ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Computer Engineering
and Networks Laboratory

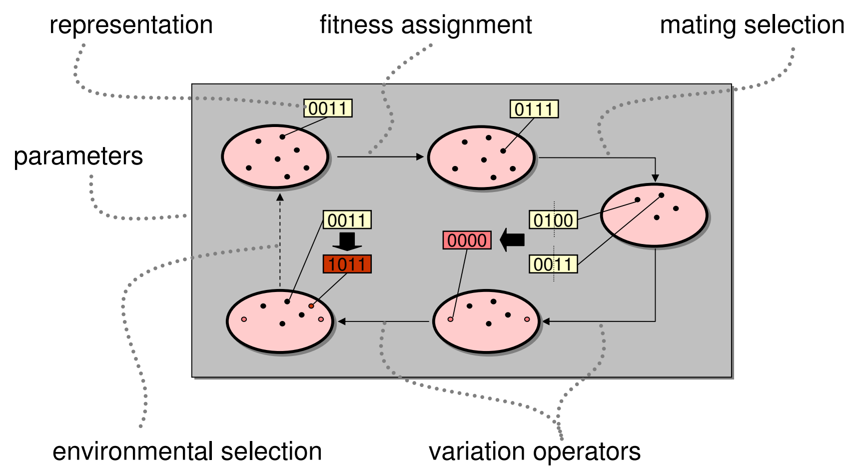
Bio-inspired Optimization and Design

Eckart Zitzler

---

3. Basic Design Issues

Basic Design Issues in a Nutshell



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

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.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Computer Engineering
and Networks Laboratory

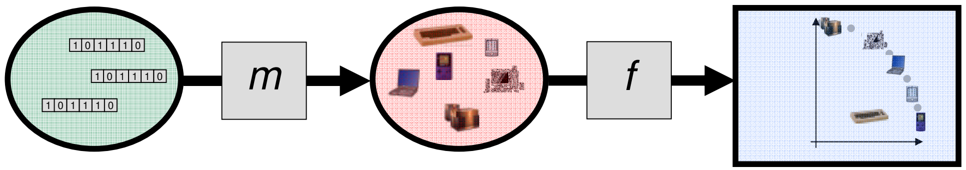
Bio-inspired Optimization and Design

Eckart Zitzler

3. Basic Design Issues

Search Space and Decision Space

search space decoder decision space objectives objective space



- The 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 $X = \Re$. 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

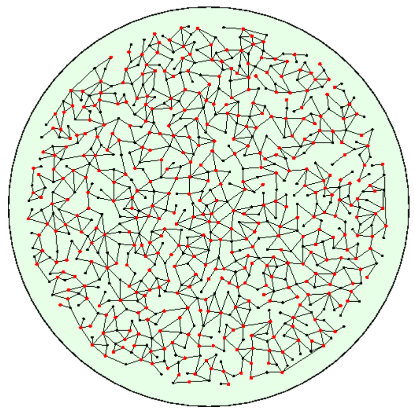
Other Types:
- matrices, general graphs, etc.
- often hybrid representations are used (e.g., binary vector + matrix)

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	A	B	C	D	E	F	…
selected?	1	0	1	1	1	0	…

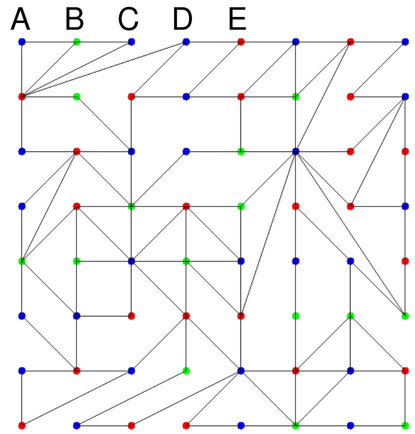


Example: Integer Vector Encoding

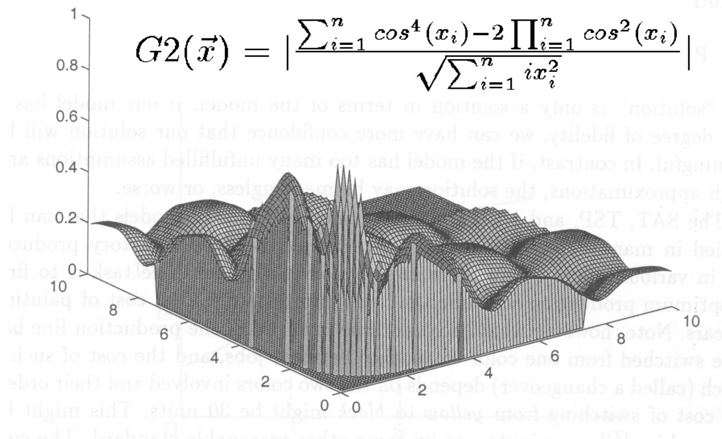
Given: graph, k colors

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)

nodes	A	B	C	D	E	F	…
colors	1	2	1	3	1	2	…



Example: Real Vector Encoding



$$G2(\vec{x}) = \left| \frac{\sum_{i=1}^{n} cos^4(x_i) - 2\prod_{i=1}^{n} cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

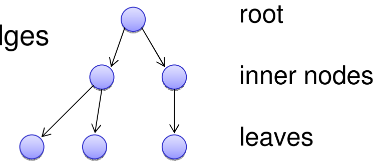
parameters	x_1	x_2	x_3	x_4	…	x_n
values	0.33	0.53	1.03	3.25	…	9.83

[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)



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:

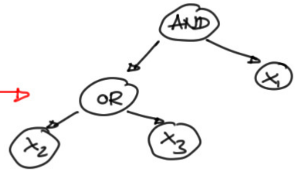
number of arguments (arity)

- 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 $S_O = \{IF_2, WHILE_2, FOR_2, =_2, \ldots\}$
  set of arguments $S_A = \{x_1, x_2, \ldots, x_k\} \cup \{0,1\}^l$ with $x_i \in \{0,1\}^l$

Some Example Trees

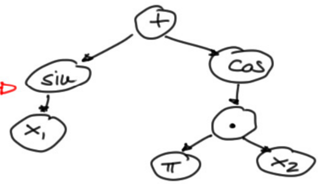
- Boolean expressions:

$(x_2 \vee x_3) \wedge x_1$



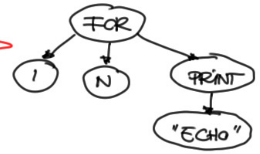
- Continuous functions:

$f(x) = \sin x_1 + \cos(\pi \cdot x_2)$



- Programs:

FOR i = 1 TO N DO
PRINT "ECHO"



What Defines the Search Space?

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

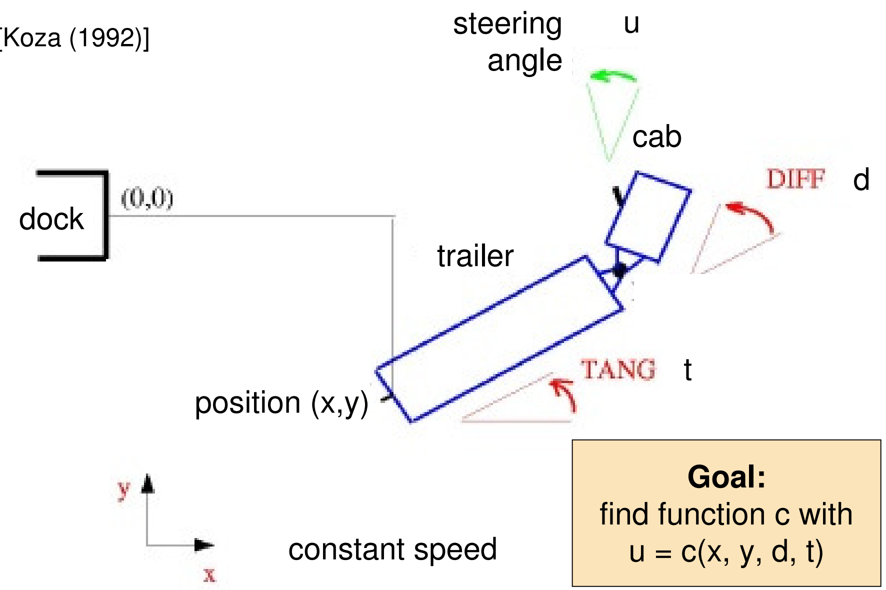
- the set of operators
- for each operator, the number of arguments and their order
- if data types are used, for each argument the data type
- the set of variables and constants
- if data types are used, for each variable/constant its type

- 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)]



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

Search Space for the Truck Problem

Operators:

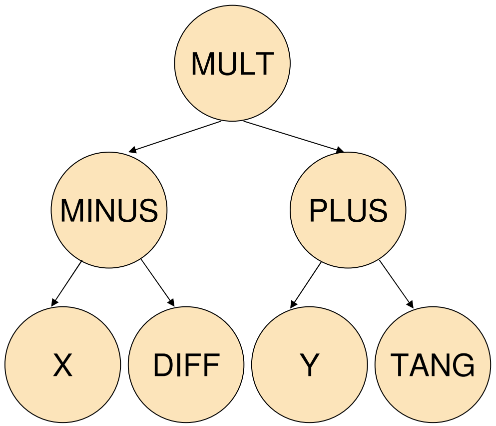
PLUS(a,b)	returns a+b
MINUS(a,b)	returns a-b
MUL(a,b)	returns a*b
DIV(a,b)	return a/b, if b <> 0, else 1
ATG(a,b)	returns atan2(a,b), if a<> 0, else 0
IFLTZ(a,b,c)	returns b, if a<0, else returns c

Arguments:

X	position x
Y	position y
DIFF	cab angle d
TANG	trailer angle t

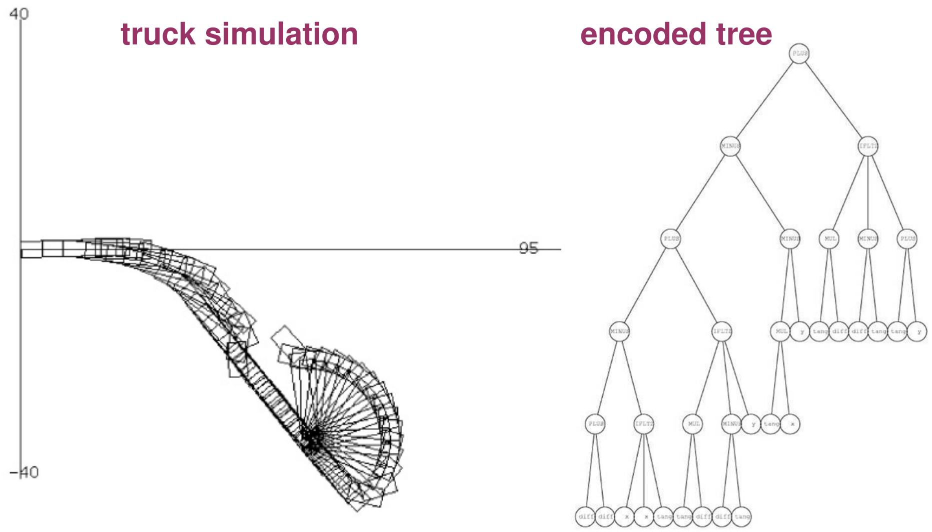
Decision space: set of symbolic expression using the above operators and arguments

Example Solution: Tree Representation



encodes the function (symbolic expression): u = (x – d) * (y + t)

A Solution Found by an EA



Fitness assignment will be discussed later...

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

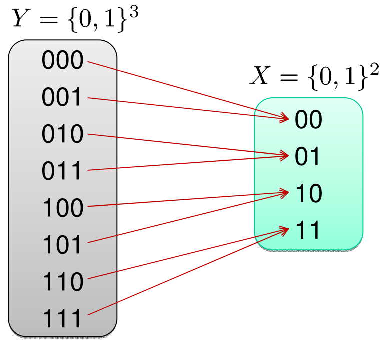
redundancy:
$r = \log_2 |Y| - \log_2 |X|$ where X, Y are finite

feasibility:
$\forall y \in Y \ : \ m(y) \in X$

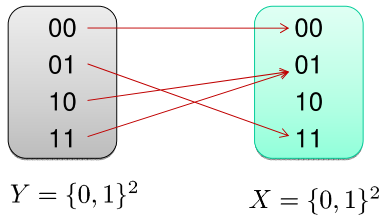
locality:
$\forall y, y', y'' \in Y : d(y, y') < d(y, y'') \Leftrightarrow d(m(y), m(y')) < d(m(y), m(y''))$

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.

Illustration of Representation Properties



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



- not complete
- not uniform
- not redundant
- feasible
- no locality within Hamming distance 1:
  d(00, 01) = 1 ≤ d(00, 11) = 2, but
  d(m(00), m(01)) = 2 >
  d(m(00), m(11)) = 1

Improving Locality

Question: Why is locality important?

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

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

X	binary numbers	gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

hamming distance: binary numbers 1, 2, 1, 3, 1, 2, 1; gray code 1, 1, 1, 1, 1, 1, 1

one-bit flip **cannot** generate both direct neighbors

one-bit flip can generate both direct neighbors (but may still make large jumps)

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 $a'_0, a'_1, \ldots, a'_M$
5: choose the following mapping

$$\begin{array}{rcl} a_0 & = & 0a'_0 \\ a_1 & = & 0a'_1 \\ & \ldots & \\ a_{2^{n-1}-1} & = & 0a'_M \\ a_{2^{n-1}} & = & 1a'_M \\ a_{2^{n-1}+1} & = & 1a'_{M-1} \\ & \ldots & \\ a_{2^n-1} & = & 1a'_0 \end{array}$$

6: **end if**

Reducing Redundancy (Example: TSP)

1. **Matrix representation:**
   - binary $n \times n$ matrix M with $\pi(i) = j \Leftrightarrow M(i,j) = 1$
   - many matrices are infeasible (unless repair mechanism is used)
   - the genotypic search space is large: $|Y| = 2^{n \cdot n}$
2. **Integer vector representation:**
   - vector $(p_1, p_2, \ldots, p_n) \in \{2, \ldots, n\}^{n-1}$ with $\pi(i) = j \Leftrightarrow p_i = j$
   - 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

ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Computer Engineering and Networks Laboratory

Bio-inspired Optimization and Design

Eckart Zitzler

3. Basic Design Issues

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

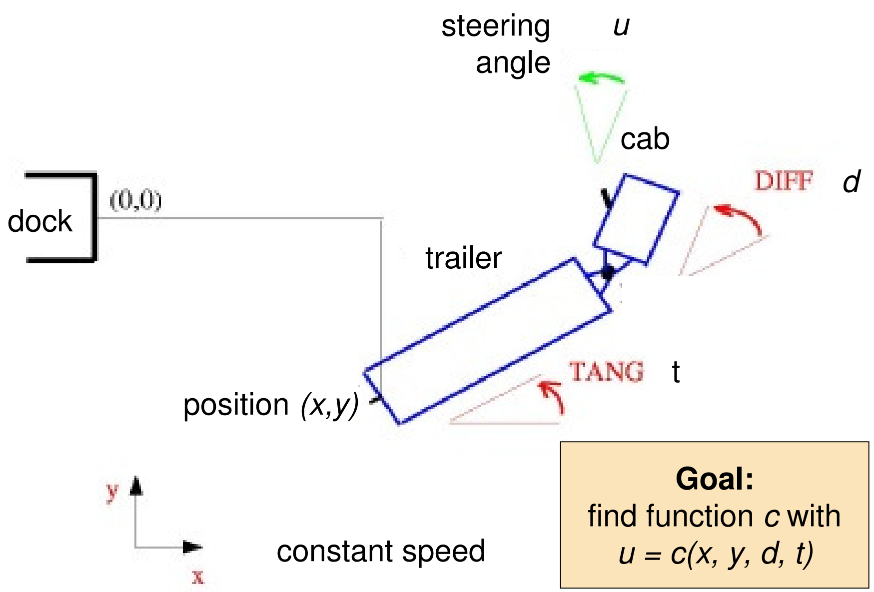
→ How to design the simulation / experiment?

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



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

Truck: Simulation

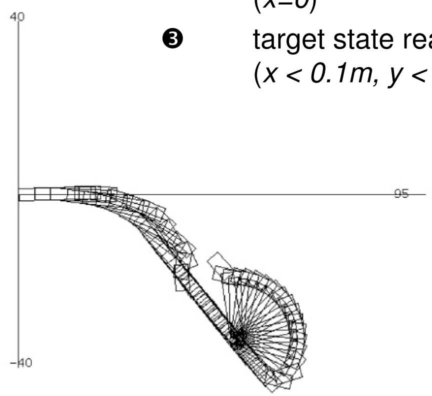
Given: start conditions x, y, t (implicitly: $d = 0$)

Algorithm: execute solution c for discrete time steps until

❶ time runs out

❷ the trailer crashes into the loading dock ($x=0$)

❸ target state reached ($x < 0.1m$, $y < 0.42m$, $t < 0.12$)



Truck: Fitness Assignment

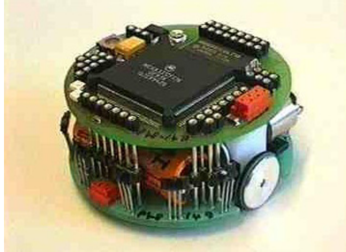
Given: eight start conditions $(x_1, y_1, t_1), \ldots, (x_8, y_8, t_8)$

Algorithm:

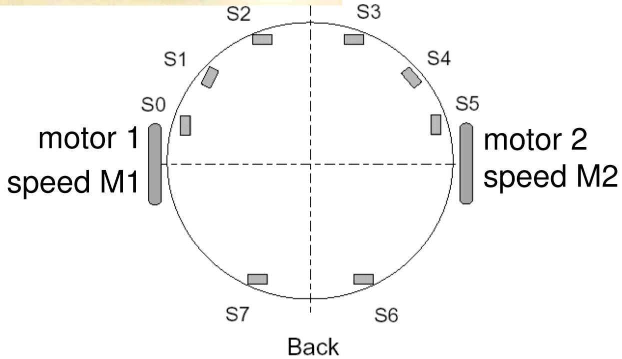
fitness F = 0
for each start condition do
    run simulation and obtain final x, y, t
    $F = F + x^2 + 2\,y^2 + 40/\pi\ t$
end

Note: Fitness is to be minimized here!

Learning Obstacle Avoiding Behavior



Goal:
find function c for the motor speeds M1, M2:
(M1, M2) = c(S0, …, S7)



[Nordin, Banzhaf (1997)]

Robot: Characteristics

Input: 8 proximity sensor measurements
S0, S1, …, S7 ∈ {0,1, …, 1023}
higher values = closer to an object

Output: 2 motor speed settings
M1, M2 ∈ {0,1, …, 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 *S0, S1, …, S7* from the robot

determine the speeds for both motors by applying the solution under evaluation:
(M1, M2) = c(S0, S1, …, S7)

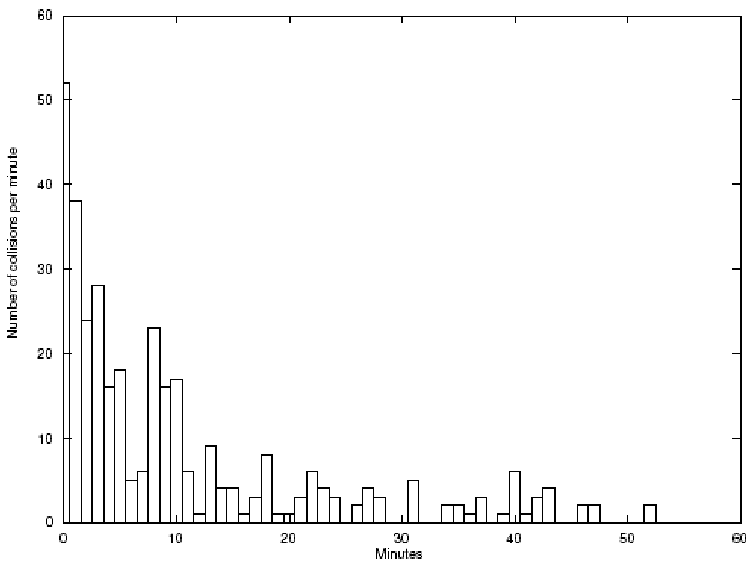
run robot for *400ms* with motorspeeds *M1, M2*

retrieve sensor measurements *S0’, …, S7’*

fitness *F* =

$$16\,(\underbrace{M1 + M2}_{\text{high speed}} - \underbrace{|\,M1 - M2\,|}_{\text{going straight}}) - \underbrace{(S0' + S1' + \ldots + S7')}_{\text{far away from any obstacle / wall}}$$

Robot: Learning Curve



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

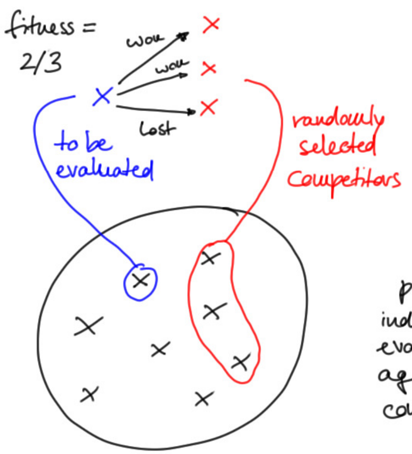
Strategy = function that maps one game situation into another one (one legal move)

TicTacToe:

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

Variants of Competitive Fitness Assignment

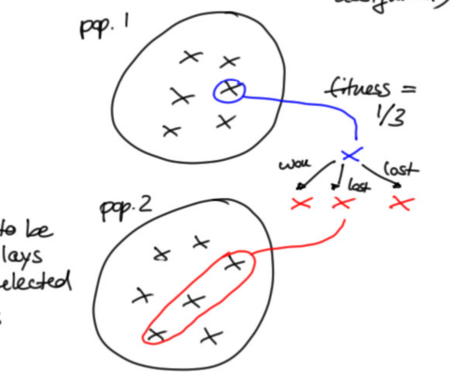
Single population



principle: individual to be evaluated plays against selected competitors

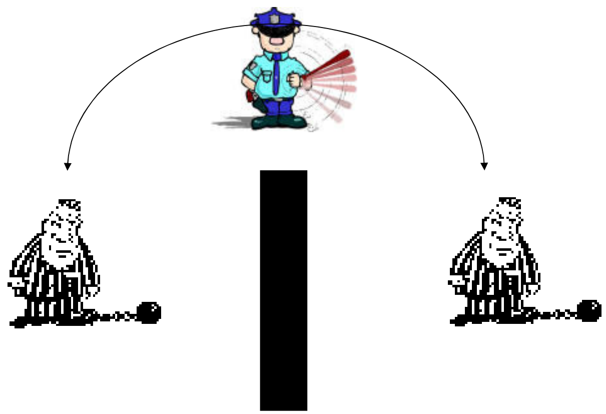
Two populations (coevolution)

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



The Prisoner's Dilemma

questioned separately



Pay-Off Matrix

	He Denies	He Confesses
You Deny	Both serve six months	He goes free; you serve ten years
You Confess	He serves ten years; you go free	Both serve five years

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

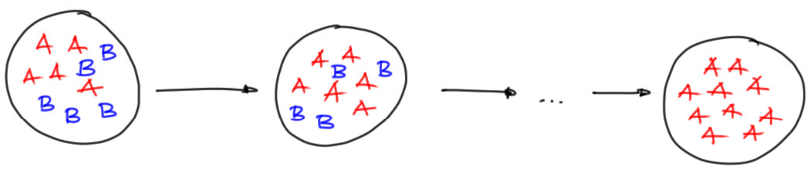
objective space decision space

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

Idea: incorporate density information into fitness

The Problem: Genetic Drift

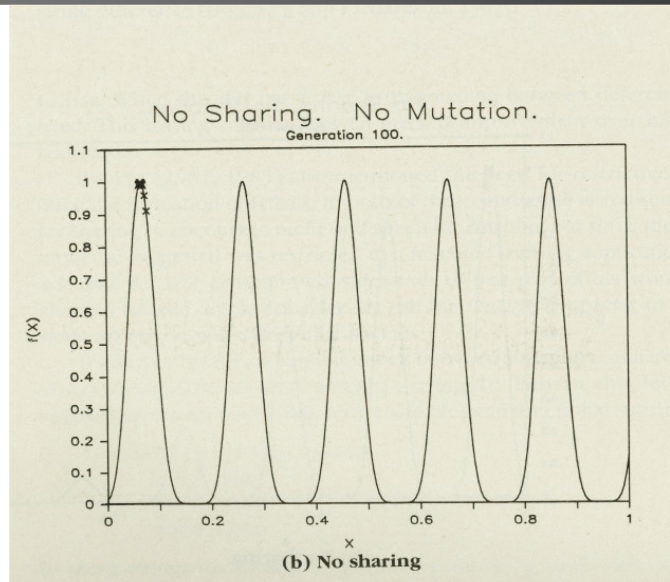
Genetic drift denotes a phenomenon in Biology where random changes in the allele frequency (allele = "different values a gene can take") can observed due to sampling errors in finite small populations.



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



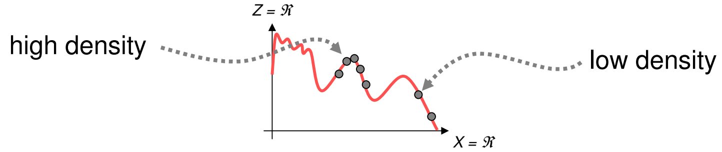
(b) No sharing

[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



- modify the fitness of an individual y_i using D_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:

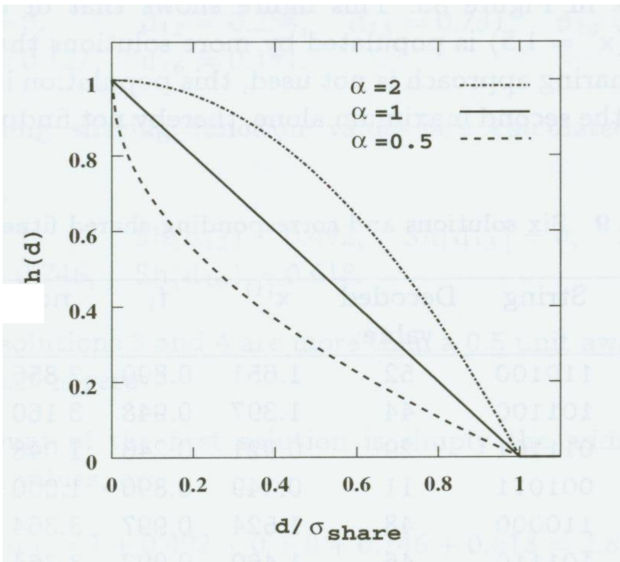
$$h(d(\mathbf{x}_i, \mathbf{x})) = \begin{cases} 1 - (d(\mathbf{x}_i, \mathbf{x})/\sigma_{\text{share}})^{\alpha} & \text{if } d(\mathbf{x}_i, \mathbf{x}) < \sigma_{\text{share}} \\ 0 & \text{else} \end{cases}$$

  where α (usually set to 1) and σ_{share} are user-defined
- modified fitness where D_i is the density around individual x_i:

$$D_i = \sum_{\mathbf{x} \in M} h(d(\mathbf{x}_i, \mathbf{x}))$$

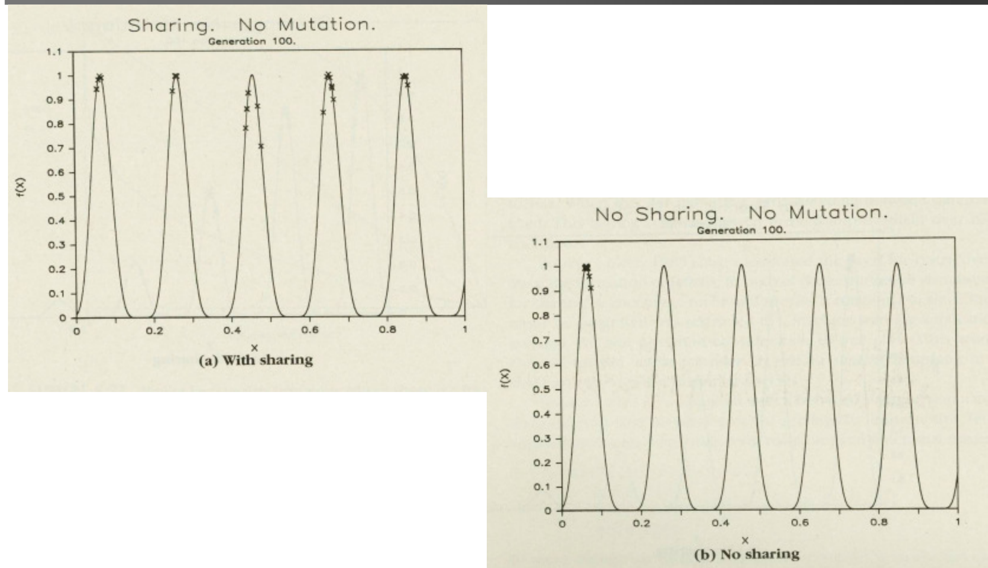
  and $F_i = F_i' / D_i$ where F_i' is the original fitness value

Fitness Sharing: Possible h Functions



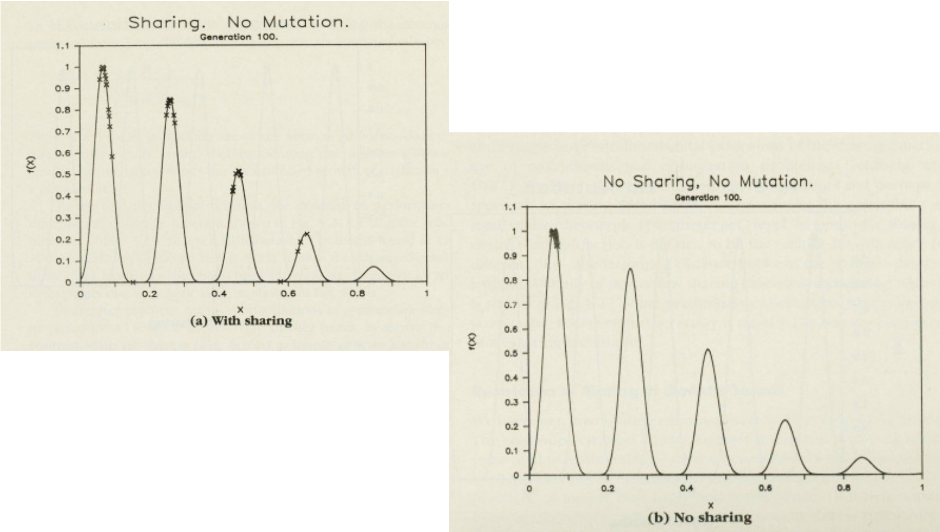
[Deb (2001)]

The Effect of Fitness Sharing



[Goldberg (1989)]

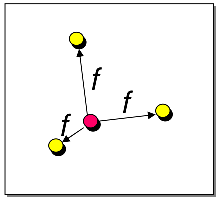
The Effect of Fitness Sharing (Cont'd)



Types of Diversity Preservation Techniques

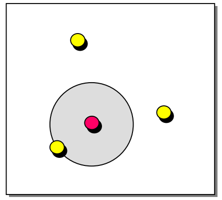
Kernel

density estimate = sum of *f* values where *f* is a function of the distance



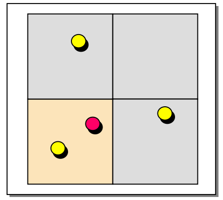
Nearest neighbor

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 k-th nearest neighbor

$D_i = h(d(x_i, x^*))$ where x^* k-th nearest individual to x_i in M

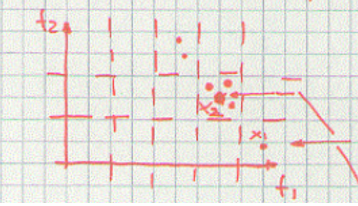
② Kernel method

$D_i = \sum_{x \in M} h(d(x_i, x))$

h can be chosen individually
d is distance function

③ Histogram method

uses grid in objective space, e.g.



$D_i = h(\text{count}(x_i))$

$\text{count}(x_i)$ = # individuals in the same box

$\text{count}(x_1) = 1$

$\text{count}(x_2) = 4$

Hybridization: General Considerations

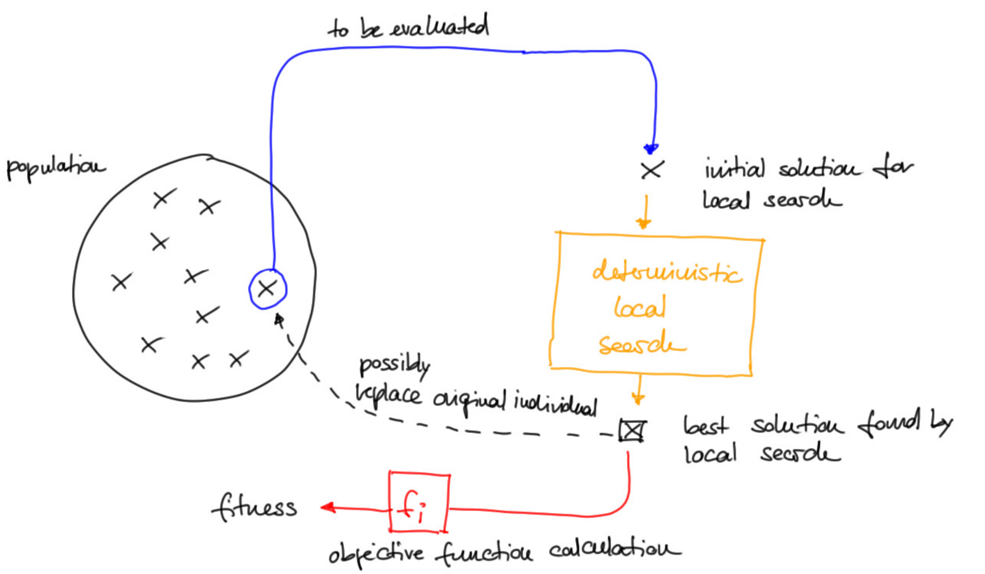
Idea: Combine a general global search strategy such as an evolutionary algorithm with a problem-specific heuristic or deterministic local search strategy

→ 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



A Usual Problem: Using Time Resources Efficiently

GSA: global search algorithm —solution→ **PLSA:** parameterized local search algorithm

←refined solution—

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

p_{min} ——————— p_{max}

low ← { accuracy / run-time } → high

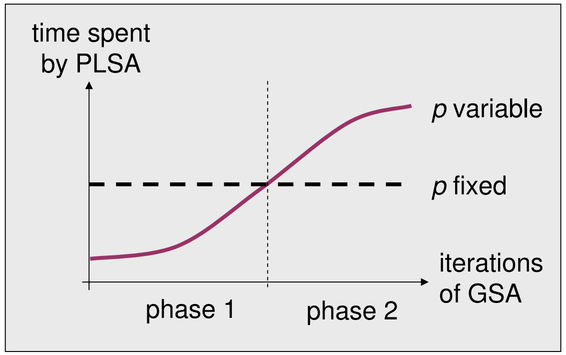
How to generate maximum quality solution in given PLSA time budget?

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)
$T_{\max}$ (maximum time budget)
Output: 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 $T > T_{\max}$ 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 $p_{\max}$ 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

❶ Fixed number of RSA iterations t_p per parameter p_i:

- $T_{\max} \geq t_p NC(p_1) + t_p NC(p_2) + \ldots + t_p NC(p_n)$
- $t_p = \left\lfloor \frac{T_{\max}}{N \sum_{i=1}^{n} C(p_i)} \right\rfloor$

❷ Fixed amount of time T_p per parameter p_i:

- $T_p = T_{\max}/n$ and therefore $t_i NC(p_i) \leq T_p \quad \forall i = 1, \ldots, n$
- $t_i = \left\lfloor \frac{T_{\max}}{nNC(p_i)} \right\rfloor$

T_{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

❶ Fixed number of RSA iterations:

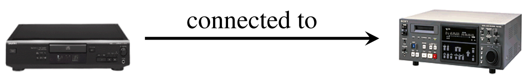
> Use next parameter value, if the quality of the best solution in M has not improved for t_{stag} RSA iterations

❷ Fixed amount of time:

> Use next parameter value, if the quality of the best solution in M has not improved for T_{stag} time units

Application Study (Details Omitted)

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



compact disc player (*44.1kHz*) digital audio tape (*48kHz*)

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: $T_{max} = 5h$ (Sun Ultra 60)
- Stagnation parameters: $t_{stag} = 10$ and $T_{stag} = 900s$
- Population size: $N = 100$

Results I: Quality of The Best Solution Found

keeping *p* constant		
p = 1 (minimum)		*0.3394*
p = 153		*0.3308*
p = 305		*0.3637*
p = 457		*0.3622*
p = 612 (maximum)		*0.3692*
static heating		
p = 1, 153, 305, 457, 612	(fixed number of iterations)	*0.3558*
p = 1, 31, 62, 92, 123,153	(fixed number of iterations)	*0.2848*
p = 1, 153, 305, 457, 612	(fixed amount of time)	*0.3024*
p = 1, 31, 62, 92, 123,153	(fixed amount of time)	*0.2609*
dynamic heating		
p = 1, 153, 305, 457, 612	(fixed number of iterations)	*0.2992*
p = 1, 31, 62, 92, 123,153	(fixed number of iterations)	*0.2739*
p = 1, 153, 305, 457, 612	(fixed amount of time)	*0.2985*
p = 1, 31, 62, 92, 123,153	(fixed amount of time)	*0.2558*

Results II: Number of Iterations Spent Per Parameter

	Iterations per *p* value				
	1	*153*	*305*	*457*	*612*
keeping *p* constant					
p = 1 (minimum)	900	0	0	0	0
p = 153	0	73	0	0	0
p = 305	0	0	22	0	0
p = 457	0	0	0	12	0
p = 612 (maximum)	0	0	0	0	10
static heating *(p = 1,…,612)*					
fixed number of iterations	4	4	4	4	4
fixed amount of time	176	14	4	2	2
dynamic heating *(p = 1,…,612)*					
fixed number of iterations	99	33	12	0	0
fixed amount of time	276	11	4	1	4

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Computer Engineering and Networks Laboratory

Bio-inspired Optimization and Design

Eckart Zitzler

3. Basic Design Issues

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 = probability that individual i 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

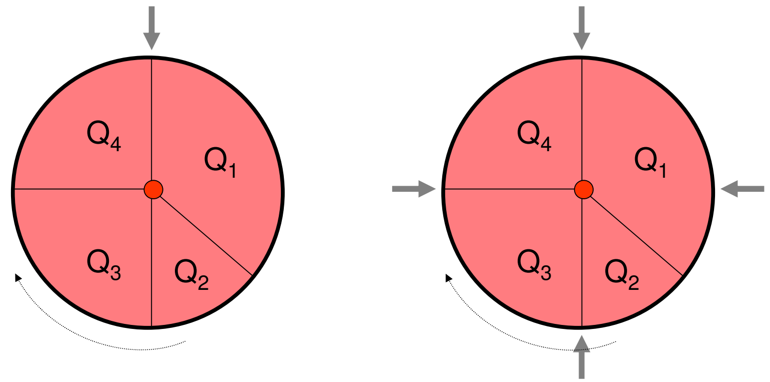
- **fitness proportionate:** (scaling dependent, e.g., adding a constant factor changes the sampling rates)
  $Q_i = F_i \,/ \sum F_k$
  Example: $F_1 = 1, F_2 = 2, F_3 = 3 \quad \Rightarrow \quad Q_1 = \frac{1}{6}, Q_2 = \frac{1}{3}, Q_3 = \frac{1}{2}$
  $F_1 = 1001, F_2 = 1002, F_3 = 1003 \quad \Rightarrow \quad Q_1 = \frac{1001}{3006}, Q_2 = \frac{1002}{3006}, Q_3 = \frac{1003}{3006}$
- **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: $P_i = 1 - e^{-R_i}$

  $Q_i = P_i \,/ \sum P_k$
- **threshold:**

$$Q_i = \begin{cases} 1/T & \text{if individual i is among the T best individuals} \\ 0 & \text{else} \end{cases}$$

Sampling Methods: Principle

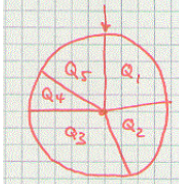
Roulette wheel | **Stochastic universal sampling (SUS)**



spin 4 times | spin once

Sampling Methods: Details

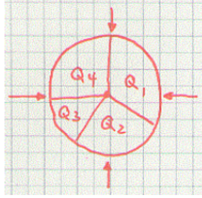
① roulette wheel



uniformly randomly choose $u \in [0,1]$
select x_i such that
$Q_1 + Q_2 + \dots + Q_{i-1} < u \leq Q_1 + Q_2 + \dots + Q_i$

may lead to high variance in the selection outcome

② SUS (stochastic universal sampling)

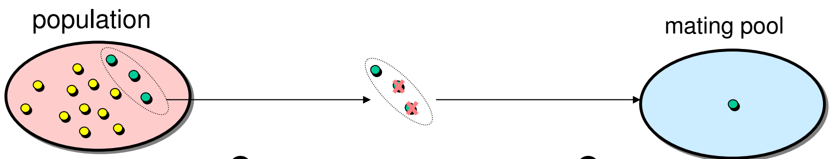


uniformly randomly choose $u \in [0,1)$
$s = 0$
for $i := 1$ to N
  $s = s + Q_i \cdot N$
  while $s > u$
    $u = u + 1$
    select x_i
  end
end

lower variance: the number of times a specific individual is selected varies only by one

Tournament Selection

= integrated sampling rate assignment and sampling



❶ uniformly choose T individuals at random independently of fitness

❷ compare fitness and copy best individual in mating pool

T = tournament size (binary tournament selection means T=2)

Deterministic Selection Schemes

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

(μ, λ) or $(\mu + \lambda)$

where μ denotes the population size $|M|$
λ denotes the number of offspring, i.e., $|M'| = |M''| = \lambda$
, means the new population is formed by the best μ individuals from M''
+ means the new population is formed by the best μ individuals from $M + M''$ (union of parents and children)

Examples: $(1+1) \hat{=}$ local search
$(\mu+1) \hat{=}$ steady-state evolutionary algorithm (1 individual / generation)
$(\mu, \mu) \hat{=}$ regular genetic algorithm

Note: in evolution strategies, usually uniform mating 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 =

  $I = (\underline{F}_{sel} - \underline{F}) / \sigma$

  where
  $\underline{F}_{sel}$ = average fitness in population after selection
  $\underline{F}$ = average fitness in population before selection
  σ = 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...

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Computer Engineering and Networks Laboratory

Bio-inspired Optimization and Design

Eckart Zitzler

3. Basic Design Issues

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.

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

bit flip mutation

| 1 | 0 | 1 | 1 | . . . . . . | 0 | 0 | 1 |

flip with prob. p_m

prob. k bits mutated $= \binom{n}{k} p_m^k (1-p_m)^{n-k}$

k follows a binomial distribution, and therefore

expected number of bits mutated $= n \cdot p_m$

$\Rightarrow$ common setting $p_m = \frac{1}{n}$

With $p_m = \frac{1}{n}$ the expected number of mutated bits is one per individual. This is usually used for (λ, μ) strategies; in the case of $(\lambda + \mu)$ environmental selection, greater mutation rate such as $p_m = 4/n$ often yield better results.

Mutation: Discrete Search Spaces In General

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

vector mutation

$Y = Y_1 \times Y_2 \times \ldots \times Y_n$

| y_1 | y_2 | y_3 | . . . . . . | y_n |

replace with prob. p_m with any $y_3' \in Y_3 \setminus \{y_3\}$

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:

| 1 | 2 | 3 | 4 | 5 | 6 | → | 1 | 4 | 3 | 2 | 5 | 6 |

swap

Scramble:

| 1 | 2 | 3 | 4 | 5 | 6 | → | 1 | 3 | 4 | 2 | 5 | 6 |

rearrange

Invert:

| 1 | 2 | 3 | 4 | 5 | 6 | → | 1 | 4 | 3 | 2 | 5 | 6 |

reverse

Insert:

| 1 | 2 | 3 | 4 | 5 | 6 | → | 1 | 4 | 2 | 3 | 5 | 6 |

move

Mutation: Real Vectors

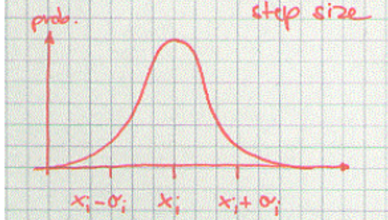
- In principle, real vectors also have a discrete representation on the computers and therefore could be treated as integer vectors. However, replacing a real value by an **arbitrary** one is usually not effective.
- An alternative (many other mutation operators for real vectors exist):

$(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$

$x_i' = x_i + N(0, \sigma_i)$

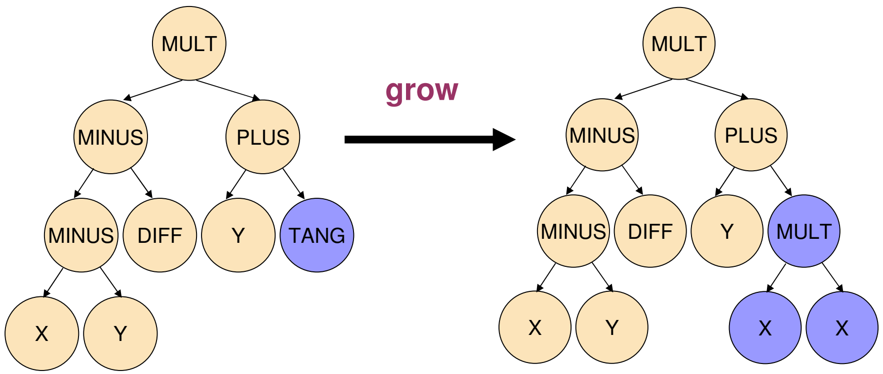
step size

where $N(0, \sigma_i)$ is a random number chosen using a Gaussian distribution with mean 0 and standard deviation σ_i

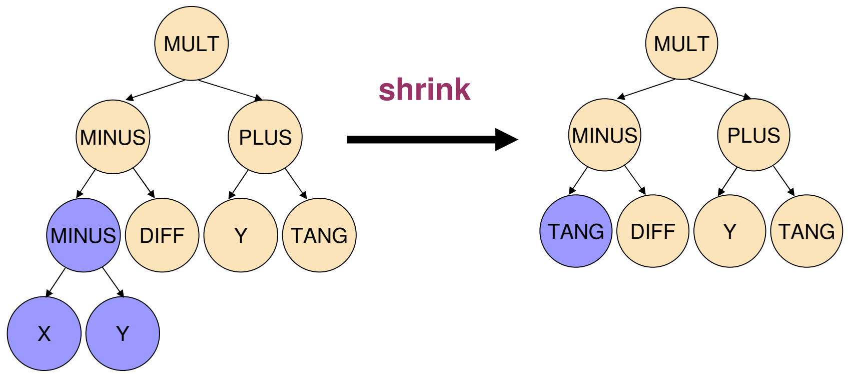


Note that the mutation rate p_m is replaced here by a vector of step sizes $(\sigma_1, \sigma_2, \ldots, \sigma_n)$; sometimes, there is only one global step size σ with $\sigma = \sigma_1 = \sigma_2 = \ldots = \sigma_n$.

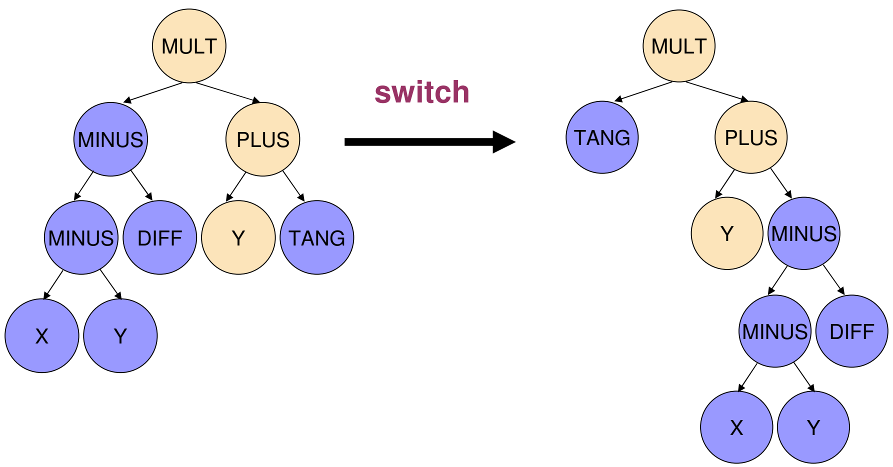
Mutation Operators on Trees: Grow



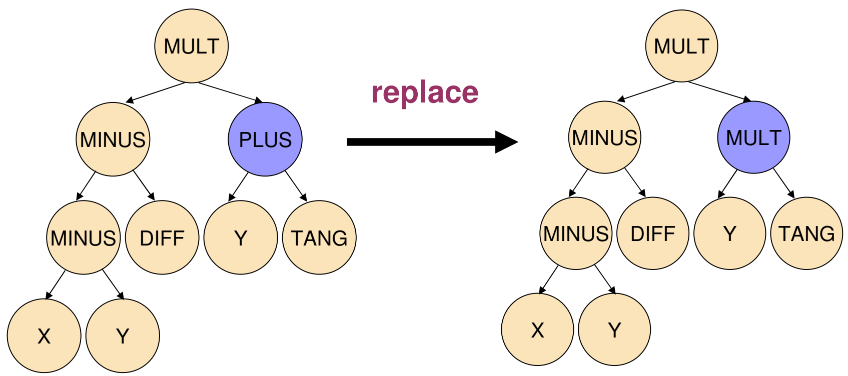
Mutation Operators on Trees: Shrink



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:

$$x'' = recomb(x, x') \Rightarrow d(x, x'') \leq d(x, x') \wedge d(x', x'') \leq d(x, x')$$

- 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: 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 new real numbers using the two values from the parents. A commonly used operator is

intermediate recombination

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

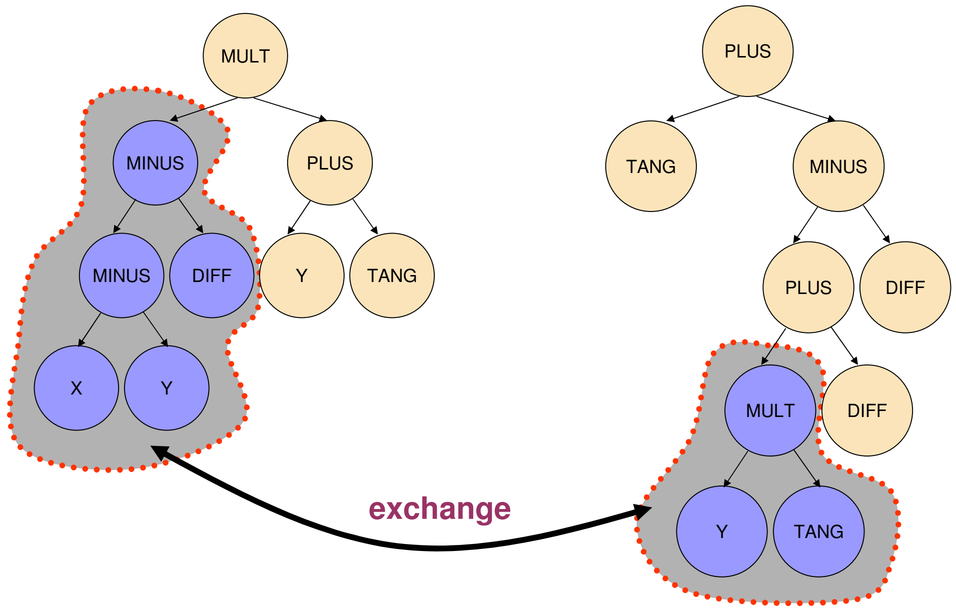
child: $(x''_1, x''_2, \ldots, x''_n) \in \mathbb{R}^n$ with

$x''_i = x_i + u \cdot (x'_i - x_i)$ where

u is a uniform random variable over $[0,1]$

Several other operators on real vectors were suggested, which, however, cannot be discussed here.

Recombination of Trees



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Engineering
and Networks Laboratory

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 (not part of the exam)

Biclustering: The Problem in A Nutshell

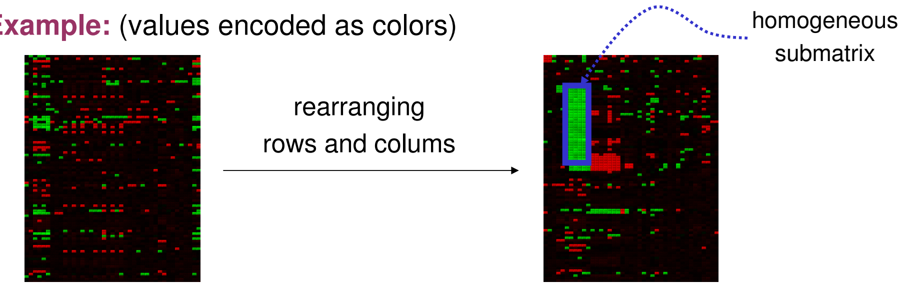
Given:

data matrix $E^{m \times n}$ with $e_{ij} \in \Re$

Goal:

find submatrix in E that minimizes a given score f_d, formally:
$(X, \Re, f_d, \leq)$ where $X = 2^{\{1, \ldots, m\}} \times 2^{\{1, \ldots, n\}}$ (2^A denotes the power set of A); each submatrix $x \in X$ is a pair (R, C) of rows R and columns C

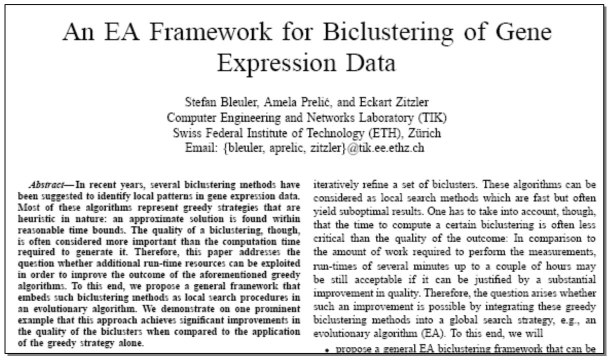
Example: (values encoded as colors)



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



An EA Framework for Biclustering of Gene Expression Data

Stefan Bleuler, Amela Prelić, and Eckart Zitzler
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH), Zürich
Email: {bleuler, aprelic, zitzler}@tik.ee.ethz.ch

Abstract—In recent years, several biclustering methods have been suggested to identify local patterns in gene expression data. Most of these algorithms represent greedy strategies that are heuristic in nature: an approximate solution is found within reasonable time bounds. The quality of a biclustering, though, is often considered more important than the computation time required to generate it. Therefore, this paper addresses the question whether additional run-time resources can be exploited in order to improve the outcome of the aforementioned greedy algorithms. To this end, we propose a general framework that embeds such biclustering methods as local search procedures in an evolutionary algorithm. We demonstrate on one prominent example that this approach achieves significant improvements in the quality of the biclusters when compared to the application of the greedy strategy alone.

iteratively refine a set of biclusters. These algorithms can be considered as local search methods which are fast but often yield suboptimal results. One has to take into account, though, that the time to compute a certain biclustering is often less critical than the quality of the outcome: In comparison to the amount of work required to perform the measurements, run-times of several minutes up to a couple of hours may be still acceptable if it can be justified by a substantial improvement in quality. Therefore, the question arises whether such an improvement is possible by integrating these greedy biclustering methods into a global search strategy, e.g., an evolutionary algorithm (EA). To this end, we will

- propose a general EA biclustering framework that can be

The full paper can be found at the end of this chapter.

References

- S. Bleuler, A. Prelic, E. Zitzler (2004): An EA Framework for Biclustering of Gene Expression Data. Congress on Evolutionary Computation (CEC 2004), Portland, pp. 166-173, IEEE Press, Pisataway, NJ.
- D. Goldberg (1989): Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA.
- J. Koza (1992): Genetic Programming. The MIT Press, Cambridge, MA. **(Chapter 11)**
- Z. Michalewicz, D. Fogel (2000): How to Solve It: Modern Heuristics. Springer, Berlin.
- P. Nordin, W. Banzhaf (1997): Real time control of a khepera robot using genetic programming. Cybernetics and Control 26(3), pp. 533-561.
- E. Zitzler, J. Teich, and S. S. Bhattacharyya. Optimizing the Efficiency of Parameterized Local Search within Global Search: A Preliminary Study. Congress on Evolutionary Computation (CEC-2000), July 2000, pp. 365-372. IEEE Press.