Metaheuristics: from Design to Implementation

Chap 3
Population based Metaheuristics

Wiley, 2009 (596pp)
ISBN: 978-0-470-27858-1
Population solution-based metaheuristics

- “Improvement” of a population of solutions
- Recombination of solutions
- Exploration Oriented:
  - Generation
  - Replacement Offsprings
High-level template of P-metaheuristics

Algorithm 3.1 High-level template of P-metaheuristics.

\[ P = P_0 \quad \text{; /* Generation of the initial population */} \]

\[ t = 0 \quad ; \]

Repeat

\[ \text{Generate}(P'_t) \quad ; \quad \text{/* Generation a new population */} \]

\[ P_{t+1} = \text{Select-Population}(P_t \cup P'_t) \quad ; \quad \text{/* Select new population */} \]

\[ t = t+1 \quad ; \]

Until Stopping criteria satisfied

Output: Best solution(s) found.

Output: Best solution found.
High-level template of P-metaheuristics
# Search memory in P-metaheuristics

<table>
<thead>
<tr>
<th>P-metaheuristic</th>
<th>Search memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evolutionary algorithms (EA)</td>
<td>Population of individuals</td>
</tr>
<tr>
<td>Scatter Search (SS)</td>
<td>Population of solutions, reference set</td>
</tr>
<tr>
<td>Ant colonies (AC)</td>
<td>Pheromone matrix</td>
</tr>
<tr>
<td>Estimation of Distribution Algorithms (EDA)</td>
<td>Probabilistic learning model</td>
</tr>
<tr>
<td>Particle Swarm Optimization (PSO)</td>
<td>Population of particles, best global and local solutions</td>
</tr>
<tr>
<td>Bee colonies (BC)</td>
<td>Population of bees</td>
</tr>
<tr>
<td>Artificial immune systems (AIS): Clonal selection</td>
<td>Population of antibodies</td>
</tr>
</tbody>
</table>
Single solution metaheuristics are exploitation oriented
Population-based metaheuristics are exploration oriented
Taxonomy (Population-based Metaheuristics)

Metaheuristics

Population

Evol. algorithms

Scatter search

Ant colony

Evol. programming

Evol. strategies

Genetic algorithms

Genetic programming
Taxonomy (Population-based Metaheuristics)

(a) Evolutionary-based P-metaheuristics: evolutionary algorithms, scatter search, ...

(b) Blackboard-based P-metaheuristics: ant colonies, estimation distribution algorithms, ...

Fig. 3.2 Evolutionary-based versus Blackboard-based strategies in P-metaheuristics.
Outline

• Common concepts for P-metaheuristics
  – Initial population
  – Stopping criteria

• Evolutionary algorithms
  – Common concepts: selection, reproduction, replacement
  – Genetic algorithms
  – Genetic programming
  – Evolution strategies, Evolutionary programming
  – Other evolutionary algorithms
    • Estimation of Distribution Algorithms (EDA)
    • Differential evolution
    • Co-evolutionary algorithms
    • Cultural algorithms

• Scatter search and path relinking

• Swarm intelligence
  – Ant colonies
  – Particle swarm optimization

• Bee colonies

• Artificial immune systems
Initial population

- Random generation
  - Pseudo-random
  - Quasi-random
- Sequential diversification
- Parallel diversification
- Heuristic initialization
Initial population: Pseudo-random/ Quasi-random

Fig. 3.3 In the Latin hypercube strategy, the search space is decomposed into 25 blocks and a solution is generated pseudo-randomly in each block.

Fig. 3.4 In the pseudo-random generation, 25 solutions are generated independently in the search space.
Initial population: Hybrid approach

(1) Generate $Q$ random solutions

(2) Sequential diversification of $P-Q$ solutions

Population of $P$ individuals

Fig. 3.6 Hybrid initialization of the population.
Stopping criteria

• **Static procedure**: known a priori
  – Number of iterations
  – CPU time
  – Number of evaluations

• **Adaptive procedure**
  – Number of iterations without improvements
  – Diversity of the population
  – Optimal or satisfactory solution is reached
Initial population

Table 3.2 Analysis of the different initialization strategies. The evaluation is better with more plus sign (+). Sequential and parallel diversification strategies provide in general the best diversity followed by the quasi-random strategy. The heuristic initialization provides in general better solutions in terms of quality but with the expense of a higher computational cost and a reduced diversity. This will depend on the fitness landscape of the tackled optimization problem. For some landscapes (e.g. flat rugged), the diversity may remain important.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Diversity</th>
<th>Computational cost</th>
<th>Quality of initial solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudo-random</td>
<td>+++</td>
<td>+++</td>
<td>+</td>
</tr>
<tr>
<td>Quasi-random</td>
<td>+++</td>
<td>+++</td>
<td>+</td>
</tr>
<tr>
<td>Sequential diversification</td>
<td>++++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Parallel diversification</td>
<td>++++</td>
<td>+++</td>
<td>+</td>
</tr>
<tr>
<td>Heuristic</td>
<td>+</td>
<td>+</td>
<td>+++</td>
</tr>
</tbody>
</table>
Evolutionary Computation
History

• L. Fogel 1962 (San Diego, CA): *Evolutionary Programming*

• J. Holland 1962 (Ann Arbor, MI): *Genetic Algorithms*

• I. Rechenberg & H.-P. Schwefel 1965 (Berlin, Germany): *Evolution Strategies*

• J. Koza 1989 (Palo Alto, CA): *Genetic Programming*
The metaphor

Based on the evolution of a population of individuals

Evolution features
- Variation operators (crossover, mutation) to increase diversity,
- Selection of parents, replacement by offspring to decrease diversity

Table 3.3  Evolution process versus solving an optimization problem.

<table>
<thead>
<tr>
<th>Metaphor</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evolution</td>
<td>Problem solving</td>
</tr>
<tr>
<td>Individual</td>
<td>Solution</td>
</tr>
<tr>
<td>Fitness</td>
<td>Objective function</td>
</tr>
<tr>
<td>Environment</td>
<td>Optimization problem</td>
</tr>
<tr>
<td>Locus</td>
<td>Element of the solution</td>
</tr>
<tr>
<td>Allele</td>
<td>Value of the element (locus)</td>
</tr>
</tbody>
</table>
The ingredients

mutation

recombination

reproduction

selection

$t$

$\rightarrow$

$t + 1$
Evolutionary cycle

Fig. 3.7 A generation in evolutionary algorithms.
Evolutionary Algorithm procedure

Algorithm 3.2 Template of an evolutionary algorithm (EA).

Generate($P(0)$) ; /* Initial population */

t = 0 ;

While not Termination_Criterion($P(t)$) Do

Evaluate($P(t)$) ;

$P'(t)$ = Selection($P(t)$) ;

$P'(t)$ = Reproduction($P'(t)$); Evaluate($P'(t)$) ;

$P(t + 1)$ = Replace($P(t)$, $P'(t)$) ;

t = t + 1 ;

End While

Output Best individual or best population found.
Fig. 3.8  Genotype versus phenotype in evolutionary algorithms.
Domains of application

- Numerical, Combinatorial Optimisation
- System Modeling and Identification
- Planning and Control
- Engineering Design
- Data Mining
- Machine Learning
- Artificial Life
- ...

Performances

- Acceptable performance at acceptable costs on a wide range of problems
- Intrinsic parallelism (robustness, fault tolerance)
- Superior to other techniques on complex problems with
  - lots of data, many free parameters
  - complex relationships between parameters
  - many (local) optima
  - Adaptive, dynamic problems
Advantages

- No presumptions w.r.t. problem space
- Widely applicable
- Low development & application costs
- Easy to incorporate other methods
- Solutions are interpretable (unlike NN)
- Can be run interactively, accommodate user proposed solutions
- Provide many alternative solutions
- Robust regards any change of the environment (data, objectives, etc)
- Co-evolution, parallelism and distribution …
Disadvantages

- No guarantee for optimal solution within finite time (in general)
- May need parameter tuning
- Often computationally expensive, i.e. slow
Genetic Algorithms

- Developed: USA in the 1970’s
- Early names: J. Holland, K. DeJong, D. Goldberg
- Typically applied to:
  - discrete optimization
- Attributed features:
  - not too fast
  - good heuristic for combinatorial problems
- Special Features:
  - Traditionally emphasizes combining information from good parents (crossover)
  - many variants, e.g., reproduction models, operators
# Genetic Algorithms (SGA)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Representation</strong></td>
<td>Binary strings</td>
</tr>
<tr>
<td><strong>Recombination</strong></td>
<td>N-point or uniform</td>
</tr>
<tr>
<td><strong>Mutation</strong></td>
<td>Bitwise bit-flipping with fixed probability</td>
</tr>
<tr>
<td><strong>Parent selection</strong></td>
<td>Fitness-Proportionate</td>
</tr>
<tr>
<td><strong>Survivor selection</strong></td>
<td>All children replace parents</td>
</tr>
<tr>
<td><strong>Speciality</strong></td>
<td>Emphasis on crossover</td>
</tr>
</tbody>
</table>
Evolution Strategies

- Developed: Germany in the 1970’s
- Early names: I. Rechenberg, H.-P. Schwefel
- Typically applied to:
  - numerical optimisation
- Attributed features:
  - fast
  - good optimizer for real-valued optimisation
  - relatively much theory
- Special:
  - self-adaptation of (mutation) parameters standard
## Evolution Strategies

<table>
<thead>
<tr>
<th>Representation</th>
<th>Real-valued vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recombination</td>
<td>Discrete or intermediary</td>
</tr>
<tr>
<td>Mutation</td>
<td>Gaussian perturbation</td>
</tr>
<tr>
<td>Parent selection</td>
<td>Uniform random</td>
</tr>
<tr>
<td>Survivor selection</td>
<td>($\mu, \lambda$) or ($\mu + \lambda$)</td>
</tr>
<tr>
<td>Specialty</td>
<td>Self-adaptation of mutation step sizes</td>
</tr>
</tbody>
</table>
Evolutionary Programming

- Developed: USA in the 1960’s
- Early names: D. Fogel
- Typically applied to:
  - traditional EP: machine learning tasks by finite state machines
  - contemporary EP: (numerical) optimization
- Attributed features:
  - very open framework: any representation and mutation op’s OK
  - crossbred with ES (contemporary EP)
  - consequently: hard to say what “standard” EP is
- Special:
  - no recombination
  - self-adaptation of parameters standard (contemporary EP)
**Evolutionary Programming**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Representation</strong></td>
<td>Real-valued vectors</td>
</tr>
<tr>
<td><strong>Recombination</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Mutation</strong></td>
<td>Gaussian perturbation</td>
</tr>
<tr>
<td><strong>Parent selection</strong></td>
<td>Deterministic</td>
</tr>
<tr>
<td><strong>Survivor selection</strong></td>
<td>Probabilistic ($\mu+\mu$)</td>
</tr>
<tr>
<td><strong>Specialty</strong></td>
<td>Self-adaptation of mutation step sizes (in meta-EP)</td>
</tr>
</tbody>
</table>
Genetic Programming

- Developed: USA in the 1990’s
- Early names: J. Koza
- Typically applied to:
  - machine learning tasks (prediction, classification…)
- Attributed features:
  - competes with neural nets and alike
  - needs huge populations (thousands)
  - slow
- Special:
  - non-linear chromosomes: trees, graphs
  - mutation possible but not necessary (disputed!)
# Genetic Programming

<table>
<thead>
<tr>
<th>Representation</th>
<th>Tree structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recombination</td>
<td>Exchange of subtrees</td>
</tr>
<tr>
<td>Mutation</td>
<td>Random change in trees</td>
</tr>
<tr>
<td>Parent selection</td>
<td>Fitness proportional</td>
</tr>
<tr>
<td>Survivor selection</td>
<td>Generational replacement</td>
</tr>
</tbody>
</table>
Common search components for evolutionary algorithms

• Selection strategies: which parents are selected for reproduction

• Reproduction strategies: which variation operators we use

• Replacement strategies: how the current population is updated according to the generated offsprings
The selection strategy

- The main rule: “The better is an individual, the higher is its chance of being parent”.
- Such a selection pressure will drive the population forward
- Worst individuals shouldn’t be discarded but have some chance to be selected. This may lead to useful genetic material
The most common selection strategies

- Proportional fitness assignment: absolute fitnesses are associated to individuals

- Rank based fitness assignment: relative fitnesses are associated to individuals (e.g. rank)
The most common selection strategies

- Roulette wheel selection
- Stochastic Universal Sampling (SUS)
- Tournament selection
- Rank based selection
Fig. 3.11  Roulette selection strategies. In the standard roulette selection, each spin selects a single individual. In stochastic universal sampling (SUS), a spin will select as individuals asouters (e.g. 4 individuals in the example).
Fig. 3.12  Tournament selection strategy. For instance, a tournament of size 3 is performed. Three solutions are picked randomly from the population. The best solution from the picked individuals is then selected.
The rank based selection

- Individuals are sorted on their fitness value from best to worse. The place in this sorted list is called rank.
- Instead of using the fitness value of an individual, the rank is used by a function to select individuals from this sorted list. The function is biased towards individuals with a high rank (i.e. good fitness).
Fig. 3.13  Rank-based selection strategy using a linear ranking.
Replacement strategies

- **Generational replacement:** the replacement will concern the whole population of size $\mu$. The offspring population will replace systematically the parent population. This strategy is applied in the canonical GA as proposed by J. Holland.

- **Steady state replacement:** at each generation of an EA, only one offspring is generated. For instance, it replaces the worst individual of the parent population.
The replacement strategy

- Selection pressure is also affected in the replacement step (survivors of both population and offspring)
- Stochastic methods/deterministic strategies
- Elitism (i.e. should fitness ever improve?) → Reintroduce in the new generation the best solution found during the search
Variation operators

- Crossover operators
  - **Heritability**: should inherit material from the two parents
  - **Valid**: provides valid solutions

- Mutation operators
  - **Ergodicity**: every solution of the search space to be reached
  - **Validity**: provides valid solutions
  - **Locality**: minimal change (perturbation) → neighborhood concept in S-metaheuristics
Recombination operators

- We might have one or more recombination operators for our representation.

- Some important points are:
  - The child should inherit something from each parent. If this is not the case then the operator is a copy operator.
  - The recombination operator should be designed in conjunction with the representation.
  - Recombination should produce valid chromosomes.
Example. Recombination for discrete representation

- N-points crossover (e.g. 1 point)

- Uniform crossover
Example. Recombination for real valued representation

- Intermediate binary recombination (arithmetic crossover). Given two parents one child is created as follows

\[
\begin{align*}
0.1 & \quad 0.4 & \quad 0.3 & \quad 0.7 & \quad 0.4 \\
0.5 & \quad 0.8 & \quad 0.5 & \quad 0.2 & \quad 0 \\
\frac{0.1+0.5}{2} & \quad \frac{0.4+0.8}{2} & \quad \frac{0.3+0.5}{2} & \quad \frac{0.7+0.2}{2} & \quad \frac{0.4+0}{2}
\end{align*}
\]
Example. Recombination for real valued representation

Fig. 3.18 Offsprings distribution using the crossover operators UNDX, SPX and PCX.
Recombination for order based representation

- Choose an **arbitrary part** from the first parent and **copy** this to the first child
- **Copy the remaining genes** that are not in the copied part to the first child
  - starting right from the cut point of the copied part
  - using the order of genes from the second parent
  - wrapping around at the end of the chromosome
- Repeat this process with the parent roles reversed
Example. Recombination for order based representation (Order 1)

Parent 1

7 3 1 8 2 4 6 5

Parent 2

4 3 2 8 6 7 1 5

Child 1

7 5 1 8 2 4 3 6

7, 3, 4, 6, 5

4, 3, 6, 7, 5

order
Example. Recombination for order based representation (PMX)

Fig. 3.20 The partially mapped crossover (PMX) for permutations.
Recombination for tree based representation

Two sub-trees are selected for swapping.

\[ \pi \times (r + \left( \frac{1}{r} \right)) \]
Recombination for tree based representation

π * + r /

π * * r 2 + r /

Resulting in 2 new expressions

2 * * r r

π * r r

2 + 1 r
Mutation operators

- We might have **one or more** mutation operators for our representation.
- Some important points are:
  - At least one mutation operator should **allow every part of the search space to be reached**.
  - The **size of mutation** is important and should be controllable.
  - Mutation should produce **valid chromosomes**.
Example. Mutation for discrete representation

Mutation usually happens with probability $p_m$ for each gene

- It could affect only one gene too
Example. Mutation for real valued representation

- Perturb values by adding some random noise
- Often, a Gaussian/normal distribution $N(0, \sigma)$ is used, where
  - $0$ is the mean value
  - $\sigma$ is the standard deviation and

\[ x'i = xi + N(0, \sigma i) \]
Example. Mutation for order based representation

- Randomly select two different genes and swap them

```
before 1 0 1 0
       |   |
       1 0
after 1 0 0 1 1
```
Example. Mutation for tree-based representation

- Single point mutation selects one node and replaces it with a similar one

```
  *  ->  *  
 / \      /   \ 
 2   *    pi   * 
 /   \    /   \
 r    r   r    r
```
Swarm Intelligence
Swarm Intelligence

- Collective system capable of accomplishing difficult tasks in dynamic and varied environments without any external guidance or control and with no central coordination

- Achieving a collective performance which could not normally be achieved by an individual acting alone

- Constituting a natural model particularly suited to distributed problem solving
Inherent features

• Inherent parallelism
• Stochastic nature
• Adaptivity
• Use of positive feedback (reinforcement learning)
• Autocatalytic in nature
Swarm intelligence

Bird flocking  Fish
Swarm intelligence

Bees

Ants
Ant colony Optimization (ACO)
Ant colonies

- Artificial ants: Dorigo (1992)
- Imitate the cooperative behavior of ant colonies to solve optimization problems
- Use very simple communication mechanism: pheromone
  - Olfactive and volatile substance
  - Evolution: evaporation, reinforcement

A nature-inspired process

- During the trip, a pheromone is left on the ground.
- The quantity left depends on the amount of food found.
- The path is chosen accordingly to the quantity of pheromones.
- The pheromone has a decreasing action over time.
Template of Ant colony optimization

Algorithm 3.12 Template of the ant colony algorithm (ACO).

Initialize the pheromone trails ;

Repeat

For each ant  Do

Solution construction using the pheromone trail ;

Update the pheromone trails:

Evaporation ;

Reinforcement ;

Until Stopping criteria

Output: Best solution found or a set of solutions.
General ACO

- A stochastic construction procedure
- Probabilistically build a solution
- Iteratively adding solution components to partial solutions
  - Heuristic information
  - Pheromone trail
- Reinforcement Learning reminiscence
- Modify the problem representation at each iteration
Pheromone based Construction

• At the beginning of the search process, a constant amount of pheromone is assigned to all arcs. When located at a node $i$ an ant $k$ uses the pheromone trail to compute the probability of choosing $j$ as the next node:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha}{\sum_{l \in N_i^k} \tau_{il}^\alpha} & \text{if } j \in N_i^k \\ 0 & \text{if } j \notin N_i^k \end{cases}$$

• where $N_i^k$ is the neighborhood of ant $k$ when in node $i$. 
Reinforcement of the pheromone

- When the arc \((i,j)\) is traversed, the pheromone value changes as follows:

\[
\tau_{ij} \leftarrow \tau_{ij} + \Delta \tau^k
\]

- By using this rule, the probability increases that forthcoming ants will use this arc.
Evaporation

• After each ant $k$ has moved to the next node, the pheromones evaporate by the following equation to all the arcs:

$$
\tau_{ij} \leftarrow (1 - p)\tau_{ij}, \quad \forall (i, j) \in A
$$

• where $p \in (0, 1]$ is a parameter. An iteration is a completer cycle involving ants’ movement, pheromone evaporation, and pheromone deposit.
1. Represent the problem in the form of sets of components and transitions, or by a set of weighted graphs, on which ants can build solutions
2. Define the meaning of the pheromone trails
3. Define the heuristic preference for the ant while constructing a solution
4. If possible implement an efficient local search algorithm for the problem to be solved.
5. Choose a specific ACO algorithm and apply to problem being solved
6. Tune the parameter of the ACO algorithm.
How to implement in a program

• Ants: Simple computer agents

• Move ant: Pick next component in the const. solution

• Pheromone: $\Delta \tau_{i,j}^k$

• Memory: $M_K$ or $\text{Tabu}_K$

• Next move: Use probability to move ant
A simple TSP example

d_{AB} = 100; d_{BC} = 60...; d_{DE} = 150
**Iteration 1**

- **[A]** at position 1
- **[B]** at position 2
- **[C]** at position 3
- **[D]** at position 4
- **[E]** at position 5

Points:
- A
- B
- C
- D
- E
How to build next sub-solution?

\[
p_{ij}^k(t) = \begin{cases} 
\frac{\left[\tau_{ij}(t)\right]^\alpha \left[\eta_{ij}\right]^\beta}{\sum_{k \in \text{allowed}} \left[\tau_{ik}(t)\right]^\alpha \left[\eta_{ik}\right]^\beta} & \text{if } j \in \text{ allowed} \quad k \\
0 & \text{otherwise}
\end{cases}
\]
Iteration 2

A

[B,C]

C

[B,C]

D

[A,D]

E

[D,E]
Iteration 4

[B,C,D,A]  
[D,E,A,B]  
[E,A,B,C]  
[C,B,E,D]  
[A,DCE]  

A  2  
B  4  
C  5  
D  3  
E  1
Iteration 5

[C, B, E, D, A]

[A, D, C, E, B]

[D, E, A, B, C]

[E, A, B, C, D]

[B, C, D, A, E]
Path and Pheromone Evaluation

1. $\text{[A,D,C,E,B]}$

$L_1 = 300$

$\Delta \tau_{i,j}^k = \begin{cases} 
\frac{Q}{L_k} & \text{if } (i, j) \in \text{tour} \\
0 & \text{otherwise}
\end{cases}$

2. $\text{[B,C,D,A,E]}$

$L_2 = 450$

3. $\text{[C,B,E,D,A]}$

4. $\text{[D,E,A,B,C]}$

$L_4 = 280$

5. $\text{[E,A,B,C,D]}$

$L_5 = 420$

$\Delta \tau_{total}^{A,B} = \Delta \tau_{A,B}^1 + \Delta \tau_{A,B}^2 + \Delta \tau_{A,B}^3 + \Delta \tau_{A,B}^4 + \Delta \tau_{A,B}^5$
End of First Run

Save Best Tour (Sequence and length)

All ants die

New ants are born

\[ t = 0; \text{NC} = 0; \tau_{ij}(t) = c \text{ for } \Delta \tau_{ij} = 0 \]
Place the m ants on the n nodes

\[ \tau_{ij}(t) = \tau_{ij}(t) + \Delta \tau_{ij} \]

\[ \Delta \tau_{ik}^k = \begin{cases} \frac{Q}{L_k} & \text{if } (i, j) \in \text{tour described by tabu}_k \\ 0 & \text{otherwise} \end{cases} \]

\[ p_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{k \in \text{allowed}_k} [\tau_{ik}(t)]^\alpha [\eta_{ik}]^\beta} & \text{if } j \in \text{allowed}_k \\ 0 & \text{otherwise} \end{cases} \]
Stopping Criteria

- Stagnation
- Max Iterations
Particle Swarm Optimization (PSO)
Particle Swarm

- Population based stochastic metaheuristic
- Dr. Eberhart and Dr. Kennedy (1995)
- Inspired by social behavior of bird flocking or fish schooling
- Similarities with genetic algorithm

Fig. 3.33  Particle swarm with their associated positions and velocities. At each iteration, a particle moves from one position to another one in the decision space. PSO uses no gradient information during the search.
A nature-inspired process

- Particles fly through the problem space
- Flight = add a velocity to the current position
- Social adaptation of knowledge
- Particles follow the current optimum particles («follow the bird which is nearest to the food »)
Algorithm 3.14 Template of the particle swarm optimization (PSO) algorithm.

Random initialization of the whole swarm;

Repeat
Evaluate $f(x_i)$;
For all particles $i$
Update velocities:
$$v_i(t) = v_i(t - 1) + \rho_1 \times (p_i - x_i(t - 1)) + \rho_2 \times (p_g - x_i(t - 1)) ;$$
Move to the new position: $x_i(t) = x_i(t - 1) + v_i(t) ;$
If $f(x_i) < f(pbest_i)$ Then $pbest_i = x_i ;$
If $f(x_i) < f(gbest)$ Then $gbest = x_i ;$
Update($x_i, v_i$);
EndFor
Until Stopping criteria
Swarm construction

- Initialize positions $P_i$:
  
  $P_i = \text{random}$

- Initialize the first best position $P_{best}$ of each particle:
  
  $P_{i,\text{best}} = P_i$ (standard strategy)

- Initialize the global best $P_{gbest}$ particle:
  
  $P_{gbest} = \text{best} (P_i)$ (standard strategy)
Make the particles flying

- Evaluate the velocities:
  \[ V_i = V_i + c_1 \times (P_{i\text{best}} - P_i) + c_2 \times (P_{g\text{best}} - P_i) \]

  local direction           global direction
  \[ \Rightarrow c_1 \text{ and } c_2 = \text{learning factors} \]

- Perform the flight
  \[ P_i = P_i + V_i \]
Make the particles flying

Fig. 3.35  Movement of a particle and the velocity update.
Particle oscillation

Objective

1 2 3

Landscape

1 2 3

Solution

1 2 3

Strong oscillation

Smooth oscillation
Topology (neighborhoods)

- Determines how the solution spread through the population
- Local, global, neighbourhood best?
- Affects the rate of convergence
- Advanced parallel search

Mean degree, Clustering, Heterogeneity
Topology (neighborhoods)

(a) Complete graph

(b) Local structure: a ring

(c) "Small world graph"
Update the particle’s best

• Update the best fitness value of each particle:
  • If $P_i$ better than $P_{i,\text{best}}$
    $P_{i,\text{best}} = P_i$

• Update the global best:
  • If $P_i$ better than $P_{\text{gbest}}$
    $P_{\text{gbest}} = P_i$
Discrete PSO

Fig. 3.36  Geometric crossover in the GPSO algorithm.
Oscillation

Fig. 3.37 Strong oscillations versus smooth ones in a particle swarm.
Bee colony in nature: organisation

- Queen
- Drones
- Workers
- Broods
Bee colony

- Nest site selection
- Food foraging
- Marriage process
Fig. 3.38  The waggle dance: the direction is indicated by the angle from the sun; the distance is defined by the duration of the waggle part of the dance. A waggle run oriented 40 degrees to the right indicates a food source 40 degrees to the right of the direction of the sun outside the hive. The bee runs through a figure-eight pattern on a vertical comb. It passes through the central, and performs the waggle run with vibrating her body laterally. The waggle run duration is related to the food source distance with a rate of increase of about 75 milliseconds per 100 meters [692].
Bee colony: Food foraging

- Unemployed foragers
- Employed foragers

Table 3.8 Analogy between natural and artificial bee colonies.

<table>
<thead>
<tr>
<th>Natural bee colony</th>
<th>Artificial bee colony</th>
</tr>
</thead>
<tbody>
<tr>
<td>Food source</td>
<td>Solution</td>
</tr>
<tr>
<td>Quality of nectar</td>
<td>Objective function</td>
</tr>
<tr>
<td>Onlookers</td>
<td>Exploitation of search</td>
</tr>
<tr>
<td>Scout</td>
<td>Exploration of search</td>
</tr>
</tbody>
</table>

- Exploration of food sources
- Exploitation of food sources
Fig. 3.39 Bee colony behavior for food source (nectar) discovering. We assume two discovered food sources $A$ and $B$ [446]. An unemployed bee has no knowledge about food source. It can be a scout bee $S$, starting to explore the neighborhood of the nest, or a onlooker bee $O$, watching for the waggle dances. After the localization of a food source, the bee becomes an employed bee. After unloading the nectar, the employed bee has three alternatives: abandoning the food source ($A$), recruit other bees from the nest ($R$), foraging without recruiting other bees ($F$).
Template of the bee algorithm (BA)

Algorithm 3.15 Template of the bee algorithm (BA).

Random initialization of the whole colony of bees;
Evaluate the fitness of the population of bees;
Repeat /* Forming new population */
  Select sites for neighborhood search; Determine the patch size;
  Recruit bees for selected sites and evaluated their fitness;
  Select the representative bee from each patch;
  Assign remaining bees to search randomly and evaluate their fitness;
Until Stopping criteria
Algorithm 3.16 Template of the Marriage in honeyBees Optimization (MBO) algorithm.

Random initialization of the queen’s;
Improve the queen with workers (S-metaheuristic);
For predefined maximum number of mating-flights Do
Initialize energy and speed;
While queen’s energy > 0 Do
 The queen moves between states and probabilistically chooses drones;
 If a drone is selected Then
 Add its sperm to the queen’s spermatheca;
 Update the queen’s internal energy and speed;
 Endwhile
Generate broods by haploid-crossover and mutation;
Use the workers (S-metaheuristic) to improve the broods;
Update workers’ fitness;
If The best brood is fitter than the queen Then
 Replace the queen’s chromosome with the best brood’s chromosome;
 Kill all broods;
 Endfor
Artificial Immune Systems

- **Representation**: representation of AIS components (e.g. antigens, antibodies, cells, molecules, ...)

- **Affinity**: evaluation of the interaction of the system’s components (i.e. each other and with the environment)

- **Adaptation**: procedure that govern the dynamics of the whole system
Artificial Immune Systems

- **Population-based**: do not take into account the immune network
  - Clonal selection
  - Negative selection

- **Network-based**: inspired by the network theory
  - Immune networks
  - Danger theory
AIS in nature

• Two type of immunity
  – **Innate immune system**: the body is born with the ability to recognize a microbe and immediately destroy it. It protects our body from non-specifics pathogens
  
  – **Adaptive immune system**: acquired immune system complete the innate one and removes the pathogens that persist to it

  • **Lymphocytes**: two type of cells: B cells and T cells responsible for recognizing and destroying any antigen
AIS: Clonal selection

Selection

Paratope

Epitope

B cell repertoire

Antigens

Cloning

Proliferation

Antigen binding

Differentiation

Memory cells

Plasma cells

Secreted antibodies

M

M
**AIS: Clonal selection**

Table 3.9 Analogy between the natural immune system and the optimization problem according to the clonal selection theory.

<table>
<thead>
<tr>
<th>Natural immune system</th>
<th>Optimization problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antibody</td>
<td>Solution</td>
</tr>
<tr>
<td>Affinity</td>
<td>Objective function</td>
</tr>
<tr>
<td>Antigen</td>
<td>Optimization problem</td>
</tr>
<tr>
<td>Cloning</td>
<td>Reproduction of solutions</td>
</tr>
<tr>
<td>Somatic mutation (hypermutation)</td>
<td>Multiple mutation of a solution</td>
</tr>
<tr>
<td>Affinity maturation</td>
<td>Mutation and selection of best solutions</td>
</tr>
<tr>
<td>Receptor editing</td>
<td>Diversification</td>
</tr>
</tbody>
</table>
Algorithm 3.17 Template of the CLONALG (CLOnal selection ALGorithm).

Input: Initial population $P_0$.

$P = P_0$ ; /* Generation of the initial population of random antibodies */

Repeat
    Evaluate all existing antibodies and compute their affinities ;
    Select $N\%$ of antibodies with highest affinities ;
    Clone the selected antibodies ;
    Maturate the cloned antibodies ;
    Evaluate all cloned antibodies ;
    Add $R\%$ of best cloned antibodies to the pool of antibodies ;
    Remove worst members of the antibodies pool ;
    Add new random antibodies into the population ;

Until Stopping criteria satisfied

Output: Best population found.
Scatter search and path relinking
Scatter Search and Path Relinking

- Scatter Search and its generalized form Path Relinking provide unifying principles for joining (or recombining) solutions based on generalized path constructions in Euclidean or neighborhood spaces.


Main operators

- **Diversification Generation Method**: Generate a collection of diverse trial solutions
- **Improvement Method**: Transform a trial solution into one or more enhanced trial solutions
- **Reference Set Update Method**: Build and maintain a reference set - "best" solutions found (quality, diversity)
- **Subset Generation Method**: Operate on the reference set, to produce a subset of its solutions as a basis for creating combined solutions
- **Solution Combination Method**: Transform a given subset of solutions produced by the Subset Generation Method
Scatter search components

Fig. 3.29  Search components of scatter search algorithms.
Algorithm 3.10 Template of the scatter search algorithm (SS).

/* Initial phase */
Initialize the population $Pop$ using a diversification generation method;
Apply the improvement method to the population;
Reference set Update Method;
/* Scatter search iteration */
Repeat
  Subset generation method;
  Repeat
    Solution Combination Method;
    Improvement Method;
  Until Stopping criteria 1
Reference Set Update Method;
Until Stopping criteria
Output: Best found solution or set of solutions.
Path-Relinking Solutions

Initiating solution

Guiding solution

Original path

Relinked path
Template of Path relinking

Algorithm 3.11 Template of the basic path relinking (PR) algorithm.

**Input:** Starting solution $s$ and target solution $t$.

$x = s$;

**While** $\text{dist}(x, t) \neq 0$ **Do**

Find the best move $m$ which decreases $\text{dist}(x \oplus m, t)$;

$x = x \oplus m$; /* Apply the move $m$ to the solution $x$ */;

**Output:** Best solution found in trajectory between $s$ and $t$. 
Multiple Guiding Solutions

- Initiating solution
- Guiding solution
- Original path
- Relinked path
Linking Solutions

- Initiating solution
- Guiding solution

- Original path
- Relinked path
Fig. 3.31  Different path relinking strategies in terms of the starting and guiding solutions.
Intermediate Solutions
Estimation of Distribution Algorithm (EDA)
Estimation of Distribution Algorithm

- Based on the use of (unsupervised) density estimators/generative statistical models
- Idea is to convert the optimization problem into a search over probability distributions
- The probabilistic model is in some sense an explicit model of (currently) promising regions of the search space
Estimation of Distribution Algorithm

(a) Genetic algorithm (GA)

(b) Estimation distribution algorithm (EDA)

Fig. 3.23 Estimation of distribution algorithms versus genetic algorithms.
Template of the EDA algorithm

Algorithm 3.4 Template of the EDA algorithm.

\[
t = 1;
\]

Generate randomly a population of \( n \) individuals;

Initialize a probability model \( Q(x) \);

While Termination criteria are not met Do

Create a population of \( n \) individuals by sampling from \( Q(x) \);

Evaluate the objective function for each individual

Select \( m \) individuals according to a selection method;

Update the prob. model \( Q(x) \) using selected population and \( f() \) values;

\[
t = t + 1;
\]

End While

Output: Best found solution or set of solutions.
Level of variable interaction in the definition of the probabilistic model

- **Univariate**: no interaction between variables
- **Bivariate**: interaction between two variables
- **Multivariate**: more than two variables
EDA simplest probability model: Population-based incremental Learning (PBIL)

Algorithm 3.5 Template of the PBIL algorithm (Population-based incremental learning).

\[
\text{Initial distribution } D = (0.5, ..., 0.5) ; \\
\text{Repeat} \\
\text{Generation of the population ;} \\
\text{If } r < D_i \text{ (} r \text{ uniform in } [0, 1]) \text{ Then } X_i = 1 \\
\text{Else } X_i = 0 ; \\
\text{Evaluate and sort the population ;} \\
\text{Update the distribution } D = (1 - \alpha)D + \alpha X_{best} ; \\
\text{Until Stopping criteria}
\]
Fig. 3.24  EDA and GA in solving a bit vector optimization problem.
Fig. 3.25 Probability distribution in the PBIL (Population-based Incremental Learning) algorithm.
Other probability models

- Mutual Information Maximization for Input Clustering (MIMIC) regards pairwise dependances
- **Gaussian networks** for continuous optimization
- **Bayesian networks** for discrete optimization: Bayesian Optimization Algorithm (BOA) for multivariate dependances
Algorithm 3.6 Template of the BOA algorithm (Bayesian Optimization Algorithm).

Initialize the population $P(0)$ randomly; Evaluate ($P(0)$);

Repeat

Select a set of promising solutions $S(t)$ from $P(t)$;

Construct the network $B$ using a given metric and constraints;

Generate new solutions $O(t)$ according to the joint distribution encoded by $B$;

Create population $P(t+1)$ by replacing some solutions from $P(t)$ with $O(t)$;

Evaluate $P(t))$; $t = t + 1$;

Until Stopping criteria
EDA: Applications

- Field of EDA is quite young. Much effort is focused on methodology rather than performance

- First applications
  - Knapsack problem
  - Job Shop Scheduling
Co-evolutionary algorithms: competitive

Fig. 3.26 Competitive coevolutionary algorithms based on the predator-prey model.
Co-evolutionary algorithms: cooperative

Fig. 3.27 A cooperative coevolutionary algorithm.
Cultural algorithms

Fig. 3.28  Search components of cultural algorithms.
Cultural algorithms

Algorithm 3.9 Template of the cultural algorithm (CA).

Initialize the population $Pop(0)$;

Initialize the Belief $BLF(0)$;

$t = 0$

Repeat

Evaluate population $Pop(t)$;

Adjust($BLF(t)$, Accept($POP(t)$))

Evolve($Pop(t+1)$, Influence($BLF(t)$))

$t = t + 1$

Until Stopping criteria

Output: Best found solution or set of solutions.
Guideline to design and implement a P-metaheuristic
Common concepts for metaheuristics
- Representation
- Objective function
- Constraint handling

Common concepts for P-metaheuristics
- Initial population
- Stopping criteria

Common concepts for evolutionary-based P-metaheuristics
(Evolutionary algorithms, differential evolution, scatter search, marriage bee process, clonal selection, ...)
- Selection
- Variation (e.g. recombination)

Common concepts for blackboard-based P-metaheuristics
(Ant colonies, estimation of distribution algorithms, particle swarm optimization, food foraging bees, ...)
- Construction of shared memory
- Construction of a solution from the shared memory

Implementation of a P-metaheuristic
- From scratch or no reuse
- Code reuse
- Design and code reuse (e.g. software framework ParadisEO-EO)

Parameter tuning
Performance evaluation
Landscape analysis