Search-based (Metaheuristic) Optimization

Optimization Techniques (ENGG*6140)

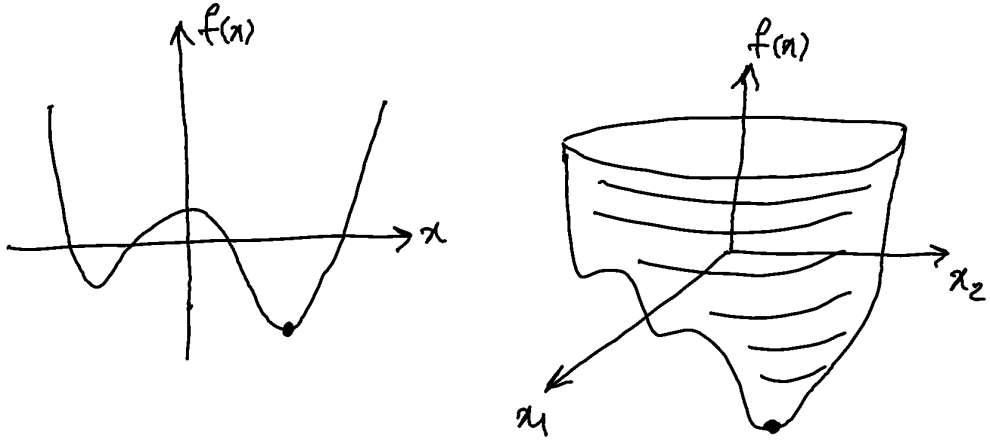
School of Engineering,
University of Guelph, ON, Canada

Course Instructor: Benyamin Ghojogh
Winter 2023

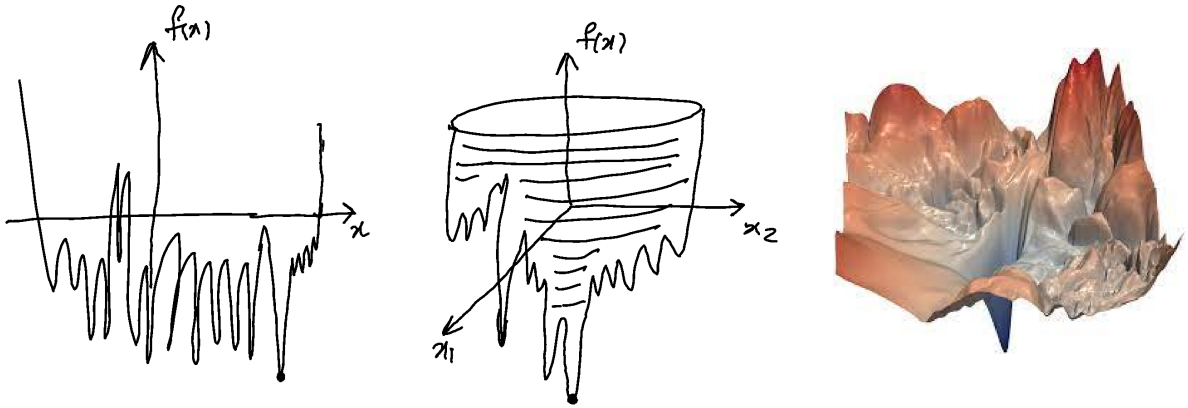
Introduction

Optimization versus search

- If the objective problem is **simple enough**, we can solve it using **classic optimization** methods. We will learn important classic methods.

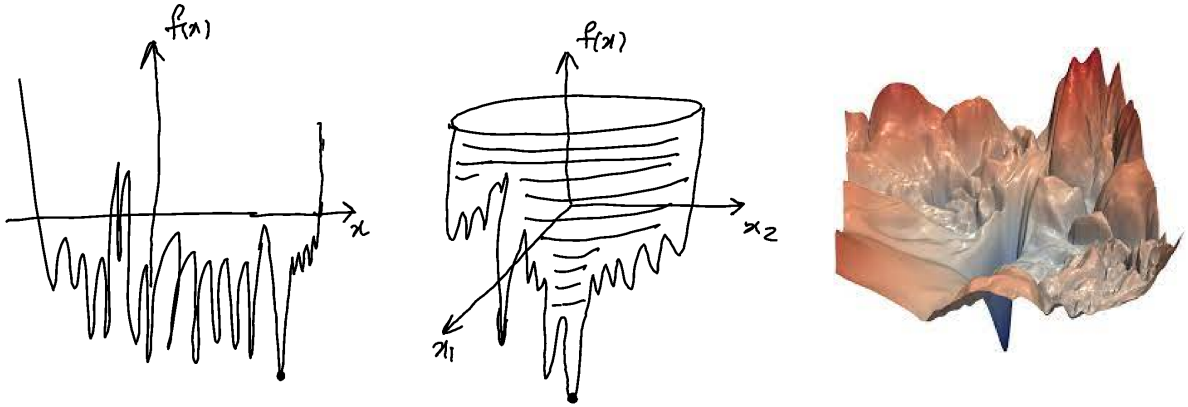


- If the objective function is **complicated** or if we have **too many constraints**, we can use **search** for finding a good solution (credit of third image: [1]).



When to use search-based (metaheuristic) optimization

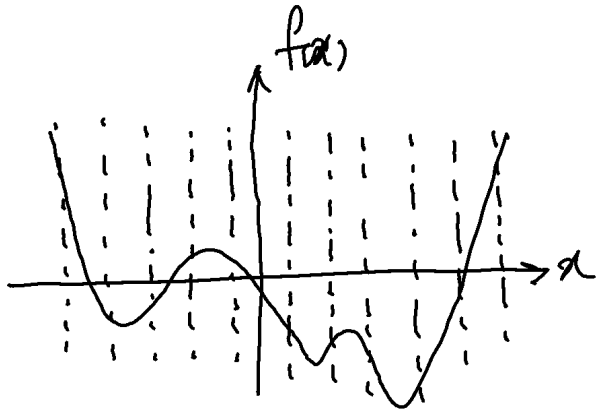
- When we have a **complicated** (highly con-convex) optimization landscape:



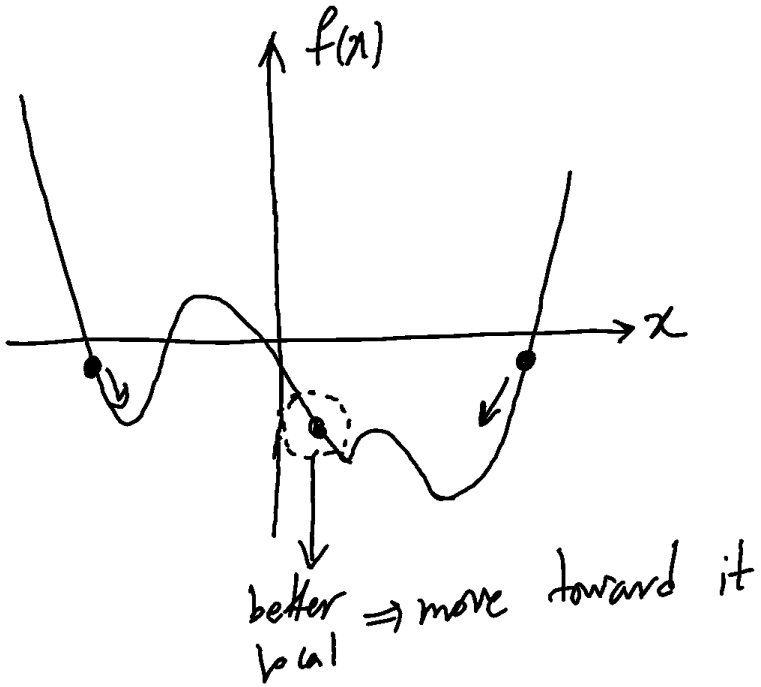
- or when the **gradient** of function is **hard to compute**,
- or when the function is not known but it works as a **black-box**, i.e., it outputs a value for each input fed to it.
- In these cases, we need:
  - either **non-convex optimization**,
  - or **search-based** optimization (**metaheuristic** optimization).

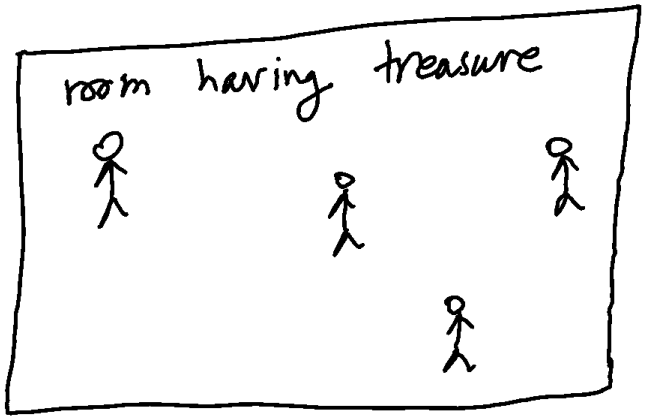
Search for optimization

- We can do **grid search** or **brute-force search**.



- Or we can **search wisely** by **metaheuristic optimization**. We will learn several important metaheuristic optimization methods.





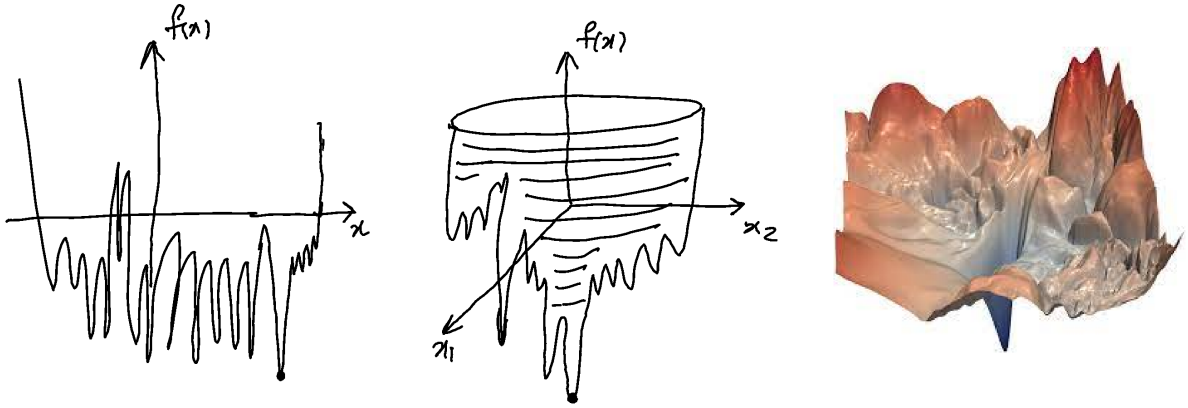
Heuristic and Metaheuristic Methods

- When the problem is complicated, a **heuristic** method approximates its solution.
- It gives a **good enough guess** of the solution to the problem, but that you may not really know **how good** it is.
- **Heuristics** are often **problem-dependent**, i.e., you define a heuristic for a given problem.
- **Metaheuristics** are **problem-independent** techniques that can be applied to a broad range of problems.
- **Metaheuristic optimization** methods can solve various complicated optimization problems using wise search.
- **Metaheuristic** methods are considered as a family of methods in **soft computing**.

Exploration vs. Exploitation

Some things need to be defined:

- **Optimization landscape**: the optimization cost function
- **Local best** vs. **global best** in the landscape

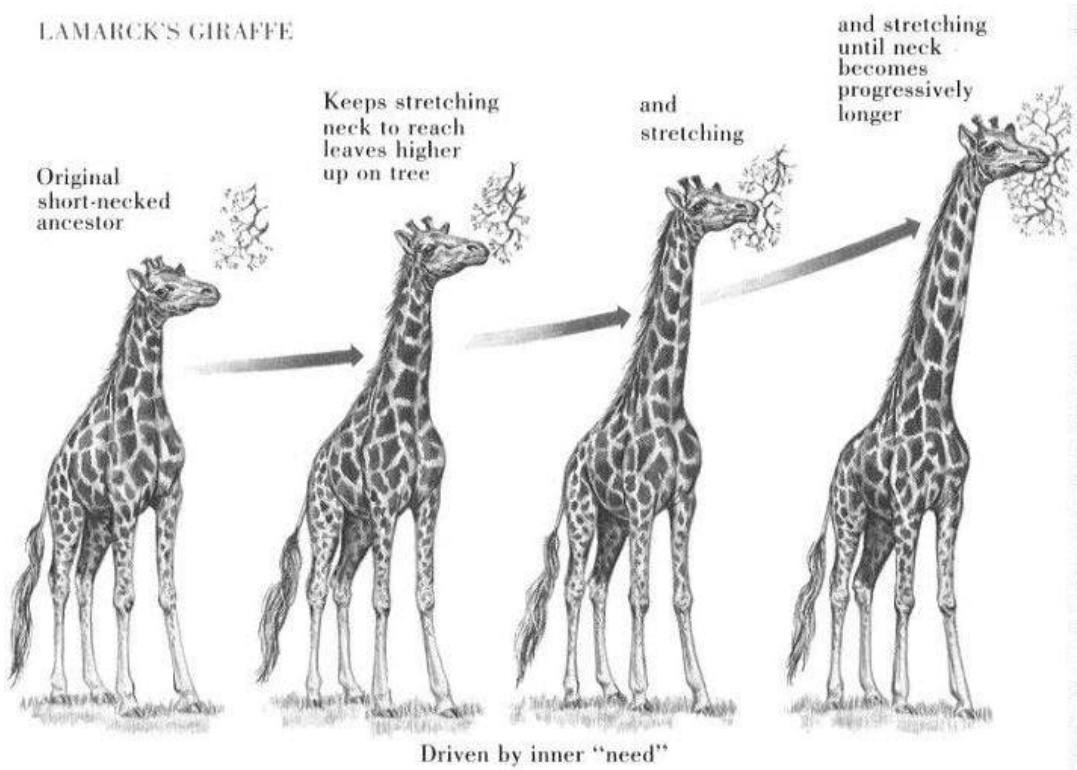


- **Exploitation**: local search around the solution because the global optimum might be close to the current solution.
- **Exploration**: search far away from the solution (explore the landscape) because the global optimum might be far away from the current solution. It helps not to get **stuck in local optimum**.

Genetic Algorithm (GA)

Genetic Algorithm: Natural Selection and Darwinism

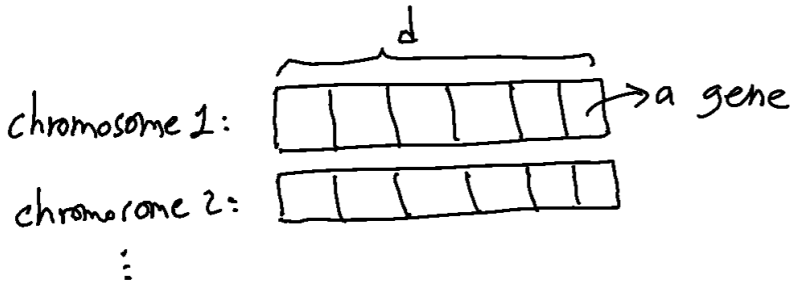
- According to **natural selection**, every generation tries to adapt better to the environment compared to the previous generation.
- **Genetic Algorithm (GA)**, almost proposed by John H. Holland in 1975 [2], is inspired by natural selection and Darwinism. It is one of the **evolutionary algorithms**.



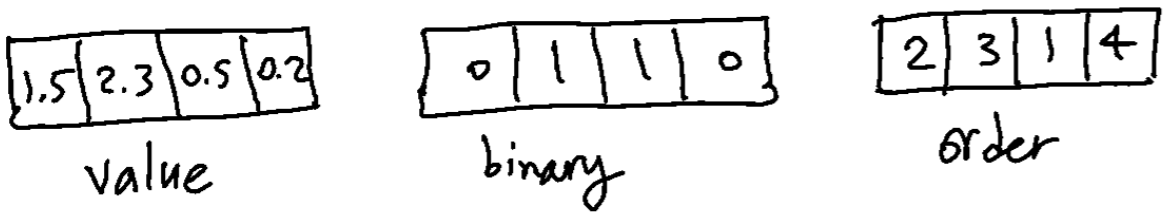
Credit of image: `https://www.zmescience.com/science/what-is-natural-selection/`

Genetic Algorithm: Chromosomes

- The candidate solutions are called the **chromosomes**.
- Every chromosome has several **genes**. The genes are the features/dimensions of the candidate solutions.

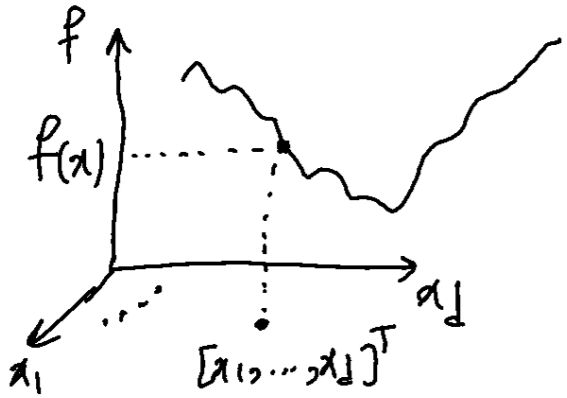


- Possible ways to create the chromosomes:
  - value encoding: the genes are float values of the features/dimensions of vector solutions.
  - binary encoding: every solution is a scalar and the scalar is converted to a binary string. Every gene is binary $\{0, 1\}$.
  - order encoding: the genes are integers if we want to find an optimal order of some integers.



Genetic Algorithm: Fitness function

- Fitness function is the evaluation of function at a candidate solution.



- Usually we want to maximize the fitness function and minimize the cost function. In other words, fitness function is minus cost function.
- We do it iteratively.
- **Fitness calculation**: at every iteration, we calculate the fitness of chromosomes to see how good they are.
- **Natural selection**: at every iteration, we **select** the best chromosomes to be the parents of the next generation. The other chromosomes which are not selected will die without having **offspring (children)**.

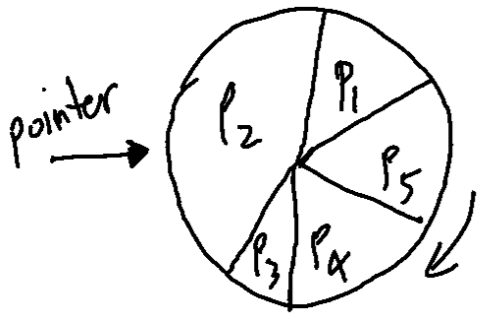
Genetic Algorithm: Natural selection

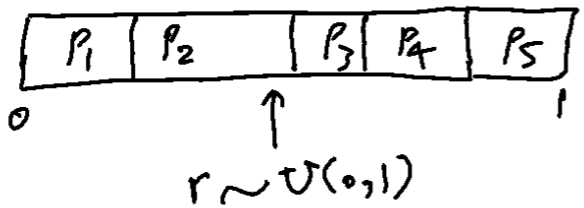
The ways for natural selection of chromosomes to be parents of the next generation:

- **Elitism selection**: selecting the top k chromosomes with the best fitness values. It has the risk of getting **stuck in local optimum**.
- **Roulette wheel selection**: for every chromosome, we calculate a probability of being good (compatible with the environment / having good fitness value):

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}, \quad \forall i \in \{1, \ldots, n\}, \tag{1}$$

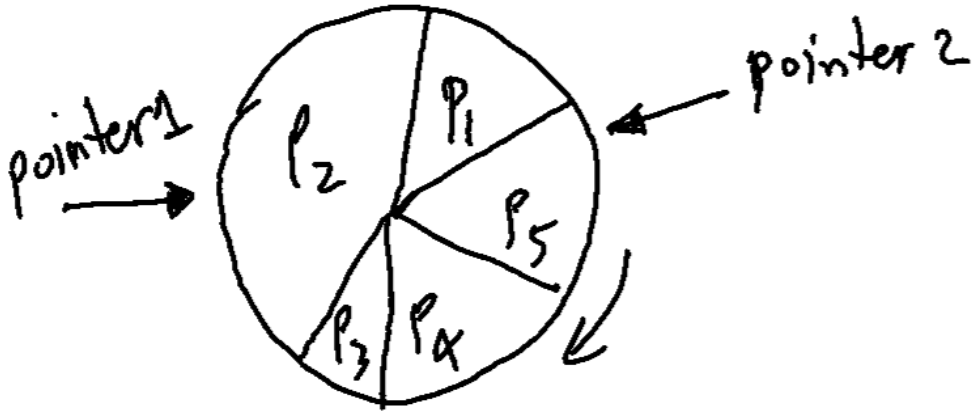
where f_i is the fitness value for the i-th chromosome and n is the population of chromosomes at the generation. Every chromosome gets a part of the roulette wheel based on its probability.





Genetic Algorithm: Natural selection

- **Stochastic Universal Sampling (SUS) selection**: it is similar to the roulette wheel but with more than one selection pointer. It gives **more chance** to chromosomes with less fitness value.

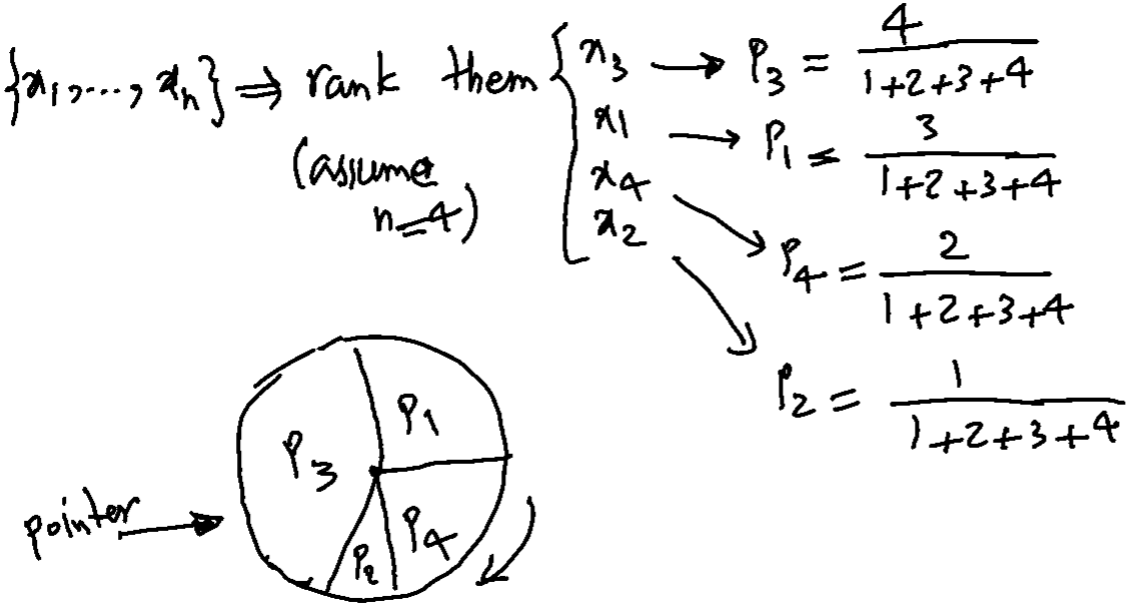


- **Tournament selection**: first we randomly select k_1 chromosomes and then, we select k_2 best chromosomes among them with best fitness values.

$$\{x_1, \ldots, x_{k_1}\} \Rightarrow \text{rank them} \begin{Bmatrix} x_3 \\ x_1 \\ x_5 \\ \vdots \end{Bmatrix} \rightarrow \text{get top } k_2$$

Genetic Algorithm: Natural selection

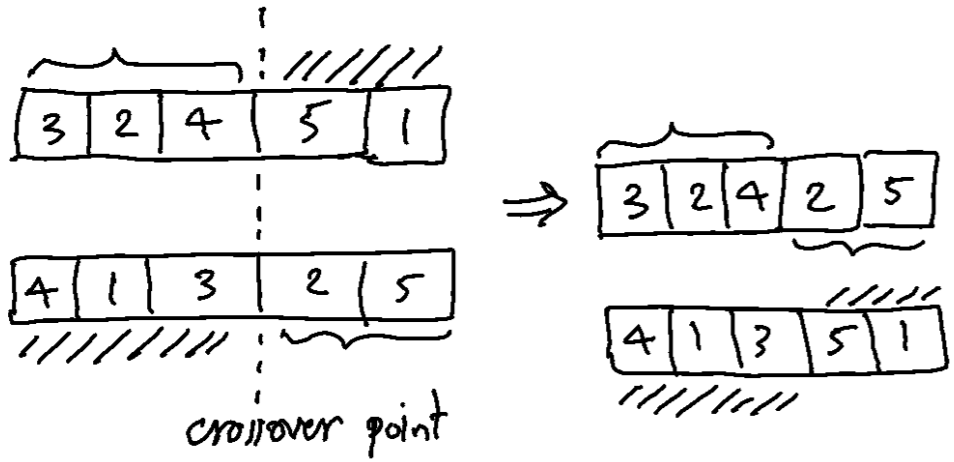
- **Rank selection**: Close to the termination, the probabilities in the roulette wheel with probabilities will become very close to each other. So, it will be better to rank the chromosomes by their fitness values in the late iterations. Then, we create the roulette wheel using the probabilities proportional to the ranks. This gives more chance to all chromosomes.



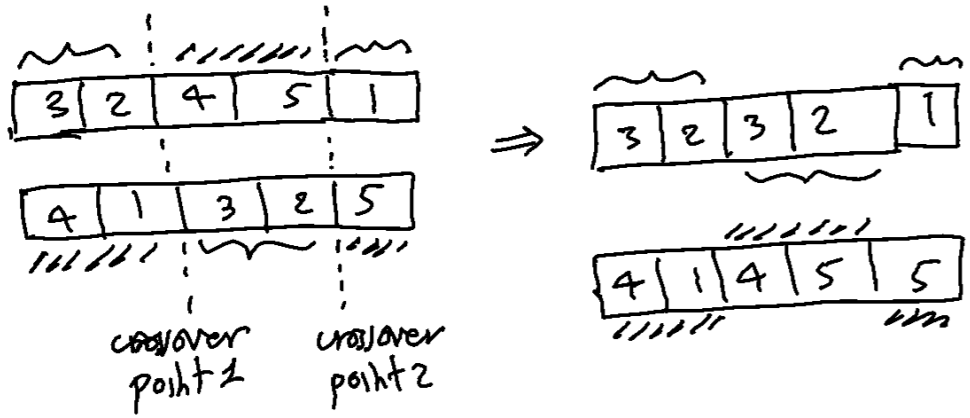
Genetic Algorithm: Cross-over

Cross-over: Mating of parent chromosomes to generate offsrpings (children). Useful for **exploration**.

- **One-point cross-over**:

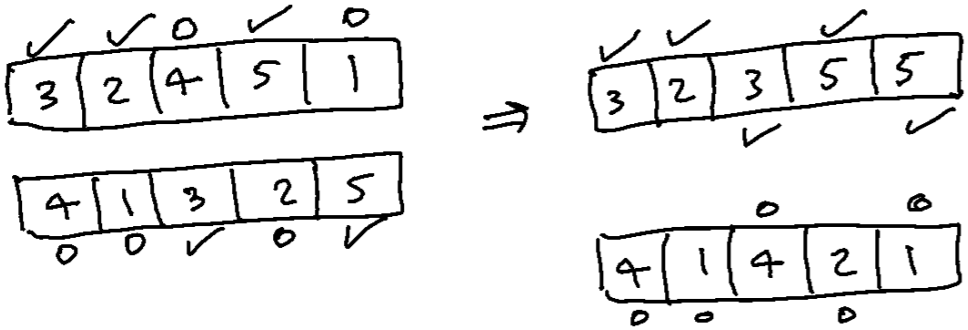


- **Multi-point cross-over**:



Genetic Algorithm: Cross-over

- **Uniform cross-over**:

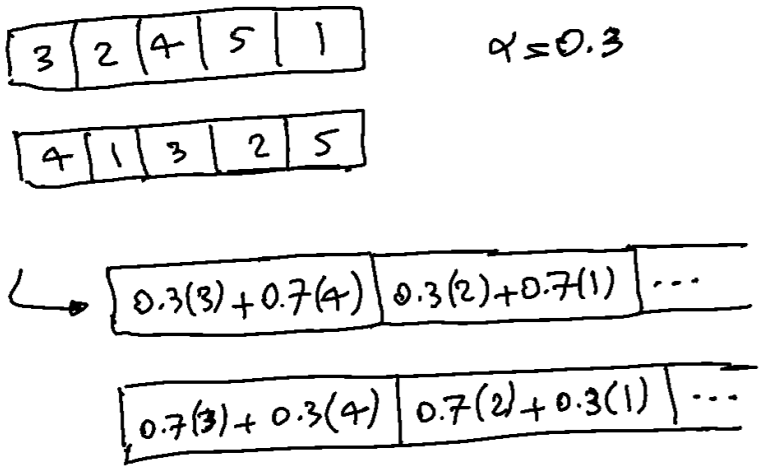


- **Whole arithmetic cross-over**:

$$\text{offspring1} = \alpha \boldsymbol{x}_1 + (1 - \alpha)\boldsymbol{x}_2, \tag{2}$$

$$\text{offspring2} = \alpha \boldsymbol{x}_2 + (1 - \alpha)\boldsymbol{x}_1, \tag{3}$$

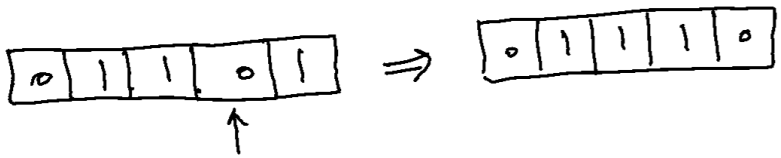
where $\alpha \in (0, 1)$.



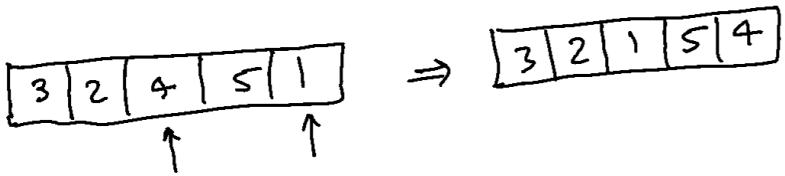
Genetic Algorithm: Mutation

Mutation: Random changes to the offsprings after cross-over. Useful for **exploitation**.

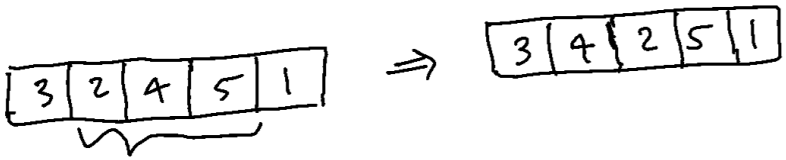
- **Bit-flip mutation**:



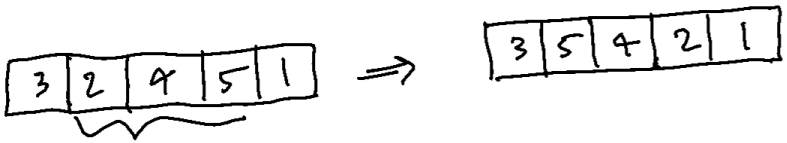
- **Swap mutation**:



- **Scramble mutation**:

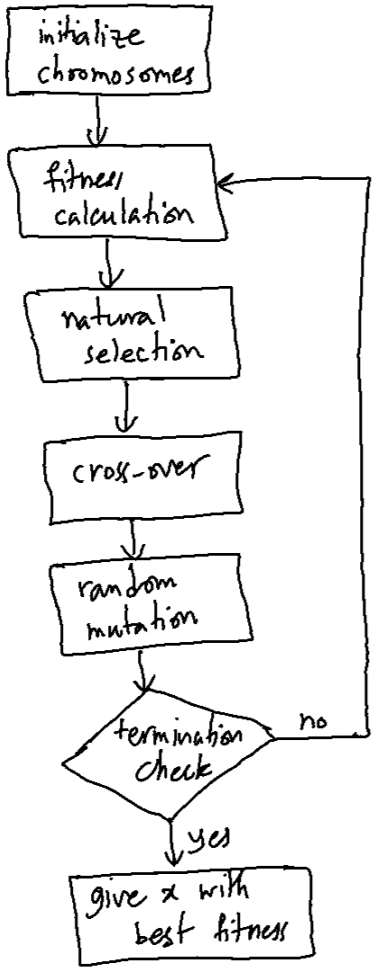


- **Inversion mutation**:



Genetic Algorithm: Termination and Flowchart

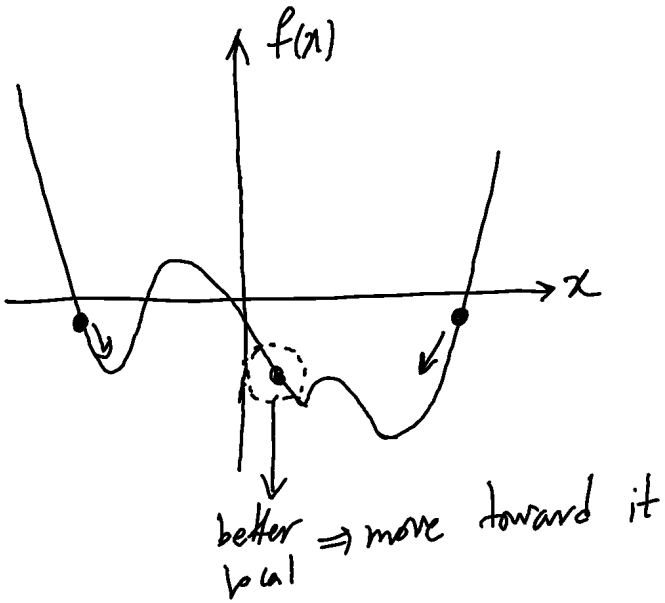
- Termination criterion:
  - reaching maximum number of iterations
  - reaching maximum elapsed (passed) time
  - reaching small enough cost (large enough fitness) compared to a known threshold

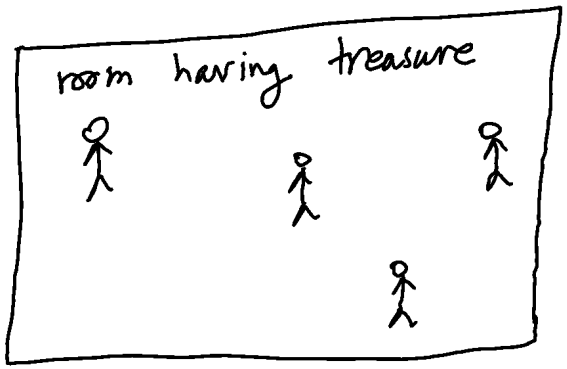


Particle Swarm Optimization (PSO)

Particle Swarm Optimization: the Idea

- **Particle Swarm Optimization (PSO)** was proposed in 1995 [3].
- The idea of PSO is like finding a treasure by a group of people.





- It is inspired a flock of birds or group of fish. Hence, it can be seen as one of the **bio-inspired** metaheuristic algorithms or **swarm optimization**.
- Many other bio-inspired or swarm metaheuristic algorithms exist such as:
  - **Ant colony**: 1996 [4, 5]
  - **Grey wolf** optimizer: 2014 [6]
  - **Whale** optimization algorithm: 2016 [7]
  - A scholar in this area: **Seyedali Mirjalili**, Torrens University Australia, Australia, `https://scholar.google.com/citations?user=TJHmrREAAAAJ&hl=en&oi=sra`

Particle Swarm Optimization: the Formula

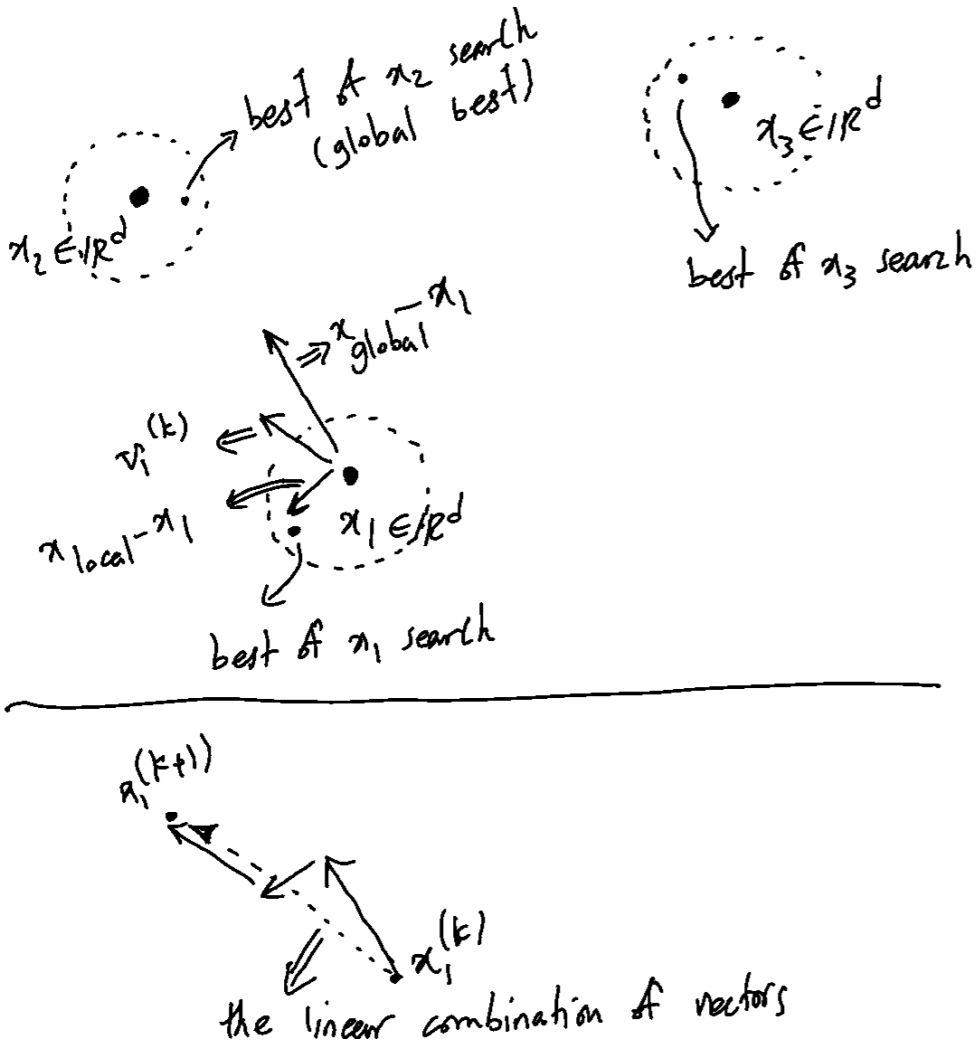
- The candidate solutions are the **particles** (vectors).
- Every particle **searches locally** in a local neighborhood.
- Three components for the velocity vector for updating the solution:
  - the **momentum (history)** of previous velocity (fro exploitation): $\alpha_1 \boldsymbol{v}_i^{(k)}$
  - update according to the **local best** in the iteration: $\alpha_2(\boldsymbol{x}_{\text{localBest}}^{(k)} - \boldsymbol{x}_i^{(k)})$
  - update according to the **global best** in the iteration: $\alpha_3(\boldsymbol{x}_{\text{globalBest}}^{(k)} - \boldsymbol{x}_i^{(k)})$
- The update of every particle:

$$\boldsymbol{x}_i^{(k+1)} := \alpha_1 \boldsymbol{v}_i^{(k)} + \alpha_2(\boldsymbol{x}_{\text{localBest}}^{(k)} - \boldsymbol{x}_i^{(k)}) + \alpha_3(\boldsymbol{x}_{\text{globalBest}}^{(k)} - \boldsymbol{x}_i^{(k)}), \tag{4}$$

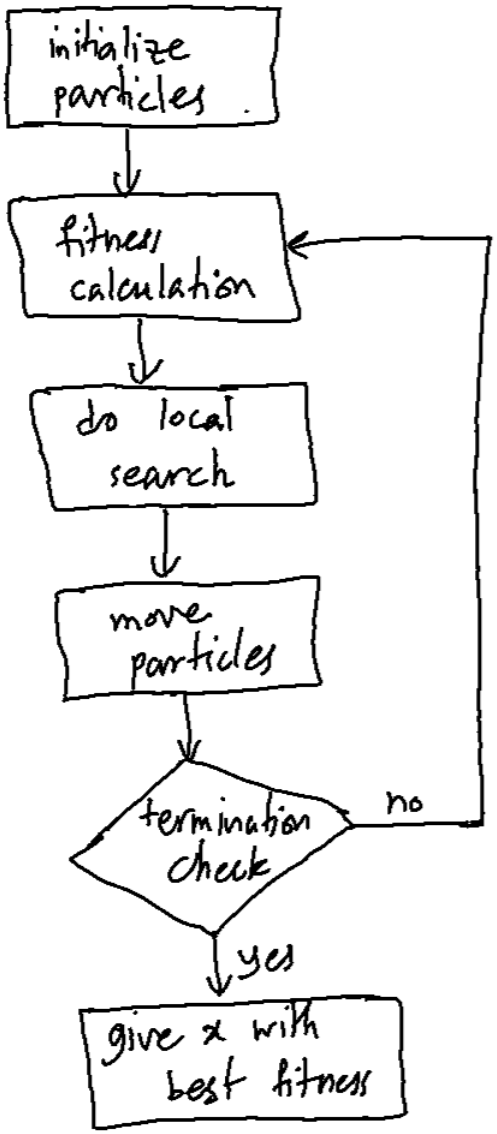
where α_1, α_2, and α_3 are weight (regularization) hyper-parameters.

Particle Swarm Optimization: Visualizing the Formula

$$\boldsymbol{x}_i^{(k+1)} := \alpha_1 \boldsymbol{v}_i^{(k)} + \alpha_2(\boldsymbol{x}_{\text{localBest}}^{(k)} - \boldsymbol{x}_i^{(k)}) + \alpha_3(\boldsymbol{x}_{\text{globalBest}}^{(k)} - \boldsymbol{x}_i^{(k)}),$$



Particle Swarm Optimization: Flowchart



Simulated Annealing

Simulated Annealing: Idea

- The **Boltzmann distribution** [8], also called the **Gibbs distribution** [9], can show the probability that a physical system can have a specific state. i.e., every of the particles has a specific state. The probability mass function of this distribution is [10]:

$$\mathbb{P}(x) = \frac{e^{-\frac{E(x)}{t}}}{Z}, \tag{5}$$

  where $E(x)$ is the energy of variable x, and t is the Kelvin temperature, and Z is the normalization constant so that the probabilities sum to one. We can write it as:

$$\mathbb{P}(\Delta E) = \frac{e^{-\frac{\Delta E}{t}}}{Z}, \tag{6}$$

  where ΔE is the difference of energy.

- **Simulated annealing** was proposed in 1983 [11] and is inspired by the **annealing schedule** in high-energy physics for forming the shape of materials.
- It starts with high temperature and cools down the temperature gradually.
  - **linear reduction rule**: $t = t - \alpha$
  - **geometric reduction rule**: $t = t \times \alpha$, where $\alpha \in (0, 1)$
  - **slow-decrease rule**: $t = \frac{t}{1+\beta t}$, where β is a hyper-parameter

Simulated Annealing: algorithm

- step 1: choose some random initial candidates and an initial temperature
- step 2: in every iteration, do a local search in a neighborhood of candidates and choose a neighbor point for every candidate.
  - for every candidate, if the fitness of neighbor solution is better than the candidate: accept it and replace the candidate with that.
  - otherwise, accept it with some Boltzmann probability:

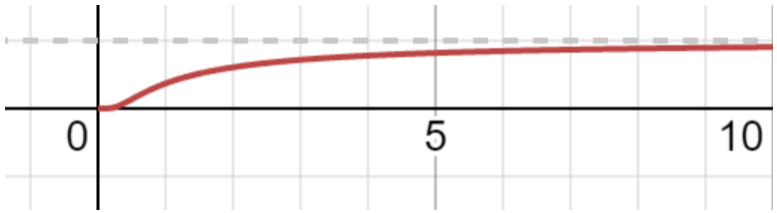
$$\mathbb{P}(\Delta E) = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{t}} & \text{if } \Delta E > 0 \end{cases} \tag{7}$$

  where ΔE is the change of cost (cost of neighbor minus cost of candidate) (or fitness of candidate minus fitness of neighbor).
- This gives a chance to even worse candidates for **exploration** (not to get stuck in local optimum).

Simulated Annealing: Analysis of temperature

$$\mathbb{P}(\Delta E) = \begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\frac{\Delta E}{t}} & \text{if } \Delta E > 0 \end{cases}$$

The $e^{-\frac{1}{t}}$ graph:



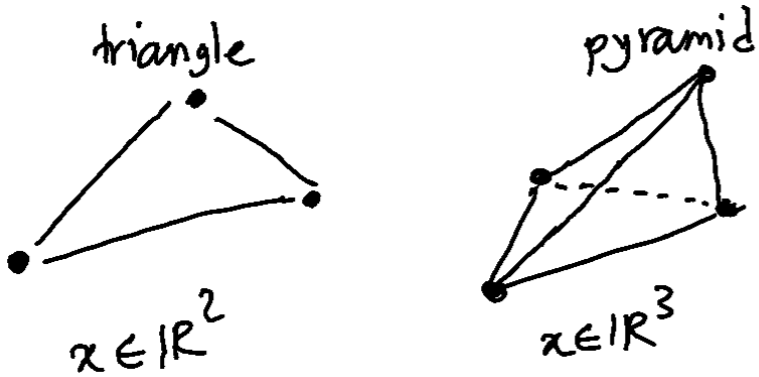
Analysis of temperature:

- In initial iterations, temperature t is high so $e^{-\frac{\Delta E}{t}}$ is large (closer to one) so we give more chance to worse candidates so we have **more exploration**.
- In the end iterations, temperature t is low so $e^{-\frac{\Delta E}{t}}$ is small (closer to zero) so we give less chance to worse candidates so we have **more exploitation**.
- It is like starting with large learning rate in gradient descent initially and then decrease the learning rate gradually.

Nelder-Mead Simplex Algorithm

Nelder-Mead Simplex Algorithm

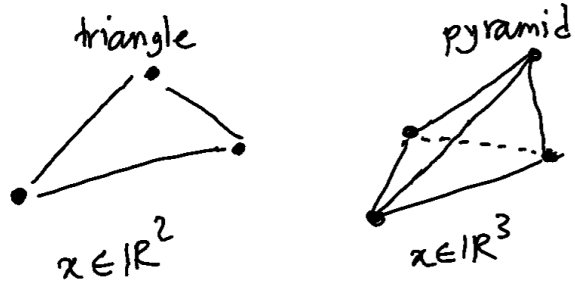
- The **Nelder-Mead simplex algorithm**, also called the **Nelder-Mead method**, was proposed by John A. Nelder and Roger Mead in 1965 [12].
- Used in "**fminsearch**" function of **MATLAB**:
  `https://www.mathworks.com/help/matlab/ref/fminsearch.html`
- Its idea:
  - If the dimensionality of optimization variable is d, choose $d+1$ random points in the feasible set to make a **simplex**.



  - Update this simplex iteratively until it converges to the optimal solution (it gradually moves toward the solution and shrinks to the solution.)

Nelder-Mead Method: initial simplex

- If the dimensionality of optimization variable is d, choose $d+1$ random points in the feasible set to make a **simplex**.



- This initial simplex is important. A too small simplex may get stuck in a local optimum (cannot do enough exploration).
- It is suggested in [12] to select the initial simplex as the following:
  - a random point for $\boldsymbol{x}_1 = [x_{11}, x_{12}, \ldots, x_{1d}]^\top$
  - each of $\{\boldsymbol{x}_2, \ldots, \boldsymbol{x}_{d+1}\}$ is a fixed step along each dimension in turn:

$$\begin{aligned}
\boldsymbol{x}_2 &= [x_{11} + \delta, x_{12}, \ldots, x_{1d}]^\top, \\
\boldsymbol{x}_3 &= [x_{11}, x_{12} + \delta, \ldots, x_{1d}]^\top, \\
&\vdots \\
\boldsymbol{x}_{d+1} &= [x_{11}, x_{12}, \ldots, x_{1d} + \delta]^\top,
\end{aligned}$$

where $\delta > 0$ is a not-too-small number.

Nelder-Mead Method: order

- We want to **minimize** the function $f(.)$. In the feasibility set, we make some simplex and change it in the feasibility set iteratively to converge to the solution.
- **order**: at the start of every iteration, order (sort) the corners of simplex:

$$f(\boldsymbol{x}_1) \leq f(\boldsymbol{x}_2) \leq \cdots \leq f(\boldsymbol{x}_{d+1}),$$

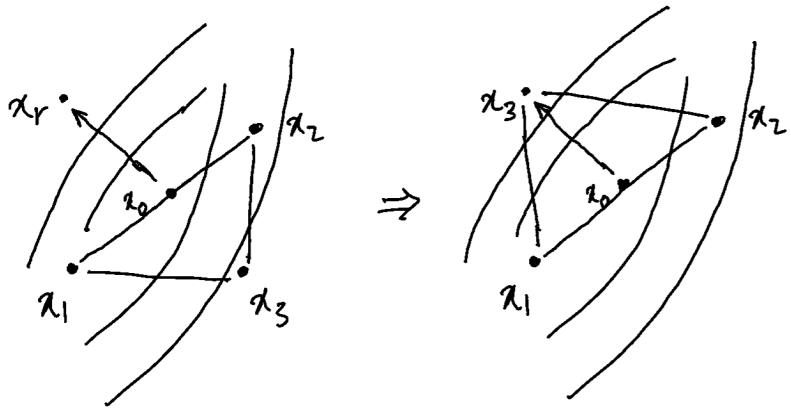
where we denote the best and worst corners by $\boldsymbol{x}_1$ and $\boldsymbol{x}_{d+1}$, respectively.

Nelder-Mead Method: order and reflection

- **centroid**: the centroid of all points except $\boldsymbol{x}_{d+1}$ as: $\boldsymbol{x}_o = \frac{1}{d}\sum_{i=1}^{d} \boldsymbol{x}_i$.
- **reflection**:
  - the reflected point:

$$\boldsymbol{x}_r = \boldsymbol{x}_o + \alpha(\boldsymbol{x}_o - \boldsymbol{x}_{d+1}).$$

  - $\alpha > 0$, usually $\alpha = 1$.
  - if $f(\boldsymbol{x}_1) \leq f(\boldsymbol{x}_r) < f(\boldsymbol{x}_d)$:
    - ★ replace the worst point with the reflected point, $\boldsymbol{x}_{d+1} := \boldsymbol{x}_r$.
    - ★ go to the next iteration and order the points again.



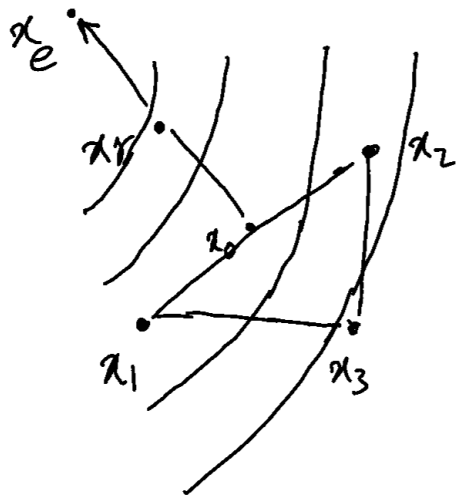
- it has some connection with **opposition learning** proposed in 2005 [13] and used in metaheuristic optimization in 2008 [14].

Nelder-Mead Method: expansion

- **expansion**: if $f(\boldsymbol{x}_r) \leq f(\boldsymbol{x}_1)$:
  - the expanded point:

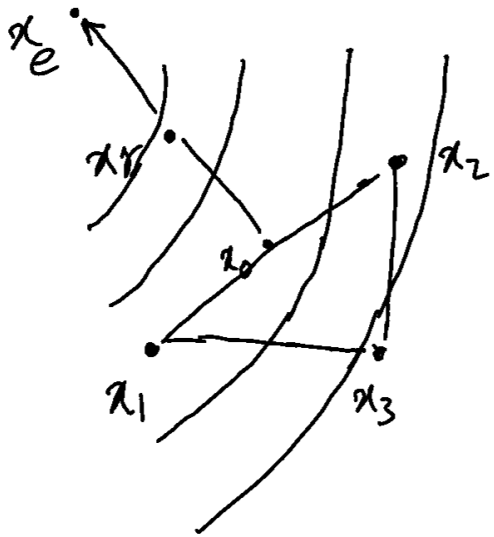
$$\boldsymbol{x}_e = \boldsymbol{x}_o + \gamma(\boldsymbol{x}_r - \boldsymbol{x}_o).$$

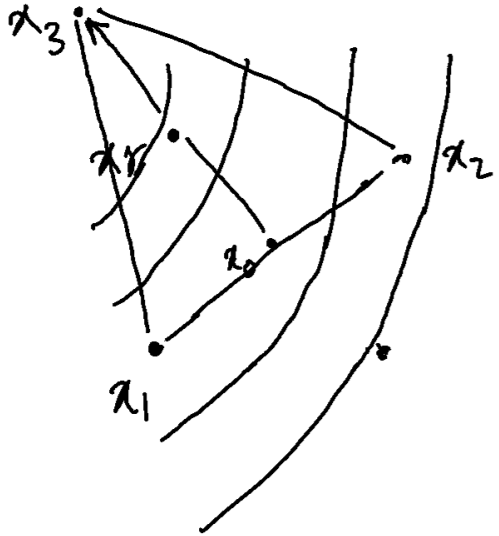
  - $\gamma > 1$, usually $\gamma = 2$.

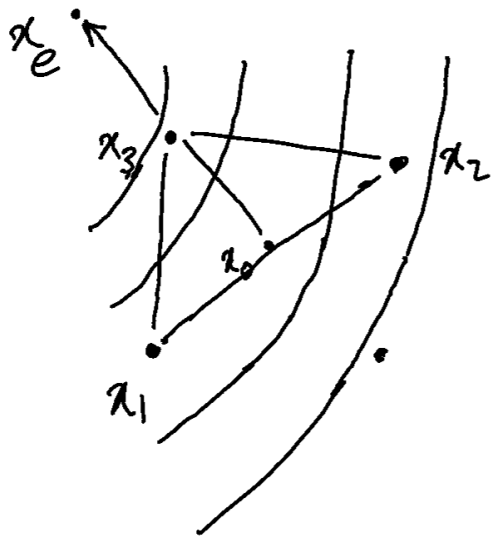


Nelder-Mead Method: expansion

- if the expanded point is better than the reflected point, $f(\boldsymbol{x}_e) < f(\boldsymbol{x}_r)$:
  - replace the worst point with the expanded point, $\boldsymbol{x}_{d+1} := \boldsymbol{x}_e$.
- else, $f(\boldsymbol{x}_e) \geq f(\boldsymbol{x}_r)$:
  - replace the worst point with the reflected point, $\boldsymbol{x}_{d+1} := \boldsymbol{x}_r$.
- go to the next iteration and order the points again.





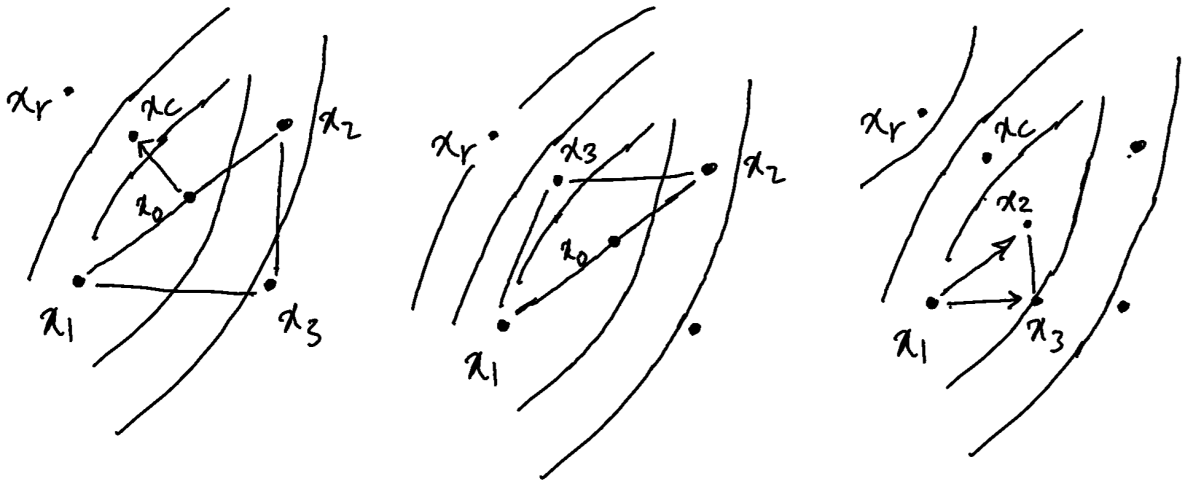


Nelder-Mead Method: contraction & shrinking

- **contraction**: if $f(\boldsymbol{x}_r) \geq f(\boldsymbol{x}_d)$:
  - if $f(\boldsymbol{x}_r) < f(\boldsymbol{x}_{d+1})$:
    - ★ the contracted point **outside**:

$$\boldsymbol{x}_c = \boldsymbol{x}_o + \rho(\boldsymbol{x}_r - \boldsymbol{x}_o).$$

    - ★ $0 < \rho \leq 0.5$, usually $\rho = 0.5$.
    - ★ if the contracted point is better than the reflected point, $f(\boldsymbol{x}_c) < f(\boldsymbol{x}_r)$:
      replace the worst point with the contracted point, $\boldsymbol{x}_{d+1} := \boldsymbol{x}_c$.
    - ★ else, $f(\boldsymbol{x}_c) \geq f(\boldsymbol{x}_r)$:
      shrinking: replace all points (except the best point $\boldsymbol{x}_1$) with the reflected point, $\boldsymbol{x}_i := \boldsymbol{x}_1 + \sigma(\boldsymbol{x}_i - \boldsymbol{x}_1)$, where $\sigma = 0.5$.
    - ★ go to the next iteration and order the points again.

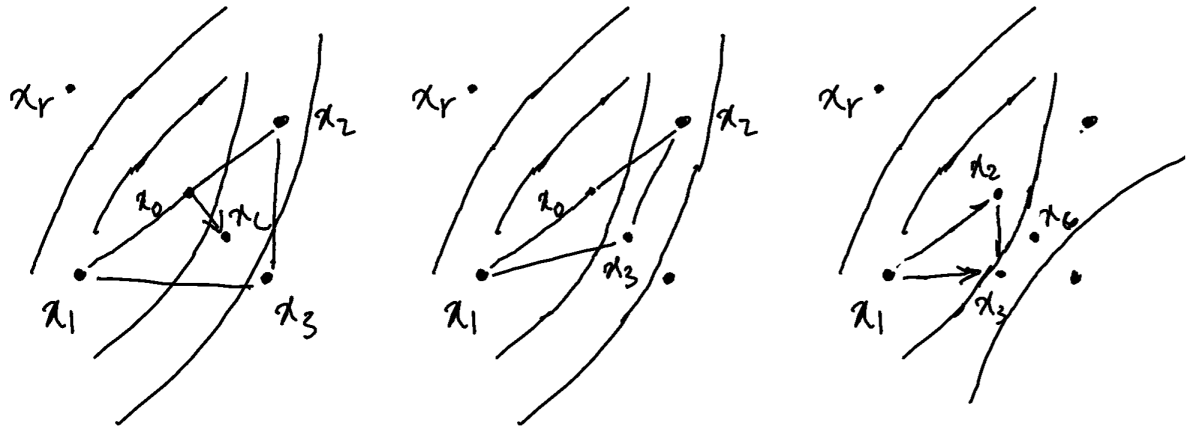


Nelder-Mead Method: contraction & shrinking

- **contraction**: if $f(\boldsymbol{x}_r) \geq f(\boldsymbol{x}_d)$:
  - if $f(\boldsymbol{x}_r) \geq f(\boldsymbol{x}_{d+1})$:
    - ★ the contracted point **inside**:

$$\boldsymbol{x}_c = \boldsymbol{x}_o + \rho(\boldsymbol{x}_{d+1} - \boldsymbol{x}_o).$$

    - ★ $0 < \rho \leq 0.5$, usually $\rho = 0.5$.
    - ★ if the contracted point is better than the worst point, $f(\boldsymbol{x}_c) < f(\boldsymbol{x}_{d+1})$:
      replace the worst point with the contracted point, $\boldsymbol{x}_{d+1} := \boldsymbol{x}_c$.
    - ★ else, $f(\boldsymbol{x}_c) \geq f(\boldsymbol{x}_{d+1})$:
      shrinking: replace all points (except the best point $\boldsymbol{x}_1$) with the reflected point, $\boldsymbol{x}_i := \boldsymbol{x}_1 + \sigma(\boldsymbol{x}_i - \boldsymbol{x}_1)$, where $\sigma = 0.5$.
    - ★ go to the next iteration and order the points again.



Nelder-Mead Method: summary

initialize simplex with $(d+1)$ points in feasibility set

loop until convergence:

- calculate centroid: x_0
- calculate reflected point: x_r
- if $f(x_1) \leq f(x_r) \leq f(x_d)$:
  - $x_{d+1} \leftarrow x_r$
  - continue
- elseif $f(x_r) \leq f(x_1)$:
  - $x_e = x_0 + \gamma(x_r - x_0)$
  - if $f(x_e) < f(x_r)$:
    - $x_{d+1} \leftarrow x_e$
  - else:
    - $x_{d+1} \leftarrow x_r$
  - continue

Nelder-Mead Method: summary

loop until convergence:

  elseif $f(x_r) \geq f(x_j)$:

    if $f(x_r) < f(x_{j+1})$:

      $x_c = x_0 + \rho(x_r - x_0)$

      if $f(x_c) < f(x_r)$:

        $x_{j+1} \leftarrow x_c$

      else:

        $x_i \leftarrow x_1 + \sigma(x_i - x_1)$ $\forall i$

      continue

    else:

      $x_c = x_0 + \rho(x_{j+1} - x_0)$

      if $f(x_c) < f(x_{j+1})$:

        $x_{j+1} \leftarrow x_c$

      else:

        $x_i \leftarrow x_1 + \sigma(x_i - x_1)$ $\forall i$

      continue

Acknowledgement

- Some slides of this slide deck are inspired by the teachings of Prof. **Saeed Sharifian** at Amirkabir University of Technology, Tehran, Iran (his course "Biological intelligence").
- Some slides of this slide deck are inspired by:
  - Genetic algorithm: `https://medium.com/@AnasBrital98/genetic-algorithm-explained-76dfbc5de85d`
  - Particle swarm optimization: `https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-particle-swarm-optimization-algorithm/`
  - Simulated annealing: `https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7`
  - Nelder-Mean method: `https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method`

References

[1] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," *Advances in neural information processing systems*, vol. 31, 2018.

[2] J. H. Holland, "Adaptation in natural and artificial systems, univ. of mich. press," *Ann Arbor*, vol. 7, pp. 390–401, 1975.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[4] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[5] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[6] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in engineering software*, vol. 69, pp. 46–61, 2014.

[7] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in engineering software*, vol. 95, pp. 51–67, 2016.

[8] L. Boltzmann, "Studien uber das gleichgewicht der lebenden kraft," *Wissenschafiliche Abhandlungen*, vol. 1, pp. 49–96, 1868.

References (cont.)

[9] J. W. Gibbs, *Elementary principles in statistical mechanics*. Courier Corporation, 1902.

[10] K. Huang, *Statistical Mechanics*. John Wiley & Sons, 1987.

[11] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[12] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[13] H. R. Tizhoosh, "Opposition-based learning: a new scheme for machine intelligence," in *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06)*, vol. 1, pp. 695–701, IEEE, 2005.

[14] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Opposition-based differential evolution," *IEEE Transactions on Evolutionary computation*, vol. 12, no. 1, pp. 64–79, 2008.