom used

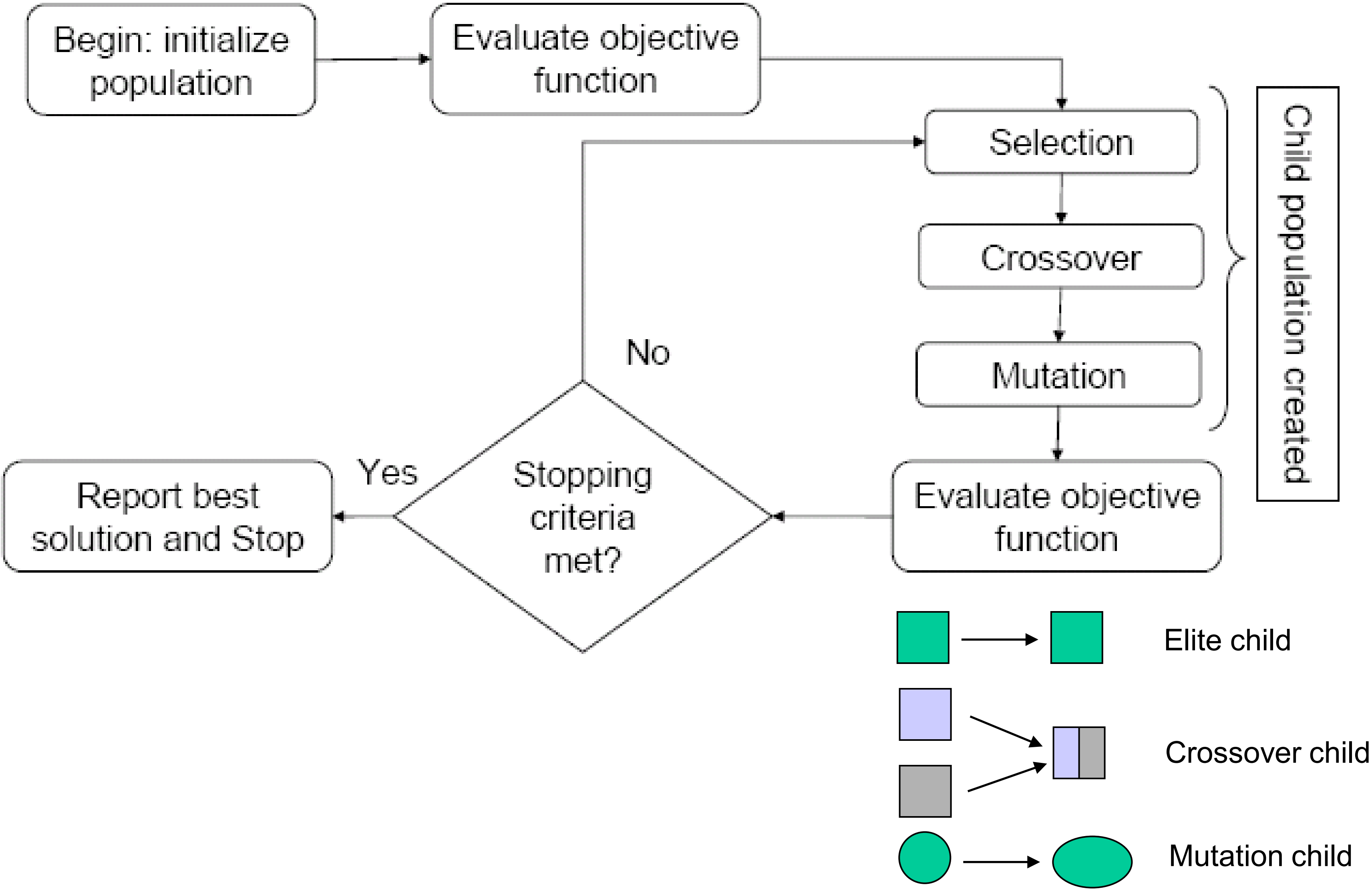


10110101 [x_1=11,x_2=5]   10011101 [x_1=9,x_2=13]

Stopping Criteria

- Number of generations
- Number of function evaluations
- No improvement for a certain number of generations

Flowchart of Simple Genetic Algorithm



Begin: initialize population

Evaluate objective function

Selection

Crossover

Mutation

Child population created

Evaluate objective function

Stopping criteria met?

No

Yes

Report best solution and Stop

Elite child

Crossover child

Mutation child

Observations

- Advantages
  - Gaining ground as practical tools
  - Relatively simple and elegant because of their analogy with nature
  - Public domain and commercial codes exist
- Disadvantages
  - A number of control parameters which affect the efficiency of solution
  - Large number of fitness function evaluations
  - Not strictly guarantee the optimality of the solution
  - Unconstrained minimization algorithm→penalty function?

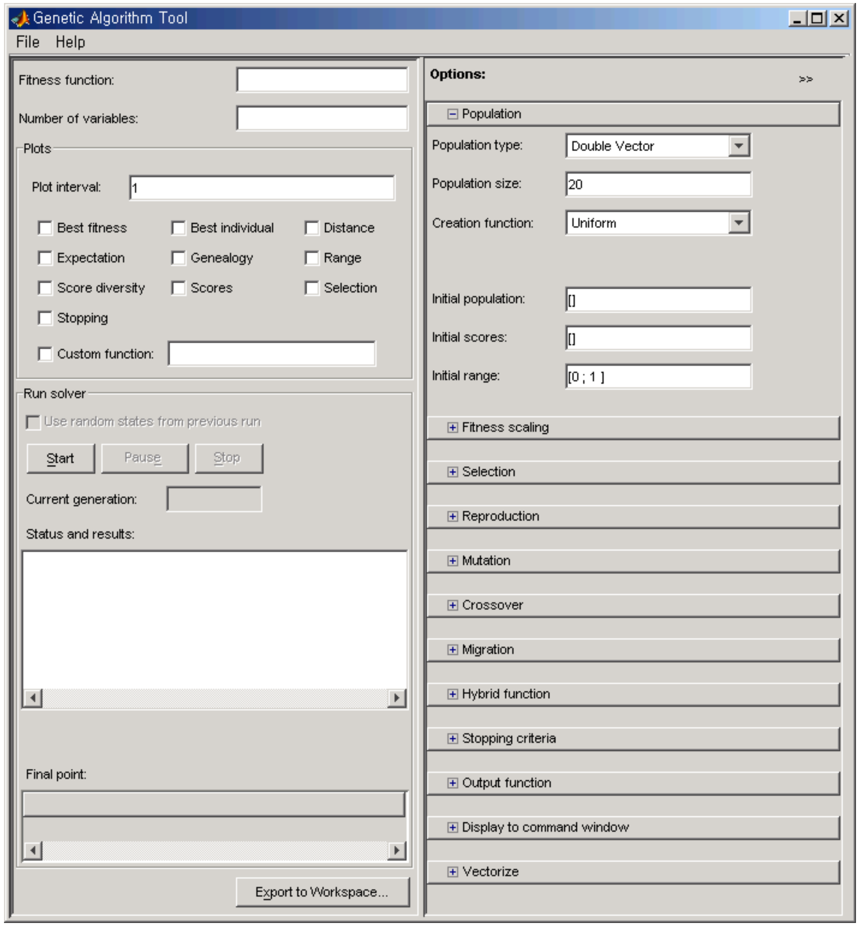
$$\text{minimize } f(\boldsymbol{x}) + R\sum_{j=1}^{m} \langle g_j(\boldsymbol{x}) \rangle^2$$

$$\text{subject to } x_l \le x_i \le x_u, \quad i = 1, \ldots, n$$

$$\Rightarrow \text{maximize } F(\boldsymbol{x}) = F_{\max} - \left[f(\boldsymbol{x}) + R\sum_{j=1}^{m} \langle g_j(\boldsymbol{x}) \rangle^2 \right]$$

MATLAB: Genetic Algorithm Toolbox

- [x fval] = ga(@fitnessfun, nvars, options)
- gatool



Plot interval: 1

Population type: Double Vector

Population size: 20

Creation function: Uniform

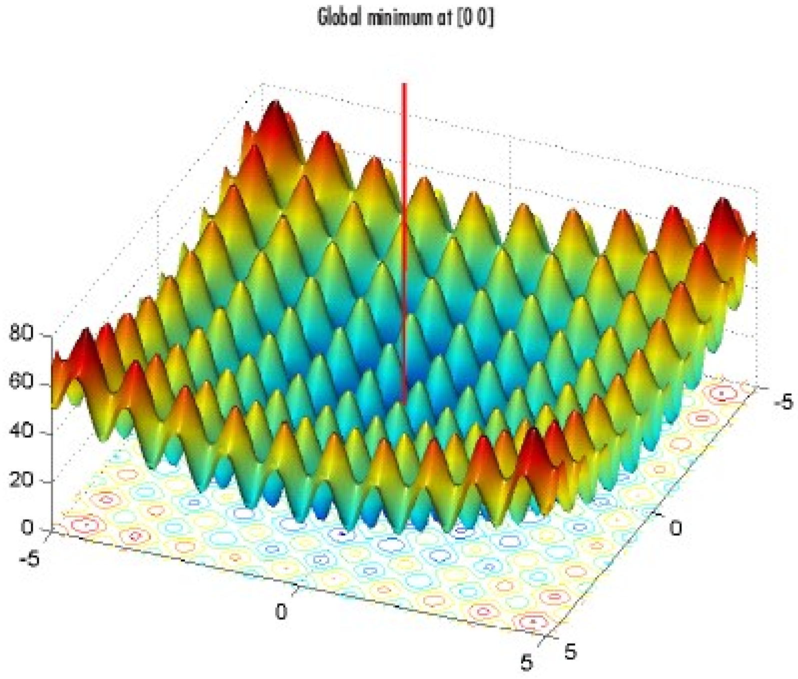
Initial population: []

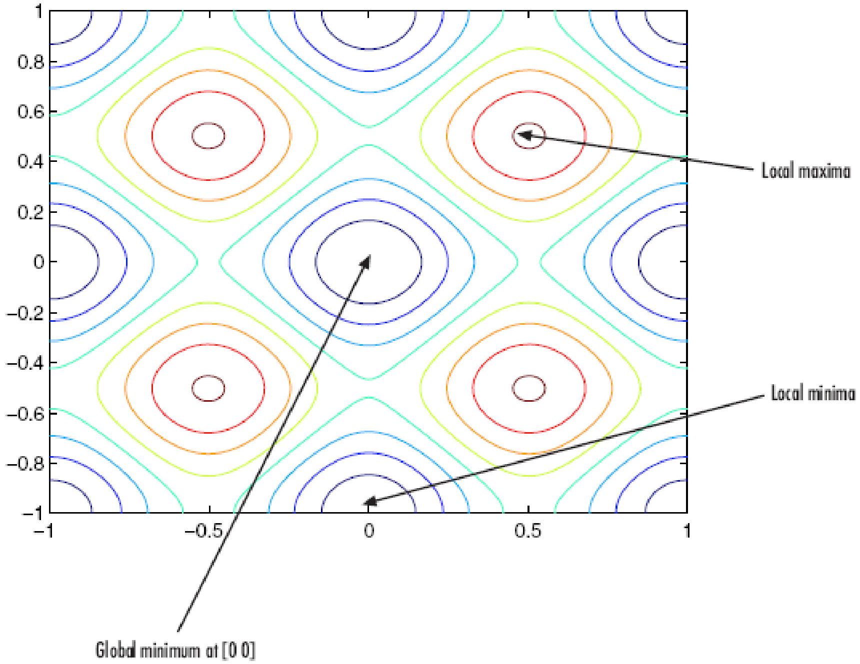
Initial scores: []

Initial range: [0 ; 1]

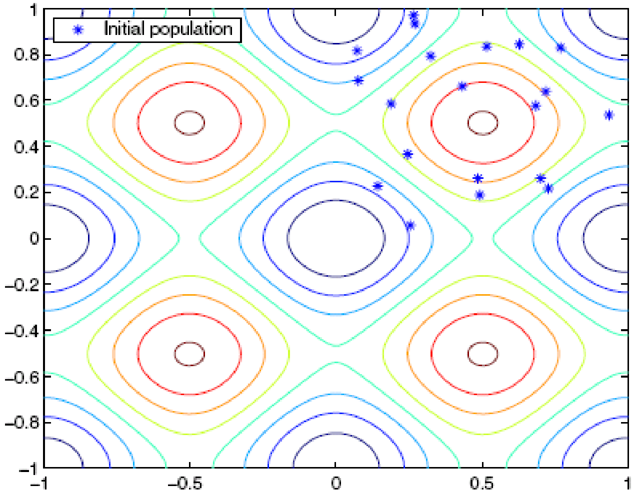
Rastrigin’s Function

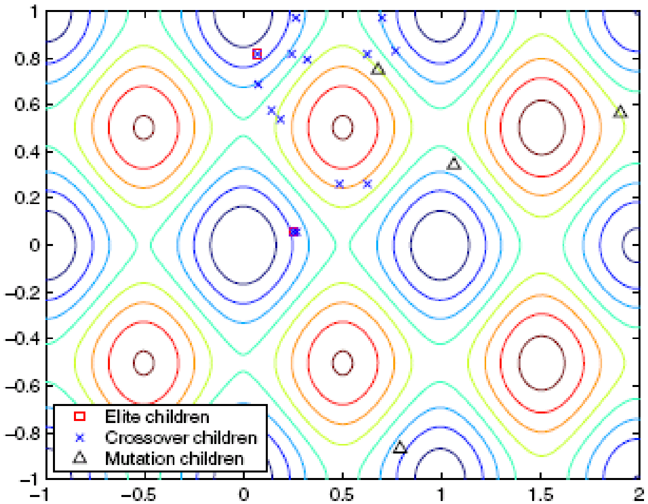
$$Ras(\boldsymbol{x}) = 20 + x_1{}^2 + x_2{}^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2)$$

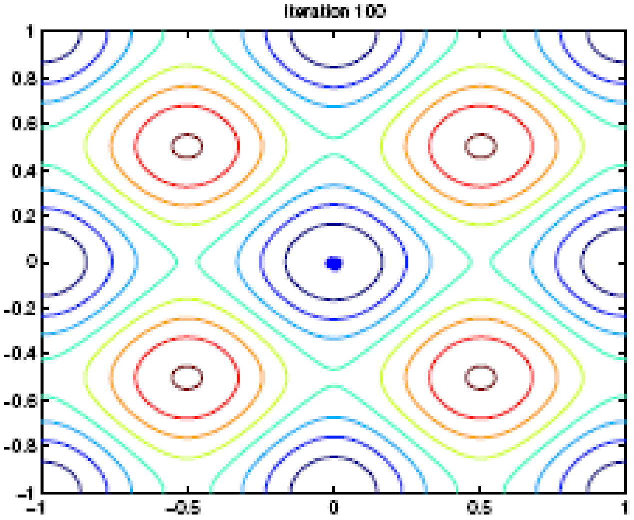


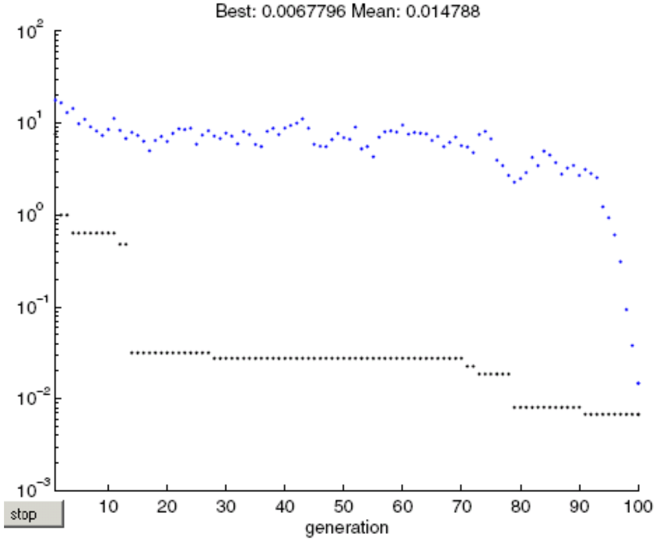


Iteration History



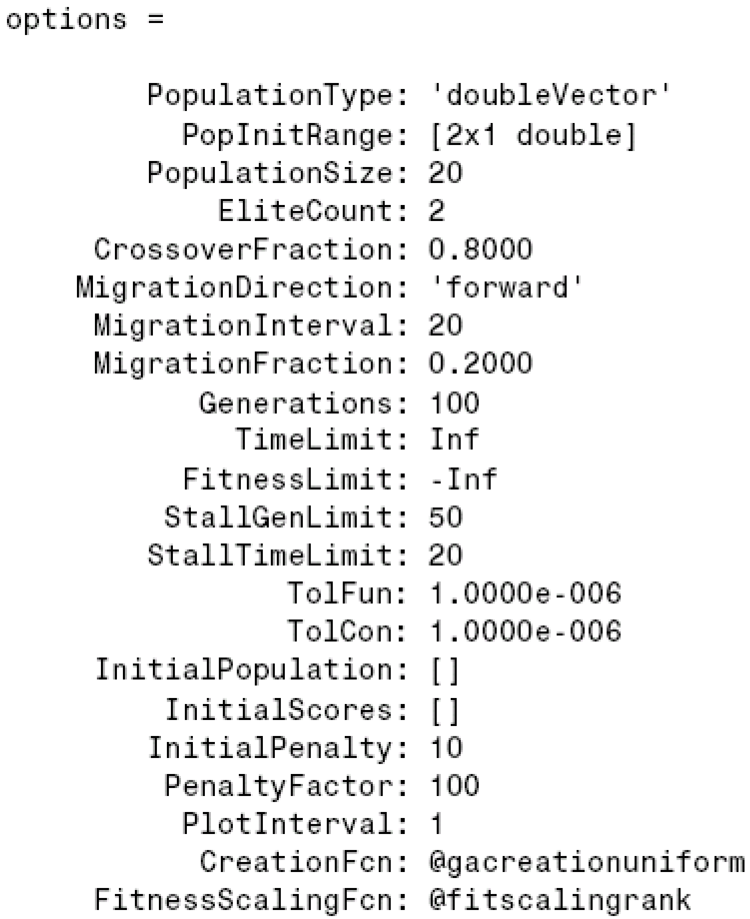






Options

- options = gaoptimset(‘option-item’, value)



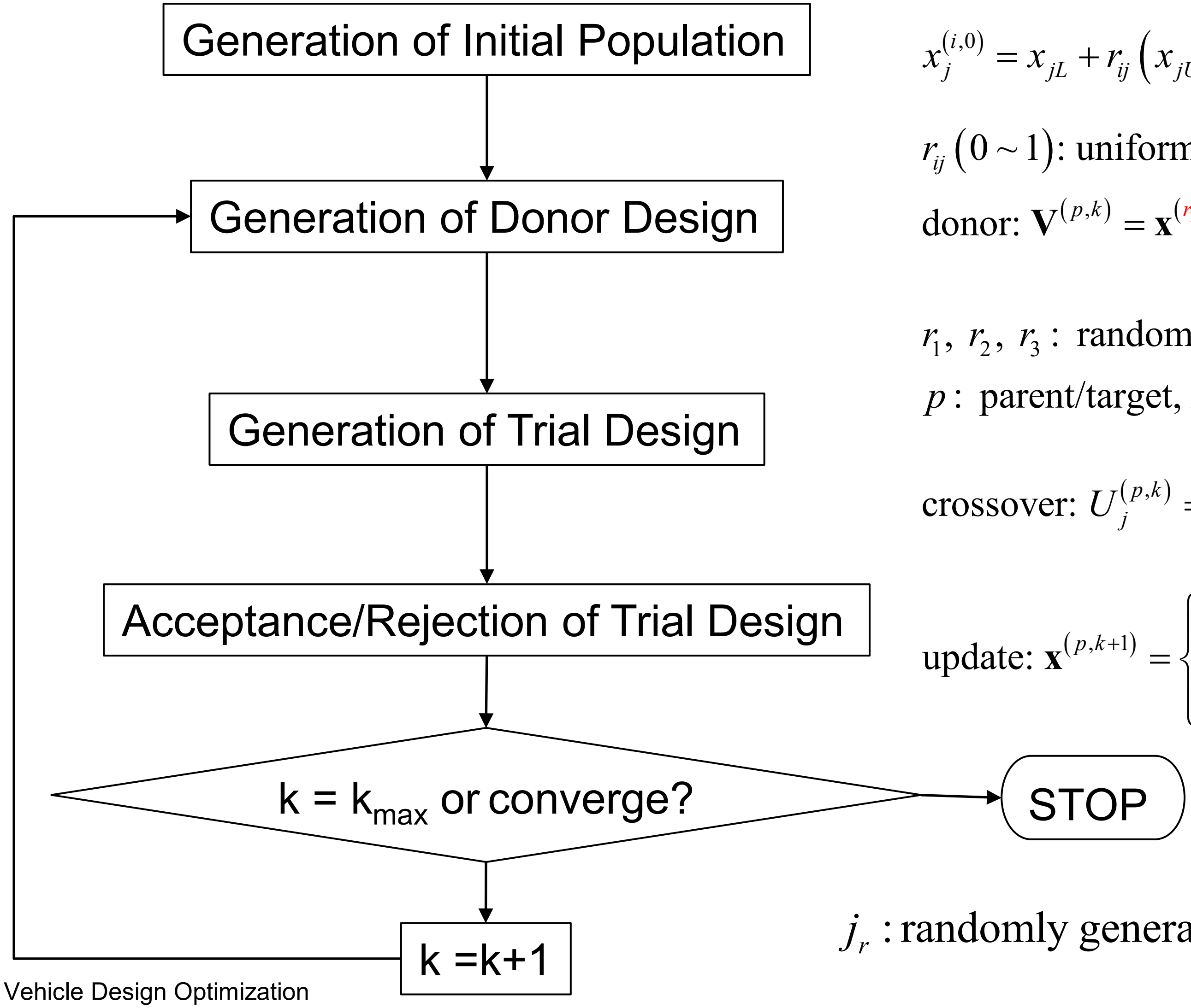
```
options =

        PopulationType: 'doubleVector'
          PopInitRange: [2x1 double]
        PopulationSize: 20
            EliteCount: 2
     CrossoverFraction: 0.8000
    MigrationDirection: 'forward'
     MigrationInterval: 20
     MigrationFraction: 0.2000
           Generations: 100
             TimeLimit: Inf
          FitnessLimit: -Inf
         StallGenLimit: 50
        StallTimeLimit: 20
                TolFun: 1.0000e-006
                TolCon: 1.0000e-006
     InitialPopulation: []
         InitialScores: []
        InitialPenalty: 10
         PenaltyFactor: 100
          PlotInterval: 1
           CreationFcn: @gacreationuniform
     FitnessScalingFcn: @fitscalingrank
```

Differential Evolution Algorithm (DEA)

  - Compared to GAs, DEAs are easier to implement on the computer
  - Unlike GAs, they do not require binary number coding and encoding
- Four steps in executing the basic DEA
  - Step 1: Generation of the initial population of designs
  - Step 2: Mutation with difference of vectors to generate a so-called donor design vector
  - Step 3: Crossover/recombination to generate a so-called trial design vector
  - Step 4: Selection, that is, acceptance or rejection of the trial design vector using the fitness function, which is usually the cost function

Differential Evolution Algorithm (DEA)



$$N_p = 5n \sim 10n$$

$$x_j^{(i,0)} = x_{jL} + r_{ij}\left(x_{jU} - x_{jL}\right) \begin{cases} i\text{-th member of poulation} \\ j = 1, \ldots, n \end{cases}$$

$r_{ij}\left(0 \sim 1\right)$: uniformly distributed random number

$$\text{donor: } \mathbf{V}^{(p,k)} = \mathbf{x}^{(r_1,k)} + F\underbrace{\left(\mathbf{x}^{(r_2,k)} - \mathbf{x}^{(r_3,k)}\right)}_{\text{difference vector}}$$

$r_1,\ r_2,\ r_3$: random three distinct design points

p : parent/target, $F\left(0.4 \sim 1\right)$: scale factor

$$\text{crossover: } U_j^{(p,k)} = \begin{cases} V_j^{(p,k)} \text{ if } \left(r_{pj} \le C_r\right) \text{ or } \left(j = j_r\right) \\ x_j^{(p,k)} \text{ otherwise} \end{cases}$$

$$\text{update: } \mathbf{x}^{(p,k+1)} = \begin{cases} \mathbf{U}^{(p,k)} \text{ if } f\left(\mathbf{U}^{(p,k)}\right) \le f\left(\mathbf{x}^{(p,k)}\right) \\ \mathbf{x}^{(p,k)} \text{ otherwise} \end{cases}$$

j_r : randomly generated index between 1 and n

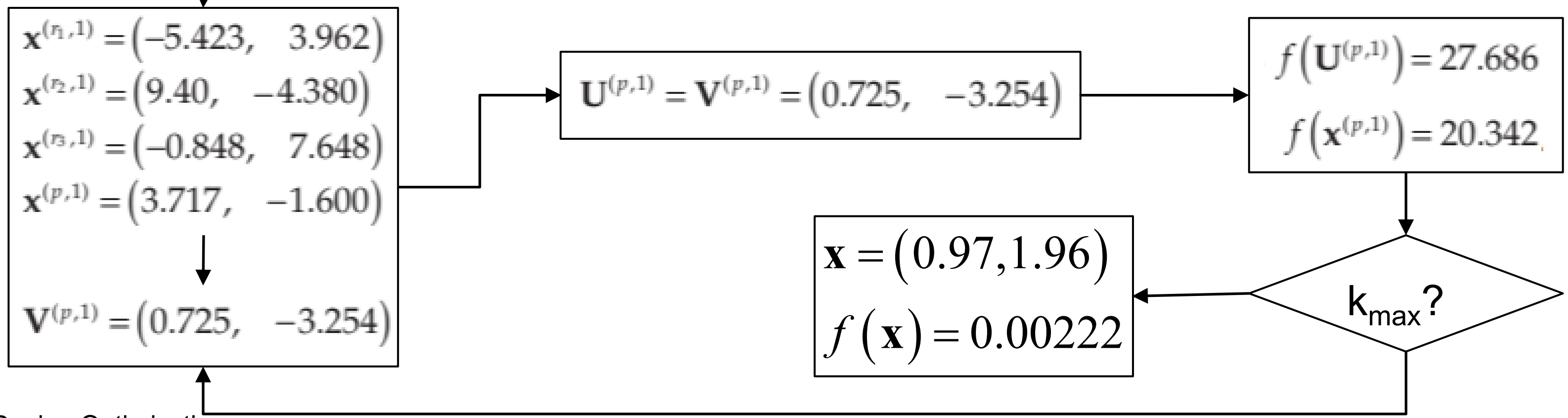
Example: DEA

$$\text{Minimize } f(\mathbf{x}) = (x_1 - 1)^2 + (x_2 - 2)^2$$

$$\text{subject to } -10 \le x_1 \le 10, \ \ -10 \le x_2 \le 10$$

$$\begin{cases} n = 2 \\ N_p = 5n = 10 \\ k_{\max} = 10{,}000 \\ C_r = 0.8 \\ F = 0.6 \end{cases}$$

x_i *number*	x_1	x_2
1	3.717	−1.600
2	9.400	−4.380
3	9.048	−8.659
4	−2.935	−2.920
5	−5.423	3.962
6	−4.442	2.470
7	−0.848	7.648
8	−8.394	−5.238
9	2.678	−2.884
10	7.059	−1.567



$$\mathbf{x}^{(r_1,1)} = (-5.423, \ \ 3.962)$$
$$\mathbf{x}^{(r_2,1)} = (9.40, \ \ -4.380)$$
$$\mathbf{x}^{(r_3,1)} = (-0.848, \ \ 7.648)$$
$$\mathbf{x}^{(p,1)} = (3.717, \ \ -1.600)$$
$$\downarrow$$
$$\mathbf{V}^{(p,1)} = (0.725, \ \ -3.254)$$

$$\mathbf{U}^{(p,1)} = \mathbf{V}^{(p,1)} = (0.725, \ \ -3.254)$$

$$f\left(\mathbf{U}^{(p,1)}\right) = 27.686$$
$$f\left(\mathbf{x}^{(p,1)}\right) = 20.342$$

k_{max}?

$$\mathbf{x} = (0.97, 1.96)$$
$$f(\mathbf{x}) = 0.00222$$

Ant Colony Optimization (ACO)

- emulates the food searching behavior of ants developed by Dorigo (1992)
- search for an optimal path for a problem represented by a graph based on the behavior of ants seeking the shortest path between their colony and a food source
- class of metaheuristics and swarm intelligence methods
- originally for discrete variable combinatorial optimization problems

ACO Terminology

- Pheromone: pherin (to transport) + hormone (to stimulate)
  - a secreted or excreted chemical factor that triggers a social response in members of the same species
- Pheromone trail
  - ants deposit pheromones wherever they go
  - other ants can smell the pheromones and are likely to follow an existing trail
- Pheromone density
  - when ants travel on the same path again and again, they continuously deposit pheromones on it
  - In this way the amount of pheromones increases and ants are likely to follow paths having higher pheromone densities
- Pheromone evaporation
  - pheromones have the property of evaporation over time
  - if a path is not being traveled by the ants, the pheromones evaporate, and the path disappears over time

Ant Behavior

- Initially ants move from their nest randomly to search for food
- Upon finding it, they return to their colony following the path they took to it while laying down pheromone trails
- If other ants find such a path, they are likely to follow it instead of moving randomly
- The path is thus reinforced, since ants deposit more pheromone on it
- However, the pheromone evaporates over time

- pheromone density is higher on shorter paths than on the longer ones → eventually all the ants follow the shortest path

Virtual Ants: Simple Model

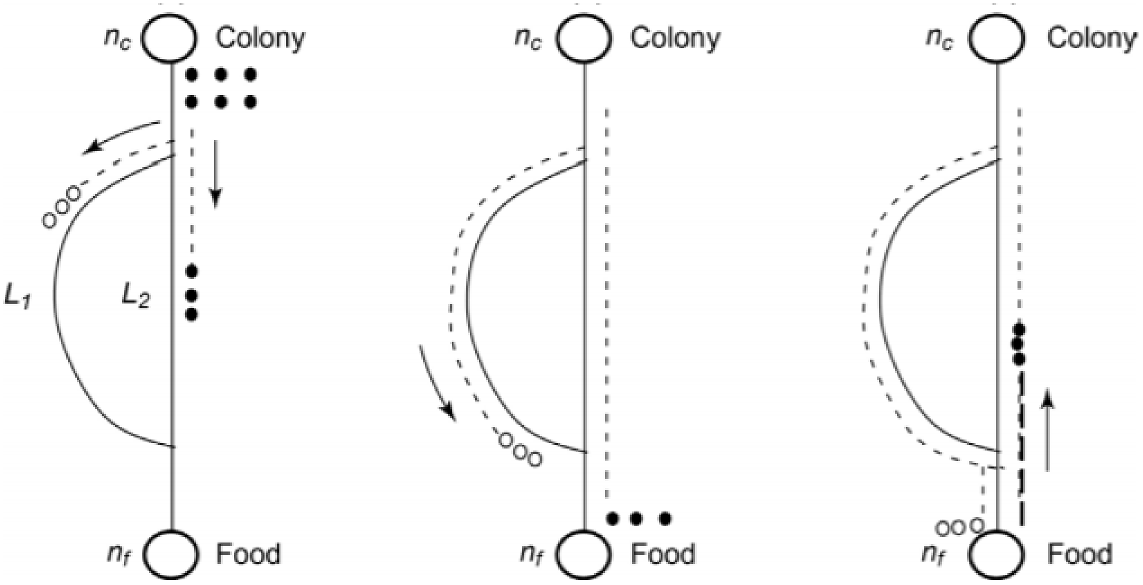
$$\underset{\text{graph}}{G} = \left(N, L\right) \text{ where } N = \text{node}\begin{cases} n_c : \text{ ant colony} \\ n_f : \text{food source} \end{cases} \text{ and } L = \text{link}\begin{cases} L_1 : \text{ length of } d_1 \\ L_2 : \text{ length of } d_2 \end{cases} \left(d_1 > d_2\right)$$

virtual pheromone value $\left(\tau_i\right)$ for *i*th-path (initially set as one): strength of the pheromone trail

$n_c \rightarrow n_f$ (finding): selection of the path based on the probability $p_i = \dfrac{\tau_i}{\tau_1 + \tau_2}$, $i = 1, 2$

$n_f \rightarrow n_c$ (returning): pheromone reinforcement $\tau_i \leftarrow \tau_i + \dfrac{Q}{d_i}$ where Q : positive constant

evaporation : $\tau_i \leftarrow \left(1-\rho\right)\tau_i$ where ρ : pheromone evaporation rate, $\rho \in \left(0,1\right]$



Traveling Salesman Problem (1)

- classical combinatorial optimization problem
  - traveling salesman is required to visit a specified number of cities (called a tour)
  - The goal is to visit a city only once while minimizing the total distance traveled
- Assumptions
  - While a real ant can take a return path to the colony that is different from the original path depending on the pheromone values, a virtual ant takes the return path that is the same as the original path
  - The virtual ant always finds a feasible solution and deposits pheromone only on its way back to the nest
  - While real ants evaluate a solution based on the length of the path from their nest to the food source, virtual ants evaluate their solution based on a cost function value

Traveling Salesman Problem (2)

- "finding a path from the nest to the food source" to "finding a feasible solution to the TS problem."

x_j : j-th component of the design variable vector $\mathbf{x}$ (link selected from the j-th city)

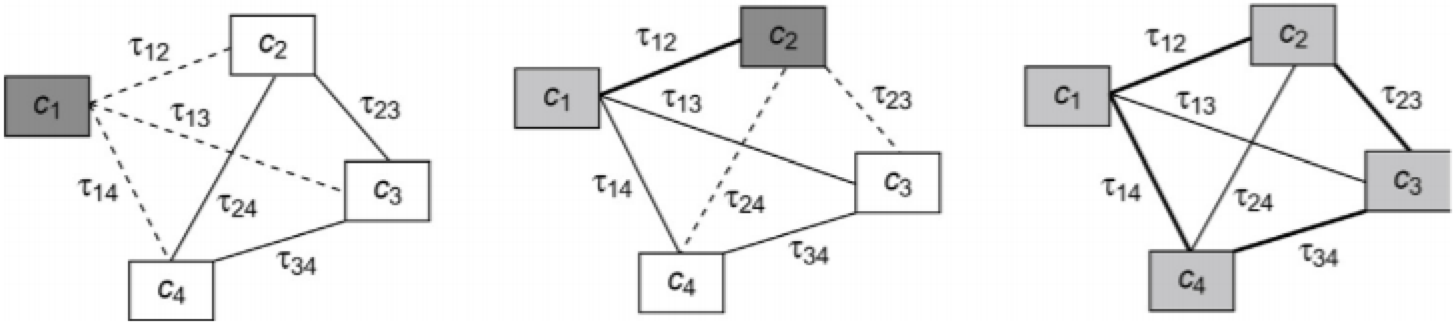
x_{ij} : link between the i-th city and the j-th city (distance between them)

D_i : list of integers corresponding to the cities that can be visited from the i-th city

$$D_1 = \{2,3,4\} \Leftrightarrow \text{feasible links} \{x_{12}, x_{13}, x_{14}\} \rightarrow p_{1j} = \frac{\tau_{1j}}{\tau_{12} + \tau_{13} + \tau_{14}}$$

$$D_2 = \{3,4\} \Leftrightarrow \text{feasible links} \{x_{23}, x_{24}\} \rightarrow p_{2j} = \frac{\tau_{2j}}{\tau_{23} + \tau_{24}}$$

$$c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_4 \rightarrow c_1 \Leftrightarrow \mathbf{x} = \begin{bmatrix} x_{12} & x_{23} & x_{34} & x_{41} \end{bmatrix}$$



Design Optimization: ACO Algorithm (1)

- Problem definition
  - unconstrained discrete variable design optimization problem

$$\left.\begin{array}{l}\text{Minimize } f(\mathbf{x}) \\ x_i \in D_i = \left(d_{i1},\ldots,d_{iq_i}\right) \quad i=1,\ldots n\end{array}\right\} \leftarrow \text{only parameters: } N_a, \rho, Q$$

- Finding feasible solutions
  - Selection of an initial link
  - Selection of a link from layer R
  - Obtaining feasible solutions for all ants

$$p_{1j}^{(00)} = \frac{\tau_{1j}^{(00)}}{\sum_{r=1}^{q_i} \tau_{1r}^{(00)}};\ j=1,\ldots,q_i$$

$$p_{ij}^{(rs)} = \frac{\tau_{ij}^{(rs)}}{\sum_{l=1}^{q_i} \tau_{il}^{(rs)}};\ \begin{cases} j=1,\ldots,q_i \\ i=r+1 \end{cases}$$

$$\mathbf{x}^{(k)}, f\left(\mathbf{x}^{(k)}\right);\ k=1,\ldots,\ N_a$$

$5n\sim10n$

$\tau_{ij}^{(rs)}$: pheromone value for the link from node rs to node ij

$p_{ij}^{(rs)}$: probability of selection of the link from node rs to node ij

$$\begin{cases} r: \text{layer number (design variable number)} \\ s: \text{allowable value number for the design variable number } r \end{cases}$$

Design Optimization : ACO Algorithm (2)

- Pheromone Evaporation
  - Once all of the ants have reached their destination (all of them have found solutions), pheromone evaporation (ie, reduction in the pheromone level) is performed for all links

$$\tau_{ij}^{(rs)} \leftarrow \left(1 - \underset{0.4\sim0.8}{\rho}\right)\tau_{ij}^{(rs)} \text{ for all } r, s, i, j$$

- Pheromone Deposit
  - After pheromone evaporation, the ants start their journey back to their nest, which means that they will deposit pheromone on the return trail

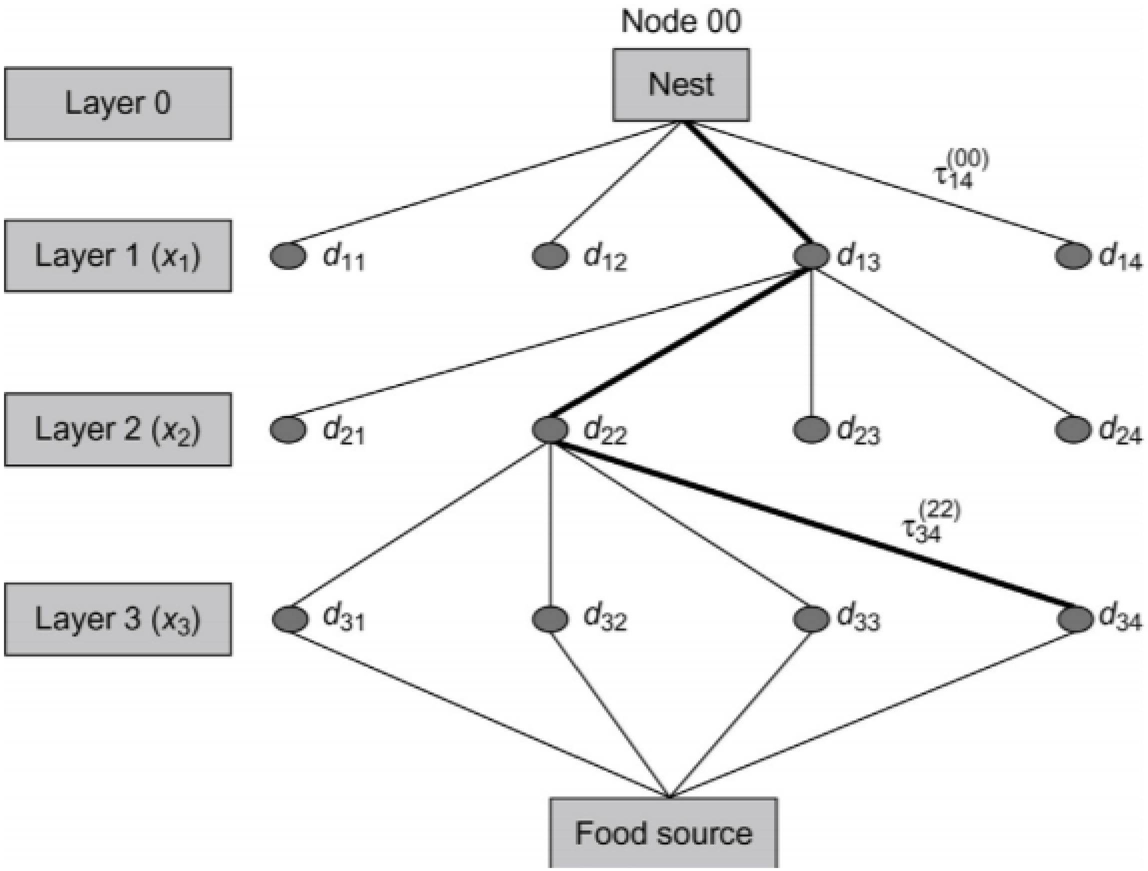
$$\tau_{ij}^{(rs)} \leftarrow \tau_{ij}^{(rs)} + \frac{Q}{f\left(\mathbf{x}^{(k)}\right)} \text{ for all } r, s, i, j \text{ belonging to } k\text{-th ant's solution}$$

Example: ACO

$$\text{Minimize } f(\mathbf{x})$$

$$x_i \in D_i = \left(d_{i1}, \ldots, d_{iq_i}\right) \quad i = 1, \ldots n$$

$$n = 3,\ q_i = 4$$



Particle Swarm Optimization (PSO)

- Population-based stochastic optimization technique, introduced by Kennedy and Eberhart (1995)
- Mimics the social behavior of bird flocking or fish schooling
- Class of metaheuristics and swarm intelligence methods
- Many similarities with evolutionary computation techniques such as GA and DE
  - Starts with a randomly generated set of solutions (initial population)
  - An optimum solution is then searched by updating generations
  - Fewer algorithmic parameters to specify compared to GAs
  - Not use any of the GAs' evolutionary operators (crossover, mutation) → easier to implement

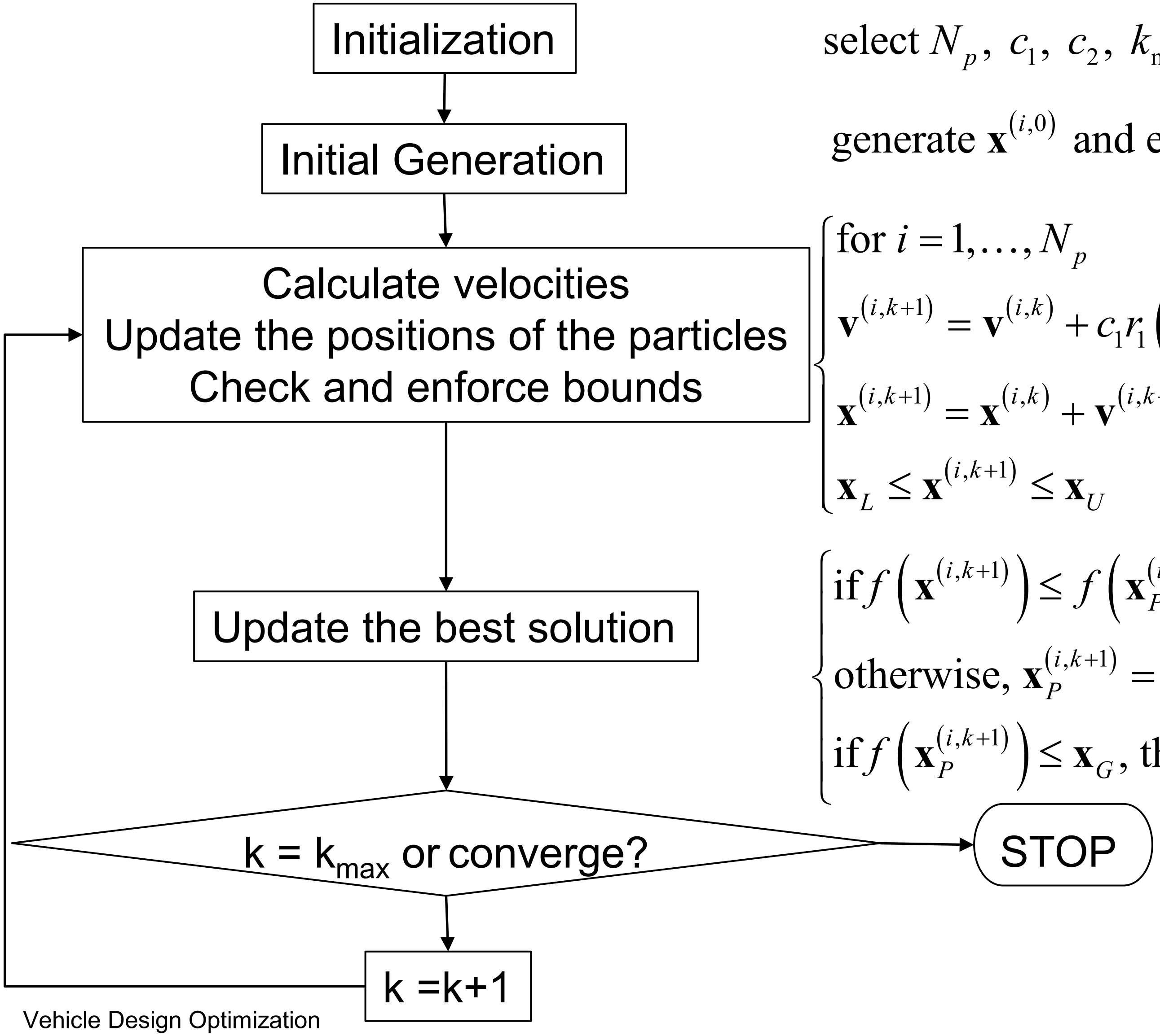
Swarm Behavior

- Emulate the social behavior of a swarm of animals, such as a flock of birds or a school of fish (moving in search for food)
- An individual behaves according to its limited intelligence as well as to the intelligence of the group
- Each individual observes the behavior of its neighbors and adjusts its own behavior accordingly
- If an individual member discovers a good path to food, other members follow this path no matter where they are situated in the swarm

PSO Terminology

- Particle: identify an individual in the swarm (eg, a bird in the flock or a fish in the school)
- Particle position: refers to the coordinates of the particle ↔ design point
- Particle velocity: The term refers to the rate at which the particles are moving in space ↔ design change
- Swarm leader: particle having the best position ↔ design point having the smallest value for the cost function

Particle Swarm Optimization Algorithm



select $N_p,\ c_1,\ c_2,\ k_{\max}$. set $\mathbf{v}^{(i,0)} = 0,\ k = 1.$

generate $\mathbf{x}^{(i,0)}$ and evaluate $f\left(\mathbf{x}^{(i,0)}\right) \rightarrow \mathbf{x}_G^{(k)}$

$$\begin{cases} \text{for } i = 1, \ldots, N_p \\ \mathbf{v}^{(i,k+1)} = \mathbf{v}^{(i,k)} + c_1 r_1 \left(\mathbf{x}_P^{(i,k)} - \mathbf{x}^{(i,k)}\right) + c_2 r_2 \left(\mathbf{x}_G^{(i,k)} - \mathbf{x}^{(i,k)}\right) \\ \mathbf{x}^{(i,k+1)} = \mathbf{x}^{(i,k)} + \mathbf{v}^{(i,k+1)} \\ \mathbf{x}_L \le \mathbf{x}^{(i,k+1)} \le \mathbf{x}_U \end{cases}$$

$$\begin{cases} \text{if } f\left(\mathbf{x}^{(i,k+1)}\right) \le f\left(\mathbf{x}_P^{(i,k)}\right), \text{ then } \mathbf{x}_P^{(i,k+1)} = \mathbf{x}^{(i,k+1)} \\ \text{otherwise, } \mathbf{x}_P^{(i,k+1)} = \mathbf{x}_P^{(i,k)} \\ \text{if } f\left(\mathbf{x}_P^{(i,k+1)}\right) \le \mathbf{x}_G, \text{ then } \mathbf{x}_G = \mathbf{x}_P^{(i,k+1)} \end{cases}$$

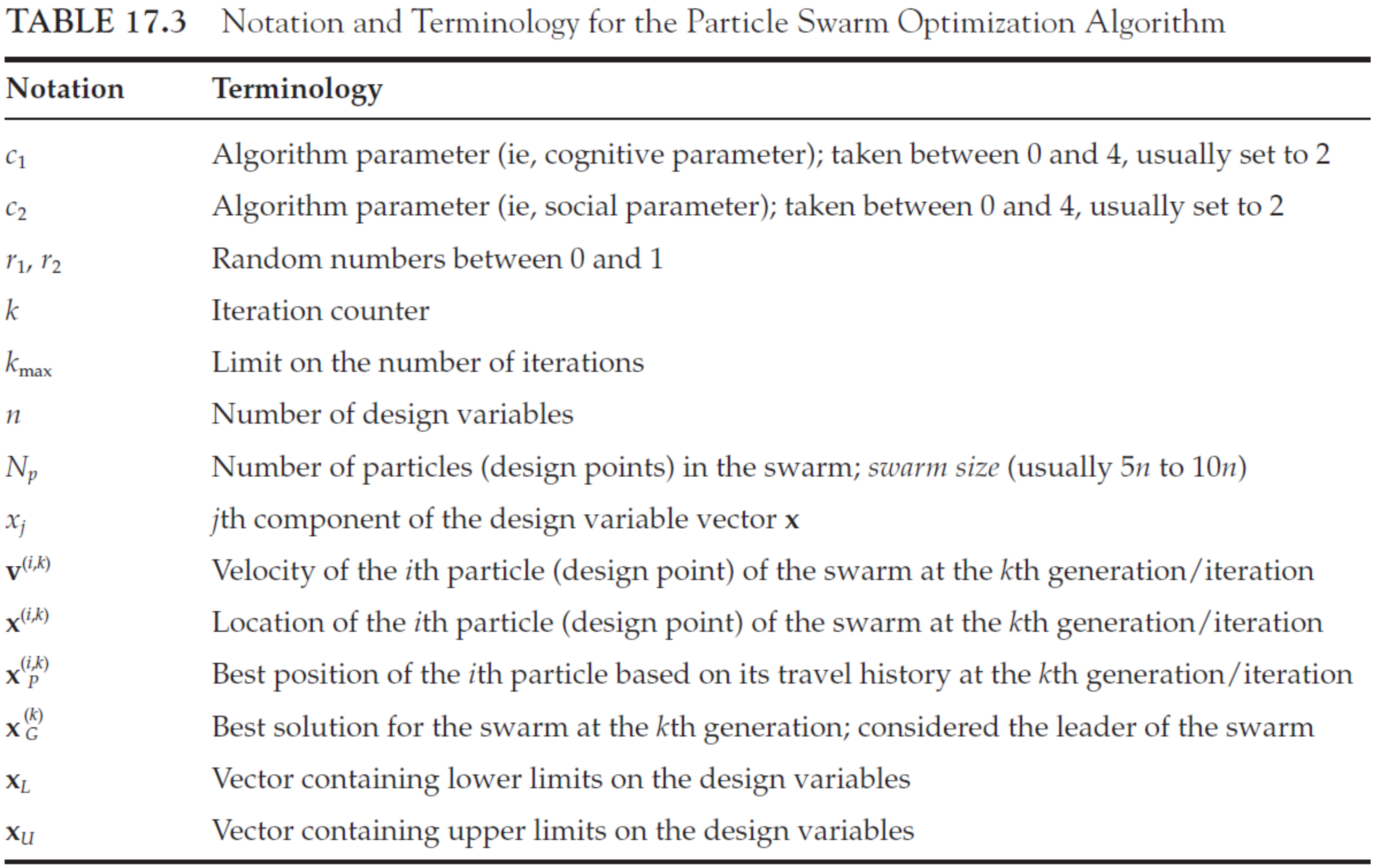


TABLE 17.3 Notation and Terminology for the Particle Swarm Optimization Algorithm

Notation	Terminology
c_1	Algorithm parameter (ie, cognitive parameter); taken between 0 and 4, usually set to 2
c_2	Algorithm parameter (ie, social parameter); taken between 0 and 4, usually set to 2
r_1, r_2	Random numbers between 0 and 1
k	Iteration counter
$k_{\max}$	Limit on the number of iterations
n	Number of design variables
N_p	Number of particles (design points) in the swarm; *swarm size* (usually 5n to 10n)
x_j	jth component of the design variable vector $\mathbf{x}$
$\mathbf{v}^{(i,k)}$	Velocity of the ith particle (design point) of the swarm at the kth generation/iteration
$\mathbf{x}^{(i,k)}$	Location of the ith particle (design point) of the swarm at the kth generation/iteration
$\mathbf{x}_P^{(i,k)}$	Best position of the ith particle based on its travel history at the kth generation/iteration
$\mathbf{x}_G^{(k)}$	Best solution for the swarm at the kth generation; considered the leader of the swarm
$\mathbf{x}_L$	Vector containing lower limits on the design variables
$\mathbf{x}_U$	Vector containing upper limits on the design variables

Simulated Annealing (SA)

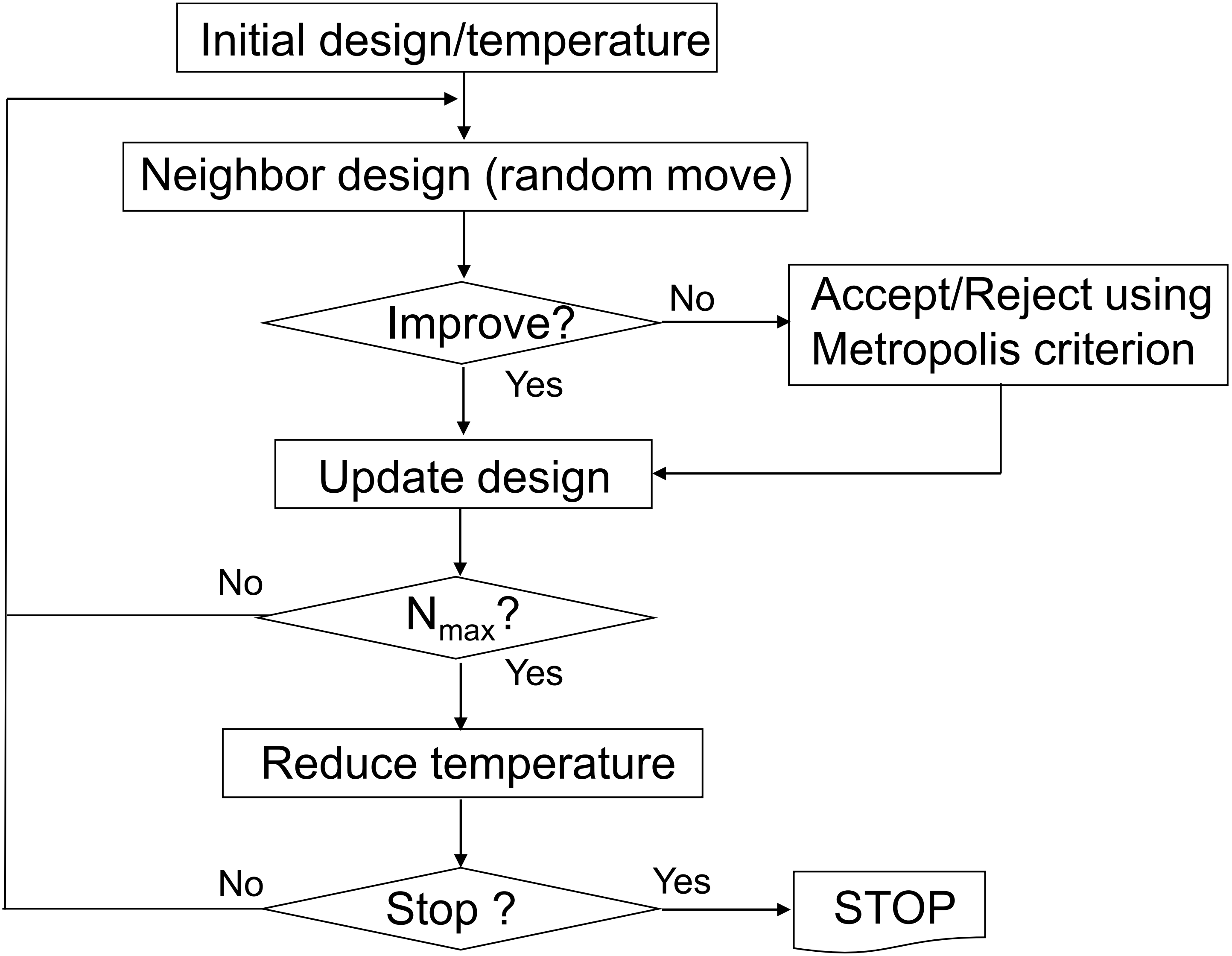
- S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, “Optimization by simulated annealing”, Science, 220, pp.671-680, 1983
- 금속의 풀림: 고체를 녹을 때까지 가열한 후 그것을 완전한 격자상태의 결정체가 될 때까지 식히는 물리적 과정. 이런 과정 중에 그 고체의 자유에너지는 최소화
- 개선되지 않는 이웃해로의 이동을 확률적으로 허용
  - 지역최적해에 머무는 것을 방지, 이웃해 탐색방법의 하나
- 물리적 어닐링과 시뮬레이티드 어닐링의 관계

어닐링	시뮬레이티드 어닐링
물질	최적화문제
물리적 상태	가능해
에너지	비용함수
기저상태	최적화
냉각	국부탐색법

Metropolis Algorithm

- At each iteration, an “atom” is randomly displaced a small amount (random move).
- The energy is calculated for each atom and the difference with the energy at its original location is calculated.
- Boltzmann probability factor $p(\Delta E) = e^{-\frac{\Delta E}{kT}}$ is calculated.
  - T: temperature of the body, k: Boltsmann’s constant
  - ∆E: energy difference between the two atom states
- If $\Delta E \leq 0$, then the new location is accepted.
- Otherwise, a random number is generated between 0 and 1.
  - If the random number < *p*, the higher energy state is accepted.
  - Otherwise, the old atom location is retained and the algorithm generates a new location.

SA: Algorithms



Initial design/temperature

Neighbor design (random move)

Improve?

No

Accept/Reject using Metropolis criterion

Yes

Update design

No

N_{max}?

Yes

Reduce temperature

No

Stop ?

Yes

STOP

SA: Characteristics

- The quality of the final solution is not affected by the initial guess, except that the computational effort may increase with wrong starting designs.
- Because of the discrete nature of the function and constraint evaluation, 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.
- For problems involving behavior constraints, an equivalent unconstrained function is to be formulated.

Random Search

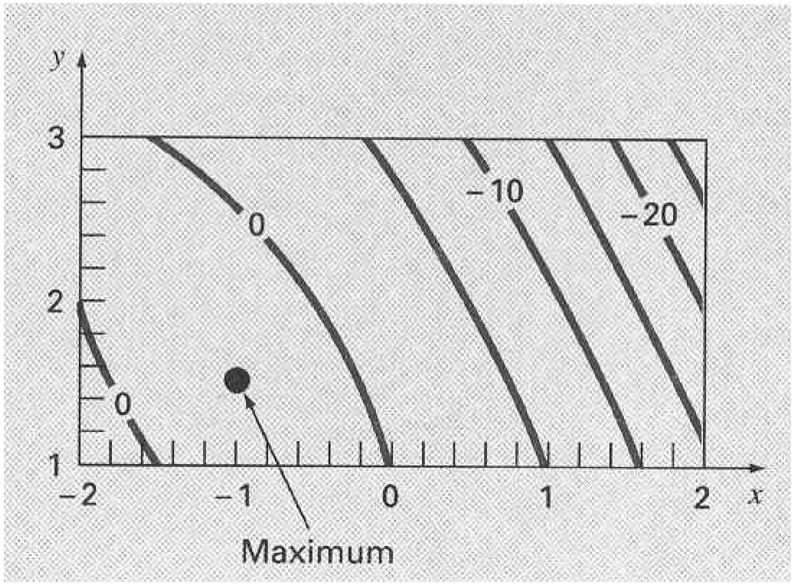
- Brute force approach
  - Evaluate the function repeatedly at randomly selected values of the independent variables
  - If a sufficient number of samples are conducted, the optimum will eventually be located
  - Work even for discontinuous and nondifferentiable functions ☺
  - Always find the global optimum rather than a local optimum ☺
  - Not efficient because it takes no account of the behavior of the underlying function ☹

Example

$$\text{Maximize } f(x,y) = y - x - 2x^2 - 2xy - y^2$$

$$\text{subject to } -2 \le x \le 2,\ 1 \le y \le 3$$

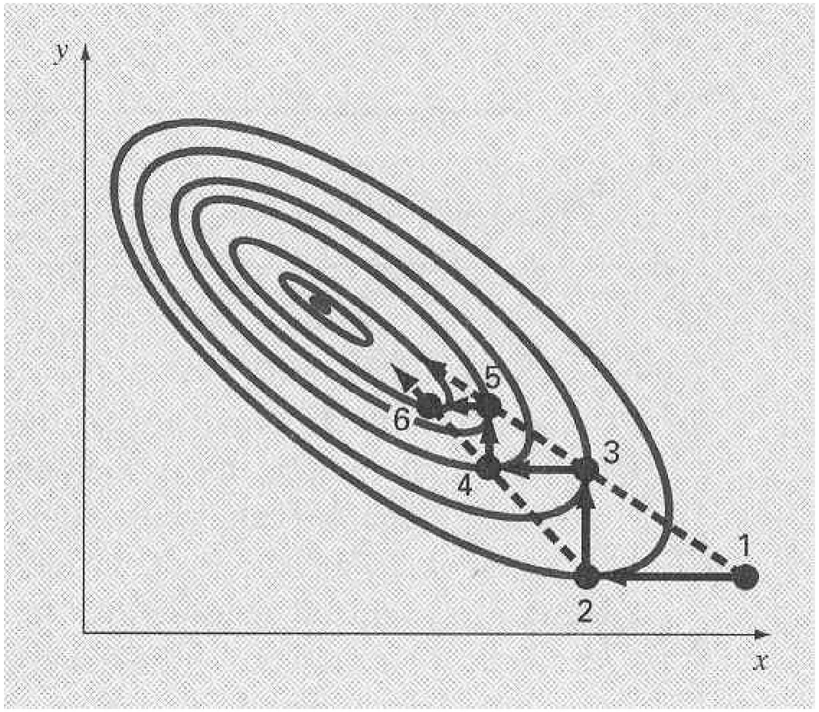
$$f_{\max} = 1.25 @ (-1,1.5)$$



n	x	y	f(x,y)
1000	−1.0206	1.4511	1.2448
5000	−1.0282	1.5263	1.2492
10000	−0.9964	1.5069	1.2499
50000	−0.9996	1.5103	1.2499
100000	−0.9958	1.4940	1.2500
500000	−0.9990	1.4974	1.2500
1000000	−0.9990	1.4974	1.2500
5000000	−0.9997	1.4999	1.2500

Univariate Search

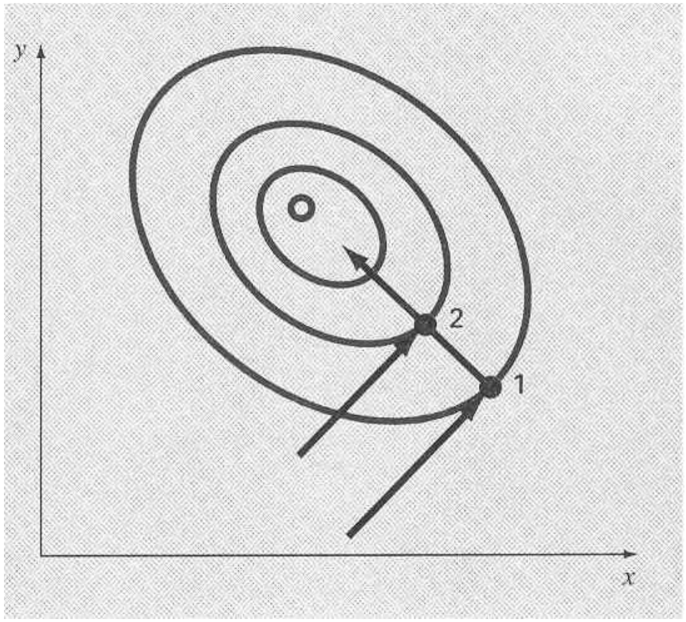
- Change one variable at a time to improve the approximation while the other variables are held constant
- Sequence of one-dimensional search
- Less efficient
- Pattern directions: 1-3, 3-5 or 2-4, 4-6

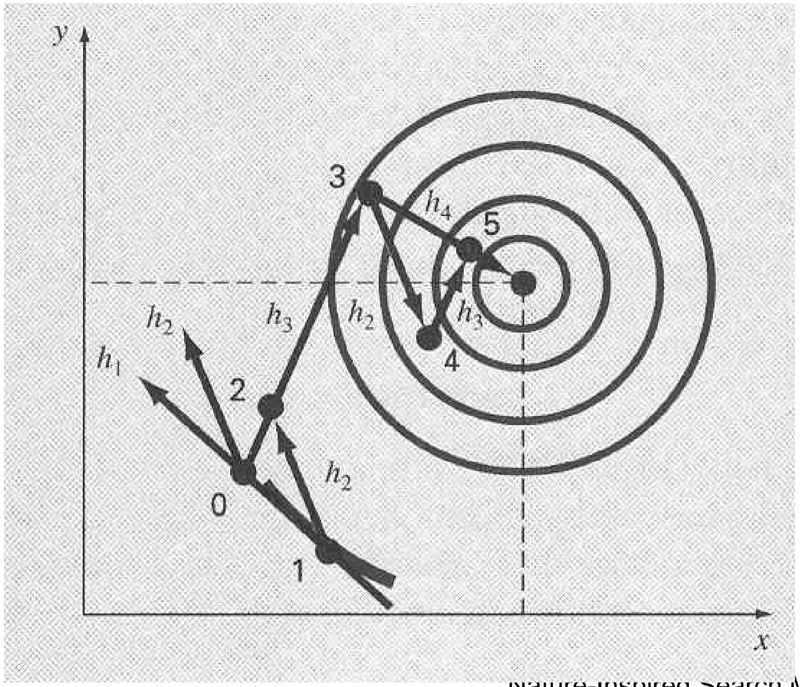


Powell’s method

- Conjugate directions
  - Line formed by point 1 and 2 obtained by one dimensional searches in the same direction but from different starting points
  - Nonlinear function ? → approximated by a quadratic function

point 0→(h1, point 1)→(h2, point 2) →(h3, point 3)





Conjugate Directions

minimize $q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{c}^T\mathbf{x}$

$\rightarrow \mathbf{g} = \nabla q = \mathbf{A}\mathbf{x} + \mathbf{c}$

directions $\mathbf{d}_i$ and $\mathbf{d}_j$ are conjugate with respect to $\mathbf{A}$

$\mathbf{d}_i^T \mathbf{A} \mathbf{d}_j = 0 \;\; if \;\; i \neq j$

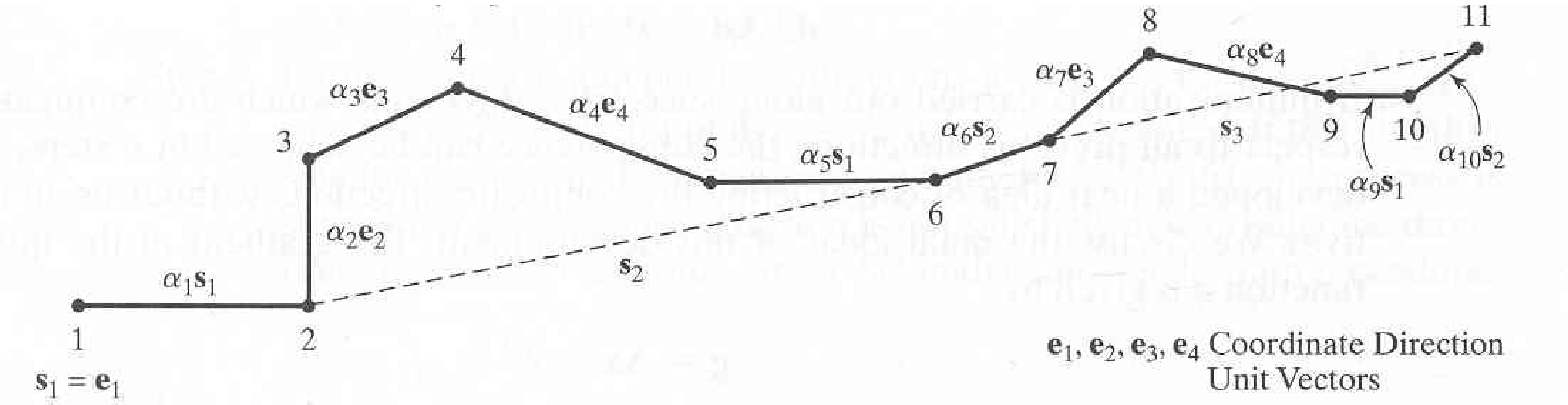
(parallel subspace property)

if two points of minima are obtained along parallel directions, then the direction given by the line joining the minima is $\mathbf{A}$ conjugate with respect to the parallel direction

$$\boldsymbol{s}_1^T \boldsymbol{A} \boldsymbol{s}_2 = 0 \leftarrow \boldsymbol{s}_2 = \left(\boldsymbol{x}^6 - \boldsymbol{x}^2\right)$$

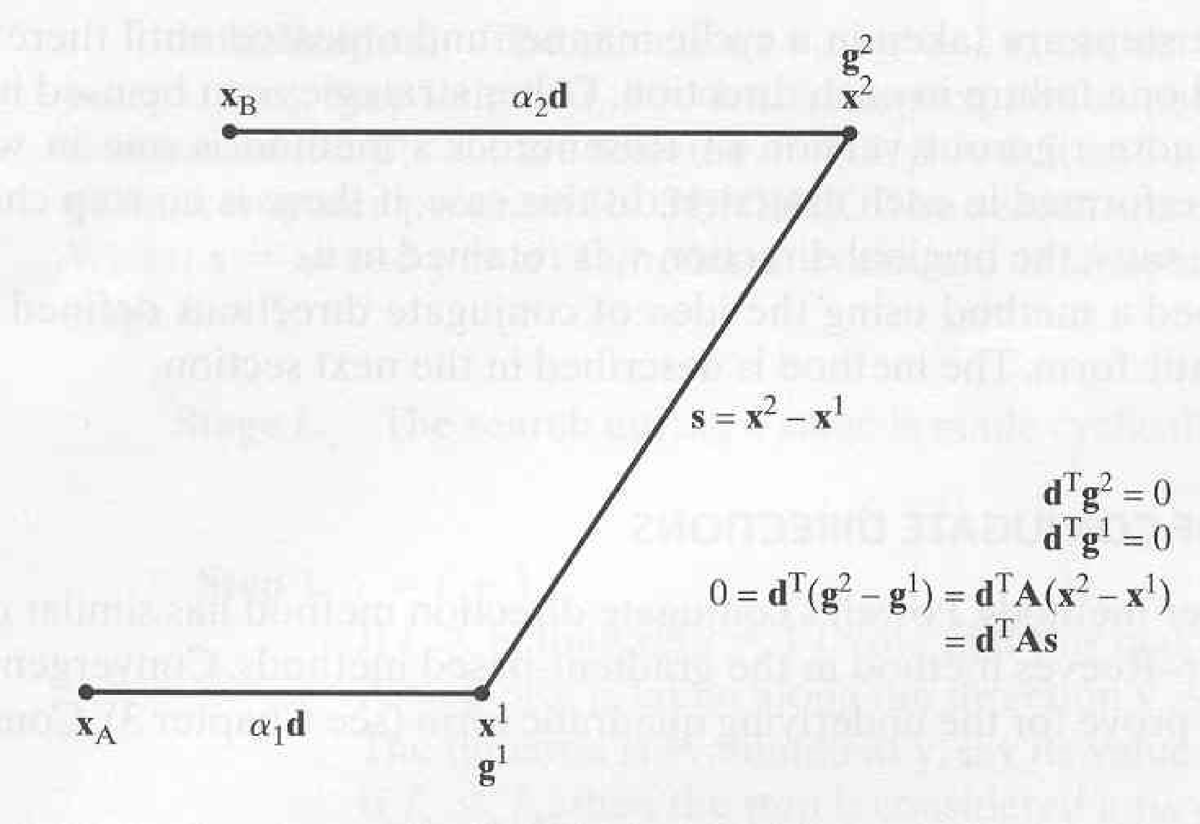
$\boldsymbol{x}^2, \boldsymbol{x}^6$: minima from points 1 and 5 along $\boldsymbol{s}_1$

$$\begin{aligned} 0 &= \boldsymbol{s}_1^T \boldsymbol{A}\left(\boldsymbol{x}^{10} - \boldsymbol{x}^6\right) \\ &= \boldsymbol{s}_1^T \boldsymbol{A}\left(\alpha_6 \boldsymbol{s}_2 + \boldsymbol{s}_3 - \alpha_{10} \boldsymbol{s}_2\right) \\ &= \boldsymbol{s}_1^T \boldsymbol{A} \boldsymbol{s}_3 \end{aligned}$$



Powell’s Method of Conjugate Directions

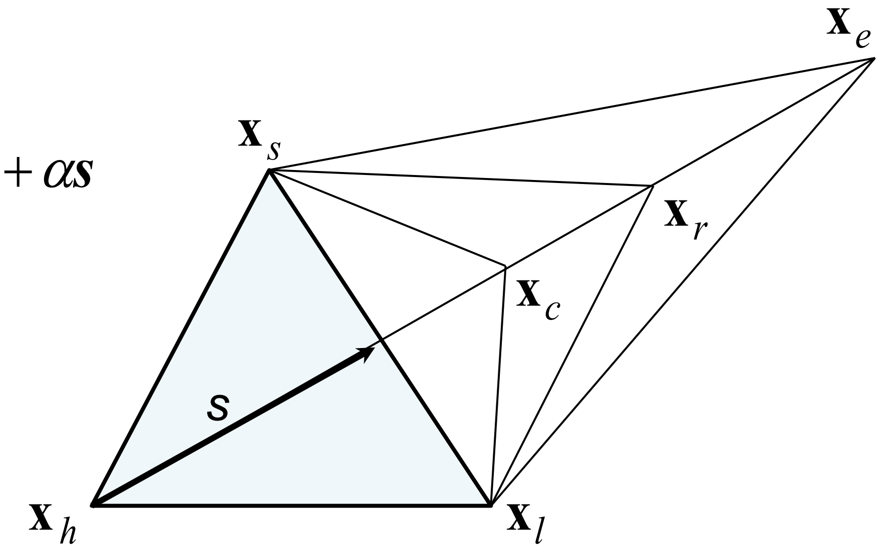
- Similar characteristics as Fletcher-Reeves method
- Convergence in *n* iterations for the quadratic form
- Successive directions conjugate w.r.t. all previous directions
- Parallel subspace property



Simplex Method

- Nelder-Mead (1965)
- *Simplex*: geometric shape formed by (n+1) points that do not lie in the same plane in the n-dimensional space
    - Search direction

$$\left.\begin{aligned} & f(\boldsymbol{x}_h) > f(\boldsymbol{x}_s) > f(\boldsymbol{x}_l) \\ & \boldsymbol{s} = \frac{\boldsymbol{x}_l + \boldsymbol{x}_s - 2\boldsymbol{x}_h}{2} \end{aligned}\right\} \rightarrow \boldsymbol{x}_r = \boldsymbol{x}_h + \alpha \boldsymbol{s}$$



    - Reflection: $\boldsymbol{x}_r$
    - Expansion: $\boldsymbol{x}_e$
    - Contraction: $\boldsymbol{x}_c$