

# Evolutionary Algorithms

## Introduction

**Prof. Dr. Rudolf Kruse**     **Pascal Held**

`{kruse,pheld}@iws.cs.uni-magdeburg.de`

Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Institut für Wissens- und Sprachverarbeitung

# Contents of the lecture

1. Introduction
2. Metaheuristics and related optimization methods I/II
3. Encoding, Fitness, Selection
4. Variation and genetic operators
5. Metaheuristics and related optimization methods I/II
6. The Scheme Theorem
7. Genetic programming
8. Evolution strategies and Verhaltenssimulation
9. No Free Lunch, parallelization, random numbers
10. Multi Criteria optimization
11. Application Example

# Referenced Books



## Further reading I



Bäck, T. and Schwefel, H. (1993).

An overview of evolutionary algorithms for parameter optimization.

*Evolutionary Computation*, 1(1):1–23.



Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998).

*Genetic Programming — An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*.

Morgan Kaufmann Publisher, Inc. and dpunkt-Verlag, San Francisco, CA, USA and Heidelberg, Germany.



Darwin, C. (1859).

*On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*.

John Murray, London, United Kingdom.

## Further reading II



Dawkins, R. (1986).  
*The Blind Watchmaker*.  
Norton, New York, NY, USA.



Dawkins, R. (1989).  
*The Selfish Gene*.  
Oxford University Press, United Kingdom, 2nd edition.







Dawkins, R. (1990).  
*Der blinde Uhrmacher: ein neues Plädoyer für den Darwinismus*.  
Deutscher Taschenbuch-Verlag, Munich, Germany.





Dawkins, R. (1998).  
*Das egoistische Gen*.  
Rowohlt, Reinbek bei Hamburg, Germany.

## Further reading III

-  Dorigo, M. and Stützle, T. (2004).  
*Ant Colony Optimization*.  
MIT Press, Cambridge, MA, USA.
-  Gerdes, I., Klawonn, F., and Kruse, R. (2004).  
*Evolutionäre Algorithmen*.  
Vieweg, Wiesbaden, Germany.
-  Michalewicz, Z. (1996).  
*Genetic Algorithms + Data Structures = Evolution Programs*.  
Springer-Verlag, New York, NY, USA, 3rd (extended) edition.
-  Nissen, V. (1997).  
*Einführung in evolutionäre Algorithmen: Optimierung nach dem Vorbild der Evolution*.  
Vieweg, Braunschweig/Wiesbaden, Germany.

## Further reading IV

-  Vollmer, G. (1995).  
Der wissenschaftstheoretische status der evolutionstheorie. einwände und gegenargumente.  
In Vollmer, G., editor, *Biophilosophie*, page 92–106. Reclam, Stuttgart, Germany.
-  Weicker, K. (2007).  
*Evolutionäre Algorithmen*.  
Teubner Verlag, Stuttgart, Germany, 2nd edition.

# Outline

## 1. Organisational

## 2. Introduction

Optimization problems

Approach

## 3. Biological basics

## 4. Principles of evolutionary algorithms

## 5. Introduction Example: The n-Queens Problem



# Classification: Computational Intelligence

**Computational Intelligence** = **Neural Networks** (Summer Term)  
+ **Fuzzy-Systems** (Winter Term)  
+ **Evolutionary Algorithms**

Computational Intelligence can be classified by:

- approaches without specific models
- approximation instead of analytical solutions
- finding fast and acceptable solutions

# Solving optimization problems

## Definition (Optimization problem)

An *optimization problem*  $(\Omega, f, \succ)$  is given by a (search) space  $\Omega$ , an evaluation function  $f : \Omega \rightarrow \mathbb{R}$ , that assigns a quality assessment to all candidate solutions, as well as a (comparison) relation  $\succ \in \{<, >\}$ . Then, the *set of global optima*  $\mathcal{H} \subseteq \Omega$  is defined as

$$\mathcal{H} = \{x \in \Omega \mid \forall x' \in \Omega : f(x) \succeq f(x')\}.$$

- given: an optimization problem  $(\Omega, f, \succ)$
- wanted: an element  $x \in \Omega$  which optimizes the function  $f$  in the whole search space

# Fundamental approaches

## *Analytical solution:*

- efficient, but rarely applicable

## *Exhausting exploration:*

- very inefficient, so only usable in small search spaces

## *Random search:*

- always usable, but mostly inefficient

## ***Guided search:***

- Precondition: similar elements in  $\Omega$  have similar function values

# Application examples

## Examples of optimization problems I

### Parameter Optimization

- e.g. curvature of pipes (e.g. with a minimum of drag)
- generally: looking for a set of parameters which optimizes a (real-valued) function as global as possible

### Packing/Cutting Problems

- e.g. filling of a knapsack with respect to a maximum value
- wrapping of goods with a minimum of cases (bin packing problem)

### Routing Problems

- e.g. Traveling salesman problem (e.g. drilling of holes in printed circuit boards)
- optimization of delivery routes, arrangement of printed circuit board track

# Application examples

## Examples of optimization problems II

### Allocation/Arrangement Problems

- facility allocation problem (dt: Steiner-Problem):
- positioning of distribution nodes e.g in a telephone network

### Scheduling Problems

- e.g. time schedules, working plans, sequences of operations
- even compiler optimization – Reordering of instructions

### Strategy Problems

- e.g. prisoner's dilemma and other models in game theory
- Behavior modeling of different actors in economic life

### Biological modeling

- e.g. Netspinner (describes the web building behavior of certain spiders)
- EA optimizes set of parameters, comparing with reality  $\Rightarrow$  very applicable model

# Outline

1. Organisational

2. Introduction

**3. Biological basics**

4. Principles of evolutionary algorithms

5. Introduction Example: The n-Queens Problem

# Motivation

- EA are grounded on **theory of biological evolution** [Darwin, 1859].
- recommended: [Dawkins, 1986, Dawkins, 1989] (english), [Dawkins, 1990, Dawkins, 1998] (german)
- fundamental principle:
  - Beneficial traits resulting from random variation are favored by natural selection
  - better chances of procreation and multiply of individuals with beneficial traits– „*differential reproduction*“
- Evolution theory explains diversity and complexity of species
- allows unification of all different disciplines in biology

# Principles of organismic evolution I

according to [Vollmer, 1995]

## Diversity

- all forms of life (even of the same species) **differ** from each other
- even different genetic material  $\Rightarrow$  *diversity of species*
- currently existing life forms = tiny fraction of all theoretically possible ones

## Variation

- *new variants* are continuously created by mutation and genetic recombination (sexual reproduction)

## Inheritance

- variations are *heritable*, as long as entering the germ line
- are genetically passed to the next generation
- gen. no inheritance of acquired traits (*Lamarckisms*)



# Principles of organismic evolution II

## Speciation

- *genetically diverge* of individuals and populations
- ⇒ new *species* (no crossbreeding of the members)
- charact. branching structure of phylogenet. „pedigree“

## Birth surplus/Overproduction, nearly all life forms:

- *more offspring* that can ever become mature enough to procreate themselves

## Adaptation/Natural Selection/Differential Reproduction

- on average: hereditary variations of the survivors of a population
- ⇒ *increases adaptation* to the local environment
- Herbert Spencers Slogan “survival of the fittest” is misleading
  - rather: „different fitness ⇒ different reproduction“

# Principles of organismic evolution III

## Randomness/Blind Variation

- Variations are *triggered/initiated/caused by random*
- no concentration on certain traits/beneficial adaptations
- *non teleological*, from the Greek:  $\tau\epsilon\lambda\omicron\varsigma$  — goal, purpose

## Gradualism

- Variations happen in comparatively *small steps* (as measured by the complete information content(entropy) or the complexity of an organism)
- ⇒ phylogenetic changes = *gradual* and relatively slow  
(In contrast: saltationism — large changes in development)

## Evolution / Transmutation / Inheritance with Modification

- Adaptation to environment ⇒ species *evolve* in the course of time
- theory of evolution opposes creationism  
(claim: immutability of the species)

# Principles of organismic evolution IV

## Discrete Genetic Units

- Store/Transfer/Change of genetic information in discrete units
- no continuously blend of hereditary traits
- otherwise: *Jenkins nightmare* through recombination (complete disappearance of any differences in a population)

## Opportunism

- processes of evolution work exclusively on what is present
- better/optimal solutions are not found if intermediary stages(are necessary for solutions) exhibit certain fitness handicaps

## Evolution-strategic Principles

- not only organisms are optimized, but also the mechanisms of evolution: reproduction and mortality rates, life spans, vulnerability to mutations, mutation step sizes, etc.

# Principles of organismic evolution V

## Ecological Niches

- competitive species can tolerate each other if they occupy different ecological niches (“biospheres“)
- biological diversity of species is possible in spite of competition and natural selection

## Irreversibility

- course of evolution is irreversible and unrepeatable

## Unpredictability

- course of evolution is neither determined, nor programmed  $\Rightarrow$  not predictable

## Increasing Complexity

- biological evolution has led to increasingly more complex systems
- open problem: how can we actually measure the complexity of life forms?

# Outline

## 1. Organisational

## 2. Introduction

## 3. Biological basics

## 4. Principles of evolutionary algorithms

Fundamental terms

Ingredients

Formal definitions

## 5. Introduction Example: The n-Queens Problem

# Fundamental terms and meaning I

notion	biology	computer science
individual	living organism	solution candidate
chromosome	DNA-histone-protein-strand	sequence of comp. objects
	describes „construction plan“ or (some of the traits) of an individual in encoded form	
	usually multiple chromosomes per individual	usually only one chromosome per individual
gene	part of a chromosome	computational object
	is the fundamental unit of inheritance which determines a (partial) characteristic of an individual	
allele (allelomorph)	form or „value“ of gene	value of comp. object
	in each chromosome at most one form/value of a gene	
locus	position of a gene	position of comp. object
	at each position in chromosome exactly one gene	

## Fundamental terms and meaning II

<b>notion</b>	<b>biology</b>	<b>computer science</b>
phenotype	physical appearance of a living organism	implementation of a solution candidate
genotype	genetic constitution of a living organism	encoding of a solution candidate
population	set of living organism	bag/multiset of chromosomes
generation	population at a point in time	
reproduction	creating offspring of one or multiple (usually two) (parent) organisms	creating (child) chromosomes from one or multiple (parent) chromosomes
fitness	aptitude/conformity of a living organism	aptitude/quality of a solution candidate
	determines chances of survival and reproduction	

# Ingredients of an evolutionary algorithm I

**Encoding** for the solution candidates

- highly problem-specific
- no general rules
- later: discussion of aspects that attention should be paid to when choosing an encoding

A method to create an **initial population**

- commonly created by simple generation of random sequences
- depending on the chosen encoding: more complex methods needed

**Evaluation function** (fitness function) to evaluate the individuals

- represents environment and assess quality of individuals
- often: identical to the function to optimize
- may also contain additional elements (e.g. constraints)



# Ingredients of an evolutionary algorithm II

**Selection method** on the basis of the fitness function

- chooses parental individuals to create offspring
- selects individuals transferred to the next generation without change

A set of **genetic operators** to modify chromosomes

- *Mutation* — randomly changes of individual genes
- *Crossover* — recombination of chromosomes
  - better: “crossing over” (meiosis-process, cell division phase)
  - chromosomes are dissipated and assembled cross-over

**Various parameters** (population size, mutation probability, etc.)

**Termination criterion**

- user-specified number of generations have been created

## Ingredients of an evolutionary algorithm III

- no improvement (of the best solution candidate) for a user-specified number of generations
- user-specified minimum solution quality has been obtained

# Formal definition: Decoding function

according to [Weicker, 2007]

- for every optimization problem: different representations of solution candidates
- EA separates space  $\Omega$  (so called phenotype) from representation of the solution candidate in individual (so called genotype  $\mathcal{G}$ )
- Fitness function  $f$  is defined on  $\Omega$
- Mutation und Recombination is defined on  $\mathcal{G}$
- for evaluation: transformation of genotype represented individual in  $\Omega$

## Definition (Decoding function)

A decoding function  $\text{dec} : \mathcal{G} \rightarrow \Omega$  is a transformation of a genotype  $\mathcal{G}$  to the phenotype  $\Omega$ .

## Formal definitions: individual

An individual contains in general:

1. *genotype*  $A.G \in \mathcal{G}$  of an individual  $A$
2. *additional information* or *strategy parameters*  $A.S \in \mathcal{Z}$ 
  - e.g. parameter settings for genetic operators
  - space  $\mathcal{Z}$  of all possible additional information
  - $A.S$  as well as  $A.G$  are modifiable by operators
3. *quality* or *fitness*  $A.F \in \mathbb{R}$

### Definition (individual)

An *individual*  $A$  is a tuple  $(A.G, A.S, A.F)$  containing the solution candidate (genotype  $A.G \in \mathcal{G}$ ), the optional additional information  $A.S \in \mathcal{Z}$  and the quality assessment  $A.F = f(\text{dec}(A.G)) \in \mathbb{R}$ .

## Formal definitions: genetic operators

- $\Xi$  („Xi“): set of all states of the random number generator
- no definition of the change of the actual state  $\xi \in \Xi$

### Definition (genetic operators)

A *mutation operator* (which is applied on a  $\mathcal{G}$ -encoded optimization problem and  $\mathcal{Z}$ ) is defined by the mapping

$$\text{Mut}^\xi : \mathcal{G} \times \mathcal{Z} \rightarrow \mathcal{G} \times \mathcal{Z}.$$

Analogously, a *recombination operator* with  $r \geq 2$  parents and  $s \geq 1$  offspring ( $r, s \in \mathbb{N}$ ) is defined by the mapping

$$\text{Rek}^\xi : (\mathcal{G} \times \mathcal{Z})^r \rightarrow (\mathcal{G} \times \mathcal{Z})^s.$$

## Formal definitions: Selection operator

- Input: population of  $r$  individuals, whereas  $s$  are chosen
- selection is changing/creating no new individuals
- selection defines indices of individuals only by fitness

### Definition (Selection operator)

A *selection operator*  $\text{Sel}$  is applied on a population

$$P = \langle A^{(1)}, \dots, A^{(r)} \rangle:$$

$$\text{Sel}^\xi : (\mathcal{G} \times \mathcal{Z} \times \mathbb{R})^r \rightarrow (\mathcal{G} \times \mathcal{Z} \times \mathbb{R})^r$$

$$\langle A^{(i)} \rangle_{1 \leq i \leq r} \mapsto \langle A^{(\text{IS}^\xi(c_1, \dots, c_r)_k)} \rangle_{1 \leq k \leq s} \quad \text{mit } A^{(i)} = (a_i, b_i, c_i).$$

The underlying index-selektion has the shape

$$\text{IS}^\xi : \mathbb{R}^r \rightarrow \{1, \dots, r\}^s.$$

## Simple example for a selection operator

- Parental population consists of individuals  $A^{(1)}, A^{(2)}, \dots, A^{(5)}$
- related quality assessments of the individuals are given by
  1.  $A^{(1)}.F = 2.5$
  2.  $A^{(2)}.F = 1.9$
  3.  $A^{(3)}.F = 3.7$
  4.  $A^{(4)}.F = 4.1$
  5.  $A^{(5)}.F = 2.4$
- selection chooses with  $IS^\xi : \mathbb{R}^5 \rightarrow \{1, \dots, 5\}^3$  indices 4, 3 and 1 respectively individuals  $A^{(4)}, A^{(3)}$  and  $A^{(1)}$

# Fundamental Genetic Algorithm

## Definition

A *simple evolutionary algorithm* on an optimization problem  $(\Omega, f, \succ)$  is an 8-tuple  $(\mathcal{G}, \text{dec}, \text{Mut}, \text{Rek}, \text{IS}_{\text{Parents}}, \text{IS}_{\text{Environment}}, \mu, \lambda)$ . Here,  $\mu$  describes the amount of individuals of the parental population and  $\lambda$  defines the offspring per generation. In addition, it holds

$$\text{Rek} : (\mathcal{G} \times \mathcal{Z})^k \rightarrow (\mathcal{G} \times \mathcal{Z})^{k'},$$

$$\text{IS}_{\text{parents}} : \mathbb{R}^{\mu} \rightarrow (1, \dots, \mu)^{\frac{k}{k'} \cdot \lambda} \quad \text{with } \frac{k}{k'} \cdot \lambda \in \mathbb{N},$$

$$\text{IS}_{\text{Environment}} : \mathbb{R}^{\mu + \lambda} \rightarrow (1, \dots, \mu + \lambda)^{\mu}.$$



# Generic algorithm

---

## Algorithm 1 General Scheme of an Evolutionary Algorithm

---

**Input:** optimization problem  $(\Omega, f, \succ)$

$t \leftarrow 0$

$\text{pop}(t) \leftarrow$  create the initial population of size  $\mu$

evaluate  $\text{pop}(t)$

**while** not termination criterion {

$\text{pop}_1 \leftarrow$  select parents of offsprings with size  $\lambda$  from  $\text{pop}(t)$

$\text{pop}_2 \leftarrow$  create offspring by recombination of  $\text{pop}_1$

$\text{pop}_3 \leftarrow$  mutate individuals in  $\text{pop}_2$

    evaluate  $\text{pop}_3$

$t \leftarrow t + 1$

$\text{pop}(t) \leftarrow$  select  $\mu$  individuals from  $\text{pop}_3 \cup \text{pop}(t - 1)$

}

**return** best individual of  $\text{pop}(t)$

---

# Genetic vs. Evolutionary algorithm

## separation of the terms

### Genetic algorithm:

- Encoding: Sequence of ones and zeros

⇒ Chromosome is *Bitstring* (word on alphabet  $\{0, 1\}$ )

### Evolutionary algorithm:

- Encoding: problem-related  
(Sequence of letters, graphs, formulas, etc.)
- genetic operators: defined in relation to encoding and problem

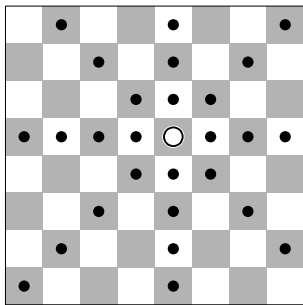
# Outline

1. Organisational
2. Introduction
3. Biological basics
4. Principles of evolutionary algorithms
- 5. Introduction Example: The n-Queens Problem**
  - Backtracking Solution of the n-Queens Problem
  - Analytical Solution
  - Solution by using EA
  - Programme

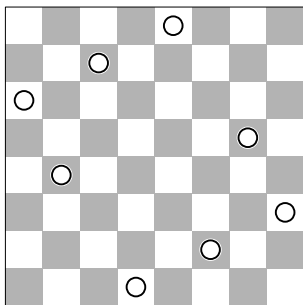
# The $n$ -Queens Problem

place  $n$  queens onto a  $n \times n$  chessboard in such a way that no rank(row), no file(column) and no diagonal contains more than one queen

or: place queens in such a way that no queen is in the way of another queen



Draw options of a queen



Solution of the  $n$ -Queen Problem

# Backtracking Solution of the n-Queens Problem

1. place queens rank-by-rank bottom-up  
(or column by column from left to right, o.ä.)
2. consider each row as follows:
  - place one queen in a rank sequentially from left to right onto the squares of the board
  - for each placement: check if queen collides with queens in lower ranks
  - if not, work on next rank recursively
  - afterwards: shift queen one square rightwards
3. return solution if queen is placed in top line without any collision

# Backtracking Solution of the n-Queens Problem

```
int search (int y)
{
    /* --- depth first search */
    int x, i, d;          /* loop variables, buffer */
    int sol = 0;         /* solution counter */

    if (y >= size) {     /* if a solution has been found, */
        show(); return 1; /* show it and abort the function */
    }
    for (x = 0; x < size; x++) { /* traverse fields of the current row */
        for (i = y; --i >= 0; ) { /* traverse the preceding rows */
            d = abs(qpos[i] -x); /* and check for collisions */
            if ((d == 0) || (d == y-i)) break;
        } /* if there is a colliding queen, */
        if (i >= 0) continue; /* skip the current field */
        qpos[y] = x; /* otherwise place the queen */
        sol += search(y+1); /* and search recursively */
    }
    return sol; /* return the number of */
} /* search() */ /* solutions found */
```

## Analytical Solution

if only *one* solution (a placement of queens) is required, calculation of positions for all  $n > 3$  is defined as:

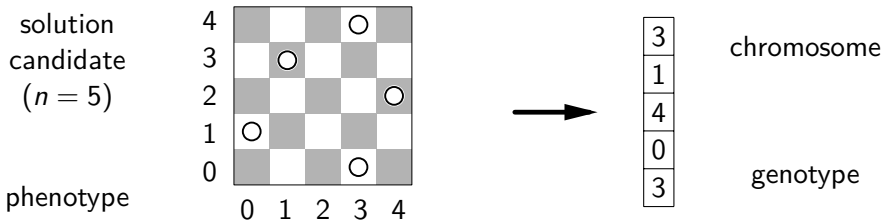
- $n \bmod 2 = 1 \Rightarrow$  place 1 queen on  $(n - 1, n - 1)$  and  $n \leftarrow n - 1$
- $n \bmod 6 \neq 2 \Rightarrow$  place queens  
in the rows  $y = 0, \dots, \frac{n}{2} - 1$  in the columns  $x = 2y + 1$ ,  
in the rows  $y = \frac{n}{2}, \dots, n - 1$  in the columns  $x = 2y - n$
- $n \bmod 6 = 2 \Rightarrow$  place queens  
in the rows  $y = 0, \dots, \frac{n}{2} - 1$  in the columns  $x = (2y + \frac{n}{2}) \bmod n$ ,  
in the rows  $y = \frac{n}{2}, \dots, n - 1$  in the columns  $x = (2y - \frac{n}{2} + 2) \bmod n$

Hence: it is not quite appropriate to approach the n-queens problem with an evolutionary algorithm

Nevertheless: good illustration of certain aspects of evolutionary algorithms on this problem

## EA: Encoding

- Representation: 1 solution candidate = 1 chromosome with  $n$  genes
- each gene: one rank of the board with  $n$  possibles alleles
- value of the gene: position of the queen in corresponding rank



- solution candidates with  $> 1$  queen each rank not permitted
- ⇒ smaller search space



## EA: data structure

- data type for 1 chromosome, which stores the fitness
- data type for 1 chromosome with buffer for „intermediary population“ and flag for the best individual

```
typedef struct {                                /* --- an individual --- */
    int fitness;                               /* fitness (number of collisions) */
    int cnt;                                   /* number of genes (number of rows) */
    int genes[1];                             /* genes (queen positions in rows) */
} IND;                                         /* (individual) */
```

```
typedef struct {                                /* --- a population --- */
    int size;                                 /* number of individuals */
    IND **inds;                              /* vector of individuals */
    IND **buf;                               /* buffer for individuals */
    IND *best;                               /* best individual */
} POP;                                       /* (population) */
```

## EA: main loop

shows basic form of an EA:

```
pop_init(pop);                /* initialize the population */
while ((pop_eval(pop) < 0)    /* while no solution found and */
&&    (--gencnt >= 0)) {    /* not all generations computed */
    pop_select(pop, tmsize, elitist);
    pop_cross (pop, frac);    /* select individuals, */
    pop_mutate(pop, prob);    /* do crossover, and */
}                             /* mutate individuals */
```

parameters:

gencnt	maximum amount of remaining generations
tmsize	size of tournament selection
elitist	indicates, if best individual will always be taken
frac	fraction of individuals, which will be submitted by cross-over
prob	mutation probability

## EA: Initialize

create random series of  $n$  numbers from  $\{0, 1, \dots, n - 1\}$

```
void ind_init (IND *ind)
{
    /* --- initialize an individual */
    int i;          /* loop variable */

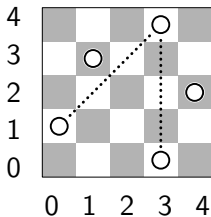
    for (i = ind->n; --i >= 0; ) /* initialize the genes randomly */
        ind->genes[i] = (int)(ind->n *drand());
    ind->fitness = 1;          /* fitness is not known yet */
} /* ind_init() */
```

```
void pop_init (POP *pop)
{
    /* --- initialize a population */
    int i;          /* loop variable */

    for (i = pop->size; --i >= 0; )
        ind_init(pop->inds[i]); /* initialize all individuals */
} /* pop_init() */
```

## EA: Evaluation

- fitness: negated number of columns and diagonals with  $\geq 1$  queen (negated number due to maximizing fitness)



2 collisions  $\rightarrow$  fitness = -2

- if queens in 1 column/diagonal  $\geq 2$ : count each pair (easier to implement)
- fitness-function results immediately in termination criterion: Solution has (highest possible) fitness 0
- also: termination is guaranteed when maximal generation is reached

## EA: Evaluation

count collisions by computation on chromosomes:

```
int ind_eval (IND *ind)
{
    int i, k;
    int d;
    int n;

    if (ind->fitness <= 0)
        return ind->fitness;
    for (n = 0, i = ind->n; --i > 0; ) {
        for (k = i; --k >= 0; ) {
            d = abs(ind->genes[i] - ind->genes[k]);
            if ((d == 0) || (d == i-k)) n++;
        }
    }
    return ind->fitness = -n;
} /* ind_eval() */
```

## EA: Evaluation

- calculation of the fitness of all individuals of the population
- simultaneously: determination of best individual
- best individual fitness 0  $\Rightarrow$  solution is found

```
int pop_eval (POP *pop)
{
    /* --- evaluate a population */
    int i; /* loop variable */
    IND *best; /* best individual */

    ind_eval(best = pop->inds[0]);
    for (i = pop->size; --i > 0; )
        if (ind_eval(pop->inds[i]) >= best->fitness)
            best = pop->inds[i]; /* find the best individual */
    pop->best = best; /* note the best individual */
    return best->fitness; /* and return its fitness */
} /* pop_eval() */
```

# EA: selection of individuals

## tournament selection:

- consider `tmsize` arbitrarily chosen individuals
- best (of these) individual „wins“ tournament and will be chosen
- the higher the fitness the better chance to get chosen

```
IND* pop_tmsel (POP *pop, int tmsize)
{
    IND *ind, *best;          /* --- tournament selection */
                              /* competing/best individual */

    best = pop->inds[(int)(pop->size *drand())];
    while (--tmsize > 0) {    /* randomly select tmsize individuals */
        ind = pop->inds[(int)(pop->size *drand())];
        if (ind->fitness > best->fitness) best = ind;
    }                        /* det. individual with best fitness */
    return best;            /* and return this individual */
} /* pop_tmsel() */
```

## EA: selection of individuals

- tournament selection for individuals of the next population generation
- perhaps best individuals will be applied (and not changed)

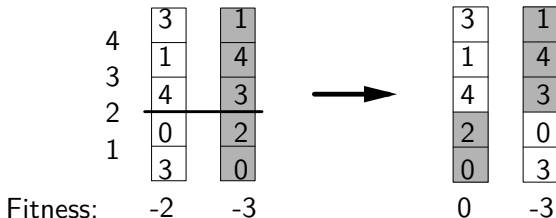
```
void pop_select (POP *pop, int tmsize, int elitist)
{
    /* --- select individuals */
    int i; /* loop variables */
    IND **p; /* exchange buffer */

    i = pop->size; /* select 'popsize' individuals */
    if (elitist) /* preserve the best individual */
        ind_copy(pop->buf[--i], pop->best);
    while (--i >= 0) /* select (other) individuals */
        ind_copy(pop->buf[i], pop_tmsel(pop, tmsize));
    p = pop->inds; pop->inds = pop->buf;
    pop->buf = p; /* set selected individuals */
    pop->best = NULL; /* best individual is not known yet */
} /* pop_select() */
```



## EA: Crossover

- Exchange of a piece of the chromosomes between two individuals
- here: so called **One-Point-Crossover**
  - choose cutting line between two genes by random
  - change sequences of genes on one side of the cutting line
  - Example: choose cutting line 2



## EA: Crossover

Exchange of pieces of the chromosomes between two individuals

```
void ind_cross (IND *ind1, IND *ind2)
{
    /* --- crossover of two chromosomes */
    int i;          /* loop variable */
    int k;          /* gene index of crossover point */
    int t;          /* exchange buffer */

    k = (int)(drand() *(ind1->n-1)) +1; /* choose a crossover point */
    if (k > (ind1->n >> 1)) { i = ind1->n; }
    else { i = k; k = 0; }
    while (--i >= k) { /* traverse smaller section */
        t = ind1->genes[i];
        ind1->genes[i] = ind2->genes[i];
        ind2->genes[i] = t; /* exchange genes */
    } /* of the chromosomes */
    ind1->fitness = 1; /* invalidate the fitness */
    ind2->fitness = 1; /* of the changed individuals */
} /* ind_cross() */
```

## EA: Crossover

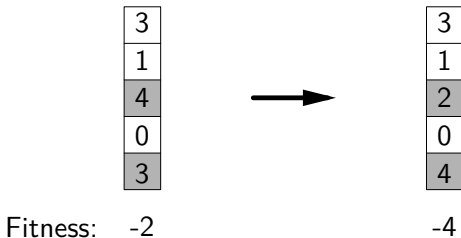
- certain rate of individuals is submitted by crossover
- include of both crossover-products in new population
- „parental individuals“ are getting lost
- no crossover on best individual (if taken over)

```
void pop_cross (POP *pop, double frac)
{
    /* --- crossover in a population */
    int i, k;          /* loop variables */

    k = (int)((pop->size -1) *frac) & ~1;
    for (i = 0; i < k; i += 2) /* crossover of pairs of individuals */
        ind_cross(pop->inds[i], pop->inds[i+1]);
} /* pop_cross() */
```

## EA: Mutation

- replacement of randomly chosen genes (changing of alleles)
- perhaps number of replaced genes is chosen by random (number of replaced genes should be as small as possible)



- mutations are mostly damaging (decrease the fitness)
- not existing alleles can be created by mutation

## EA: Mutation

- decide whether to continue mutating or not
- best individual (if taken over) won't be submitted by mutation

```
void ind_mutate (IND *ind, double prob)
{
    /* --- mutate an individual */
    if (drand() >= prob) return; /* det. whether to change individual */
    do ind->genes[(int)(ind->n *drand())] = (int)(ind->n *drand());
    while (drand() < prob);      /* randomly change random genes */
    ind->fitness = 1;            /* fitness is no longer known */
} /* ind_mutate() */
```

```
void pop_mutate (POP *pop, double prob)
{
    /* --- mutate a population */
    int i; /* loop variable */
    for (i = pop->size -1; --i >= 0; )
        ind_mutate(pop->inds[i], prob);
} /* pop_mutate() */ /* mutate individuals */
```

# Programs

- the discussed methods on solving the  $n$ -queens-problem,
  - Backtracking,
  - Analytical Solution,
  - Evolutionary Algorithms,

can be found on the lecture website as console programs  
(C-Programs `queens.c` und `qga.c`)

- Call without parameters: List of options of the program
- please note: EA not always finds a solution (Fitness  $< 0$ )
- different properties of the methods will be discussed in several exercise sheets