Biologically Inspired Algorithms (A4M33BIA)

Optimization: Conventional and Unconventional Optimization Techniques

Jiří Kubalík

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Slides for the Softcomputing course, CTU in Prague, Department of Cybernetics, 2008.
Biologically Inspired Optimization Techniques

What is this subject about?
What are Biologically Inspired Optimization Techniques?

Optimization

Conventional Constructive Methods

Conventional Generative Methods

Unconventional Methods Inspired by Nature

Biologically Inspired Optimization Techniques
What is this subject about?

About **problem solving** by means of **biologically inspired optimization techniques**:
- Optimization using Evolutionary Algorithms
- Modelling (classification, regression) using Neural Networks
What is this subject about?

About **problem solving** by means of **biologically inspired optimization techniques**:

- Optimization using Evolutionary Algorithms
- Modelling (classification, regression) using Neural Networks

What are ‘problematic’ problems?

- no low-cost, analytic and complete solution is known

Why are they ‘problematic’?

1. *number of possible solutions* grows very quickly with the problem size
2. the goal is *noisy* or *time dependent*
3. the goal must fulfill some *constraints*
4. *barriers inside the people* solving the problem
About **problem solving** by means of **biologically inspired optimization techniques**:  
✔ Optimization using Evolutionary Algorithms  
✔ Modelling (classification, regression) using Neural Networks  

What are ‘problematic’ problems?  
✔ no low-cost, analytic and complete solution is known  

Why are they ‘problematic’?  
1. *number of possible solutions* grows very quickly with the problem size  
   ✔ complete enumeration impossible  
2. the goal is *noisy* or *time dependent*  
3. the goal must fulfill some *constraints*  
4. *barriers inside the people* solving the problem
What is this subject about?

About **problem solving** by means of **biologically inspired optimization techniques**:

- Optimization using Evolutionary Algorithms
- Modelling (classification, regression) using Neural Networks

What are ‘problematic’ problems?

- no low-cost, analytic and complete solution is known

Why are they ‘problematic’?

1. *number of possible solutions* grows very quickly with the problem size
2. the goal is *noisy* or *time dependent*
   - the solution process must be repeated over and over
   - averaging to deal with noise
3. the goal must fulfill some *constraints*
4. *barriers inside the people* solving the problem
What is this subject about?

About **problem solving** by means of **biologically inspired optimization techniques**:
- Optimization using Evolutionary Algorithms
- Modelling (classification, regression) using Neural Networks

What are ‘problematic’ problems?
- no low-cost, analytic and complete solution is known

Why are they ‘problematic’?
1. **number of possible solutions** grows very quickly with the problem size
2. the goal is **noisy** or **time dependent**
3. the goal must fulfill some **constraints**
   - on one hand, the constraints decrease the number of feasible solutions
   - on the other hand, they make the problem much more complex, sometimes it is very hard to find **any feasible solution**
4. **barriers inside the people** solving the problem
What is this subject about?

About **problem solving** by means of **biologically inspired optimization techniques**:

✔ Optimization using Evolutionary Algorithms
✔ Modelling (classification, regression) using Neural Networks

What are ‘problematic’ problems?

✔ no low-cost, analytic and complete solution is known

Why are they ‘problematic’?

1. *number of possible solutions* grows very quickly with the problem size
2. the goal is *noisy* or *time dependent*
3. the goal must fulfill some *constraints*
4. *barriers inside the people* solving the problem
   ✔ insufficient equipment (money, knowledge, …)
   ✔ psychological barriers (insufficient abstraction or intuition ability, ‘fossilization’, influence of ideology or religion, …)
What are Biologically Inspired Optimization Techniques?

**Biologically Inspired Optimization Techniques**

- computational methods inspired by evolution, by nature, and by the brain are being used increasingly to solve complex problems in engineering, computer science, robotics and artificial intelligence.
- studies, models and analyzes very complex phenomena for which no low-cost, analytic, or complete solution is known
- consists especially of:
  - evolutionary algorithms, swarm intelligence
  - artificial neural networks
  - fuzzy systems
  - probability theory, possibility theory, Bayesian networks
- tolerance for
  - imprecision
  - partial truth
  - uncertainty
  - noise
Optimization

Biologically Inspired Optimization Techniques

Optimization
Optimization problems
Neighborhood, local optimum
Conventional optimization methods
What's next?

Conventional Constructive Methods

Conventional Generative Methods

Unconventional Methods Inspired by Nature
Among all possible objects $x \in \mathcal{F} \subseteq S$, we want to determine such an object $x_{\text{OPT}}$ that optimizes (minimizes) the function $f$:

$$x_{\text{OPT}} = \arg \min_{x \in \mathcal{F} \subseteq S} f(x)$$  \hfill (1)
Optimization problems

Among all possible objects $x \in \mathcal{F} \subset S$, we want to determine such an object $x_{\text{OPT}}$ that optimizes (minimizes) the function $f$:

$$x_{\text{OPT}} = \arg \min_{x \in \mathcal{F} \subset S} f(x)$$

The space of candidate solutions $S$ and the objective function $f$:

1. Representation of the solution
   - syntactical structure that holds the ‘solution’
   - induces the search space $S$ and its feasible part $\mathcal{F}$

2. Optimization criterion, objective function, evaluation function $f$
   - function that is optimized
   - ‘understands’ the solution representation
   - adds meaning (semantics) to the solution representation
   - gives us the measure of the solution quality (or, at least, allows us to say that one solution is better than the other)
Among all possible objects \( x \in \mathcal{F} \subseteq S \), we want to determine such an object \( x_{\text{OPT}} \) that optimizes (minimizes) the function \( f \):

\[
 x_{\text{OPT}} = \arg \min_{x \in \mathcal{F} \subseteq S} f(x)
\]

(1)

The space of candidate solutions \( S \) and the objective function \( f \):

1. Representation of the solution
   - syntactical structure that holds the ‘solution’
   - induces the search space \( S \) and its feasible part \( \mathcal{F} \)

2. Optimization criterion, objective function, evaluation function \( f \)
   - function that is optimized
   - ‘understands’ the solution representation
   - adds meaning (semantics) to the solution representation
   - gives us the measure of the solution quality (or, at least, allows us to say that one solution is better than the other)

**Black-box optimization**: the inner structure of function \( f \) is unknown—it is black box for us (and for the algorithm); the function can be

- \( \times \) continuous \( \times \) discrete
- differentiable
- decomposable
- noisy, time dependent
Neighborhood, local optimum

The **neighborhood** of a point \( x \in S \):

\[
N(x, d) = \{ y \in S | \text{dist}(x, y) \leq d \}
\]  

(2)

Measure of the **distance between points** \( x \) and \( y \): \( S \times S \to \mathcal{R} \)
- ✔ binary space: Hamming distance
- ✔ real space: Euclidean, Manhattan (City-block), Mahalanobis, …
- ✔ matrices: Amari
- ✔ in general: number of applications of some operator that would transform \( x \) into \( y \) in \( \text{dist}(x, y) \) steps
Neighborhood, local optimum

The neighborhood of a point $x \in S$:

$$N(x, d) = \{y \in S | \text{dist}(x, y) \leq d\}$$  \hspace{1cm} (2)

Measure of the distance between points $x$ and $y$: $S \times S \rightarrow \mathbb{R}$

- binary space: Hamming distance
- real space: Euclidean, Manhattan (City-block), Mahalanobis, …
- matrices: Amari
- in general: number of applications of some operator that would transform $x$ into $y$ in $\text{dist}(x, y)$ steps

Local optimum:

- Point $x$ is local optimum, if $f(x) \leq f(y)$ for all points $y \in N(x, d)$ for some positive $d$.
- Small finite neighborhood (or the knowledge of derivatives) allows for validation of local optimality of $x$. 
Conventional optimization methods

Why are there so many of them?
Conventional optimization methods

Why are there so many of them?
✔ they are not robust, small change in the problem definition usually requires a completely different method

Classification of optimization algorithms (one of many possible):

1. Constructive algorithms
   ✔ work with partial solutions, construct full solutions incrementally
   ✔ not suitable for black-box optimization, must be able to evaluate partial solutions
   ✔ require discrete search space
   ✔ based on decomposition, arranging the search space in the form of a tree, …

2. Generative algorithms
   ✔ work with complete candidate solutions, generate them as a whole
   ✔ suitable for black-box optimization, only complete solutions need to be evaluated
   ✔ can be interrupted anytime—always have a solution to provide; this solution is usually not optimal (needn’t be even locally optimal)
What’s next?

1. Review of conventional constructive methods
   - greedy algorithm, divide and conquer, dynamic programming, branch and bound, $A^*$

2. Conventional generative methods
   - Complete enumeration, local search, Nelder-Mead simplex search, gradient methods, linear programming

3. Unconventional methods inspired by nature
Conventional Constructive Methods
Divide and Conquer

1. Problem decomposition
2. Subproblem solutions
3. Combination of partial solutions

Issues:
✔ Is the problem decomposable? (If we combine optimal solutions of subproblems, do we get the optimal solution of the problem as a whole?)
✔ Is the decomposition worth the work? (Are the expenses for points 1–3 lower than the expenses of solution of the problem as a whole?—Very often yes.)

Suitable for:
✔ decomposable problems (huh, surprise!), no matter what is the type of subproblems
Dynamic Programming

Bellman’s principle of optimality:

If the point B lies on the optimal path from point A to point C, then the paths A–B and B–C are optimal as well.

Principle:

✔ Recursively solves high number of smaller subproblems

Suitable for:

1. Problem is decomposable on a sequence of decisions on various levels.
2. Several possible states on each level.
3. The decision brings the current state on next level.
4. The price of transition from one state to another is well defined.
5. Optimal decisions on one level are independent of decisions on previous levels.
Dynamic Programming: TSP Example

Traveling Salesperson Problem:
✓ Find the shortest path from depot through all the cities back to depot visiting each city only once.
✓ NP-complete

Application of DP on TSP:
✓ Decomposition to smaller subproblems:
  \[ f(i, S) \] — length of the shortest path from city \( i \) to city 1 through all the cities in \( S \) (in any order)
✓ \( f(4, \{5, 2, 3\}) \) — length of the shortest path from city 4 to city 1 through cities 5, 2, and 3
✓ TSP solution reduces to finding \( f(1, V - \{1\}) \) where \( V \) is the set of all cities (including the depot 1)
✓ Recursive transition from smaller problems to bigger ones:
  \[
  f(i, S) = \min_{j \in S} (dist(i, j) + f(j, S - \{j\}))
  \] (3)
✓ To find the optimal solution, DP often performs complete enumeration!
Best-First Search

✔ The search space forms a tree (empty solution in the root node, complete solutions in leaves).
✔ Each node $x$ has a value assigned, $f(x)$, that describes our estimate of the quality of the final complete solution using the subsolution of the node $x$.
✔ Maintains two lists of nodes: OPEN and CLOSED.
✔ Algorithm:

1. OPEN = {start}, CLOSED = {}
2. remove $x$ with best $f(x)$ from OPEN
3. for all children of $x$, $y_i \in \text{children}(x)$
   - $y_i \notin$ OPEN, $y_i \notin$ CLOSED: add $y_i$ to OPEN
   - $y_i \in$ OPEN: update $f(y_i)$ if it is better than the current one
   - $y_i \in$ CLOSED: if $f(y_i)$ is better than the current one, remove $y_i$ from CLOSED and put it in OPEN
4. repeat until OPEN is empty
Best-First Search

- The search space forms a tree (empty solution in the root node, complete solutions in leaves).
- Each node \( x \) has a value assigned, \( f(x) \), that describes our estimate of the quality of the final complete solution using the subsolution of the node \( x \).
- Maintains two lists of nodes: OPEN and CLOSED.
- Algorithm:
  1. \( \text{OPEN} = \{\text{start}\}, \text{CLOSED} = \{\} \)
  2. remove \( x \) with best \( f(x) \) from OPEN
  3. for all children of \( x \), \( y_i \in \text{children}(x) \)
     - \( y_i \notin \text{OPEN}, y_i \notin \text{CLOSED}: \text{add } y_i \text{ to OPEN} \)
     - \( y_i \in \text{OPEN}: \text{update } f(y_i) \text{ if it is better then the current one} \)
     - \( y_i \in \text{CLOSED}: \text{if } f(y_i) \text{ is better then the current one, remove } y_i \text{ from CLOSED and put it in OPEN} \)
  4. repeat until OPEN is empty

Special cases:
- **Depth-first search**: \( f(j) = f(i) - 1, j \in \text{children}(i) \), uninformed
- **Breadth-first search**: \( f(j) = f(i) + 1, j \in \text{children}(i) \), uninformed
- **Greedy search**: \( f(i) = \text{estimate of dist}(i, \text{target}) \), only short-term profit
**Branch and Bound**

- Search of the tree structured state spaces
- During the search, it maintains lower and upper limit on the optimal value of $f$
- It does not expand those nodes that do not offer chance for better solution

---

<table>
<thead>
<tr>
<th>Biologically Inspired Optimization Techniques</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td></td>
</tr>
<tr>
<td>Conventional Constructive Methods</td>
<td></td>
</tr>
<tr>
<td>Divide and Conquer</td>
<td></td>
</tr>
<tr>
<td>Dynamic Programming</td>
<td></td>
</tr>
<tr>
<td>Dynamic Programming: TSP Example</td>
<td></td>
</tr>
<tr>
<td>Best-First Search</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Branch and Bound</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A and A-star Algorithm</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conventional Generative Methods</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconventional Methods Inspired by Nature</td>
<td></td>
</tr>
</tbody>
</table>
A and A* Algorithm

A Algorithm

☑️ Best-first search with function $f$ of special meaning.
☑️ Function $f$ has the following form:

$$f(i) = g(i) + h(i),$$  \hspace{1cm} (4)

where $g(i)$ is the price of the optimal path from root node to node $i$
$h(i)$ is the price of the optimal path from node $i$ to the goal node

☑️ Pays attention to future decisions

A* Algorithm

☑️ Function $h(i)$ in A algorithm expresses the price of future decisions, but they are not known when evaluating node $i$
☑️ Function $h(i)$ is approximated with $h^*(i)$ heuristic
☑️ Function $h^*(i)$ is admissible if it is a non-negative lower estimate of $h(i)$
☑️ Algorithm A is admissible (always finds the optimal path if it exists) if it uses admissible heuristic function $h^*(i)$ and in such case it is called A* algorithm.
Conventional Generative Methods
Gradient Methods

1. Select initial point $x_0$.
2. Perform update of point $x$: $x_{k+1} \leftarrow x_k - \alpha_k \cdot \nabla f(x_k)$
3. Go to 2 until convergence.

Newton’s method: special case of gradient method

✔ Iteratively improves an approximation to the root of a real-valued function:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

✔ Finds minimum of quadratic bowl in 1 step
✔ Uses Hessian matrix to compute the update step length:

$$\alpha_k^{-1} = H(f(x_k)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_D} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_D x_1} & \frac{\partial^2 f}{\partial x_D x_2} & \cdots & \frac{\partial^2 f}{\partial x_D^2} \end{bmatrix} \begin{bmatrix} x_k \end{bmatrix}$$

(5)
Linear Programming

- Simplex method (completely different from the Nelder-Mead simplex search)
- $f$ must be linear
- Constraints in the form of linear equations and inequations
- Extremum is found on the boundaries of polyhedra given by the optimized function and the constraints.
- Unusable in BB optimization, since the function $f$ is generally nonlinear.
Exhaustive, Enumerative Search

- All possible solutions are generated and evaluated sequentially
- Often realized using depth-first or breadth-first search
- Applicable for small discrete search spaces
- For more complex tasks inappropriate—the number of possible solutions grows usually exponentially
- Continuous spaces cannot be searched exhaustively!!!
Algorithm 1: LS with First-improving Strategy

begin
\[ x \leftarrow \text{Initialize()} \]
while not TerminationCondition() do
\[ y \leftarrow \text{Perturb}(x) \]
if BetterThan($y, x$) then
\[ x \leftarrow y \]
Features:
✓ usually stochastic, possibly deterministic, applicable in discrete and continuous spaces
Local Search, Hill-Climbing

**Algorithm 1: LS with First-improving Strategy**

1. **begin**
2. \( x \leftarrow \text{Initialize}() \)
3. while not \( \text{TerminationCondition}() \) do
4. \( y \leftarrow \text{Perturb}(x) \)
5. if BetterThan\( (y, x) \) then
6. \( x \leftarrow y \)

**Features:**
- usually stochastic, possibly deterministic,
- applicable in discrete and continuous spaces

**Algorithm 2: LS with Best-improving Strategy**

1. **begin**
2. \( x \leftarrow \text{Initialize}() \)
3. while not \( \text{TerminationCondition}() \) do
4. \( y \leftarrow \text{BestOfNeighborhood}(N(x, D)) \)
5. if BetterThan\( (y, x) \) then
6. \( x \leftarrow y \)

**Features:**
- deterministic, applicable only in discrete spaces, or in discrete-real-valued spaces, where \( N(x, d) \) is finite and small
Local Search, Hill-Climbing

**Algorithm 1**: LS with First-improving Strategy

```
begin
  x ← Initialize()
  while not TerminationCondition() do
    y ← Perturb(x)
    if BetterThan(y, x) then
      x ← y
```

Features:

✔ usually stochastic, possibly deterministic, applicable in discrete and continuous spaces

The influence of the neighborhood size:

✔ Small neighborhood: fast search, huge risk of getting stuck in local optimum (in zero neighborhood, the same point is generated over and over)

✔ Large neighborhood: lower risk of getting stuck in LO, but the efficiency drops. If \( N(x, d) = S \), the search degrades to
  ✖ random search in case of first-improving strategy, or to
  ✖ exhaustive search in case of best-improving strategy.

**Algorithm 2**: LS with Best-improving Strategy

```
begin
  x ← Initialize()
  while not TerminationCondition() do
    y ← BestOfNeighborhood(N(x, D))
    if BetterThan(y, x) then
      x ← y
```

Features:

✔ deterministic, applicable only in discrete spaces, or in descretized real-valued spaces, where \( N(x, d) \) is finite and small
Local Search Demo

Local Search on Sphere Function

Local Search on Rosenbrock Function
Rosenbrock’s Optimization Algorithm

Described in [?]:

**Algorithm 3: Rosenbrock’s Algorithm**

**Input:** $\alpha > 1$, $\beta \in (0, 1)$

1. begin
2. $x \leftarrow \text{Initialize}(); x_0 \leftarrow x$
3. $\{e_1, \ldots, e_D\} \leftarrow \text{InitOrtBasis}()$
4. $\{d_1, \ldots, d_D\} \leftarrow \text{InitMultipliers}()$
5. while not TerminationCondition() do
6.   for $i=1 \ldots D$ do
7.     $y \leftarrow x + d_i e_i$
8.     if BetterThan($y, x$) then
9.       $x \leftarrow y$
10.      $d_i \leftarrow \alpha \cdot d_i$
11.     else
12.       $d_i \leftarrow -\beta \cdot d_i$
13.     if AtLeastOneSuccInAllDirs() and AtLeastOneFailInAllDirs() then
14.       $\{e_1, \ldots, e_D\} \leftarrow \text{UpdOrtBasis}(x - x_0)$
15.       $x_0 \leftarrow x$
16.   end

**Features:**
- $D$ candidates generated each iteration
- neighborhood in the form of a pattern
- adaptive neighborhood parameters
- distances
- directions

DEMO
Rosenbrock’s Algorithm Demo

Rosenbrock Method on Sphere Function

Rosenbrock Method on Rosenbrock Function
Simplex downhill search (amoeba) [?]:

**Algorithm 4: Nelder-Mead Simplex Algorithm**

```plaintext
begin
  \( (x_1, \ldots, x_{D+1}) \leftarrow \text{InitSimplex}() \)
  so that \( f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{D+1}) \)

  while not \text{TerminationCondition}() do
    \( \bar{x} \leftarrow \frac{1}{D} \sum_{d=1}^{D} x_d \)
    \( y_r \leftarrow \bar{x} + \rho(\bar{x} - x_{D+1}) \)
    if BetterThan(\( y_r, x_D \)) then \( x_{D+1} \leftarrow y_r \)
    if BetterThan(\( y_r, x_1 \)) then
      \( y_e \leftarrow \bar{x} + \chi(x_r - \bar{x}) \)
      if BetterThan(\( y_e, y_r \)) then \( x_{D+1} \leftarrow y_e \); Continue
    if not BetterThan(\( y_r, x_D \)) then
      if BetterThan(\( y_r, x_{D+1} \)) then
        \( y_{oc} \leftarrow \bar{x} + \gamma(x_r - \bar{x}) \)
        if BetterThan(\( y_{oc}, y_r \)) then \( x_{D+1} \leftarrow y_{oc} \); Continue
      else
        \( y_{ic} \leftarrow \bar{x} - \gamma(\bar{x} - x_{D+1}) \)
        if BetterThan(\( y_{ic}, x_{D+1} \)) then \( x_{D+1} \leftarrow y_{ic} \); Continue
end
```

**Features:**

- ✔ universal algorithm for BBO in real space
- ✔ in \( \mathbb{R}^D \) maintains a simplex of \( D + 1 \) points
- ✔ neighborhood in the form of a pattern (reflection, extension, contraction, reduction)
- ✔ static neighborhood parameters!
- ✔ adaptivity caused by changing relationships among solution vectors!
- ✔ slow convergence, for low \( D \) only
Nelder-Mead Simplex Search on Sphere Function

Nelder-Mead Simplex Search on Rosenbrock Function
Lessons Learned

To search for the optimum, the algorithm must maintain at least one base solution (fulfilled by all algorithms).

To adapt to the changing position in the environment during the search, the algorithm must either

- adapt the neighborhood (model) structure or parameters (as done in Rosenbrock method), or
- adapt more than 1 base solutions (as done in Nelder-Mead method), or
- both of them.

The neighborhood
- can be finite or infinite
- can have a form of a pattern or a probabilistic distribution.

Candidate solutions can be generated from the neighborhood of
- one base vector (LS, Rosenbrock), or
- all base vectors (Nelder-Mead), or
- some of the base vectors (requires selection operator).
Unconventional Methods Inspired by Nature
The Problem of Local Optimum

Goal of these algorithms:
✔ to lower the risk of getting stuck in local optimum.

How can we avoid local optimum?
### The Problem of Local Optimum

Goal of these algorithms:

✔ to lower the risk of getting stuck in local optimum.

How can we avoid local optimum?

1. Run the optimization algorithm from a different initial point.
   ✔ e.g. iterated hill-climber
The Problem of Local Optimum

Goal of these algorithms:
✔ to lower the risk of getting stuck in local optimum.

How can we avoid local optimum?

1. Run the optimization algorithm from a different initial point.
   ✔ e.g. iterated hill-climber

2. Introduce memory and do not stop the search in LO
   ✔ e.g. taboo search
The Problem of Local Optimum

Goal of these algorithms:
✔ to lower the risk of getting stuck in local optimum.

How can we avoid local optimum?
1. Run the optimization algorithm from a different initial point.
   ✔ e.g. iterated hill-climber
2. Introduce memory and do not stop the search in LO
   ✔ e.g. taboo search
3. Introduce a probabilistic aspect
   ✔ stochastic hill-climber
   ✔ simulated annealing
   ✔ evolutionary algorithms, swarm intelligence
The Problem of Local Optimum

Goal of these algorithms:
✔ to lower the risk of getting stuck in local optimum.

How can we avoid local optimum?

1. Run the optimization algorithm from a different initial point.
   ✔ e.g. iterated hill-climber

2. Introduce memory and do not stop the search in LO
   ✔ e.g. taboo search

3. Introduce a probabilistic aspect
   ✔ stochastic hill-climber
   ✔ simulated annealing
   ✔ evolutionary algorithms, swarm intelligence

4. Perform the search in several places in parallel
   ✔ evolutionary algorithms, swarm intelligence (population-based optimization algorithms)
Taboo Search

Algorithm 5: Taboo Search

1 begin
2 \( x \leftarrow \text{Initialize()} \)
3 \( y \leftarrow x \)
4 \( M \leftarrow \emptyset \)
5 \( \text{while not } \) TerminationCondition() \( \text{do} \)
6 \( y \leftarrow \text{BestOfNeighborhood}(N(y, D) - M) \)
7 \( M \leftarrow \text{UpdateMemory}(M, y) \)
8 \( \text{if BetterThan}(y, x) \text{ then} \)
9 \( x \leftarrow y \)

Meaning of symbols:

✔ \( M \) — memory holding already visited points that become taboo
✔ \( N(x, d) \cap M \) — set of states which would arise by taking back some of the previous decisions

Features:

✔ canonical version is based on the local search with best-improving strategy
✔ first-improving can be used as well
✔ difficult use in real domain, usable mainly in discrete spaces
Assuming minimization:

**Algorithm 6: Stochastic Hill-Climber**

1. begin
2. \( x \leftarrow \text{Initialize}() \)
3. while not \( \text{TerminationCondition}() \) do
4. \( y \leftarrow \text{Perturb}(x) \)
5. \( p = \frac{1}{1 + e^{\frac{f(y) - f(x)}{T}}} \)
6. if \( \text{rand} < p \) then
7. \( x \leftarrow y \)

**Features:**

- It is possible to move to a worse point *anytime*.
- \( T \) is the algorithm parameter and stays constant during the whole run.
- When \( T \) is low, we get local search with first-improving strategy.
- When \( T \) is high, we get random search.
Stochastic Hill-Climber

Assuming minimization:

**Algorithm 6: Stochastic Hill-Climber**

\[
\begin{align*}
\text{begin} & \\
x & \leftarrow \text{Initialize}() \\
\textbf{while not } & \text{TerminationCondition()} \textbf{ do} \\
y & \leftarrow \text{Perturb}(x) \\
p & = \frac{1}{1 + e^{\frac{f(y) - f(x)}{T}}} \\
\textbf{if } & \text{rand} < p \textbf{ then} \\
x & \leftarrow y
\end{align*}
\]

Features:

- It is possible to move to a worse point anytime.
- \( T \) is the algorithm parameter and stays constant during the whole run.
- When \( T \) is low, we get local search with first-improving strategy.
- When \( T \) is high, we get random search.

Probability of accepting a new point \( y \) when \( f(y) - f(x) = -13 \):

<table>
<thead>
<tr>
<th>( T )</th>
<th>( e^{-\frac{13}{T}} )</th>
<th>( p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>5</td>
<td>0.074</td>
<td>0.931</td>
</tr>
<tr>
<td>10</td>
<td>0.273</td>
<td>0.786</td>
</tr>
<tr>
<td>20</td>
<td>0.522</td>
<td>0.657</td>
</tr>
<tr>
<td>50</td>
<td>0.771</td>
<td>0.565</td>
</tr>
<tr>
<td>( 10^{10} )</td>
<td>1.000</td>
<td>0.500</td>
</tr>
</tbody>
</table>

Probability of accepting a new point \( y \) when \( T = 10 \):

<table>
<thead>
<tr>
<th>( f(y) - f(x) )</th>
<th>( e^{\frac{f(y) - f(x)}{10}} )</th>
<th>( p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-27</td>
<td>0.067</td>
<td>0.937</td>
</tr>
<tr>
<td>-7</td>
<td>0.497</td>
<td>0.668</td>
</tr>
<tr>
<td>0</td>
<td>1.000</td>
<td>0.500</td>
</tr>
<tr>
<td>13</td>
<td>3.669</td>
<td>0.214</td>
</tr>
<tr>
<td>43</td>
<td>73.700</td>
<td>0.013</td>
</tr>
</tbody>
</table>
Simulated Annealing

Algorithm 7: Simulated Annealing

begin
  x ← Initialize()
  T ← Initialize()
  while not TerminationCondition() do
    y ← Perturb(x)
    if BetterThan(y, x) then
      x ← y
    else
      p = e^(-(f(y) - f(x))/T)
      if rand < p then
        x ← y
    if InterruptCondition() then
      T ← Cool(T)
end
Algorithm 7: Simulated Annealing

begin
1. \( x \leftarrow \text{Initialize()} \)
2. \( T \leftarrow \text{Initialize()} \)
3. while not TerminationCondition() do
   4. \( y \leftarrow \text{Perturb}(x) \)
   5. if BetterThan\( (y, x) \) then
      6. \( x \leftarrow y \)
   7. else
      8. \( p = e^{-\frac{f(y) - f(x)}{T}} \)
      9. if rand < p then
         10. \( x \leftarrow y \)
   11. if InterruptCondition() then
      12. \( T \leftarrow \text{Cool}(T) \)
end

Very similar to stochastic hill-climber

Main differences:

✔ If the new point \( y \) is better, it is always accepted.
✔ Function \( \text{Cool}(T) \) is the cooling schedule.
✔ SA changes the value of \( T \) during the run:
   × \( T \) is high at beginning: SA behaves like random search
   × \( T \) is low at the end: SA behaves like deterministic hill-climber
Simulated Annealing

Algorithm 7: Simulated Annealing

begin
  x ← Initialize()
  T ← Initialize()
  while not TerminationCondition() do
    y ← Perturb(x)
    if BetterThan(y,x) then
      x ← y
    else
      p = $e^{-\frac{f(y) - f(x)}{T}}$
      if rand < p then
        x ← y
    if InterruptCondition() then
      T ← Cool(T)
  if TerminationCondition() then
    break
end

Very similar to stochastic hill-climber

Main differences:

✔ If the new point \( y \) is better, it is always accepted.

✔ Function \( \text{Cool}(T) \) is the cooling schedule.

✔ SA changes the value of \( T \) during the run:
   
   ✘ \( T \) is high at beginning: SA behaves like random search
   ✘ \( T \) is low at the end: SA behaves like deterministic hill-climber

Issues:

✔ How to set up the initial temperature \( T \) and the cooling schedule \( \text{Cool}(T) \)?

✔ How to set up the interrupt and termination condition?
### SA versus Taboo

<table>
<thead>
<tr>
<th>Feature</th>
<th>Taboo</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>When can it accept worse solution?</td>
<td>After visiting LO</td>
<td>Anytime</td>
</tr>
<tr>
<td>Deterministic/Stochastic</td>
<td>Usually deterministic (based on best-improving strategy)</td>
<td>Usually stochastic (based on first-improving strategy)</td>
</tr>
<tr>
<td>Parameters</td>
<td>Neighborhood size</td>
<td>Neighborhood size</td>
</tr>
<tr>
<td></td>
<td>Termination condition</td>
<td>Termination condition</td>
</tr>
<tr>
<td></td>
<td>Type of memory</td>
<td>Temperature</td>
</tr>
<tr>
<td></td>
<td>Memory behaviour (forgetting)</td>
<td>Cooling schedule</td>
</tr>
</tbody>
</table>
Black-box optimization:
- no information about objective function is known
- all we can do is ask the objective function to evaluate some candidate solution

Constructive vs. generative methods
- constructive methods are not suitable for BBO — require ability to evaluate partial solutions

2 sources of adaptivity for generative methods:
1. definition of local neighborhood
2. ‘population’ of candidate solutions
…next five lectures on Artificial Neural Networks with Jan Drchal.

Thank you for your attention.