Evolutionary Algorithms: Ant Colony Optimization and Particle Swarm Optimization

---

Jiří Kubalík
Czech Institute of Informatics, Robotics, and Cybernetics
Czech Technical University in Prague

http://cw.felk.cvut.cz/doku.php/courses/a4m33bia/start

Outline

- **Ant Colony Optimization**
  - original version proposed for discrete optimization problems
  - ACO_R – one of the most popular ACO-based algorithms for continuous domains
- **Particle Swarm Optimization**
  - original version proposed for real-valued parameter optimizations
  - PSO for problems with discrete binary variables

ACO: Basic Idea

Hard problems – no algorithm exists that could solve large instances of these problems to (guaranteed) optimality

- Discrete combinatorial problems

Approximate methods – can find solutions of good quality in reasonable time

- **Local search/optimization** – iteratively improve a complete solution (typically initialized at random) till it reaches some local optimum.
- **Construction algorithms** – build a solution making use of some problem-specific heuristic information.

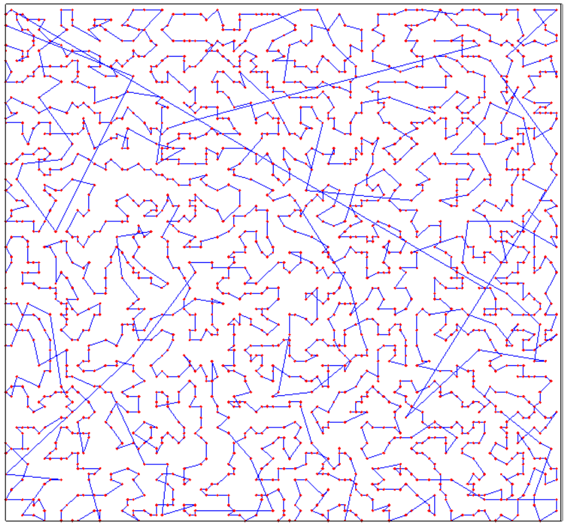
Ant Colony Optimization (ACO) algorithms – **extend** traditional **construction heuristics** with an ability to exploit experience gathered during the optimization process.

Construction Algorithms

Build solutions to a problem under consideration in an incremental way

- starting with an empty initial solution and
- iteratively adding opportunely defined solution components without backtracking
- until a complete solution is obtained.

```
procedure GreedyConstructionHeuristic
  s_p = empty_solution
  while not complete(s_p) do
    e = GreedyComponent(s_p)
    s_p = s_p ⊗ e
  end
  return s_p
end
```



Pros/Cons

(+) fast, solutions of reasonable quality

(–) solution may be far from optimum; generate only limited number of different solutions; decisions made at early stages reduce a set of possible steps at latter stages

Ant Algorithms: Biological Inspiration

Inspired by behavior of real ants living in an ant colony

- Social insects – behave towards survival of the colony
- Simple individual behavior $\times$ complex behavior of the colony

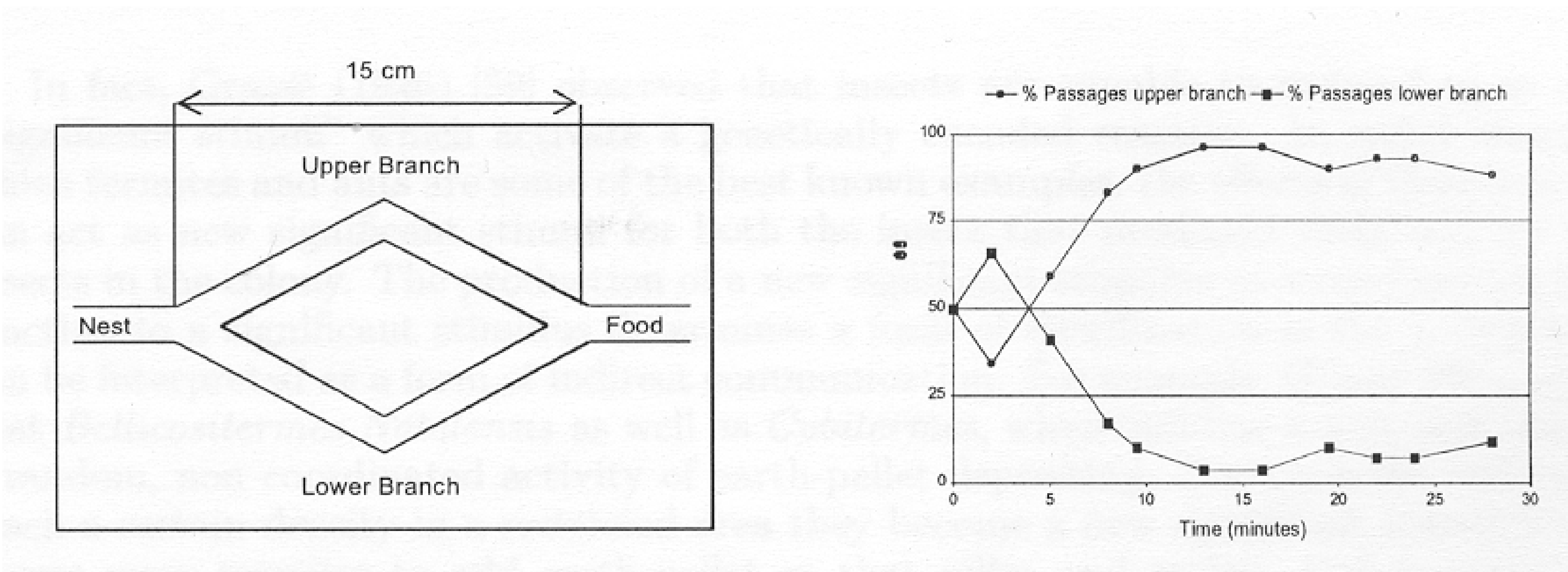
Ability to find the shortest path from the colony to the source of food and back using an **indirect communication via pheromone**

- **Write** — ants lay down pheromone on their way to food
- **Read** – ant detects pheromone (can sense different intensity) laid down by other ants and can choose a direction of the highest concentration of pheromone.
- **Emergence** — this simple behavior applied by the whole colony can lead to emergence of the shortest path.

Experiments with Real Ants

Nest separated from food with a double-bridge

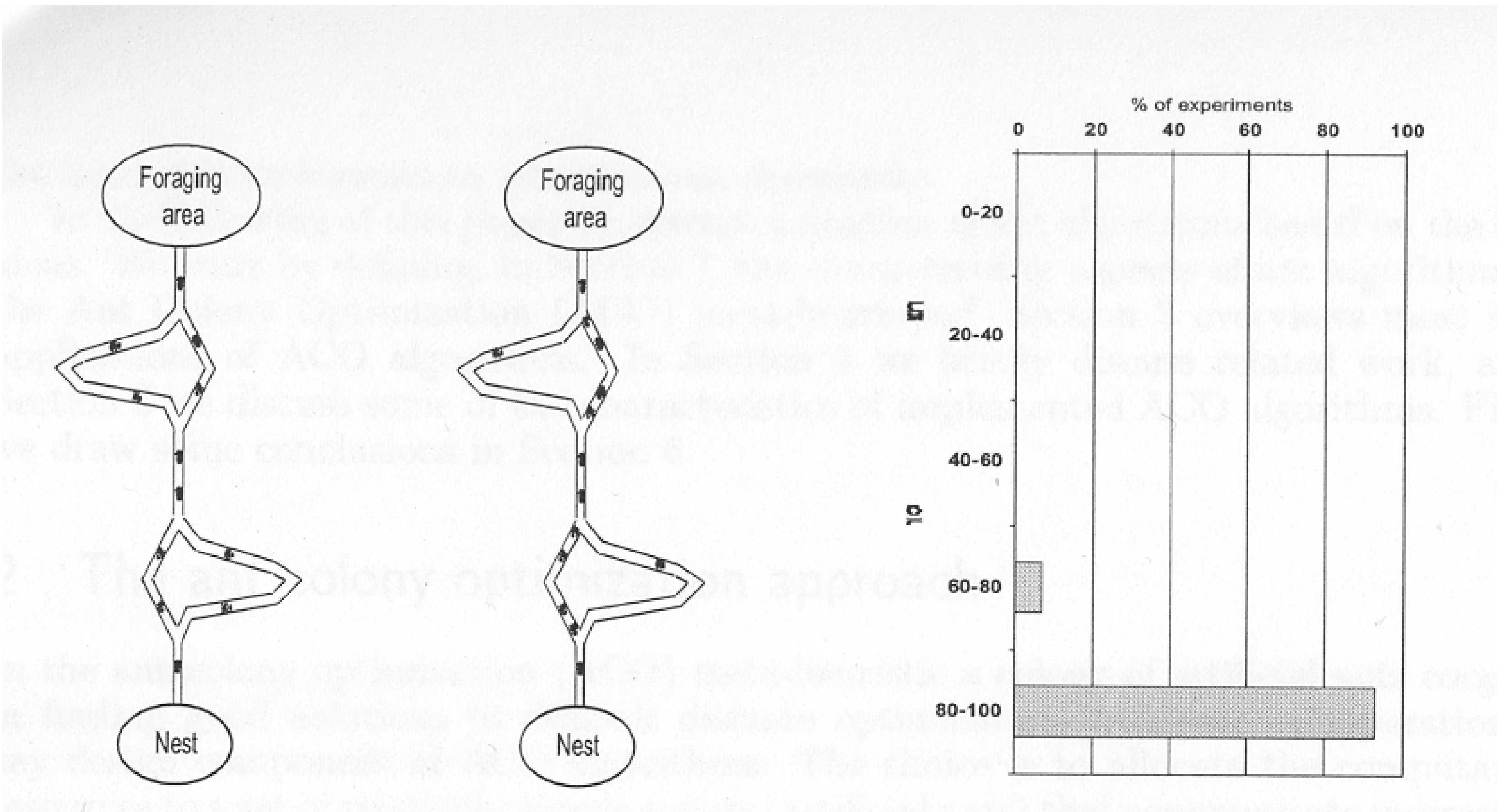
- Both paths have the same length
- At the beginning there is no pheromone
- After some time one of the alternatives gets dominant due to random fluctuations



(a) (b)

Bridges with Different Branches

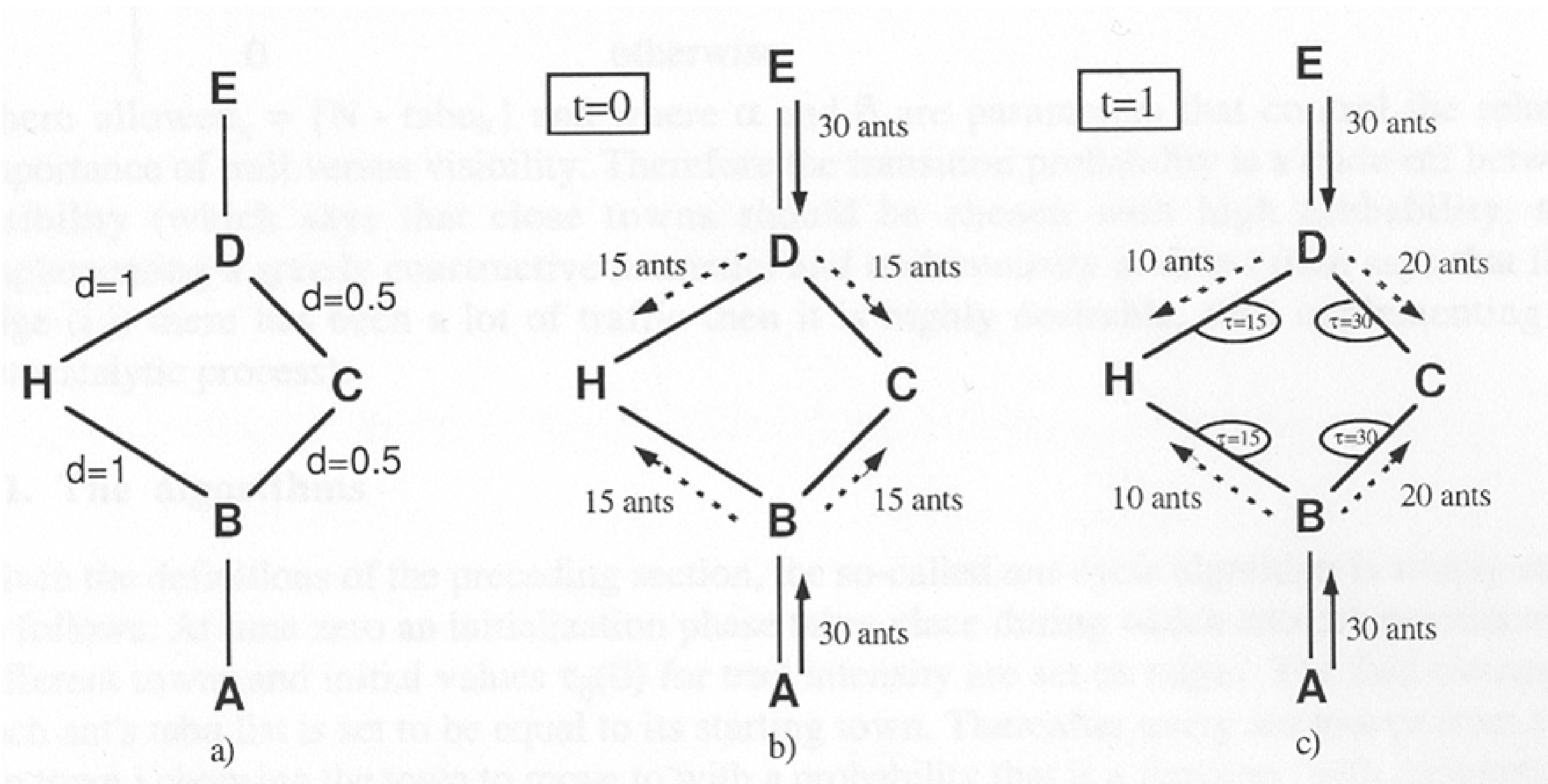
Influence of random fluctuations is significantly reduced and majority of ants go for the shorter path in the end.



Bridges with Different Branches

In each step 30 new ants go from A to B, and 30 ants from E to D

- All ants go with the same speed 1 s^{-1}
- Each ant deposits down 1 unit of pheromone per 1 time unit



Stigmergy

Stigmergy – two individuals interact indirectly when one of them modifies the environment and the other responds to the new environment at a later time.

- **Physically** – by depositing a pheromone the ants modify the place they have visited.
- **Locality of information** – pheromone is “visible” only to ants that are in its close vicinity.
- **Autocatalytic behavior** – the more ants follow a trail, the more attractive that trail becomes for being followed.

  The process is thus characterized by a **positive feedback loop**, where the probability of a discrete path choice increases with the number of times the same path was chosen before.

Pheromone evaporation – realizes forgetting, which prevents premature convergence to sub-optimal solutions.

Real Ants Summary

Real ants characteristics:

- Almost blind
- Use *stigmergic* communication via pheromone trails
- Incapable of achieving complex tasks alone
- Capable of establishing shortest-route paths from their colony to feeding sources and back

Artificial Ants

Similarity with real ants:

- Colony of cooperating ants
- Pheromone trail and stigmergy
- Probabilistic decision making, locality of the strategy
  - Prior information given by the problem specification
  - Local modification of states, induced by preceding ants

Artificial Ants

Similarity with real ants:

- Colony of cooperating ants
- Pheromone trail and stigmergy
- Probabilistic decision making, locality of the strategy
  - Prior information given by the problem specification
  - Local modification of states, induced by preceding ants

Differences from real ants:

- Discrete world
- Inner states – personal memory with already performed actions
- Ants are not completely blind
- Amount of deposited pheromone is a function of the quality of the solution
- Problem dependent timing of depositing the pheromone
- Extras – local optimization, elitism

Ant Colony Optimization Metaheuristic

ACO can be applied to any discrete optimization problem for which some heuristic solution construction mechanism can be conceived.

Artificial ants are stochastic solution construction heuristics that probabilistically build a solution by iteratively adding solution components to partial solutions by taking into account

- **heuristic information** of the problem instance being solved, if available,
- **(artificial) pheromone trails** which change dynamically at run-time to reflect the ants' acquired search experience.

Stochastic component allows generating a large number of different solutions.

General ACO Metaheuristic

```
procedure ACO metaheuristic
    scheduleActivities
        manageAntActivity()
        evaporatePheromone() // forgetting
        daemonActions(){optional} // centralized actions
                                  // local search, elitism
    end scheduleActivities
end ACO metaheuristic
```

Steps for implementing ACO:

- Choose appropriate graph representation
- Define positive feedback
- Choose constructive heuristic
- Choose a model for constraint handling (tabu list in TSP)

Ant System (AS) for the TSP

Problem: Given n cities, the goal is to find the shortest path going through all cities and visiting each exactly once.

- Considers a complete graph.
- d_{ij} is Euclidean distance from city i to city j

Definition

- m is the number of ants
- $\tau_{ij}(t)$ is the intensity of pheromone on the link (i, j) in time t
- η_{ij} is visibility (heuristic information) expressed by $1/d_{ij}$
- $(1 - \rho)$ evaporation factor, $(\rho \in (0, 1))$ is constant through the whole opt. process
- $tabu_k$ is dynamically growing vector of cities that have already been visited by k-th ant
- AS iteration – each ant adds one city to the constructed route
- AS cycle – composed of n iterations during which all ants complete their routes

AS: Pheromone Deposition

1. $\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k & \text{, if } k\text{-th ant used the edge } (i,j) \\ 0 & \text{, otherwise} \end{cases}$
2. $\Delta\tau_{ij} = \sum_k \Delta\tau_{ij}^{k}$
3. $\tau_{ij}(t+n) = (1-\rho)\cdot\tau_{ij}(t) + \rho\cdot\Delta\tau_{ij}$

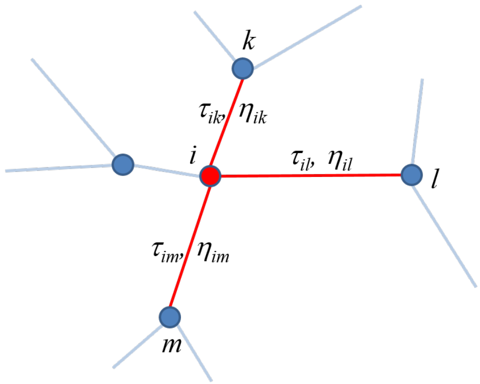
where

- $\Delta\tau_{ij}^{k}$ is the amount of pheromone deposited on the edge (i,j) by k-th ant within a time interval $(t, t+n)$
- Q is a constant
- L_k is the length of the route constructed by k-th ant
- ρ must be smaller than 1, otherwise the pheromone would accumulate unboundedly (recommended is 0.5)
- $\tau_{ij}(0)$ is set to small positive values

AS: Probabilistic Decision Making

At each **construction step**, decisions on which component should be added to the constructed solution given the ant is staying in node i and can move to one of $N - tabu_k$ neighboring nodes, j, are made based on the probabilities assigned to each possible transition (i, j) as

$$p_{ij}^k = \begin{cases} \frac{(\tau_{ij}(t))^\alpha \cdot (\eta_{ij})^\beta}{\sum_{l \in \{N - tabu_k\}} (\tau_{il}(t))^\alpha \cdot (\eta_{il})^\beta} & \text{, if } j \in \{N - tabu_k\} \\ 0 & \text{, otherwise} \end{cases}$$



where α, β define relative importance of the pheromone and the visibility.

Probability p_{ij}^k is a compromise between

- **visibility** that prefers closer cities to more distant ones and
- **intensity of pheromone** that prefers more frequently used edges.

AS: Outline

1. Initialization
   - time: $t = 0$
   - number of cycles: $NC = 0$
   - pheromone: $\tau_{ij} = c$
   - Initial positioning of m ants to n cities
2. Initialization of *tabu* lists
3. Ants' action
   - Each ant iteratively builds its route
   - Calculate length of the routes L_k for all ants $k \in (1, \ldots, m)$
   - Update the shortest route found
   - Calculate $\Delta\tau_{ij}^k$ and update $\tau_{ij}(t+n)$
4. Increment discrete time
   - $t = t + n$, $NC = NC + 1$
5. If($NC < NC_{max}$) then goto step 2

   else stop.

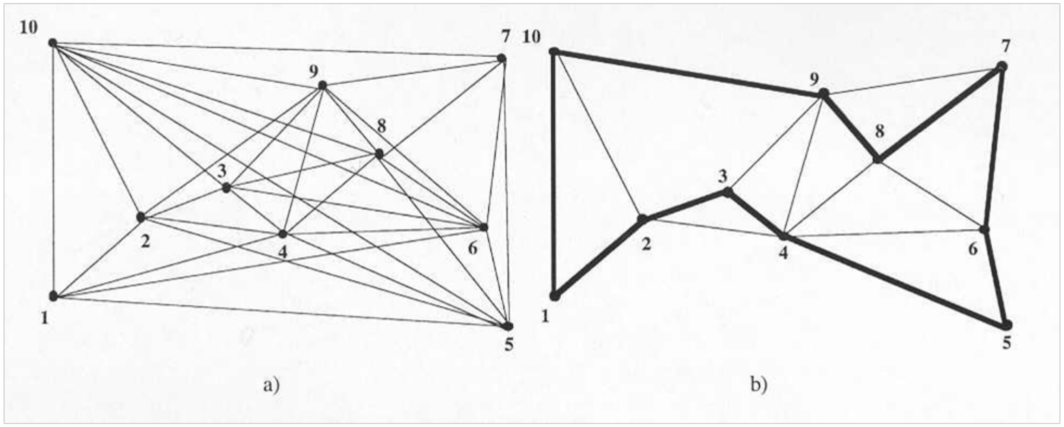
AS: Elitism

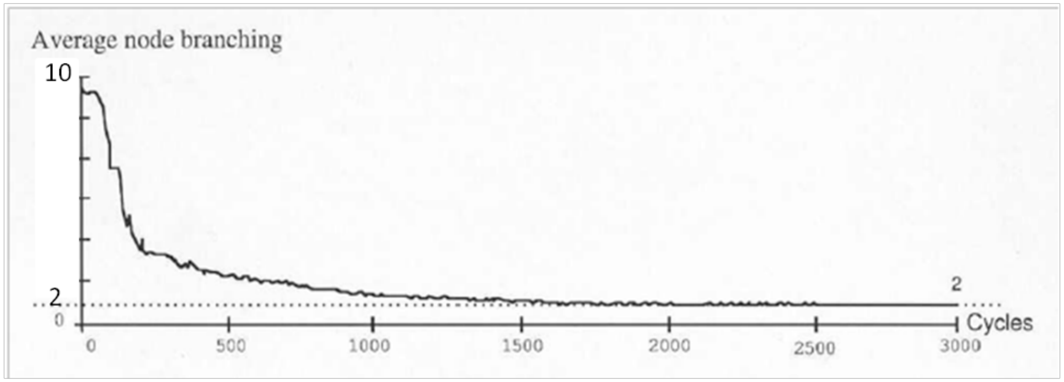
Intensity of pheromone is strengthened on edges that lie on the **shortest path** out of all generated paths

- Amount of added pheromone: $e \cdot Q/L^*$,
  where e is a number of *elite* ants and L^* is the shortest path
- **Beware of premature convergence!**

AS: Evolution of Solution for 10 Cities

After greedily searching the space it is desirable to adapt global information stored in $\tau_{ij}(t)$ (it is necessary to partially forget)





Stagnation – branching factor is 2, all ants go the same way.

Applications of ACO Algorithms

Static problems

- Traveling salesman problem
- Quadratic assignment problem
- Job-shop scheduling problem
- Vehicle routing problem
- Shortest common supersequence problem

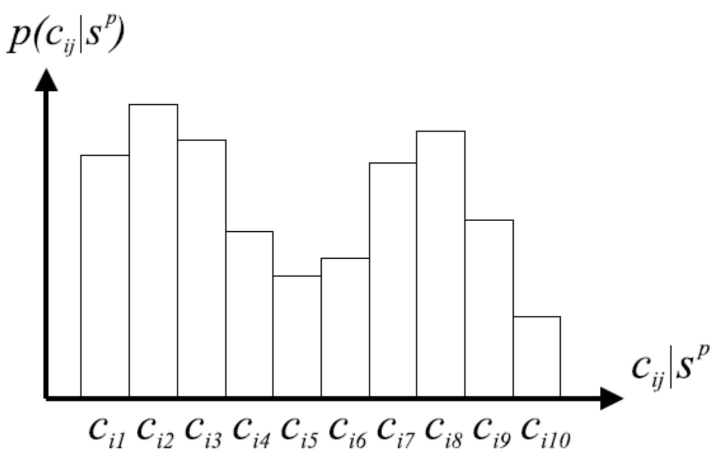
Dynamic problems

- Network routing

ACO for Continuous Domain: ACO_R

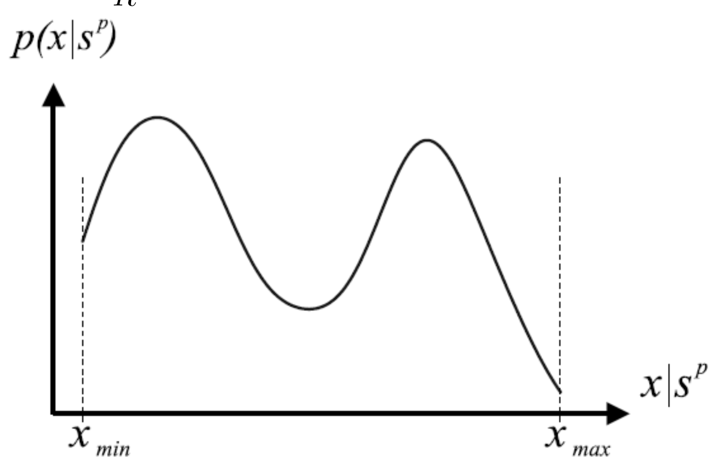
Idea: Instead of using a discrete probability distribution to make a probabilistic choice of the new solution component at each construction step, a **continuous probability density function (PDF) is used** to choose a value for variable X_i at construction step i, for $i = 1, \ldots, n$.

ACO



Discrete probability distribution $P_d(c_{ij}|s^p)$ of a finite set $c_{i1}, \ldots, c_{i10} \in N(s^p)$ of available components.

ACO_R

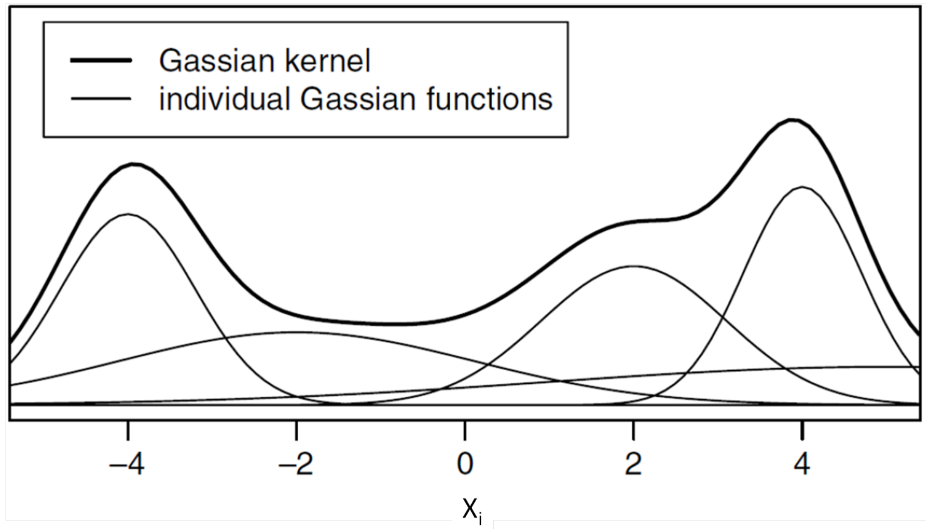


Continuous probability density function $P_c(x|s^p)$ with possible range $x \in [x_{min}, x_{max}]$.

Note that $\sum_{j=1}^{10} p(c_{ij}|s^p) = \int_{x_{min}}^{x_{max}} p(x|s^p)dx = 1$ for given **partial solution** s^p.

ACO_R: Gaussian Kernel PDF

Gaussian kernel PDF – an estimation of a multimodal one-dimensional PDF.



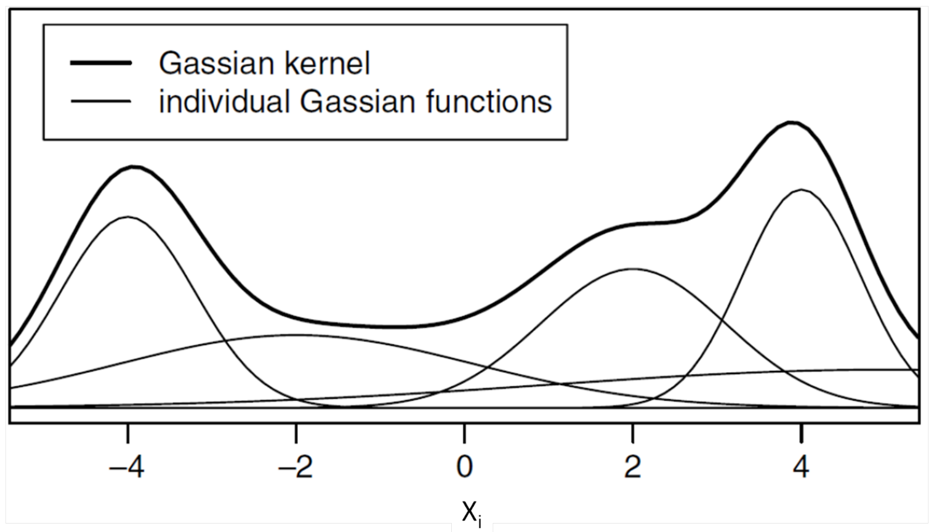
Advantages:

- reasonably easy way of sampling using a uniform random generator in conjunction with, for instance, the Box–Muller method
- it is able to describe a situation where multiple disjoint areas of the search space are promising

ACO_R: Gaussian Kernel PDF

Gaussian kernel for a variable i, $G^i(x)$, **as a weighted sum of k one-dimensional Gaussian functions** $g_l^i(x)$.

$$G^i(x) = \sum_{l=1}^{k} \omega_l g_l^i(x) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i \sqrt{2\pi}} e^{-\frac{(x-\mu_l^i)^2}{2{\sigma_l^i}^2}}$$



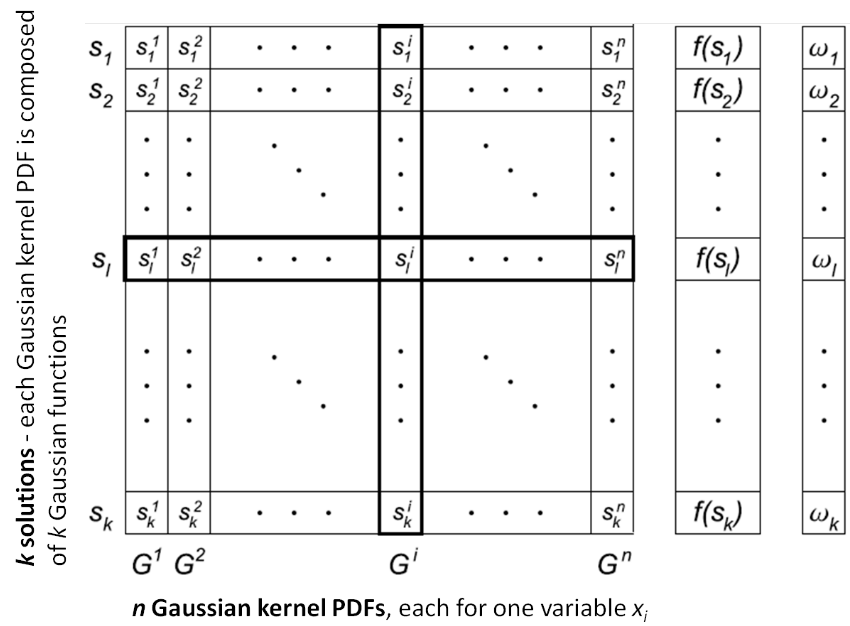
ACO_R: Solution Archive

ACO_R keeps track of a number of good solutions in a **solution archive**, which represents the **pheromone model**. For each solution s_l, values of its n variables, s_l^i, and the objective value $f(s_k)$ are stored.

Parameter vectors of Gaussian kernels

- ω – vector of weights,
- μ^{i} – vector of means,
- σ^{i} – vector of standard deviations

are calculated from k solutions kept in **solution archive**.
Solutions in the archive are sorted according to their rank from the best to the worst one (a solution s_l has rank l).



Solution archive

Note

- $f(s_1) \leq f(s_2) \leq \cdots \leq f(s_l) \leq \cdots \leq f(s_k)$
- $\omega_1 \geq \omega_2 \geq \cdots \geq \omega_l \geq \cdots \geq \omega_k$

ACO_R: Gaussian Kernel Parameters

- **Means** – the values of the ith variable of all the solutions in the archive become the elements of the vector μ^i.

$$\mu^i = \{\mu_1^i, \ldots, \mu_k^i\} = \{s_1^i, \ldots, s_k^i\}$$

- **Weights** – are calculated using a Gaussian function

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}}$$

  with argument l, mean 1.0 and standard deviation qk, where q is a parameter of the algorithm.
  Small $q \rightarrow$ the best-ranked solutions are strongly preferred.
  Large $q \rightarrow$ more uniform weights.

- **Standard deviations** – for a particular Gaussian function g_l^i, the standard deviation σ_l^i is calculated as the average distance from the chosen solution s_l to other solutions in the archive

$$\sigma_l^i = \xi \sum_{e=1}^{k} \frac{s_e^i - s_l^i}{k-1}$$

  The parameter ξ realizes the *pheromone evaporation* – the higher the value of ξ, the less biased is the search towards the solutions stored in the archive.

ACO_R: Solution Generation

Each new solution is generated in n **construction steps**.

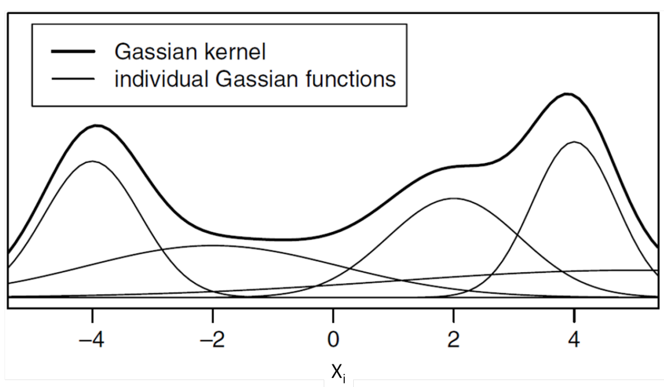
Two-phase sampling

- Select one Gaussian function g_l^i with probability proportional to its weight.

$$p_l = \frac{\omega_l}{\sum_{r=1}^{k} \omega_r}$$

- Sample the chosen Gaussian function g_l^i.

which is equivalent to sampling the Gaussian kernel PDF G_i.



ACO_R: Pheromone Update

Pheromone update

1. Set of newly generated solutions are added to the solution archive.
2. The worst solutions are removed from archive so that the total size of the archive does not change.

ACO_R: Algorithm Outline

Input: k, m, n, q, ξ
Output: The best solution found

```
initialize and evaluate k solutions s_1, ..., s_k
// sort the solutions and store them in the Archive
Archive = Sort(s_1, ..., s_k)
while (termination condition is not reached) do
    // Generate m new solutions
    for l = 1 to m do
        // construct solution
        for i = 1 to n do
            Select Gaussian g_j^i according to weights
            Sample Gaussian g_j^i with parameters μ_j^i, σ_j^i
        end for
        Store and evaluate newly generated solution
    end for
    // Sort solutions and store the best k
    Archive = Best(Sort(s_1, ..., s_{k+m}), k)
end while
```

Particle Swarm Optimization

Inspired by biological and sociological motivations

- Bird flocks
- Fish schools
- Swarms of insects

PSO: Characteristics

Population-based optimization technique – originally designed for solving real-valued function optimizations.

- Applicable for optimizations in rough, discontinuous and multimodal surfaces.
- Suitable for black-box optimizations – does not require any gradient information of the function to be optimized.
- Conceptually very simple.

PSO: Characteristics

Each candidate solution of continuous optimization problem, called a **particle**, is described (encoded) by a real vector N-dimensional search space: $\mathbf{x} = x_1, \ldots, x_n$.

A population of particles, called a **swarm**, is evolved in an iterative process.

A **neighborhood** relation N is defined in the swarm that determines for any two particles P_i and P_j whether they are neighbors or not. Different neighborhood topologies can have different effect on the swarm performance. Often, the whole search space is used as the neighborhood for each particle.

The **particles** change their components and **fly** through the multi-dimensional search space while **interacting** to each other.

Particles calculate their **fitness value as the quality of their actual position** in the search space w.r.t. the optimized function.

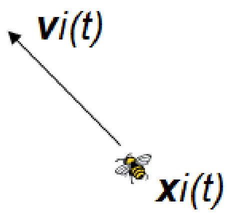
Particles also compare themselves to their neighbors and imitate the best of that neighbors.

PSO: Particle's Position and Velocity

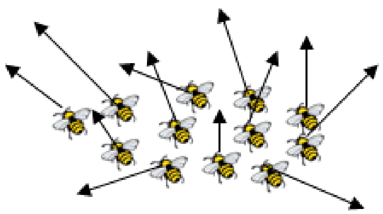
Swarm of particles is flying through the parameter space and searching for the optimum.

Each particle is characterized by

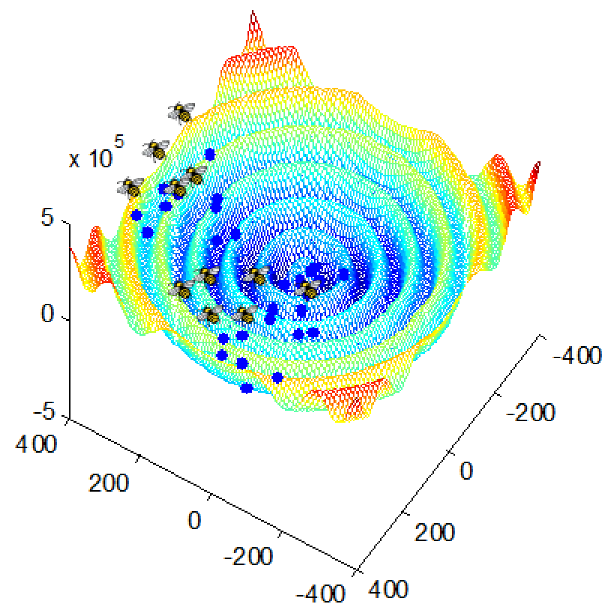
- **Position** vector $x_i(t)$
- **Velocity** vector $v_i(t)$



particle i



swarm



swarm flying over search space

PSO: Velocity Update

Update of the i-th particle velocity:

$$v_i(t+1) = \omega v_i(t) + C_1\varphi_1(pbest_i(t) - x_i(t)) + C_2\varphi_2(gbest(t) - x_i(t))$$

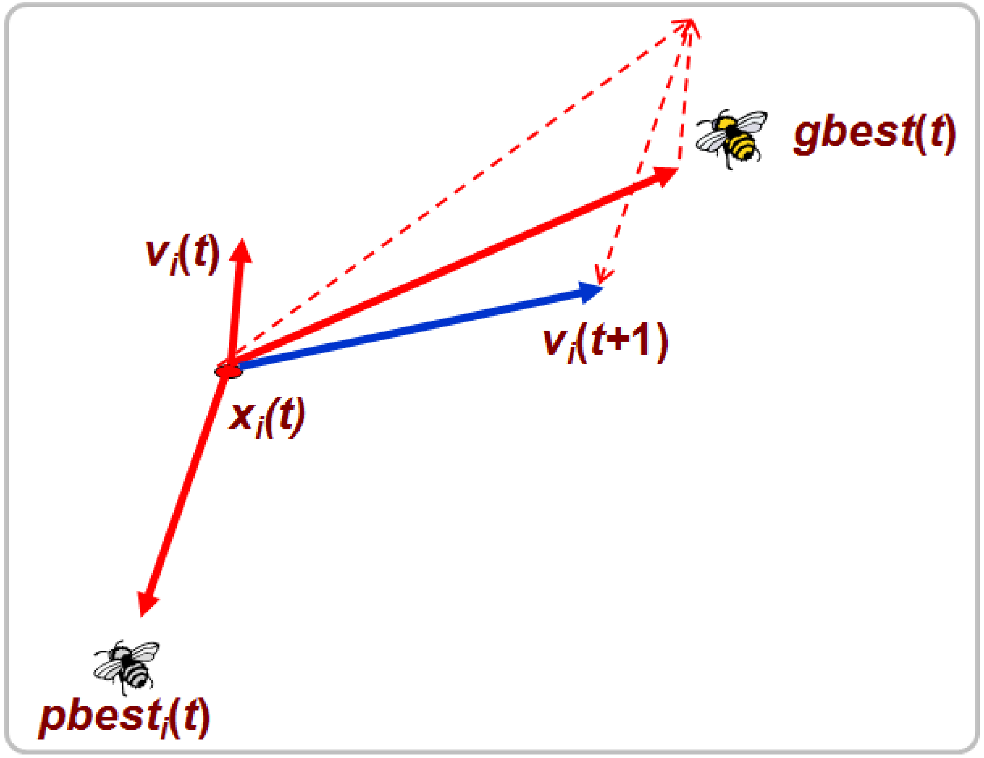
where

- $pbest_i(t)$ – personal best experience; the best value of the fitness function found by the i-th particle up to time t.
- $gbest(t)$ – global best experience; the best $pbest_j(t)$ value of all particles in the neighborhood of i (i.e. $j \in N(i)$) or the best value out of $pbest_j(t)$ values of all particles in the swarm found up to time t.
- ω – inertia vector.
- φ_1 and φ_2 – uniformly distributed random numbers that determine the influence of $pbest_i(t)$ and $gbest(t)$.
- C_1 – particle's self-confidence; controls the contribution towards the self-exploration.
- C_2 – swarm confidence; controls the contribution towards the global direction.

PSO: Velocity Update

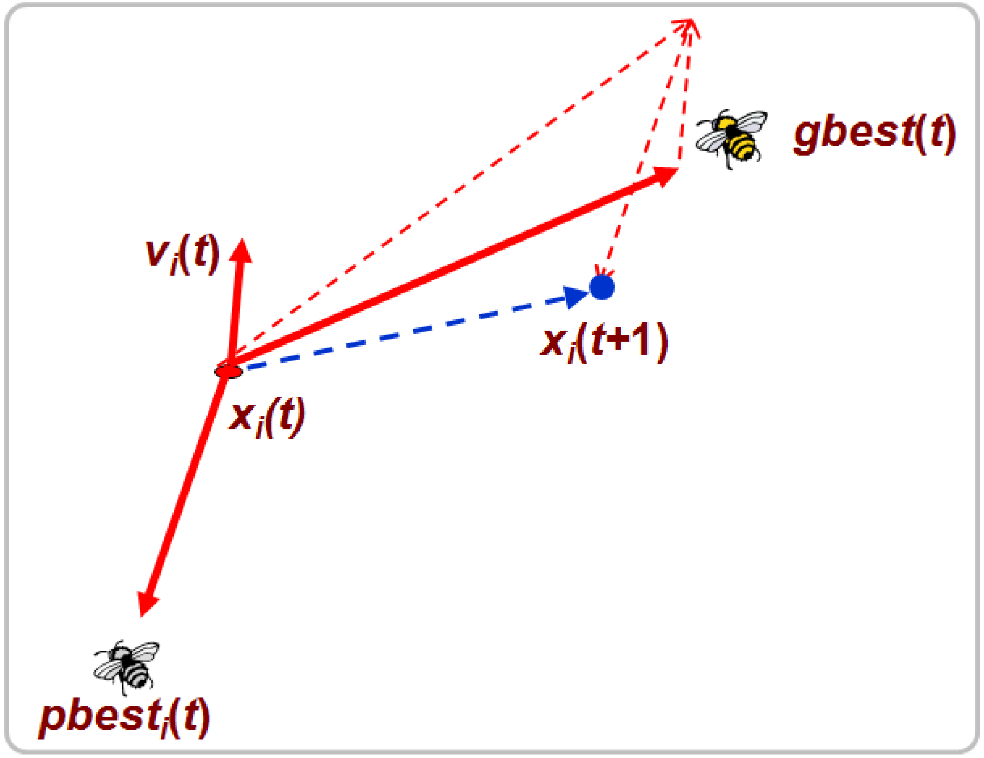
Update of the i-th particle velocity:

$$v_i(t+1) = \omega v_i(t) + C_1\varphi_1(pbest_i(t) - x_i(t)) + C_2\varphi_2(gbest(t) - x_i(t))$$



PSO: Position Update

Update of the i-th particle position:



PSO: Algorithm

Input: Number of particles in the swarm, $swarmSize$. Typical values are between 20-60.
Output: Position of the approximate global optimum $\mathbf{X}^*$

```
begin
    t = 0
    Randomly initialize position and velocity of particles: X_i(0) and V_i(0)
    while (termination condition is not reached) do
    begin
        t = t + 1
        calculate fitness f(X_i) of particles in the swarm
        update pbest_i(t) of particles
        update gbest(t) value observed so far in the swarm
        adapt velocity of all particles
        adapt position of all particles
    end
begin
```

PSO: Setting the Inertia Factor ω

Static parameter setting

- $\omega \ll 1$ – only little momentum is preserved from the previous time-step.

  $\omega = 0$ – the particle moves in each step totally ignoring information about the past velocity.

- $\omega > 1$ – particles can hardly change their direction which implies a reluctance against convergence towards optimum.

  $\omega > 1$ is always used with V_{max} to avoid *swarm explosion*.

Dynamic parameter setting – annealing scheme; ω decreases linearly with time from $\omega = 0.9$ to $\omega = 0.4$.

- Globally explores the search space in the beginning of the run.
- Performs local search in the end.

PSO: Acceleration Coefficients C_1 and C_2

Static setting – usually $C_1 = C_2$ and range within (0, 4), for example $C_1 = C_2 = 1.494$.

Dynamic setting – coefficients vary with time according to

$$C_1 = (C_{1f} - C_{1i})\frac{i}{MAXITER} + C_{1i}$$

$$C_2 = (C_{2f} - C_{2i})\frac{i}{MAXITER} + C_{2i}$$

- where C_{1f} and C_{2f} are final values for C_1 and C_2,
  C_{1i} and C_{2i} are current values at iteration i, and $MAXITER$ is the maximum number of iterations.
- Particular scheme: C_1 decreases from 2.5 to 0.5; C_2 increases from 0.5 to 2.5.
- Effect: Global search during the early phase of the optimization process; convergence to global optimum at the final stage of the optimization process.

Discrete Binary Particle Optimization Swarm Algorithm

Now, the optimization domain are functions of D binary variables, thus the solution is a binary vector $\mathbf{X} = \{0, 1\}^D$.

Particle characteristics

- **Velocity** – each particle i has its velocities, v_{id}, representing probabilities of having variable d set to 1.

  Ex.: Velocity $v_{id} = 0.2$ means that there is a twenty percent chance that the i-th particle will have its d-th variable set to one (80% chance it will be a zero).

- **Position** – a particular vector of binary values, x_{id}.

  The values are sampled from the vector of particle's velocities when the particle is evaluated.

  Ephemeral position – a particle might have a different actual position at every generation.

Discrete PSO: Velocity and Position Update

Velocity update: $v_{id} = v_{id} + \varphi_1(p_{id} - x_{id}) + \varphi_2(p_{gd} - x_{id})$

- x_{id}, p_{id} and p_{gd} are integers in $\{0, 1\}$.
- Since v_{id} is a probability, a logistic transformation $S(v_{id})$ is used to constrain its values within the interval $[0.0, 1.0]$.

  As v_{id} grows, the function $S(v_{id})$ approaches a one, thus the "position" of the particle fixes more probably on the value 1, with less chance of change.
- Parameter V_{max} is used to control the ultimate mutation rate of the bit vector; $|v_{id}| < V_{max}$ for all dimensions $d \in \{1, \ldots, D\}$.

  Ex.: If $V_{max} = 6.0$, then probabilities will be limited to $0.0025 \leq S(v_{id}) \leq 0.9975$. Thus exploration is ensured to some extent even after the population (swarm) has converged w.r.t. velocities.

  The smaller V_{max}, the higher mutation rate, and vice versa.

Position update:

$$x_{id} = \begin{cases} 1 & \text{, if } rand() < S(v_{id}) \\ 0 & \text{, otherwise} \end{cases}$$

Reading

- Das S. et al.: Particle Swarm Optimization and Differential Evolution Algorithms: Technical Analysis, Applications and Hybridization Perspectives, 2008
  `http://www.softcomputing.net/aciis.pdf`
- Dorigo, M. and Stützle, T.: The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances
  `http://www.agent.ai/doc/upload/200302/dori02.pdf`
- Socha, K. and Dorigo, M.: Ant colony optimization for continuous domains
  `http://www.sciencedirect.com/science/article/pii/S0377221706006333`
- Kennedy, J. and Eberhart, R.C.: A discrete binary version of the particle swarm algorithm
  `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=637339&tag=1`