3. Basic Design Issues

3.1 Representation
3.2 Fitness Assignment
3.3 Selection
3.4 Variation
3.5 Example Application: Clustering

In the Following...

...you learn:

- what the basic design choices are when implementing a randomized search algorithm;
- what general techniques are available for each of these design issues;
- how these techniques work and can be implemented;
- how these issues have been addressed in an example application.

Note: The above scheme represents an evolutionary algorithm, but also applies to other randomized search algorithms.
Search Space and Decision Space

- **Search Space** $Y$ defines the space on which the variation operators (neighborhood function, mutation, recombination, etc.) are applied.
- The **decoder function** $m: Y \rightarrow X$ defines the mapping from the search space to the objective space.
- In the evolutionary computation field, the search space is also denoted as **genotypic search space** and the decision space as **phenotypic search space**.

Why Distinguishing Search and Decision Space?

- Ideally, search space and decision space are identical, i.e., $Y = X$ and $m(y) = y$ for all $y \in Y$.
  - **Examples:** $f_{ONEMAX}$, $f_{NEEDLE}$
    
    $Y = X = \{0,1\}^n$

    where each solution is represented by a bitvector and can be implemented via an array of length $n$.

- For many applications, though, the solutions in $X$ need to be appropriately encoded in order to process them on a computer, e.g., if $X = \mathbb{R}$. In other words, $Y$ is the representation of $X$ in the computer.
  - **Examples:** graph problems, scheduling, symbolic regression, etc.

Types of Encoding

- **Vectors:**
  - usually of fixed length
  - usually implemented by means of arrays or lists
  - often represent n-tuples of binary, integer, or real values

- **Trees:**
  - size usually not fixed
  - usually implemented by means of list-based data structures
  - often represent symbolic expressions such as LISP programs

- **Other Types:**
  - matrices, general graphs, etc.
  - often hybrid representations are used (e.g., binary vector + matrix)

Example: Binary Vector Encoding

**Given:** graph

**Goal:** find minimum subset of nodes such that each edge is connected to at least one node of this subset (minimum vertex cover)

nodes selected? 1 0 1 1 1 0 ...
### Example: Integer Vector Encoding

**Given:** graph, k colors  
**Goal:** assign each node one of the k colors such that the number of connected nodes with the same color is minimized (graph coloring problem)

![Graph Coloring Diagram](image)

### Example: Real Vector Encoding

\[
G^2(\vec{x}) = \left| \sum_{i=1}^{n} \cos^4(x_i) - 2 \prod_{i=1}^{n} \cos^2(x_i) \right| 
\]

<table>
<thead>
<tr>
<th>parameters</th>
<th>values</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>(x_2)</td>
<td>0.53</td>
<td></td>
</tr>
<tr>
<td>(x_3)</td>
<td>1.03</td>
<td></td>
</tr>
<tr>
<td>(x_4)</td>
<td>3.25</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x_n)</td>
<td>9.83</td>
<td></td>
</tr>
</tbody>
</table>

[Michalewicz, Fogel (2000)]

### Tree Representations

**Trees...**
- are connected, acyclic (undirected) graphs  
  here: rooted, ordered trees with directed edges  
- are flexible in size  
- are mainly used to represent symbolic expressions or programs (therefore the term Genetic Programming)

![Tree Diagram](image)

**Note:**
- trees can be implemented in different ways (see data structures lecture)  
- lists are specific trees where each node except of the leaf has exactly one successor (good to represent size-flexible vectors)

### Type of Tree Representations

**Usual usage:**
- inner nodes = operators (each operator takes a certain number of arguments, the arguments are the children / immediate successors in the tree)  
- leaves = arguments (constants, variables)

Both operators and arguments define the space of possible trees and need to be specified in advance

**Examples:**
- Boolean expressions:
  - set of operators \(S_O = \{AND_2, OR_2, NOT_1\}\)  
  - set of arguments \(S_A = \{x_1, x_2, \ldots, x_k\}\) with \(x_i \in \{0, 1\}\)
- Continuous functions:
  - set of operators \(S_O = \{\sin_1, \cos_1, +_2, -_2, \cdot_2, /_2, \ldots\}\)  
  - set of arguments \(S_A = \{x_1, x_2, \ldots, x_k\}\) with \(x_i \in \mathbb{R}\)
- Programs:
  - set of operators \(S_O = \{IF_2, WHILE_2, FOR_2, \ldots\}\)  
  - set of arguments \(S_A = \{x_1, x_2, \ldots, x_k\} \cup \{0, 1\}^l\) with \(x_i \in \{0, 1\}\)
What Defines the Search Space?

If using a tree representation, the following needs to be specified:

- the set of operators
- for each operator, the number of arguments and their order
- if data types are used, for each argument the data type
- the set of variables and constants
- if data types are used, for each variable/constant its type
- an interpretation function that, given for variable a specific value, ‘executes’ the tree (required for fitness evaluation)

All trees that fulfill the above specifications are members of the genotypic search space.

**Tree Example: Parking a Truck**

[Koza (1992)]

**Search Space for the Truck Problem**

**Operators:**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLUS(a,b)</td>
<td>returns a+b</td>
</tr>
<tr>
<td>MINUS(a,b)</td>
<td>returns a-b</td>
</tr>
<tr>
<td>MUL(a,b)</td>
<td>returns a*b</td>
</tr>
<tr>
<td>DIV(a,b)</td>
<td>return a/b, if b &lt;&gt; 0, else 1</td>
</tr>
<tr>
<td>ATG(a,b)</td>
<td>returns atan2(a,b), if a&lt;&gt;0, else 0</td>
</tr>
<tr>
<td>IFLTZ(a,b,c)</td>
<td>returns b, if a&lt;0, else returns c</td>
</tr>
</tbody>
</table>

**Arguments:**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>position x</td>
</tr>
<tr>
<td>Y</td>
<td>position y</td>
</tr>
<tr>
<td>DIFF</td>
<td>cab angle d</td>
</tr>
<tr>
<td>TANG</td>
<td>trailer angle t</td>
</tr>
</tbody>
</table>

**Decision space:** set of symbolic expression using the above operators and arguments.
Example Solution: Tree Representation

\[
\begin{array}{c}
\text{MULT} \\
\text{MINUS} \quad \text{PLUS} \\
X \quad \text{DIFF} \quad Y \quad \text{TANG}
\end{array}
\]

encodes the function (symbolic expression): \( u = (x - d) \ast (y + t) \)

Properties of Representations

- completeness: \( \forall x \in X \exists y \in Y : x = m(y) \)
- uniformity: \( \forall x \in X : |\{ y \mid m(y) = x \}| = c \) where \( Y \) is finite
- redundancy: \( r = \log_2 |Y| - \log_2 |X| \) where \( X, Y \) are finite
- feasibility: \( \forall y \in Y : m(y) \in X \)
- locality: \( \forall y, y', y'' \in Y : d(y, y') < d(y, y'') \Leftrightarrow d(m(y), m(y')) < d(m(y), m(y'')) \)

Illustration of Representation Properties

- complete
- uniform (c=2)
- redundant (r=1)
- feasible
- locality within Hamming distance 1

In general, one is interested in a complete, uniform, non-redundant, feasible representation preserving locality. However, if not all of these criteria are met does not necessarily imply that the performance of the search algorithm is negatively affected.
Improving Locality

Question: Why is locality important?

Answer: Effects in the search space and the decision space should be highly correlated.

Example: \( X = \{0, 1, \ldots, 2^n - 1\}, Y = \{0, 1\}^3 \)

1. \( X \) binary numbers
2. \( Y \) gray code

<table>
<thead>
<tr>
<th>X</th>
<th>Binary Numbers</th>
<th>Gray Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>011</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>010</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>111</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>101</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>100</td>
</tr>
</tbody>
</table>

- One-bit flip cannot generate both direct neighbors.
- One-bit flip can generate both direct neighbors (but may still make large jumps).

X: binary numbers
Y: gray code

The Gray Code

A Gray Code is a binary encoding that ensures that the Hamming distance between two consecutive neighbors is always 1.

- This does not necessarily mean that locality is preserved.

Recursive method for determining a Gray Code:

**Goal:** encode integers \(0, 1, \ldots, 2^n - 1\) as \(a_0, a_1, \ldots, a_{2^n - 1}\)

**Procedure:**
1. if \( n = 1 \) then
2. set \( a_0 = 0 \) and \( a_1 = 1 \)
3. else
4. recursively encode \(0, 1, \ldots, 2^{n-1} - 1\) as \(a'_0, a'_1, \ldots, a'_{2^{n-1} - 1}\)
5. choose the following mapping:
   - \( a_0 = 0a'_0 \)
   - \( a_1 = 0a'_1 \)
   - \( a_{2^{n-1}-1} = 0a'_{2^{n-1}-1} \)
   - \( a_{2^{n-1}} = 1a'_{2^{n-1}} \)
   - \( a_{2^n-1} = 1a'_{2^n-1} \)
6. end if

Reducing Redundancy (Example: TSP)

1. **Matrix representation:**
   - binary \( n \times n \) matrix \( M \) with \( \pi(i) = j \iff M(i, j) = 1 \)
   - many matrices are infeasible (unless repair mechanism is used)
   - the genotypic search space is large: \( |Y| = 2^{n \times n} \)

2. **Integer vector representation:**
   - vector \((p_1, p_2, \ldots, p_n) \in \{2, \ldots, n\}^{n-1} \) with \( \pi(i) = j \iff p_i = j \)
   - many vectors are infeasible (unless repair mechanism is used)
   - the genotypic search space is large: \( |Y| = (n - 1)^{n-1} \)

3. **Bit vector representation:**
   - each permutation is assigned a unique number
   - all vectors are feasible
   - no redundancy: \( |Y| = \log(n - 1)! \)
   - mapping function difficult to compute
   - locality is not preserved

Bio-inspired Optimization and Design

3. Basic Design Issues
   3.1 Representation
   3.2 Fitness Assignment
   3.3 Selection
   3.4 Variation
   3.5 Example Application: Clustering
Fitness Assignment: General Remarks

Fitness = scalar value representing quality of an individual (usually)

The simple case:

\[ F_i = f(m(y_i)) \]

More difficult cases:
- “informal” objectives (simulations, experiments)
- multiple optima need to be approximated (diversity)
- local search methods are integrated (hybridization)
- multiple objectives have to be considered (Section 4.1)
- constraints are involved which have to be met (Section 4.2)

“Informal” Objective Functions

Simulations / experiments:
- The objective function is difficult to formalize, i.e., the available models are not accurate enough.
- Examples: parking a truck, training a robot

→ How to design the simulation / experiment?

Competitive fitness evaluation:
- The fitness of an individual depends on (some of) the other individuals currently stored in the memory
- Example: iterated prisoner dilemma

→ How to carry out the competition?

Parking a Truck

Steering angle \( u \)

Dock \((0,0)\)

Trailer

Position \((x, y)\)

Constant speed

Goal: find function \( c \) with \( u = c(x, y, d, t) \)

Truck: Simulation

Given: start conditions \( x, y, t \) (implicitly: \( d = 0 \))

Algorithm:
1. time runs out
2. the trailer crashes into the loading dock \((x=0)\)
3. target state reached

\((x < 0.1m, y < 0.42m, t < 0.12)\)
### Truck: Fitness Assignment

**Given:**
- eight start conditions \((x_1, y_1, t_1), \ldots, (x_8, y_8, t_8)\)

**Algorithm:**
- fitness \(F = 0\)
  - for each start condition do
    - run simulation and obtain final \(x, y, t\)
    - \(F = F + x^2 + 2y^2 + \frac{40}{\pi}t\)
  - end

**Note:** Fitness is to be minimized here!

### Learning Obstacle Avoiding Behavior

**Goal:**
- find function \(c\) for the motor speeds \(M_1, M_2:\)
  \[(M_1, M_2) = c(S_0, \ldots, S_7)\]

### Robot: Characteristics

**Input:**
- 8 proximity sensor measurements
  - \(S_0, S_1, \ldots, S_7 \in \{0,1, \ldots, 1023\}\)
  - higher values = closer to an object

**Output:**
- 2 motor speed settings
  - \(M_1, M_2 \in \{0,1, \ldots, 15\}\)
  - higher values = higher speed

**Training environment:**

### Robot: Fitness Assignment

**Given:**
- robot is standing at an arbitrary position within the environment

**Algorithm:**
- retrieve sensor measurements \(S_0, S_1, \ldots, S_7\) from the robot
- determine the speeds for both motors by applying the solution under evaluation:
  \[(M_1, M_2) = c(S_0, S_1, \ldots, S_7)\]
- run robot for 400ms with motorspeeds \(M_1, M_2\)
- retrieve sensor measurements \(S'_0, \ldots, S'_7\)
- fitness \(F = 16(M_1 + M_2 - |M_1 - M_2|) - (S'_0 + S'_1 + \ldots + S'_7)\)

- high speed going straight far away from any obstacle / wall
Competitive Fitness Assignment

**Main idea:** The fitness of an individual depends on other individuals... (e.g., used in multiobjective and multimodal optimization)

**Example:** evolving game strategies

Strategy = function that maps one game situation into another one (one legal move)

**TicTacToe:**

- **current state** $S$
- **next move** red
- **next state** $S'$

Sought: function $MOVE_{red}$ with $S' = MOVE(S)$ for all possible game states where it is red's turn

Variants of Competitive Fitness Assignment

Simple population

Two populations (coevolution)

Coevolution = two, synchronized evolutionary algorithms (exchange only for fitness assignment)

- **pp 1**
  - fitness = $1/2$
  - principle: individual to be evaluated plays against selected competitors

- **pp 2**
  - fitness = $2/3$
  - to be evaluated randomly selected competitors

The Prisoner’s Dilemma

questioned separately
**Pay-Off Matrix**

<table>
<thead>
<tr>
<th></th>
<th>He Denies</th>
<th>He Confesses</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>You Deny</strong></td>
<td>Both serve six months</td>
<td>He goes free; you serve ten years</td>
</tr>
<tr>
<td><strong>You Confess</strong></td>
<td>He serves ten years; you go free</td>
<td>Both serve five years</td>
</tr>
</tbody>
</table>

**Iterated Prisoner’s Dilemma:** sum of payoffs for \( n \) subsequent games

Strategy = function which takes the moves (of both players) of the previous \( k \) games as input and outputs the next move for one player

---

**Multiple Optima**

**Goal:** find multiple optima

Multiple objectives

single objective

**Problem:** genetic drift (tendency to converge to single optima)

**Idea:** incorporate density information into fitness

---

**The Problem: Genetic Drift**

Genetic drift denotes a phenomenon in Biology where random changes in the allele frequency (allele = “different values a gene can take”) can observed due to sampling errors in finite small populations.

Here: we assume that only selection takes place (mating selection = binary tournament, environmental selection = offspring population replaces old population)

Observation: the smaller the population size, the less iterations are required until the entire population contains only copies of the same solution (only As or only Bs)

---

**The Effect of Genetic Drift**

[Goldberg (1989)]
The Principle of Density Estimation

General idea:
- take information about how close individuals are to each other into account
- compute the “density” $D_i$ around a given individual $y_i$; the larger $D_i$ the more crowded is the region around $y_i$

Remark: The density can be calculated either in the search space, in the decision space, or in the objective space. In single objective optimization, usually the search space is considered, while in multiobjective optimization in general the objective space is of interest.

A Possible Solution: Fitness Sharing

Idea:
- the density is, roughly speaking, antiproportionate to the sum of the distances to the other individuals in the population
- decrease an individual’s fitness the more individuals are close to it

Approach:
- kernel function $h$ defined on the basis of a distance metric $d$:

$$ h(d(x_i, x)) = \begin{cases} 
1 - (d(x_i, x)/\sigma_{\text{share}}) & \text{if } d(x_i, x) < \sigma_{\text{share}} \\
0 & \text{else} 
\end{cases} $$

where $\alpha$ (usually set to 1) and $\sigma_{\text{share}}$ are user-defined
- modified fitness where $D_i$ is the density around individual $x_i$:

$$ D_i = \sum_{x \in M} h(d(x_i, x)) $$
and $F_i = F_i' / D_i$ where $F_i'$ is the original fitness value
The Effect of Fitness Sharing (Cont’d)

Types of Diversity Preservation Techniques

**Kernel**
- density estimate = sum of \( f \) values where \( f \) is a function of the distance

**Nearest neighbor**
- density estimate = volume of the sphere defined by the nearest neighbor

**Histogram**
- density estimate = number of solutions in the same box

Density Estimation Approaches

1. **Nearest neighbor**
   - \( D_i = \min_{x \in M} d(x, x_i) \)
   - where \( x_i \) is the nearest individual to \( x \) in \( M \)

2. **Kernel method**
   - \( D_i = \sum_{x \in M} \frac{1}{d(x, x_i)} \)
   - \( d \) is distance function

3. **Histogram method**
   - uses grid in objective space, e.g.,
     - \( f(x_1) \): number of individuals in the same box
     - \( f(x_2) \): number of individuals

Hybridization: General Considerations

**Idea:** Combine a general global search strategy such as an evolutionary algorithm with a problem-specific heuristic or deterministic local search strategy

→ often the key to success...

**Approaches:**
- **Baldwinian scheme:** each time a solution \( y \) in the RSA is evaluated,
  1. first the deterministic local search algorithm is applied with the initial solution \( y \), and then
  2. the resulting solution \( y' \) is evaluated and the corresponding objective function value is returned, i.e., \( f(y) = f(y') \)
- **Lamarckian scheme:** each time a new solution \( y \) is created in the RSA,
  1. first the deterministic local search algorithm is applied with the initial solution \( y \), and then
  2. \( y \) is replaced by the resulting solution \( y' \) (initial population, offspring)
**The Concept of Hybridization**

![Diagram of hybridization](image)

**A Usual Problem: Using Time Resources Efficiently**

- **GSA:** Global search algorithm
- **PLSA:** Parameterized local search algorithm

controlled by the parameter $p$
(e.g., the size of the neighborhood or the maximum of iterations of the LSA):

$$p_{\text{min}} \leq p \leq p_{\text{max}}$$

How to generate maximum quality solution in given PLSA time budget?

**Simulated Heating: Underlying Idea**

Start with low $p$ and systematically increase $p$ over time

**Phase 1:**
Focus on GSA (exploration)

**Phase 2:**
Focus on PLSA (exploitation)

Adaptation function is called **simulated heating scheme**
(by analogy to simulated annealing)

- [Zitzler et al. (2000)]

**Simulated Heating: General Scheme**

- **Input:** $N$ (size of the solution candidate set)
- $T_{\text{max}}$ (maximum time budget)
- $s$ (best solution found)

**Step 1:** **Initialization:** Create an initial multi-set $M$ containing $N$ randomly generated solution candidates. Set $T = 0$ (time used) and $t = 0$ (iterations performed).

**Step 2:** **Parameter adaptation:** Choose local search parameter $p$ according to a given scheme $H$: $p = H(t)$.

**Step 3:** **Local search:** Apply local search algorithm $L$ with parameter $p$ to each $y_i \in M$ and assign it a quality (fitness) $F_i$. Set $T = T + N \cdot C(p)$.

**Step 4:** **Termination:** If $T > T_{\text{max}}$ then go to Step 6.

**Step 5:** **Global search:** Based on $M$ and the fitness values $F_i$, generate a new set $M'$ of solution candidates using the global search algorithm $G$. Set $M = M'$ and increase the iteration counter $t$. Go to Step 2.

**Step 6:** **Output:** Apply $L$ with parameter $p_{\text{max}}$ to the best solution in $M$; the resulting solution $y$ is the outcome of the algorithm.
Static Heating Schemes

Given: parameters $p_1, p_2, ..., p_n$ in increasing order

1. Fixed number of RSA iterations $t_p$ per parameter $p_i$;
   - $T_{\text{max}} \geq t_p NC(p_1) + t_p NC(p_2) + \ldots + t_p NC(p_n)$
   - $t_p = \left\lceil \frac{T_{\text{max}}}{N \sum_{i=1}^n C(p_i)} \right\rceil$

2. Fixed amount of time $T_p$ per parameter $p_i$;
   - $T_p = \frac{T_{\text{max}}}{n}$ and therefore $t_i NC(p_i) \leq T_p \ \forall i = 1, \ldots, n$
   - $t_i = \left\lceil \frac{T_{\text{max}}}{n C(p_i)} \right\rceil$

$T_{\text{max}}$ = maximum time resources  
$C(p_i)$ = time needed to run local search algorithm with parameter $p_i$

Dynamic Heating Schemes

Given: parameters $p_1, p_2, ..., p_n$ in increasing order

1. Fixed number of RSA iterations:
   - Use next parameter value, if the quality of the best solution in $M$ has not improved for $t_{\text{stag}}$ RSA iterations

2. Fixed amount of time:
   - Use next parameter value, if the quality of the best solution in $M$ has not improved for $T_{\text{stag}}$ time units

Application Study (Details Omitted)

Application benchmark: Construct schedule for digital signal processor such that the overall buffer memory size is minimized

Sample-rate conversion from a CD player to a DAT player

Setting:
- Five parameter values: $p_1, p_2, p_3, p_4, p_5$
- Time budget: $T_{\text{max}} = 5h$ (Sun Ultra 60)
- Stagnation parameters: $t_{\text{stag}} = 10$ and $T_{\text{stag}} = 900s$
- Population size: $N = 100$

Results I: Quality of The Best Solution Found

keeping $p$ constant

<table>
<thead>
<tr>
<th>$p$</th>
<th>Quality of The Best Solution Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (minimum)</td>
<td>0.3394</td>
</tr>
<tr>
<td>153</td>
<td>0.3308</td>
</tr>
<tr>
<td>305</td>
<td>0.3637</td>
</tr>
<tr>
<td>457</td>
<td>0.3622</td>
</tr>
<tr>
<td>612 (maximum)</td>
<td>0.3692</td>
</tr>
</tbody>
</table>

static heating

<table>
<thead>
<tr>
<th>$p$</th>
<th>(fixed number of iterations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 153, 305, 457, 612</td>
<td>0.3558</td>
</tr>
<tr>
<td>1, 31, 62, 92, 123,153</td>
<td>0.2848</td>
</tr>
<tr>
<td>1, 153, 305, 457, 612,123,153</td>
<td>0.3024</td>
</tr>
<tr>
<td>1, 31, 62, 92, 123,153</td>
<td>0.2609</td>
</tr>
</tbody>
</table>

dynamic heating

<table>
<thead>
<tr>
<th>$p$</th>
<th>(fixed amount of time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 153, 305, 457, 612</td>
<td>0.2992</td>
</tr>
<tr>
<td>1, 31, 62, 92, 123,153</td>
<td>0.2739</td>
</tr>
<tr>
<td>1, 153, 305, 457, 612</td>
<td>0.2985</td>
</tr>
<tr>
<td>1, 31, 62, 92, 123,153</td>
<td>0.2558</td>
</tr>
</tbody>
</table>
Selection: General Remarks

- Selection is the major determinant for specifying the trade-off between exploitation and exploration.

- Two types of selection schemes can be distinguished:
  1. **Stochastic selection** consisting of
     - **Sampling rate assignment**
       \[ Q_i = \text{probability that individual } i \text{ is chosen} \]
     - **Sampling**
       choose \( N \) individuals according to their sampling rates
  2. **Deterministic selection**

- Mating selection (selection for variation) is usually implemented using a stochastic scheme, while environmental selection (selection for survival) is often based on a deterministic scheme (exception: Metropolis, Simulated Annealing)

Sampling Rate Assignment

- **Fitness proportionate**: (scaling dependent, e.g., adding a constant factor changes the sampling rates)
  \[ Q_i = \frac{F_i}{\sum F_k} \]
  Example: \( F_1 = 1, F_2 = 2, F_3 = 3 \)
  \[ Q_1 = \frac{1}{6}, Q_2 = \frac{1}{3}, Q_3 = \frac{1}{2} \]

- **Rank-based**: (scaling independent)
  sort population; \( R_i \) = rank of individual within resulting order
  - **Linear**:
    \[ P_i = R_i + \alpha \]
  - **Quadratic**:
    \[ P_i = R_i^2 + \alpha \]
  - **Geometric**:
    \[ P_i = \alpha (1 - \alpha R_i) \]
  - **Exponential**:
    \[ P_i = 1 - e^{-R_i} \]
  \[ Q_i = \frac{P_i}{\sum P_k} \]

- **Threshold**:
  \[ Q_i = \begin{cases} 
  1/T & \text{if individual } i \text{ is among the } T \text{ best individuals} \\
  0 & \text{else}
  \end{cases} \]
**Sampling Methods: Principle**

- **Roulette wheel**
  - Spin 4 times

- **Stochastic universal sampling (SUS)**
  - Spin once

**Sampling Methods: Details**

- **Roulette wheel**
  - May lead to high variance in the selection outcome

- **Stochastic universal sampling (SUS)**
  - Lower variance: the number of times a specific individual is selected varies only by one

**Tournament Selection**

- Integrated sampling rate assignment and sampling

1. Uniformly choose $T$ individuals at random independently of fitness
2. Compare fitness and copy best individual in mating pool

$T = \text{tournament size (binary tournament selection means } T=2)$

**Deterministic Selection Schemes**

The following notation has been widely used in the context of evolution strategies; we here use it for classifying environmental selection:

\[
(\mu, \lambda) \quad \text{or} \quad (\mu + \lambda)
\]

- $\mu$ denotes the population size, $1 \leftarrow \mu$
- $\lambda$ denotes the number of offspring, i.e., $1 \leftarrow \lambda = \lambda$
- $\mu$ means the new population is formed by the best $\mu$ individuals from $M^n$
- $\lambda$ means the new population is formed by the best $\lambda$ individuals from $M + M^n$ (union of parents and children)

Examples:

- $(1+1) \Rightarrow \text{local search}$
- $(\mu + 1) \Rightarrow \text{steady-state evolutionary algorithm (1 individual/generation)}$
- $(\mu, \mu) \Rightarrow \text{regular genetic algorithm}$

Note: in evolution strategies, usually uniform random selection is used.
Properties of Selection Schemes

- Takeover time = expected number of generations required until the population contains only copies of the best individual at the beginning (no variation takes place)

- Selection intensity =

\[ I = \frac{(\overline{F}_{sel} - \overline{F})}{\sigma} \]

where

- \( \overline{F}_{sel} \) = average fitness in population after selection
- \( \overline{F} \) = average fitness in population before selection
- \( \sigma \) = standard deviation of fitness in population before selection

Many other properties have been suggested and used in theoretical investigations. However, the effect of certain properties on the performance of the search algorithm is difficult to capture...

Variation: General Remarks

- Variation aims at generating promising new solutions resp. individuals on the basis of those individuals selected for mating.

- Usually, two types of variation operators:

  - mutation: \( mut: Y \rightarrow Y \)
  - recombination: \( recomb: Y^r \rightarrow Y^s \) where \( r \geq 2 \) and \( s \geq 1 \)

- The choice of the operators always depends on the problem and the chosen representation; however, there are some operators that are applicable to a wide range of problems and tailored to standard representations such as vectors, trees, etc.

- Popular standard operators will be discussed in the following.

Bio-inspired Optimization and Design

Eckart Zitzler

3. Basic Design Issues

3.1 Representation
3.2 Fitness Assignment
3.3 Selection
3.4 Variation
3.5 Example Application: Clustering

Mutation: Guidelines

Question: What properties should a mutation operator have?

- The mutation operator can be seen as the counterpart to the neighborhood function in local search; however, there usually two differences:

  1. Every solution can be generated from every other solutions by means of mutation with a probability greater than 0.

  2. \( d(x, x') < d(x, x'') \Rightarrow Prob[mut(x) = x'] > Prob[mut(x) = x''] \)

- The above two criteria represent recommendations that not always can be fulfilled in practice.
Mutation: Binary Search Space

In the case that \( Y = \{0,1\}^n \), the most commonly used mutation strategy is to flip each bit independently with probability \( p_m \).

\[ \text{Bit flip mutation: } \binom{n}{k} p_m^k (1-p_m)^{n-k} \]

This follows a binomial distribution, and therefore, the expected number of bits mutated is \( n \cdot p_m \).

With \( p_m = \frac{1}{n} \), the expected number of mutated bits is one per individual. This is usually used for \((\lambda+\mu)\) strategies, in the case of \((\lambda+\mu)\) environmental selection; a fixed mutation rate such as \( p_m = \frac{1}{n} \) often yields better results.

Mutation: Discrete Search Spaces In General

Assume that \( Y = Y_1 \times Y_2 \times \ldots \times Y_k \), where \( Y_i \) is finite, i.e., each individual is a vector of \( n \) elements. A straightforward extension of the bit flip mutation is to replace the \( i \)-th element of the vector with probability \( p_m \) by another element in \( Y_i \).

\[ \text{Vector mutation: } Y = Y_1 \times Y_2 \times \ldots \times Y_k \]

\[ y \leftarrow \text{replace with } \text{prob } p_m \text{ with any } y' \in Y \nabla Y \]

For permutations, though, this mutation operator does not preserve the permutation property. Several other operators have been proposed for this purpose.

Mutation Operators for Permutations

- **Swap:**
  \[ \begin{array}{cccccc}
  1 & 2 & 3 & 4 & 5 & 6 \\
  \end{array} \]
  \[ \rightarrow \begin{array}{cccccc}
  1 & 4 & 3 & 2 & 5 & 6 \\
  \end{array} \]

- **Scramble:**
  \[ \begin{array}{cccccc}
  1 & 2 & 3 & 4 & 5 & 6 \\
  \end{array} \]
  \[ \rightarrow \begin{array}{cccccc}
  1 & 3 & 4 & 2 & 5 & 6 \\
  \end{array} \]

- **Invert:**
  \[ \begin{array}{cccccc}
  1 & 2 & 3 & 4 & 5 & 6 \\
  \end{array} \]
  \[ \rightarrow \begin{array}{cccccc}
  1 & 4 & 3 & 2 & 5 & 6 \\
  \end{array} \]

- **Insert:**
  \[ \begin{array}{cccccc}
  1 & 2 & 3 & 4 & 5 & 6 \\
  \end{array} \]
  \[ \rightarrow \begin{array}{cccccc}
  1 & 4 & 2 & 3 & 5 & 6 \\
  \end{array} \]

Mutation: Real Vectors

- In principle, real vectors also have a discrete representation on the computers and therefore could be treated as integer vectors. However, replacing a real value by an arbitrary one is usually not effective.

- An alternative (many other mutation operators for real vectors exist):
  \[ (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \]
  \[ x_i = x_i + N(0, \sigma^2) \]
  where \( N(0, \sigma^2) \) is a random number drawn using a Gaussian distribution with mean 0 and standard deviation \( \sigma \).

Note that the mutation rate \( p_m \) is replaced here by a vector of step sizes \((\sigma_1, \sigma_2, \ldots, \sigma_n)\), sometimes, there is only cut global step size \( \sigma \) with \( \sigma_i = \sigma \) for all \( i \).
Mutation Operators on Trees: Grow

Mutation Operators on Trees: Shrink

Mutation Operators on Trees: Switch

Mutation Operators on Trees: Cycle
**Recombination: Guidelines**

**Question:** What properties should a recombination operator have?

- The recombination operator distinguishes evolutionary algorithms from other randomized search algorithms; similarly to mutation operators, a desirable property of a recombination operator is:

\[ x'' = \text{recomb}(x, x') \Rightarrow d(x, x'') \leq d(x, x') \land d(x', x'') \leq d(x, x') \]

- As before, this criterion represents a recommendation that not always can be fulfilled in practice.

**Vector Recombination**

**One-point crossover:**

```
1 1 0 0 1 0
1 0 1 0 0 1
```

```
1 1 0 0 0 1
```

**N-point crossover:**

```
1 1 0 0 1 0
1 0 1 0 0 1
```

```
1 0 1 0 1 0
N = 2
```

**Uniform crossover:**

```
1 1 0 0 1 0
1 0 1 0 0 1
```

```
1 0 0 0 0 0
```

*For each position it is determined separately (at random) whether the value is copied from parent 1 or parent 2*

**Recombination of Trees**

- **MULT**
- **PLUS**
- **MINUS**
- **TANG**
- **DIFF**

- **exchange**

---

**Recombination: Real Vectors**

One possibility in the case of real vectors is to use any of the above general vector recombination operators. Alternatively, a number of operators can be used that create a new real number using the two values from the parents. A commonly used operator is:

- **Intermediate recombination**

  parents: \((x_1, x_2, ..., x_n), (x'_1, x'_2, ..., x'_n) \in \mathbb{R}^n\)

  children: \((x^1_1, x^1_2, ..., x^1_n) \in \mathbb{R}^n\) with

  \[ x^1_i = x_i + u \cdot (x'_i - x_i) \text{ where } u \text{ is a uniform random variable over } [0, 1] \]

- Several other operators on real vectors were suggested, which however, cannot be discussed here.
3. Basic Design Issues

3.1 Representation
3.2 Fitness Assignment
3.3 Selection
3.4 Variation
3.5 Example Application: Clustering (not part of the exam)

A Hybrid Evolutionary Algorithm for Biclustering

Outline:
- Based on a previously proposed greedy strategy for biclustering
- Uses an evolutionary algorithm for exploring the space of submatrices
- The greedy heuristic is integrated as local search method

References