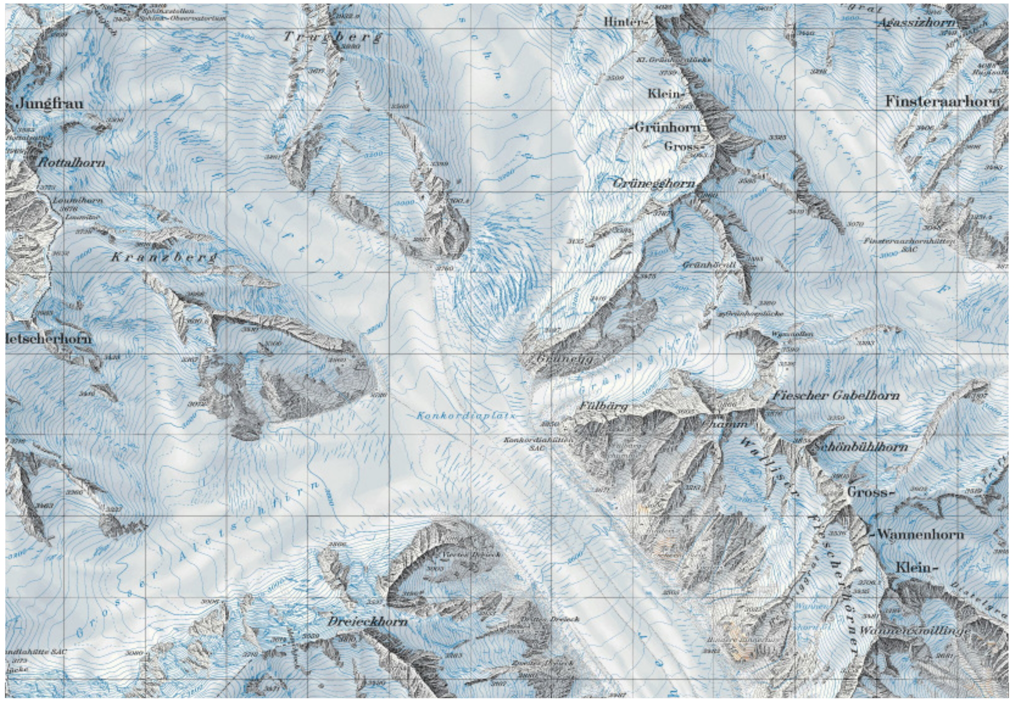
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

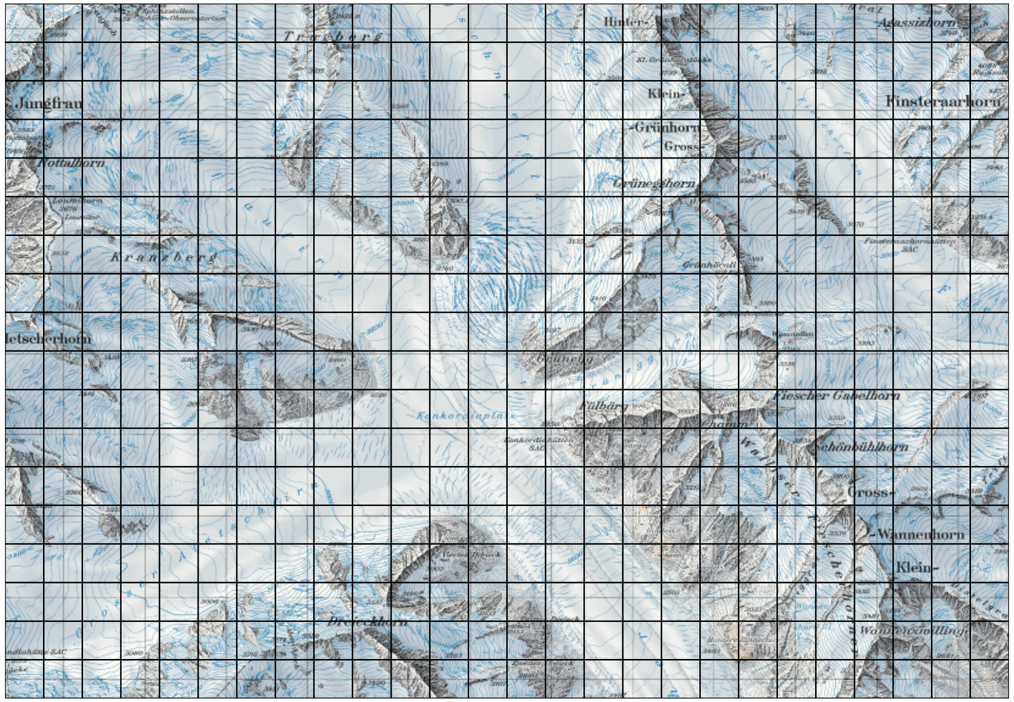
TIK
Computer Engineering
and Networks Laboratory

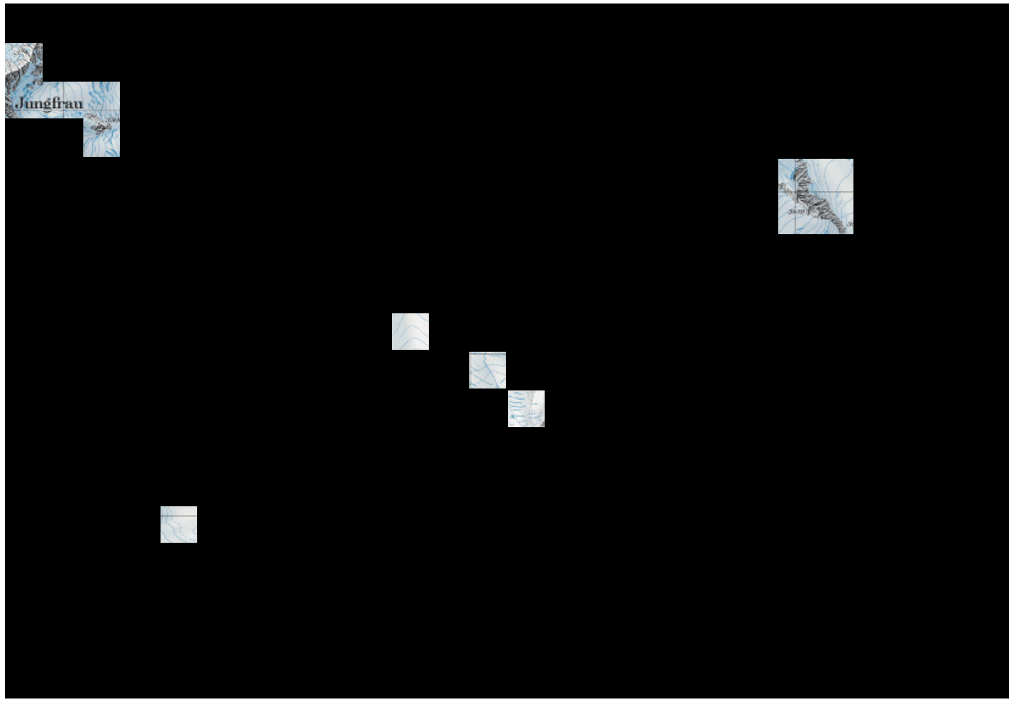
Bio-inspired Optimization and Design

Eckart Zitzler

2. Randomized Search Algorithms



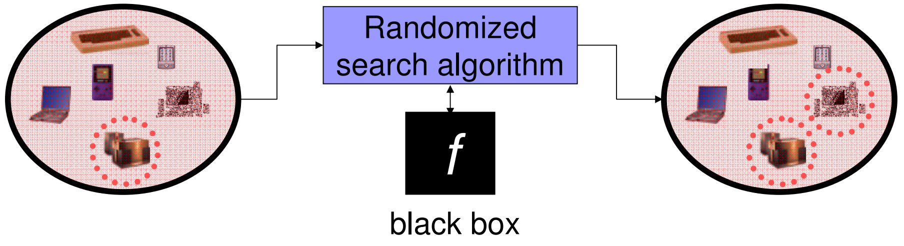




Randomized Search Algorithms in a Nutshell

Idea: find good solutions without investigating all solutions

Assumption: better solutions can be found in the neighborhood of good solutions (the problem is structured)



$t = 1$:
(randomly) choose a solution x_1 to start with

$t \rightarrow t + 1$:
(randomly) choose a solution x_{t+1} using solutions $x_1, \ldots, x_t$

In the Following...

...you learn:

- what is understood under the term randomized search algorithm;
- how popular randomized search heuristics work;
- that specific biologically inspired optimization methods, namely evolutionary algorithms, represent a general framework for randomized search heuristics.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

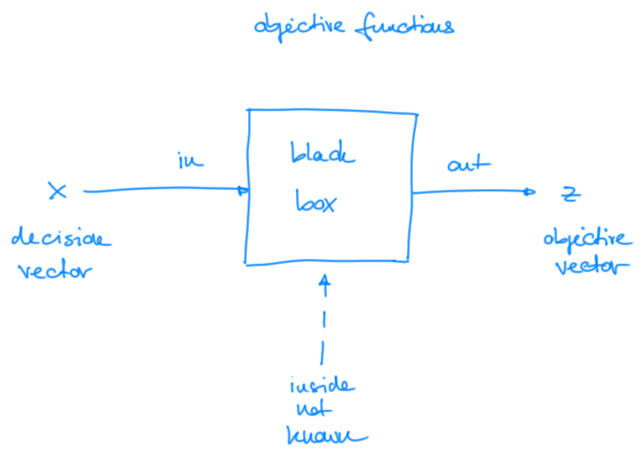
TIK
Computer Engineering
and Networks Laboratory

Bio-inspired Optimization and Design

Eckart Zitzler

2. Randomized Search Algorithms

Black Box Scenarios



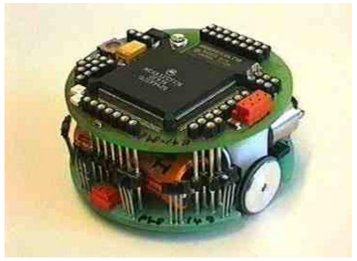
Black Box Optimization

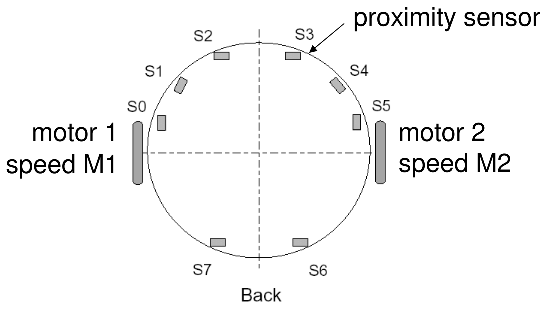
- In a black box scenario, the mapping function $f: X \rightarrow Z$ is treated as a black box like an executable procedure in a computer program for which the code is not known or not visible.
  - ⇒ unclear whether optimal solution has been found, unless the entire decision space has been sampled
- Black box scenarios arise whenever the objective functions
  1. are not given in closed from, i.e., if the objective function values are determined via complex computations, simulations, or experiments; or
  2. are highly complex and/or poorly understood.
- In real-world applications, often partial black box scenarios emerge where to some extent knowledge about the underlying problem is available.

Example: Training a Robot to Avoid Obstacles

Goal: find a symbolic expression for a function *c* that determines the speed of the two motors depending on the proximity sensor inputs such that the robot does not hit any obstacles

$(M1, M2) = c(S0, S1, \ldots, S7)$ where $c: \{0,1, \ldots, 1023\}^8 \rightarrow \{0,1, \ldots, 15\}^2$

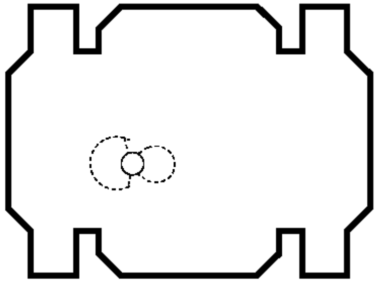




[Nordin, Banzhaf (1997)]

Example: Evaluation of A Controller Function

The controller function under consideration is loaded onto the robot and executed directly on the robot in a given environment for 400ms:



Details will be given in Section 3.2.

[Nordin, Banzhaf (1997)]

What Are Randomized Search Algorithms?

Randomized search algorithm (RSA):

- In general, optimization method that explicitly makes use of random decisions.
- Here, stochastic optimization heuristic for black box scenarios.

Goal of randomized search:

Systematically sample the decision space such that

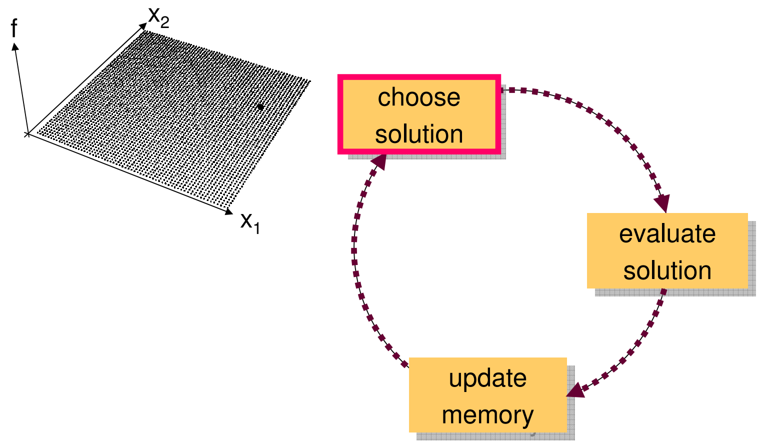
1. the number of considered solutions is minumum, and
2. the quality of the best solution found is maximum.

Why discuss RSA here?

Most biologically inspired computation techniques belong to the class of randomized search algorithms...

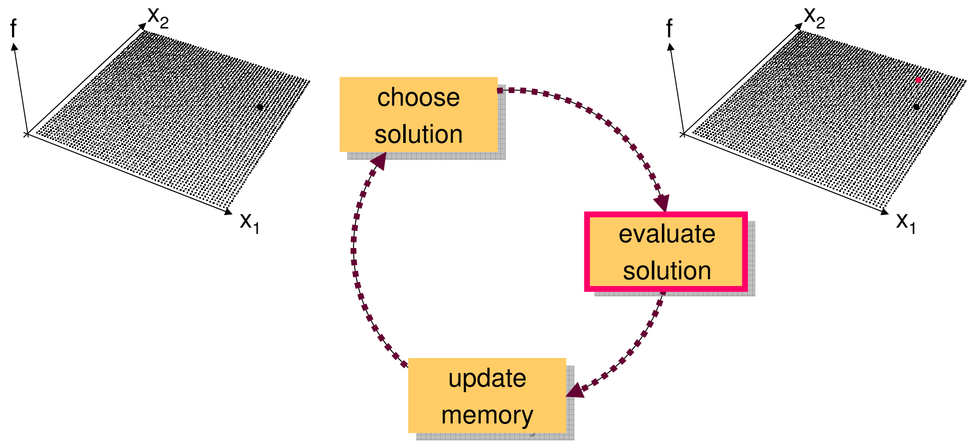
Randomized Search: Principle

Strategy: defines the way how the search space is sampled



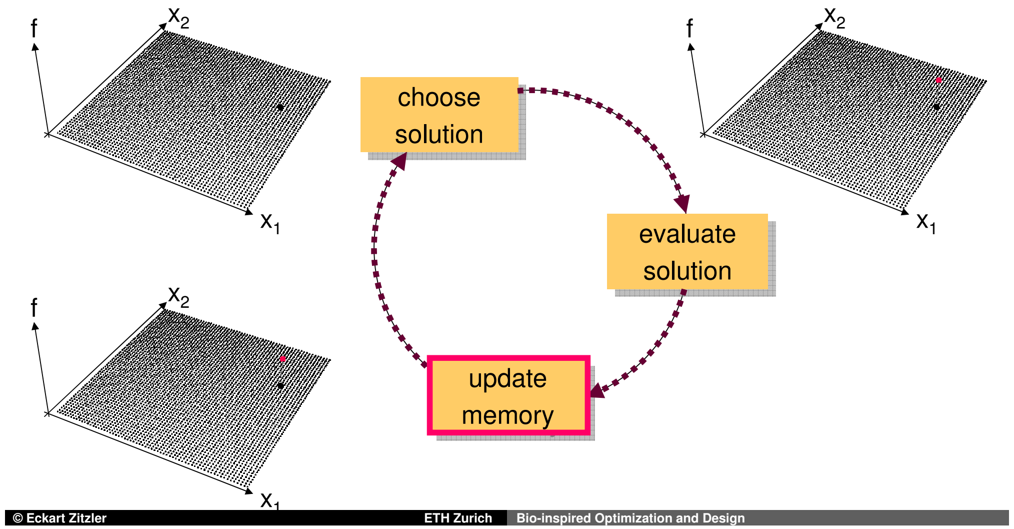
Randomized Search: Principle

Strategy: defines the way how the search space is sampled



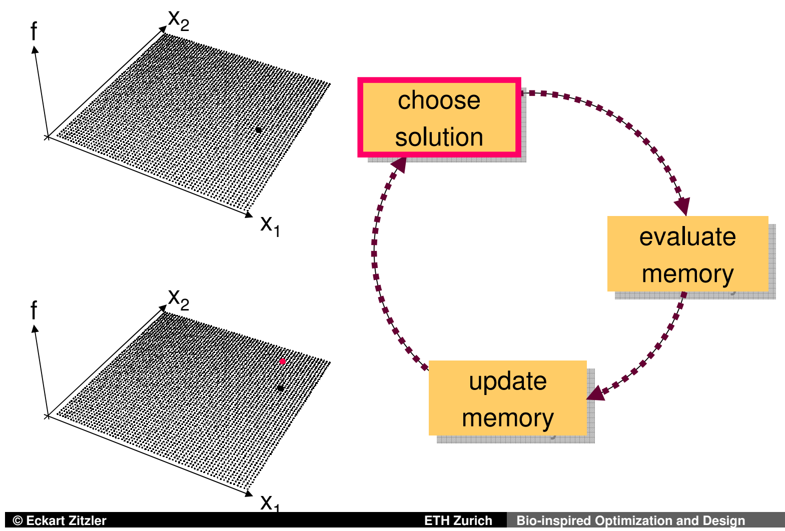
Randomized Search: Principle

Strategy: defines the way how the search space is sampled



Randomized Search: Principle

Strategy: defines the way how the search space is sampled



A General Randomized Search Algorithm

```
1:  Randomly choose an initial solution $x_1$ from $X$
2:  Calculate $f(x_1)$
3:  Initialize memory, i.e., $M = \{(x_1, f(x_1))\}$
4:  Set iteration counter $t = 0$
5:  loop
6:     $t = t + 1$
7:     Randomly choose $x_t \in X$ using $M$
8:     Calculate $f(x_t)$
9:     Add $(x_t, f(x_t))$ to the memory, i.e., $M = M \uplus \{(x_t, f(x_t))\}$
10:    if $t \geq t_{MAX}$ then
11:       Output best solution $x^*$ stored in $M$
12:       Stop
13:    end if
14: end loop
```

- *M* can be regarded as a multiset, i.e., it may contain duplicates.
- Various other termination criteria, e.g., no improvement over the last t_{MAX} iterations, can be used.

Background: Multisets

- A multiset differs from a regular set in that an element can be contained several times in the multiset. For instance, *{1, 1, 2, 3, 3}* and *{1, 2, 2, 2, 3, 3}* represents two different multisets, while the corresponding set interpretation would yield *{1, 2, 3}* in both cases.
- Formally, a multiset *A* is defined via a corresponding member function $m_A: U \rightarrow \mathbb{N}_0$ which gives for each potential element *a* of a universe *U* the number $m_A(a)$ of occurences in *A*.
- The regular set operations can be extended to multisets in the following way:
  - $C = A \cup B :\Leftrightarrow m_C(a) = \max\{m_A(a), m_B(a)\}$ for all $a \in U$ (union)
  - $C = A \uplus B :\Leftrightarrow m_C(a) = m_A(a) + m_B(a)$ for all $a \in U$ (join)
  - $C = A \cap B :\Leftrightarrow m_C(a) = \min\{m_A(a), m_B(a)\}$ for all $a \in U$ (intersection)
  - $C = A \setminus B :\Leftrightarrow m_C(a) = \max\{m_A(a) - m_B(a), 0\}$ for all $a \in U$ (difference)

Example: Random Search

Maximization problem:
(X, ℜ, f, ≥)

```
1:  Randomly choose an initial solution $x_1$ from $X$
2:  Calculate $f(x_1)$
3:  Initialize memory, i.e., $M = \{(x_1, f(x_1))\}$
4:  Set iteration counter $t = 0$
5:  loop
6:     $t = t + 1$
7:     Randomly choose $x_t \in X$
8:     Calculate $f(x_t)$
9:     if $f(x_t) \geq f(x)$ where $M = \{(x, f(x))\}$ then
10:       Keep $x_t$ as best solution seen so far, i.e., $M = \{(x_t, f(x_t))\}$
11:    end if
12:    if $t \geq t_{MAX}$ then
13:       Output solution $x$ stored in $M$ with $M = \{(x, f(x))\}$
14:       Stop
15:    end if
16: end loop
```

Random search does not make use of previously visited solutions.

Problem Structure and Information Content

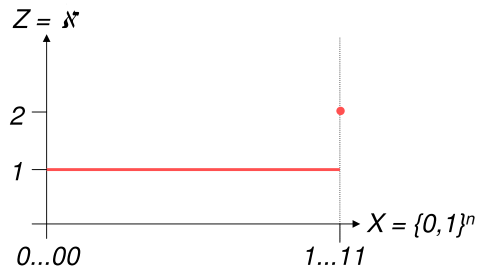
Assumptions underlying all randomized search heuristics:

- Partial sampling is sufficient to reveal the problem landscape;
- Better solutions can be found in the neighborhood of good solutions.

But: not all problem landscapes are well structured...

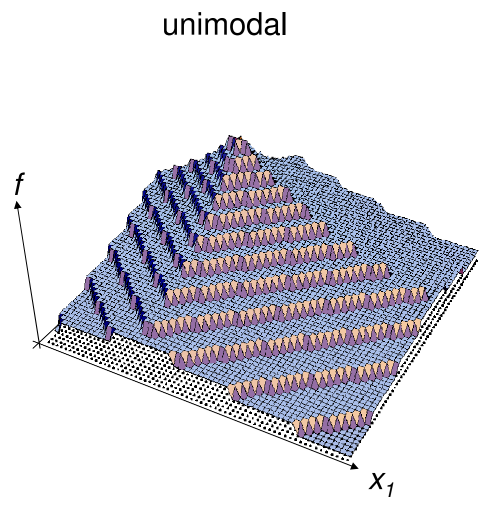
$$f_{NEEDLE}(\mathbf{x}) = 1 + \prod_{i=1}^{n} x_i \text{ with } \mathbf{x} = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$$

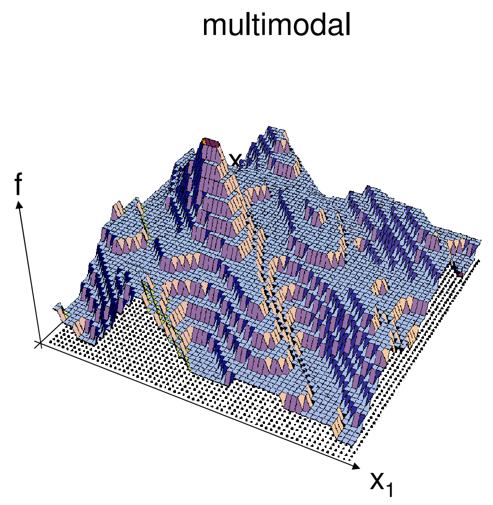
Here random search is the best one can do.



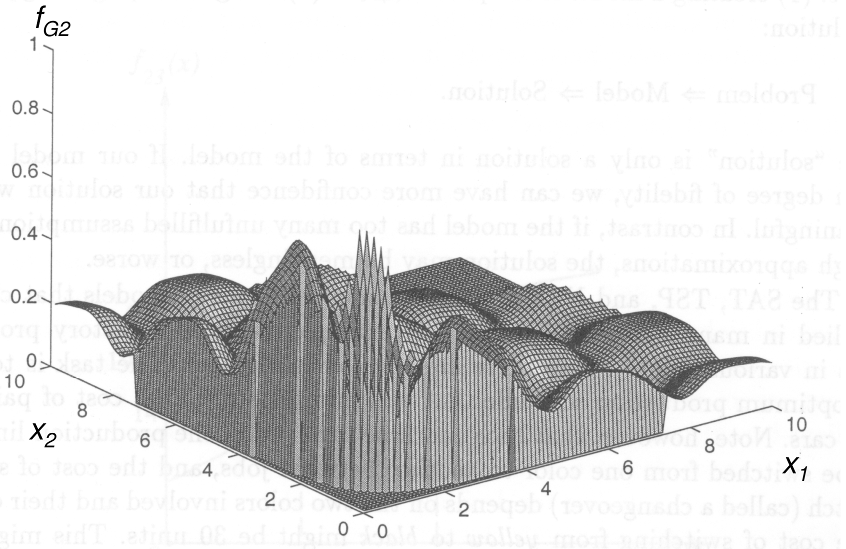
$(\{0,1\}^n, \aleph, f_{NEEDLE}, \geq)$

Highly Structured Problem Landscapes





A Partially Structured Problem Landscape

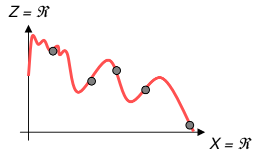


[Michalewicz, Fogel (2000), page 15]

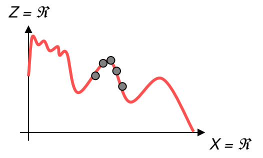
Exploration Versus Exploitation

In black box optimization, there are two conflicting principles:

Exploration = sampling everywhere
(explore new, promising regions)



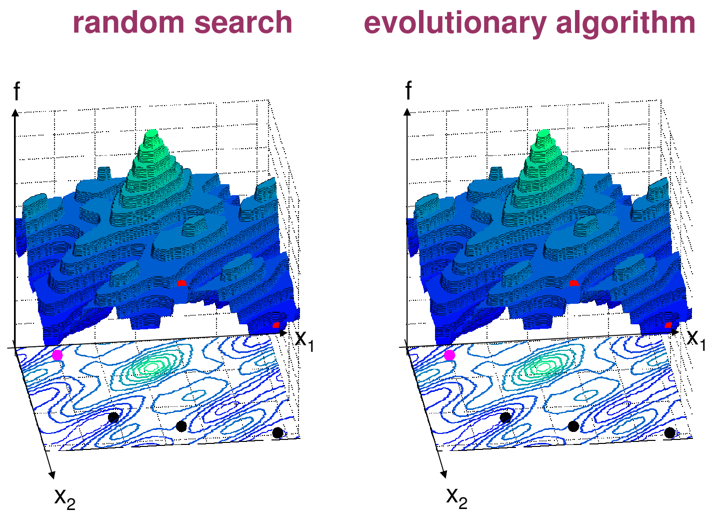
Exploitation = sampling around promising solutions found so far
(exploit available information about problem)



Which principle is more important depends on the optimization problem:

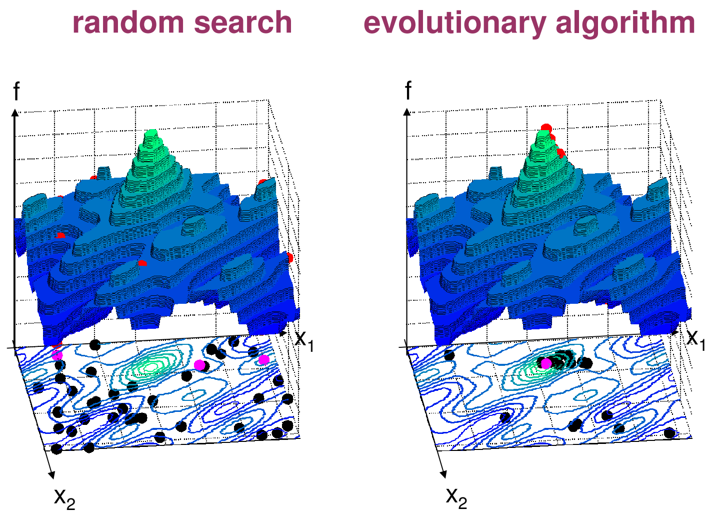
single local optimum ⟷ many local optima
exploitation ⟷ exploration

Exploitation Usually Helps...



initial state (four solutions per iteration)

Exploitation Usually Helps...



final state

Types of Randomized Search Algorithms

- Before discussing RSA inspired by biological models, some widely-used, non-bioinspired RSA will be presented. Each of these methods represents a specific trade-off between exploitation and exploration.

- Specific RSA variants usually differ in the following aspects:

  - The number of solutions stored in the memory (*|M| = 1 or |M| > 1*).
  - The strategy that defines which solutions are kept in the memory.
  - The way how the information stored in M is used to generate new solutions.
  - The method of how to represent the information gained during the optimization process in the memory.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Computer Engineering and Networks Laboratory

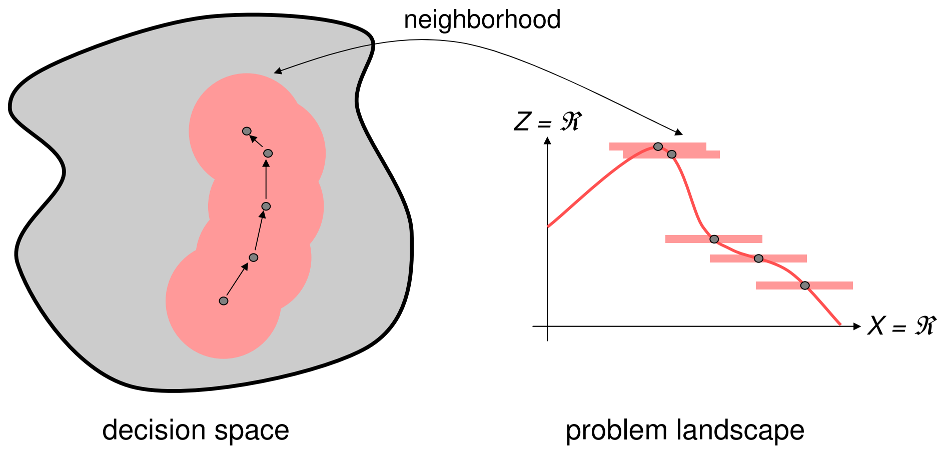
Bio-inspired Optimization and Design

Eckart Zitzler

2. Randomized Search Algorithms

Local Search: Principle

Idea: Instead of looking at the entire decision space in each iteration, only consider solutions in the proximity of the best solution so far.



Local Search Algorithm

```
1: Randomly choose an initial solution $x_1$ from $X$
2: Calculate $f(x_1)$
3: Initialize memory, i.e., $M = \{(x_1, f(x_1))\}$
4: Set iteration counter $t = 0$
5: **loop**
6:   $t = t + 1$
7:   Choose $x_t \in N(x) \subseteq X$ where $M = \{(x, f(x))\}$
8:   Calculate $f(x_t)$
9:   **if** $f(x_t) \geq f(x)$ **then**
10:    $M = \{(x_t, f(x_t))\}$
11:  **end if**
12:  **if** $t \geq t_{MAX}$ **then**
13:    Output best solution $x^*$ stored in $M$
14:    Stop
15:  **end if**
16: **end loop**
```

Maximization problem:
(X, ℜ, f, ≥)

- There is always just one solution in the memory.
- A new solution is generated by locally perturbing the current one.

The Neighborhood Function

The neighborhood function $N(x) \subseteq X$ defines for each solution x the set of its neighbors.

Example:

$(\{0,1\}^n, \{1, 2, ..., n\}, f_{ONEMAX}, \geq)$ where

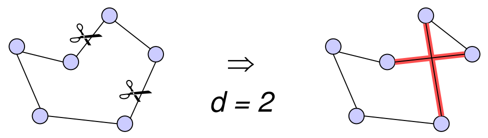
- $f_{ONEMAX}(\mathbf{x}) = \sum_{i=1}^{n} x_i$ with $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$
- $N_d(\mathbf{x}) = \{\mathbf{x}' \in X \mid \text{Hamming distance } H(\mathbf{x}, \mathbf{x}') \leq d\}$
- $H(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} |x_i - x_i'|$

The parameter *d* determines the size of the neighborhood, e.g. (n=3):

- $N_0((0,0,1)) = \{(0,0,1)\}$
- $N_1((0,0,1)) = \{(0,0,1), (1,0,1), (0,1,1), (0,0,0)\}$
- $N_2((0,0,1)) = \{(0,0,1), (1,0,1), (0,1,1), (0,0,0), (0,0,1), (1,1,1), (1,0,0), (0,1,0)\}$

A Neighborhood Function for the TSP

Principle: perform *d* cuts on the current route and rearrange the resulting pieces ⇒ neighboring solution



As $|N_d|$ grows fast with increasing d, usually values $d \leq 3$ are used.

In detail:

- $N_d(\pi) = \{\pi' \in X \mid \mathit{diff}(\pi, \pi') \leq d\}$
- $\mathit{diff}(\pi, \pi') = |\{1 \leq i \leq n \mid \mathit{succ}(\pi, i) \neq \mathit{succ}(\pi', i)\}|$
- $\mathit{succ}(\pi, i) = \begin{cases} 1 & \text{if } \pi^{-1}(i) = n \\ \pi(2) & \text{if } i = 1 \\ \pi(\pi^{-1}(i) + 1) & \text{else} \end{cases}$

where i denotes the city

Choosing A Solution: Determinism Vs. Randomization

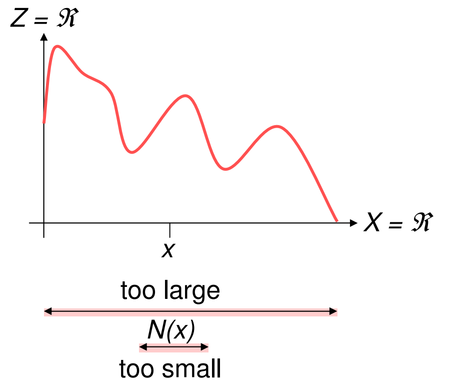
Two variants of local search can be distinguished:

1. **Deterministic local search:** all solutions in the neighborhood *N(x)* are evaluated and the best one is chosen
   - Always ends in a local optimum (↑exploitation)
   - Large neighborhoods are computationally infeasible (↓exploration)
2. **Randomized local search:** $K < |N(x)|$ solutions are chosen randomly from the neighborhood *N(x)* and evaluated; the best of the *K* solutions is chosen
   - May not reach local optimum (↓exploitation)
   - Large neighborhoods can be considered (↑exploration)

Influence of the Size of the Neighborhood

N(x) too large
$\Rightarrow$ exhaustive / random search

N(x) too small
$\Rightarrow$ getting stuck in local optima



Example: f_{ONEMAX}

$$|N_d(\mathbf{x})| = \binom{n}{d} + \binom{n}{d-1} + \ldots + \binom{n}{0} \le d \cdot n^d$$

- With $x_1 = 00...0$ and $d = 1$ overall *n* iterations are needed to find the optimal solution.
- Exhaustive search needs $|X| = 2^n$ in the worst case.
- The number of visited solutions is less than $(n / d) \cdot d \cdot n^d = n^{d+1}$
$\Rightarrow$ d = 1 is optimal

Question

What extensions can you envision to overcome the problem of getting stuck in local optima?

Multistart Strategies

Problem: How to increase the chance that the global optimum is found?

Idea: Restart the algorithm several times, each time with a different initial solution; keep best solution found so far.

- Success probability of one run: δ
- Success probability of T runs: $1 - (1 - \delta)^T$
- For $\delta = 0.5$ and *10* runs: success probability > *99.9%*
- In practice, though, δ can be very small...

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Engineering
and Networks Laboratory

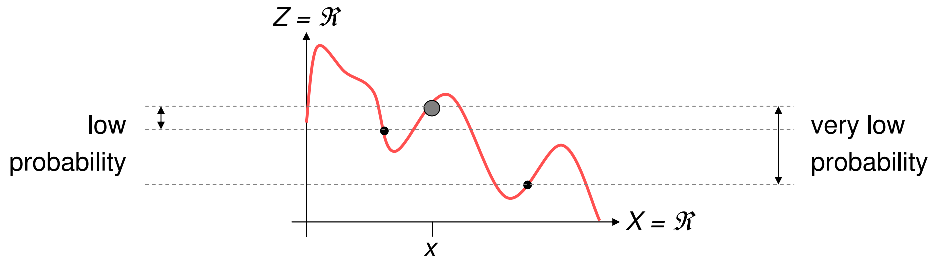
Bio-inspired Optimization and Design

Eckart Zitzler

2. Randomized Search Algorithms

Metropolis Algorithm: Principle

Idea: avoid getting stuck in local optima by occasionally accepting worse solutions; the probability is lower the higher the quality difference.



Original publication:

[Metropolis, Rosenbluth, Rosenbluth, Teller, Teller (1953)]

Metropolis Algorithm

```
Randomly choose an initial solution $x_1$ from $X$
Calculate $f(x_1)$
Initialize memory, i.e., $M = \{(x_1, f(x_1))\}$
Set iteration counter $t = 0$
loop
  $t = t + 1$
  Choose $x_t \in N(x) \subseteq X$ where $M = \{(x, f(x))\}$
  Calculate $f(x_t)$
  if $f(x_t) \geq f(x) \vee rand[0,1] \leq e^{-\frac{f(x)-f(x_t)}{T}}$ then
    $M = \{(x_t, f(x_t))\}$
  end if
  if $t \geq t_{MAX}$ then
    Output solution $x$ stored in $M$
    Stop
  end if
end loop
```

Maximization problem:
$(X, \Re, f, \geq)$

- Allows temporary deterioration depending on difference in quality
- *rand[0,1]* : uniformly randomly chosen number in *[0,1]* ; *T*: temperature

Local Versus Global Optimization Methods

- The local search algorithm is by definition a local optimization method, i.e., a method that locally improves the initial solution. Depending on the choice of the neighborhood and the problem, for some or even most of the possible initial solutions the probability that a global optimum is found is *0*. However, the probability that any local optimum is found is always greater than *0*.

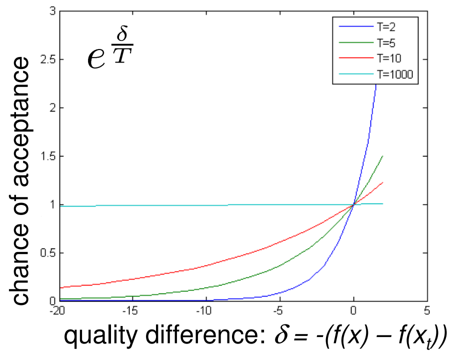
- With global optimization methods, the probability that a globally optimal solution is found is greater than *0* for all possible initial solutions. This does not mean that a global optimum is actually found, but at least there is a chance that this is the case.

  The Metropolis algorithm is a global optimization method as there is a mechanism included that allows to escape local optima. Also the algorithms presented in the following (simulated annealing, tabu search, and evolutionary algorithms) are global optimization methods.

  For all global optimization method, it is useful to keep the best solution found so far as there is always a chance that the next accepted solution is worse than the current one. In the following, we will always assume that this mechanism is included, although it will not be explicitly be listed in the algorithm descriptions.

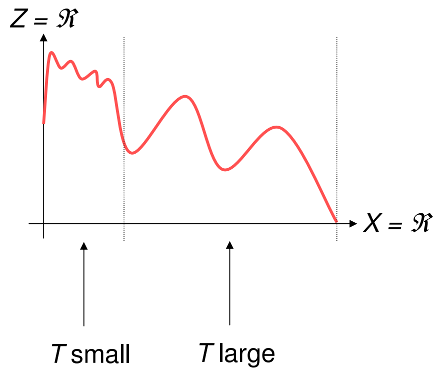
Influence of the Temperature

- $T \to 0$: local search
- $T \to \infty$: random search



Key issue: T should be chosen depending on the problem, but how to choose optimal T?

- "Jumping over large gaps": T large
- "Staying on the plateau": T small



ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Computer Engineering and Networks Laboratory

Bio-inspired Optimization and Design

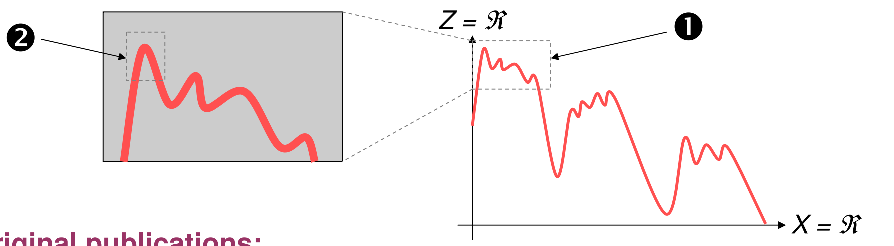
Eckart Zitzler

2. Randomized Search Algorithms

Simulated Annealing Algorithm: Principle

Idea: decrease the temperature systematically such that

- at the beginning large gaps can be overcome and a promising plateau can be identified (exploration), and
- at the end deterioration is avoided and the (locally) optimal solution within the identified plateau can be approximated (exploitation).



Original publications:

- [Kirkpatrick, Gelatt, Vecchi (1983)]
- [V. Cerny (1985)]

Simulated Annealing Algorithm

```
1:  Randomly choose an initial solution $x_1$ from $X$
2:  Calculate $f(x_1)$
3:  Initialize memory, i.e., $M = \{(x_1, f(x_1))\}$
4:  Set iteration counter $t = 0$
5:  Set initial temperature $T = T_{INIT}$
6:  loop
7:    $t = t + 1$
8:    Choose $x_t \in N(x) \subseteq X$ where $M = \{(x, f(x))\}$
9:    Calculate $f(x_t)$
10:   if $f(x_t) \geq f(x) \lor rand[0,1] \leq e^{-\frac{f(x)-f(x_t)}{T}}$ then
11:     $M = \{(x_t, f(x_t))\}$
12:   end if
13:   Update temperature $T = cool(t, T)$
14:   if $t \geq t_{MAX}$ then
15:     Output solution $x$ stored in $M$
16:     Stop
17:   end if
18: end loop
```

Cooling schedule *cool*:

- decreases temperature based on t and T
- can also be adaptive, i.e., dependent on the improvement over time

Question

Question:
What is the reason for computing a probability and making a random decision whether to take a solution or not? Alternatively, one might consider an error threshold that is decreased.

Answer:
If Step 10 would be deterministic, one always would accept worse solution within the range defined by the temperature. That means at a certain stage, solutions exceeding the threshold will never be accepted anymore, which in turn can cause the algorithm to get stuck. This can never happen with the stochastic version as there is always a chance greater than zero that a worse solution is accepted.

Cooling Schedules

- Geometric:

  $cool(t, T) = \alpha \cdot T$ with $\alpha < 1$

- Linear:

  $cool(t, T) = T - \alpha$ with $\alpha > 0$

Is Simulated Annealing Better Than Metropolis?

Key question: is the cooling schedule essential, does it actually help?

- The cooling schedule usually does not make performance worse, e.g., when considering the commonly used geometric schedule:

  $cool(t, T) = \alpha \cdot T$ with $\alpha < 1$

- Many problems cannot be solved more efficiently by simulated annealing than by the Metropolis algorithm with the best setting for the temperature parameter *T*.

- Until recently, it was an open problem whether there exists a non-artificial problem for which simulated annealing outperforms the Metropolis algorithm with optimal temperature setting.

  I. Wegener showed in 2004 that simulated annealing can do better on specific instances of the minimum spanning tree problem.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Computer Engineering and Networks Laboratory

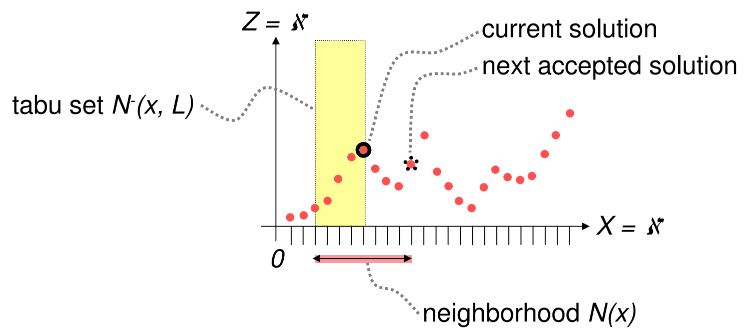
Bio-inspired Optimization and Design

Eckart Zitzler

2. Randomized Search Algorithms

Tabu Search Algorithm: Principle

Idea: Unlike local search, Metropolis, and simulated annealing, keep not only the one solution but the last *K* visited solutions in the memory. These previously visited solutions mark regions in the decision space which are prohibited for a certain time, i.e., they are tabu.



First extensive discussion:

[Glover, Laguna (1997)]

Tabu Search Algorithm

```
1:  Randomly choose an initial solution $x_1$ from $X$
2:  Calculate $f(x_1)$
3:  Initialize memory, i.e., $M = \{(x_1, f(x_1))\}$
4:  Initialize tabu list $L = []$
5:  Set iteration counter $t = 0$
6:  loop
7:    $t = t + 1$
8:    Choose $x_t \in N(x) \setminus N^-(x, L) \subseteq X$ where $M = \{(x, f(x))\}$
9:    Calculate $f(x_t)$
10:   $M = \{(x_t, f(x_t))\}$
11:   Update tabu list $L$
12:   if $t \geq t_{MAX}$ then
13:     Output solution $x$ stored in $M$
14:     Stop
15:   end if
16: end loop
```

- *N⁻(x, L)* denotes the set of solutions that are tabu w.r.t. *x*
- *x* is contained by definition in *N⁻(x, L)*

The Tabu Set: Solution-Based Implementation

The tabu set *N⁻(x, L)* for a particular solution is determined by the tabu list.

- The simplest form to implement the tabu list is by means of a FIFO queue (FIFO = first in, first out) that contains the recent *k* solutions visited (discrete problems). In this case, the tabu set is defined as

  $N^-(x, L) = \{x\} \cup \{x' \in X \mid x' \text{ is contained in } L\}$

  At each iteration, the tabu list is updated by removing the oldest element in the list by the solution entering the tabu list. Therefore, each solution stays *k* iterations in the tabu list.
- Example: f_{ONEMAX} with n = 3 and k = 2

$t = 5$:	$M = \{(0,1,0)\}$	$L = [(0,0,0), (1,1,0)]$
$t = 6$:	$M = \{(0,1,1)\}$	$L = [(1,1,0), (0,1,0)]$
$t = 7$:	$M = \{(1,1,1)\}$	$L = [(0,1,0), (0,1,1)]$

The Tabu Set: Move-Based Implementation

- Another possibility to define the tabu set is by storing not entire solutions but moves in the tabu list.
- A move is an operation that transforms one solution x into another solution x‘. In the case of binary vectors for instance, a move is described by the number of bit positions that need to be flipped for the transformation:

  $x = (0,0,0,1,1)$, $x' = (1,0,0,1,0) \rightarrow$ bit positions changed $= \{1, 5\}$
- Formally, one can define a transformation function *h* which takes as arguments (i) the solution to be transformed and (ii) the description of the move. In the case $X = \{0,1\}^n$ the function *h* can be defined as:

$$h((x_1, x_2, \ldots, x_n), F) = (x'_1, x'_2, \ldots, x'_n) \text{ where } x'_i = \begin{cases} 1 - x_i & \text{if } i \in F \\ x_i & \text{else} \end{cases}$$

The Tabu Set: Move-Based Implementation (Cont'd)

- Given *h*, the tabu list *L* is updated every time by adding the move *F* that transforms the current solution *x* to the selected solution x_t, i.e., $x' = h(x, F)$; as before, the oldest move in *L* is removed.
- Accordingly, the tabu set can now be defined as follows:

  $N^-(x, L) = \{x\} \cup \{x' \in X \mid \exists F \in L: x' = h(x, F)\}$

  It contains all solutions that can be generated from x via non-tabu moves. Note: $N^-(x, L)$ is larger than in the solution-based approach.
- Example: f_{ONEMAX} with n = 3 and k = 2

t = 5:	*M = {(0,1,0)}*	*L = [{2}, {1}]*
t = 6:	*M = {(0,1,1)}*	*L = [{1}, {3}]*
t = 7:	*M = {(1,0,1)}*	*L = [{3}, {2}]*

(1,1,1) is tabu!

ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Computer Engineering and Networks Laboratory

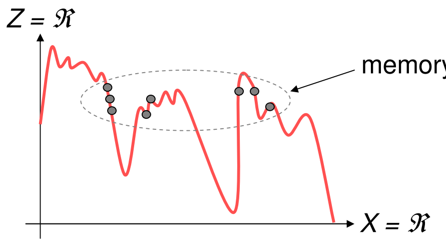
Bio-inspired Optimization and Design

Eckart Zitzler

2. Randomized Search Algorithms

Evolutionary Algorithms: Principle

Idea: explore several regions of the decision space in parallel by maintaining a population of solutions ⇔ parallel multistart strategy where the separate runs interact and are coordinated



History:

around 1960	first technical applications of simulated evolution
around 1970	three independent main branches of evolutionary algorithms
since 1985	rapidly growing field (availability of computing resources)

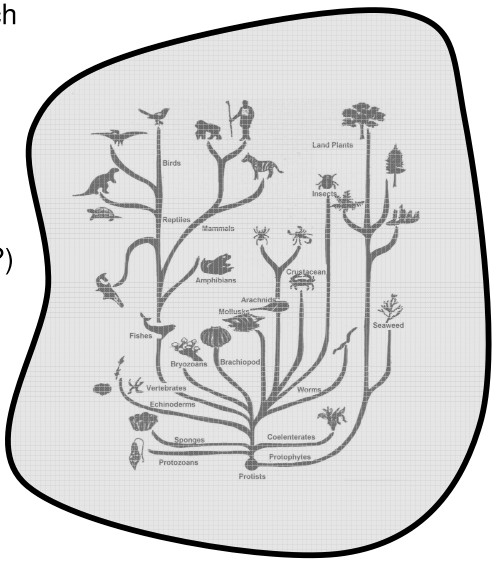
Biological Motivation

Main assumption: evolution = search

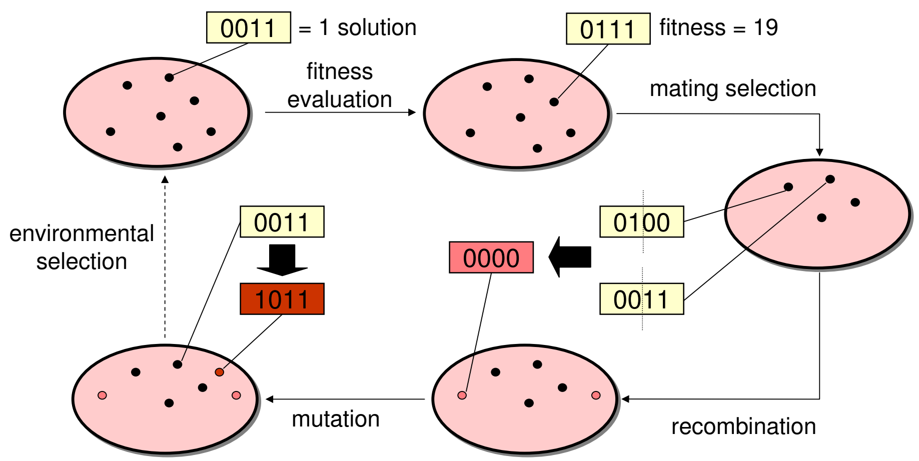
Main principles:

❶ phenotypic selection
- environmental (who survives?)
- mating (who reproduces?)

❷ genetic variation
- mutation
- recombination
- (inversion, deletion, etc.)



Scheme of An Evolutionary Algorithm



Evolutionary Algorithm (EA)

```
1: Randomly choose $x_1, x_2, \ldots, x_N$ from $X$
2: Calculate $f(x_1), f(x_2), \ldots, f(x_N)$
3: Initialize memory, i.e., $M = \{(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_N, f(x_N))\}$
4: Set iteration counter $t = 0$
5: Assign each $x_i$ in $M$ a fitness value $F_i$
6: **loop**
7:    $t = t + 1$
8:    $M'$ = mating selection from $M$
9:    $M''$ = mutation and recombination on $M'$
10:   Calculate $f(x_i)$ for $x_i$ in $M''$
11:   Assign each $x_i$ in $M''$ a fitness value $F_i \in$
12:   $M$ = environmental selection on $M$ and $M''$
13:   **if** $t > t_{MAX}$ **then**
14:      Output best solution $x^*$ stored in $M$
15:      Stop
16:   **end if**
17: **end loop**
```

New: population of solutions, mating selection, recombination

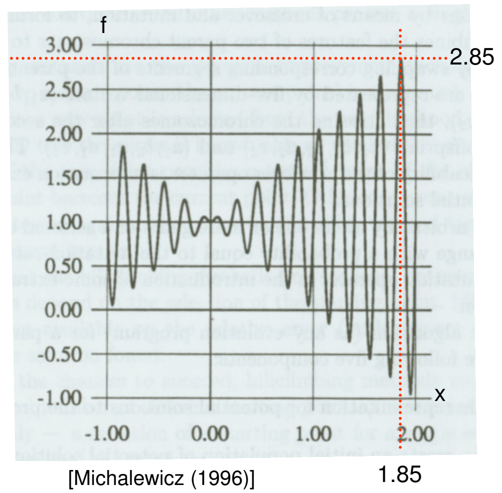
Evolutionary Algorithms: Terminology

- **Fitness:** value that reflects the quality of an individual
- **Mating selection:** chooses high quality individuals to generate new ones
- **Mutation operator:** generates a new solution by modifying a given solution; similar to the neighborhood function
- **Recombination:** combines two or more solutions to generate a new solution
- **Environmental selection:** decides which of the old and the newly created solutions will be kept in the memory
- **Population:** M (multiset!), usually its size is fixed with $N = |M|$
- **Individual:** synonymous for a solution stored in M
- **Parent:** element of the mating pool M'
- **Offspring:** element of M''
- **Generation:** one iteration of the algorithm; also: the *t*-th generation stands for the population after *t* iterations

Example: Design of An Evolutionary Algorithm

$(X, \Re, f_{SIMPLE}, \geq)$ with

- $X = \{x \in \Re \mid -1 \leq x \leq 2\}$
- $f_{SIMPLE}(x) = x \cdot \sin(10\,\pi\, x) + 1$
- optimal solution $x^* \approx 1.85\ldots$ (easy to analyze)



[Michalewicz (1996)]

Aim: step-by-step design of an EA for this problem

Step 1: Representation

Question: how to encode real values (7-digit representation) on the basis of binary vectors?

- values to be represented: *{-1.000000, -1.000001, ..., 2.000000}*
  $\Rightarrow$ overall $3 \cdot 10^6 + 1$ different values need to be represented
  $\Rightarrow$ *22* Bits needed ($2^{21} < 3 \cdot 10^6 + 1 \leq 2^{22}$)
- search space on which the EA works: $Y = \{0,1\}^{22}$
- mapping *m* from *Y* to *X*:

$$m(\mathbf{x}) = -1 + \frac{3}{2^{22}-1} \sum_{i=1}^{22} x_i 2^{(i-1)} \text{ where } \mathbf{x} = (x_1, x_2, \ldots, x_{22}) \in \{0,1\}^{22}$$

$$\begin{aligned} 0000000000000000000000 &\xrightarrow{m} -1.000000 \\ 0000000000000000000001 &\longrightarrow -0.999999 \\ \ldots & \\ 1111111111111111111111 &\longrightarrow 2.000000 \end{aligned}$$

Steps 2 + 3: Fitness Assignment and Mating Selection

- **Fitness:** $F_i = f_{SIMPLE}(m(x_i))$
- **Mating selection:** create a temporary population M‘ by repeatedly holding tournamens between two randomly chosen individuals

```
1: $M' = \emptyset$
2: for $i = 1$ to $N$ do
3:   Randomly select $x_j, x_k$ from $M$
4:   if $F_j > F_k$ then
5:     $M' = M' = \uplus\{(x_j, f_{SIMPLE}(m(x_j)))\}$
6:   else
7:     $M' = M' = \uplus\{(x_k, f_{SIMPLE}(m(x_k)))\}$
8:   end if
9: end for
```

Step 4: Variation

- **Mutation:** flip each bit with probability p_m (mutation rate)
- **Recombination:** with probability p_c (crossover rate) split both parents as the same position and create two children by joining the complementary halves from both parents; otherwise, return the parent individuals unmodified
- **Variation part:**

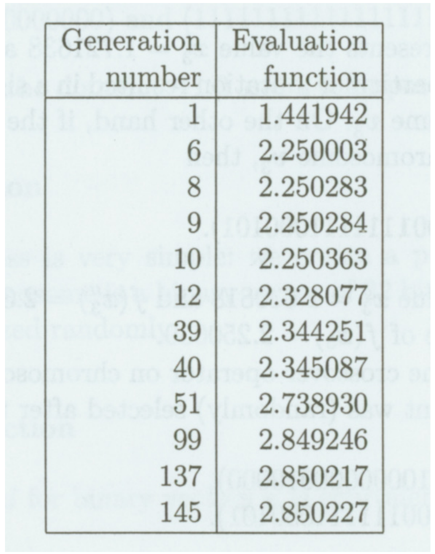
```
1: $M'' = \emptyset$
2: while $M' \neq \emptyset$ do
3:   Randomly select $x_j, x_k$ from $M'$
4:   $M' = M' \setminus \{(x_j, f_{SIMPLE}(m(x_j))), (x_k, f_{SIMPLE}(m(x_k)))\}$
5:   $x'_j, x'_k = recombination(x_j, x_k)$
6:   $x''_j = mutation(x'_j)$
7:   $x''_k = mutation(x'_j)$
8:   $M'' = M'' \uplus \{(x''_j, f_{SIMPLE}(m(x''_j))), (x''_k, f_{SIMPLE}(m(x''_k)))\}$
9: end while
```

Steps 5 + 6: Environmental Selection and Parameters

- **Environmental selection:**
  M = M“
- **Parameters:**
  N = 50
  $p_c = 0.25$
  $p_m = 0.01$

  $\Rightarrow$ chance that a selected individual is copied unmodified to the next generation:

  $(1 - p_c) \cdot (1 - p_m) \approx 60\%$



Generation number	Evaluation function
1	1.441942
6	2.250003
8	2.250283
9	2.250284
10	2.250363
12	2.328077
39	2.344251
40	2.345087
51	2.738930
99	2.849246
137	2.850217
145	2.850227

[Michalewicz (1996)]

Roots Of Evolutionary Algorithms

Today:
- evolutionary algorithm = umbrella term
- differences blurr

Genetic Programming
John Koza (later)

Genetic Algorithms
John Holland

Evolutionary Programming
Lawrence Fogel

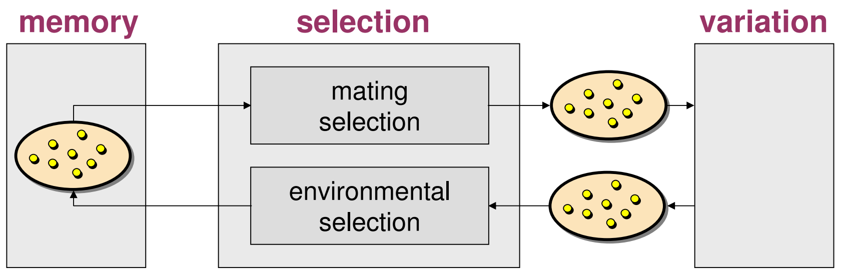
Evolution Strategies
Ingo Rechenberg
Hans-Paul Schwefel

See [Fogel (1998)]

Main Branches: Historical Differences

	Genetic Algorithms	Genetic Programming	Evolution Strategies	Evolutionary Programming
Biological focus	genotype	genotype	phenotype	phenotype
Representation	bit vector	trees	real vector	real vector (automata)
Mating selection	randomized fitness-based	randomized fitness-based	randomized uniform	deterministic fitness-based
Variation	mutation recombination	mutation recombination	mutation (recombination)	mutation (recombination)
Environmental selection	replacement: M'' = M	replacement: M'' = M	deterministic: μ best	randomized: μ best
Population size	$\|M\| = \|M'\| = \|M''\|$	$\|M\| = \|M'\| = \|M''\|$	$\|M\| = \mu$ $\|M'\| = \|M''\| = \lambda$	$\|M\| = \|M'\| = \|M''\|$
Misc			adaptive mutation rates	

Randomized Search Heuristics in the EA Framework



EA evolutionary algorithm	≥ 1	both	≥ 1 ≥ 1	N : M randomized
TS tabu search	1	no mating selection	1 ≥ 1	1 : M deterministic
SA simulated annealing	1	no mating selection	1 ≥ 1	1 : M randomized
ACO ant colony optimization	1	neither	1 1	1 : 1 randomized

References

- V. Cerny (1985): A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. Journal of Optimization Theory and Applications, 45, pp. 41-51.
- D. Fogel (1998): Evolutionary Computation – the Fossil Record. IEEE Press, Piscataway, NJ. **(History of evolutionary algorithms, original articles)**
- F. Glover, M. Laguna (1997): Tabu Search. Kluwer, Norwell, MA.
- S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi (1983): Optimization by Simulated Annealing, Science, Vol 220, Number 4598, pp. 671-680.
- N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller (1953): Equation of State Calculations by Fast Computing Machines. Journal of Chemical Physics, 21, pp. 1087-1092.
- Z. Michalewicz (1996): Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin.
- Z. Michalewicz, D. Fogel (2000): How to Solve It: Modern Heuristics. Springer, Berlin. **(Chapters 2, 5 + 6, local search, simulated annealing, tabu search, evolutionary algorithms)**
- P. Nordin, W. Banzhaf (1997): Real time control of a khepera robot using genetic programming. Cybernetics and Control 26(3), pp. 533-561.
- I. Wegener (2004): Simulated Annealing Beats Metropolis in Combinatorial Optimization. Electronic Colloquium on Computational Complexity, Report No. 89.