Evolutionary Algorithms and its applications

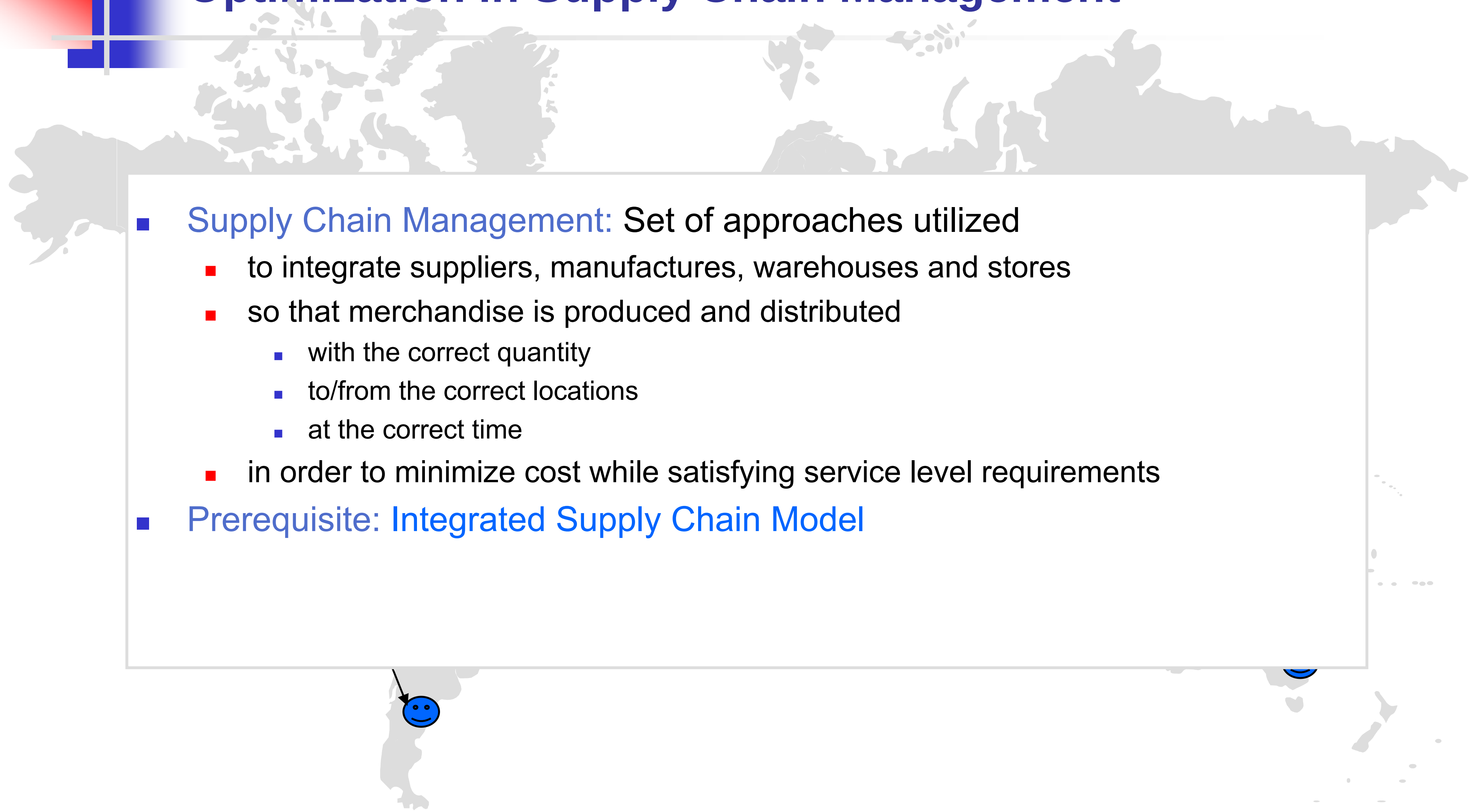
Heinrich Braun
BA Karlsruhe

Overview

- Introduction
- Overview Optimization Methods
- Evolutionary Algorithms

Optimization in Supply Chain Management

- Supply Chain Management: Set of approaches utilized
  - to integrate suppliers, manufactures, warehouses and stores
  - so that merchandise is produced and distributed
    - with the correct quantity
    - to/from the correct locations
    - at the correct time
  - in order to minimize cost while satisfying service level requirements
- Prerequisite: Integrated Supply Chain Model

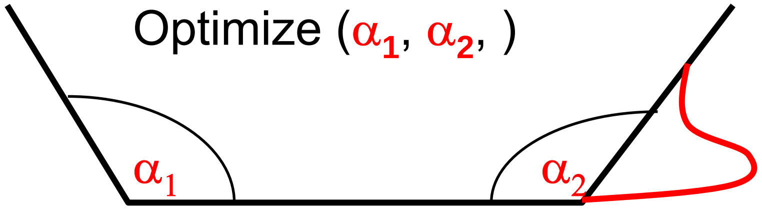


Supplier Plant DC ☺ Customer

Introduction: Nature versus Engineers

- Typical Engineering Approach for Optimization
  - Specify
    - Model of the real world problem
    - Objective Function for evaluating alternative solutions
  - Optimize the free parameters of the model
- Typical Failure
  - Model is simple enough to optimize
  - But too simple for good solutions

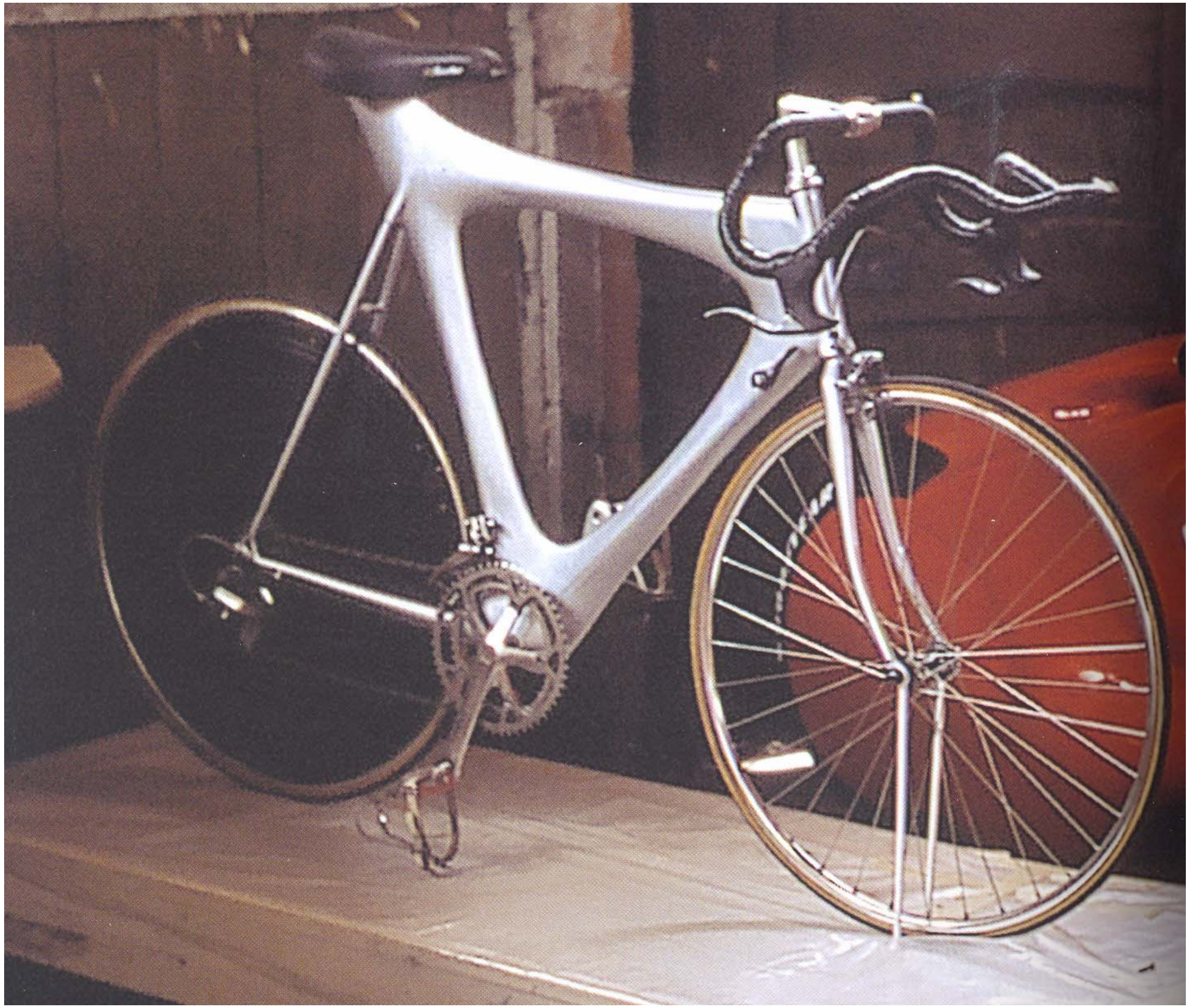
Ship Design



Optimize (α_1, α_2,)

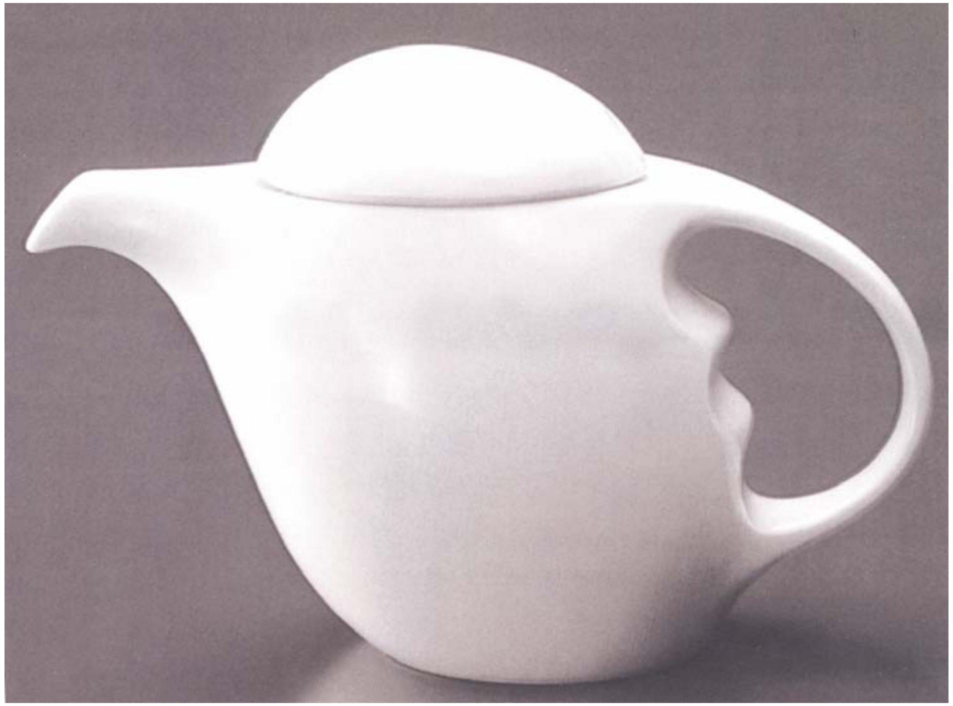
- Mind the difference
  - Engineers model with simple geometric: Straight lines, circles
  - Nature is not so simple minded!!

Natural Design by famous Designer Colani (Karlsruhe)

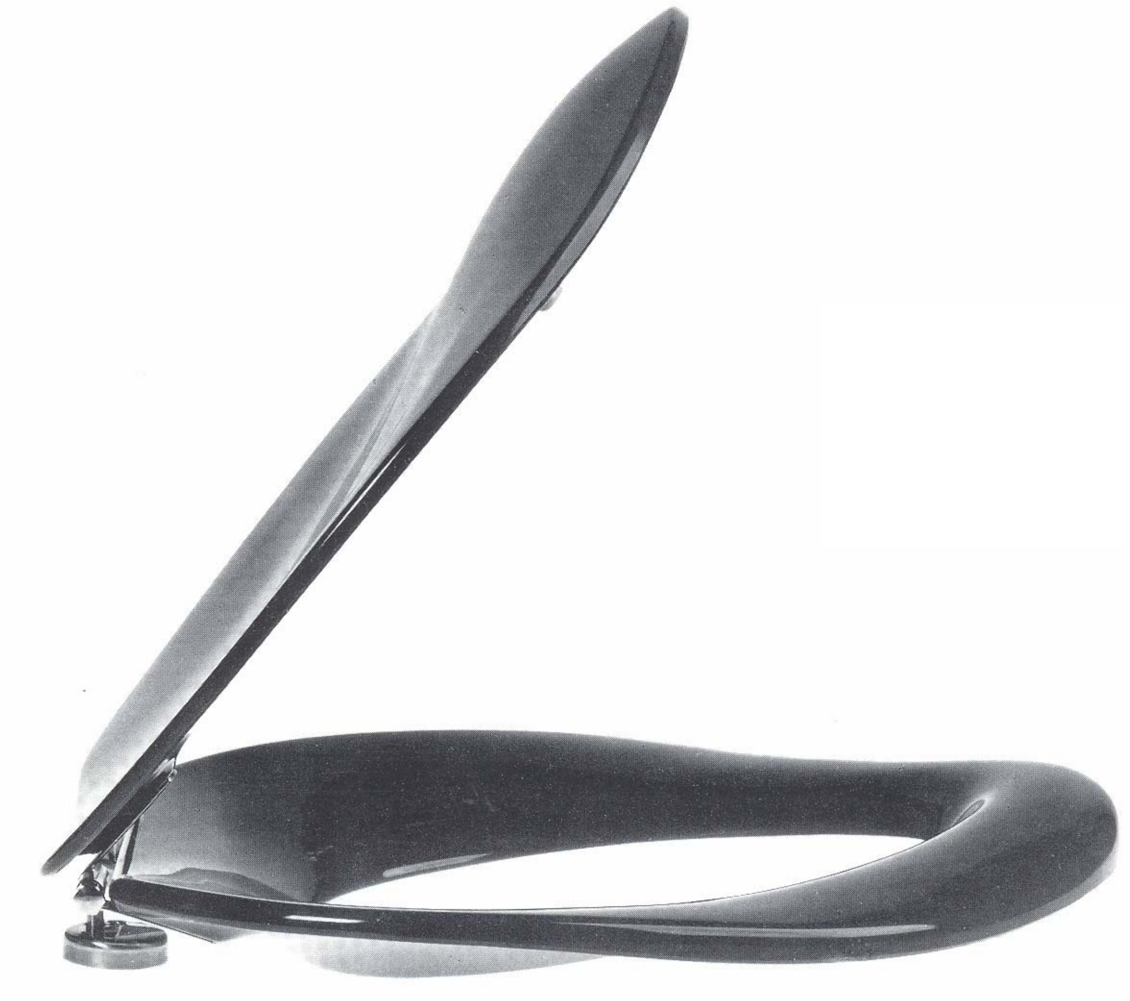


Natural Design by famous Designer Colani (Karlsruhe)

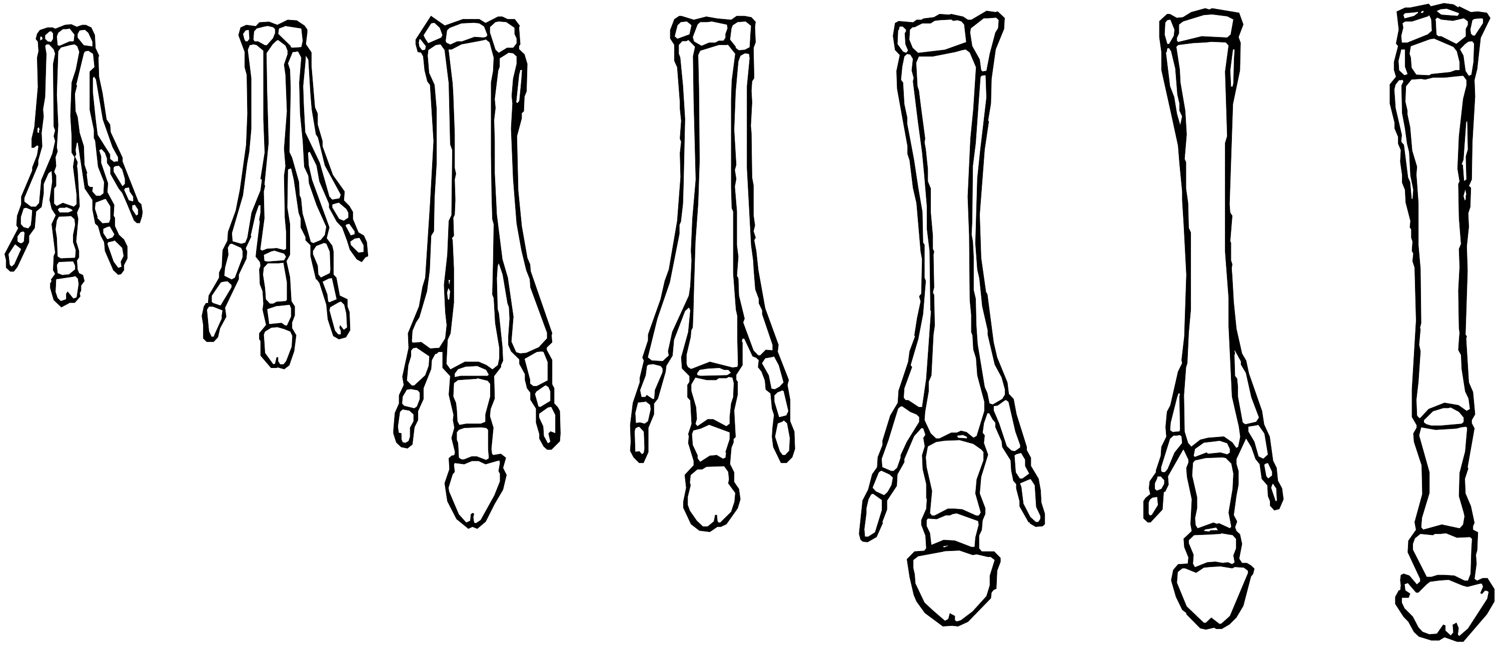
Natural Design by famous Designer Colani (Karlsruhe)



Natural Design by famous Designer Colani (Karlsruhe)

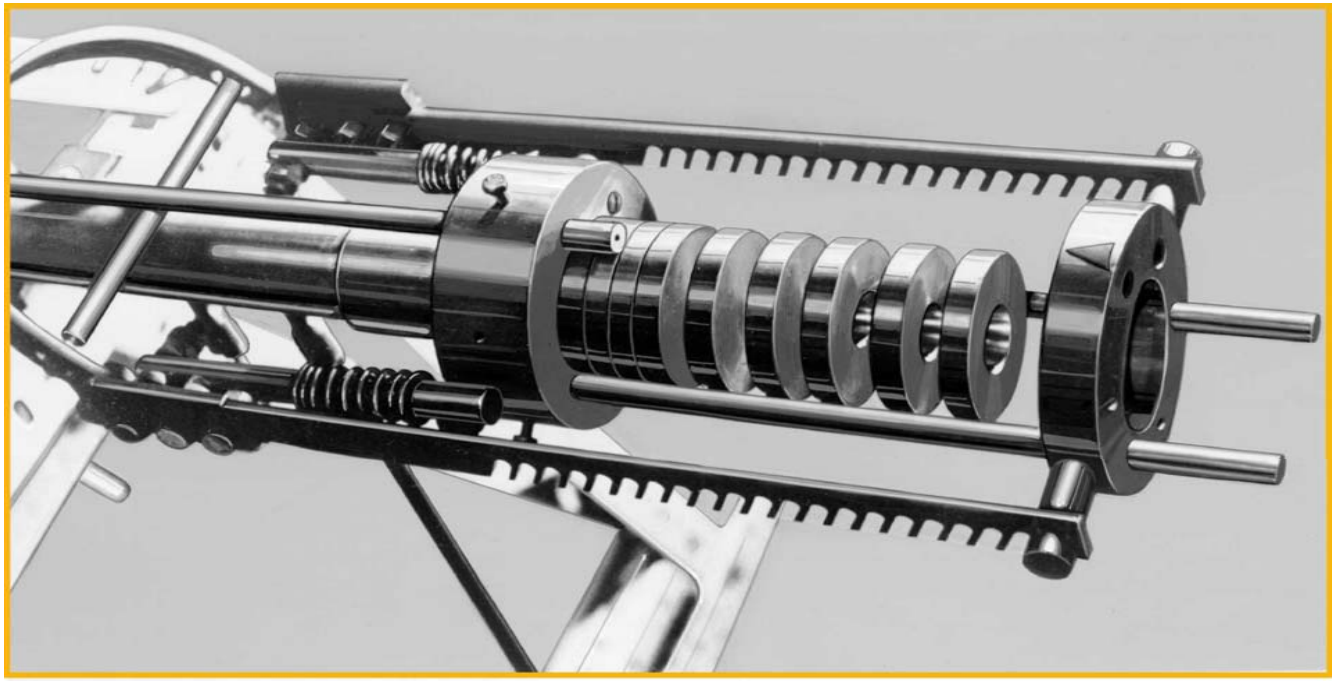


Evolution of the horse foot



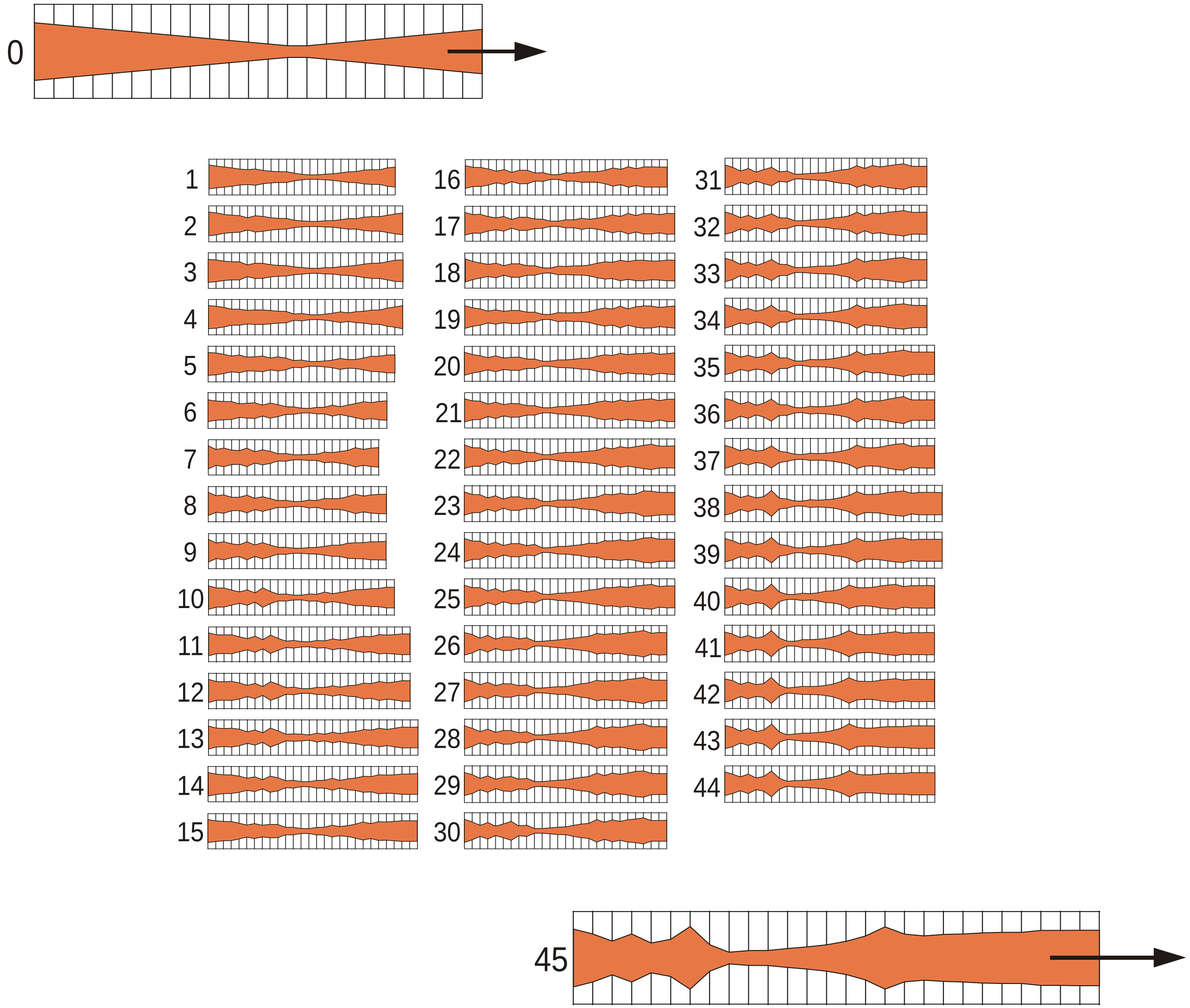
From Eohippus to Equus (60 Millionen Years)

Evolution of a water steam pipe in 1965

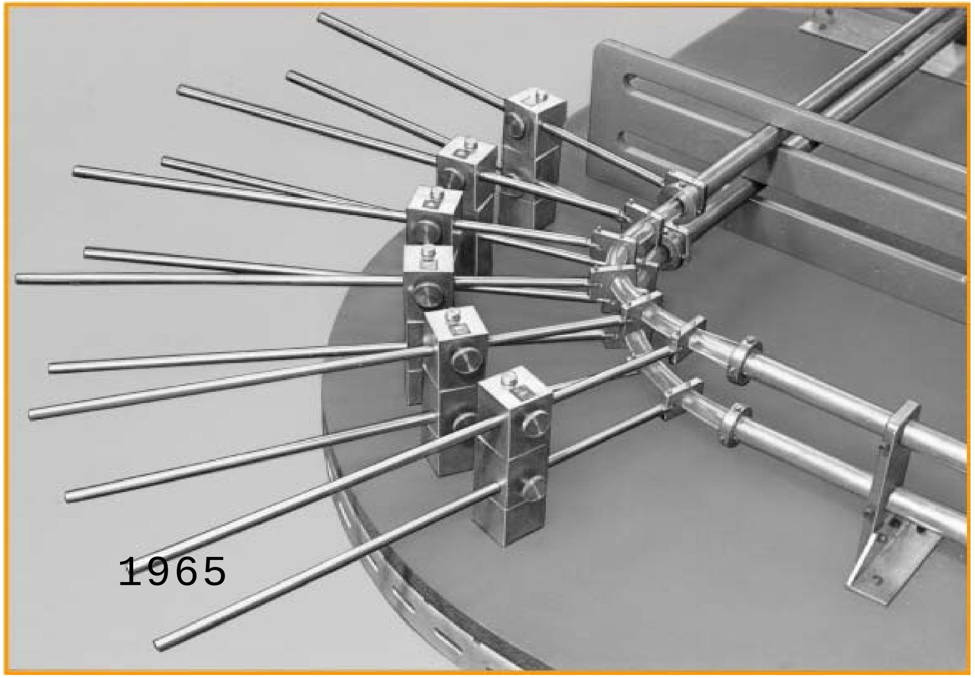


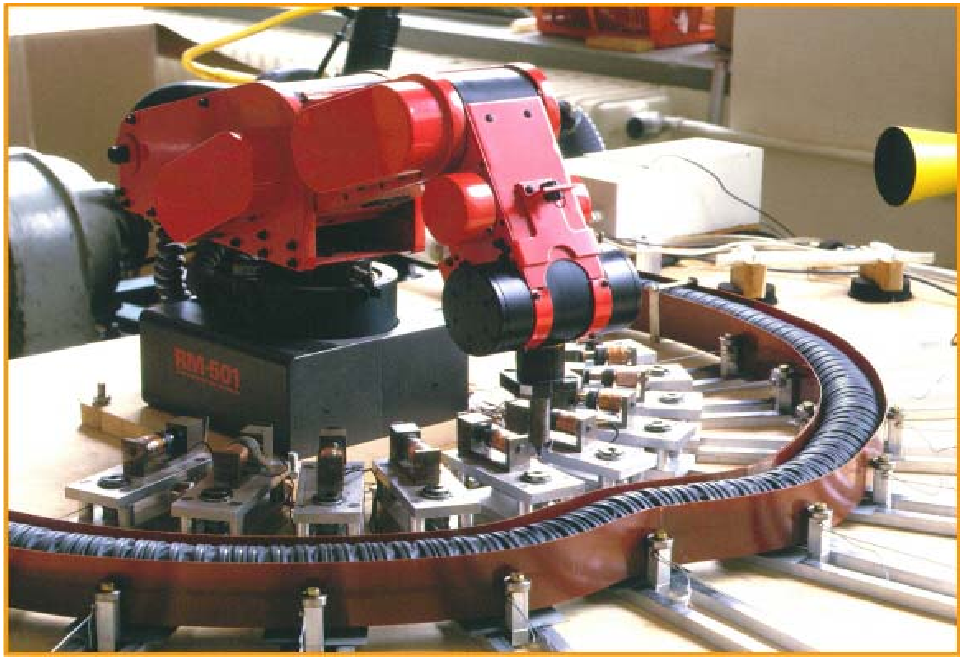
by Schwefel

Evolution of a steam pipe in 1965



Evolution of a curvature by Rechenberg





1980

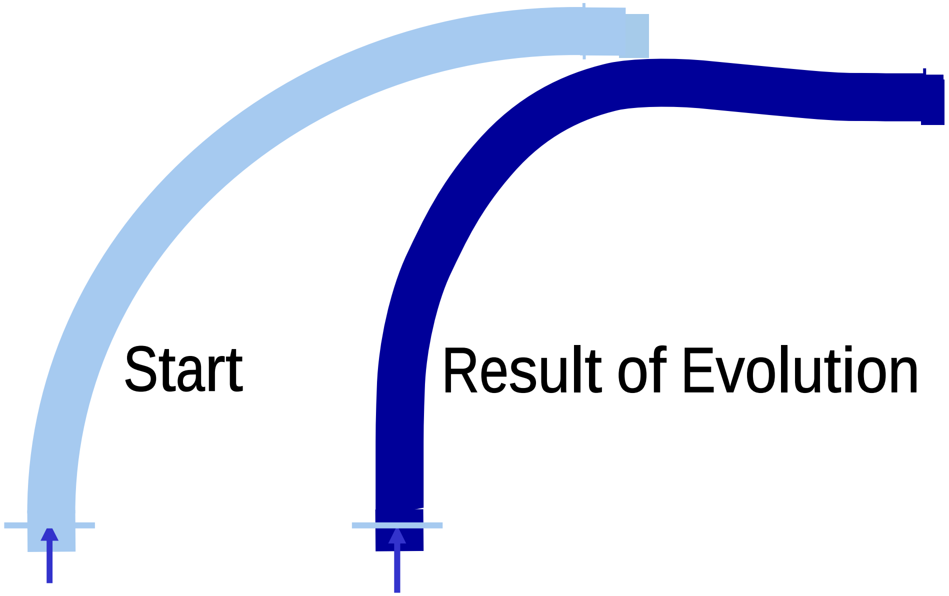
Manuel Adjustments

-> 6 hand driven contrals

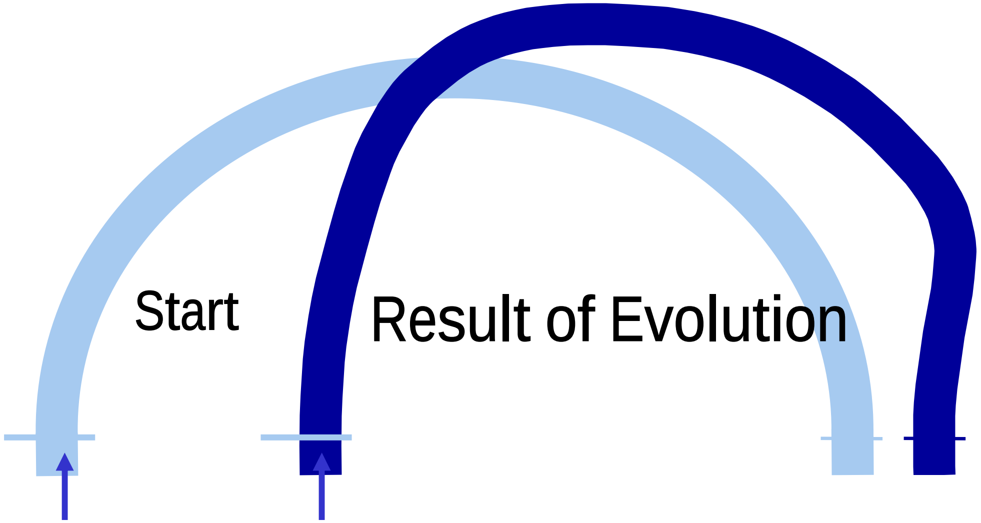
Automatic Adjustments

-> 10 robot driven controls

Evolution of a curvature by Rechenberg



Optimal 90°- solution



Optimal 180°- solution

Optimization Methods

model too simpel

exponential time

- Local Search
  - Deterministic
    - Hill Climbing
    - Gradient Descent
    - Tabu Search
  - Probabilistic
    - Simulated Annealing
    - Iterated local Search

- Global Search
  - Deterministic
    - Linear Optimization
    - Branch&Bound, Divide&Conquer
    - Dynamic Programming
  - Probabilistic
    - **Genetic algorithms**
    - **Evolution Strategies**

Production Planning

- **Decison Variables**
  - x_A = lot size for product A $\in |R^+$
  - x_B = lot size for product B $\in |R^+$

- **Objective Function**
  - Maximize $200x_A + 400x_B$ **Profit**

- **Constraints**
  - Assembling: $4x_A + 6x_B \leq 120$
  - Painting: $2x_A + 6x_B \leq 72$

Resource consumption

Resource capacity

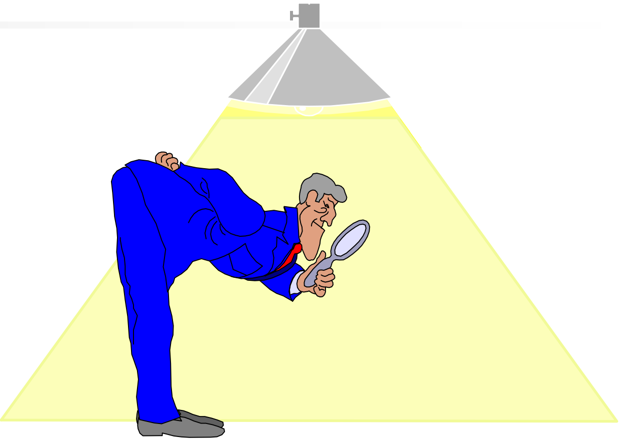
Problem

Linear model (objective function, constraints)

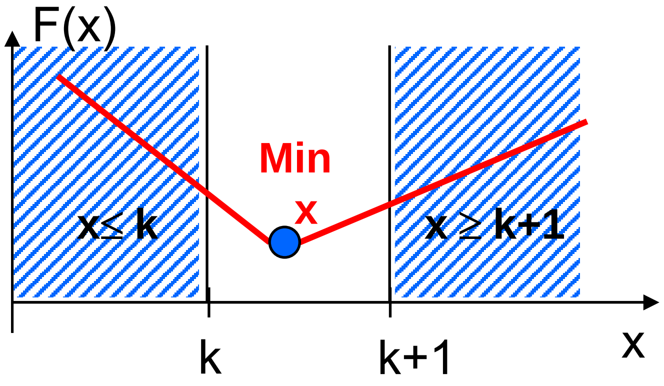
Integer solutions are NP-hard -> Branch&Bound

Classic Operations Research

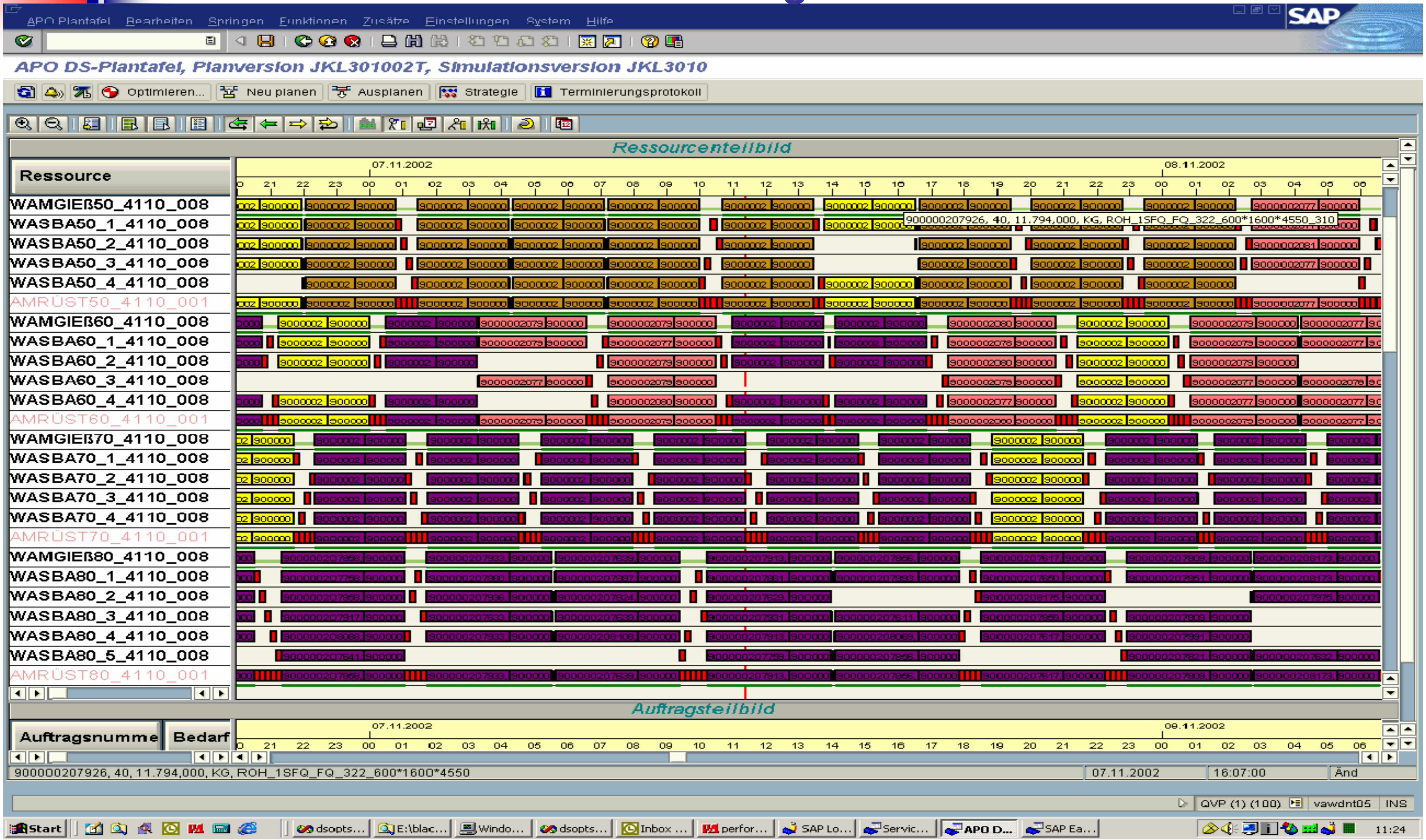
- **Linear Model**
  - Linear objective function
  - Linear constraint
  - Efficient algorithms (**Simplex Algorithm**)



- **Complex: modelling “integer” variables**
  - Optimize by relaxation
    - neglecting „integer“ constraint
    - Search optimum in both branches
  - **Branch&Bound method**
  - NP-hard



Detailed Production Planning



APO DS-Plantafel, Planversion JKL301002T, Simulationsversion JKL3010

Ressource

WAMGIEB50_4110_008
WASBA50_1_4110_008
WASBA50_2_4110_008
WASBA50_3_4110_008
WASBA50_4_4110_008
AMRÜST50_4110_001
WAMGIEB60_4110_008
WASBA60_1_4110_008
WASBA60_2_4110_008
WASBA60_3_4110_008
WASBA60_4_4110_008
AMRÜST60_4110_001
WAMGIEB70_4110_008
WASBA70_1_4110_008
WASBA70_2_4110_008
WASBA70_3_4110_008
WASBA70_4_4110_008
AMRÜST70_4110_001
WAMGIEB80_4110_008
WASBA80_1_4110_008
WASBA80_2_4110_008
WASBA80_3_4110_008
WASBA80_4_4110_008
WASBA80_5_4110_008
AMRÜST80_4110_001

900000207926, 40, 11.794,000, KG, ROH_1SFQ_FQ_322_600*1600*4550_310

900000207926, 40, 11.794,000, KG, ROH_1SFQ_FQ_322_600*1600*4550 | 07.11.2002 | 16:07:00 | Änd

QVP (1) (100) | vawdnt05 | INS

Knappsack Problem

- **Decision Variables**
  - $x_{i,} \in \{0,1\}$
  - $x_{i,} = 1 \quad \Leftrightarrow$ take object i

- **Objective Function**
  - Maximize $120\,x_1 + 175x_2 + 200\,x_3 + 150\,x_4 + 30\,x_5 + 60\,x_6$

- **Constraint**
  - Limitation $20x_1 + 35x_2 + 50\,x_3 + 50\,x_4 + 15\,x_5 + 60x_6 \leq 100$

Truck Load Building

- **Decision Variables**
  - $x_{i,} \in \{0,1\}$
  - $x_{i,} = 1 \Leftrightarrow$ load order i on the truck

- **Objective function**
  - Maximize $10\,x_1 + 20x_2 + 50\,x_3 + 200\,x_4 + 150\,x_5 + 250\,x_6 + 150\,x_7$

- **Constraints**
  - Weight $0{,}4\,x_1 + 0{,}7x_2 + 0{,}2\,x_3 + 2\,x_4 + 2\,x_5 + x_6 + 3\,x_7 \leq 5$
  - Volumen $0{,}4\,x_1 + 0{,}2\,x_2 + 3\,x_3 + 4\,x_4 + 3\,x_5 + 5_6 + 0{,}9\,x_7 \leq 5$

Truck Load Building
= Multidimensional Knappsack Problem

- **Decision Variables**
  - $x_{i,} \in \{0,1\}$
  - $x_{i,} = 1 \Leftrightarrow$ load order i on the truck
- **Objective function**
  - Maximize $\sum_i w_i x_i$
- **Constraints**
  - Weight $\sum_i G_i x_i \leq G$
  - Volumen $\sum_i V_i x_i \leq V$

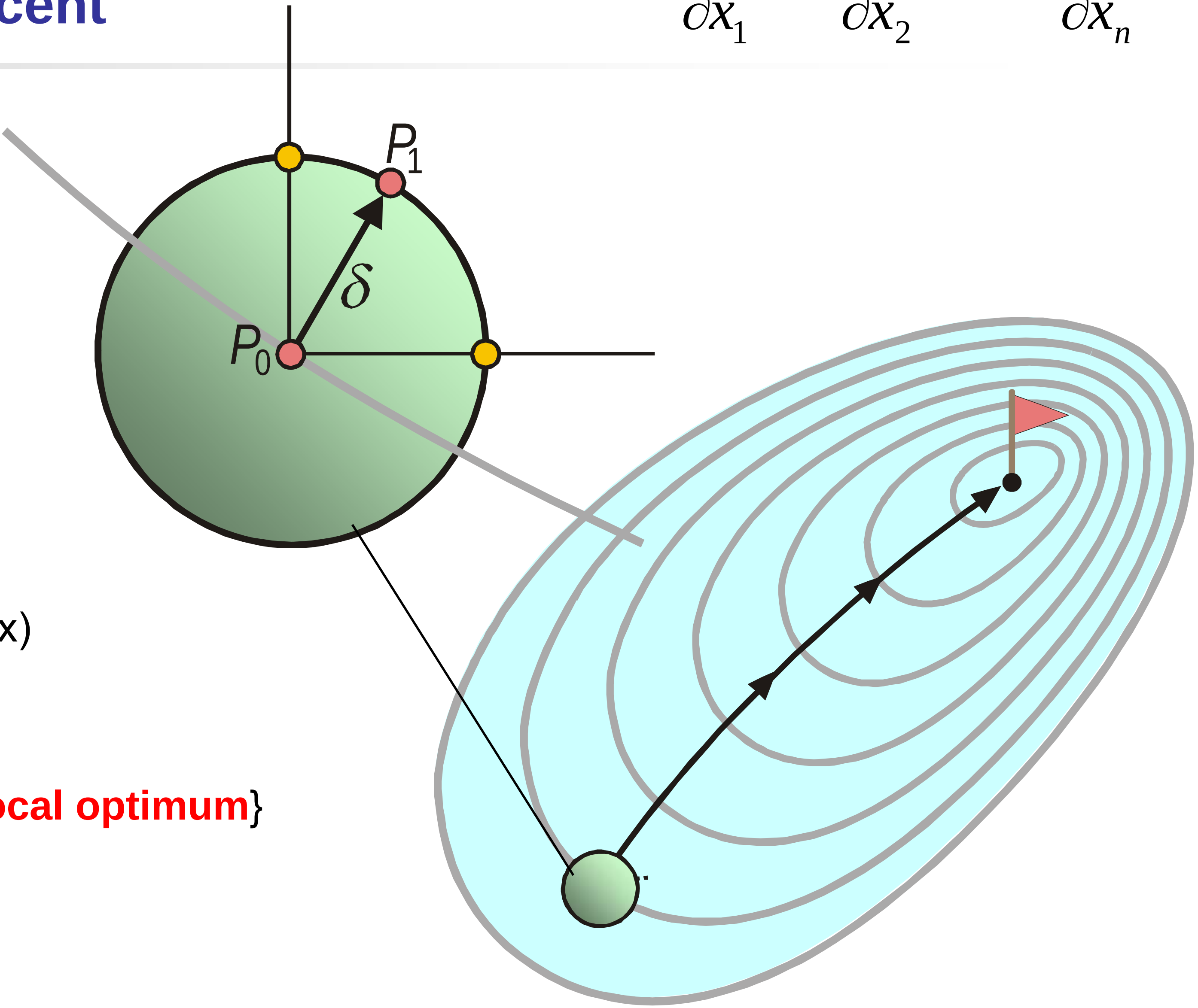
Optimization Methods

- Local Search
  - Deterministic
    - **Hill Climbing**
    - **Gradient Descent**
    - Tabu Search
  - Probabilistic
    - Simulated Annealing
    - **Iterated local Search**

- Global Search
  - Deterministic
    - Linear Optimization ✓
    - Branch&Bound, Divide&Conquer ✓
    - Dynamic Programming
  - Probabilistic
    - Genetic algorithms
    - Evolution Strategies

Gradient Descent

$$gradient\ f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n}\right)$$



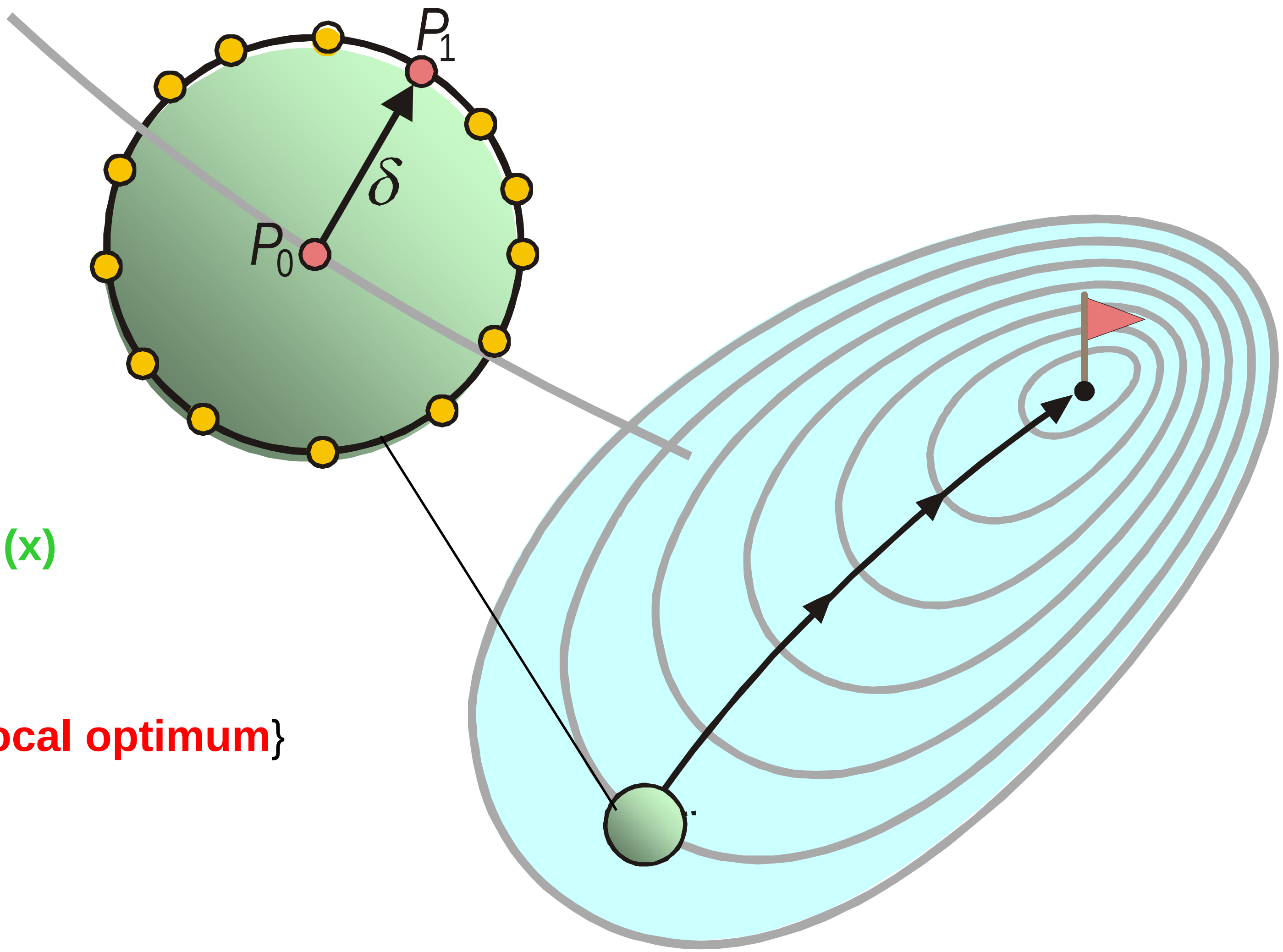
- **Initialization**

  X := random solution

- **Improvement Loop**

```
X´ := X +   δ* gradient f (x)
If f(X´) < f(X)
    Then      X:=X´
    Else      Stop     {local optimum}
```

Hill Climbing



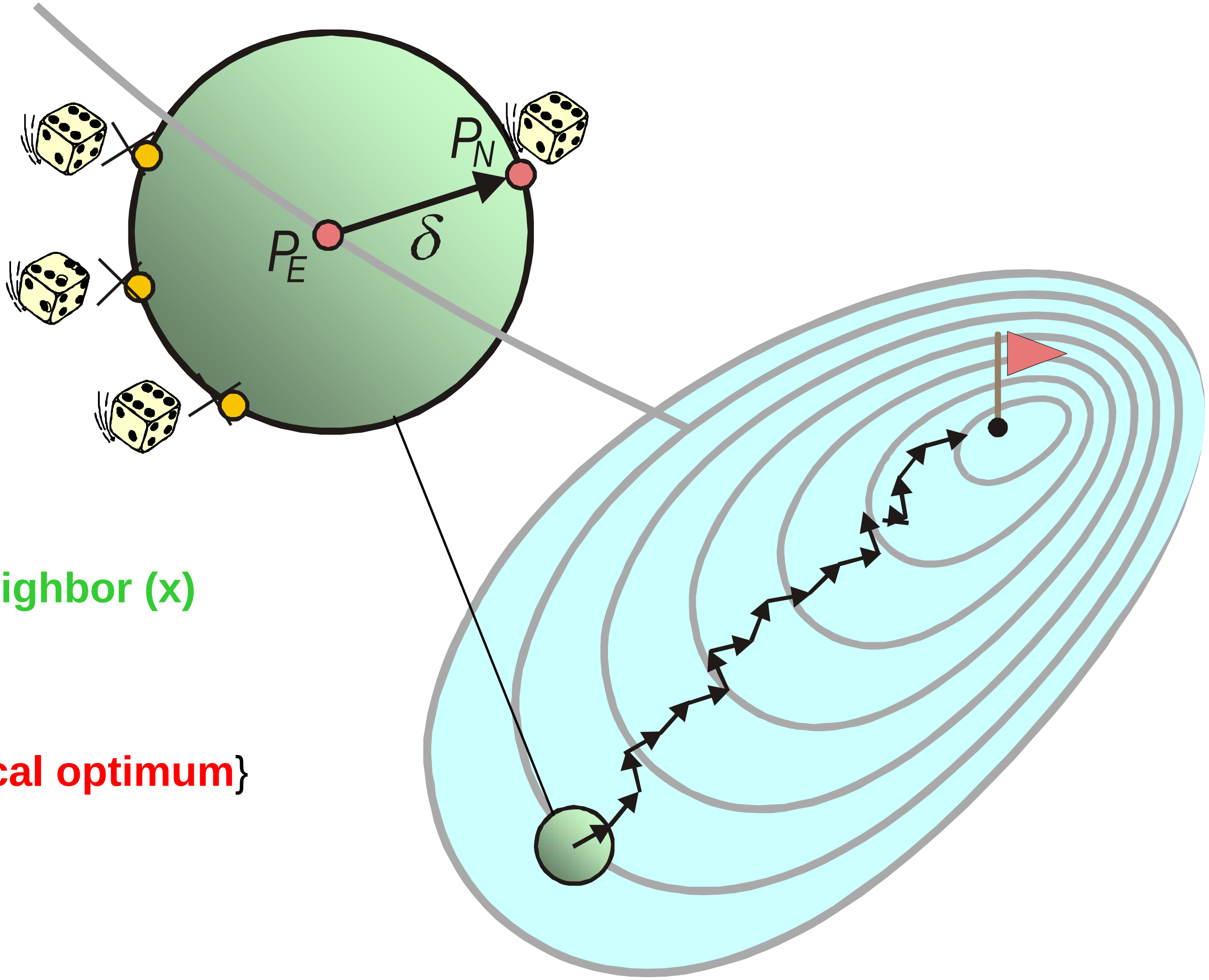
- **Initialization**

```
X := random solution
```

- **Improvement Loop**

```
X´ := Best of Neighbor (x)
If f(X´) < f(X)
    Then    X:=X´
    Else    Stop    {local optimum}
```

Probabilistic Hill Climbing



- **Initialization**

```
X := random solution
```

- **Improvement Loop**

```
X´ := Best of random Neighbor (x)
If f(X´) < f(X)
    Then    X:=X´
    Else    Stop    {local optimum}
```

Iterated Local Search

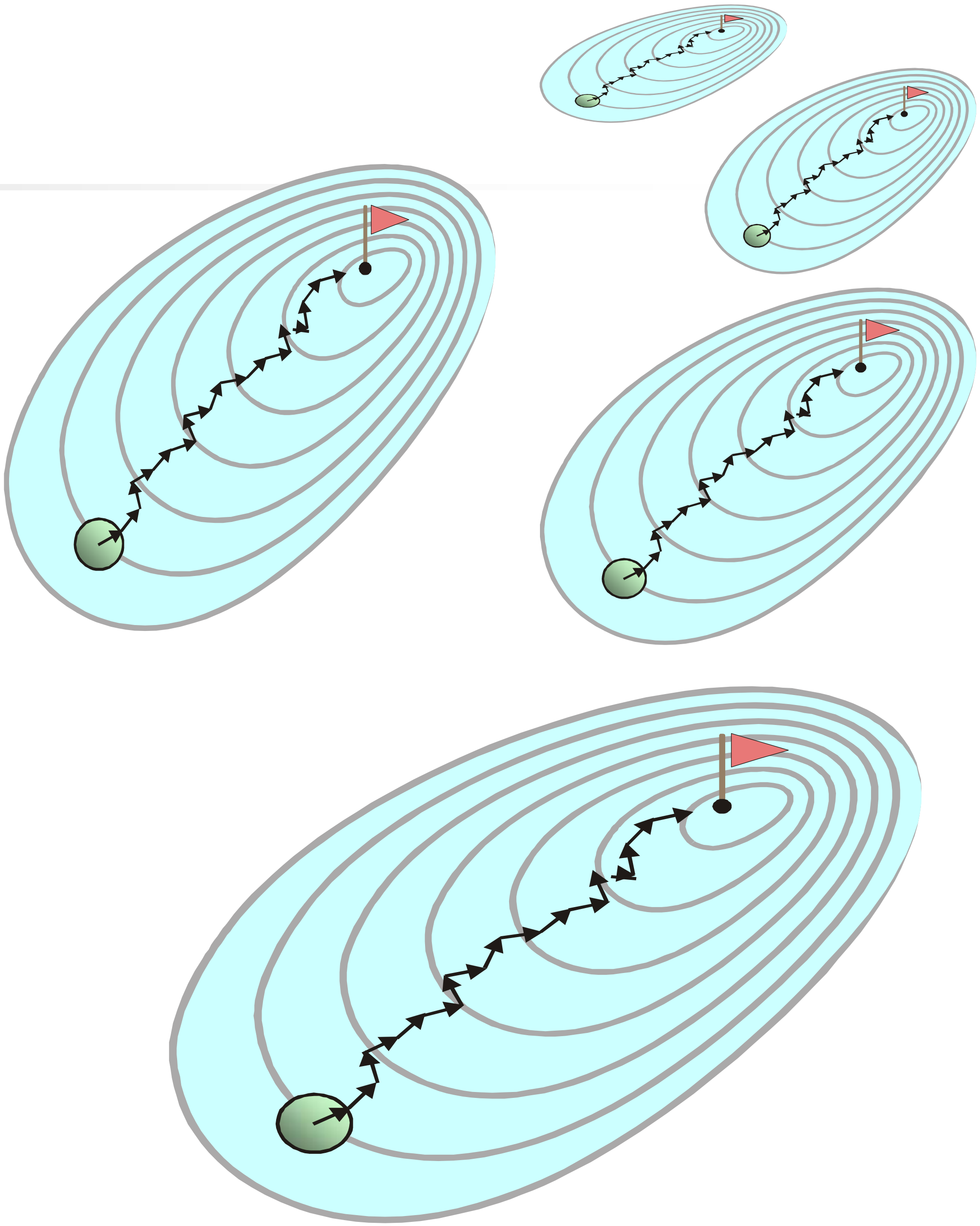
- **Iterate until time out:**

- **Initialization**

```
X := random solution
```

- **Improvement Loop**

```
X´ := Best of random Neighbor (x)
If f(X´) < f(X)
    Then    X:=X´
    Else    Stop    {local optimum}
```



Evolution

- **Iterate until time out:**

- **Initialization Population P**

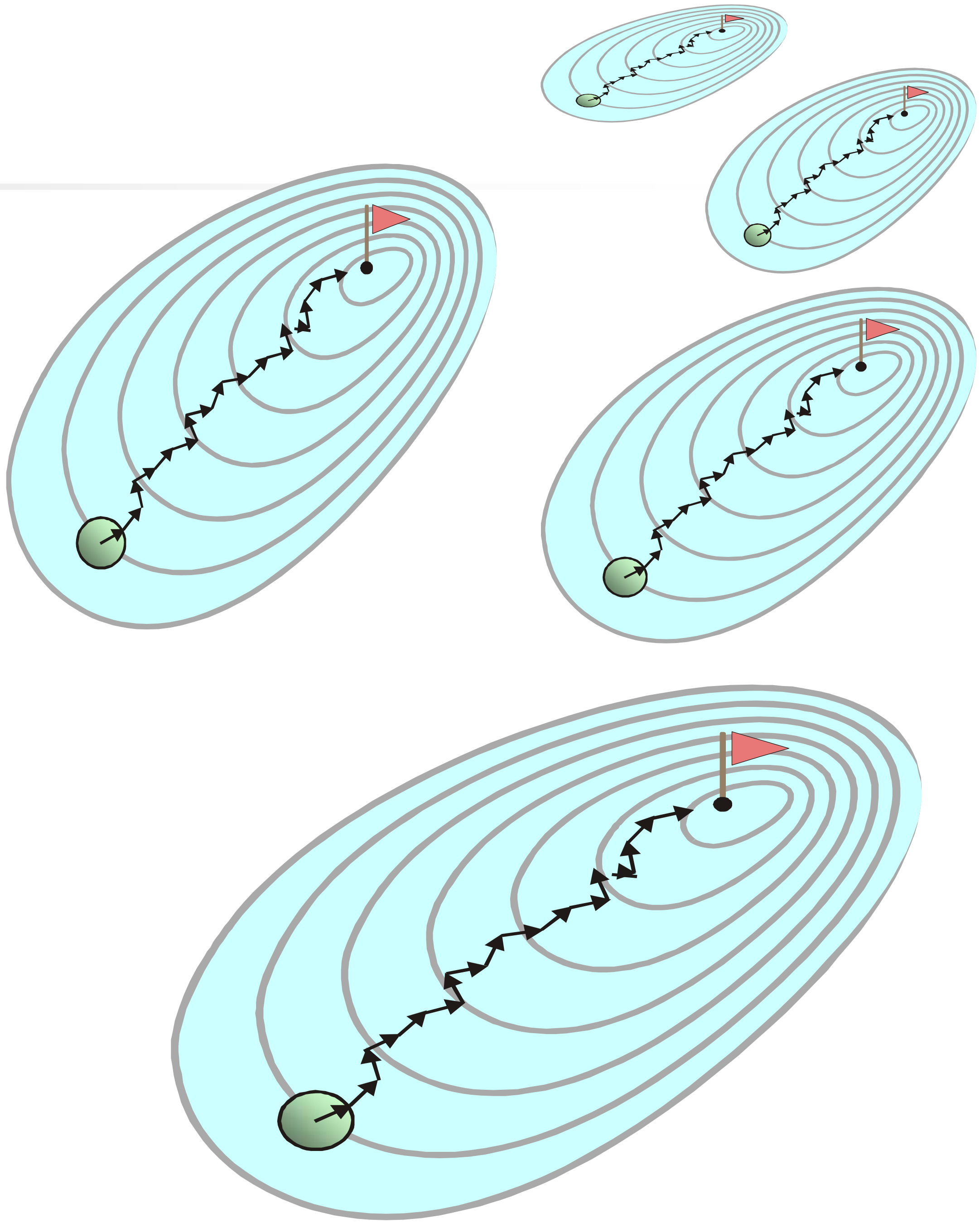
  P := random solutions

- **Improvement Loop**

  Generate **offsprings** (P)

  P := **select** **best** of offsprings (P)

  „survival of the fittest“



Optimization Methods

- Local Search
  - Deterministic
    - Hill Climbing ✓
    - Gradient Descent ✓
    - Tabu Search
  - Probabilistic
    - Simulated Annealing
    - Iterated local Search ✓

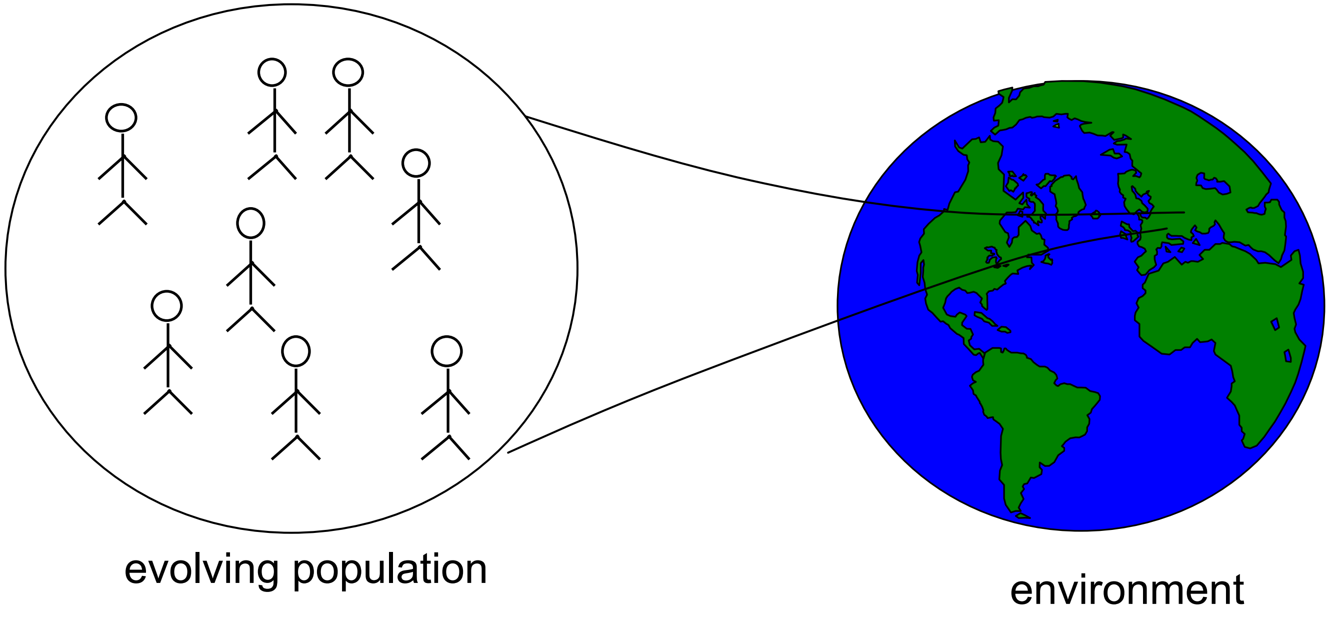
- Global Search
  - Deterministic
    - Linear Optimization ✓
    - Branch&Bound, Divide&Conquer ✓
    - Dynamic Programming
  - Probabilistic
    - **Genetic algorithms**
    - **Evolution Strategies**

Inspiration from nature

Darwin’s prinziple of natural evolution:

survival of the fittest

in populations of individuals (plants, animals), the better the individual is adapted to the environment, the higher its chance for survival and reproduction.



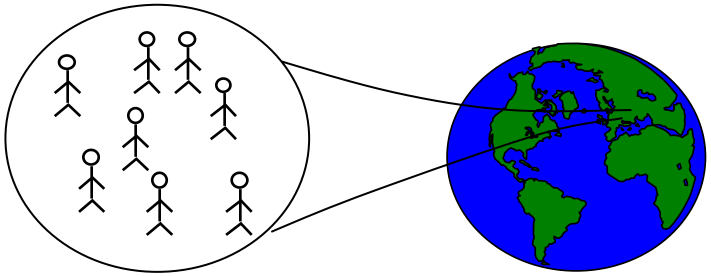
Analogies

Natural evolution

- individual
- environment
- fitness/how well adapted
- survival of the fittest
- mutation
- crossover

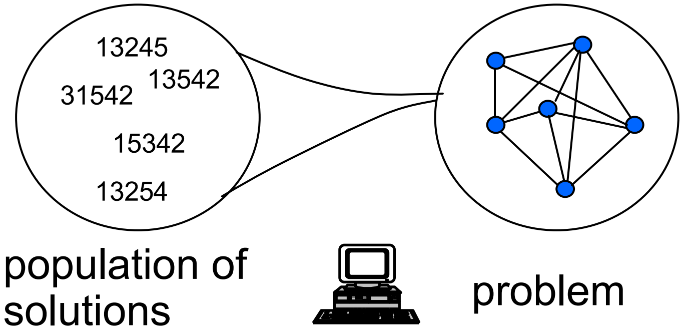
Evolutionary algorithms

- potential solution
- problem
- cost/quality of solution
- good solutions are kept
- small, random perturbations
- recombination of partial solutions



population of individuals

environment



population of solutions

problem

How are new individuals created?

- by **sexual reproduction**, i.e.
  - two individuals are selected
    (randomly, or actively through competition etc.,...),
  - a new individual is created based on the two parent‘s genetic material (recombination/crossover)
- by **mutation**, i.e. random change of
  - specific genes
  - the structure of chromosomes

Important principles in evolution

- Exploration: Increase diversity by
  - sexual reproduction (recombination/crossover)
  - mutation
- Exploitation: Reduce diversity by
  - selecting good parents
  - survival of the fittest

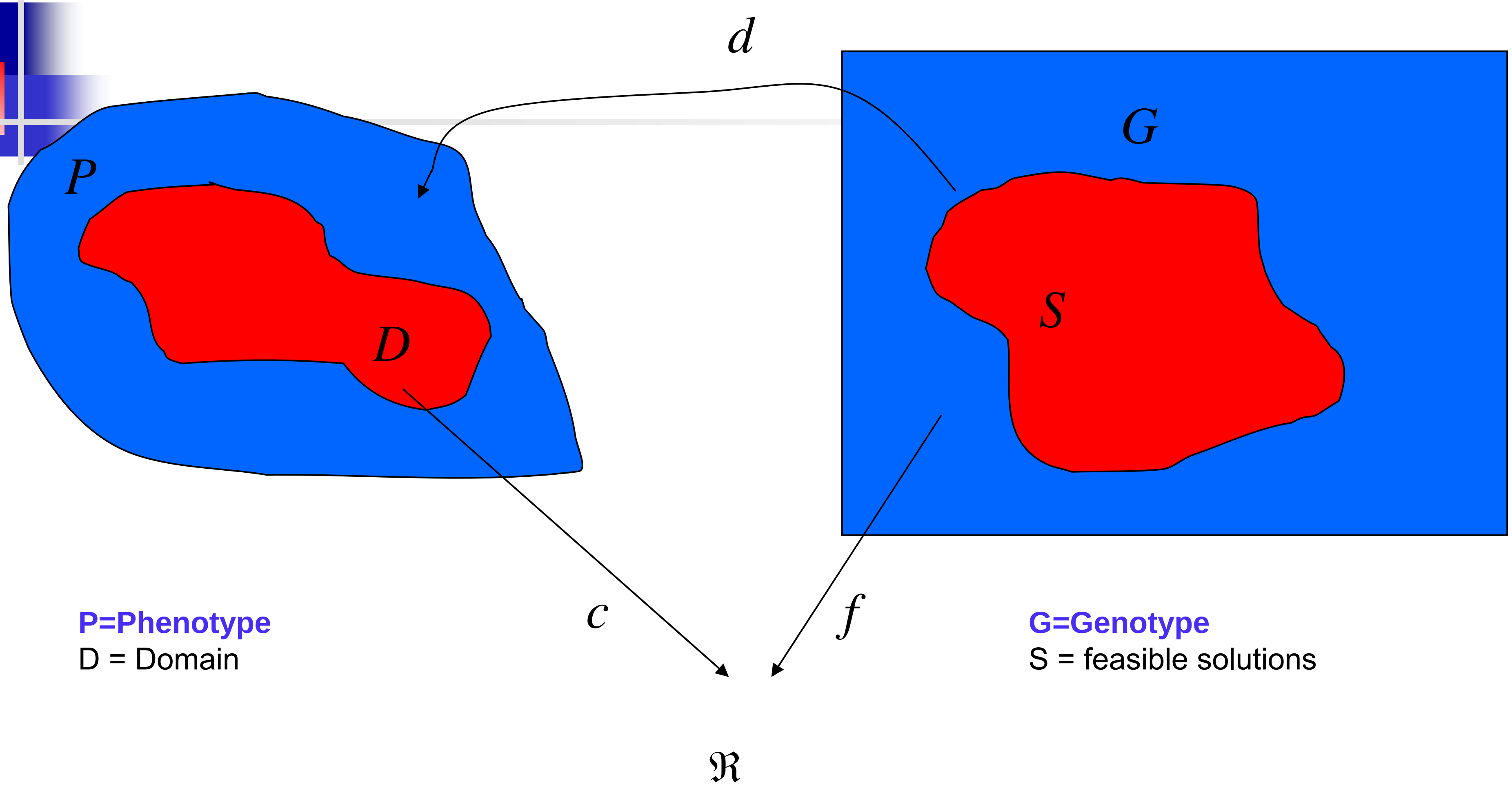
Major design decisions

- **representation**
- **fitness function**

- mutation operator
- crossover and mutation probabilities
- selection operator
- reproduction scheme
- crossover operator / recombination
- population size
- stopping criterion

Standard Representation

- Bit String
  - $x = b_1b_2\ldots b_n$ with $b_i \in \{0,1\}$
  - Similar to genetic representations
  - Difference: Chromosoms use an alphabet with 4 letters
- Real-valued Vector
  - $x = x_1x_2\ldots x_n$ with $x_i \in |R$
  - Favorable for engineering applications
- Universal Reprentations
  - Digital = Bit string
  - Analog = Real-valued Vector
  - Confer: MP3- Encoding



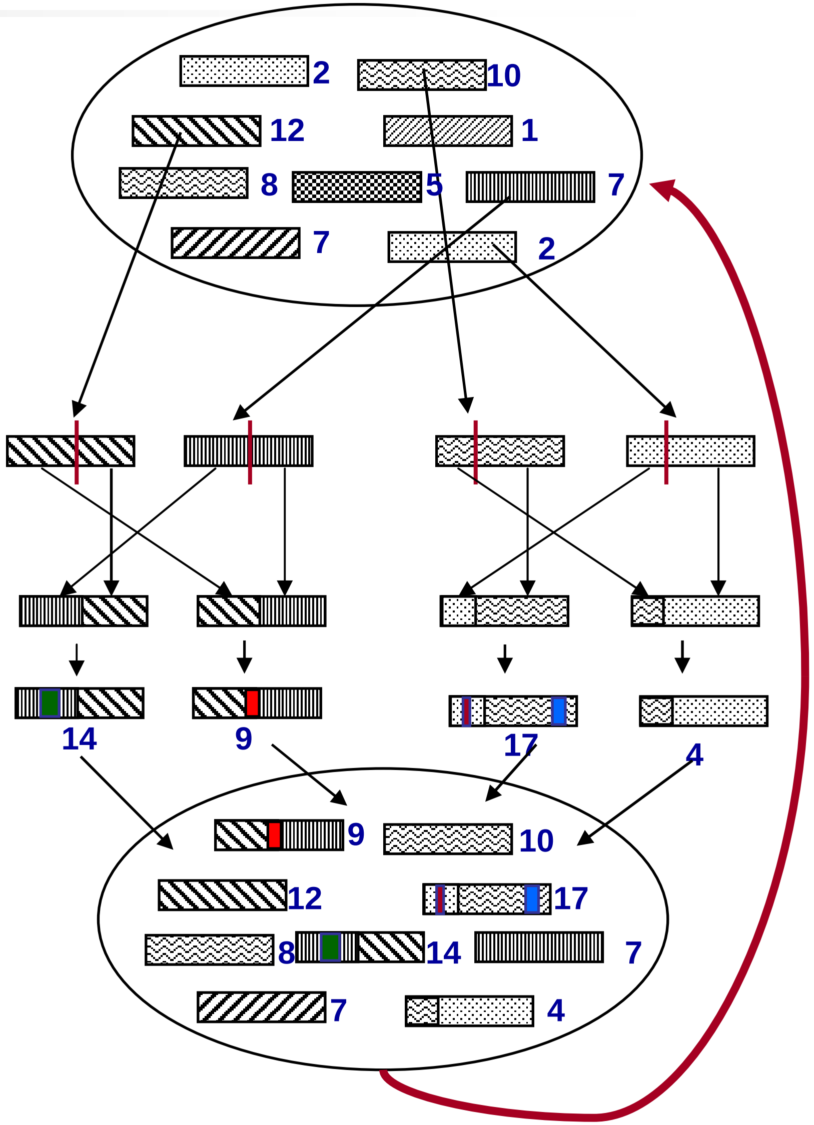
P=Phenotype
D = Domain

G=Genotype
S = feasible solutions

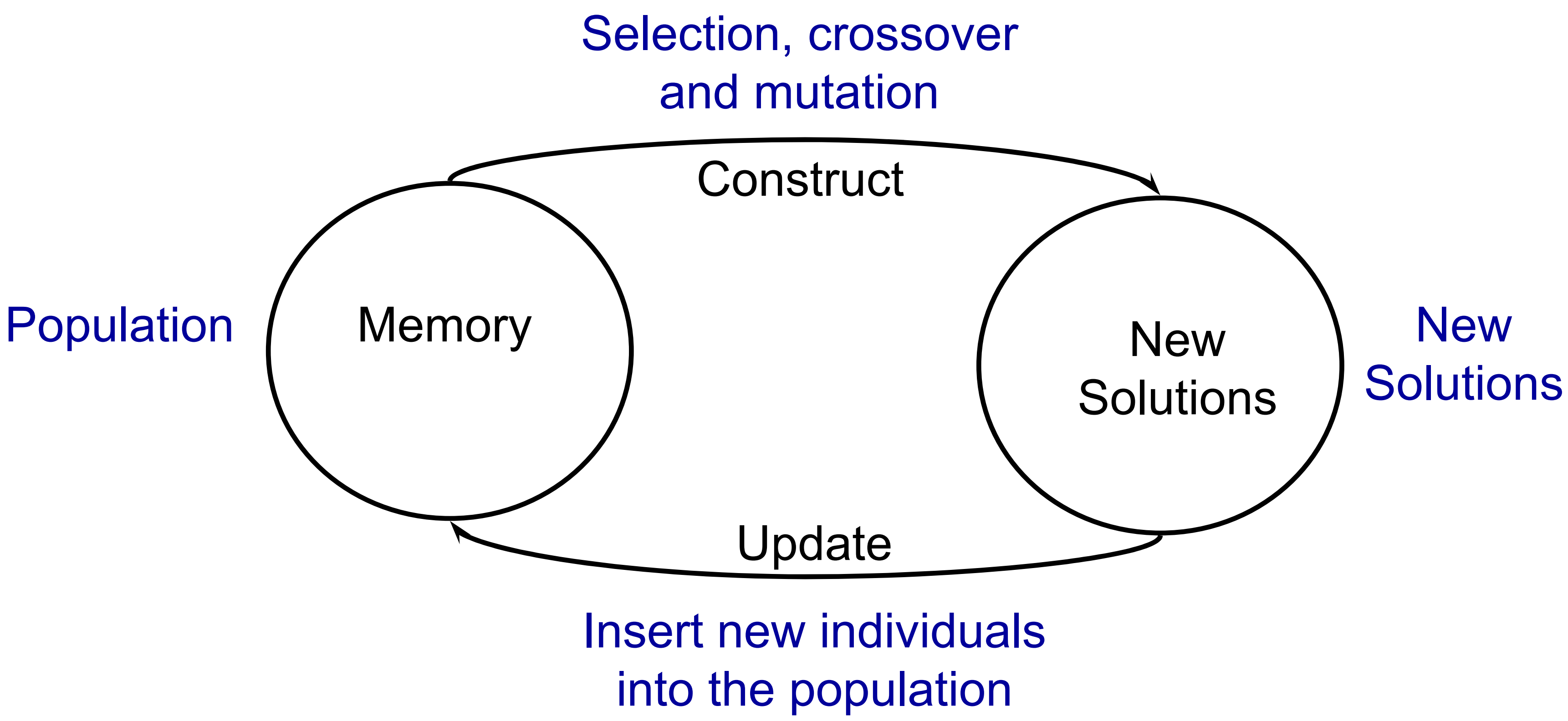
$$S = d^{-1}(D) \text{ und } f = c \circ d = fitness$$

Evolutionary Algorithm

```
INITIALIZE population
        (set of solutions)

  EVALUATE Individuals
        according to goal ("fitness")

  REPEAT

    SELECT parents

    RECOMBINE parents (CROSSOVER)

    MUTATE offspring

    EVALUATE offspring

   FORM next population

  UNTIL   termination-condition
```



Unified Model



Mutation - one parent

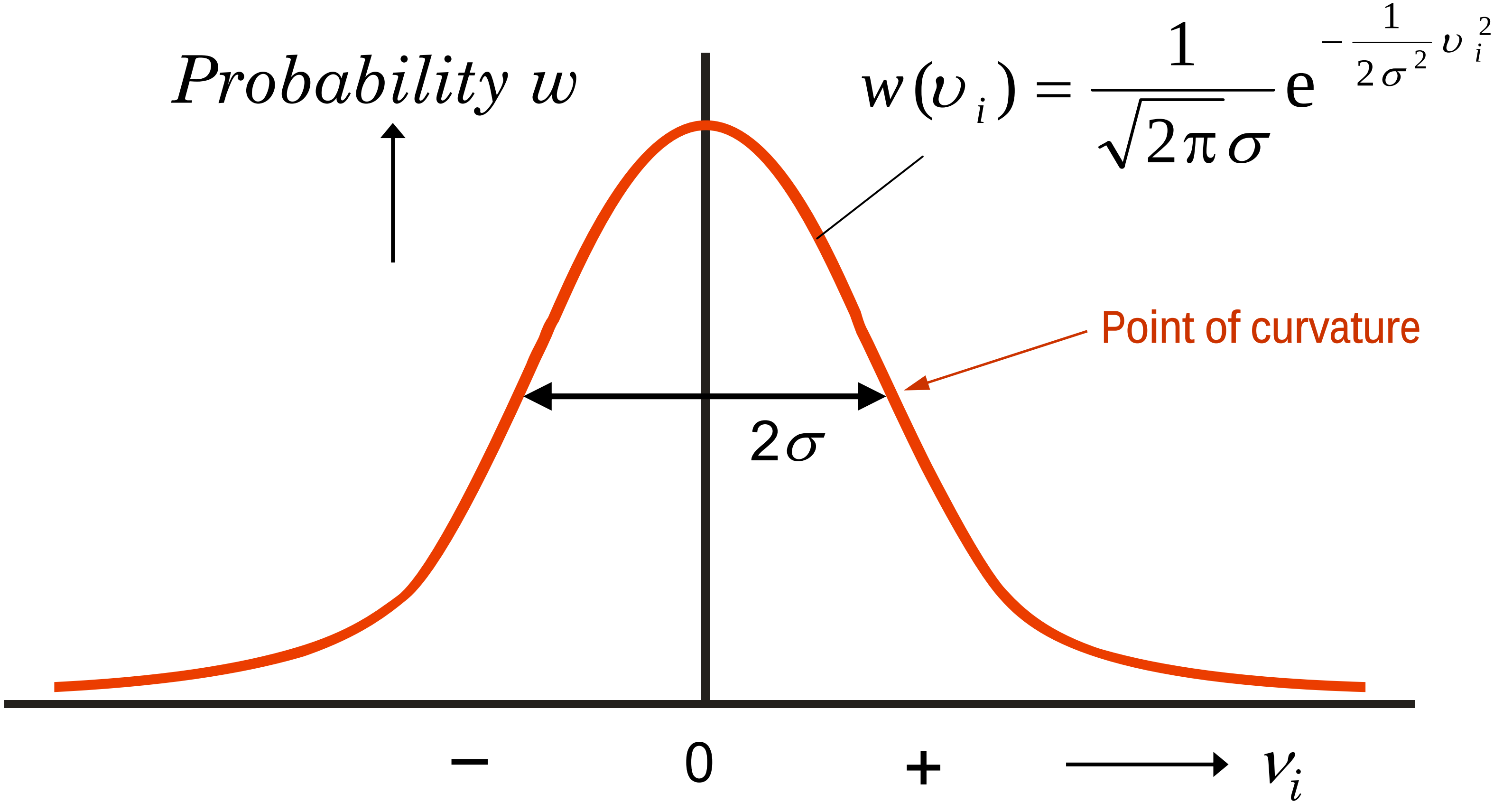
- Standard mutation operators
  - Bit string (e.g. x = $b_1 b_2 \ldots b_n$ with $b_i \in \{0,1\}$)
    flip each bit with probability p_m,
    i.e.

$$y_i = \begin{cases} 1 - x_i & \text{with probability } p_m \\ x_i & \text{otherwise} \end{cases}$$

  - Real-valued vector (e.g. x = $x_1 x_2 \ldots x_n$ with $x_i \in$ |R)
  - Input: $\vec{x}$
  - Output: $\vec{y}$ with

$$y_i = \begin{cases} x_i + v_i, \; v_i \in N(0, \sigma^2) & \text{with probability } p_m \\ x_i & \text{otherwise} \end{cases}$$

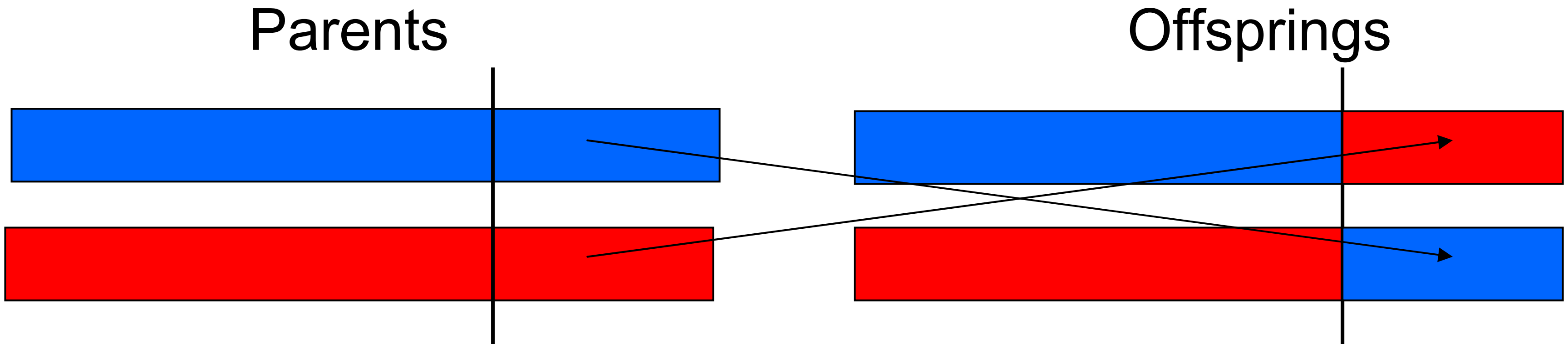
- Difficulty: how to select mutation probability p_m and mutation step size σ?



Gaussian random number v_i for mutation of variable x_i

Crossover - Two Parents

Exchange genes of the genotypes



Variants of evolutionary algorithms

- Genetic algorithms (Holland 1965 and Goldberg 1989)
  - binary representations
  - main focus on crossover, mutation only with minor role
- Evolution strategies (Rechenberg and Schwefel 1965)
  - real-valued representation
  - mutation as primary operator
  - self-adaptive mutation

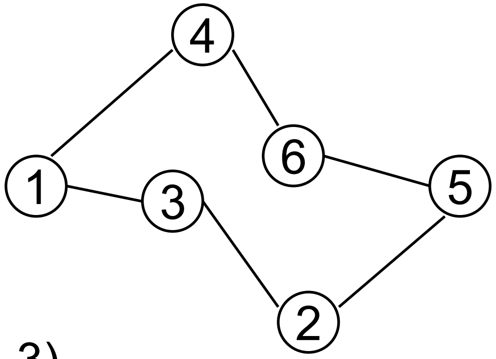
Example: Traveling Salesperson Problem (TSP)

- Given: a set of cities $C = \{c_1,..c_N\}$ and a distance function $d(c_1,c_2)$ that defines the distance for all possible paths from city c_i to city c_j.

- Goal: find a permutation of cities π which minimizes the

  total tour length

$$\sum_{i=1}^{N-1} d(c_{\pi(i)},c_{\pi(i+1)}) + d(c_{\pi(N)},c_{\pi(1)})$$



- Representation
  - Digital: Permutation, e.g. (1 – 4 – 6 – 5 – 2 – 3)
  - Analog: Preferences, e.g. (~~1.~~8, ~~0.~~3, ~~0.~~2, ~~1.~~2, ~~0.~~5, ~~0.~~7)

Special mutation for Traveling Salesman Problem

- Permutation Representation
  - $\pi(1)\ \pi(2)\ \pi(3)\ \ldots\ \pi(n)$ $\pi(i)$ = city visited in position i
  - e.g. 1-4-6-5-2-3

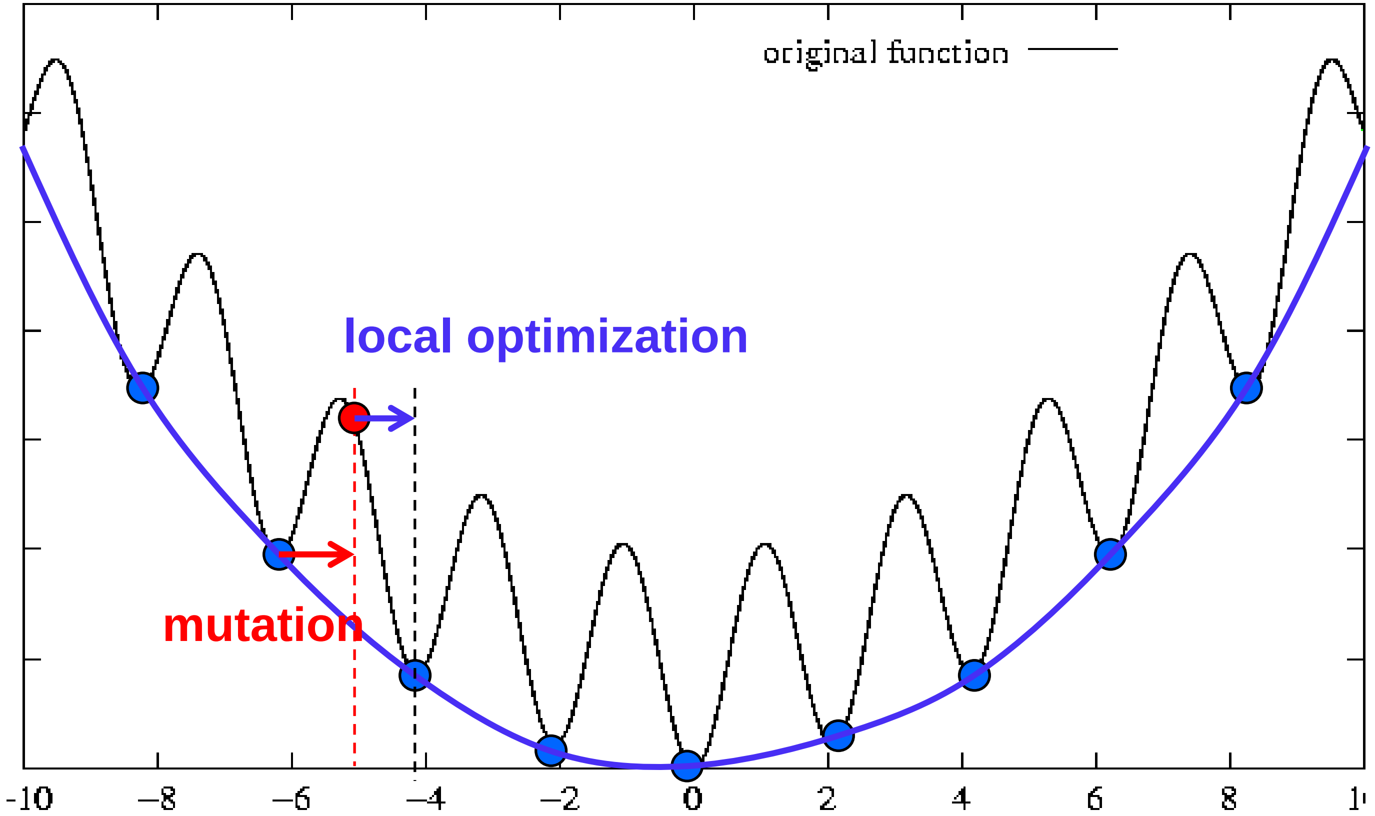
- Mutation Operators
  - Swap
    - exchange two cities
    - e.g. 1-**2**-6-5-**4**-3
  - Insert
    - remove one city and insert it at another position
    - e.g. 1-6-5-2-**4**-3
  - Inversion
    - select a random subtour and inverse the order of these cities
    - e.g. 1-4-**2-5-6**-3

Neighborhood Idea for mutation

- Focus mutation on promising changes
  - Exchange similar partners (neighbors)
  - No disruptive changes
- Example
  - Traveling Salesman Problem
  - Magic Square
- Mutation Operators
  - Swap (Magic Square)
    - exchange **two neighboring numbers**
    - e.g. 1-**2**-6-5-**4**-3
  - Insert (TSP)
    - remove one city and insert it at another position **of a neighboring city**
    - e.g. 1-6-5-2-**4**-3
  - Inversion (TSP)
    - select a random subtour **with neighboring ends** and inverse the order of these cities
    - e.g. 1-4-**2-5-6**-3

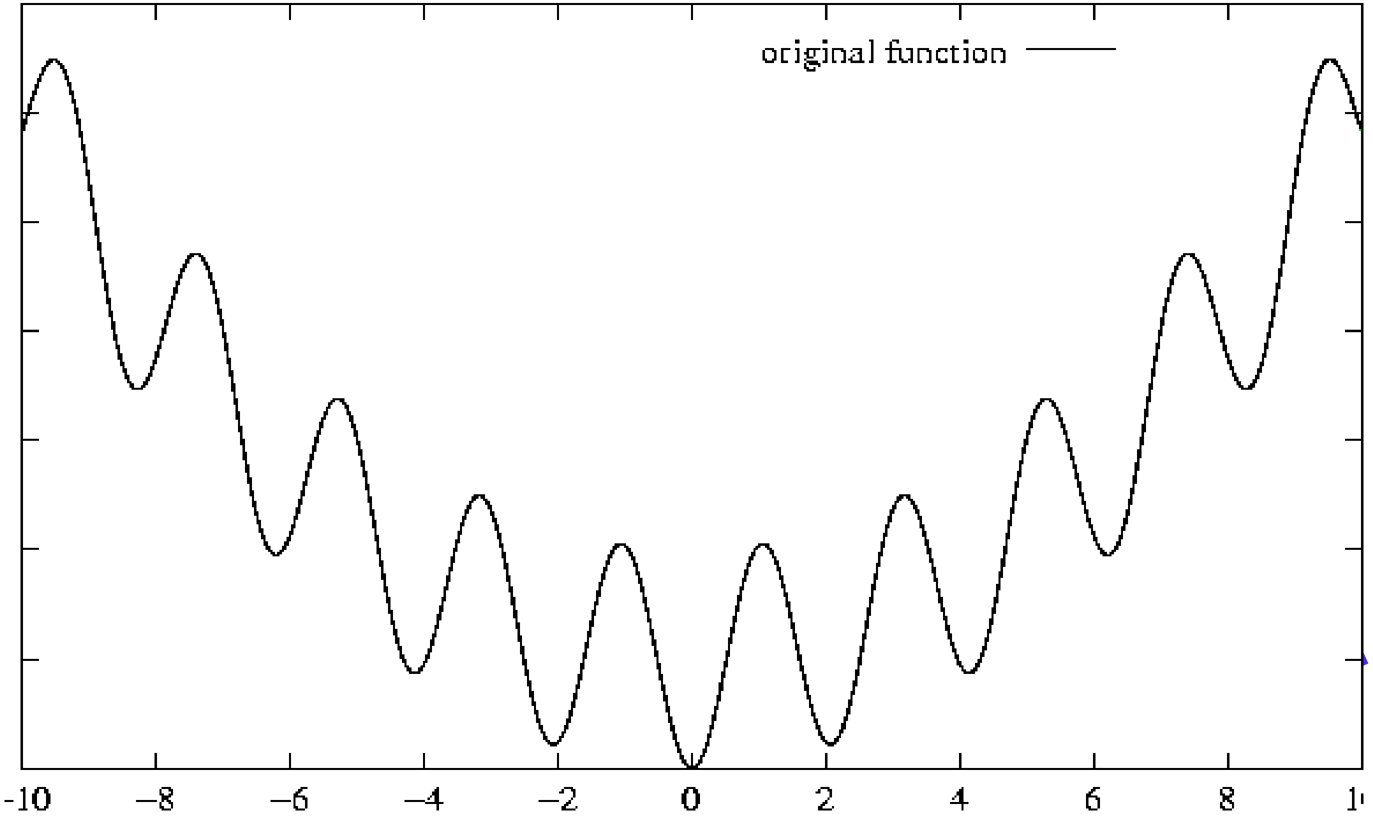
Local optimization of each solution generated

- Domain Reduction: Search on local optimas
- Required: fast local optimizer
- removes “obvious” flaws from the offsprings
- Effect on the fitness landscape: smoothing



Local optimization of each solution generated

- Domain Reduction: Search on local optimas
- Required: fast local optimizer
- removes “obvious” flaws from the offsprings
- Effect on the fitness landscape: smoothing
- **Cf. Hillclimbing in the Swiss alps or Spanish Pyrenees**



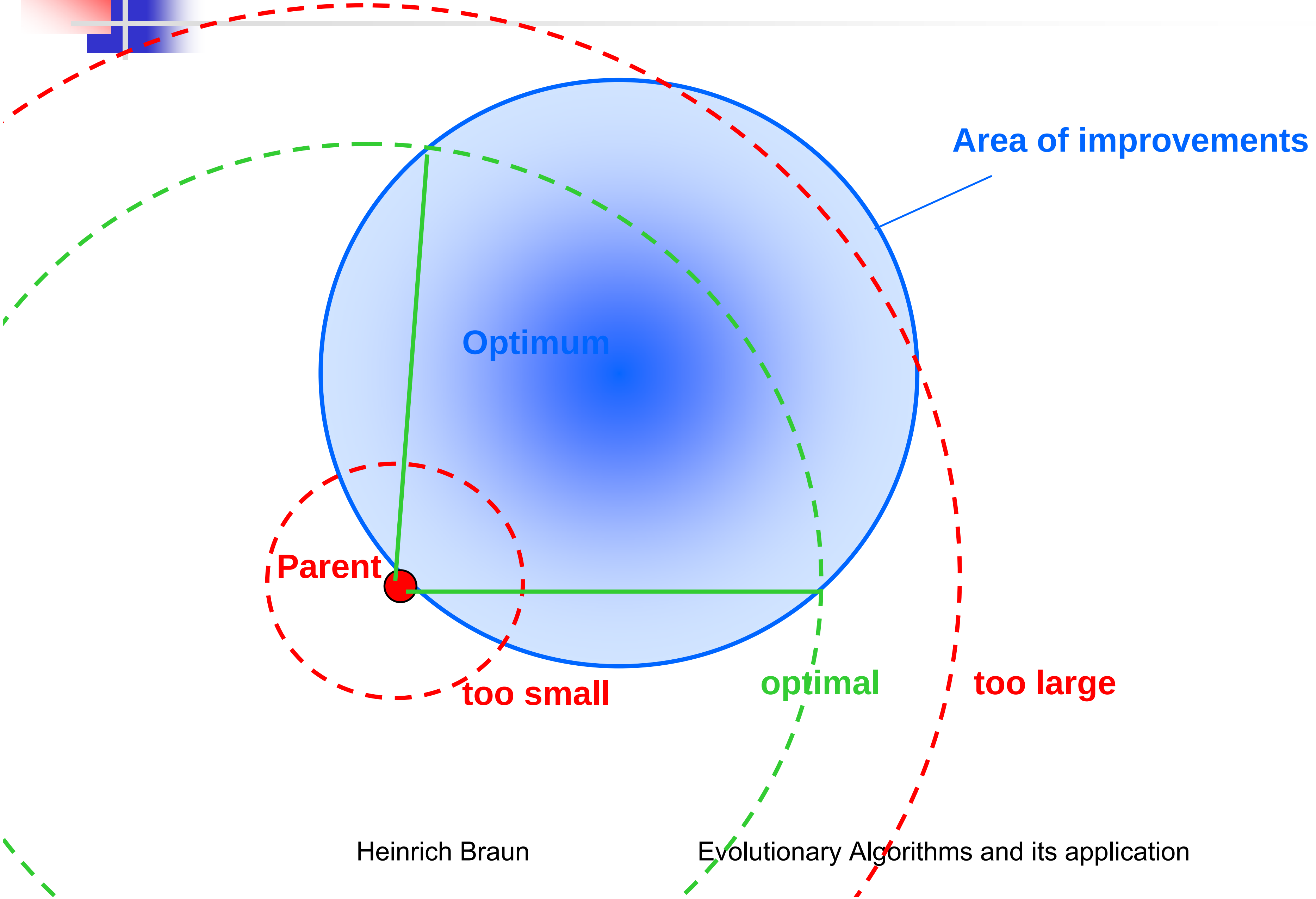
Mutation probability and step size

- For some simple problems
  - mutation rate of 1/n is optimal
  - n: chromosome length

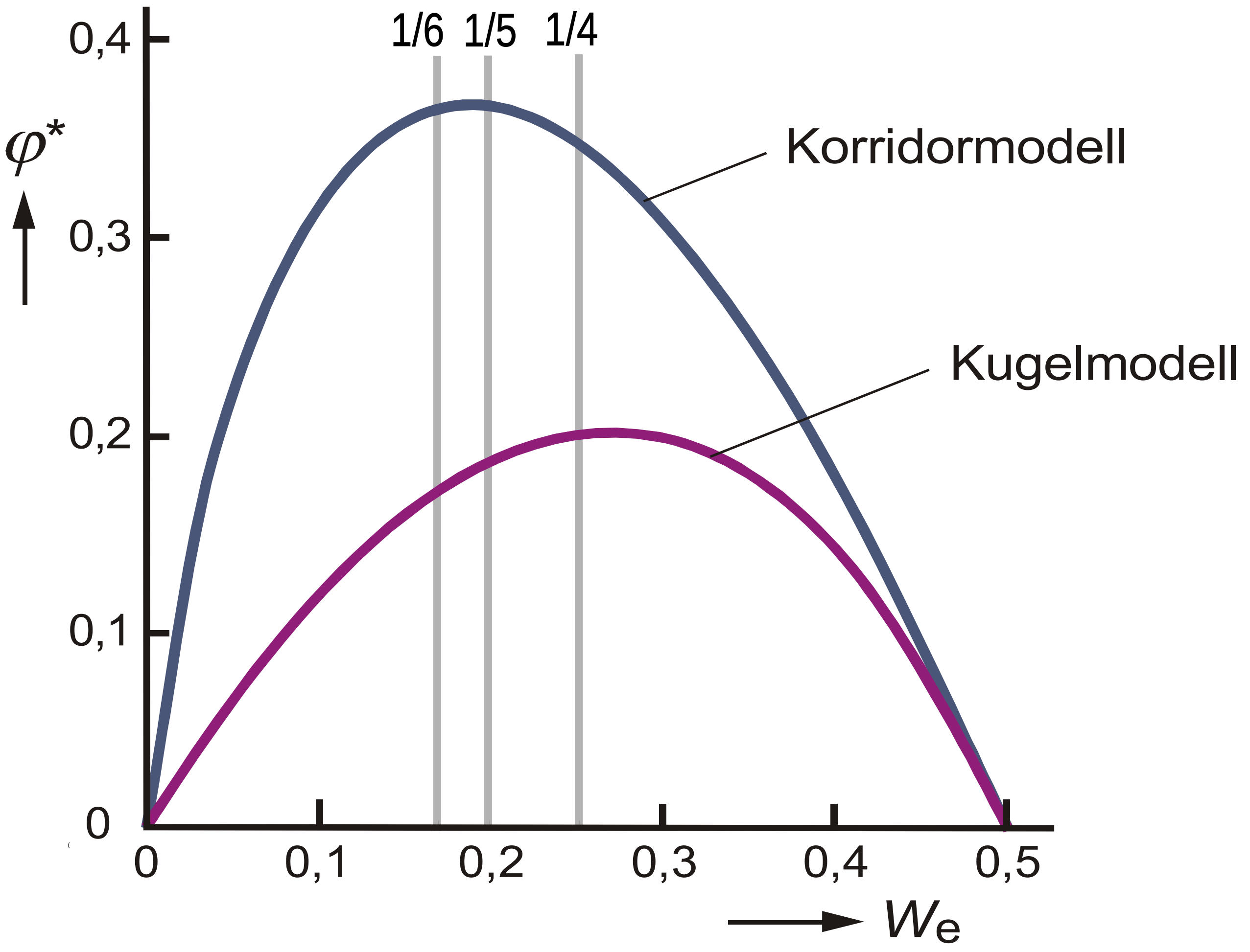
- 1/5 rule
  - 1/5 of mutations should be successful (generate a superior solution)
  - **increase** step size, If rate of successful offspring **>** 1/5,
  - **decrease** step size, If rate of successful offspring **<** 1/5,
  - danger of getting stuck in a local minimum, as mutation rate is decreased there

- self-adaptive mutation
  - expand genotype by control information
  - Step size

Mutation probability and step size – 1/5-rule



Berechnungder optimalen “Schrittweite” von Rechenberg



(**1**+1)-Evolutionsstrategie: 1/5-Erfolgsregel

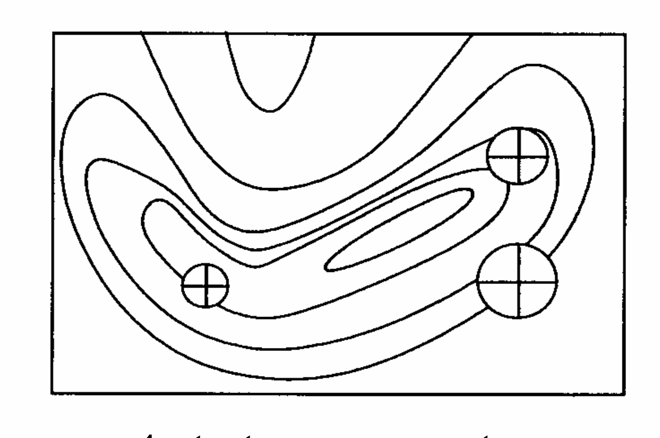
Self-adaptive mutation

- Extend genotype by **strategy parameters**
- These strategy parameters are also subject to evolution
- Simplest example: only one strategy parameter defining the mutation step size (same in every dimension)

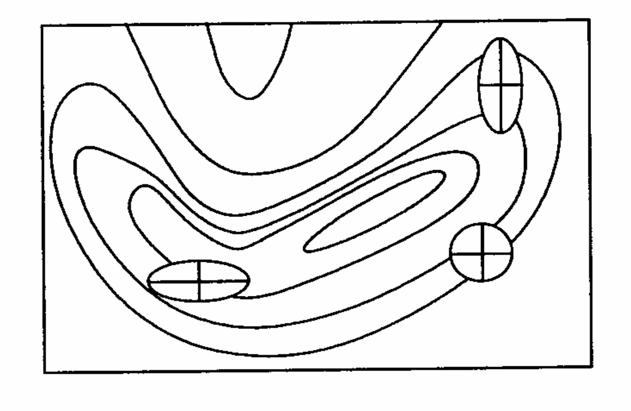
$x = (x_1, x_{2,} ..., x_n, \sigma)$ with $x_i \in \Re$

```
procedure self-adaptive mutation
Input: Individual   $x = (x_1, x_{2,} ..., x_n, \sigma_x)$
begin
    $\sigma_y \leftarrow \sigma_x \exp\left(u / \sqrt{n}\right)$ where u~N(0,1)
  for i=1 to n do
      $y_i \leftarrow x_i + N(0, \sigma_y^2)$
  end //for
  Output: Individual   $y = (y_1, y_{2,} ..., y_n, \sigma_y)$
end
```

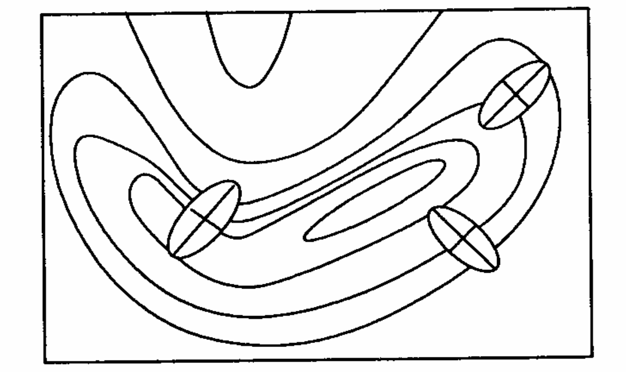
- Remark: it is possible to extend self-adaptation to allow independent mutation step sizes in every dimension. Note, however, that with an increasing number of strategy parameters, self-adaptation becomes slower and slower.



1 strategy parameter



d strategy parameters



d(d+1)/2 strategy parameters

Crossover

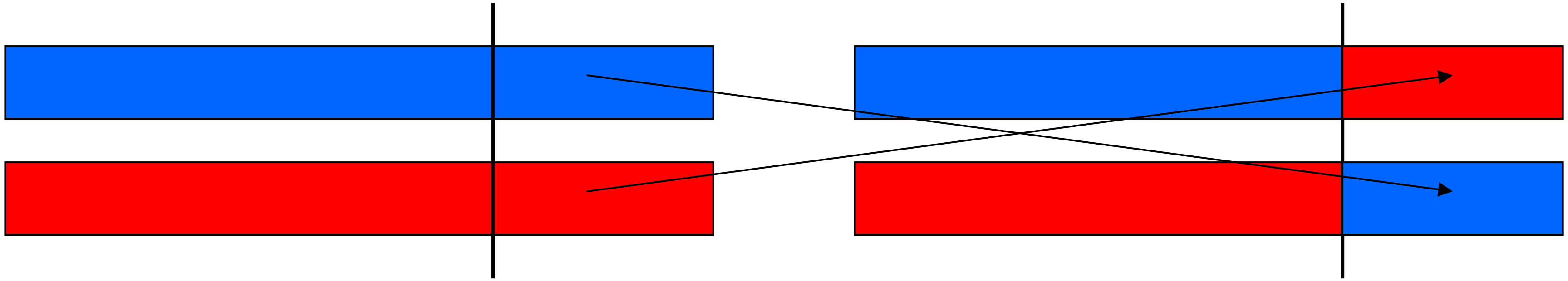
- Main idea
  - combine partial solutions of parents
  - to form new, promising solution

- Try to use as much information from the parents as possible

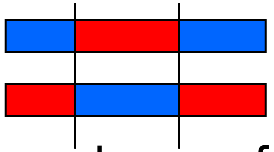
- Refinement
  - similar considerations as neighborhood move operator
  - rather problem specific

Standard crossover

- Exchange genes of the genotypes
- One point crossover:



- Assumption: closely related information should be encoded closely together on the genotype, since this reduces the probability of disruption (cf. Schema Theorem)
- Alternatives with smaller dependence on ordering of genes:
  - two point crossover



  - uniform crossover (decide for each gene from which parent it is chosen)

Special crossover for real values

Let x_i, y_i be the i-th gene of the two parent individuals x and y

- Arithmetic crossover: $z_i = \lambda_i x_i + (1 - \lambda_i) y_i, \quad \lambda_i \in [0 - \varepsilon, 1 + \varepsilon]$

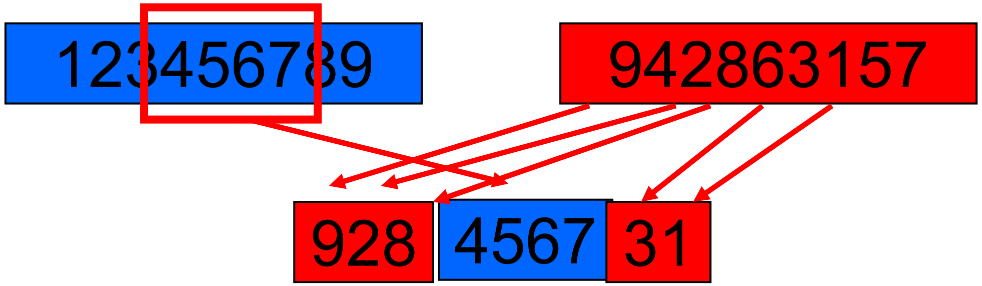
  ⇨

  - ε determines degree of extrapolation
  - if $\lambda_i = \lambda_j = \lambda \quad \forall i, j$ restricted to line between x and y

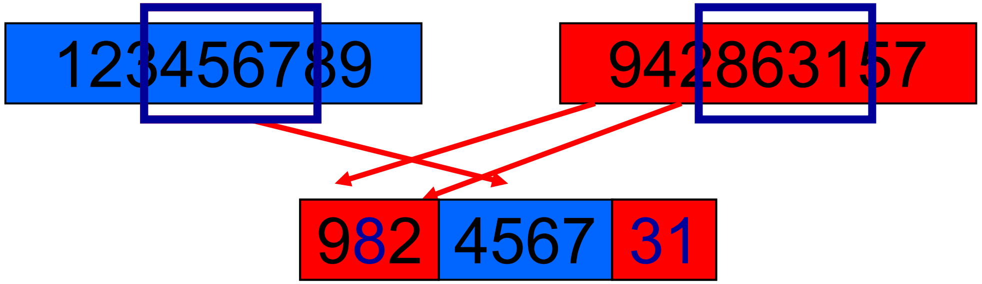
- Discrete Crossover
  - z_i = x_i or y_i
  - Random recombination of the parental genes

Special crossover for TSP

- Order crossover (OX)
  - select partial sequence from one parent, fill up in order of other parent



- Partially mapped crossover (PMX)
  - select partial sequence from one parent, fill up from other parent, resolve conflicts by mapping defined by partial sequence



move without conflict

mapping 4-8, 5-3, 7-1

- Edge recombination crossover (ERX)
  - idea: maintain as many edges from parents as possible (ERX achieves 95% usage of old edges)
  - computationally more expensive

```
procedure edge recombination crossover
$E_x$:= edges from parent x, $E_y$:= edges from parent y
$E_i$:= {e∈$E_x$∪$E_y$|e is incident to city i}
U:= {1,2,3,…,n} // list of all unvisited cities
begin
   select first city c(0) randomly from  $\{i \mid |E_i| = \min_j |E_j|\}$
   for t:=0 to n-2 do
      U ←U \{c(t)}
      if (|$E_{c(t)}$|>0)  // parental edge can be used
         select c(t+1) randomly from  $\{i \mid (|E_i| = \min_{j\in U} |E_j|) \wedge ((i,c(t)) \in E_{c(t)})\}$
      else
         select c(t+1) randomly from  $\{i \mid |E_i| = \min_{j\in U} |E_j|\}$
      end //if
      remove (c(t),i) from all edge sets $E_i$ if contained
   done
end
```

Preferring better individuals

- Necessary to advance the search
- Selection pressure
  - too high: loss of diversity, risk of getting stuck in local optimum
  - too low: no search focus, similar to random search
- Two aspects:
  - preferring better individuals when selecting the parents (usually termed selection step)
  - preferring better individuals when deciding who survives to the next generation (usually termed the reproduction scheme)

Reproduction schemes (who survives to next generation)

Evolutionsstrategien

- (μ,λ)-selection
  - from μ parents, generate λ children
  - the best μ out of the λ children forms the next parent generation
- $(\mu+\lambda)$-selection
  - from μ parents, generate λ children
  - the best μ out of the combined μ parent and λ child individuals form the next parent generation

Genetische Algorithmen

- Generational reproduction ($\cong$ (μ, μ)-selection)
  - generate n children, the children replaces the parent generation
  - usually with **elitism**, i.e. the best solution found so far survives
- Steady-state reproduction ($\cong$ $(\mu+1)$-selection)
  - in each iteration, select only two parents, generate one child
  - the child replaces the worst individual in the population.
- Hillclimbing ($\cong$ $(1+\lambda)$-selection)

Selection probabilities

Let p_i: probability of individual i to be selected as parent

f_i: fitness of individual i

- **Fitness proportional selection**

$$p_i = \frac{f_i}{\sum f_j}$$

  - most common selection scheme for early GAs
  - assumes maximization problem and positive fitness values
  - $f(x)$ und $f(x)+c$ are handled differently
  - if fitness values are all very large, basically no selection pressure
  - basically no selection pressure towards the end of the run (when all individuals are similar)
  - super-individual can take over population quickly (reduced diversity)
  - some pitfalls can be avoided by using normalized fitness values, e.g.
    - subtract minimum fitness value,
      but: influence of worst individual becomes very high

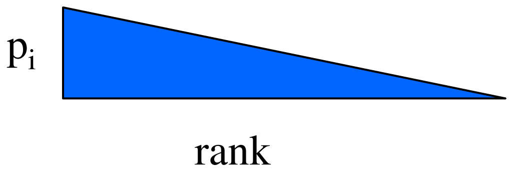
$$f'_i = \frac{f_i - f_{worst}}{f_{best} - f_{worst}}$$

Rank-based selection

Instead of considering fitness values, consider ranks

- Linear ranking selection
  - sort individuals
  - if $r_i \in [1 \ldots n]$ is the rank of individual i, select individuals according to

$$p_i = \frac{b}{n} - \left(\frac{2b-2}{n}\right)\left(\frac{r_i - 1}{n-1}\right)$$



  - constant selection pressure defined by b∈[1,2]

- Tournament selection
  - randomly choose t individuals from the population (usually t=2)
  - select the better one as parent
  - easy to implement, efficient to compute (no sorting)
  - same expected probabilities as linear ranking selection with b=2

Sampling

- Determining an individual’s selection probability, and actually choosing the parents (sampling) are two different things.
- When parents are sampled independently, variance may be high (it is possible that the worst individual is selected n times) high genetic drift

→

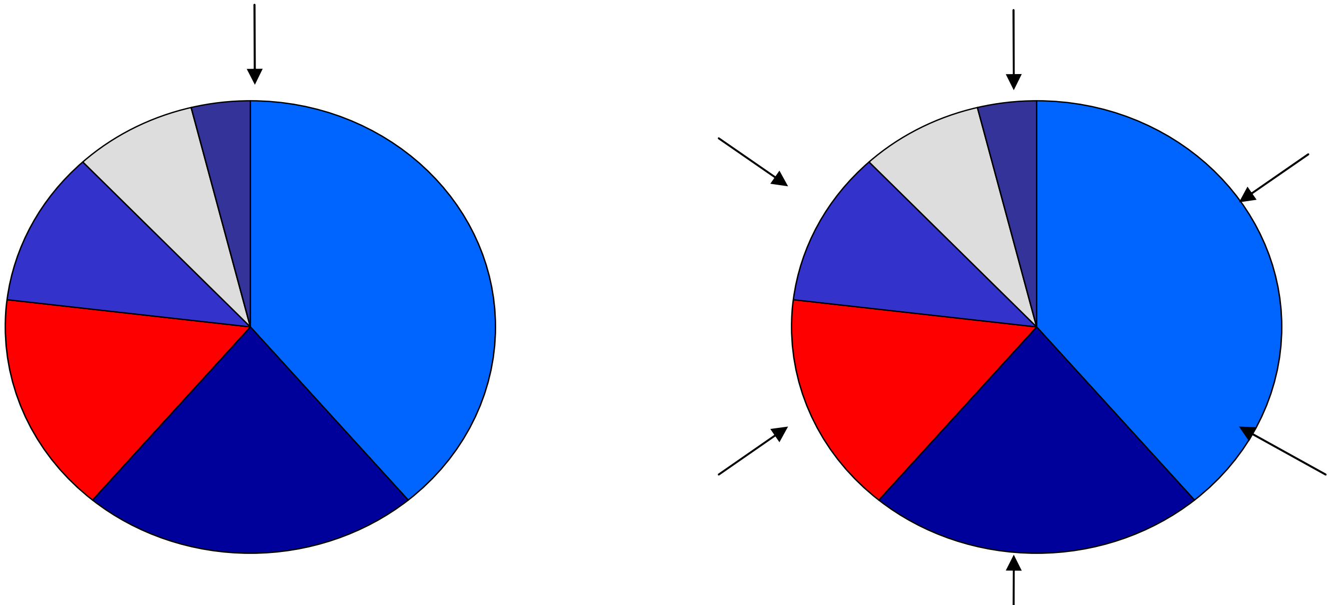
- Genetic Drift: Sampling error due to stochastic nature of selection and finite population size (decreases with increasing population size).

Ind.	f_i	p_i	a_i	$E_i=p_i*n$
1	6.0	0.39	1 0	2.34
2	3.4	0.22	1 0	1.32
3	2.5	0.16	2 0	0.96
4	1.7	0.11	1 0	0.66
5	1.2	0.08	0	0.48
6	0.6	0.04	1 6	0.24
	∑=15.4	∑=1.0	∑=6	∑=6.0

with probability $4.1*10^{-9}$

Stochastic Universal Sampling

- General idea:
  - generate one random variable $\nu\in[0,1]$,
  - select n individuals sequentially:
    for each $k\in\{0,1,..,n\}$ select the individual i, if: $\sum_{k=1}^{i} p_k \,{}_i\!\leq \nu+k/n - \lfloor \nu+k/n \rfloor < \sum_{k=1}^{i+1} p_k$



- each individual is selected at least $\lfloor np_i \rfloor$ and at most $\lceil np_i \rceil$ times
- only one random variable has to be generated to select n individuals

Population size

- If too small
  - not enough diversity in population for crossover to be useful
  - premature convergence
- If too large
  - slow convergence (in terms of fitness evaluations)
- Rule of thumb: 10 – 30

Stopping criteria

- low diversity in population (e.g. avg. fitness = best fitness)
- maximum number of iterations
- no improvement for k iterations