Multi-objective optimization and Symbolic Regression

Classical and meta-heuristic methods

Veronica Guidetti 14-22/02/2024, UNIMORE

Table of contents

Four lectures course: 4x2 hrs

Lecture 1:

- Introduction to MOO framework and taxonomy
- Classical scalarization methods

Lecture 2:

- Meta-heuristic methods
- Simulated annealing
- Swarm Particle
- Genetic algorithms

Lecture 3:

- Evaluation metrics
- Python packages

Lecture 4:

- Symbolic regression
- Multi-objective symbolic regression
- Case studies

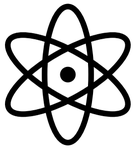
Meta-heuristic methods

Evolutionary algorithms

- Genetic algorithms

- Particle Swarm Optimization

- Simulated annealing



- Ant-Colony Optimization

Meta-heuristic methods

Benefits

- Objectives: differentiable and NOT
- Pareto Set in one run
- Parallel computing

Nomenclature

Method you use, dictionary you find

Genetic Algorithms

Mutations
Individual
Population
Generation

Particle Swarm

Flock
Cognition
Social intelligence
Turbulence

Simulated annealing

Cooling rate
State
Boltzmann temperature
Metropolis sampling

Ant Colony

Colony
Path
Evaporation
Pheromone

Solution ranking

Goldberg

Starting from a set of candidates S

- Non-dominated in S: rank 1
- Non-dominated in S/PF1: rank 2
- …

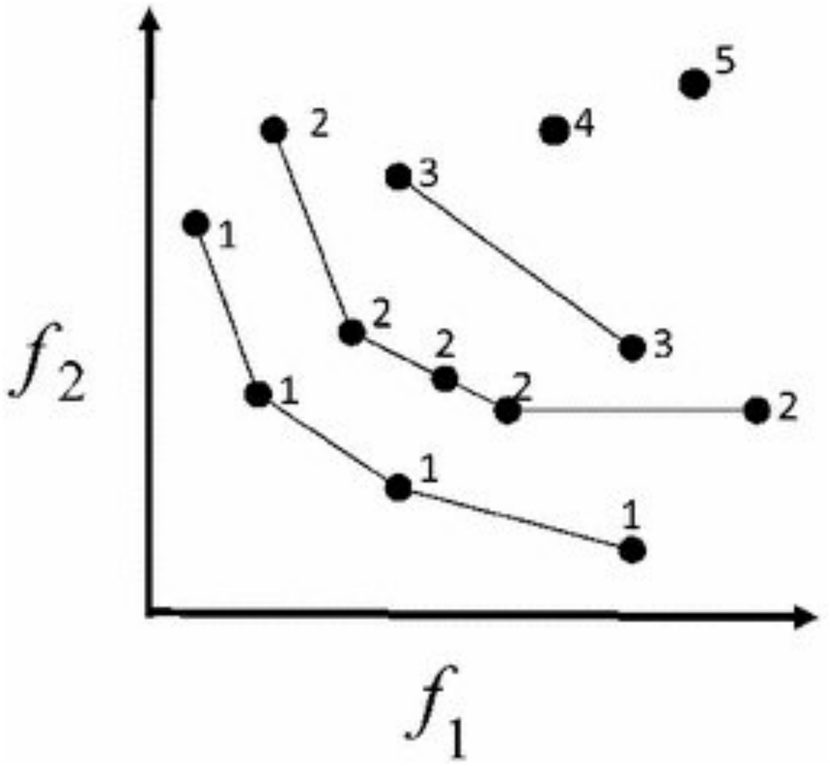
Fonseca and Fleming

Starting from a set of candidates S

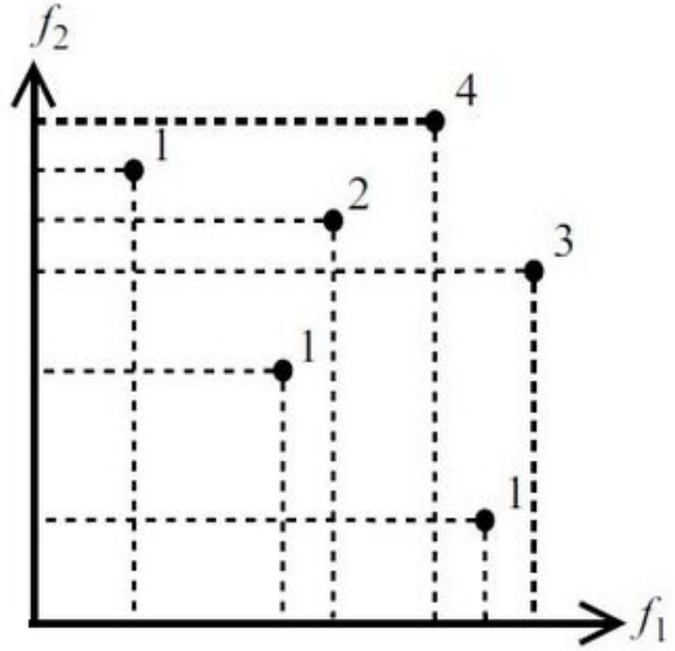
- Non-dominated in S: rank 1
- Rank = # dominating solutions +1

Solution ranking

Goldberg

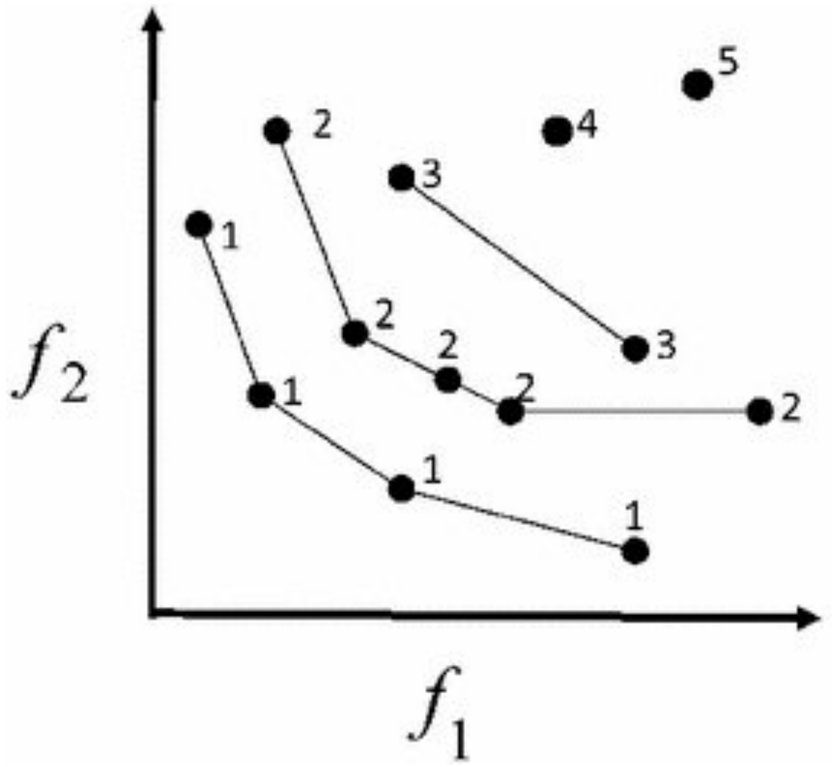


Fonseca and Fleming



Solution ranking

Goldberg



`fast-non-dominated-sort`(P) ***(from Deb et al. 2002)***

for each $p \in P$
  $S_p = \emptyset$ <- *individuals dominated by p*
  $n_p = 0$ <- number of individuals dominating p
  for each $q \in P$
    if $(p \prec q)$ then — If p dominates q
      $S_p = S_p \cup \{q\}$ — Add q to the set of solutions dominated by p
    else if $(q \prec p)$ then
      $n_p = n_p + 1$ — Increment the domination counter of p
  if $n_p = 0$ then — p belongs to the first front
    $p_{\text{rank}} = 1$
    $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$

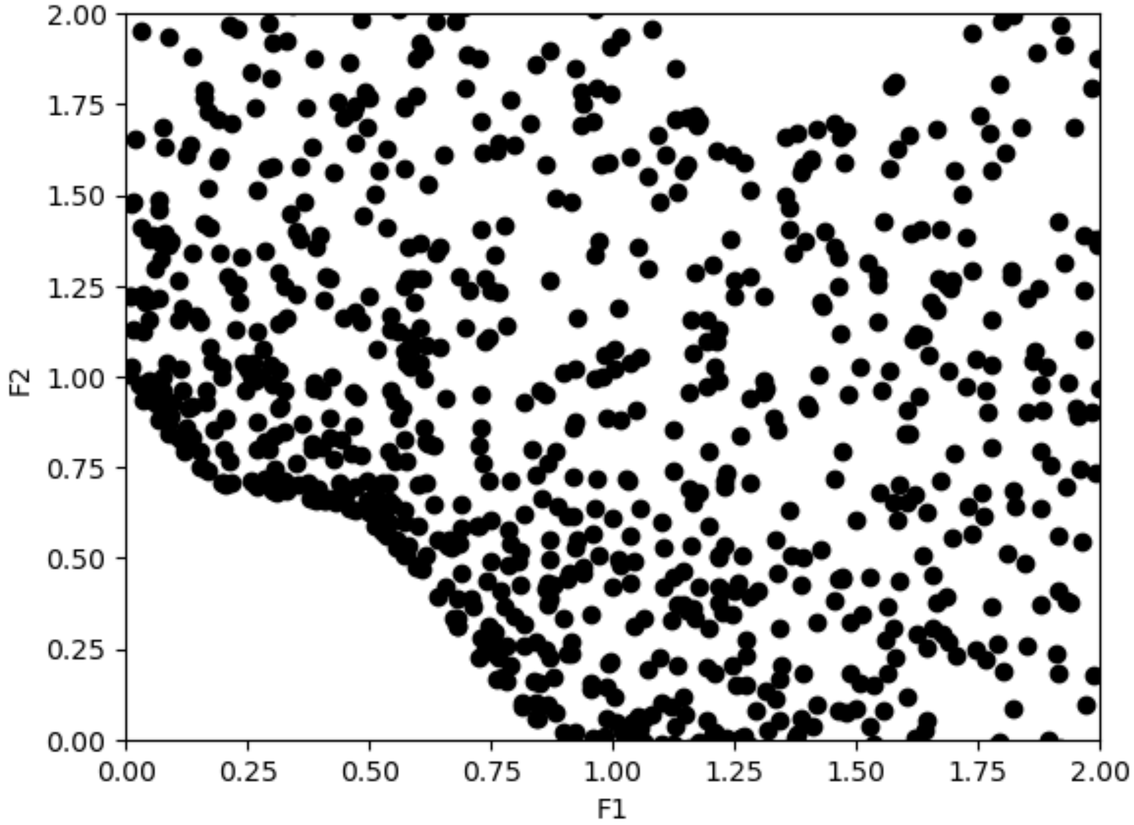
$i = 1$ — Initialize the front counter
while $\mathcal{F}_i \neq \emptyset$
  $Q = \emptyset$ — Used to store the members of the next front
  for each $p \in \mathcal{F}_i$ <- *for each element belonging to the current Pareto front*
    for each $q \in S_p$ <- *for each element dominated by p*
      $n_q = n_q - 1$ <- *decrease its domination number*
      if $n_q = 0$ then — q belongs to the next front
        $q_{\text{rank}} = i + 1$
        $Q = Q \cup \{q\}$
  $i = i + 1$
  $\mathcal{F}_i = Q$

Our good old concave task

$$f_1(x_1, x_2) = x_1$$

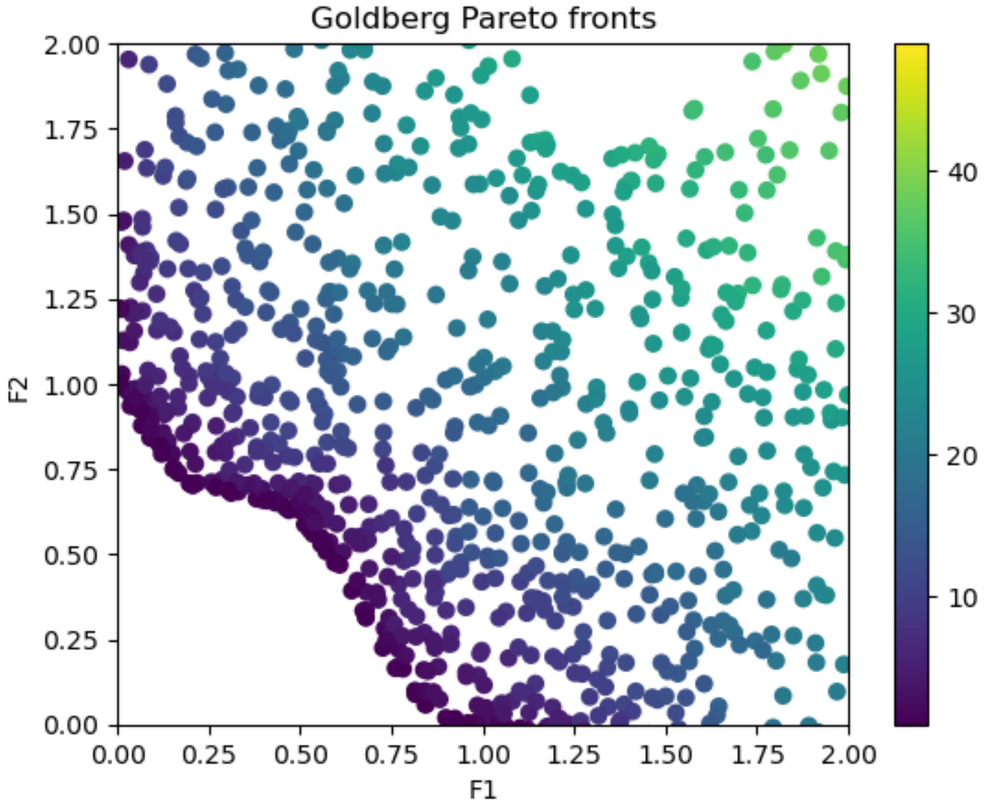
$$f_2(x_1, x_2) = 1 - x_1 - \alpha \sin(\beta \pi x_1) + x_2^2$$

$$\alpha = 1/10 \qquad \beta = 3$$

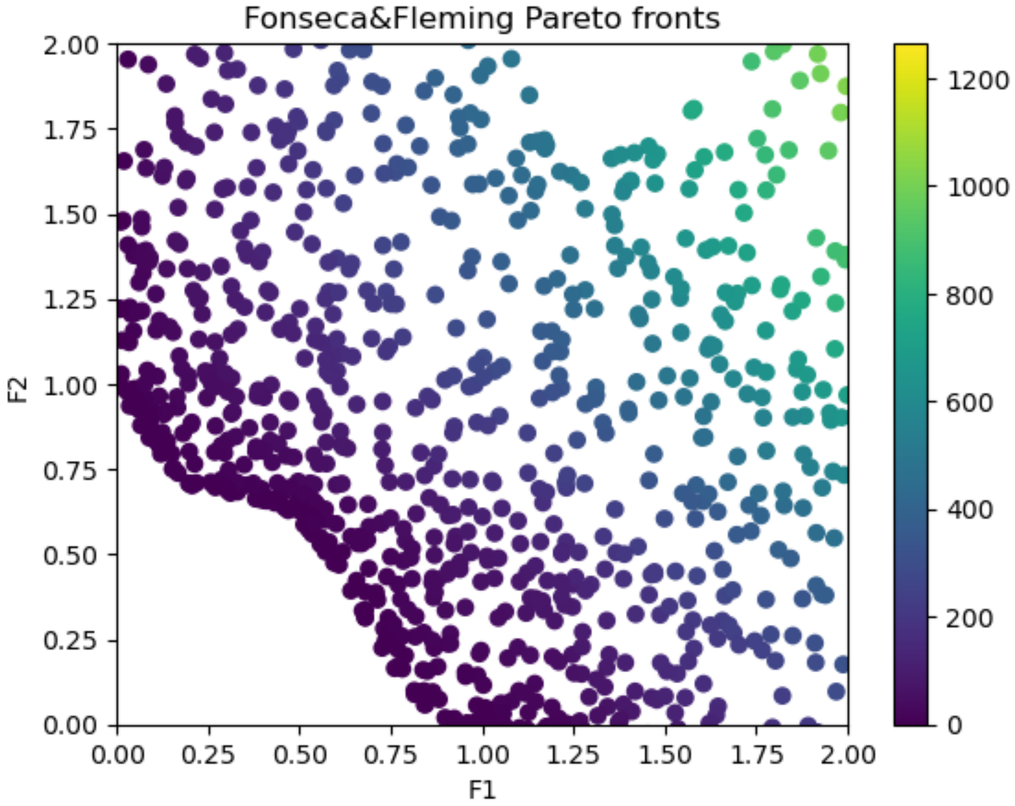


Our good old concave task

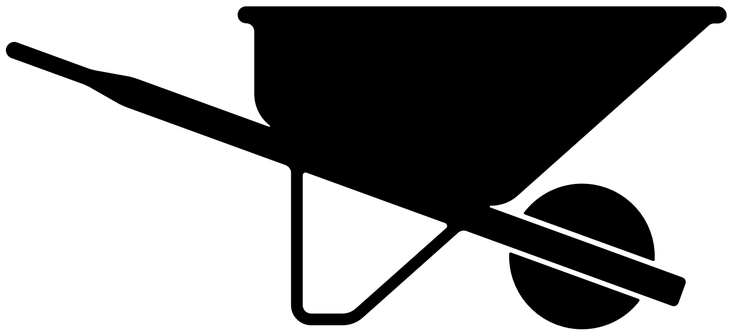
Goldberg



Fonseca and Fleming



Hands on



Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Physics-inspired algorithm

NP-hard problems can use the Metropolis's model for simulating the annealing of solids

Annealing: heat up a material up to the melting temperature, then slowly cool the material up to a stable crystalline state with minimum energy.

Molecular positions	—>	decision variables
Temperature	—>	control parameter
Energy of the system	—>	objective functions
System state	—>	possible solution
Metastable state	—>	local minimum
Ground state	—>	global minimum

Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Physics-inspired algorithm

Ising model

https://www.youtube.com/watch?v=g1h71X55hyw

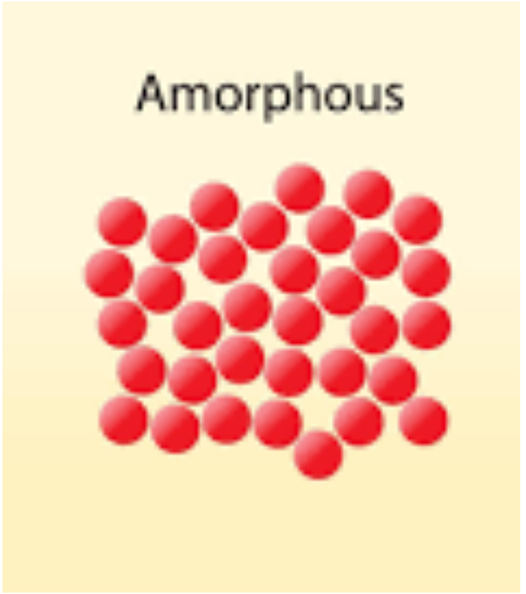
Traveling salesman problem

https://www.youtube.com/watch?v=W-aAjd8_bUc

Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Why does this work in Nature?

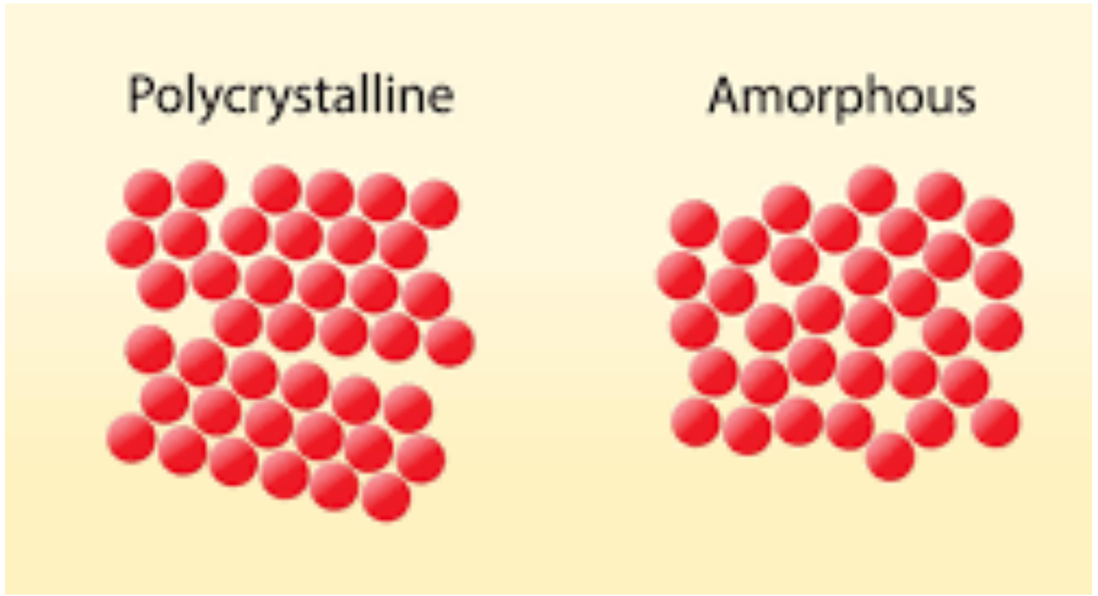
- Melt a solid and then cool it **fast**



Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Why does this work in Nature?

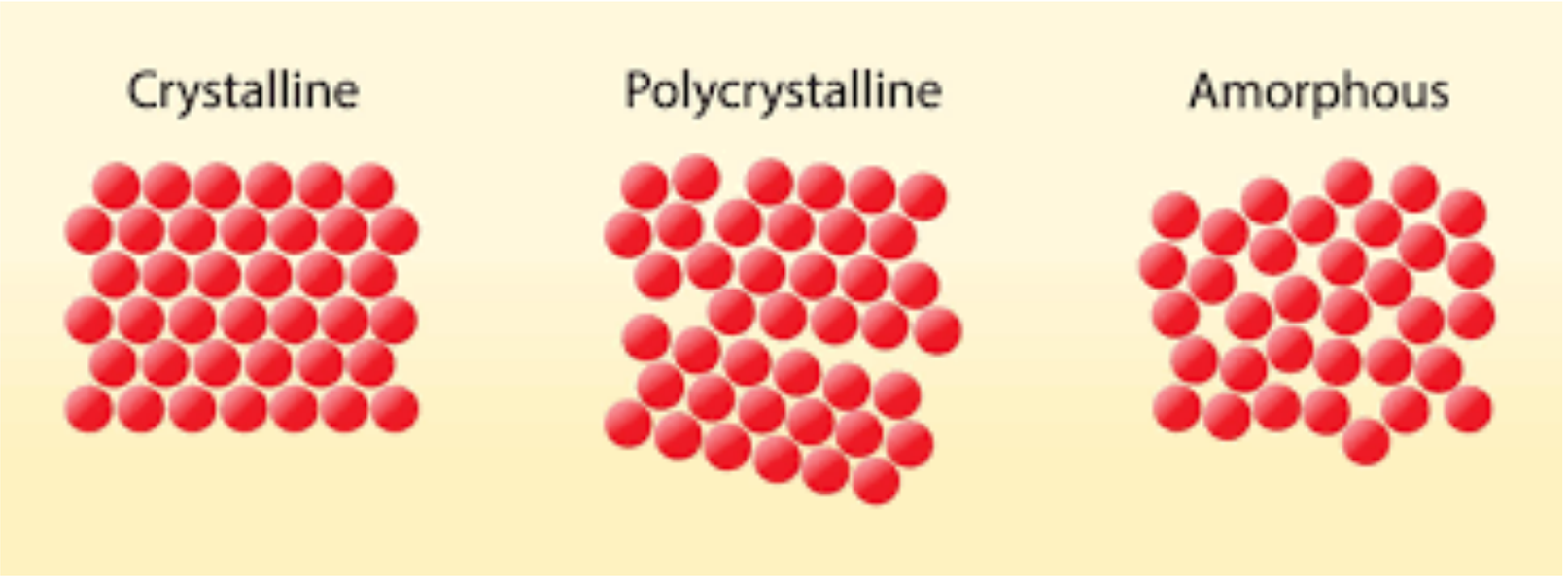
- Melt a solid and then cool it **slowly**



Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Why does this work in Nature?

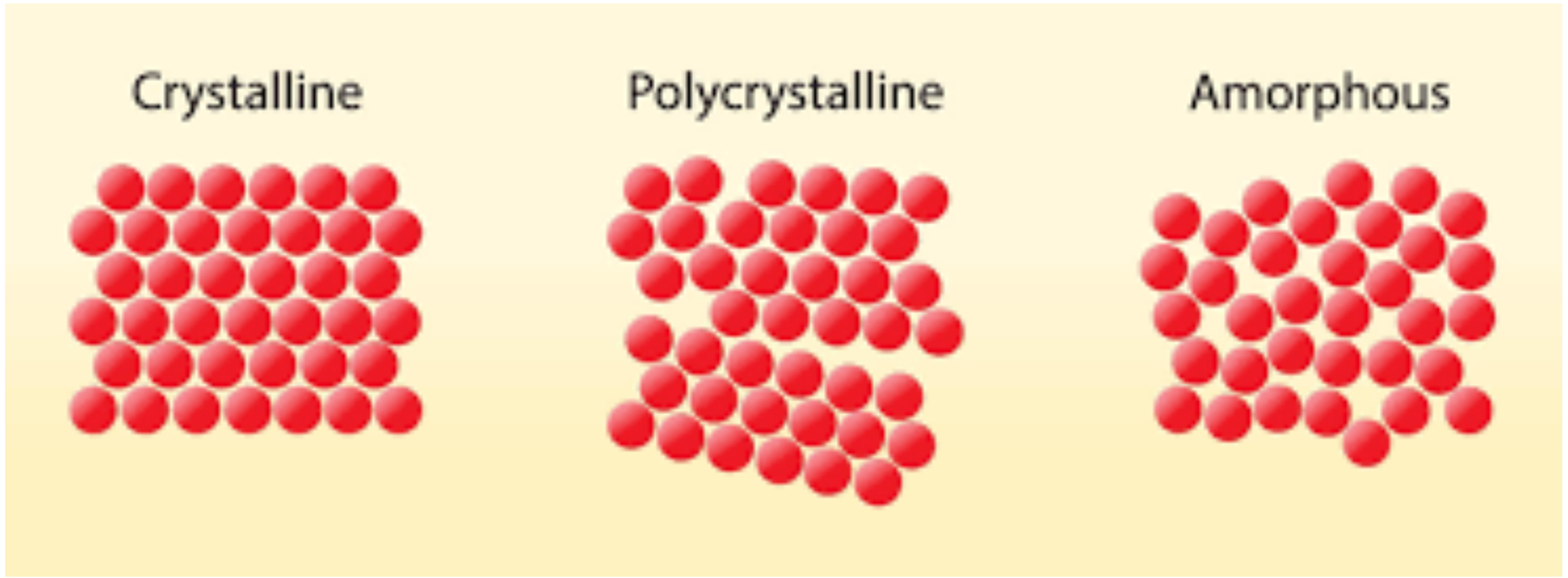
- Melt a solid and then cool it **very slowly**



Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Why does this work in Nature?

- Melt a solid and then cool it **very slowly**



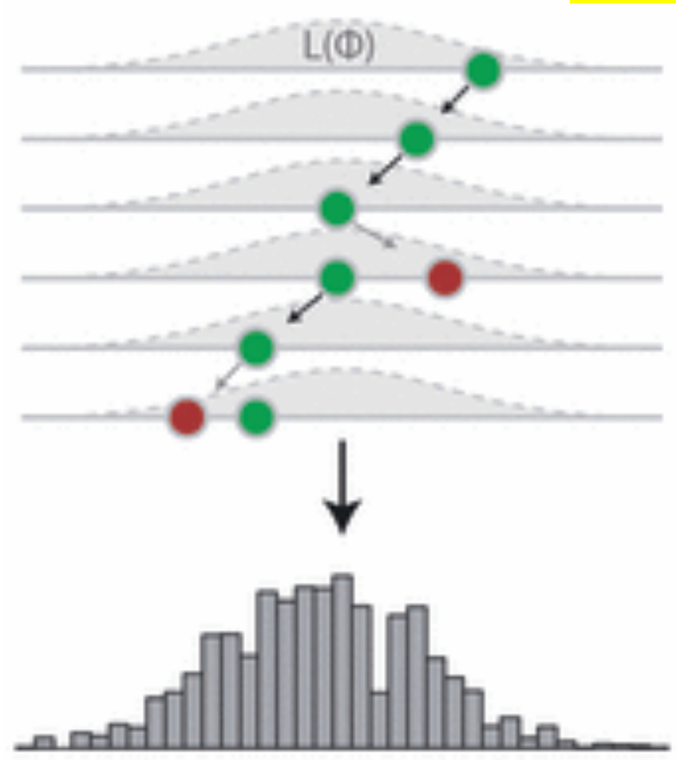
Lowest entropy/energy higher entropy/energy highest entropy/energy

Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Physics-inspired algorithm

Probabilistic evolution:

$$P\{x \to x' : x(t) = x\} = \frac{1}{Z} \exp\left\{ -\frac{1}{T} \max(0, f(x') - f(x)) \right\}$$



Simulated Annealing *(Kirkpatrick,Černy 1980s)*

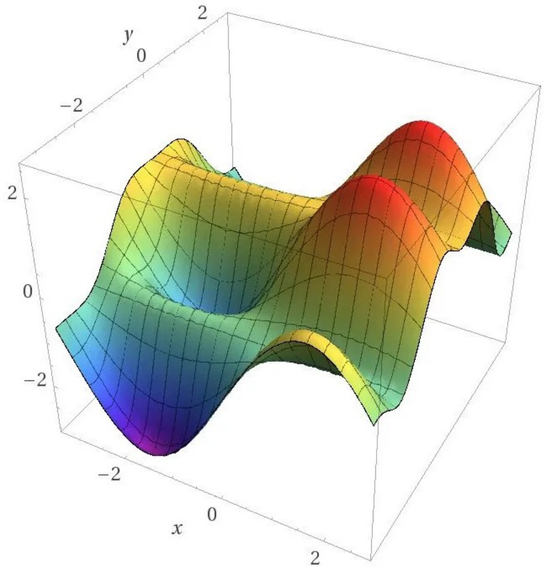
Physics-inspired algorithm

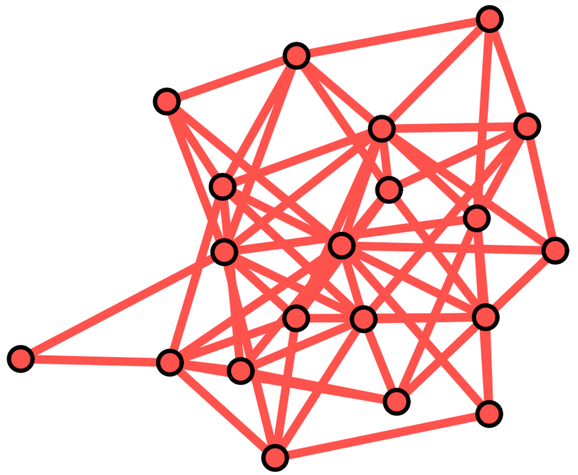
Probabilistic evolution:

$$P\{x \to x' : x(t) = x\} = \frac{1}{Z} \exp\left\{ -\frac{1}{T} \max(0, f(x') - f(x)) \right\}$$

Need to specify:

- Problem representation: define solution space and cost function





Simulated Annealing *(Kirkpatrick,Černy 1980s)*

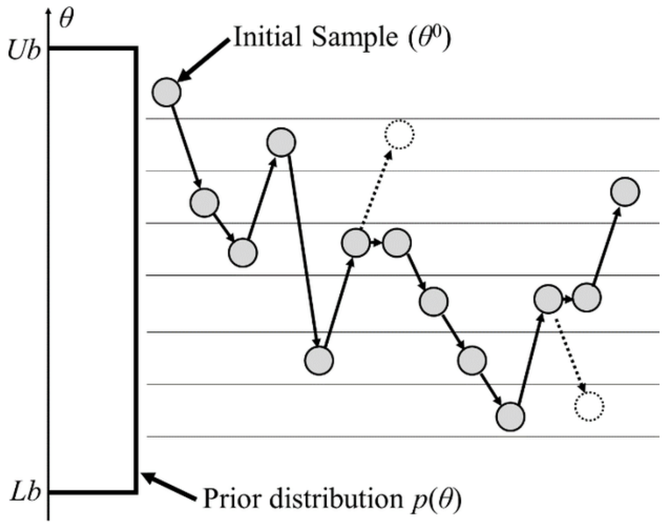
Physics-inspired algorithm

Probabilistic evolution:

$$P\{x \to x' : x(t) = x\} = \frac{1}{Z} \exp\left\{ -\frac{1}{T} \max(0, f(x') - f(x)) \right\}$$

Need to specify:

- Problem representation: define solution space and cost function
- Generation mechanism:
  - How to generate a new solution x—>x’
  - Neighbourhood and sampling function



Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Physics-inspired algorithm

Probabilistic evolution:

$$P\{x \to x' : x(t) = x\} = \frac{1}{Z} \exp\left\{ -\frac{1}{T} \max(0, f(x') - f(x)) \right\}$$

Need to specify:

- Problem representation: define solution space and cost function
- Generation mechanism:
  - How to generate a new solution x—>x’
  - Neighbourhood and sampling function
- Cooling procedure: how to decrease T at each step

Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Physics-inspired algorithm

Probabilistic evolution:

$$P\{x \to x' : x(t) = x\} = \frac{1}{Z} \exp\left\{ -\frac{1}{T} \max(0, f(x') - f(x)) \right\}$$

Need to specify:

- Problem representation: define solution space and cost function
- Generation mechanism:
  - How to generate a new solution x—>x’
  - Neighbourhood and sampling function
- Cooling procedure: how to decrease T at each step
- Handling constraint: $Obj(x, \alpha) = f(x) + \sum_{j=1}^{m} \alpha_j \, (g_j(x))^2$

Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Cooling methods

- Linear cooling

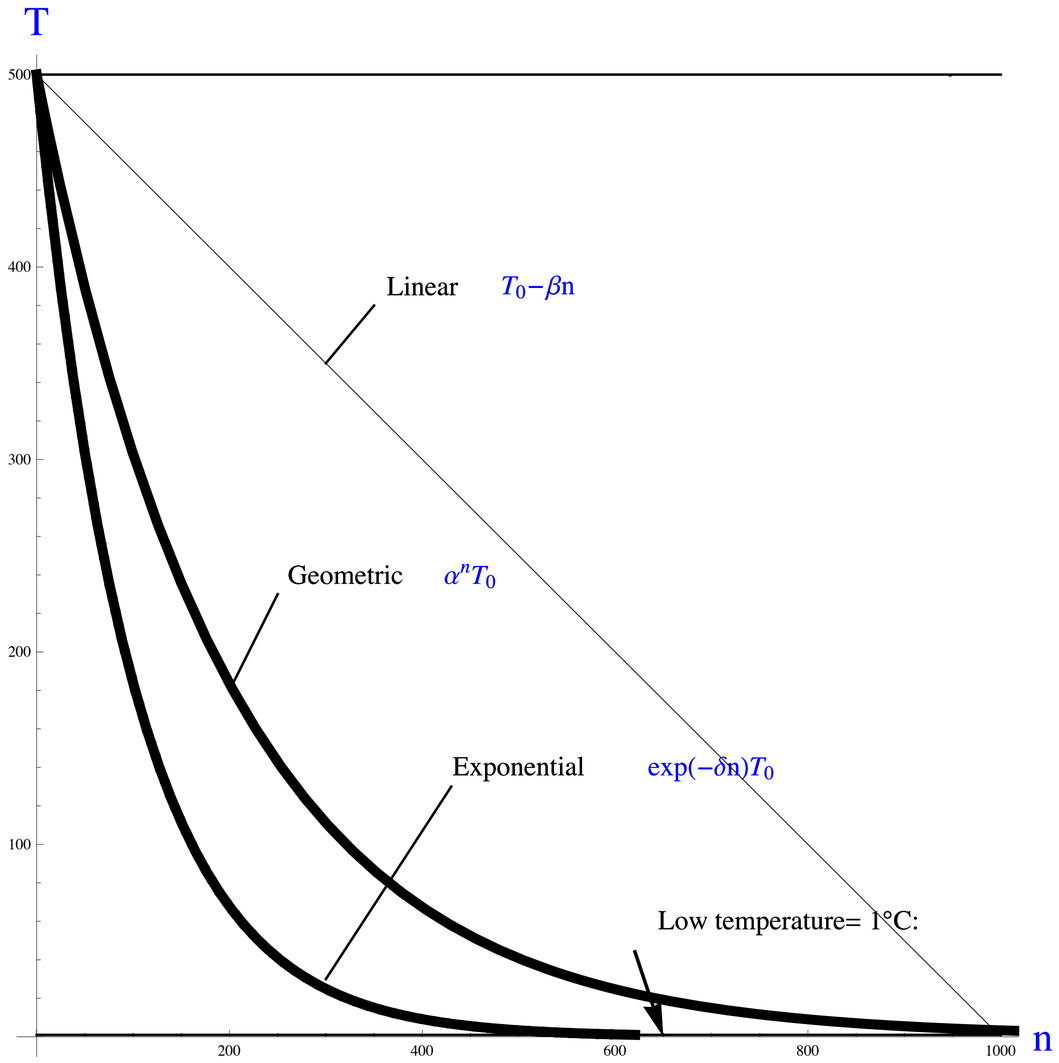
$$T[n] = T[n-1] - \Delta_0$$

- Geometric cooling

$$T[n] = \alpha T[n-1] \qquad \alpha \in [0,1] \qquad \alpha = 0.995$$

- Exponential cooling

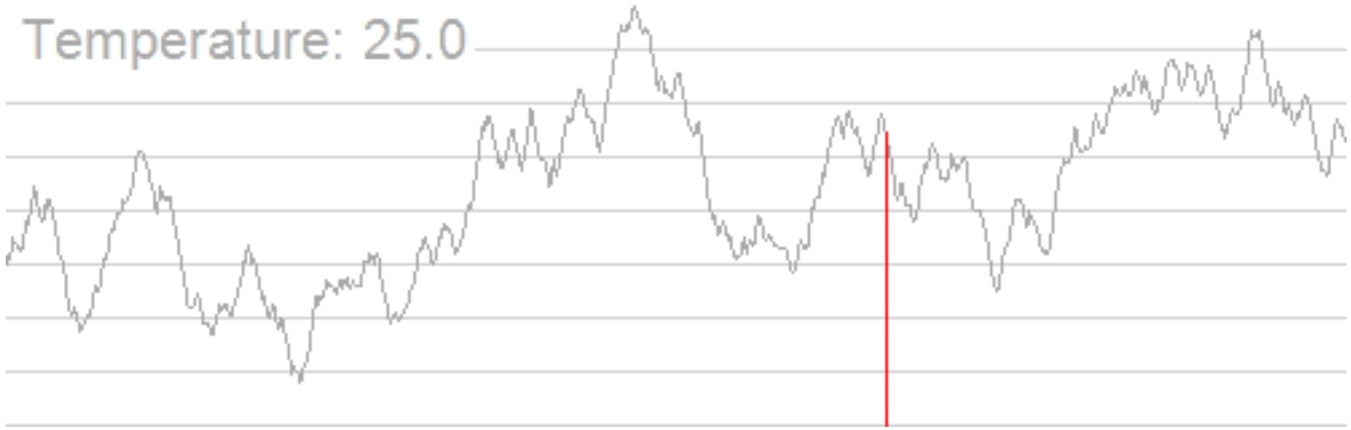
$$T[n] = e^{-\beta} T[n-1] \qquad \beta \in [0,1] \qquad \beta = 0.01$$



Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Pseudo-algorithm

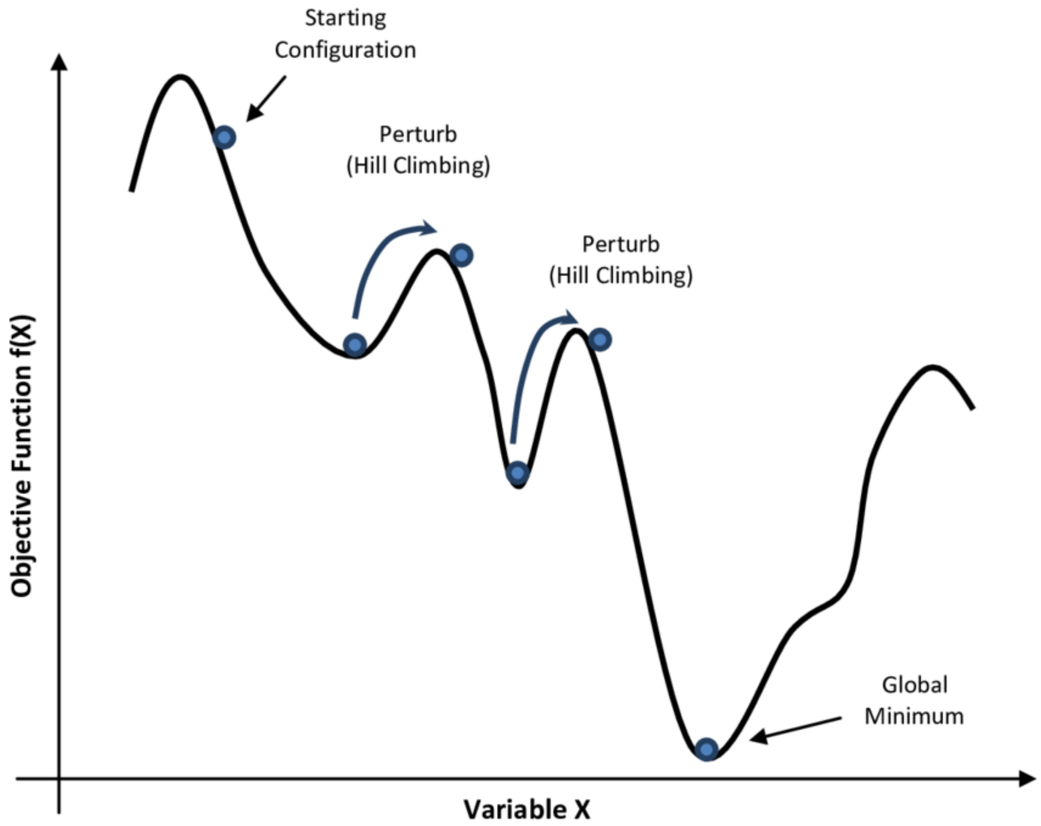
```
T=T0     set initial temperature
X = X0   random initial solution
E = Energy(X)    compute the initial energy
While T>0 or not (termination_criterion):
        X’ = generate_solution(x,N(x))
        E’ = Energy(X’)
        Delta = E’-E
        If Delta < 0:
                X=X’
                E=E’
        Else:
                p=exp( - Delta/ (bT) )
                If p <= random(0,1):
                        X=X’
                        E=E’
        T = cooling(T)
```



Simulated Annealing *(Kirkpatrick,Černy 1980s)*

Pseudo-algorithm

```
T=T0     set initial temperature
X = X0   random initial solution
E = Energy(X)    compute the initial energy
While T>0 or not (termination_criterion):
        X' = generate_solution(x,N(x))
        E' = Energy(X')
        Delta = E'-E
        If Delta < 0:
                X=X'
                E=E'
        Else:
                p=exp( - Delta/ (bT) )
                If p <= random(0,1):
                        X=X'
                        E=E'
        T = cooling(T)
```



MO Simulated Annealing *(Ulungu et al. 1999)*

What changes?

Transition probability x —> x’ should adapt to multiple objectives.

- If x’ dominates x (case A):

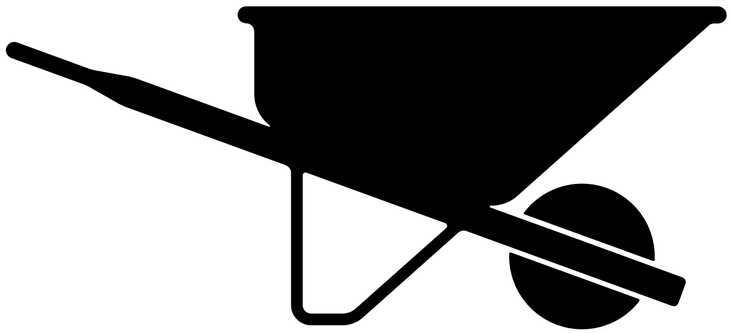
$$P\{x \to x' : x(t) = x\} = 1$$

- If x’ is dominated by x (case C) or if there is no clear dominance relation

$$P\{x \to x' : x(t) = x\} = \prod_{i=1}^{n_{obj}} \exp\left\{-\frac{\Delta f_i}{T(t)}\right\}$$

	$\Delta f_1 \leq 0$	$\Delta f_1 > 0$
$\Delta f_2 \leq 0$	**A**	**B**
$\Delta f_2 > 0$	**B**	**C**

Hands on

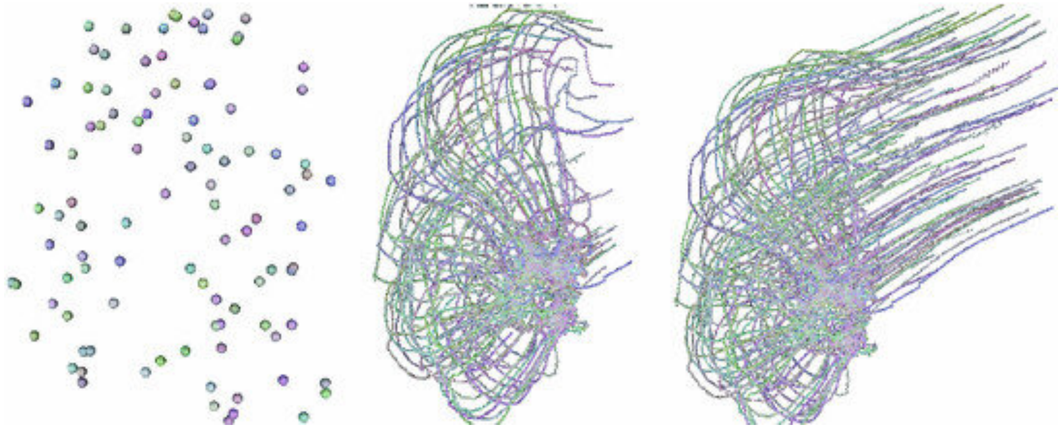


Particle Swarm Optimization *(Kennedy, Eberhart 1995)*

Nature-inspired social behavior

The swarm/population is the set of current solutions

A particle is a single solution



Social behavior is sketched using:

- Memory/Cognition
- Social interaction and communication

Particle Swarm Optimization *(Kennedy, Eberhart 1995)*

Nature-inspired social behavior

We need to track

- Position $\overrightarrow{x}(t) = (x_1(t), \ldots, x_D(t))^T$
- Velocity $\overrightarrow{v}(t) = (v_1(t), \ldots, v_D(t))^T$
- Personal best $\overrightarrow{x_b}(t) = (x_{b1}(t), \ldots, x_{bD}(t))^T$
- Leader's best (global or local) $\overrightarrow{x_n}(t) = (x_{n1}(t), \ldots, x_{nD}(t))^T$

Leader: particle guiding another particle in the search space

Particle Swarm Optimization *(Kennedy, Eberhart 1995)*

Visual inspection

Each iteration requires updating the particles' velocity and position

$$\vec{v}_i(t+1) = \omega \vec{v}_i(t) + c_1 r_1 \left(\vec{x}_{bi} - \vec{x}_i(t) \right) + c_2 r_2 \left(\vec{x}_{ni} - \vec{x}_i(t) \right)$$

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

Velocity is given by:

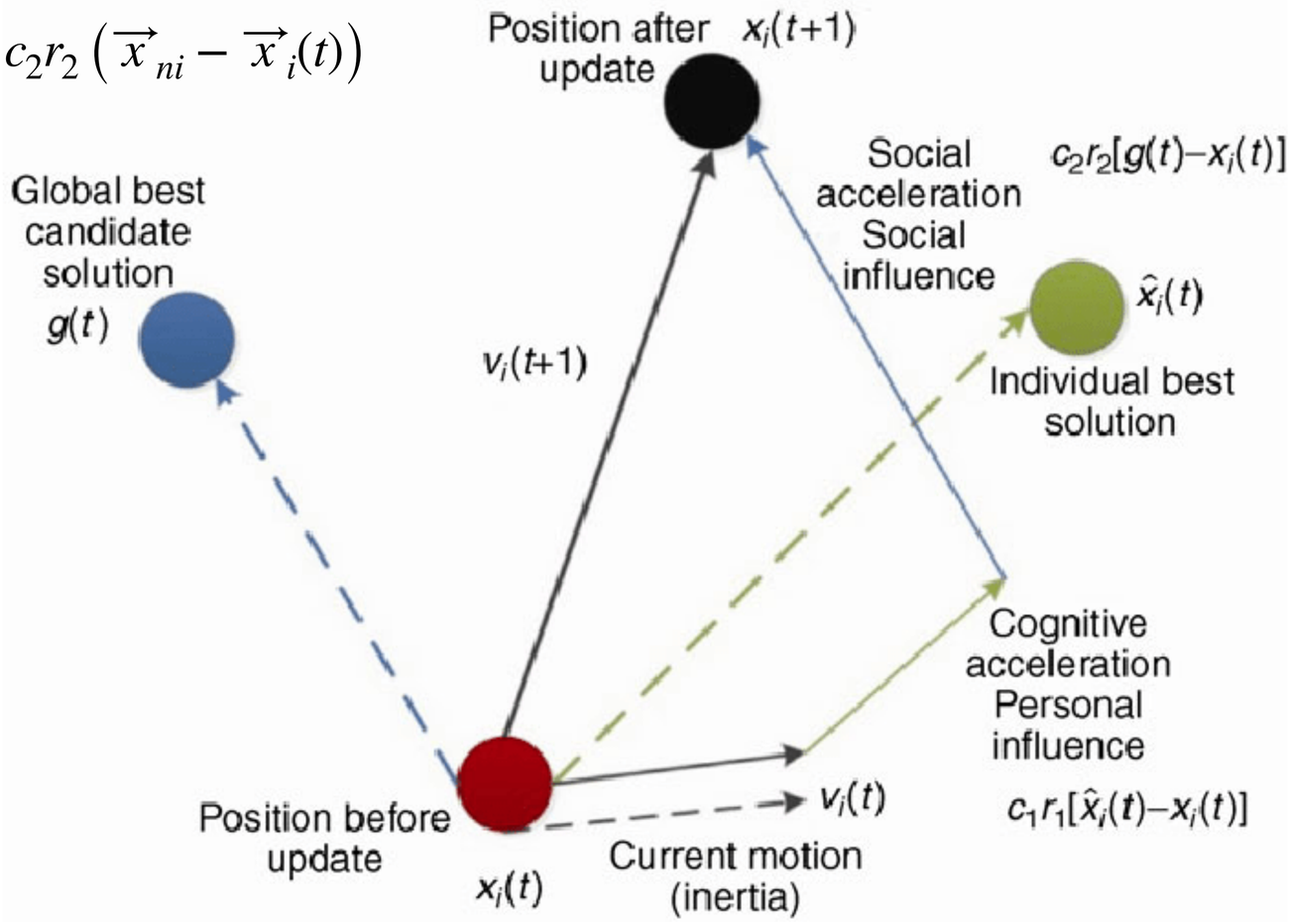
- Current flight direction —> ω inertia
- Cognitive component —> c_1 cognitive learning factor
- Social component —> c_2 social learning factor
- Randomness —> $r_1, r_2 \in$ Random$[0,1]$

Particle Swarm Optimization *(Kennedy, Eberhart 1995)*

Dynamic equations

$$\vec{v}_i(t+1) = \omega \vec{v}_i(t) + c_1 r_1 \left(\vec{x}_{bi} - \vec{x}_i(t)\right) + c_2 r_2 \left(\vec{x}_{ni} - \vec{x}_i(t)\right)$$

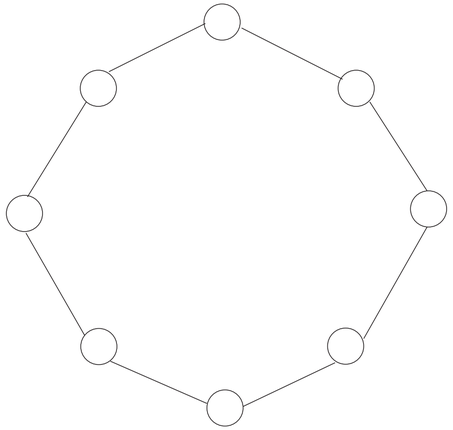
$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1)$$



Particle Swarm Optimization *(Kennedy, Eberhart 1995)*

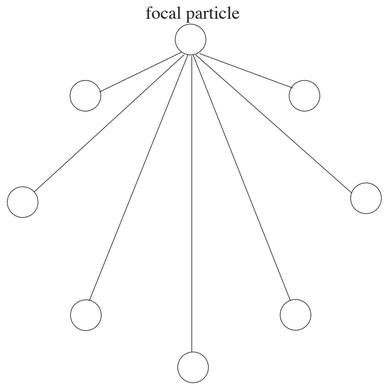
Neighborhood definitions

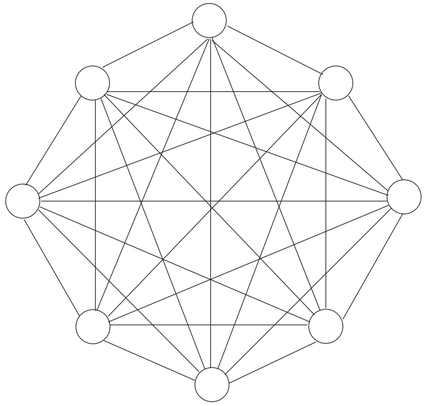
Social interaction can be represented as a graph:



RING KNN=2

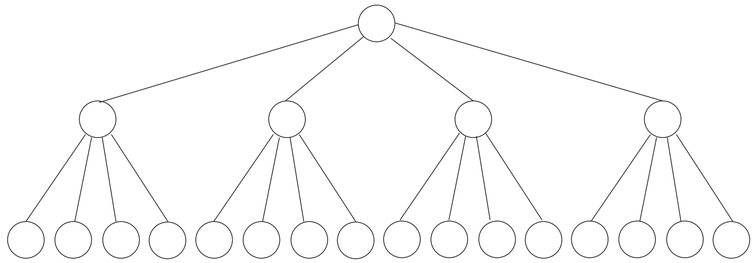
STAR NETWORK





FULLY CONNECTED

TREE HIERARCHICAL



Particles are influenced by the success of anyone they are connected to!

PSO

Pseudocode

Begin
    Initialize swarm
    Locate leader
    $g = 0$
    While $g < gmax$
        For each particle
            Update Position (Flight)
            Evaluation
            Update $pbest$
        EndFor
        Update leader
        g++
    EndWhile
End

PSO

Pseudocode

```
Begin
    Initialize swarm
    Locate leader
    g = 0
    While g < gmax
            For each particle
                    Update Position (Flight)
                    Evaluation
                    Update pbest
            EndFor
            Update leader
            g++
    EndWhile
End
```

The outcome is strongly influenced by:

- Hyperparameters
- Social topology

PSO

Pseudocode

```
Begin
    Initialize swarm
    Locate leader
    g = 0
    While g < gmax
            For each particle
                    Update Position (Flight)
                    Evaluation
                    Update pbest
            EndFor
            Update leader
            g++
    EndWhile
End
```

The outcome is strongly influenced by:

- Hyperparameters
- Social topology

$\omega : 0 \rightarrow 1$ min-to-max momentum
$c_1 : 0 \rightarrow 1$ min-to-max impact of memories
$c_2 : 0 \rightarrow 1$ min-to-max social pressure

In particular $c_2 = 0 \cap c_1 > 0$ leads to an “everyone for himself” situation, and the system does not converge.

No particle is an island!

Multi-Objective PSO (MOPSO)

What changes?

Need to:

- Find as many optimal solutions as possible
    - The notion of best is not unique anymore
    - Need to initialize P archives for personal best (non-dominated) solutions
    - Need to initialize an archive for global best solutions
- Maximize their spread/search space

But:

- How to select among potential leaders/personal best?
- How to update the internal archive?
- How to maintain diversity in the archive?

Multi-Objective PSO (MOPSO)

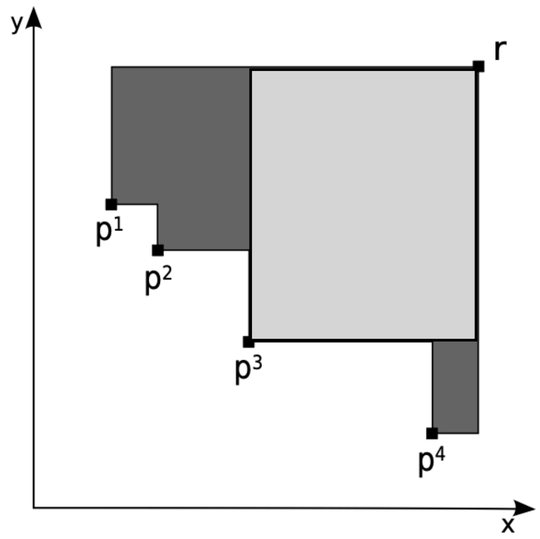
What changes?

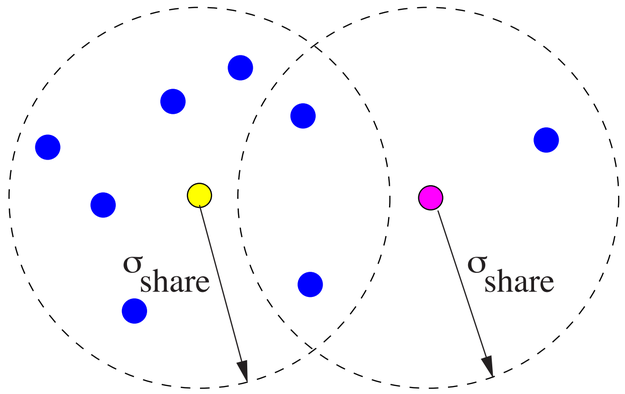
- How to select among potential leaders/personal best?
    - Random selection

Multi-Objective PSO (MOPSO)

What changes?

- How to select among potential leaders/personal best?
    - Random selection
    - Selection based on performance indicators

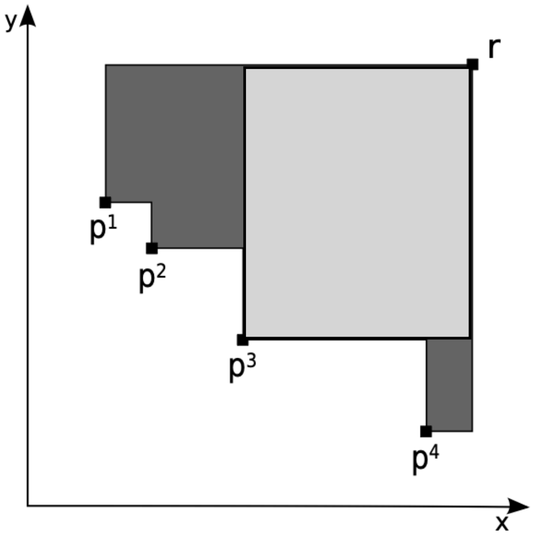


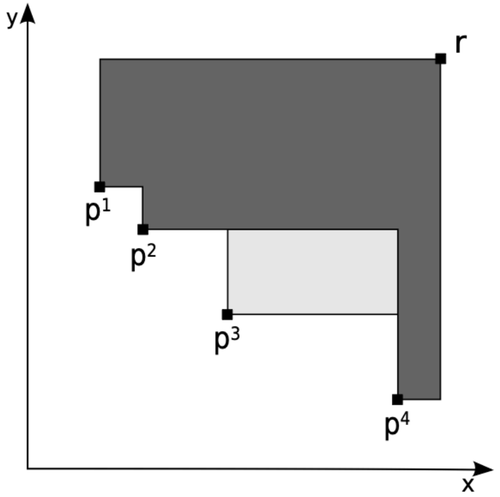


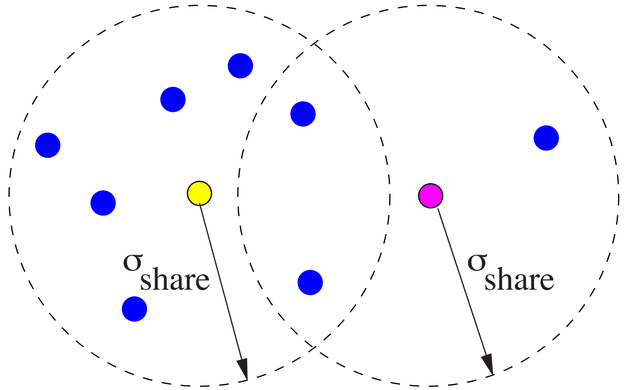
Multi-Objective PSO (MOPSO)

What changes?

- How to select among potential leaders/personal best?
  - Random selection
  - Selection based on performance indicator
  - Selection based on exclusive dominance hypervolume



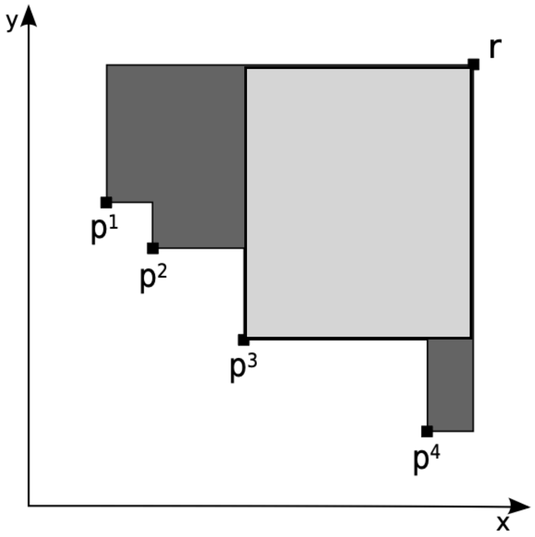


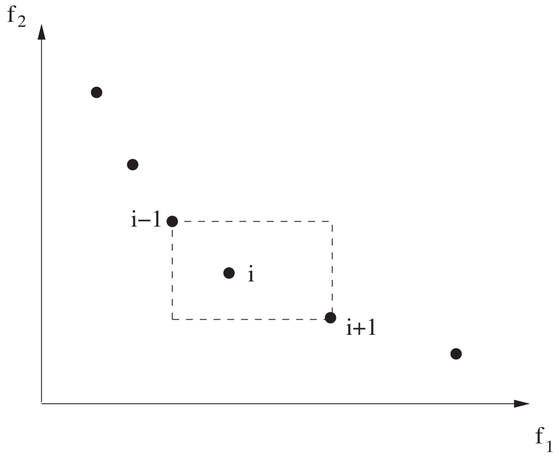


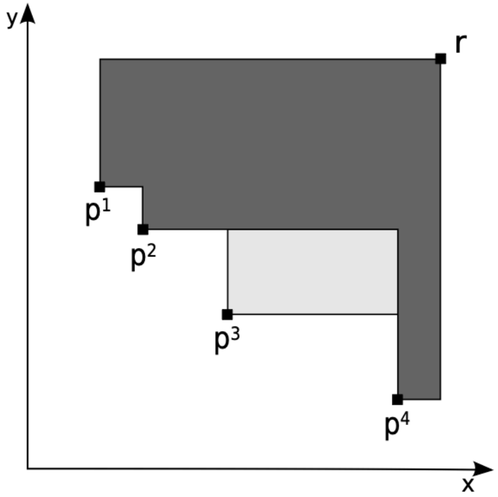
Multi-Objective PSO (MOPSO)

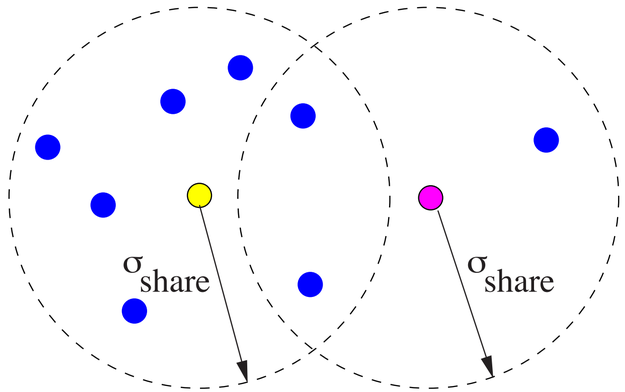
What changes?

- How to select among potential leaders/personal best?
  - Random selection
  - Selection based on performance indicator
  - Selection based on exclusive dominance hypervolume
  - Selection based on NN density estimate





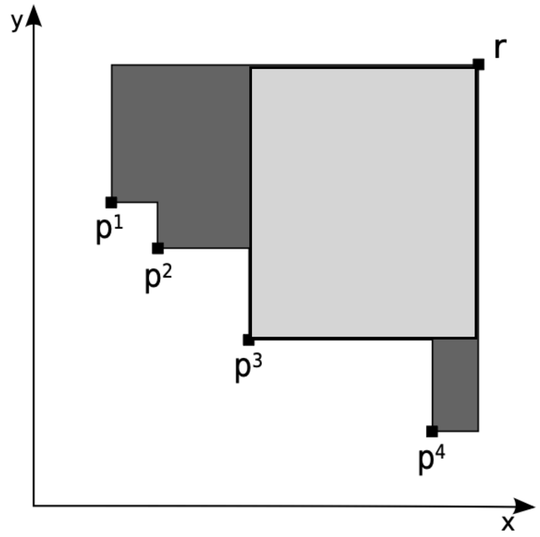




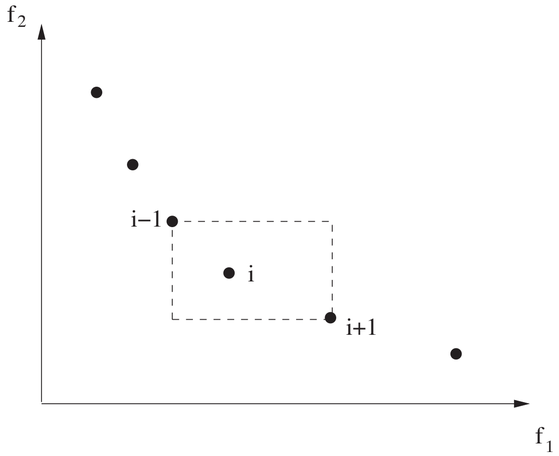
Multi-Objective PSO (MOPSO)

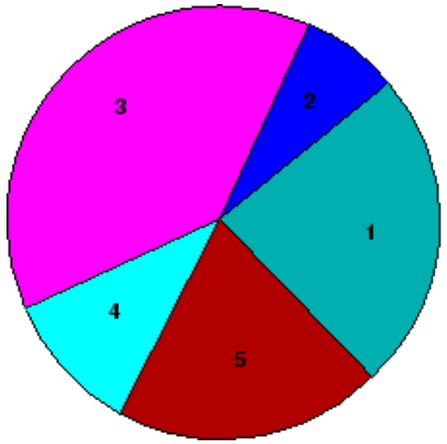
What changes?

- How to select among potential leaders/personal best?
  - Random selection
  - Selection based on performance indicator
  - Selection based on exclusive dominance hypervolume
  - Selection based on NN density estimate
  - …

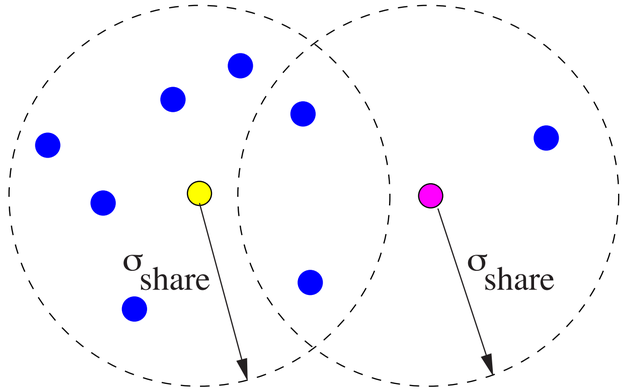


(best or roulette selection)





Population	Fitness
1	25.0
2	5.0
3	40.0
4	10.0
5	20.0



Multi-Objective PSO (MOPSO)

What changes?

- How to update the internal archive?
  - If no clear dominance relation between the archive and the new solution

    -> add position and scores to the archive list
  - If new solution dominates those archived

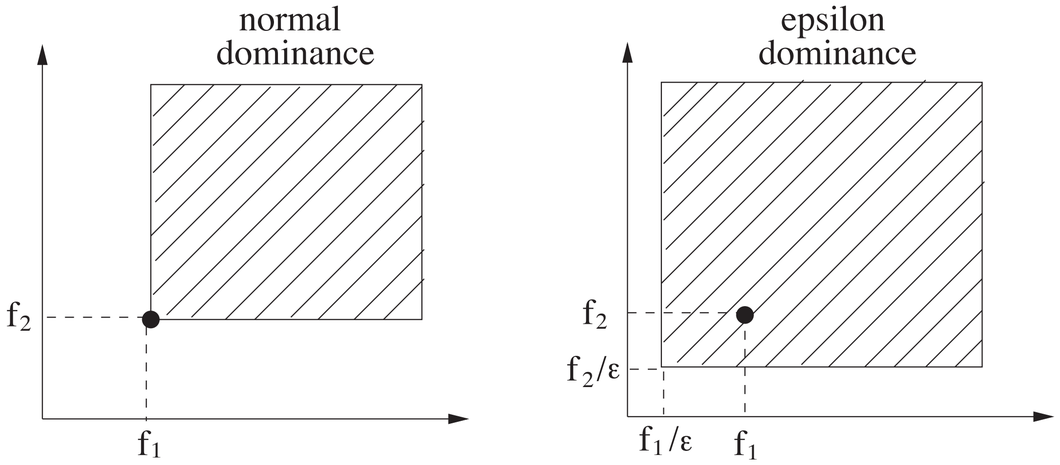
    -> re-initialize the archive with the new position and scores

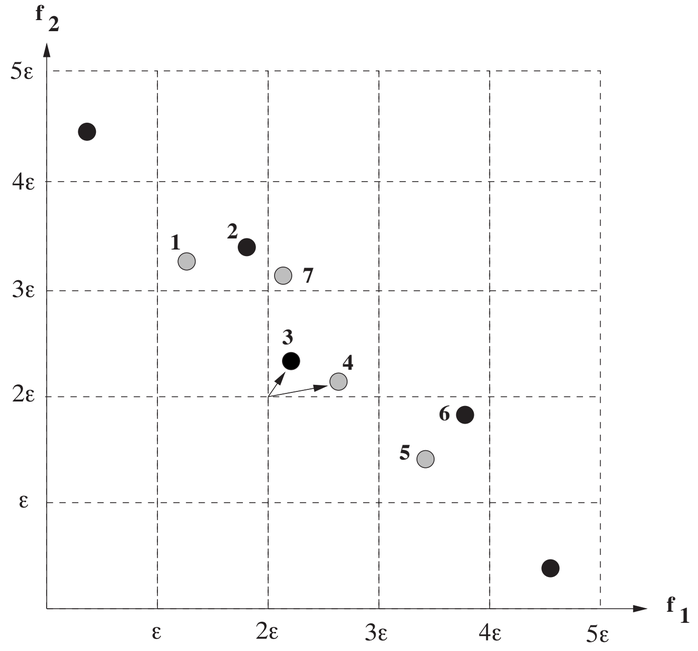
This is true for both the personal archives and the global archive

Multi-Objective PSO (MOPSO)

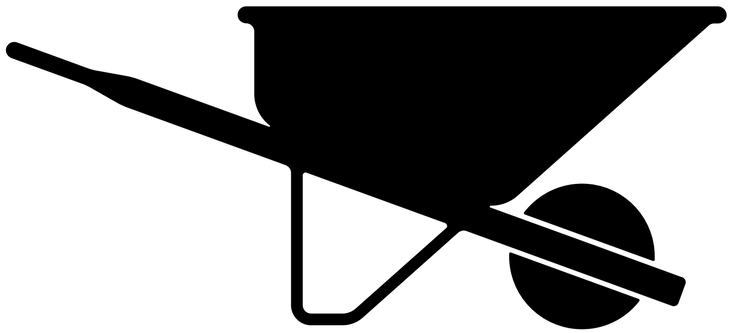
What changes?

- How to maintain diversity in the archive? When the archive limit is reached…
    - Use clustering technique
    - Use niche count to drop solutions (cutoff + roulette?)
    - Use epsilon-dominance
    - …





Hands on



Ant Colony Optimization (Dorigo, 1992)

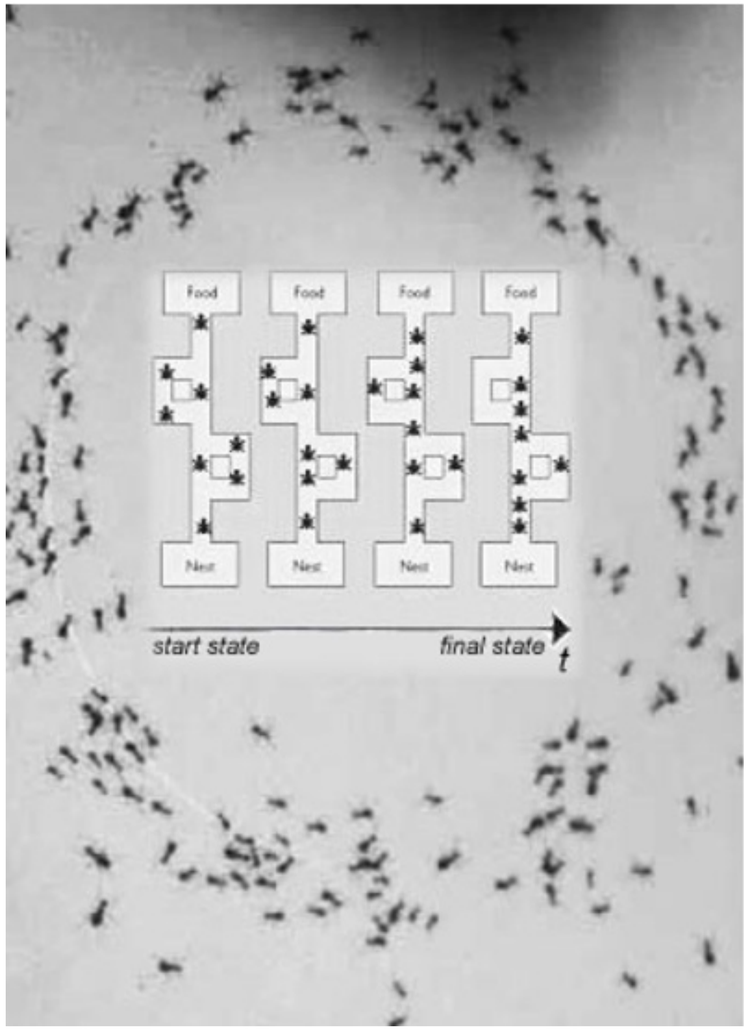
Multi-agent method inspired by ant behavior

Some ant species are able to communicate without physical interaction.

Once moved to a new place, ants move at random (initially)

After they find some food source, ants leave pheromone on their way back to the nest

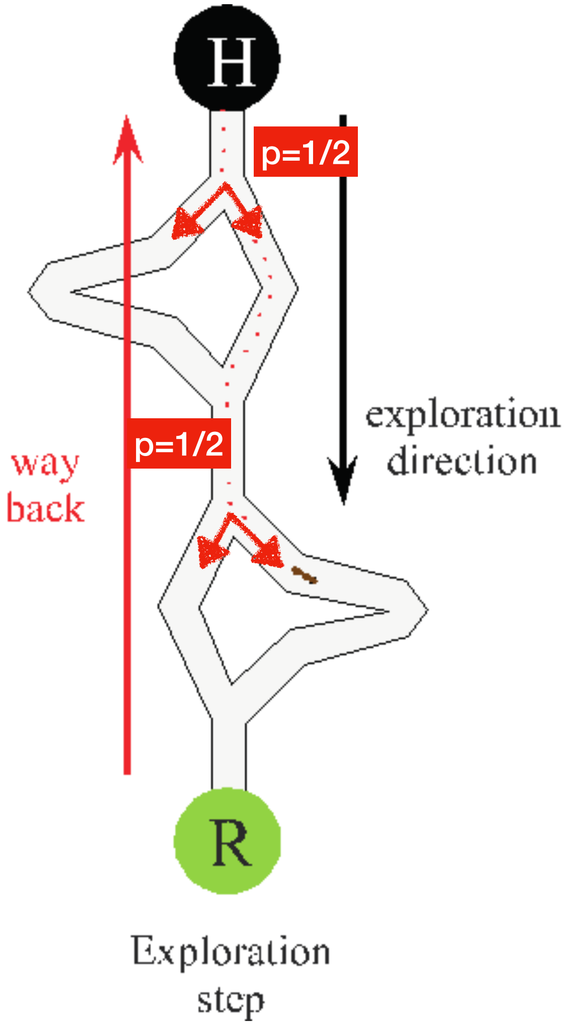
Pheromone evaporates in time



Ant Colony Optimization (Dorigo, 1992)

Warm-up

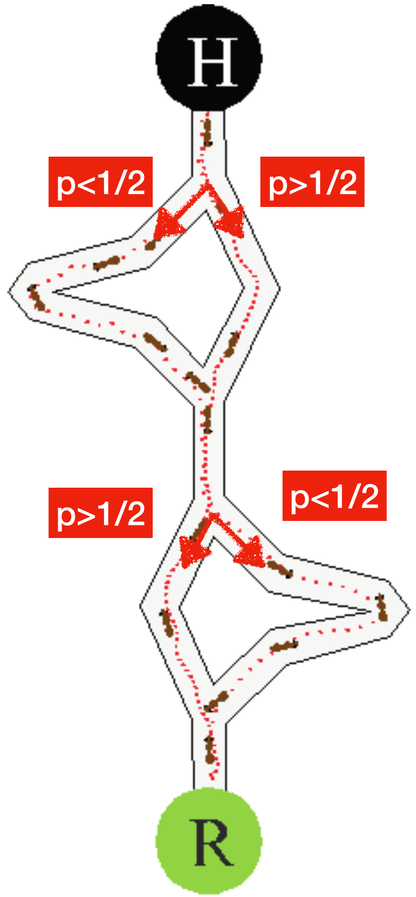
Probabilistic evolution



Ant Colony Optimization (Dorigo, 1992)

Warm-up

Probabilistic evolution

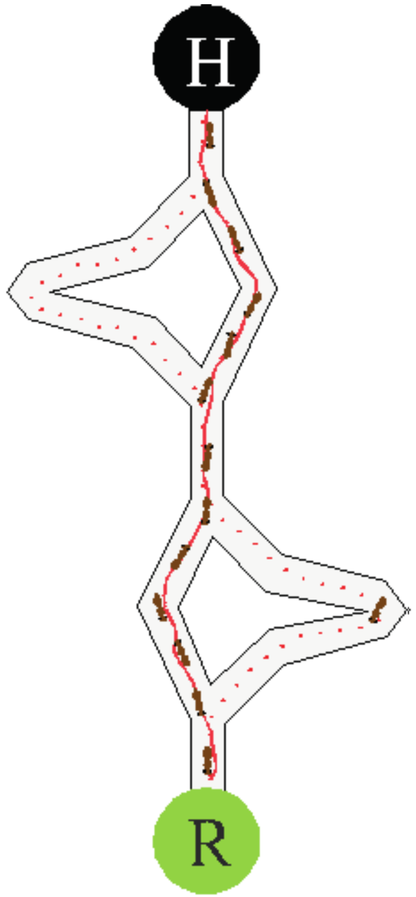


Optimisation step

Ant Colony Optimization (Dorigo, 1992)

Warm-up

Probabilistic evolution

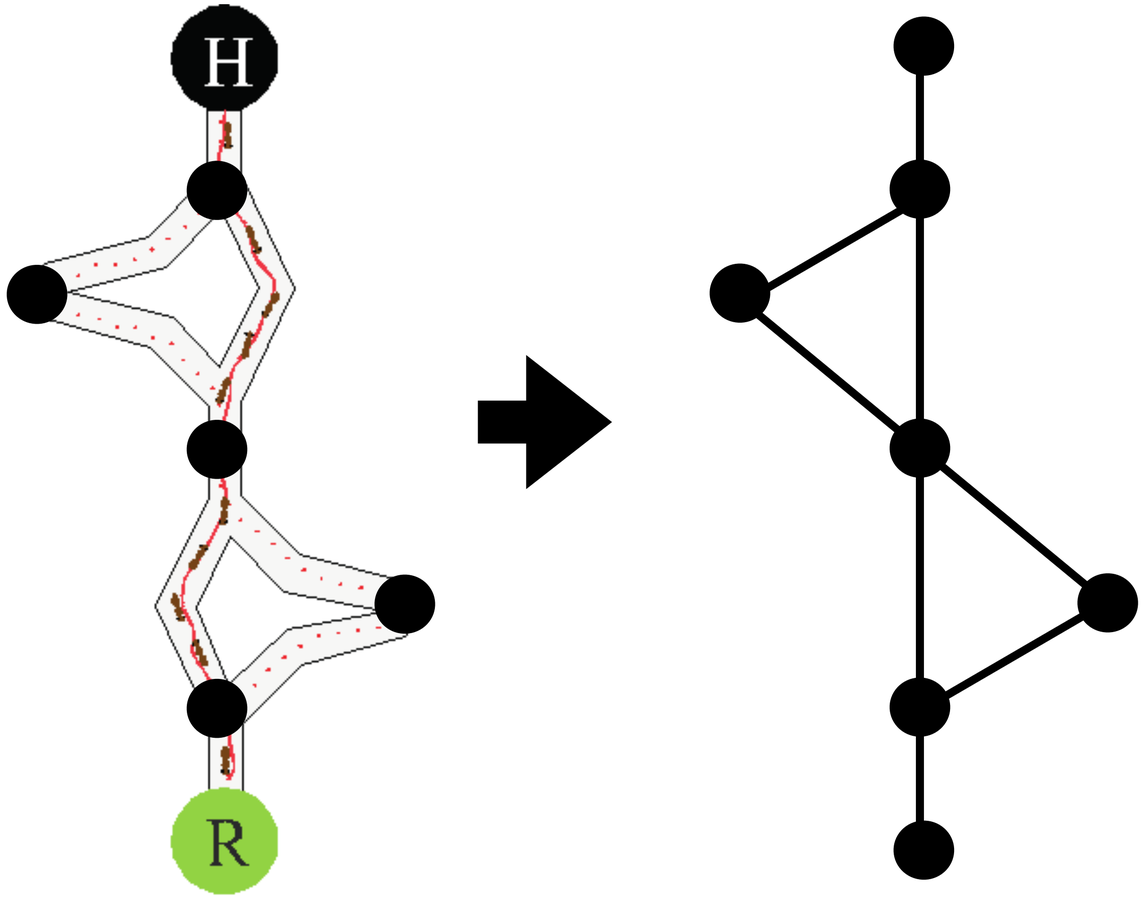


Research extraction step

Ant Colony Optimization (Dorigo, 1992)

Warm-up

Probabilistic evolution



Research extraction step

Ant Colony Optimization (Dorigo, 1992)

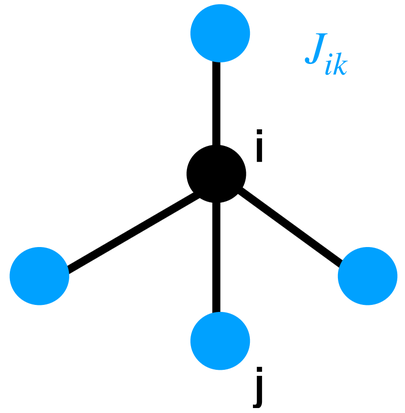
The algorithm

Introduce a heuristic (benefit/cost) matrix: η_{ij} $\quad \eta_{ij} = 1/d_{ij}$

Reciprocal of the distance, proximity, between nodes i and j

Initialize a pheromone matrix: τ_{ij}

Pheromone between nodes i and j



Probability of a path:

$$p_{ij,k}(t) = \begin{cases} \dfrac{(\tau_{ij}(t))^{\alpha}(\eta_{ij}(t))^{\beta}}{\sum_{l \in J_{ik}} (\tau_{il}(t))^{\alpha}(\eta_{il}(t))^{\beta}} & \text{if} \quad j \in J_{ik} \\ 0 & \text{if} \quad j \notin J_{ik} \end{cases}$$

Ant Colony Optimization (Dorigo, 1992)

The algorithm

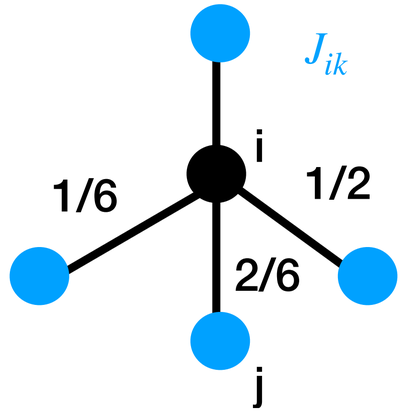
Introduce a heuristic (cost) matrix: η_{ij}

$$\eta_{ij} = 1/d_{ij}$$

Reciprocal of the distance, proximity, between nodes i and j

Initialize a pheromone matrix: τ_{ij}

Pheromone between nodes i and j

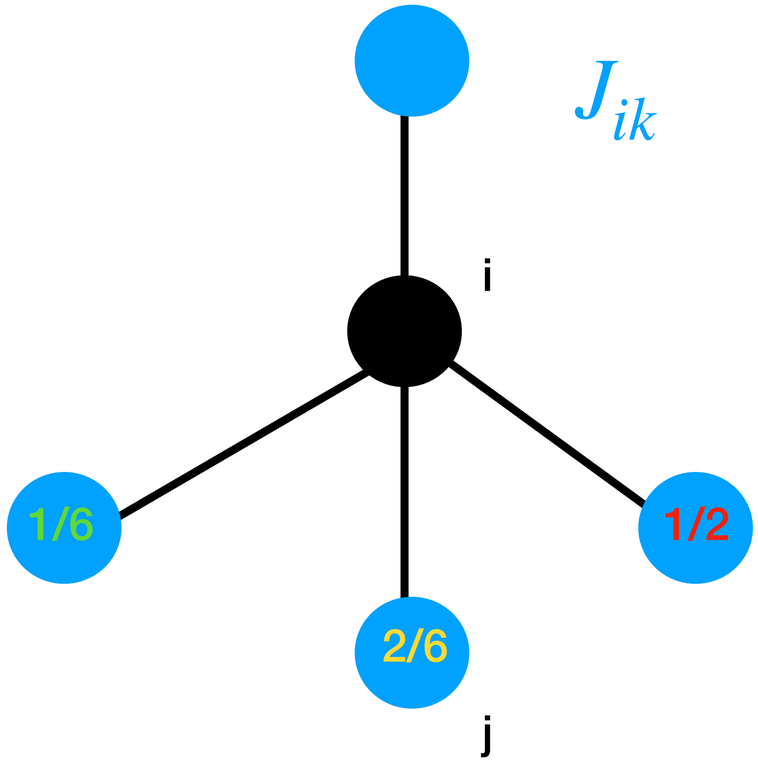


Probability of a path:

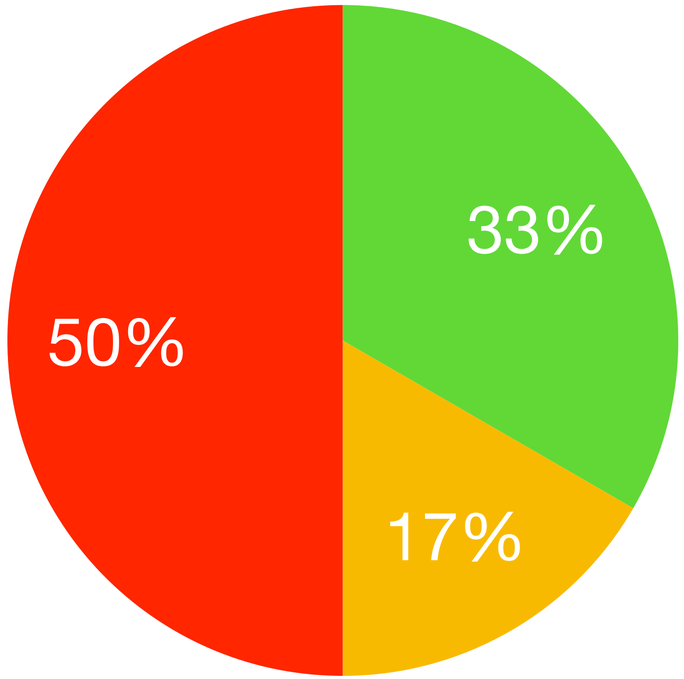
$$p_{ij,k}(t) = \begin{cases} \dfrac{(\tau_{ij}(t))^{\alpha}(\eta_{ij}(t))^{\beta}}{\sum_{l \in J_{ik}} (\tau_{il}(t))^{\alpha}(\eta_{il}(t))^{\beta}} & \text{if} \quad j \in J_{ik} \\ 0 & \text{if} \quad j \notin J_{ik} \end{cases}$$

Ant Colony Optimization (Dorigo, 1992)

Path selection



Roulette: random number in [0,1]



Ant Colony Optimization (Dorigo, 1992)

Pheromone distribution

Each ant releases pheromone on its way back to the nest: $\Delta\tau_{ij}^{k} = \frac{Q}{L_k(t)} \quad \forall (i,j) \in T_k(t)$

Where L_k is the total length of the path of the k-th ant.

The total fresh pheromone deposited on the edge (i,j) becomes $\Delta\tau_{ij} = \sum_{k=1}^{C} \Delta\tau_{ij}^{k}$

and the pheromone update rule is $\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \Delta\tau_{ij}(t)$

where ρ is the pheromone evaporation rate.

In the original algorithm, the pheromone is updated post-factum.

Ant Colony Optimization (Dorigo, 1992)

Tips

- Standard approach: the number of ants is equal to the number of nodes. This basically determines the updating frequency of the pheromone matrix.
- Randomly change the initial point of the search so as to avoid biases.
- Pay attention to the problem definition. The graph structure has a strong impact on the outcome. The problem may not be symmetric, meaning $\eta_{ij} \neq \eta_{ji}$.

Ant Colony Optimization (Dorigo, 1992)

Pseudocode

t=0 #generation counter

(V, E) = problem Graph

η = heuristic tensor

ρ = evaporation rate

v_0 = generate_random_starting_points(V)

τ = InitializePheromoneTensor( )

While (t<G) or not (termination_criterion):

    Paths=[ ]

    For each $k \in [0, C]$: *ant in the colony*

        Path= CreatePath(τ, η, v_0)

        Paths.append(Path)

    τ = PheromoneUpdate(τ, η, Paths)

    v_0 = generate_random_starting_points(V)

    t=t+1

Ant Colony Optimization (Dorigo, 1992)

Extensions

Elitist ant system: some elite ants are used to follow the best route found so far. Push optimal solutions.

Max-min ant system: min and max cutoffs to the pheromone update. Expand the search space and keep all parts of the graph visitable. Reinitialize poor trails when the system is near-stagnation.

Rank-based ant system: solutions are ranked according to their length. Shorter path, higher rank. The amount of pheromone deposited is multiplied by the rank. Ants with low rank leave no pheromones. It increases hierarchy in pheromone release.

…

Multi-Objective ACO

What changes?

From heuristic matrix to heuristic tensor: a benefit/cost vector corresponds to each edge

$$\eta_{ij} -> \eta_{ij,m} \quad m = 1,\ldots,M \qquad \eta_{ij,m} = s_{jk}/c_{ij}$$

Approaches may differ in choosing:

- The number of pheromone matrices
- The number of heuristic matrices
- The aggregation strategy of the above matrices and the hyperparameters associated to it
- The method used to select solutions that update the pheromone information

we present few examples

Multiple Ant Colony System (MACS) *(Barán, 2003)*

Single pheromone matrix: τ_{ij}

Pheromone matrix initialized as. $\tau_0 = \dfrac{1}{\prod_{i=1}^{M} \langle f_m \rangle_{\text{initial pop.}}}$

Multiple cost matrices: $\eta_{ij,m} \quad m = 1,\ldots,M$

Each ant uses a different normalized weight for cost aggregation: $\vec{\lambda}_k = (\lambda_{1,k} \ldots \lambda_{M,k}) \qquad k = 1,\ldots,C$

The update rule becomes:

$$p_{ij,m}^{(k)}(t) = \frac{\tau_{ij}^{\alpha} \left(\prod_{m=1}^{M} \eta_{ij,m}^{\lambda_{m,k}} \right)^{\beta}}{\sum_{l \in \mathcal{N}_{i,k}} \tau_{il}^{\alpha} \left(\prod_{m=1}^{M} \eta_{il,m}^{\lambda_{m,k}} \right)^{\beta}}$$

$\mathcal{N}_{i,k}$ = neighbors of i visitable by the k-th ant

Multiple Ant Colony System (MACS) *(Barán, 2003)*

The update rule becomes:

$$p_{ij,m}^{(k)}(t) = \frac{\tau_{ij}^{\alpha}\left(\prod_{m=1}^{M}\eta_{ij,m}^{\lambda_{m,k}}\right)^{\beta}}{\sum_{l\in\mathcal{N}_{i,k}}\tau_{il}^{\alpha}\left(\prod_{m=1}^{M}\eta_{il,m}^{\lambda_{m,k}}\right)^{\beta}}$$

$\mathcal{N}_{i,k}$ = neighbors of i visitable by the k-th ant

The pheromone to be distributed at each iteration is given by: $\tau_0' = \frac{1}{\prod_{i=1}^{M}\langle f_m\rangle_{rank=1}}$

If $\tau_0' > \tau_0$ → set $\tau_0 = \tau_0'$ and re-initialize possible trails using τ_0

Else → perform global updating using

$$\tau_{ij} \leftarrow (1-\rho)\,\tau_{ij} + \rho\Delta_k \quad \forall(i,j) \in k \in \text{Paths}_{rank=1} \quad \text{and} \quad \Delta_k = \frac{1}{\prod_{i=1}^{M} f_m^{(k)}}$$

Multiple Ant Colony System (MACS) *(Barán, 2003)*

Pseudocode

t=0 #generation counter
(V, E) = problem Graph
η, ρ = heuristic tensor, evaporation rate
$\tau_0 =$ Estimate$(f_1, \ldots, f_M)$ *estimate initial pheromone value*
λ, v_0 = generate_random_starting_point_and_weights(V)
τ = InitializePheromoneMatrix(τ_0)
NonDominatedPaths=[ ]
While (t<G) or not (termination_criterion):
    For each $k \in [0,C]$: *ant in the colony*
        Path = CreatePath_and(τ, η, λ, v_0)
        If not dominates(NonDominatedPaths,Path):
            NonDominatedPaths.append(Path)
            NonDominatedPaths = non_dominated_solutions(NonDominatedPaths)
    τ_0'= EvaluateNonDominatedPheromone(NonDominatedPaths)
    If $\tau_0' > \tau_0$:
        $\tau_0 \leftarrow \tau_0'$
        τ = InitializePheromoneMatrix(τ_0)
    Else:
        τ = PheromoneUpdateOnNonDominatedPaths(τ, η, Improved_Paths)
    t=t+1

Pareto ACO (P-ACO) *(Doerner, 2004)*

Multiple pheromone matrices: $\tau_{ij,m} \quad m = 1,\ldots,M$

Multiple cost matrices: $\eta_{ij,m} \quad m = 1,\ldots,M$

Each ant used a different normalized weight:

$$\overrightarrow{\lambda}_k = (\lambda_{1,k}\ldots\lambda_{M,k}) \qquad k = 1,\ldots,C$$

The update rule becomes:

$$p_{ij,m}^{(k)}(t) = \frac{\sum_{m=1}^{M}\left(\tau_{ij,m}^{\alpha}\eta_{ij,m}^{\beta}\lambda_{m,k}\right)}{\sum_{l\in\mathcal{N}_{i,k}}\sum_{m=1}^{M}\left(\tau_{il,m}^{\alpha}\eta_{il,m}^{\beta}\lambda_{m,k}\right)}$$

$\mathcal{N}_{i,k}$ = neighbors of i visitable by the k-th ant

Pareto ACO (P-ACO) *(Doerner, 2004)*

The update rule becomes:

$$p_{ij,m}^{(k)}(t) = \frac{\sum_{m=1}^{M} \left(\tau_{ij,m}^{\alpha} \eta_{ij,m}^{\beta} \lambda_{m,k} \right)}{\sum_{l \in \mathcal{N}_{i,k}} \sum_{m=1}^{M} \left(\tau_{il,m}^{\alpha} \eta_{il,m}^{\beta} \lambda_{m,k} \right)}$$

$\mathcal{N}_{i,k}$ = neighbors of i visitable by the k-th ant

A local pheromone update ensures diversification $\tau_{ij,m} \leftarrow (1-\rho)\ \tau_{ij,m} + \rho\ \tau_0$

$$(1-\rho)\ \tau_{ij,m} + \rho\ \tau_0 \leq \tau_{ij,m} \implies \tau_{ij,m} \geq \tau_0$$

Decrease probability if famous and already visited

A globe pheromone update ensures to highlight rank 1 and 2 solutions $\tau_{ij,m} \leftarrow (1-\rho)\ \tau_{ij,m} + \rho\ \Delta\tau_{ij,m}$

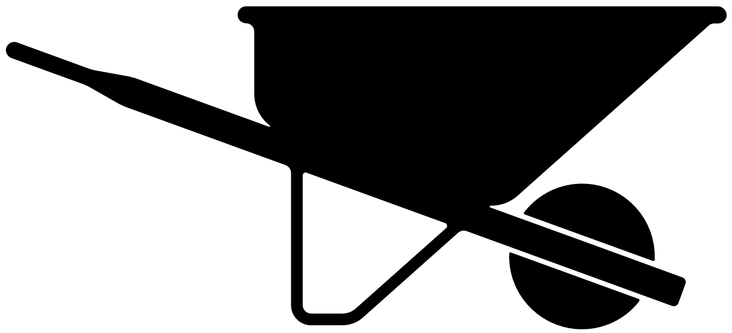
The best empirical value: $\Delta\tau_{ij,m} = \tau_0\ (rank = 1), \frac{\tau_0}{2}\ (rank = 2)$

Pareto ACO (P-ACO) *(Doerner, 2004)*

Pseudocode

t=0 #generation counter
(V, E) = problem Graph
η = heuristic tensor
ρ = evaporation rate
λ, v_0 = generate_random_starting_point_and_weights(V)
$\tau_0 =$ Estimate$(f_1, \ldots, f_M)$ *estimate initial pheromone value*
τ = InitializePheromoneTensor( )
While (t<G) or not (termination_criterion):
    Paths=[ ]
    For each $k \in [0,C]$: *ant in the colony*
        Path, τ = CreatePath_and_LocalPheromoneUpdate(τ, η, λ, v_0)
        Paths.append(Path)
    Improved_Paths = Improve_solutions(Paths)
    Rank1_Paths, Rank2_Paths = SelectBestImprovedSolutions(Improved_Paths)
    τ = GlobalPheromoneUpdate(τ, η, Improved_Paths)
    λ, v_0 = generate_random_starting_point_and_weights(V)
    t=t+1

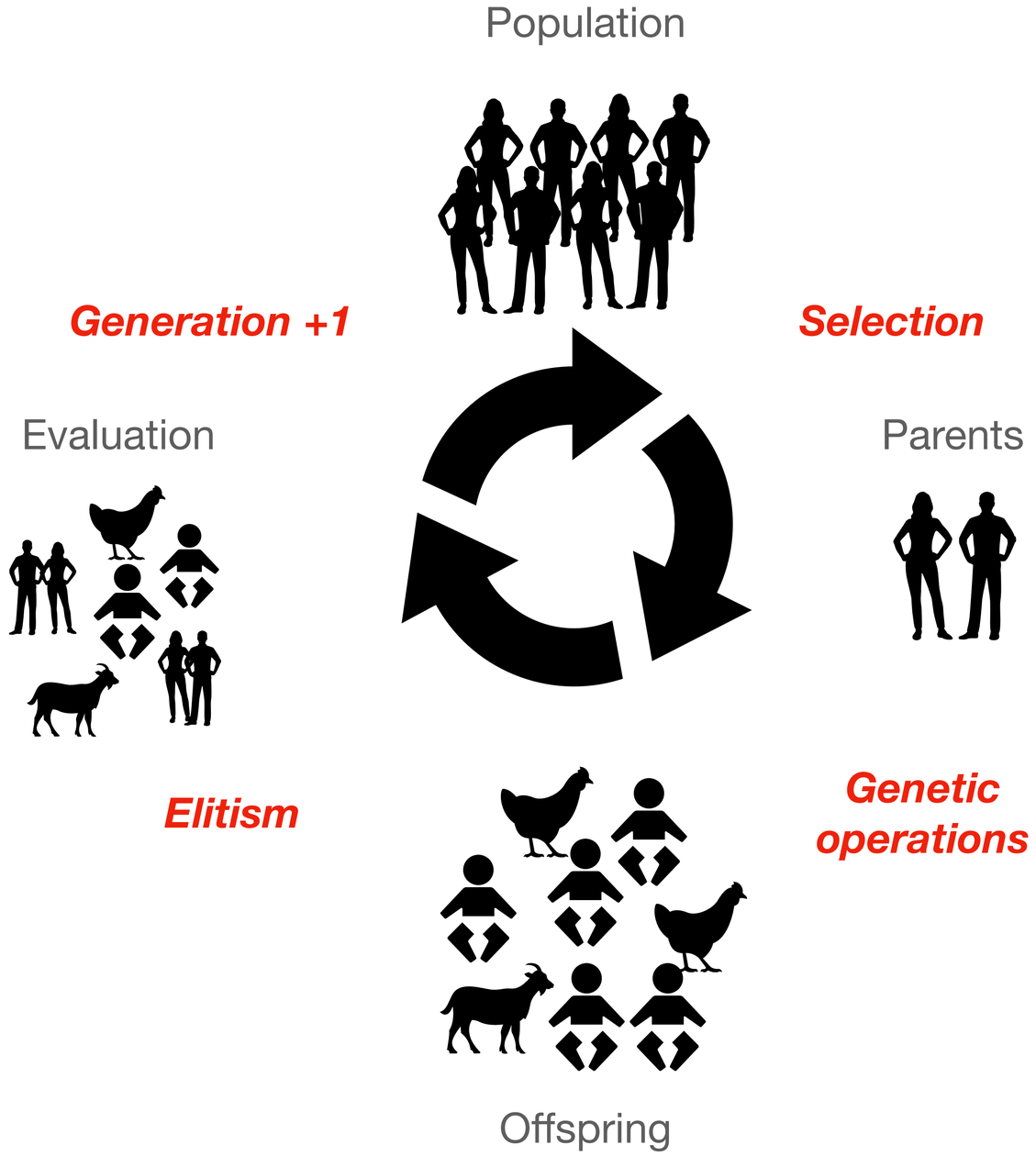
Hands on



Genetic algorithm

Nomenclature and outline

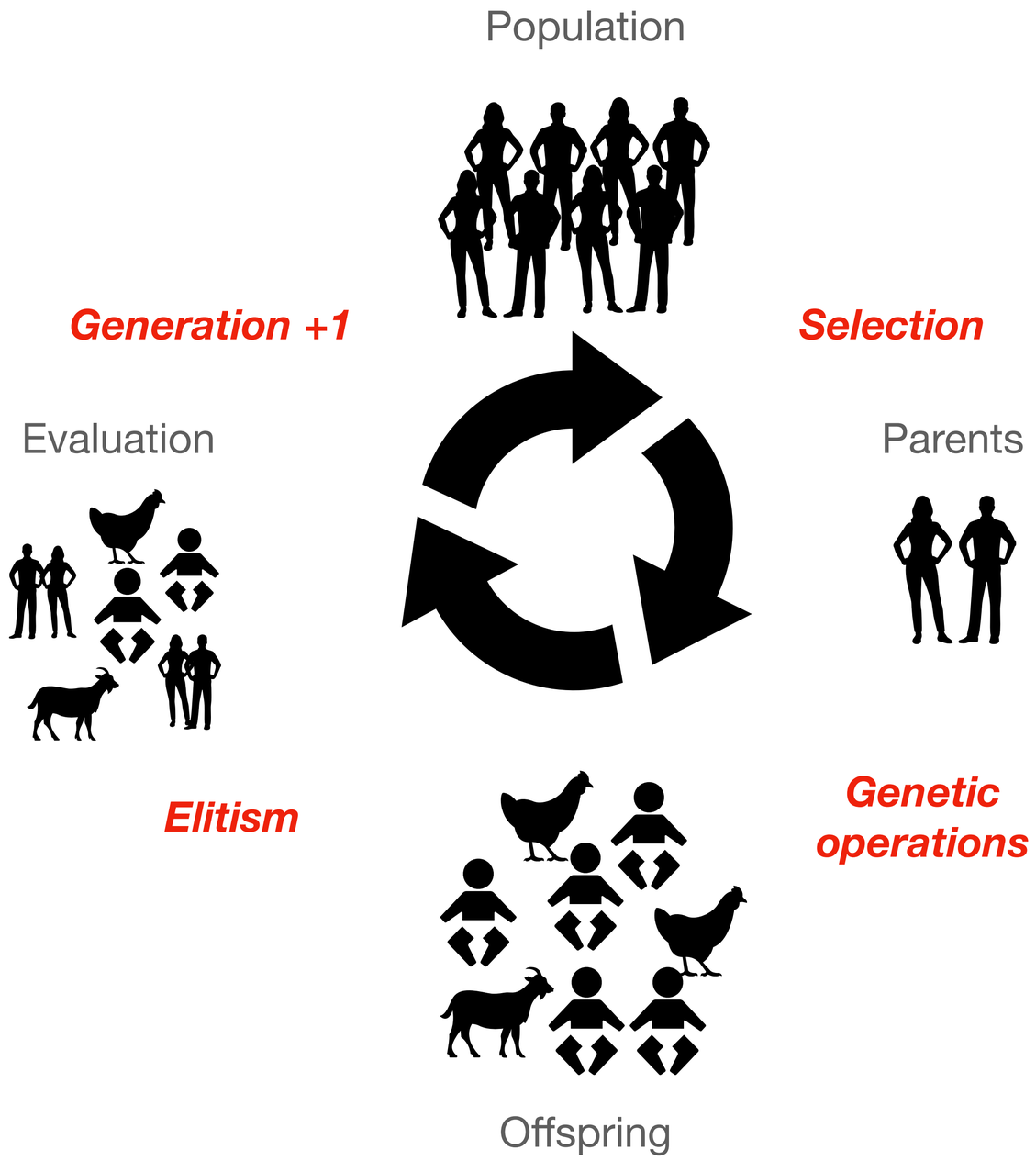
- Individual: feasible solution x
- Population: set of individuals
- Generation: iteration counter
- Genetic operation: operation acting on parents (selected individuals) and outputting children.



Genetic algorithm

Nomenclature and outline

- Selection:
  - random
  - random + tournament
- Genetic operations
  - Mutation
  - Crossover
  - Tailored operations
- Elitism/survival
  - define “better” based on objective



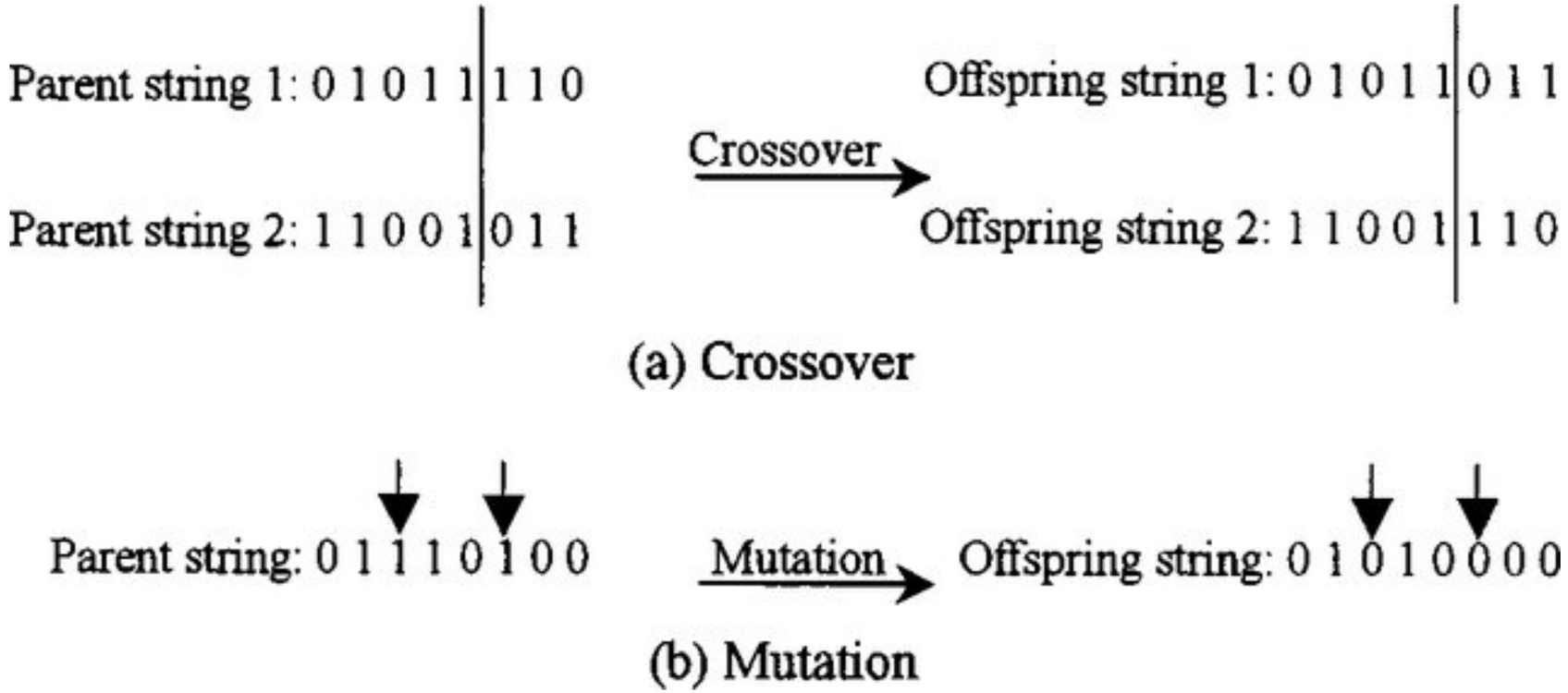
What is tournament selection?

- Define tournament size T
- Select T individuals from the population at random
- Compare their performance so as to keep only one of them
- The individual (or its variations) are allowed into the next generation

- The higher T the higher the selection pressure

What are genetic operations?

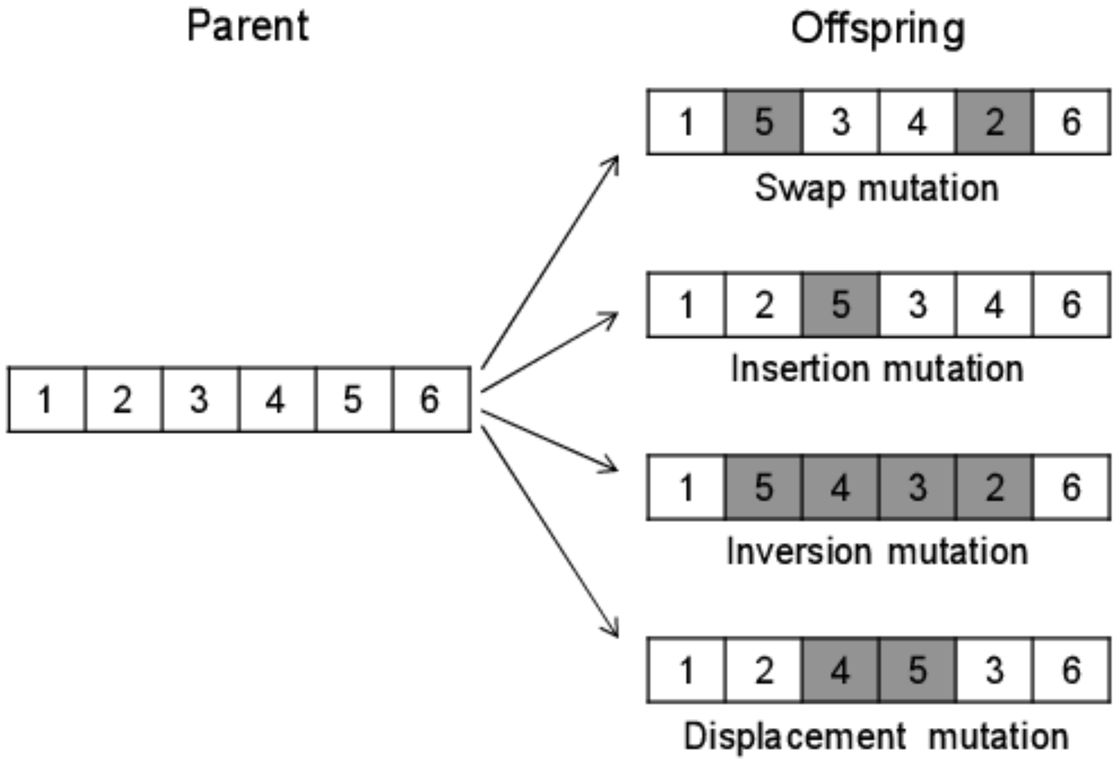
Classical examples

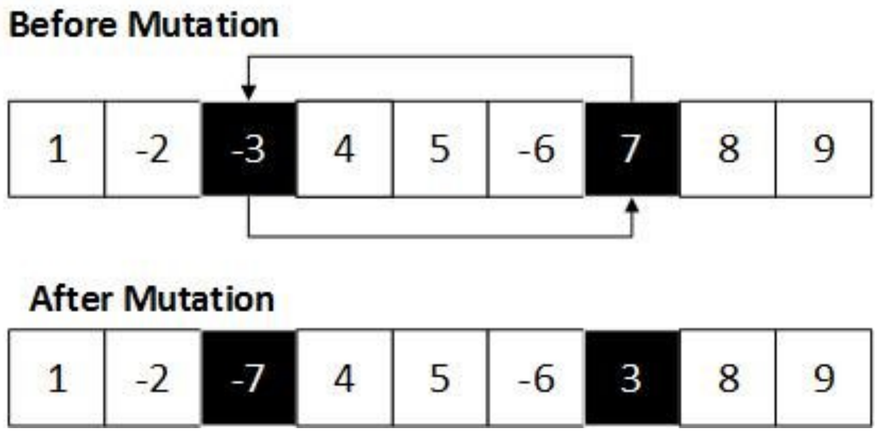


(a) Crossover

(b) Mutation

What are genetic operations?



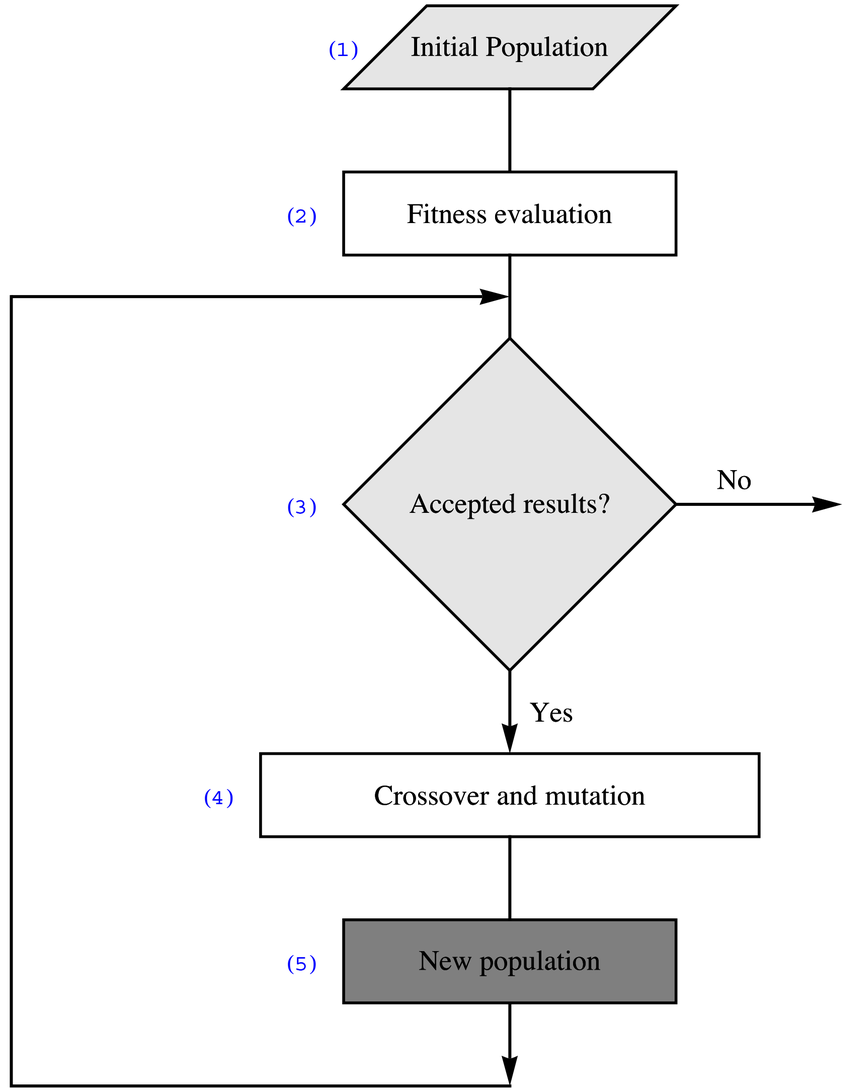


Genetic algorithm

Pseudocode

```
g=0 #generation counter
P(g) = generate_random_population(N)
Fitness(g) = evaluate_population_fitness(P(g))
While (g<G) or not (termination_criterion):
        g=g+1
        offspring= [ ]
        For i=1,…,N:
                variation = select([mutation,crossover])
                parents = select_parents(P(g), Fitness(g),variation)
                child = variation(parents)
                offspring.append(child)
        P(g) = [P(g-1) , offspring]
        Fitness(g) = [Fitness(g-1) , evaluate_population_fitness(offspring)]

        P(g), Fitness(g) = select_best_individuals(P(g), Fitness(g),N)
        termination_criterium = evaluate_termination_criterium(P(g), Fitness(g))
return P(g), Fitness(g)
```



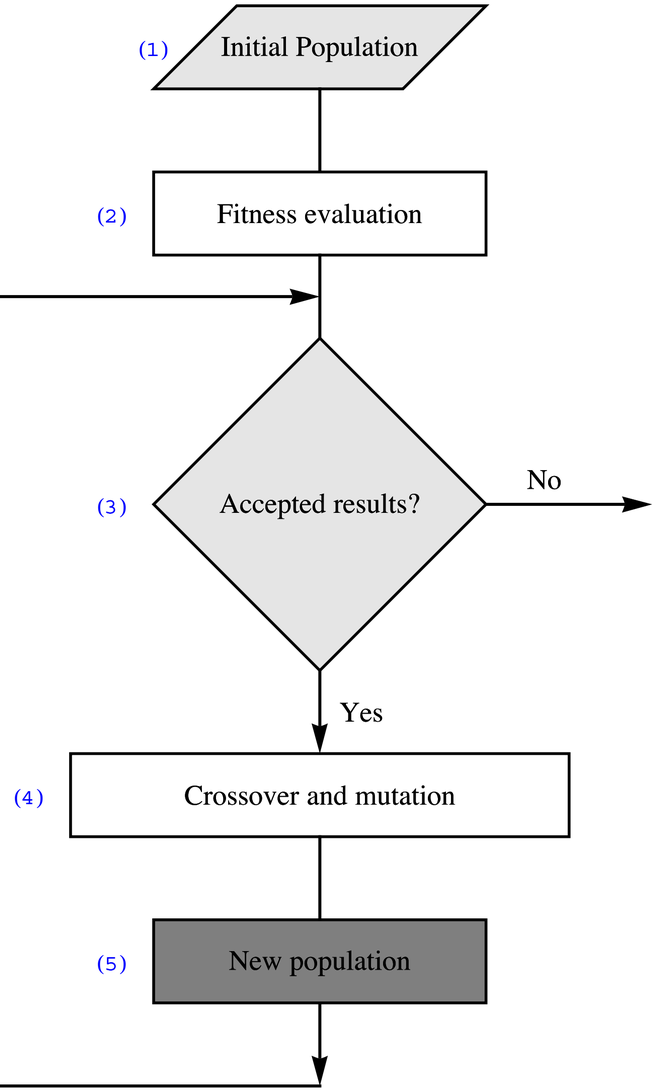
Genetic algorithm

Pseudocode

```
g=0 #generation counter
P(g) = generate_random_population(N)
Fitness(g) = evaluate_population_fitness(P(g))
While (g<G) or not (termination_criterion):
        g=g+1
        offspring= [ ]
        For i=1,…,N:
                variation = select([mutation,crossover])
                parents = select_parents(P(g), Fitness(g),variation)
                child = variation(parents)
                offspring.append(child)
        P(g) = [P(g-1) , offspring]
        Fitness(g) = [Fitness(g-1) , evaluate_population_fitness(offspring)]

        P(g), Fitness(g) = select_best_individuals(P(g), Fitness(g),N)
        termination_criterium = evaluate_termination_criterium(P(g), Fitness(g))
return P(g), Fitness(g)
```

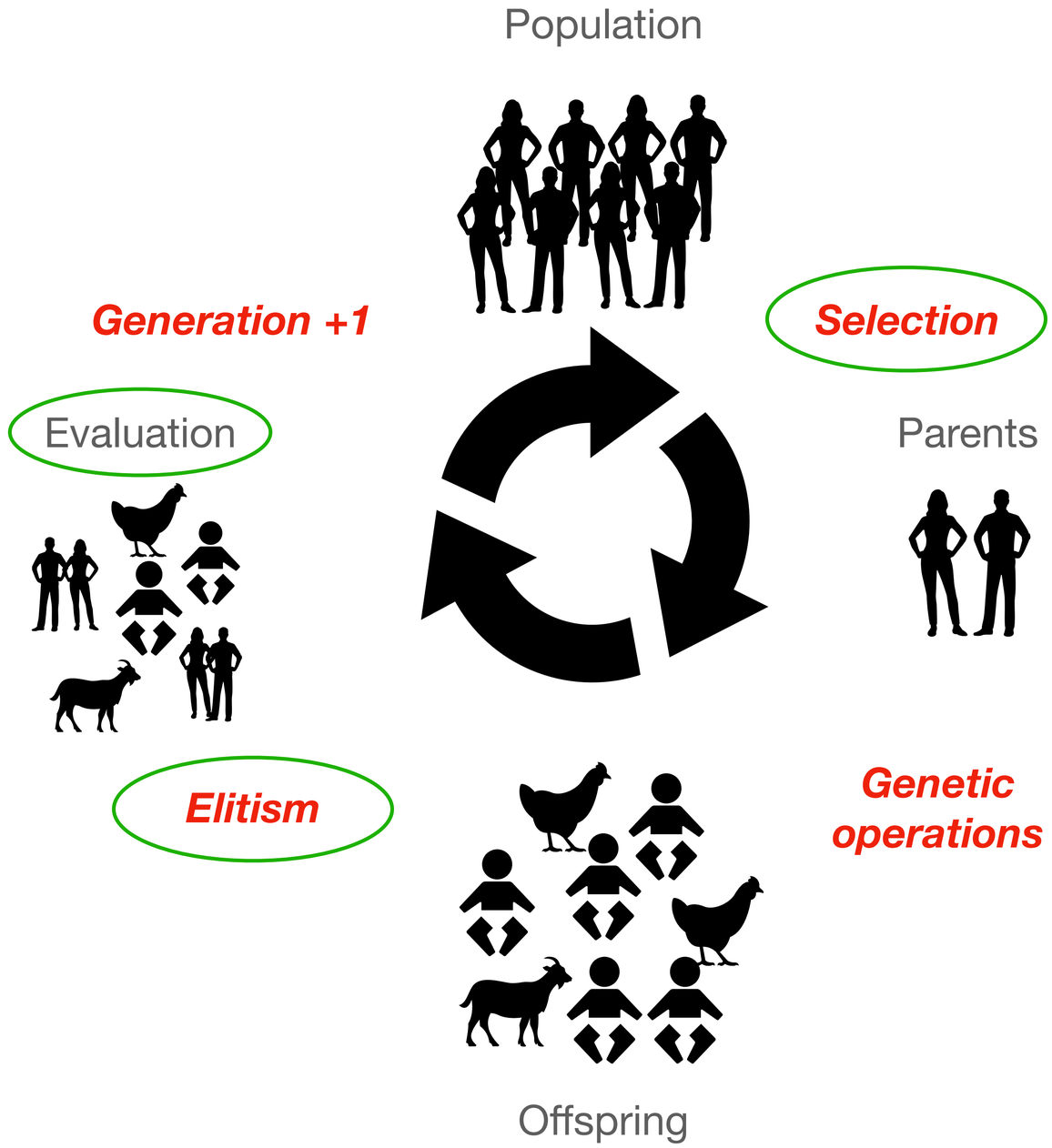
There exist variants contemplating that only a fraction of selected individuals are subject to genetic mutations!



MO Genetic algorithm

Where does MO appear?

- Evaluation:
  - multiple objectives/constraints
- Selection:
  - tournament selection rules
- Elitism/Survival
  - rank
  - rank + density
  - rank + niching

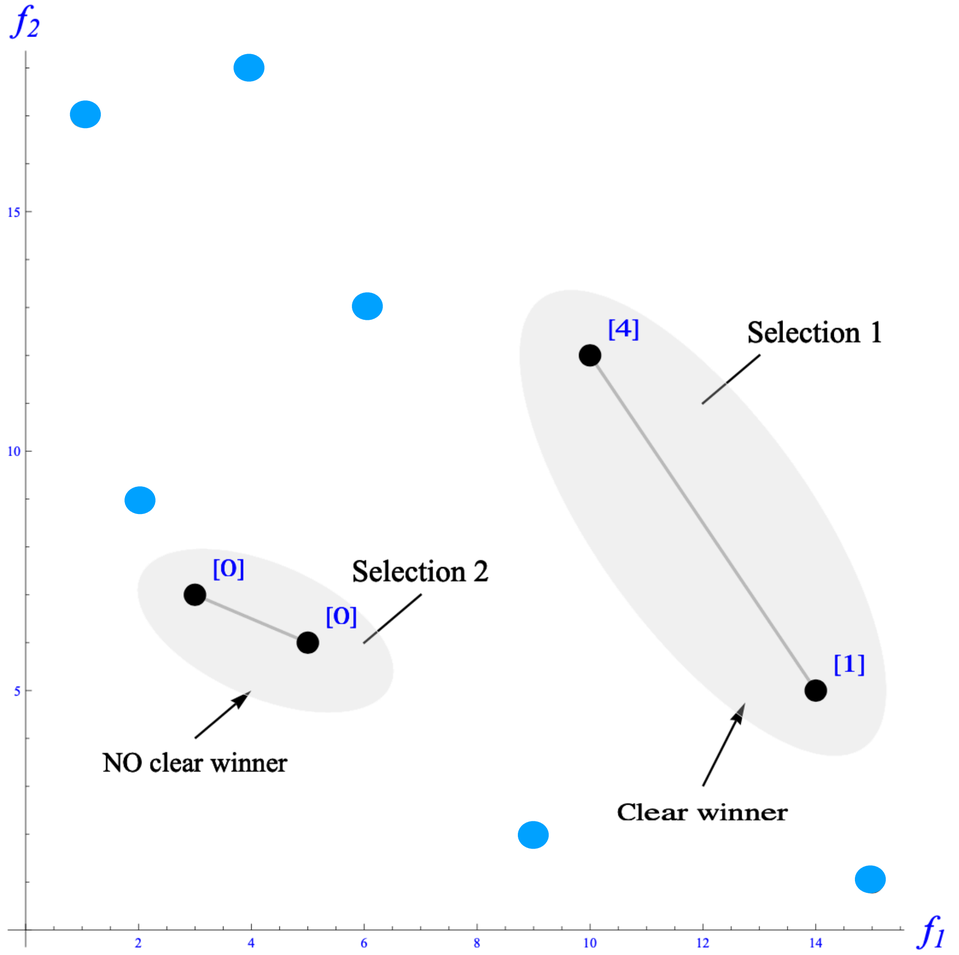


Niched Pareto Genetic Algorithm *(Horn et al. 1994)*

Selection for parenting uses rank and niche count

- Select at random T (2) individuals
- Select at random a comparison set (control over selection pressure)
- Primary criterion: compare ranks (Fonseca/Fleming)
- Secondary criterion: compare counts within a niche radius

The main aim is to preserve diversity and prevent early convergence

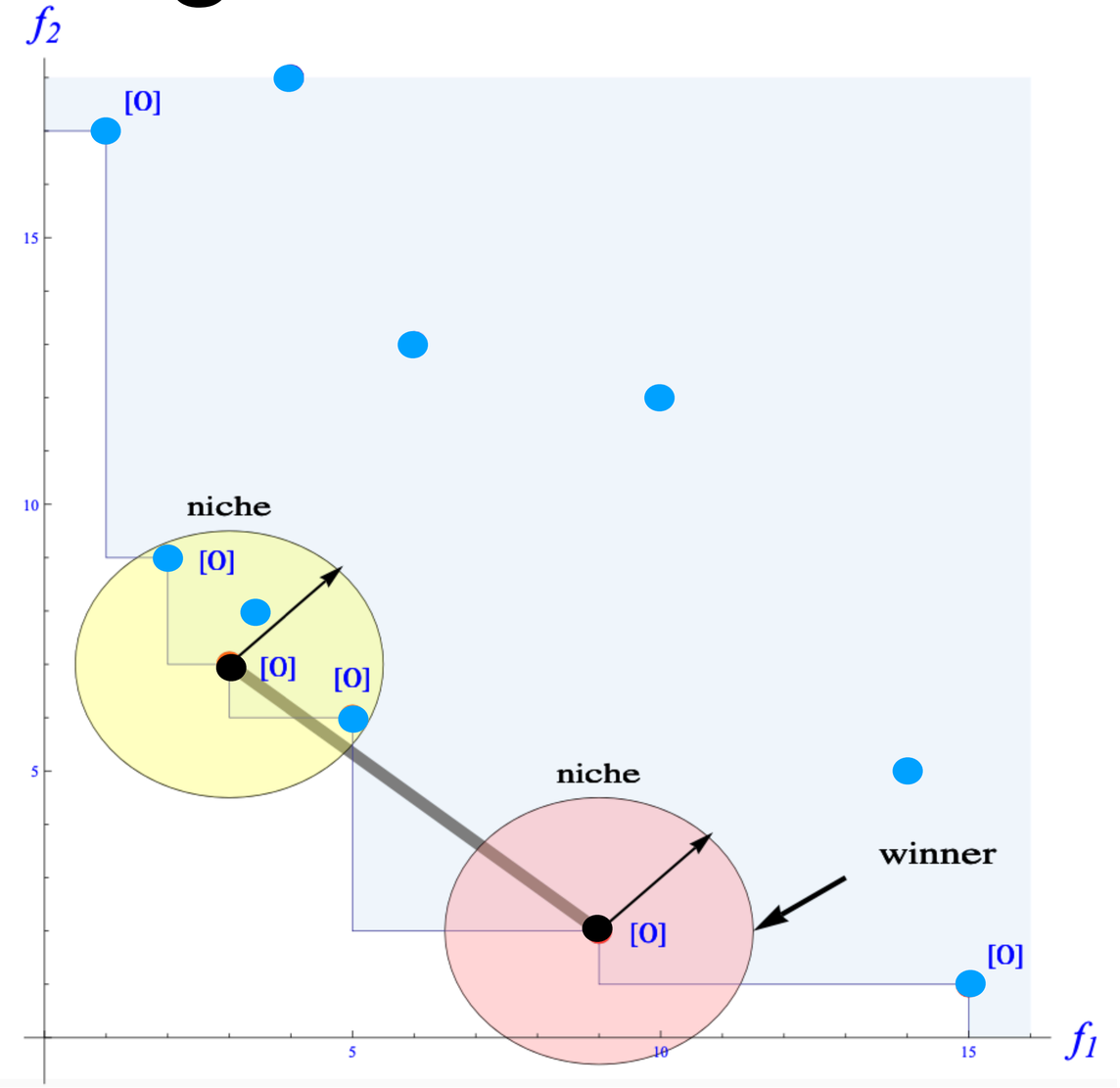


Niched Pareto Genetic Algorithm *(Horn et al. 1994)*

Selection for parenting uses rank and niche count

- Select at random T (2) individuals
- Select at random a comparison set (control over selection pressure)
- Primary criterion: compare ranks (Fonseca/Fleming)
- Secondary criterion: compare counts within a niche radius

The main aim is to preserve diversity and prevent early convergence



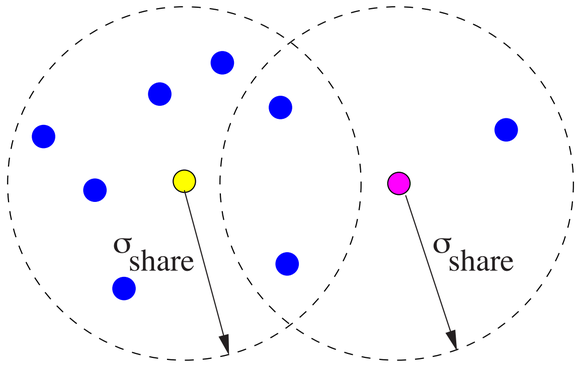
NPGA *(Horn et al. 1994)*

Niche counts and fitness rescaling are computed starting from a sharing function

Niche counting becomes $m_i = \sum_{j=1}^{N} sh(d_{ij})$

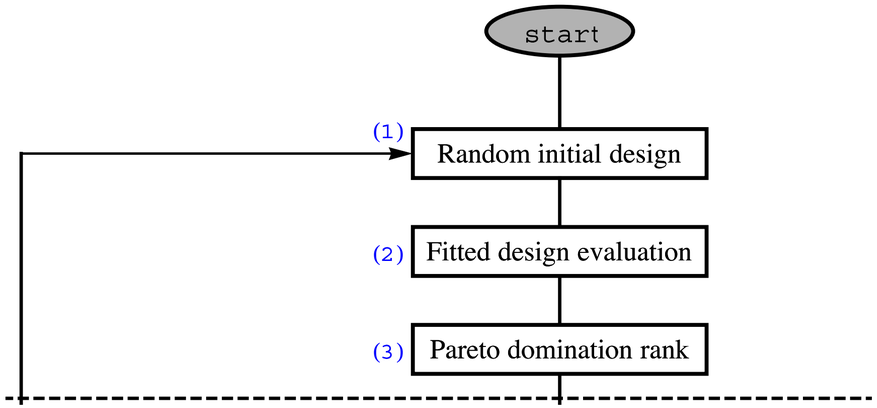
$$sh(d_{ij}) = 1 - \left(\frac{d_{ij}}{\sigma_{share}}\right)^{\alpha} \quad if \quad d_{ij} < \sigma_{share}$$

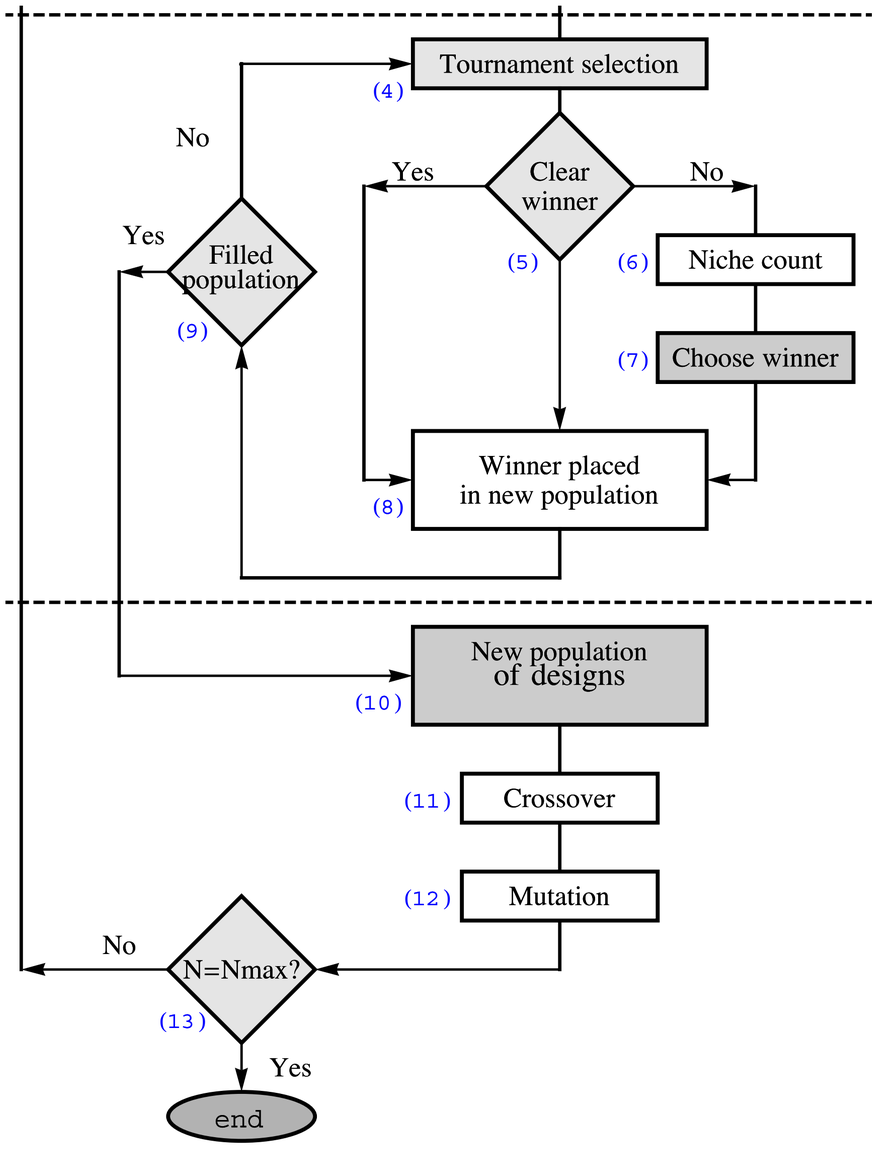
Fitness rescaling $f_i' = \frac{f_i}{m_i}$



“The goal of fitness sharing is to distribute the population over a number of different peaks in the search space, with each peak receiving a fraction of the population proportional to the height of that peak”

NPGA *(Horn et al. 1994)*



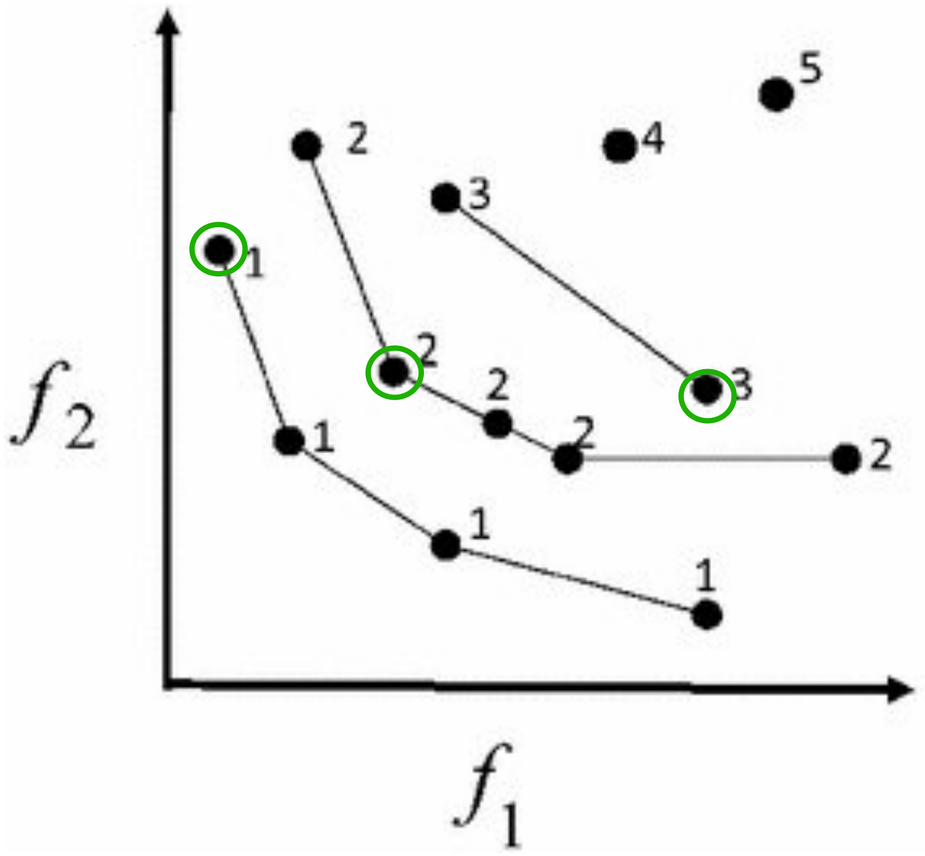


Non-dominated Sorting GA II *(Deb et al. 2002)*

Selection for parenting uses rank and crowding distance.

Select at random T individuals (control over selection pressure)

- Primary criterion: compare ranks (Goldberg)



Non-dominated Sorting GA II *(Deb et al. 2002)*

Selection for parenting uses rank and crowding distance.

Select at random T individuals (control over selection pressure)

- Primary criterion: compare ranks (Goldberg)

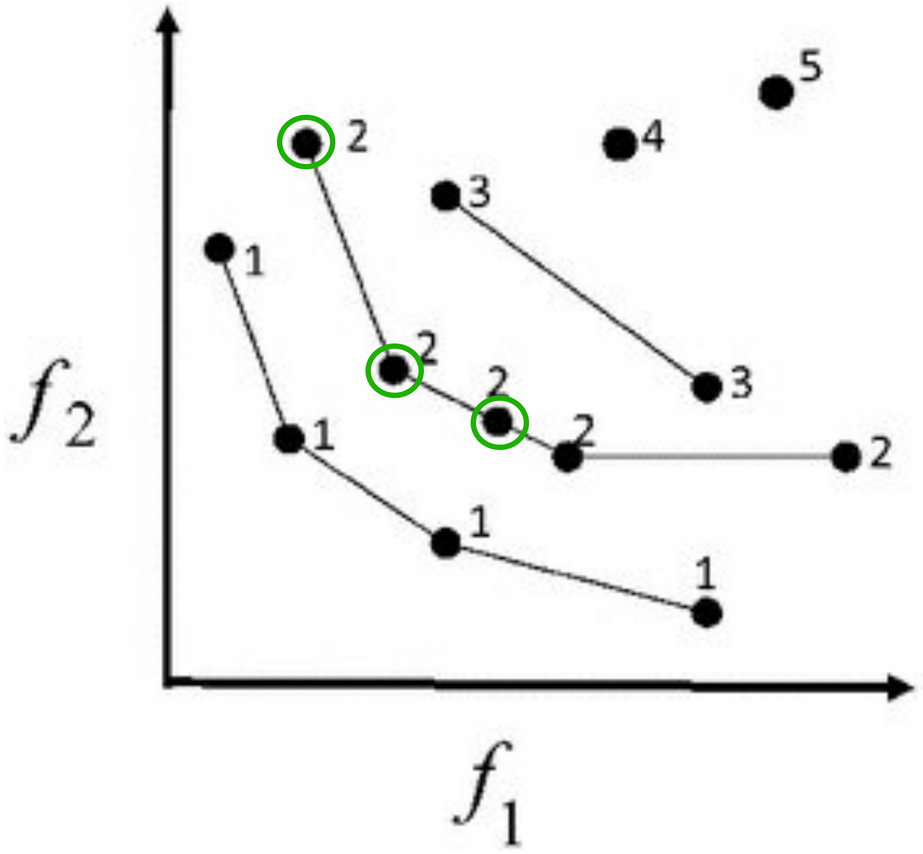
```
fast-non-dominated-sort(P)
for each p ∈ P
   S_p = ∅                <- individuals dominated by p
   n_p = 0                <- number of individuals dominating p
   for each q ∈ P
      if (p ≺ q) then                    If p dominates q
         S_p = S_p ∪ {q}                 Add q to the set of solutions dominated by p
      else if (q ≺ p) then
         n_p = n_p + 1                   Increment the domination counter of p
   if n_p = 0 then                       p belongs to the first front
      p_rank = 1
      F_1 = F_1 ∪ {p}
i = 1                                    Initialize the front counter
while F_i ≠ ∅
   Q = ∅                                 Used to store the members of the next front
   for each p ∈ F_i        <- for each element belonging to the current Pareto front
      for each q ∈ S_p     <- for each element dominated by p
         n_q = n_q − 1     <- decrease its domination number
         if n_q = 0 then                 q belongs to the next front
            q_rank = i + 1
            Q = Q ∪ {q}
   i = i + 1
   F_i = Q
```

Non-dominated Sorting GA II *(Deb et al. 2002)*

Selection for parenting uses rank and crowding distance.

Select at random T individuals (control over selection pressure)

- Primary criterion: compare ranks (Goldberg)



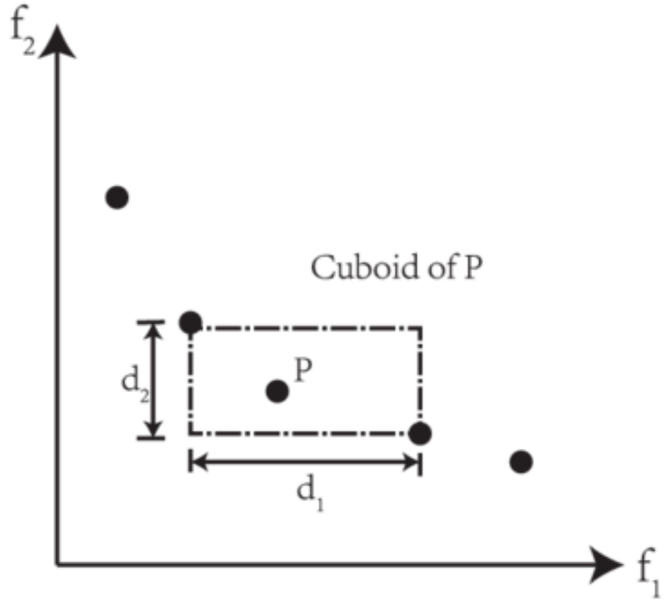
Non-dominated Sorting GA II *(Deb et al. 2002)*

Selection for parenting uses rank and crowding distance.

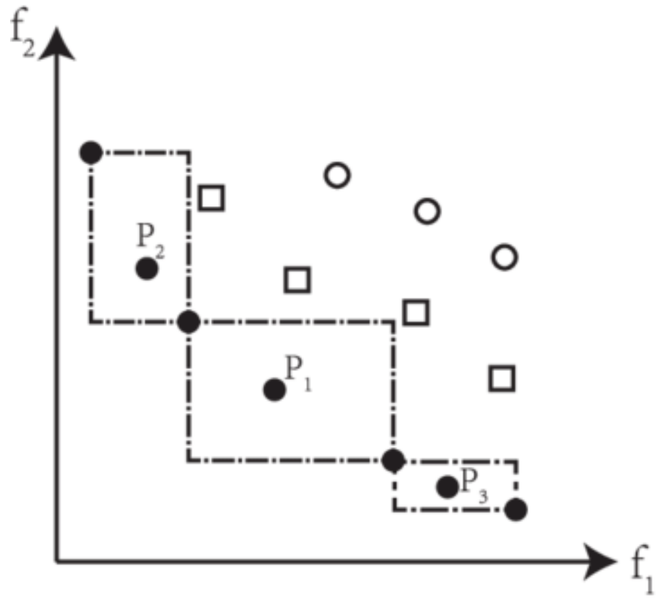
Select at random T individuals (control over selection pressure)

- Primary criterion: compare ranks (Goldberg)
- Secondary criterion: compare crowding distance

The main aim is to preserve diversity and prevent early convergence



The crowding distance is defnined as the sum of the edge length of the cuboid.



● Pareto rank 1 □ Pareto rank 2 ○ Pareto rank 3

For points of the same Pareto rank, sort with the crowding distance. i.e. $P_1 > P_2 > P_3$
Larger distance means better diversity.

`crowding-distance-assignment`$(\mathcal{I})$

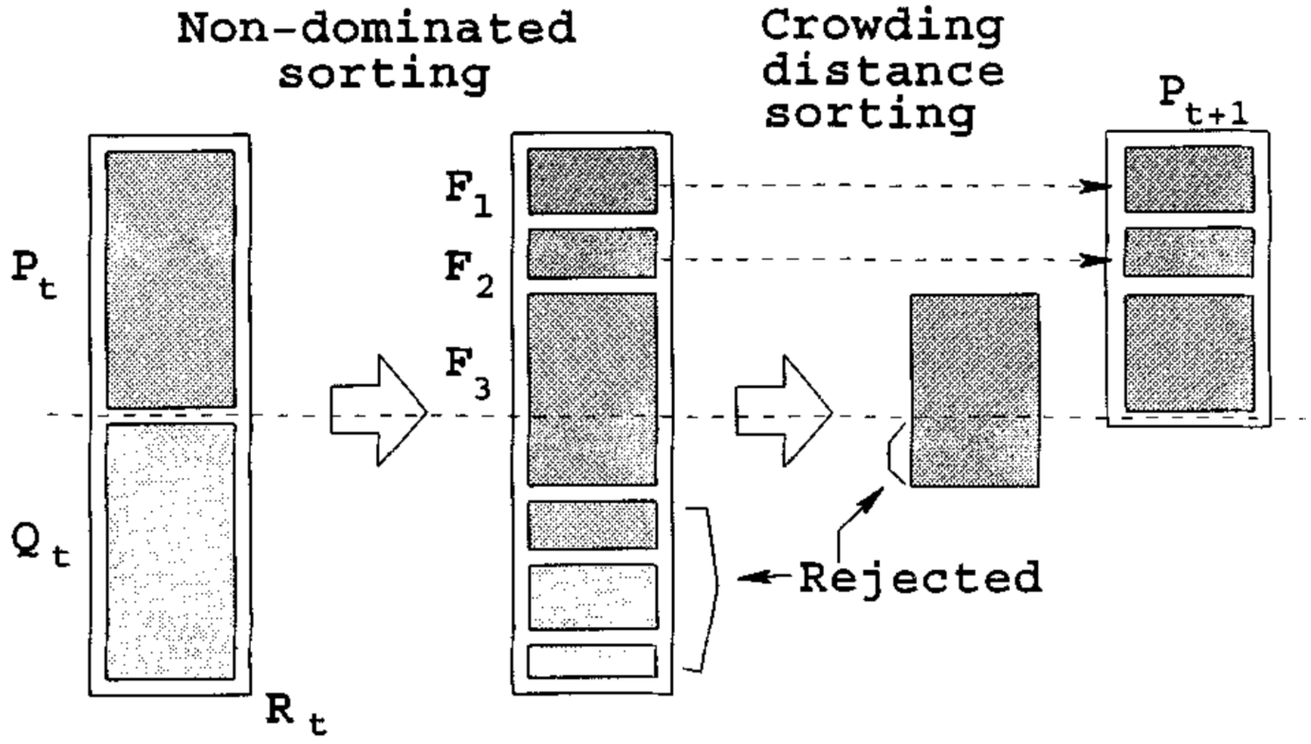
$l = \lvert\mathcal{I}\rvert$	number of solutions in $\mathcal{I}$
for each i, set $\mathcal{I}[i]_{\text{distance}} = 0$	initialize distance
for each objective m	
$\mathcal{I} = \text{sort}(\mathcal{I}, m)$	sort using each objective value
$\mathcal{I}[1]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} = \infty$	so that boundary points are always selected
for $i = 2$ to $(l-1)$	for all other points
$\mathcal{I}[i]_{\text{distance}} = \mathcal{I}[i]_{\text{distance}} + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$	

NSGA-II *(Deb et al. 2002)*

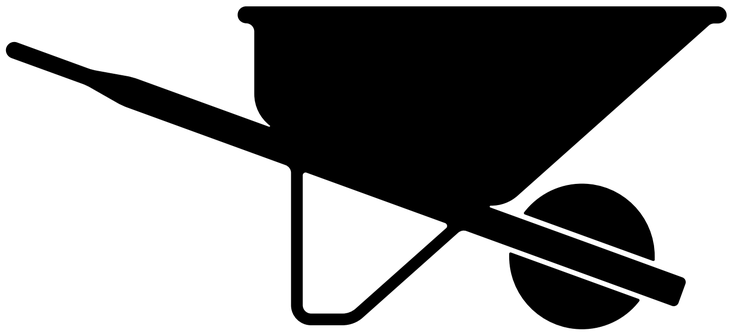
```
t=0 #generation counter
P(t) = generate_random_population(N)
Fitness(t) = evaluate_population_fitness(P(t))
Ranks = ComputeRanks(P(t), Fitness(t))
CrowdingDist = ComputeCrowdingDist(P(t) , Fitness(t) ,Ranks)
While (t<G) or not (termination_criterion):
    Q(t) =[]
    For i=1,…,N:
        variation = select([mutation,crossover])
        parents = select_parents(variation, P(g), Ranks , CrowdingDist,T)
        child = variation(parents)
        Q(t).append(child)
    P(t+1) = [P(t) , Q(t)]
    Fitness(t+1) = [Fitness(t) , evaluate_population_fitness(Q(t))]
    Ranks = ComputeRanks(P(t+1), Fitness(t+1))
    CrowdingDist = ComputeCrowdingDist(P(t+1) , Fitness(t+1) ,Ranks)
    P(t+1), Fitness(t+1), Ranks, CrowdingDist = SelectElite( P(t+1), Fitness(t+1), Ranks, CrowdingDist)
    t=t+1
```

NSGA-II *(Deb et al. 2002)*

Pseuodocode



Hands on



Using the pymoo library
https://pymoo.org/interface/display.html