

Evolutionary Algorithms and Swarm-based Optimization Methods

Dept. of Mathematics / Dept. of Computer Science
Paris-Lodron-University of Salzburg
Hellbrunner Straße 34 / Jakob-Haringer-Straße 2, 5020 Salzburg, Austria

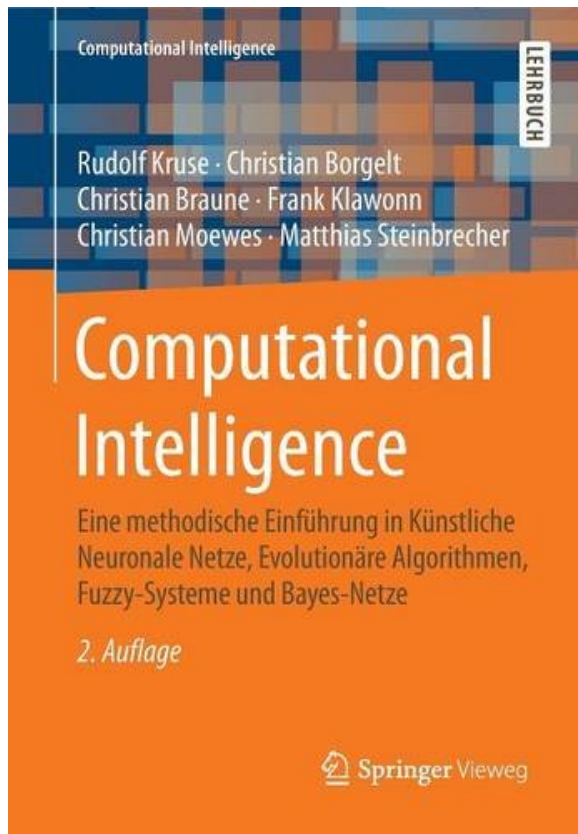
`christian.borgelt@sbg.ac.at`

`christian@borgelt.net`

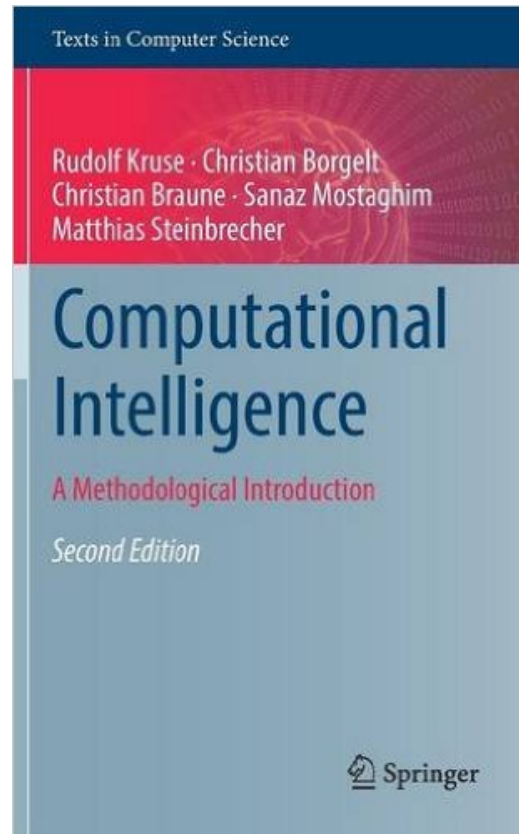
`http://www.borgelt.net/`

`https://meet.google.com/zsy-ksvs-gdv`

Textbooks



Textbook, 2nd ed.
Springer-Verlag
Heidelberg, DE 2015
(in German)



Textbook, 2nd ed.
Springer-Verlag
Heidelberg, DE 2016
(in English)

This lecture follows the second part of these books fairly closely, which treat evolutionary algorithms & swarm intelligence.

Introduction

Classification: Soft Computing / Computational Intelligence

Soft Computing = **Artificial Neural Networks** (in parallel)
+ **Evolutionary Algorithms & Swarm Intelligence**
+ **Fuzzy Systems**
+ **Probabilistic Graphical Models**

Soft Computing / Computational Intelligence is characterized by:

- Often “**model free**” approaches
(that is, no explicit model of the real-world domain is needed;
“model-based” on the other hand: e.g. solving differential equations).
- **Approximation** instead of exact solutions (not always sufficient!)
- Getting usable/sufficiently good solutions in less time,
possibly even without a deeper analysis of the problem.

Contents

- **Metaheuristics**
- **Evolutionary Algorithms**
 - Basic Notions and Concepts
(optimization problems, biological/simulated evolution, related optimization techniques)
 - Elements of Evolutionary Algorithms
(encoding, fitness and selection, genetic operators)
 - Fundamental Evolutionary Algorithms
(genetic algorithms, evolution strategies, genetic programming, multi-criteria optimization)
- **Computational Swarm Intelligence**
 - Basic Principles of Computational Swarm Intelligence
 - Particle Swarm Optimization
(including multi-objective particle swarm optimization)
 - Ant Colony Optimization

Metaheuristics

- The methods discussed in this lecture belong to the area of **metaheuristics**.
- Metaheuristics are **fairly general computational techniques** to solve **numerical and combinatorial optimization problems** in several iterations (as opposed to analytically and exactly in a single step).
- Are generally defined as an **abstract sequence of operations** on certain objects/data structures.
- Are applicable to **essentially arbitrary problems**.
- However, the objects operated on and the steps to be carried out must be **adapted to the specific problem at hand**.
- Core task is usually to find a **proper mapping** of a given problem to the abstract structures and operations that constitute the metaheuristic.
- Metaheuristics are often inspired by biological or physical processes: evolutionary algorithms, ant colony optimization, simulated annealing etc.

Metaheuristics

- Metaheuristics are usually applied to problems for which **no efficient solution algorithm is known**.
(All known algorithms have an (asymptotic) time complexity that is exponential in the problem size, e.g. NP-hard problems.)
- In practice, such problems can **rarely be solved exactly**, due to the very high demands on computing time and/or computing power.
- As a consequence, **approximate solutions** have to be accepted, and this is what metaheuristics can provide.
- There is **no guarantee** that
 - they will find the optimal solution
 - or even a solution of a given minimum quality(although this is not impossible either).
- However, they usually offer good chances of finding a **“sufficiently good” solution**.

Metaheuristics

- The success and the execution time of metaheuristics depend critically on a **proper mapping of the problem** to the steps of the metaheuristic and the efficient implementation of every single step.
- Many metaheuristics work by **iteratively improving** a set of so-called **candidate solutions**.
(This may happen either sequentially or in parallel.)
- Different metaheuristics usually differ in
 - the methods they employ to **vary solutions** in order to possibly improve them,
 - in the principles by which **partial solutions are combined** or elements of found solutions are exploited to find new solutions,
 - as well as in the principles by which a new set of candidate solutions is **selected or created** from the previously created ones.

Metaheuristics: Guided Random Search

- Metaheuristics share that they usually carry out a **guided random search** in the space of solution candidates:
 - they carry out a search that contains certain random elements to explore the search space, but
 - they are also guided by some measure of the solution quality, which governs which (parts of) solution candidates are kept or even focused on and which are discarded, because they are not promising.
- An important advantage of metaheuristics is the fact that they can usually be terminated after any iteration step (so-called **anytime algorithms**):
 - they have, at any point in time, at least some solution candidates available;
 - from these the best solution candidate found can be retrieved and returned, regardless of whether some other termination criterion is met or not.
 - However: **The solution quality is usually the better, the longer the search can run.**

Metaheuristics: Guided Random Search

- There is a wide range of metaheuristics based on various principles, many of which are **nature-inspired**:
 - evolutionary algorithms rely on principles of biological **evolution**,
 - (particle) swarm optimization mimics the behavior of **swarms of animals** (like fish or birds) that search for food in schools or flocks,
 - ant colony optimization mimics the **path finding behavior** of ants and termites.
- Other biological entities that inspired metaheuristics include honey bees and the immune system of vertebrates.
- Alternatives draw on **physical** rather than biological **analogies**, e.g.:
 - simulated annealing (mimics annealing processes),
 - threshold accepting or
 - the (great) deluge algorithm.

Evolutionary Algorithms: Biological Evolution

Biological Evolution

- Evolutionary algorithms are among the oldest & most popular metaheuristics.
- They are based on the **Theory of Biological Evolution**

Charles R. Darwin: “On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life”, 1859

Recommendable (non-technical) introductions to the theory of biological evolution are the books by Richard Dawkins, e.g. “The Selfish Gene” (1976), “The Blind Watchmaker” (1986) or “Climbing Mount Improbable” (1996).

- The core principle of biological evolution can be formulated as:

Beneficial traits resulting from random variation are favored by natural selection.

(Individuals with beneficial traits have better chances to procreate and multiply, which may also be captured by the expression **differential reproduction**.)

- The theory of biological evolution explains the diversity and complexity of all forms of living organisms and allows us to unify all biological disciplines.

Biological Evolution

New or at least modified traits may be created by various processes:

- The both blind and purely random modification of genes, that is, **mutation**, which affects both sexually and asexually reproducing life forms.

Mutations may occur due to exposure to *radioactivity* (e.g. caused by earth or cosmic radiation or nuclear reactor disasters) or to so-called *mutagens* (i.e. chemical compounds that disturb the genetic copying process), but may also happen simply naturally, due to an unavoidable susceptibility of the complex genetic copying process to errors.

- In sexual reproduction equally blindly and purely randomly selected halves of the (diploid) chromosome sets of the parents are (re-)combined, thus creating new combinations of traits and physical characteristics.

In addition, during *meiosis* (i.e. the cell division process that produces the germ cells or *gametes*, parts of (homologous) chromosomes, in a process called **crossover** (or *crossing over*), may cross each other, break, and rejoin in a modified fashion, thus exchanging genetic material between (homologous) chromosomes.

Biological Evolution

- As a result offspring with **new or at least modified genetic plans** and thus physical traits is created.
- The vast majority of (genetic) modifications are unfavorable or even harmful, in the worst case rendering the resulting individual unable to live.
- However, there is a (small) chance that some of them result in (small) improvements, endowing the individual with traits that help it to survive.
 - For example, they may make it better able to find food, defend itself against predators or hide or run from them, attract mates for reproduction etc.
 - Generally speaking,
each individual is put to the test in its natural environment
 - * where it either proves to have a high fitness, making it more likely to procreate and multiply,
 - * or where it fails to survive or mate, thus causing it or its traits to disappear.

Biological Evolution

- The natural selection process is driven by both
 - the **natural environment** and
 - the **individual traits**,leading to different reproduction rates or probabilities.
- Life forms with traits that are better fitted to their environment usually have more offspring on average.
Consequently their traits become more frequent with each generation of individuals.
- On the other hand, life forms with traits less favorable in their environment usually have less offspring on average.
Consequently they might even become extinct after some generations (at least in this environment).

Biological Evolution

**Important: A trait is not beneficial or harmful in itself,
but only w.r.t. the environment.**

- While the dark skin color of many Africans protects their skin against the intense sun in regions close to the equator, their skin pigmentation can turn out to be a disadvantage in regions where sunlight is scarce, because it increases the risk of vitamin D deficiency, as vitamin D is produced in the skin under the influence of ultraviolet light.
- On the other hand, humans with little skin pigmentation may be less likely to suffer from vitamin D deficiency, but may have to take more care to protect their skin against sunlight to reduce the risk of premature skin aging and skin cancer.
- Sickle cell anemia is a usually harmful misshaping of hemoglobin (red blood cells), because it reduces the capability of hemoglobin to transport oxygen.
- However, it has certain protective effects against malaria and thus can be an advantage in regions in which malaria is common.

Biological Evolution

- It is the interaction of random variation and natural selection that explains why there are so many complex life forms, even though these life forms are extremely unlikely to come into existence as the result of pure chance.
- **Evolution is *not* just pure chance.**
 - Although the *variations are blind and random, their selection is not*, but strictly driven by their benefit for the survival and procreation of the individuals endowed with them.
 - As a consequence, small improvements can accumulate over many generations and finally lead to surprising complexity and strikingly fitting adaptations to an environment.
- We tend to underestimate the chance of complex life forms evolving, because
 - we only see the successes (the “survivors”)
 - we have trouble imagining the time (actually billions of years) that has passed since the first, extremely simple cells assembled.

Principles of Organismic Evolution I

A more detailed analysis reveals that there are more evolutionary principles than **variation** (mutation and recombination) and **selection**:

- **Diversity**

All forms of life—even organisms of the same species—*differ* from each other, and not just physically, but already in their genetic material (*biological diversity* or *diversity of species*). Nevertheless, the currently actually existing life forms are only a tiny fraction of the theoretically possible ones.

- **Variation**

Mutation and genetic recombination (in sexual reproduction) continuously create *new variants*. These new variants may exhibit a new combination of already existing traits or may introduce a modified and thus new trait.

- **Inheritance**

As long as variations enter the germ line, they are *heritable*, that is, they are genetically passed on to the next generation. However, there is generally no inheritance of acquired traits (so-called *Lamarckism* [Jean Baptiste de Lamarck 1809]).

(Gerhard Vollmer: “Der wissenschaftstheoretische Status der Evolutionstheorie — Einwände und Gegenargumente”; in: “Biophilosophie”, Reclam, Stuttgart 1995)

Principles of Organismic Evolution II

- **Speciation**

Individuals and populations *diverge genetically*.

Thus new *species* are created once their members cannot crossbreed any longer. Speciation gives the phylogenetic “pedigree” its typical branching structure.

- **Birth surplus / Overproduction**

Nearly all life forms produce *more offspring*

than can ever become mature enough to procreate themselves.

- **Adaptation / Natural Selection / Differential Reproduction:**

On average, the survivors of a population exhibit such hereditary variations which *increase* their *adaptation* to the local environment.

Herbert Spencer’s “survival of the fittest” is rather misleading, though.

We prefer to speak of *differential reproduction due to different fitness*, because mere survival without offspring is without effect in the long run, especially since the life span of most organisms is limited.

(Gerhard Vollmer: “Der wissenschaftstheoretische Status der Evolutionstheorie — Einwände und Gegenargumente”; in: “Biophilosophie”, Reclam, Stuttgart 1995)

Principles of Organismic Evolution III

- **Randomness / Blind Variation**

Variations are *random*.

Although *triggered, initiated, or caused by* something, they do not favor certain traits or beneficial adaptations.

In this sense they are *non-teleological* (Greek *τελος*: goal, purpose, objective).

- **Gradualism**

Variations happen in comparatively *small steps* as measured by the complete information content (entropy) or the complexity of an organism. Thus phylogenetic changes are *gradual* and relatively slow.

(In contrast to this *saltationism*—from Latin *saltare*: to jump—means fairly large and sudden changes in development.)

- **Evolution / Transmutation / Inheritance with Modification**

Due to the adaptation to the environment, species are not immutable. They rather *evolve* in the course of time. Hence the theory of evolution opposes *creationism*, which claims that species are immutable.

(Gerhard Vollmer: “Der wissenschaftstheoretische Status der Evolutionstheorie — Einwände und Gegenargumente”; in: “Biophilosophie”, Reclam, Stuttgart 1995)

Principles of Organismic Evolution IV

- **Discrete Genetic Units**

The genetic information is stored, transferred and changed in discrete (“atomic”, from the Greek *ἄτομος*: indivisible) units. There is no continuous blend of hereditary traits.

Otherwise we might see the so-called *Jenkin's Nightmare* [Fleeming Jenkin 1867], that is, a disappearance of any differences in a population due to averaging.

- **Opportunism**

The processes of evolution are extremely opportunistic. They work only with what is present and not with what once was or could be. Better or optimal solutions are not found if intermediary stages (that are evolutionarily necessary to build these solutions) exhibit certain fitness handicaps.

- **Evolution-strategic Principles**

Not only organisms are optimized, but also the mechanisms of evolution. These include parameters like reproduction and mortality rates, life spans, vulnerability to mutations, mutation step sizes, evolutionary speed etc.

(Gerhard Vollmer: “Der wissenschaftstheoretische Status der Evolutionstheorie — Einwände und Gegenargumente”; in: “Biophilosophie”, Reclam, Stuttgart 1995)

Principles of Organismic Evolution V

- **Ecological Niches**

Competitive species can tolerate each other if they occupy different ecological *niches* (“biospheres” in a wider sense) or even create them themselves.

This is the only way the observable biological diversity of species is possible in spite of competition and natural selection.

- **Irreversibility**

The course of evolution is irreversible and unrepeatable.

- **Unpredictability**

The course of evolution is neither determined, nor programmed, nor purposeful and thus not predictable.

- **Increasing Complexity**

Biological evolution has led to increasingly more complex systems.

However, an open problem in evolutionary biology is the question how we can actually measure the complexity of life forms.

(Gerhard Vollmer: “Der wissenschaftstheoretische Status der Evolutionstheorie — Einwände und Gegenargumente”; in: “Biophilosophie”, Reclam, Stuttgart 1995)

Evolutionary Algorithms: Simulated Evolution

Simulated Evolution

- Biological evolution has created complex life forms and solved difficult adaptation problems.
- Therefore it is reasonable to assume that the same optimization principles can be used to find good solutions for (complex) optimization problems.

Optimization Problem

- Given:
 - a search space Ω (of potential solutions)
 - a function $f : \Omega \rightarrow \mathbb{R}$ to optimize (w.l.o.g.: to maximize)
 - possibly constraints that have to be respected.
- Desired:
 - an element $\omega \in \Omega$, that optimizes the function f . (preferably the global optimum of f)

We may also say that the function $f : \Omega \rightarrow \mathbb{R}$ assigns a quality assessment $f(\omega)$ to each candidate solution $\omega \in \Omega$. The objective is to find the best candidate solution as specified by f .

Optimization Problems: Example

- Find the side lengths of a box with fixed surface area S such that its volume is maximized!
 - The search space Ω is the set of all triplets (x, y, z) , that is, the three side lengths, with $x, y, z \in \mathbb{R}^+$ (i.e., the set of all positive real numbers) and $2xy + 2xz + 2yz = S$.
 - The evaluation function f is simply $f(x, y, z) = xyz$.
 - In this case the solution is unique, namely $x = y = z = \sqrt{S/6}$, that is, the box is a cube.
- Note that this example already exhibits an important feature, namely that the **search space is constrained**:

We do not consider all triplets of real numbers, but only those with positive elements that satisfy $2xy + 2xz + 2yz = S$.
- Here this is even necessary to have a well-defined solution: if x , y and z can be arbitrary real numbers, there is no (finite) optimum.

Optimization Problems

More Examples of Optimization Problems

(the following is, of course, *not* a complete list)

- **Parameter Optimization**

E.g. optimize the angle and curvature of the air intake and exhaust pipes of automobile motors to maximize power and/or minimize fuel consumption,

Generally: Find a set of (suitable) parameters such that a given real-valued function reaches a (preferably global) optimum.

- **Routing Problems**

E.g. the famous *traveling salesman problem*, in the form of

- optimized drilling of holes into a printed circuit board minimizing the distance the drill has to be moved,
- optimization of delivery routes from a central storage to individual shops,
- arrangement of printed circuit board tracks with the objective to minimize length and number of layers.

Optimization Problems

- **Packing and Cutting Problems**

- E.g. the *knapsack* (or *backpack* or *rucksack*) problem, in which a knapsack of a given (maximum) capacity is to be filled with given goods of known value and size (or weight) such that the total value is maximized,
- the *bin packing problem*, in which given objects of known size and shape are to be packed into boxes of given size and shape such that the number of boxes is minimized,
- the *cutting stock problem* in its various forms, in which geometrical shapes are to be arranged in such a way as to minimize waste (e.g., the wasted cloth after the parts of a garment have been cut out).

- **Arrangement and Location Problems**

E.g. the so-called *facility location problem*, which consists in finding the best placement of multiple facilities (e.g. distribution nodes in telephone networks), usually under certain constraints. Also known as *Steiner's problem*, because certain specific cases are equivalent to the introduction of so-called Steiner points to minimize the length of a spanning tree in a geometric planar graph.

Optimization Problems

- **Scheduling Problems**

E.g. *job shop scheduling* in its various forms, in which jobs have to be assigned to resources at certain times in order to minimize the time to complete all jobs, e.g.

- reordering of instructions by a compiler in order to maximize the execution speed of a program,
- setting up timetables for schools (constraints: the number of classrooms, the need to avoid skip hours at least for the lower grades) or
- setting up timetables for trains (constraints: the number of tracks available on certain lines and the different speeds of the trains).

- **Strategy Problems**

E.g. finding optimal strategies of how to behave in the (iterated) prisoner's dilemma and other models of game theory (common problem in economics).

Related objective: simulate the behavior of actors in economic life, where not only strategies are optimized, but also their prevalence in a population.

Optimization Problems

- **Biological Modeling**

E.g. the “Netspinner” program [Krink and Vollrath 1997], which describes the web building behavior of certain spiders by parameterized rules (e.g., number of spokes, angle of the spiral etc.).

With the help of an evolutionary algorithm, the program optimizes the rule parameters based on an evaluation function that takes both the metabolic cost of building the web as well as the chances of catching prey with it into account.

The results mimic the behavior observed in real spiders very well and thus help to understand the forces that cause spiders to build their webs the way they do.

pictures not available in online version

Optimization Problems

picture not available in online version

Optimization Problems

Optimization problems can be tackled in many different ways, but all possible approaches can essentially be categorized into four classes:

1. Analytical Solution

- Some optimization problems can be solved analytically.
- Example:
The problem of finding the side lengths of a box with given surface area S and maximal volume can be solved with the method of Lagrange multipliers. (We will not consider this method in detail in this lecture.)
- If an analytical approach exists, it is often the method of choice, because it usually guarantees that the solution is actually optimal and that it can be found in a fixed number of steps.
- However, for many practical problems no (efficient) analytical methods exists, either because the problem is not yet understood well enough or because it is too difficult in a fundamental way (e.g. because it is NP-hard).

Optimization Problems

Optimization problems can be tackled in many different ways, but all possible approaches can essentially be categorized into four classes:

2. Complete/Exhaustive Exploration

- The definition of an optimization problem already contains all candidate solutions in the form of the (search) space Ω .
- Hence one may consider simply enumerating and evaluating all of its elements.
- This approach certainly guarantees that the optimal solution will be found.
- However, it can be extremely inefficient and thus is usually applicable only to (very) small search spaces Ω .
- It is clearly infeasible for parameter optimization problems over real domains, since then Ω is infinite and thus can not possibly be explored exhaustively.

Optimization Problems

Optimization problems can be tackled in many different ways, but all possible approaches can essentially be categorized into four classes:

3. (Blind) Random Search

- Instead of enumerating all elements of the search space Ω (which may not be efficiently possible anyway), we may consider picking and evaluating random elements.
- In this process we always keep track of the best solution (candidate) found so far.
- This approach is efficient and has the advantage that it can be stopped at any time.
- However, it suffers from the severe drawback that it depends on pure luck whether we obtain a reasonably good solution.
- Therefore it usually offers only very low chances of obtaining a satisfactory solution.

Optimization Problems

Optimization problems can be tackled in many different ways, but all possible approaches can essentially be categorized into four classes:

4. Guided (Random) Search

- Instead of blindly picking random elements, we may try to exploit
 - the structure of the search space and
 - how the evaluation function f assesses similar elements.
- The fundamental idea is to exploit information that has been gained from evaluating certain solution candidates to guide the choice of the next solution candidates to examine.
- For this the evaluation of similar elements of the search space must be similar. Otherwise there is no basis on which we may transfer gathered information.
- Note that the choice of the next solution candidates to examine may still contain a random element (non-deterministic choice), though, but that the choice is constrained by the evaluation of formerly examined solution candidates.

Optimization Problems

- All **metaheuristics**, including evolutionary algorithms, fall into the last category (**guided (random) search**).
- As already pointed out, they differ mainly in
 - how the gathered information is represented
 - how it is exploited for picking the next solution candidates to evaluate.
- Although metaheuristics thus provide fairly good chances of obtaining a satisfactory solution, it should always be kept in mind that **they cannot guarantee that the optimal solution is found**.
- That is, the solution candidate they return may have a high quality, and this quality may be high enough for many practical purposes, but there might still be room for improvement.
- If the problem at hand requires to find a truly (guaranteed) optimal solution, evolutionary algorithms are not suited for the task.
- Then one has to opt for an analytical solution or an exhaustive exploration.

Optimization Problems

- It is also important to keep in mind that metaheuristics require that the evaluation function allows for **gradual improvement** (similar solution candidates have similar quality).
- In evolutionary algorithms the evaluation function is motivated by the biological fitness or aptitude in an environment and thus must differentiate better and worse candidate solutions.

However, it *must not* possess large jumps at random points in the search space. Rather, the change between neighboring points must be *gradual*.

- Consider, for example, an evaluation function that assigns a value of 1 to exactly one solution candidate and 0 to all others.
 - In this case, any evolutionary algorithm (actually any metaheuristic) cannot perform better than (blind) random search.
 - The reason is that the quality assessment of non-optimal solution candidates does not provide any information about the location of the actual optimum.

Foundations of Evolutionary Algorithms

Basic Notions and Their Meaning I

notion	biology	computer science
individual	living organism	solution candidate
chromosome	DNA histone protein strand	sequence of comp. objects
	describes the “construction plan” and thus (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 (e.g. bit, character, number ...)
	is the fundamental unit of inheritance, which determines a (partial) characteristic of an individual	
allele (allelomorph)	form or “value” of gene	value of a computational object
	in each chromosome there is at most one form/value of a gene	
locus	position of a gene	position of a comp. object
	at each position in a chromosome there is exactly one gene	

Basic Notions and Their Meaning II

notion	biology	computer science
phenotype	physical appearance of a living organism	implementation / application of a solution candidate
genotype	genetic constitution of a living organism	encoding of a solution candidate
	phenotype and genotype are not always strictly distinguished	
population	set of living organisms	bag / multiset of chromosomes
generation	population at a point in time	population at a point in time
reproduction	creating offspring from one or multiple (usually two) (parent) organisms	creating (child) chromosomes from one or multiple (parent) chromosomes
	asexual (one parent) or sexual (two parents) reproduction	
fitness	aptitude / conformity of a living organism	aptitude / quality of a solution candidate
	determines chances of survival and reproduction	

Some Remarks about the Basic Notions I

- **chromosome:**

(from the Greek *χρωμα*: color and *σωμα*: body, thus “colored body”, because they are the colorable substance in a cell nucleus)

The division into multiple chromosomes, as it exists in nature, is usually not mimicked in computer science \Rightarrow single chromosome per individual.

- **gene:**

(from ancient Greek *γενεά*: “generation”, “descent” and *γένεσις*: “origin”; coined by Wilhelm Ludvig Johannsen)

- **allele, allelomorph:**

(from the Greek *αλληλων*: “each other”, “mutual”, and *μορφή*: “shape”, “form”, because initially mainly two-valued genes were considered)

Biology: possible form of a gene, e.g. a gene may represent the color of the iris in the human eye, with alleles that code for blue, brown, green, gray etc. irises.

Computer science: simply the value of a computational object, which selects one of several possible properties of a solution candidate

In a chromosomes there is exactly one allele per gene.

Some Remarks about the Basic Notions II

- **locus:**

(from the Latin *locus*: “place”, “position”, “location”)

At any locus in a chromosome there is exactly one gene.

Usually a gene can be identified by its locus.

- **phenotype:**

Biology: the physical appearance of an organism, that is, the shape, structure and organization of its body.

The phenotype interacts with the environment and hence determines the fitness of the individual.

Computer science: implementation or application of a candidate solution, from which the fitness of the corresponding individual can be read.

- **genotype:**

The genetic configuration of an organism or encoding of a candidate solution; determines the fitness only indirectly through the phenotype it encodes.

In biology the phenotype also comprises acquired traits that are not represented in the genotype (learned behavior, bodily changes etc.).

Some Remarks about the Basic Notions III

- **population (generation):**

A set of organisms, usually of the same species (at a certain point in time).

Biology: Due to the complexity of biological genomes it is usually safe to assume that no two individuals from a population share exactly the same genetic configuration—homozygous twins being the only exception.

Computer science: Due to the usually much more limited variability of a chromosome as they are used in evolutionary algorithms, we must allow for the possibility of identical individuals (*bag* or *multiset* of individuals).

- **reproduction:**

Creation of a new generation by creating offspring from one or more organisms, in which genetic material of the parent individuals may be recombined.

Biology: If more than one parent, then usually two.

Computer science: The child creation process works directly on the chromosomes and the number of parents may exceed two.

Some Remarks about the Basic Notions IV

- **fitness:**

Measures how high the chances of survival and reproduction of an individual are due to its adaptation to its environment.

Biology; Simply defining fitness as the ability to survive can lead to a tautological “survival of the survivor”; a formally more precise notion defines the fitness of an organism as the number of its offspring organisms that procreate themselves, thus linking (biological) fitness directly to the concept of differential reproduction.

Computer science: A given optimization problem directly provides a fitness function with which solution candidates are to be evaluated.

- Simulated evolution is (usually) much simpler than biological evolution.
- Some principles of biological evolution, e.g. speciation, are usually not implemented in an evolutionary algorithm.

Elements of an Evolutionary Algorithm I

An evolutionary algorithm requires the following ingredients:

- an **encoding** for the solution candidates

Generally, the encoding of the solution candidates is highly problem-specific and there are no general rules. However, we will discuss later several aspects that attention should be paid to when choosing an encoding for a given problem.

- a method to create an **initial population**

Usually merely random strings of characters or numbers are created; however, depending on the encoding, more complex methods may be needed,

- a **fitness function** to evaluate the individuals

The fitness function mimics the environment and determines the quality of the individuals. Often the fitness function is simply identical to the function to optimize; however, it may also contain additional elements that represent constraints that need to be satisfied.

- a **selection method** on the basis of the fitness function

The selection methods determines which individuals are chosen for the creation of offspring or which individuals are transferred unchanged into the next generation.

Elements of an Evolutionary Algorithm II

Furthermore, an evolutionary algorithm consists of:

- a set of **genetic operators** to modify chromosomes
 - *Mutation* — random modification of individual genes
 - *Crossover* — recombination of chromosomes
(actually “crossing over”, named after a process in meiose (phase of cell division), in which chromosomes break and are rejoined in a crossed over fashion)
- a **termination criterion** for the search, which may be one of the following or a combination thereof:
 - a predetermined number of generations was computed
 - no improvement was observed for a predetermined number of generations
 - a predetermined minimum solution quality has been reached
- values for various **parameters**
(e.g. population size, mutation probability etc.)

Elements of an Evolutionary Algorithm: Remarks I

- **encoding:**

An encoding may be so direct that the distinction between the genotype, as it is represented by the chromosome, and the phenotype, which is the actual solution candidate, becomes blurred.

For example, for the problem of finding the side lengths of a box with given surface area that has maximum volume, we may use the triplets (x, y, z) of the side lengths, which are the solution candidates, directly as the chromosomes.

In other cases there is a clear distinction between the solution candidate and its encoding, for example, if we have to turn the chromosome into some other structure (the phenotype) before we can evaluate its fitness.

An bad choice can severely reduce the effectiveness of the evolutionary algorithm or may even make it impossible to find a sufficiently good solution.

Depending on the problem to solve, it is therefore highly recommended to spent considerable effort on finding a good encoding of the solution candidates.

Elements of an Evolutionary Algorithm: Remarks I

- **fitness function / selection method:**

A selection method may simply transform the fitness values into a selection probability, such that better individuals have higher chances of getting chosen for the next generation.

- **genetic operators:**

Depending on the problem and the chosen encoding, the genetic operators can be very generic or highly problem-specific.

The choice of the genetic operators is another element that effort should be spent on, especially in connection with the chosen encoding.

- **parameters:**

The set of parameters that need to be specified depends on the concrete problem and the choices how it is approached by an evolutionary algorithm.

Usually at least the following need to be specified:

population size, mutation probability, offspring/recombination probability.

Basic Structure of an Evolutionary Algorithm

```
procedure evolution_program;  
begin  
   $t \leftarrow 0$ ; (* initialize the generation counter *)  
  initialize pop( $t$ ); (* create an initial population *)  
  evaluate pop( $t$ ); (* and evaluate it (compute fitness) *)  
  while not termination criterion do (* termination criterion not satisfied *)  
     $t \leftarrow t + 1$ ; (* count the created generation *)  
    select pop( $t$ ) from pop( $t - 1$ ); (* select individuals according to fitness *)  
    alter pop( $t$ ); (* apply genetic operators *)  
    evaluate pop( $t$ ); (* evaluate the new population *)  
  end (* (compute new fitness) *)  
end
```

- By the selection operation a kind of “intermediate population” of individuals with (on average) high fitness is created.
- Only the individuals of this intermediate population may procreate.
- Often the steps “select” and “alter” are combined into one step.

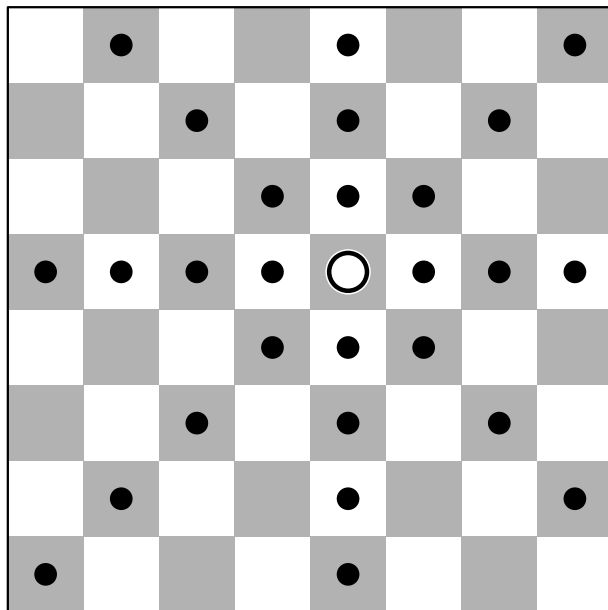
Introductory Example: The n-Queens-Problem

Introductory Example: The n-Queens Problem

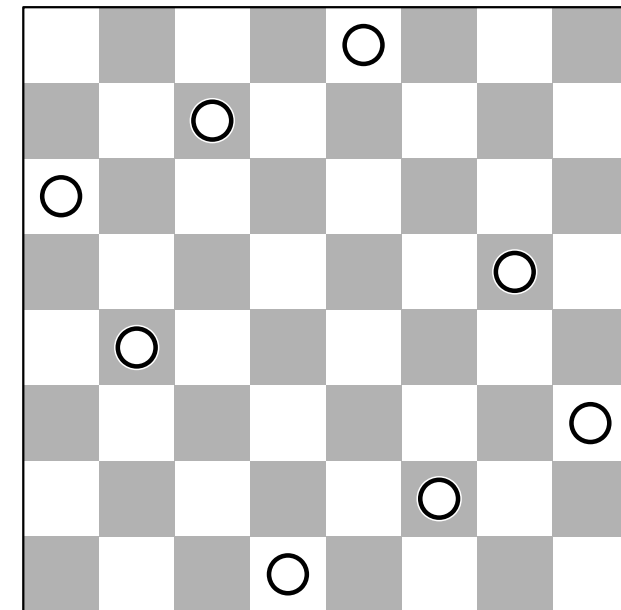
The n -queens problem consists in the task to place n queens (a piece in the game of chess) of the same color onto an $n \times n$ chessboard in such a way that no rank (chess term for row), no file (chess term for column) and no diagonal contains more than one queen.

Or: Place the queens in such a way that none obstructs any other.

possible moves of a chess queen



a solution of the 8-queens problem



n-Queens Problem: Backtracking

- A well-known approach to solve the n -queens problem is backtracking, which can be seen as an essentially exhaustive exploration of the space of candidate solutions with a depth-first search.
- Such an algorithm exploits the obvious fact that each rank (row) of the chessboard must contain exactly one queen. Hence it proceeds by placing the queens rank by rank.
- Each rank is processed as follows:
 - A queen is placed, from left to right, on each of the squares of the rank.
 - For each placement it is checked whether it causes an obstruction of the moves of any queens placed earlier (that is, whether there is already a queen on the same file (column) or the same diagonal).
 - If this is not the case, the algorithm proceeds recursively to the next rank.
 - Afterwards the queen is moved one square to the right.
- If a queen can be placed, without causing obstructions, in the last rank of the board, the found solution is reported.

n-Queens Problem: Backtracking

```
function queens (n: int, k: int, board: array of array of boolean) : boolean;
begin                                     (* recursively solve n-queens problem *)
  if  $k \geq n$  then return true;          (* if all ranks filled, abort with success *)
  for  $i = 0$  up to  $n - 1$  do begin      (* traverse the squares of rank k *)
    board[i][k]  $\leftarrow$  true;        (* place a queen on square (i, k) *)
    if not board[i][j]  $\forall j : 0 < j < k$   (* if no other queen is obstructed *)
    and not board[i - j][k - j]  $\forall j : 0 < j \leq \min(k, i)$ 
    and not board[i + j][k - j]  $\forall j : 0 < j \leq \min(k, n - i - 1)$ 
    and queens (n, k + 1, board)      (* and the recursion succeeds, *)
    then return true;                    (* a solution has been found *)
    board[i][k]  $\leftarrow$  false;      (* remove queen from square (i, k) *)
  end      (* for  $i = 0 \dots$  *)
  return false;                          (* if no queen could be placed, *)
end                                       (* abort with failure *)
```

n-Queens Problem: Backtracking

- The function on the preceding slide is called with
 - the number n of queens that defines the problem size,
 - $k = 0$ indicating that the board should be filled starting from rank 0, and
 - *board* an $n \times n$ Boolean matrix that is initialized to *false* in all elements.
- If the function returns *true*, the problem can be solved.
In this case one possible placement of the queens is indicated by the *true* entries in *board*.
- If the function returns *false*, the problem cannot be solved (the 3-queens problem, for example, has no solution).
In this case the variable *board* is in its initial state of all *false* entries.
- Note that the algorithm can easily be modified to yield all possible solutions of an n -queens problem:
Simply report a found solution and continue the search.

n-Queens Problem: Backtracking

```
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 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 */
```

- Systematically and recursively try all possibilities (exhaustive search).

n-Queens Problem: Backtracking

- The function on the preceding slide, written in C, presupposed global variables
 - `size`, the board size/number of queens to be placed and
 - `qpos`, an integer array into which the file numbers of the queen positions per rank are written.

(Note that these parameters could just as well be passed down in the recursion; this is not done here, because they are constant.)

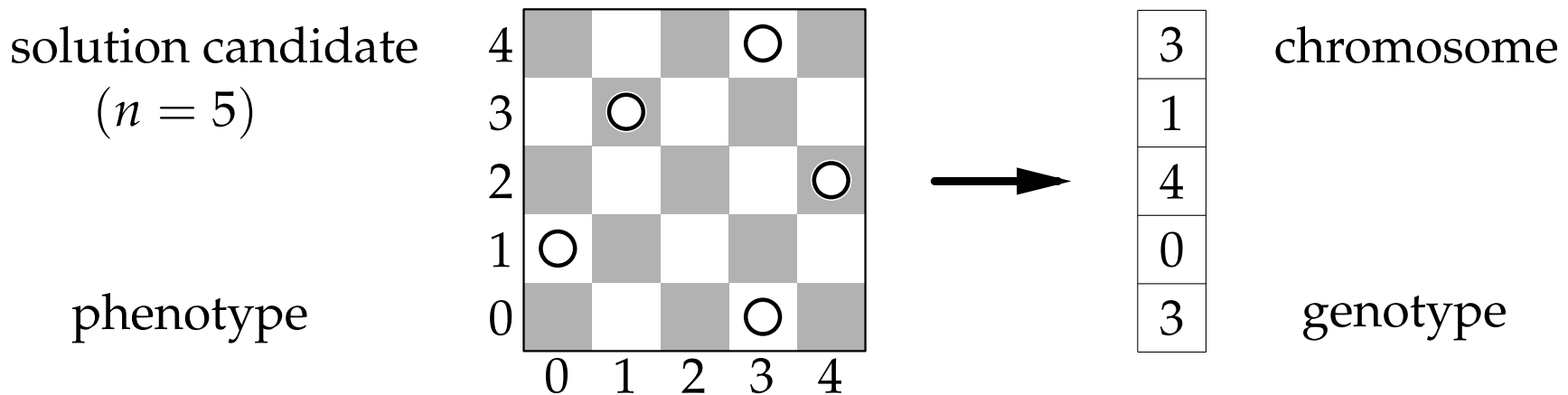
- The function is called with `y = 0` indicating that the board should be filled starting from rank 0.
- Note how the representation of a (partial) solution by a simple integer array simplifies the collision/obstruction check considerably.
- Note also that this function does not stop once a solution is found, but continues in order to report all possible solutions.
- The function finally returns the total number of solutions found.

n-Queens Problem: Analytical Solution

- Although a backtracking approach is very effective for sufficiently small n (up to, say, $n \approx 30$), it can take a long time to find a solution if n is larger.
- If we are interested in only one solution (i.e. one placement of the queens), there exists a better method, namely an analytical solution (for $n > 3$):
 - If n is odd, place a queen on the square $(n - 1, n - 1)$ and decrement n by 1.
 - If $n \bmod 6 \neq 2$: Place the queens
in the ranks $y = 0, \dots, \frac{n}{2} - 1$ into the files $x = 2y + 1$,
in the ranks $y = \frac{n}{2}, \dots, n - 1$ into the files $x = 2y - n$.
 - If $n \bmod 6 = 2$: Place the queens
in the ranks $y = 0, \dots, \frac{n}{2} - 1$ into the files $x = (2y + \frac{n}{2}) \bmod n$,
in the ranks $y = \frac{n}{2}, \dots, n - 1$ into the files $x = (2y - \frac{n}{2} + 2) \bmod n$.
- Due to this analytical solution, it is not quite appropriate to approach the n -queens problem with an evolutionary algorithm. We do so nevertheless, because this problem allows us to illustrate certain aspects very well.

Evolutionary Algorithm: Encoding

- Represent a candidate solution by a chromosome with n genes.
- Each gene refers to one rank of the chessboard and has n possible alleles. An allele indicates the file location of the queen in the corresponding rank.



- Note that this encoding the solution candidates has the advantage that we already exclude candidate solutions with more than one queen per rank. \Rightarrow smaller search space.

(Reducing the search space usually speeds up the search.)

Evolutionary Algorithm: Data Types

In the following we will look at a concrete program (written in C), that tries to solve the n -queens problem with an evolutionary algorithm.

- Data type for a chromosome, which also stores the fitness.
- Data type for a population with a buffer for the “intermediate population” and 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) */
```

Evolutionary Algorithm: Main Loop

The main loop exhibits the basic form of an evolutionary algorithm:

```
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 number of generations (still) to be computed
tmsize	size of the tournament for selecting individuals
elitist	indicates whether best individual should always be kept/transferred
frac	fraction of individuals that are subjected to crossover
prob	mutation probability

Evolutionary Algorithm: Initialization

- Creating the **initial population**:

Random sequences of n numbers in $\{0, 1, \dots, n - 1\}$ are generated.

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

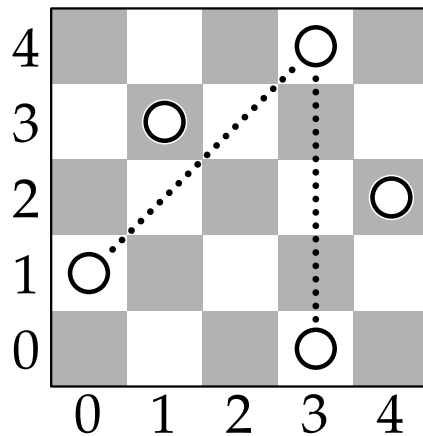
    for (i = ind->n; --i >= 0; ) /* initialize the genes randomly */
        ind->genes[i] = (int)(ind->n *drand());
    ind->fitness = 1;          /* fitness is not known yet */
} /* ind_init() */
```

```
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() */
```

Evolutionary Algorithm: Evaluation/Fitness

- **Fitness:** negated number of ranks, files and diagonals with more than one queen. (Negated number to obtain a fitness that is to be maximized.)



2 collisions \rightarrow fitness = -2

- If there are more than two queens in a rank/file/diagonal, every pair is counted (easier to implement).
- The **termination criterion** immediately follows from this fitness function: A solution has the (maximally possible) fitness 0.
- In addition: maximum number of generations, to guarantee termination. (Reminder: There is *no guarantee* that a solution will be found!)

Evolutionary Algorithm: Evaluation/Fitness

- Count collisions with computations on the chromosomes.

```
int ind_eval (IND *ind)
{
    /* --- evaluate an individual */
    int i, k;          /* loop variables */
    int d;            /* horz. distance between queens */
    int n;            /* number of collisions */

    if (ind->fitness <= 0) /* if fitness is already known, */
        return ind->fitness; /* simply return it */
    for (n = 0, i = ind->n; --i > 0; ) {
        for (k = i; --k >= 0; ) { /* traverse all pairs of queens */
            d = abs(ind->genes[i] - ind->genes[k]);
            if ((d == 0) || (d == i-k)) n++;
        } /* count number of pairs of queens */
    } /* in same column or diagonal */
    return ind->fitness = -n; /* return number of collisions */
} /* ind_eval() */
```

- Similar to procedure in backtracking approach.

Evolutionary Algorithm: Evaluation/Fitness

- The **fitness** is computed for all individuals of the population.
- At the same time the best individual is determined.
- If the best individual has fitness 0, a solution has been 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() */
```

- Note that an evolutionary algorithm cannot find all solutions (like a backtracking algorithm).

Evolutionary Algorithm: Selection of Individuals

Tournament Selection:

- Consider `tmsize` randomly chosen individuals.
- The best of these individuals “wins” the tournament and gets selected.
- The higher the fitness of an individual is, the higher is the chance that it gets selected.

```
IND* pop_tmselect (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_tmselect() */
```

Evolutionary Algorithm: Selection of Individuals

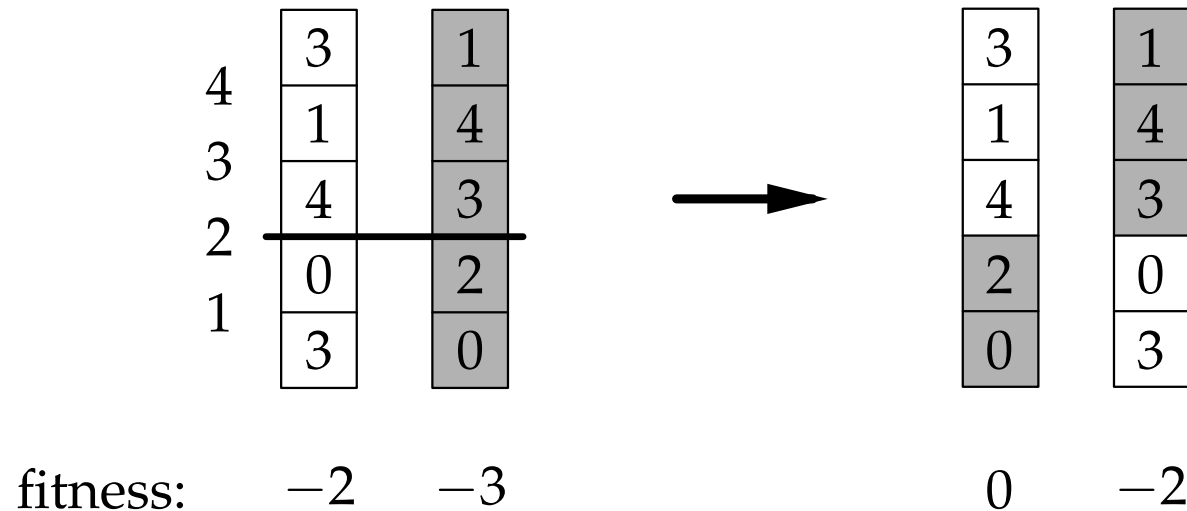
- The individuals, from which the individuals of the next generation will be created, are selected by tournament selection.
- If requested, the best individual is transferred (and not modified). (Idea: ensure that the best individual never gets worse.)

```
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() */
```


Evolutionary Algorithm: Crossover

- Exchange of a piece of a chromosome (or some other chosen subset of genes—need not be consecutive) between two individuals.
- Here: so-called **One Point Crossover**
 - Randomly choose a cut point between two genes.
 - Exchange the gene sequences on one side of the cut point.
 - Example: Choose cut point 2.



Evolutionary Algorithm: Crossover

- Exchange of a piece of a chromosome between two individuals.
(Split point is chosen randomly.)

```
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() */
```

Evolutionary Algorithm: Crossover

- A certain fraction of the individuals is subjected to **crossover**.
- Both crossover products are placed into the new population, the “parent individuals” are discarded.
- The best individual (if transferred) is **not** subjected to crossover. (Ensure that the best individual remains unchanged.)

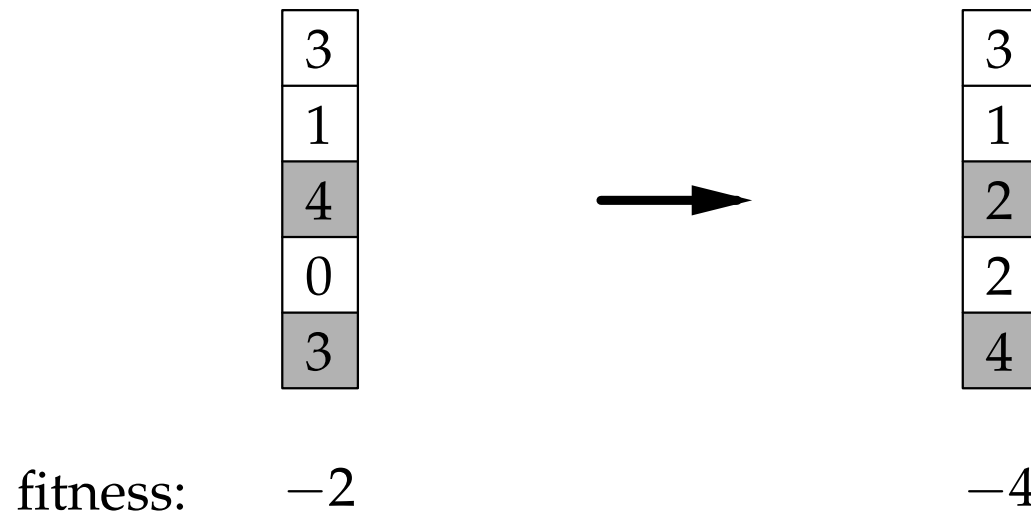
```
void pop_cross (POP *pop, double frac)
{
    /* --- crossover in a population */
    int i, k;
    /* loop variables */

    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() */
```

- The best individual is always the last in the population.
k is chosen in such a way that the last individual is never affected.

Evolutionary Algorithm: Mutation

- Randomly chosen genes are randomly changed (alleles are replaced).
- How many genes are modified may also be chosen randomly. However, the number of modified genes should be small.



- Most mutations are harmful (that is, they reduce/worsen the fitness).
- However, initially missing alleles can (only) be created by mutations.

Evolutionary Algorithm: Mutation

- It is decided whether (another) mutation is to be carried out.
- The best individual (if transferred) is **not** mutated.
(Ensure that the best individual remains unchanged.)

```
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 */
```

n-Queens Problem: Programs

- The discussed procedures to solve the n -queens problems, namely
 - backtracking,
 - analytical solution,
 - evolutionary algorithm,

are available as command line programs in the Blackboards-System.
(C-programs `queens.c` and `qea.c`, Jupyter Notebook `queens.ipynb`)

- If these programs are invoked without parameters, they print out a list of program options.
- Note that with an evolutionary algorithm it is not guaranteed that a solution is found (the reported solution candidate may have a fitness < 0).
- The properties of the discussed methods will be considered in more detail in exercises.
(The mentioned programs will be helpful for this.)

Related Optimization Methods

Related Optimization Methods

- **General Problem:**

Given a function $f : \Omega \rightarrow \mathbb{R}$ (Ω : search space),
find an element $\omega \in \Omega$, that optimizes (maximizes or minimizes) the function f .

- W.o.l.g.: Find an element $\omega \in \Omega$ that maximizes f .
(If f is to be minimized, we may consider $f' \equiv -f$ instead.)

- **Presupposition:**

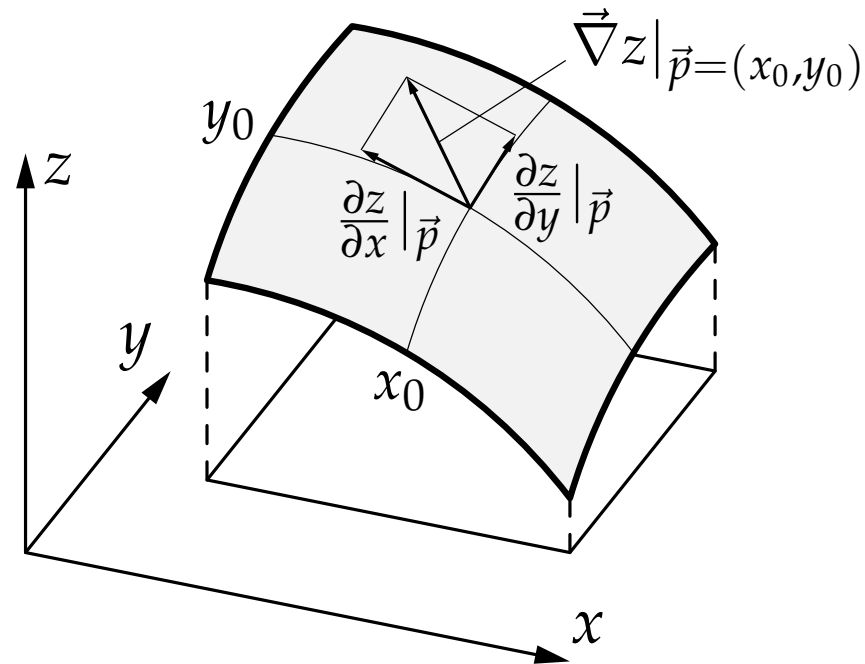
For similar elements $\omega_1, \omega_2 \in \Omega$ the function values $f(\omega_1)$ and $f(\omega_2)$
do not differ too much (no large jumps in the function values).

- **Procedures:**

- gradient methods
 - simulated annealing
 - (great) deluge algorithm
 - random ascent/descent
 - threshold accepting
 - record-to-record travel
- If one of these procedures is applicable and promises reasonable results,
it is usually preferable to evolutionary algorithms (easier to implement).

Gradient Methods

- **Presupposition:** $\Omega \subseteq \mathbb{R}^n$, $f : \Omega \rightarrow \mathbb{R}$ is differentiable
- **Gradient:** differential operator that produces a vector field.
Yields a vector in the direction of steepest ascent of a function.



- Illustration of the gradient of a function $z = f(x, y)$ at a point (x_0, y_0) .
It is $\vec{\nabla} z |_{(x_0, y_0)} = \left(\frac{\partial z}{\partial x} |_{(x_0, y_0)}, \frac{\partial z}{\partial y} |_{(x_0, y_0)} \right)$.

Gradient Methods: Cookbook Recipe

Idea: Starting from a randomly chosen point in the search space, make small steps in the search space, always in the direction of the steepest ascent (or descent) of the function to optimize, until a (local) maximum (or minimum) is reached.

1. Choose a (random) starting point $\vec{x}^{(0)} = (x_1^{(0)}, \dots, x_n^{(0)})$
2. Compute the gradient at the current point $\vec{x}^{(i)}$:

$$\nabla_{\vec{x}} f(\vec{x}^{(i)}) = \left(\frac{\partial}{\partial x_1} f(\vec{x}^{(i)}), \dots, \frac{\partial}{\partial x_n} f(\vec{x}^{(i)}) \right)$$

3. Make a small step in the direction (or against the direction) of the gradient:

$$\vec{x}^{(i+1)} = \vec{x}^{(i)} \pm \eta \nabla_{\vec{x}} f(\vec{x}) \Big|_{\vec{x}^{(i)}}.$$

+ : gradient ascent
- : gradient descent

η is a step width parameter (“learning rate” in artificial neuronal networks)

4. Repeat steps 2 and 3, until some termination criterion is satisfied.
(e.g., a certain number of steps has been executed, current gradient is small)

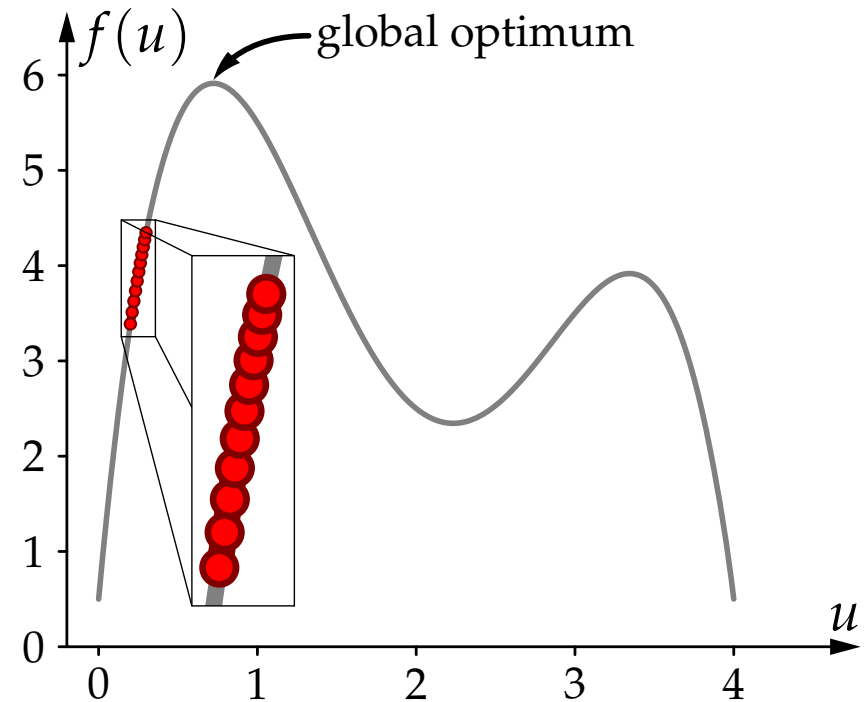
Gradient Methods: Challenges

- **Choice of the step width parameter**
 - If the value is chosen (very) small, it can take a long time until a maximum is reached (steps too small).
 - If the value is chosen too large, oscillations (jumping back and forth in the search space) may result (steps too large).
 - Possible solutions: momentum term, adaptive step width parameter (details: see lecture “Artificial Neural Networks”)
- **Getting stuck in local maxima**
 - Since only local steepness information is exploited, it may happen that only a local maximum is reached.
 - There is no *principled* solution to this problem (*no guarantees!*).
 - Improving the chances of finding the global optimum: Execute gradient ascent/descent many times with different starting points.

Gradient Methods: Examples

Example function: $f(x) = -\frac{5}{6}x^4 + 7x^3 - \frac{115}{6}x^2 + 18x + \frac{1}{2}$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	0.200	3.388	11.147	0.011
1	0.211	3.510	10.811	0.011
2	0.222	3.626	10.490	0.010
3	0.232	3.734	10.182	0.010
4	0.243	3.836	9.888	0.010
5	0.253	3.932	9.606	0.010
6	0.262	4.023	9.335	0.009
7	0.271	4.109	9.075	0.009
8	0.281	4.191	8.825	0.009
9	0.289	4.267	8.585	0.009
10	0.298	4.340		



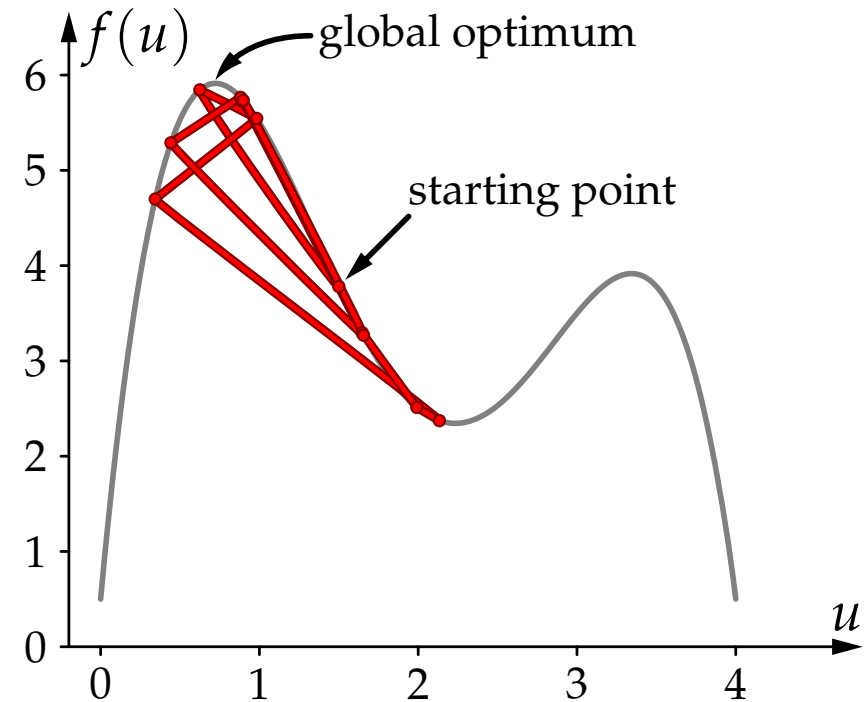
Gradient ascent with starting point 0.2 and step width parameter 0.001.

Due to the small step width/learning rate, the maximum is approached slowly.

Gradient Methods: Examples

Example function: $f(x) = -\frac{5}{6}x^4 + 7x^3 - \frac{115}{6}x^2 + 18x + \frac{1}{2}$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	1.500	3.781	-3.500	-0.875
1	0.625	5.845	1.431	0.358
2	0.983	5.545	-2.554	-0.639
3	0.344	4.699	7.157	1.789
4	2.134	2.373	-0.567	-0.142
5	1.992	2.511	-1.380	-0.345
6	1.647	3.297	-3.063	-0.766
7	0.881	5.766	-1.753	-0.438
8	0.443	5.289	4.851	1.213
9	1.656	3.269	-3.029	-0.757
10	0.898	5.734		



Gradient ascent with starting point 1.5 and step width parameter 0.25.

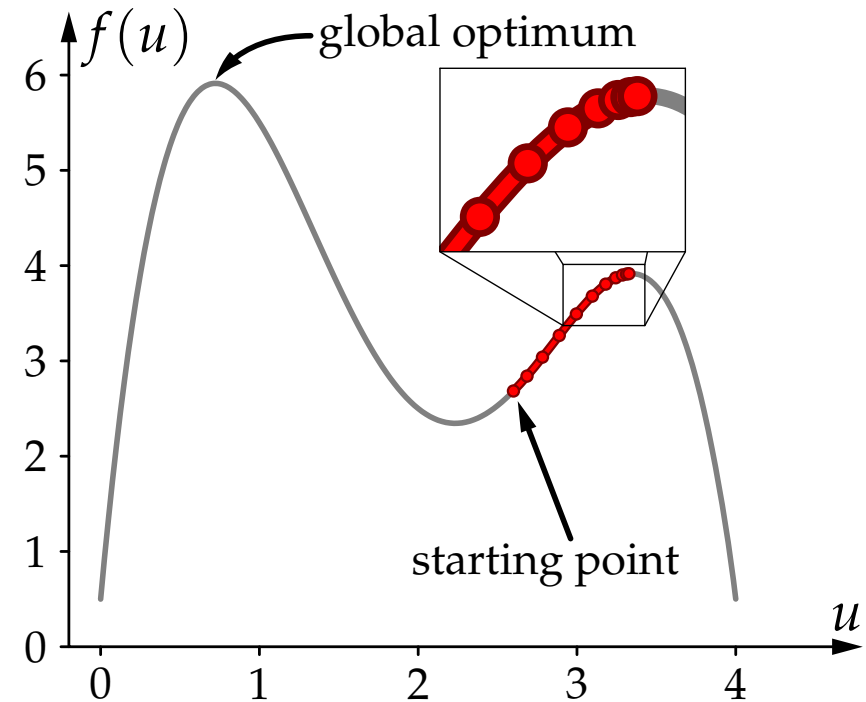
Due to the large step width/learning rate, the iterations lead to oscillations.

Gradient Methods: Examples

Example function:

$$f(x) = -\frac{5}{6}x^4 + 7x^3 - \frac{115}{6}x^2 + 18x + \frac{1}{2}'$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	2.600	2.684	1.707	0.085
1	2.685	2.840	1.947	0.097
2	2.783	3.039	2.116	0.106
3	2.888	3.267	2.153	0.108
4	2.996	3.492	2.009	0.100
5	3.097	3.680	1.688	0.084
6	3.181	3.805	1.263	0.063
7	3.244	3.872	0.845	0.042
8	3.286	3.901	0.515	0.026
9	3.312	3.911	0.293	0.015
10	3.327	3.915		



Gradient ascent with starting point 2.6 and step width parameter 0.05.

Proper step width, but due to the starting point, only a local maximum is found.

Random Ascent/Descent (Hill Climbing)

- **Idea:** If the function f is not differentiable, one may try to determine a direction in which the function f increases (or decreases) by probing random points in the vicinity of the current point.

1. Choose a random starting point $\omega_0 \in \Omega$.

2. Choose a point $\omega' \in \Omega$ “in the vicinity” of the current point ω_i .
(e.g., by a small random variation of ω_i)

3. Set

$$\omega_{i+1} = \begin{cases} \omega', & \text{if } f(\omega') \geq f(\omega_i), & (\leq \text{ for descent}) \\ \omega_i, & \text{otherwise.} \end{cases}$$

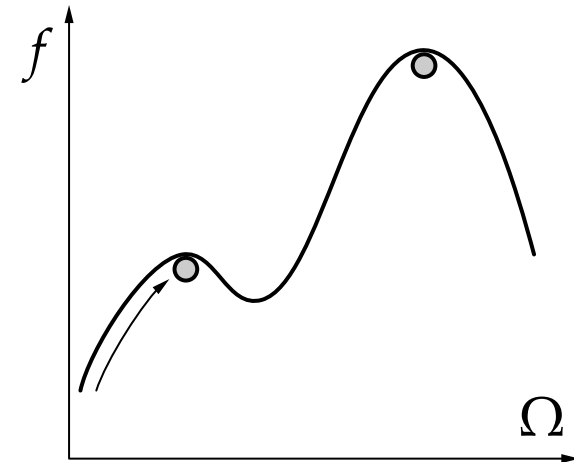
4. Repeat steps 2 and 3, until some termination criterion is satisfied.

- **Challenge/Problem:** Getting stuck at local optima.

All methods discussed in the following are attempts to mitigate this problem.

Simulated Annealing

- Can be seen as an extension of gradient and random ascent/descent that tries to avoid (or at least reduce the risk of) getting stuck.
- **Idea:** Transitions from a lower to a higher value (or even local maximum) should be more probable than the other way around.



Principle of Simulated Annealing:

- Random variations of the current solution candidate are generated.
- Better solution candidates are always accepted.
- Worse solution candidates are accepted with a certain probability that depends on
 - the quality difference between current and new solution candidate and
 - a temperature parameter, that is reduced in the course of time.

Simulated Annealing

- **Motivation:** (minimization instead of maximization)

Physical minimization of energy (to be more specific: the atom lattice energy), if a heated piece of metal is cooled down slowly.

This process is called **annealing**.

It serves the purpose to make a piece of metal easier to work or to machine by releasing internal tensions and instabilities.

(The atom lattice becomes more regular → lower atom lattice energy.)

- **Alternative Motivation:** (minimization as well)

A ball rolls around on an unevenly curved (“wavy”) surface.

The function to minimize is the potential energy of the ball.

At the beginning the ball has a certain kinetic energy, due to which it can roll uphill for some distance. However, due to friction between ball and surface the energy of the ball reduces, so that they finally comes to rest in a valley.

- **Attention:** There is *no guarantee* that the global optimum will be found!
Only the chances are better that a “good” (local) optimum will be found.

Simulated Annealing

1. Choose a (random) starting point $\omega_0 \in \Omega$.
2. Choose a (random) point $\omega' \in \Omega$ “in the vicinity” of the current point ω_i (e.g. by a small random variation of ω_i).

3. Set

$$\omega_{i+1} = \begin{cases} \omega', & \text{if } f(\omega') \geq f(\omega_i), \\ & (\leq \text{ for descent}) \\ \omega' & \text{with probability } p = e^{-\frac{\Delta f}{kT}}, \\ \omega_i & \text{with probability } 1 - p, \end{cases} \text{ otherwise.}$$

$\Delta f = f(\omega_i) - f(\omega')$ quality difference of the solution candidates
 $k = \Delta f_{\max}$ (estimate of the) range of the function values
 T temperature parameter; is (slowly) reduced over time

4. Repeat steps 2 and 3, until some termination criterion is satisfied.
- For small T the method approaches hill climbing (random ascent/descent). For larger T there is still a tendency to improve the solution candidates.

Threshold Accepting

- **Idea:** Similar to simulated annealing, worse solution candidates are accepted, but with an **upper limit for the quality reduction**.

1. Choose a (random) starting point $\omega_0 \in \Omega$.
2. Choose a (random) point $\omega' \in \Omega$ “in the vicinity” of the current point ω_i (e.g. by a small random variation of ω_i).

3. Set

$$\omega_{i+1} = \begin{cases} \omega', & \text{if } f(\omega') \geq f(\omega_i) - \theta, & \text{(ascent)} \\ & \text{if } f(\omega') \leq f(\omega_i) + \theta, & \text{(descent)} \\ \omega_i, & \text{otherwise.} \end{cases}$$

θ threshold for accepting worse solutions;
is (slowly) reduced over time.

($\theta = 0 \rightarrow$ standard hill climbing)

4. Repeat steps 2 and 3, until some termination criterion is satisfied.

(Great) Deluge Algorithm

- **Idea:** Similar to simulated annealing, worse solution candidates are accepted, but with a **lower bound for the solution quality**.

1. Choose a (random) starting point $\omega_0 \in \Omega$.
2. Choose a (random) point $\omega' \in \Omega$ “in the vicinity” of the current point ω_i (e.g. by a small random variation of ω_i).

3. Set

$$\omega_{i+1} = \begin{cases} \omega', & \text{if } f(\omega') \geq \theta, & (\leq \text{ for descent}) \\ \omega_i, & \text{otherwise.} \end{cases}$$

θ lower (upper) limit for the solution quality;

is (slowly) increased (reduced) over time, e.g. as $\theta = \theta_0 + i \cdot \eta$.

(Intuitively: “flooding” of the “function mountain range”, “deluge”, solution candidates are acceptable only if they “sit on dry land”, θ corresponds to the water level of the flood, η to the “amount of rain”)

4. Repeat steps 2 and 3, until some termination criterion is satisfied.

Record-To-Record Travel

- **Idea:** Similar to the (great) deluge algorithm, a rising “water level” is used, but this level is not absolute, but linked to the best individual found so far.

1. Choose a (random) starting point $\omega_0 \in \Omega$ and set $\omega_{\text{best}} = \omega_0$.
2. Choose a (random) point $\omega' \in \Omega$ “in the vicinity” of the current point ω_i (e.g. by a small random variation of ω_i).

3. Set

$$\omega_{i+1} = \begin{cases} \omega', & \text{if } f(\omega') \geq f(\omega_{\text{best}}) - \theta, \\ \omega_i, & \text{otherwise.} \end{cases} \quad (< \text{ and } + \text{ for descent})$$

and

$$\omega_{\text{best}} = \begin{cases} \omega', & \text{if } f(\omega') > f(\omega_{\text{best}}), \\ \omega_{\text{best}}, & \text{otherwise.} \end{cases} \quad (< \text{ for descent})$$

θ threshold for accepting solution candidates that are worse than the best solution found so far;
 θ is (slowly) reduced (increased) over time.

4. Repeat steps 2 and 3, until some termination criterion is satisfied.

Example: Traveling Salesman Problem

- **Given:**
 - A set of n cities (as points in a plane)
 - distances/costs of the paths between cities
- **Desired:**
 - a tour through all cities (and returning to the start) that has minimum total length/cost, and that does not visit any city more than once.
- **Mathematically:** Finding a Hamiltonian cycle [William Rowan Hamilton 1856] (visits each vertex once) of minimum total weight in a weighted graph.
- **Known:** This problem is NP-complete, that is, there is no known algorithm that solves the problem in polynomial time (on a deterministic machine). (And there cannot be any such algorithm unless $P = NP$.)
- **Therefore:** For large n only an approximate solution can be found in acceptable time (the best solution *may* be found by accident, but there is *no guarantee*).
- **Here:** Consider an approach by simulated annealing and hill climbing.

Traveling Salesman Problem: Formal Definition

- Let $G = (V, E, w)$ be a weighted graph with the vertex set $V = \{v_1, \dots, v_n\}$ (each v_i represents a city), the edge set $E \subseteq V \times V - \{(v, v) \mid v \in V\}$ (each edge represents a connection between two cities) and the edge weight function $w : E \rightarrow \mathbb{R}_+$ (which represents the distances or costs of the connections).
- The **traveling salesman problem** is the optimization problem $(\Omega_{\text{TSP}}, f_{\text{TSP}})$ where Ω_{TSP} contains all permutations π of the numbers $\{1, \dots, n\}$ that satisfy $\forall k; 1 \leq k \leq n : (v_{\pi(k)}, v_{(\pi(k) \bmod n) + 1}) \in E$ and the function f_{TSP} is defined as

$$f_{\text{TSP}}(\pi) = \sum_{k=1}^n w((v_{\pi(k)}, v_{\pi((k \bmod n) + 1)})).$$

- A traveling salesman problem is called **symmetric** if

$$\forall i, j \in \{1, \dots, n\}, i \neq j : \\ (v_i, v_j) \in E \Rightarrow (v_j, v_i) \in E \wedge w((v_i, v_j)) = w((v_j, v_i)),$$

that is, if all connections can be traversed in both directions and these have the same costs. Otherwise the traveling salesman problem is called **asymmetric**.

Example: Traveling Salesman Problem

1. Choose an initial (random) order of the cities, in which they are to be visited (i.e., choose a random initial tour).
2. Randomly choose two times two cities, which are visited consecutively in the current tours (all four cities are distinct). Cut the tour between the two cities of each pair and reverse the partial tour between the cut points.
3. If the new tour is better (shorter, cheaper) than the old, replace the old tour with the new one, otherwise replace the old tour only with a probability $p = e^{-\frac{\Delta Q}{kT}}$.

ΔQ quality difference between old and new tour

k range of the tour qualities

(may have to be estimated, e.g. $k_i = \frac{i+1}{i} \max_{j=1}^i \Delta Q_j$,

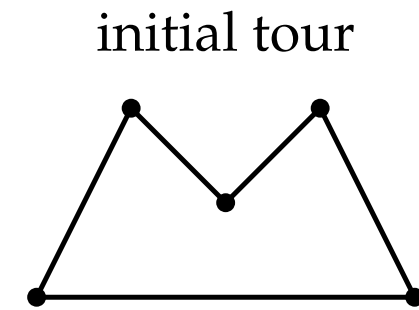
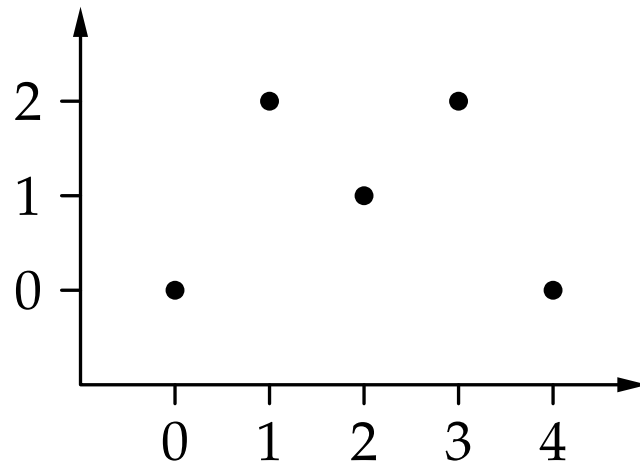
where ΔQ_j is the quality difference that was observed in the j -th step and i is the current step.)

T temperature parameter that is (slowly) reduced over time, e.g. $T = \frac{1}{i}$.

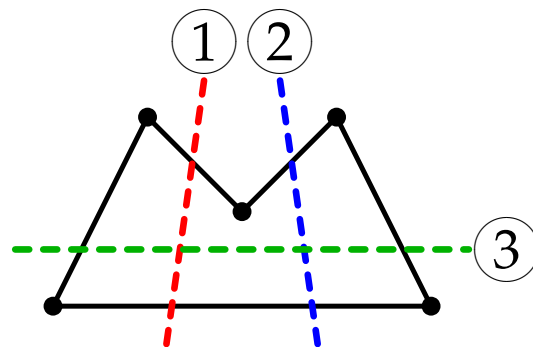
4. Repeat steps 2 and 3, until some termination criterion is satisfied.

Example: Traveling Salesman Problem

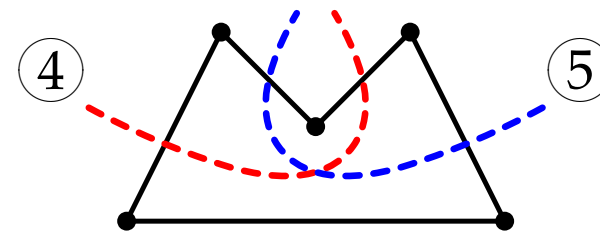
- Pure random ascent (hill climbing) may get stuck at a local minimum. This can be demonstrated with a simple example with 5 cities (costs are simply the Euclidean distances between points):



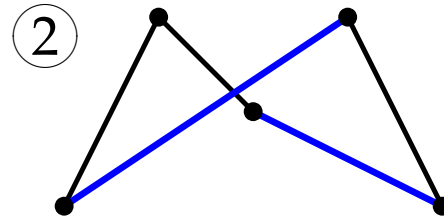
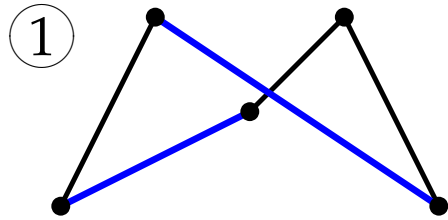
length: $2\sqrt{2} + 2\sqrt{5} + 4 \approx 11.30$



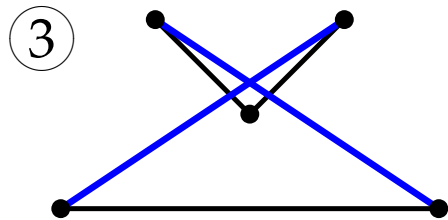
possible cuts of the initial tour



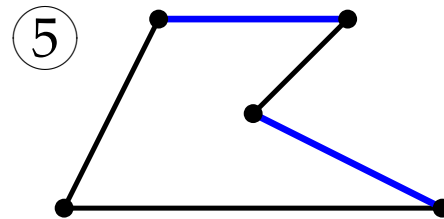
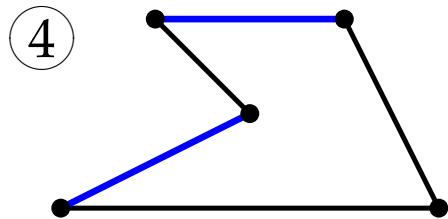
Example: Traveling Salesman Problem



length: $\sqrt{2} + 3\sqrt{5} + \sqrt{13} \approx 11.73$

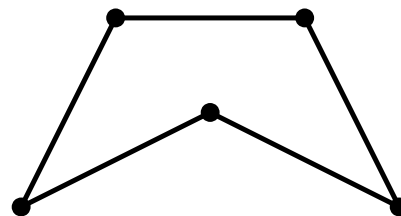


length: $2\sqrt{2} + 2\sqrt{13} + 4 \approx 14.04$



length: $\sqrt{2} + 2\sqrt{5} + 2 + 4 \approx 11.89$

best possible tour:
(global optimum)



length: $4\sqrt{5} + 2 \approx 10.94$

Example: Traveling Salesman Problem

- All modifications of the initial tour lead to alternative tours that are worse. Therefore, starting from the given initial tour, the global optimum cannot be reached with random descent (hill climbing).
- With simulated annealing, however, worse solutions are sometimes accepted, so that the global optimum may, in principle, be reached. (*However: There is no guarantee that this will happen!*)
- **Note:** It may depend on the employed set of operations, whether the search can get stuck in a local optimum.

If we add as a possible operation that the position of a city in the tour is changed (remove a city from its current position, directly connect its predecessor and successor cities, and insert the removed city at a different point in the tour), one no longer gets stuck in the discussed example.

However, one can construct an example for this extended set of operations, in which random descent (hill climbing) gets stuck in a local minimum. (*There is no general solution to this problem.*)

Related Optimization Methods: Challenges

- All considered methods search in an essentially **local** fashion:
 - Only one current solution candidate is considered.
 - The current solution candidate is modified only slightly.
(A solution candidate in its vicinity in the search space is created.)
- **Disadvantage:** Only a fairly small part of the search space may be considered.
- **Remedy:** Multiple runs of the method with different starting points.
Disadvantage: No information is transferred from one run to the next.
- **Note:** Large variations of the solution candidates, up to a (random) recreation from scratch, are not useful, since they do not transfer any / enough information from one solution candidate to the next.

(In the extreme case, one may create a random set of solution candidates and choose the best of them — blind random search.)
- ⇒ important: **information transfer between solution candidates, larger-scale exploration of the search space**

Variants of Evolutionary Algorithms

Basic Structure of an Evolutionary Algorithm

```
procedure evolution_program;  
begin  
   $t \leftarrow 0$ ; (* initialize the generation counter *)  
  initialize pop( $t$ ); (* create an initial population *)  
  evaluate pop( $t$ ); (* and evaluate it (compute fitness) *)  
  while not termination criterion do (* termination criterion not satisfied *)  
     $t \leftarrow t + 1$ ; (* count the created generation *)  
    select pop( $t$ ) from pop( $t - 1$ ); (* select individuals according to fitness *)  
    alter pop( $t$ ); (* apply genetic operators *)  
    evaluate pop( $t$ ); (* evaluate the new population *)  
  end (* (compute new fitness) *)  
end
```

- By the selection operation a kind of “intermediate population” of individuals with (on average) high fitness is created.
- Only the individuals of this intermediate population may procreate.
- Often the steps “select” and “alter” are combined into one step.

Evolutionary Algorithms: Variants

- **Standard Form:**

- Selection of an “intermediary population” with a selection method; all individuals may be mutated (even those subjected to crossover).

- **Modified Form:** (parameters r , $0 < r < \text{popsize}$, and p_c , $0 < p_c < 1$)

- From $\text{pop}(t)$ choose randomly (but taking the fitness into account) r chromosomes (**with** replacement).

Decide for each of these r chromosomes, whether it is subjected to crossover (probability p_c) or whether it is subjected to mutation (probability $1 - p_c$) and then apply these genetic operators.

- From $\text{pop}(t)$ choose randomly (but taking the fitness into account) $\text{popsize} - r$ chromosomes (**without** replacement).

These $\text{popsize} - r$ chromosomes are transferred without modification.

(That is, individuals cannot multiply in an unmodified form.)

Evolutionary Algorithms: Variants

Properties of the Modified Form:

- A chromosome is **either** subjected to crossover **or** it is mutated (but never both!).
- The unmodified chromosomes do not multiply (since they are chosen *without* replacement).

Advantages:

- It is (very) unlikely that the population contains identical individuals. (principle of *diversity* — individuals are different)
- Advantageous crossover products have better chances, since they are not harmed by additional mutations.

Disadvantages:

- Larger risk of “extinction” of good solution candidates, since there is (usually) only one copy of each individual in the population.

Evolutionary Algorithms: Variants

Steady-State Evolutionary Algorithm (parameter r , $1 \leq r \leq \text{popsize}$)

- Random selection of two parents (taking the fitness into account) and application of genetic operators to these parents.
- Duplicates (individuals already present in the population) are discarded.
- In total r offspring individuals are created, which replace r worst individuals of the population ($1 \leq r \leq \text{popsize}$, often $r = 2$).

Advantages:

- Duplicates are explicitly prevented \rightarrow better exploration of the search space
- Good/best individuals cannot become extinct (if $q < \text{popsize}$)

Disadvantages:

- Greater effort/cost for selecting the individuals.
- Somewhat higher risk of getting stuck at local optima.

Elements of Evolutionary Algorithms

1. Encoding of the Solution Candidates

Evolutionary Algorithms: Encoding

In the following we consider the elements of evolutionary algorithms in more detail.

First: **Encoding of the Solution Candidates**

- As already mentioned and emphasized, the encoding has to be chosen according to the problem to be solved.
- There is no general “cookbook recipe” for finding a (good) encoding.
- However, there are a few principles that should be paid attention to.

Desirable properties of an encoding:

- Similar phenotypes should be represented by similar genotypes.
- Solution candidates that have similar fitness, should be encoded in a similar way.
- The search space (the set of all possible solutions candidates) should, as far as possible, be closed under the used genetic operators.

Evolutionary Algorithms: Encoding

Similar phenotypes should be represented by similar genotypes.

- Clearly, mutations of individual genes lead to similar genotypes (individual changes of alleles → small change of the chromosome).
- If similar phenotypes are not represented by similar genotypes, it may not be possible to realize even improvements that are close at hand.
(In such a case a large change of the genotype is necessary in order to get to a similar (and possible better) phenotype.)

Example as an illustration:

- Optimization of a real-valued function $y = f(x_1, \dots, x_n)$.
- The (real-valued) arguments are to be represented by binary codes.
- Problem: a simple, straightforward encoding as a binary number leads to so-called “Hamming cliffs”:
Many bits have to be changed in order to change the number only slightly.

Binary Encoding of Real-Valued Numbers

- **Given:** real interval $[a, b]$ and an encoding precision ε .
- **Desired:** encoding rule for real-valued numbers $x \in [a, b]$ as binary numbers z , such that the encoding differs by no more than ε from the encoded number.
- **Approach:**
 - Divide the interval $[a, b]$ into equally sized sections of a length $\leq \varepsilon$.
→ 2^k sections with $k = \lceil \log_2 \frac{b-a}{\varepsilon} \rceil$ encoded by numbers $0, \dots, 2^k - 1$.
 - *Encoding:* $z = \lfloor \frac{x-a}{b-a} (2^k - 1) \rfloor$ (alternatively: $\lfloor \frac{x-a}{b-a} (2^k - 1) + \frac{1}{2} \rfloor$,
 - *Decoding:* $x = a + z \cdot \frac{b-a}{2^k - 1}$ in this case it suffices $k = \lceil \log_2 \frac{b-a}{2\varepsilon} \rceil$)
- **Example:** interval $[-1, 2]$, precision $\varepsilon = 10^{-6}$, $x = 0.637197$.
 - $k = \lceil \log_2 \frac{2-(-1)}{10^{-6}} \rceil = \lceil \log_2 3 \cdot 10^6 \rceil = 22$
 - $z = \lfloor \frac{0.637197-(-1)}{2-(-1)} (2^{22} - 1) \rfloor = 2288966_{10} = 100010111011010101000110_2$

Avoiding Hamming Cliffs: Gray Codes

Problem: Neighboring number may be encoded in a very different way, that is, the encodings may have a large Hamming distance (number of different bits). Large Hamming distances are difficult or sometimes even close to impossible to overcome by mutation or crossover (so-called “Hamming cliffs”).

Example: The interval of numbers from 0 to 1 is represented as 4 bit numbers, that is, by the mapping $\frac{k}{15} \rightarrow k$. Then the encodings of $\frac{7}{15}$ (0111) and $\frac{8}{15}$ (1000) have a Hamming distance of 4, since every bit is different.

Solution: Gray Codes

[Frank Gray 1953]

Neighboring numbers always differ by only a single bit.

binary	Gray
0000	0000
0001	0001
0010	0011
0011	0010

binary	Gray
0100	0110
0101	0111
0110	0101
0111	0100

binary	Gray
1000	1100
1001	1101
1010	1111
1011	1110

binary	Gray
1100	1010
1101	1011
1110	1001
1111	1000

Gray Codes: Computation

- **Gray Codes are not unique:**

Every code, in which the encodings of neighboring numbers differ by only a single bit, is called a Gray code.

- Gray Codes are usually computed from a binary number encoding (that is, from a standard encoding as it is used in computers).

- **Most commonly used method:**

- *Encoding:* $g = z \oplus \lfloor \frac{z}{2} \rfloor$ (\oplus : exclusive or of the binary representation)

- *Decoding:* $z = \bigoplus_{i=0}^{k-1} \lfloor \frac{g}{2^i} \rfloor$

- **Example:** interval $[-1, 2]$, precision $\varepsilon = 10^{-6}$, $x = 0.637197$.

- $z = \left\lfloor \frac{0.637197 - (-1)}{2 - (-1)} (2^{22} - 1) \right\rfloor = 2288966_{10} = 1000101110110101000110_2$

- $g = 1000101110110101000110_2$
 $\oplus 100010111011010100011_2$
 $= 1100111001101111100101_2$

Gray Codes: Implementation

```
unsigned int num2gray (unsigned int x)
{
    /* --- convert number to Gray code */
    return x ^ (x >> 1);      /* exclusive or with shifted input */
} /* num2gray() */
```

```
unsigned int gray2num (unsigned int x)
{
    /* --- convert Gray code to number */
    unsigned int y = x;      /* copy the input number and */
    while (x >>= 1) y ^= x;  /* do an exclusive or with all */
    return y;                /* shift downs of the input */
} /* gray2num() */          /* and return the result */
```

```
unsigned int gray2num (unsigned int x)
{
    /* --- convert Gray code to number */
    x ^= x >> 16; x ^= x >> 8; /* do an exclusive or with all */
    x ^= x >> 4; x ^= x >> 2; /* shift downs of the input */
    return x ^ (x >> 1);      /* and return the result */
} /* gray2num() */          /* (32 bit integers only) */
```


Evolutionary Algorithms: Encoding

Solution candidates that have similar fitness, should be encoded in a similar way.

- **The problem of epistasy:**

- *in biology:* An allele of a gene (the so-called *epistatic gene*) suppresses the effect (expression) of all possible alleles of some gene (the so called *hypostatic gene*) or even multiple other genes.

- *in evolutionary algorithms:*
(Strong) interdependence between the genes of a chromosomes.

How much the fitness of a solution candidate changes by a change of a certain gene depends (strongly) on what values other genes have.

- **Epistasy in biology:**

- Deviations from Mendel's rules are usually caused by epistasy.

- If homozygous beans with black and white seeds are crossbred, the second descendant generation shows beans with black, white, and brown seeds in a ratio of 12:1:3, which contradicts Mendel's rules.

Evolutionary Algorithms: Epistasy

Example of epistasy:

- **Traveling Salesman Problem**

(Find tour with minimal costs through n cities.)

- **Encoding 1:**

A tour is represented by a permutation of the cities,
(City at k -th position is visited in the k -th step.)

Low epistasy: For example, the exchange of two cities usually changes the fitness (that is, the costs of the tour) by more or less comparable amounts (local tour change).

- **Encoding 2:**

A tour is encoded by stating the position of the next city in a list of cities from which the already visited cities have been deleted.

High epistasy: Changing a single gene, especially at the beginning of a chromosome, may change almost the whole tour (global tour change) and thus often leads to large changes of the fitness.

Evolutionary Algorithms: Epistasy

Explanations about encoding 2:

effect of a mutation

	chromosome	list of cities still to visit	tour
before mutation	5	1, 2, 3, 4, 5, 6	5
	3	1, 2, 3, 4, 6	3
	3	1, 2, 4, 6	4
	2	1, 2, 6	2
	2	1, 6	6
	1	1	1

	chromosome	list of cities still to visit	tour
after mutation	1	1, 2, 3, 4, 5, 6	1
	3	2, 3, 4, 5, 6	4
	3	2, 3, 5, 6	5
	2	2, 3, 6	3
	2	2, 6	6
	1	2	2

Evolutionary Algorithms: Epistasy

- If the used encoding exhibits high epistasy, the optimization problem is often difficult to solve for an evolutionary algorithm, since regularities are scarce or even missing, which could be exploited by it. (Mutation and crossover lead to fitness changes that are (almost) random.)
- If an encoding can be found that has very low epistasy, often other methods are better suited (e.g., hill climbing), especially, if the parameters are (almost) independent.
- It has been tried to use the notion of epistasy to characterize whether certain problems are easy or difficult for an evolutionary algorithm. [Davidor 1990].

However, this is not successful:

- Epistasy is a property of the encoding, **not** of the problem itself.
(For the same problem, there may be encodings with high and with low epistasy — see the example just considered.)
- There are problems that can be encoded with low epistasy, but that are difficult to solve for an evolutionary algorithm nevertheless.

Evolutionary Algorithms: Encoding

The search space (the set of potential candidate solutions) should, as far as possible, be closed under the used genetic operators.

- What counts as leaving the search space is, depending on the concrete situation/choices made, a question of definition.
- Generally: The **search space is left** if
 - the new chromosome cannot be interpreted/decoded meaningfully,
 - the encoded solution candidate does not satisfy certain fundamental requirements a solution must possess,
 - the encoded solution candidate is misevaluated by the fitness function.
- This is mainly a problem of **adjustment** of encoding and genetic operators.
 - Using encoding-specific genetic operators.
 - Using mechanisms that “repair” chromosomes.
 - Introducing a penalty term that (considerably) reduces the fitness of chromosomes outside the search space.

Evolutionary Algorithms: Leaving the Search Space

Example of Leaving the Search Space:

- n -queens problem (placing n queens onto a $n \times n$ chess board).
- **Encoding 1:**
Chromosome of length n that contains the file positions of the queens per rank (alleles $0, \dots, n - 1$, see the introductory example).
Operators: one-point crossover, standard mutation
All mutation and crossover products are valid vectors of file positions.
→ the search space is not left.
- **Encoding 2:**
Chromosome of length n that contains the numbers (or coordinates) of the squares onto which queens are placed (alleles $0, \dots, n^2 - 1$).
Operators: one-point crossover, standard mutation
Chromosomes may result that place more than one queen onto the same square.
→ the search space may be left.

Evolutionary Algorithms: Leaving the Search Space

Possible ways to approach the problem of leaving the search space (using the n -queens problem as an example):

- **Use a different encoding:** Since the first encoding (of the two just considered) avoids the problem of leaving the search space, and leads to a much smaller search space at the same time, it is clearly the option to prefer. (If feasible, this approach is always the best option!)
- **Encoding-specific genetic operators**
 - *Mutation:* Prevent the choice of already existing alleles when mutating.
 - *Crossover:* First collect the square numbers that do not occur in the other chromosome and apply one-point crossover only to these (square numbers occurring in both chromosomes are always kept).
- **Repair mechanisms:** Find and replace duplicates (square numbers occurring more than once), so that all square numbers are different.
- **Penalty term:** Reduce the fitness by the number of duplicate (or multiple) queen placements onto the same square, possibly multiplied by a weighting factor.

Evolutionary Algorithms: Leaving the Search Space

Another example: Traveling Salesman Problem

- A tour is represented as a permutation of the cities.
(City at position k is visited in the k -th step.)
- one-point crossover may leave the space of permutations:

3	5	2	8	1	7	6	4	3	5	2	4	5	6	7	8
1	2	3	4	5	6	7	8	1	2	3	8	1	7	6	4

- **Encoding-specific Operators:**

- *Mutation*: e.g. pair exchanges, shifting a section, inversion
- *Crossover*: edge recombination (to be discussed in detail later)

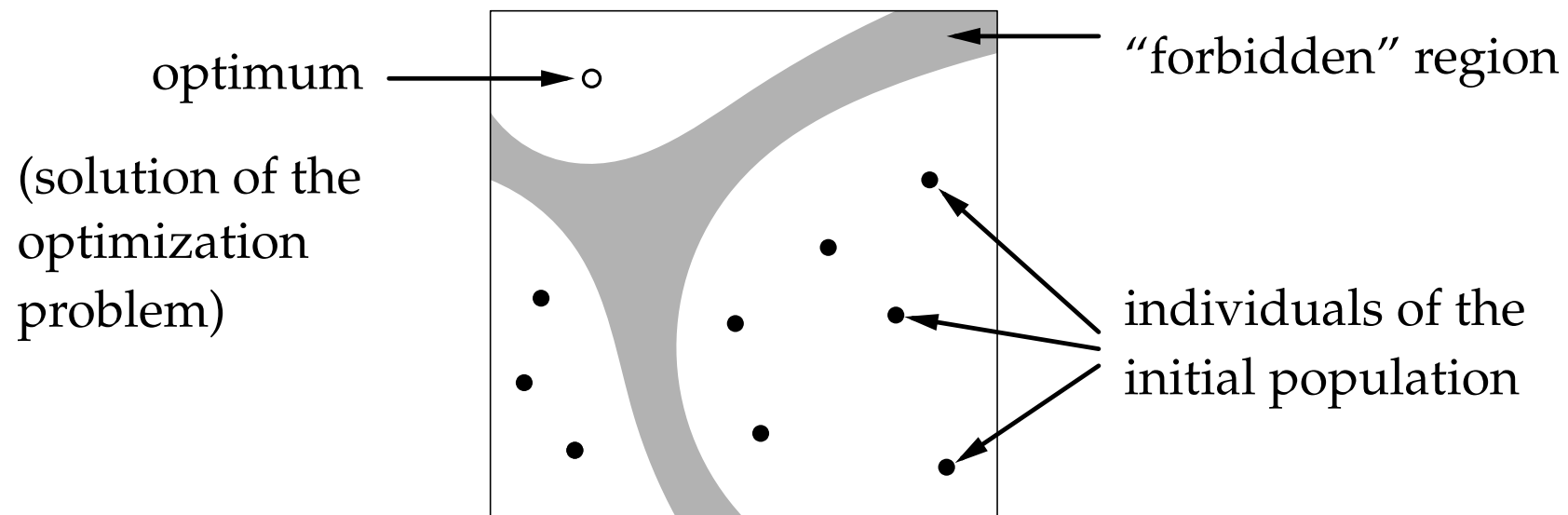
- **Repair Mechanisms:** Remove duplicate cities from a tour and append the missing cities at the end of the tour:

3	5	2	4	5	6	7	8	1
---	---	---	---	--------------	---	---	---	---

- **Penalty:** Reduce the fitness by some constant for every city that is missing.

Evolutionary Algorithms: Leaving the Search Space

- In the case of **disconnected search spaces** it may happen that **repair mechanisms** impede the search, since the solution candidates in the “forbidden” regions are “pushed back” into the admissible regions immediately.



- In such cases it is more appropriate to introduce a **penalty term** which penalizes solution candidates in the “forbidden” region, but does not eliminate them. This penalty term should grow in the course of time, in order to suppress solution candidates in the “forbidden” regions in later generations.

Elements of Evolutionary Algorithms

2. Fitness Function and Selection Methods

Evolutionary Algorithms: Selection

- **Principle of Selection:** Better individuals (better fitness) should have greater chances to reproduce, to get offspring (differential reproduction).

The strength of preference of good individuals is called **selection pressure**.

W.r.t. the selection pressure there is an antagonism between

- **Exploration of the search space:**

The individuals should be spread out as far as possible across the search space, so that the chances are maximized that the global optimum is found (although there is never any guarantee).

→ low selection pressure desirable

- **Exploitation of good individuals:**

It should be tried to approach the (possibly local) optimum in the vicinity of good individuals (convergence to the optimum), in order to approach a local optimum as closely as possible.

→ high selection pressure desirable

Evolutionary Algorithms: Selection

Choosing the selection pressure:

- **Best strategy:** time-dependent selection pressure
 - low selection pressure in early generations
 - high selection pressure in later generations
- Initially good exploration of the search space, then exploitation of the most promising region.
- The selection pressure is controlled via scaling the fitness function or via parameters of the selection procedures.
- **Important selection procedures and scaling methods:**
 - Roulette wheel selection
 - Rank selection
 - Tournament selection
 - Adaptation of the fitness function
 - Linear dynamic scaling
 - σ -scaling

Not all selection methods achieve a rising selection pressure directly.

Selection: Roulette Wheel Selection

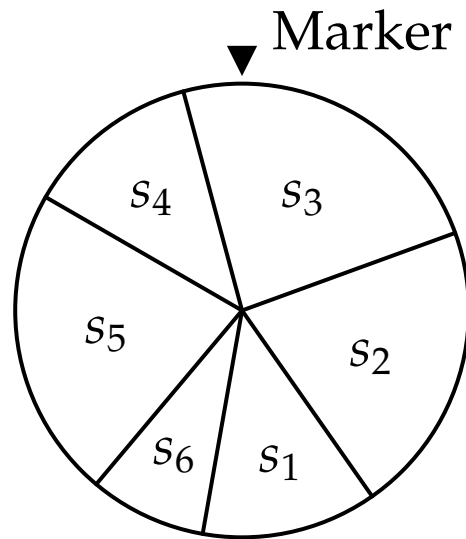
- **Roulette wheel selection** is by far the best known selection method. (Most direct way of turning fitness values into selection probabilities.)
- Approach: Compute the relative fitness of the individuals:

$$f_{\text{rel}}(\omega) = \frac{f_{\text{abs}}(\omega)}{\sum_{\omega' \in \text{pop}(t)} f_{\text{abs}}(\omega')}$$

and interpret it as a selection probability of an individual (so-called **fitness proportional selection**).

- **Note:** The absolute fitness $f_{\text{abs}}(\omega)$ must not be negative in this case; if necessary, some positive value needs to be added or all negative values must be set to zero.
- **Note:** The goal must be to maximize the fitness function. (Otherwise bad individuals would be chosen with high probability.)
- **Illustration:** Roulette wheel with one sector per individual ω . The sector sizes represent the relative fitness values $f_{\text{rel}}(\omega)$.

Selection: Roulette Wheel Selection



Selection of an individual:

- Turn roulette wheel.
- Choose chromosome, whose sector is next to the marker.

Selection of the intermediate population:

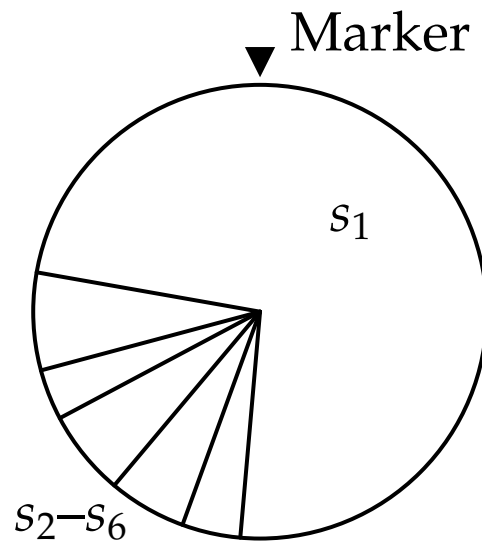
- Repeat the selection as often as there are individuals in the population.

- **Technical disadvantage of roulette wheel selection:**

In order to compute the relative fitness, the fitness values of all individuals have to be summed (normalization factor).

- Population that is selected from must be constant during selection.
- Parallelization of the implementation is made more difficult (because a central processor has to collect and the fitness value in order to compute the relative fitness).

Roulette Wheel Selection: Dominance Problem

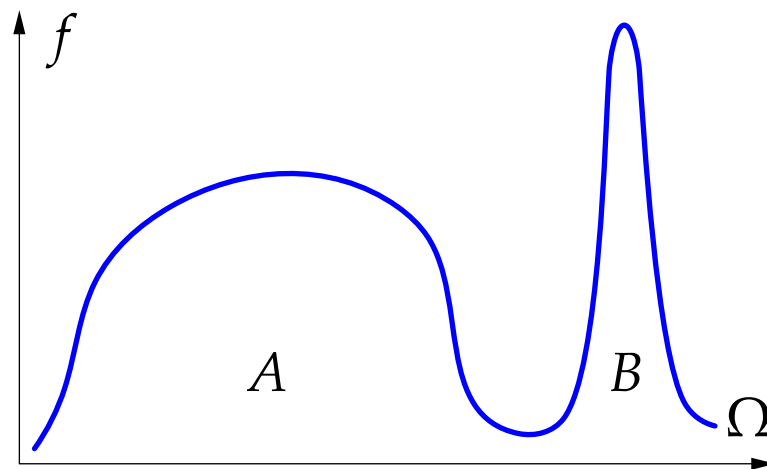


- If an individual has a very large fitness, it may **dominate** the selection.
 - In the following generations this dominance is even increased, since then copies and fairly similar individuals are present.
 - As a consequence, **crowding** may result: The population consists of identical or very similar individuals.
-
- Crowding leads to a fast convergence to a (local) optimum. (Focus is on exploitation of the good individual.)
 - **Disadvantage:** The diversity of the population is lost.
 - Exploitation of one or only few good individuals.
 - Little exploration of the search space, but local optimization. (in later generations desirable, but not desirable in earlier generations.)

Selection: Influence of the Fitness Function

The dominance problem demonstrates the strong influence of the fitness function on the effect of the fitness proportional selection.

- Problem of **premature convergence**:
If the function to maximize takes very different values (large range of possible values), premature convergence may result.
- Example: If at the beginning there is no chromosome in the region B , selection keeps the population in the vicinity of the (local) maximum in the region A , thus preventing that the global optimum is found.



Individuals that approach the transition area between the regions A and B have only very low chances of getting offspring.

Selection: Influence of the Fitness Function

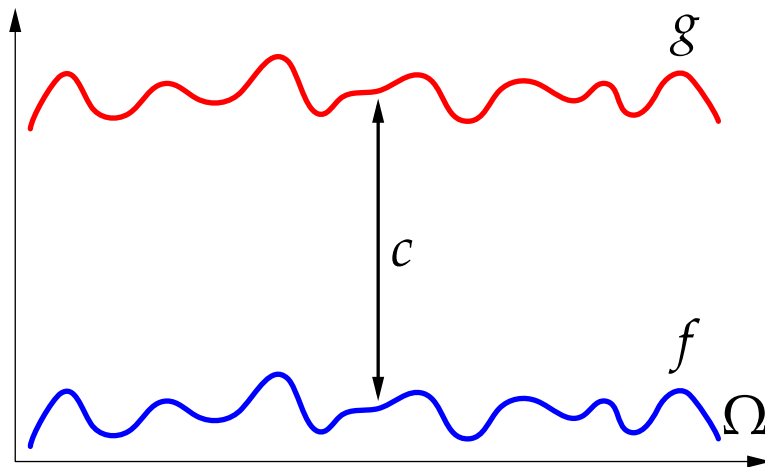
In general, there is the problem of the **absolute level** of the fitness values compared to their **variation** (range, dispersion), since there is also, vice versa, the

- Problem of **vanishing selection pressure**:

- Maximizing a function $f : \Omega \rightarrow \mathbb{R}$ is equivalent to maximizing a function $g : \Omega \rightarrow \mathbb{R}$ with $g(\omega) \equiv f(\omega) + c, c \in \mathbb{R}$.

- If $c \gg \sup_{\omega \in \Omega} f(\omega)$, we obtain $\forall \omega \in \Omega : g_{\text{rel}}(\omega) \approx \frac{1}{\text{popsize}}$.

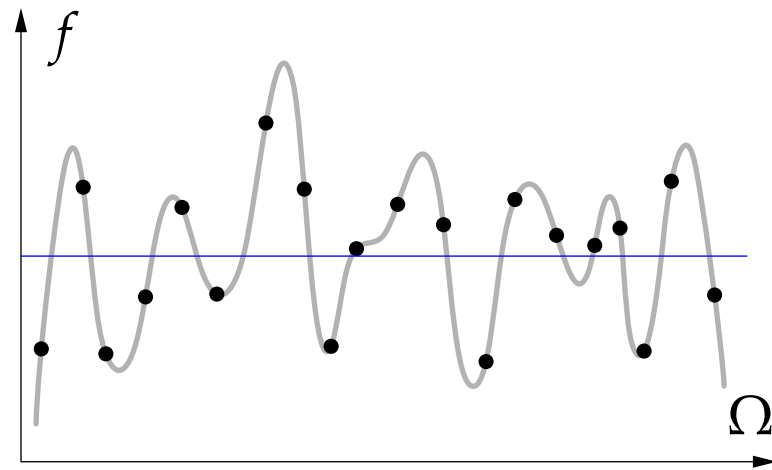
→ selection pressure is (too) low



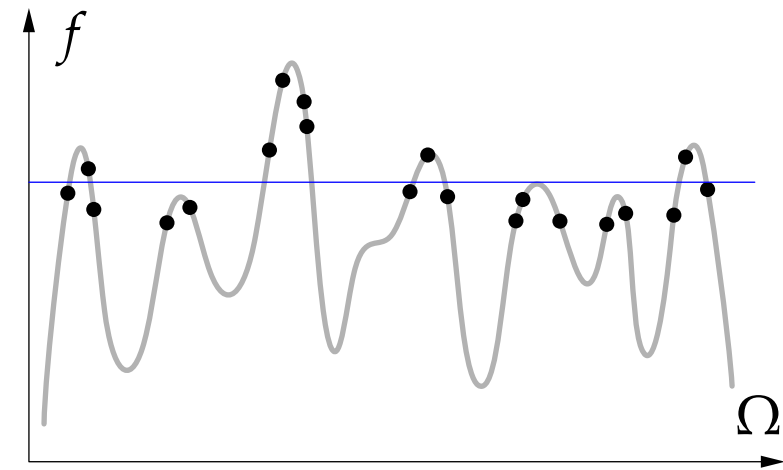
Even though the maxima are at the same locations, they are *not* equally easy to find by an evolutionary algorithm: The function g has (too) small difference in the relative fitness to produce sufficient selection pressure.

Selection: Vanishing Selection Pressure

- Since an evolutionary algorithm has the tendency to increase the (average) fitness of the individuals from generation to generation, it may create the problem of vanishing selection pressure itself, by its mere operation.
- Higher selection pressure at the beginning, since the fitness values are randomly distributed, lower selection pressure in later generations (exactly the opposite is desirable!).
- (Exaggerated) Example: The points indicate the individuals of the generation.



early generation



late generation

Selection: Adaptation of the Fitness Function

Approach to solve the discussed problems:
scaling of the fitness function/values

- **Linear dynamic scaling**

$$f_{\text{lds}}(\omega) = \alpha f(\omega) - \min\{f(\omega') \mid \omega' \in \text{pop}(t)\}, \quad \alpha > 0.$$

Instead of the minimum of the current population $\text{pop}(t)$, the minimum of the k most recent k generations may be used as well. Usually it is $\alpha > 1$.

- **σ -Scaling**

$$f_{\sigma}(\omega) = f(\omega) - (\mu_f(t) - \beta \cdot \sigma_f(t)), \quad \beta > 0.$$

$$\mu_f(t) = \frac{1}{\text{popsize}} \sum_{s \in \text{pop}(t)} f(\omega) \quad (\text{mean of the fitness values})$$

$$\sigma_f(t) = \sqrt{\frac{1}{\text{popsize} - 1} \sum_{s \in \text{pop}(t)} (f(\omega) - \mu_f(t))^2} \quad (\text{standard deviation})$$

- **Problem: How to choose the parameters α and β ?**

Selection: Adaptation of the Fitness Function

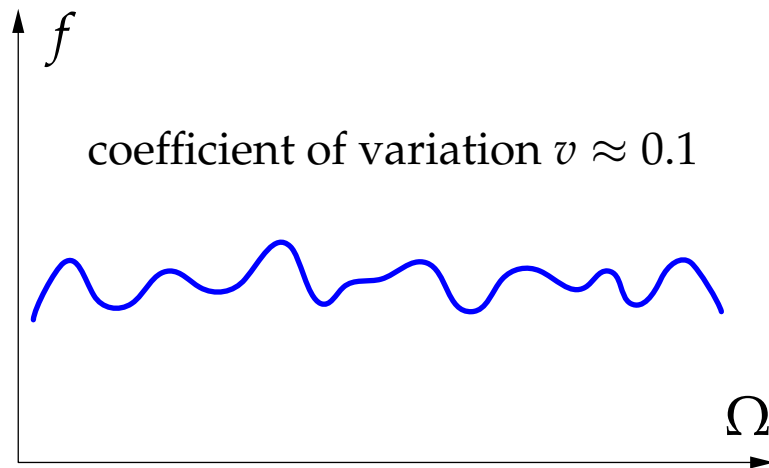
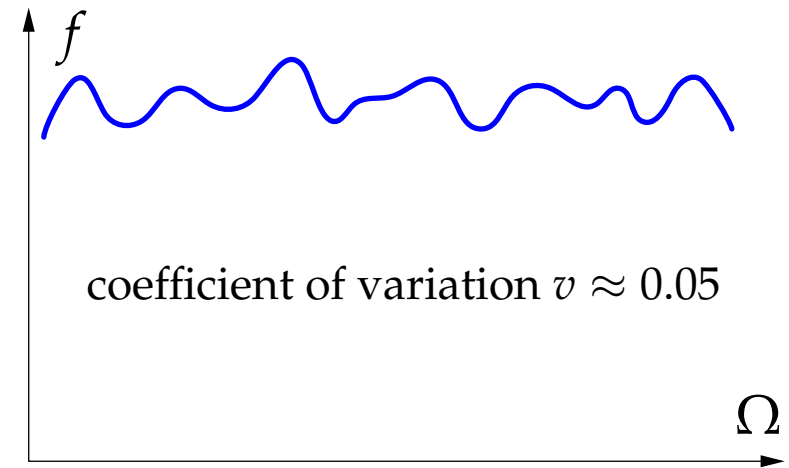
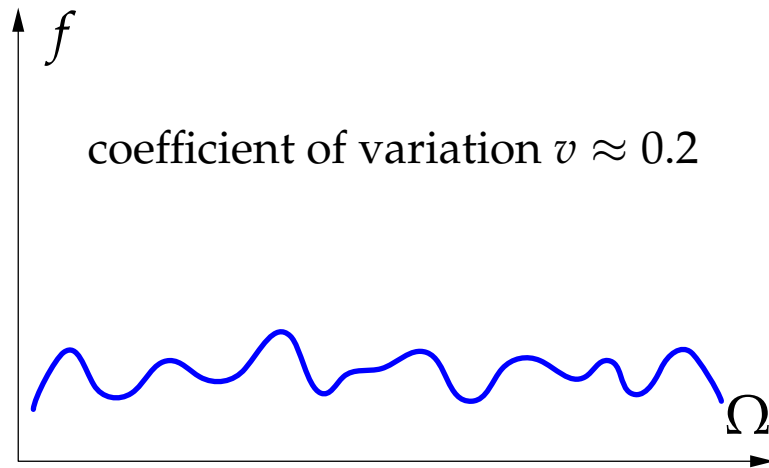
- Consider the **coefficient of variation** of the fitness function

$$v = \frac{\sigma_f}{\mu_f} = \frac{\sqrt{\frac{1}{|\Omega|-1} \sum_{\omega' \in \Omega} (f(\omega') - \frac{1}{|\Omega|} \sum_{\omega \in \Omega} f(\omega))^2}}{\frac{1}{|\Omega|} \sum_{\omega \in \Omega} f(\omega)} \quad \text{or} \quad v(t) = \frac{\sigma_f(t)}{\mu_f(t)}.$$

- Empirical investigations showed that a coefficient of variation of $v \approx 0.1$ provides a good balance of exploration and exploitation.
- If the coefficient of variation v deviates from this value, it is tried to achieve this value by scaling the fitness function f (e.g. by scaling or exponentiation).
- For a practical computation of the coefficient of variation v , the search space Ω is replaced by the current population $\text{pop}(t)$ (that is, v is not computable, but can merely be estimated).
- Then in each generation the value of $v(t)$ is computed and the fitness values are adapted accordingly (σ -scaling with $\beta = \frac{1}{v^*}$, $v^* = 0.1$).

Selection: Adaptation of the Fitness Function

Illustration of the Coefficient of Variation:



If the coefficient of variation is too large, premature convergence may result, if the coefficient of variation is too small, the selection pressure tends to vanish.

Experience indicates that $v \approx 0.1$ is a reasonable value.

Selection: Adaptation of the Fitness Function

- **Time-dependent Fitness Function:**

The relative fitness is not computed directly using the function $f(\omega)$ to optimize, but using $g(\omega) \equiv (f(\omega))^{k(t)}$.

The time-dependent exponent $k(t)$ controls the selection pressure.

- **Procedure to determine $k(t)$** [Michalewicz 1996]:
(is intended to keep the coefficient of variation v in the vicinity of $v^* \approx 0.1$)

$$k(t) = \left(\frac{v^*}{v}\right)^{\beta_1} \left(\tan\left(\frac{t}{T+1} \cdot \frac{\pi}{2}\right)\right)^{\beta_2 \left(\frac{v}{v^*}\right)^\alpha}$$

$v^*, \beta_1, \beta_2, \alpha$ parameter of the method

v coefficient of variation (e.g. estimated from initial population)

T maximum number of generations to compute

t current time step (generation number/index)

Recommendation: $v^* = 0.1, \beta_1 = 0.05, \beta_2 = 0.1, \alpha = 0.1$

Selection: Adaptation of the Fitness Function

- **Boltzmann Selection** (a different time-dependent fitness function):

The relative fitness is not computed directly

using the function $f(\omega)$ to optimize, but using $g(\omega) \equiv \exp\left(\frac{f(\omega)}{kT}\right)$.

The time-dependent **temperature parameter** T controls the development of the selection pressure over time.
 k is a normalization constant.

For example, the temperature may fall linearly down to a predetermined maximum number of generations.

- The idea of this selection method is closely related to **simulated annealing**:
 - In early generations the temperature parameter is high, the relative differences between the (adapted) fitness values therefore small.
 - In later generations the temperature parameter is lowered, causing the differences in fitness to grow larger.
 - **Consequence:**
In the course of the generations the selection pressure is increased.

Roulette Wheel Selection: Variance Problem

- Individuals are selected in proportion to their fitness, but still randomly.
- There is no guarantee that good individuals are transferred to the next generation, not even for the best individual (unless this is explicitly ensured).
- In general: The number of descendants of an individual may differ considerably from its expected value (**high variance of the number of descendants**).

(Computation of the expected value: see exercises.)

- Very simply, though not necessarily recommendable solution:

Discretization of the range of fitness values

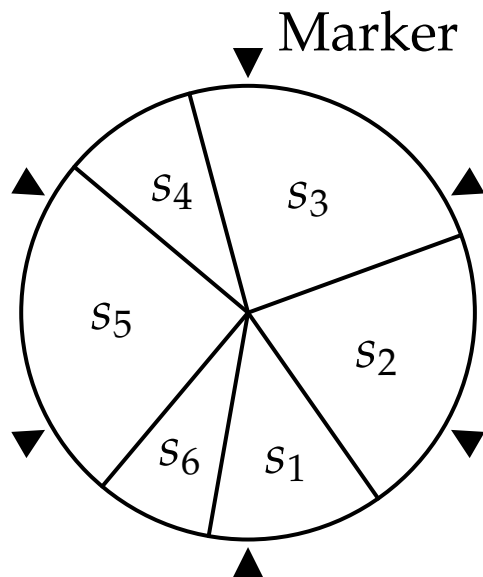
- Compute mean value $\mu_f(t)$ and standard deviation $\sigma_f(t)$ of the population. (Distinguish average, sub-average and super-average individuals.)
- If $\mu_f(t) - \sigma_f(t) > f(\omega)$: 0 children/descendants
- If $\mu_f(t) - \sigma_f(t) \leq f(\omega) \leq \mu_f(t) + \sigma_f(t)$: 1 child/descendant
- If $f(\omega) > \mu_f(t) + \sigma_f(t)$: 2 children/descendants

Selection: Expected Value Model

Possible solution of the variance problem: **Expected Value Model**

- Create $\lfloor f_{\text{rel}}(\omega) \cdot \text{popsize} \rfloor$ copies (descendants) of each individual s .
- Complete the intermediary population with roulette wheel selection.

Alternative: **Stochastic Universal Sampling**



Selection of the Intermediary Population:

- Turn the roulette wheel once.
- Choose one chromosome per marker.
- Here: $1 \times s_1, 1 \times s_2, 2 \times s_3, 2 \times s_5$.
- Individuals that are better than the average are guaranteed at least one spot in the intermediary population.

Selection: Expected Value Model

Variants of the Expected Value Model:

Generate the remaining individuals of the intermediary population by

- Procedures that are known from **election evaluation** (distribution of seats/mandates, e.g., largest remainders, Hare-Niemeyer/Hamilton/Vinton, d'Hondt etc.)
- **Roulette Wheel Selection**, but:
 - The fitness of every individual that receives a child/descendant is reduced by a certain amount Δf .
 - If this renders the fitness of an individual negative, it does not receive another descendant.
 - Apart from this, the selection is a simple roulette wheel selection.
 - Principle for choosing Δf : The best individual should receive only a certain maximum number k of descendants:

$$\Delta f = \frac{1}{k} \max\{f(\omega) \mid \omega \in \text{pop}(t)\}.$$

Selection: Rank-based Selection

- The individuals are sorted according to descending fitness. In this way each individual is assigned a **rank** in the population. (Idea from statistics: non-parametric methods, e.g. rank correlation)
- On this rank scale some probability distribution is defined: The smaller the rank, the larger the probability.
- Then roulette wheel selection is executed based on this probability distribution.
- **Advantages:**
 - The dominance problem can be avoided to a large degree, because the value of the fitness function does not directly determine the selection probability (but only indirectly via the ranks — not fitness proportional).
 - By choosing the probability distribution on the rank scale, the selection pressure can be controlled in a convenient manner.
- **Disadvantage:**
 - The individuals need to be sorted by fitness (cost: $\text{popsize} \cdot \log \text{popsize}$).

Selection: Tournament Selection

- In order to select an individual, k individuals, $2 \leq k < \text{popsize}$, are randomly chosen from the population (with or without replacement, but the selection does *not* take the fitness into account; k is the **tournament size**).
- The individuals compete in a tournament, which is won by the best individual. The tournament winner receives a descendant in the intermediary population.
- After the tournament *all* participants of the tournament (including the winner) are returned to the current population.
- **Advantages:**
 - The dominance problem can be avoided to a large degree, since the value of the fitness function does not directly determine the selection probability (but only indirectly via the tournament — not fitness proportional).
 - Selection pressure can be controlled conveniently by the tournament size.
- **Modification:** Instead of always letting the best individual win the tournament, the relative fitness of the tournament participants determines their chances of winning (roulette wheel selection among the tournament participants).

Selection: Elitism

- Only in the expected value model (or one of its variants) it is ensured that the best individual enters the intermediary population (with at least one copy/descendant).
- However, even if the best individual enters the intermediary population, it is not protected against modification by genetic operators. (This holds for the expected value model as well.)
- **Therefore:** The fitness of the best individual may as well decrease again from one generation to the next. (Naturally, this is not desirable.)
- **Solution: Elitism**
The best individual (or the k , $1 \leq k < \text{popsize}$, best individuals) are copied *without modification* into the next generation. (The *elite* of a population is preserved, hence the term *elitism*.)
- **Note:** The elite is *not* excluded from standard selection, since it may still be improved by genetic operators.

Selection: Elitism

- Most of the time descendants (mutation or crossover products) simply replace their parents (parents are discarded, descendants enter next generation).
- **“Local” Elitism** (Elitism between parents and their children)
 - *mutation*: A mutated individual only replaces its parent if it is at least as good as the parent (equal or better fitness).
 - *crossover*: The four individuals that are considered during a crossover operation (two parents and two children) are sorted by fitness. The best two individuals are transferred into the next generation.
- **Advantage**
 - Better convergence characteristics, since the local optimum is approached in a more consistent manner.
- **Disadvantage**
 - Relatively large danger of getting stuck in a local optimum, since no (local) deterioration is possible.

Selection: Niche Techniques

Objective of Niche Techniques: **Explicit Prevention of Crowding**

(Crowding: The population consists of identical or very similar individuals.)

- **Deterministic Crowding**

- *Idea:* Generated descendants replace those individuals of the population most similar to them. → the search space is locally less densely populated
- *Required:* a similarity or distance measure for the individuals (if chromosomes are binary, this may be, e.g., the Hamming distance)

- **Variant of Deterministic Crowding**

- During crossover two pairs of individuals, each consisting of one parent and one child, are formed by assigning each child to that parent that is most similar to it (breaking ties arbitrarily).
- From each pair one individual enters the intermediary population.
- *Advantage:* Considerably fewer comparisons are needed, since only few individuals are considered instead of the whole population.

Selection: Niche Techniques

- **Sharing**

- *Idea:* The fitness of an individual is reduced, if there are other (many) individuals in its vicinity.
Intuitively: **The individuals share the fitness of a niche.**
- *Required:* a similarity or distance measure for the individuals (if chromosomes are binary, this may be, e.g., the Hamming distance)
- *Example:*

$$f_{\text{share}}(\omega) = \frac{f(\omega)}{\sum_{\omega' \in \text{pop}(t)} g(d(\omega, \omega'))}$$

d : distance measure for the individuals

g : weight function that defines the shape and size of the niche, e.g. so-called **power law sharing**:

$$g(x) = \begin{cases} 1 - \left(\frac{x}{\varrho}\right)^\alpha, & \text{if } x < \varrho, \\ 0, & \text{otherwise.} \end{cases}$$

ϱ : niche radius, α : controls the influence strength within a niche

Selection Methods: Characterization

Selection methods are often characterized by the following pairs of notions:
(most of which are intuitive)

static The selection probabilities are constant.

dynamic The selection probabilities change over time.

extinctive The selection probabilities (of some individuals) may be 0.

preservative All selection probabilities must be positive.

pure Individuals may have children only in one generation.

impure Individuals may have children in multiple generations.

right All individuals of a population may have children.

left The best individuals of a population may *not*
have children (in order to avoid premature convergence).

generational The set of parents is fixed until all descendants have been created.

on the fly Generated descendants immediately replace their parents.

Elements of Evolutionary Algorithms

3. Genetic Operators

Genetic Operators

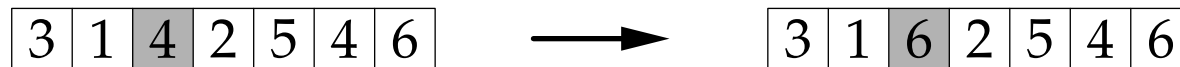
- Reminder: A part of the selected individuals (intermediary population) is subjected to **genetic operators**, in order to create variants and recombinations of the existing solution candidates.
- General classification of genetic operators based on the number of parents:
 - One-parent Operators (“Mutation”, “Variation”)
 - Two-parent Operators (“Crossover”, “Recombination”)
 - Multi-parent Operators (“Recombination”)
- Depending on the encoding, the genetic operators may have to possess certain characteristics, for example:
 - If the solution candidates are represented as permutations, the genetic operators should be permutation preserving.
 - In general: If certain combinations of alleles are nonsensical, the genetic operators should not create them (if possible).

Genetic One-parent Operators I

Genetic one-parent operators are usually called **Mutation Operators**.

- **Standard Mutation**

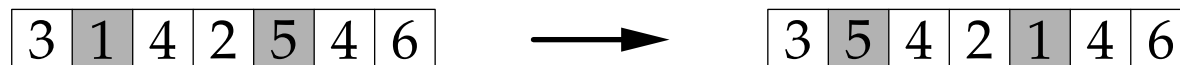
Replacing the current allele of a gene by another.



- If desired, multiple genes may be mutated (see the n -queens problem).
- *Parameter*: Mutation probability p_m , $0 < p_m \ll 1$
For strings of bits the value $p_m = \frac{1}{\text{length}(\omega)}$ is close to optimal.

- **Pair Exchange**

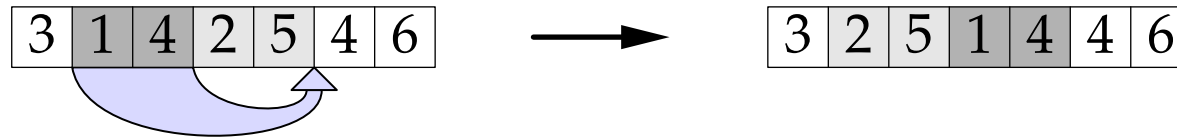
Exchange of the alleles of two genes of a chromosome.



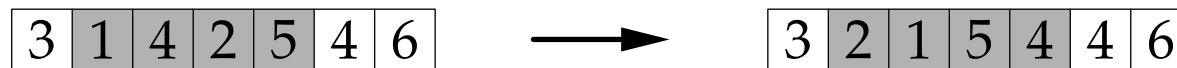
- *Presupposition*: same set of alleles for the two exchanged genes.
- *Generalization*: cyclic permutation of $3, 4, \dots, k$ genes.

Genetic One-parent Operators II

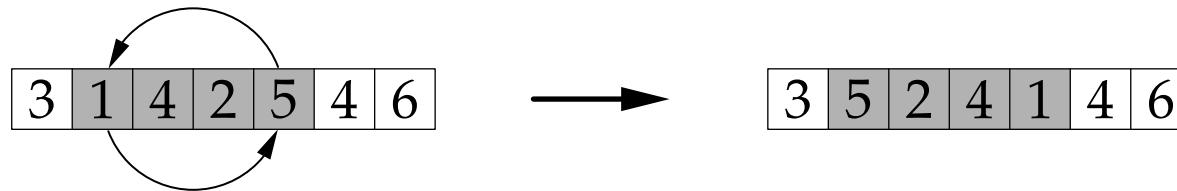
- **Moving a Part of a Chromosome**



- **Shuffling/Permuting a Part of a Chromosome**



- **Inversion (Reversing a Part of a Chromosome)**



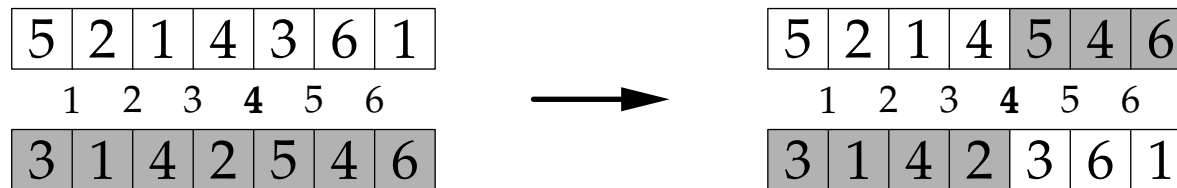
- *Presupposition:* same set of alleles of all affected genes.
- *Parameter:* possibly a probability distribution on lengths (and shift distance for moving a chromosome part).

Genetic Two-parent Operators I

Genetic Two-parent operators are generally called **Crossover**.

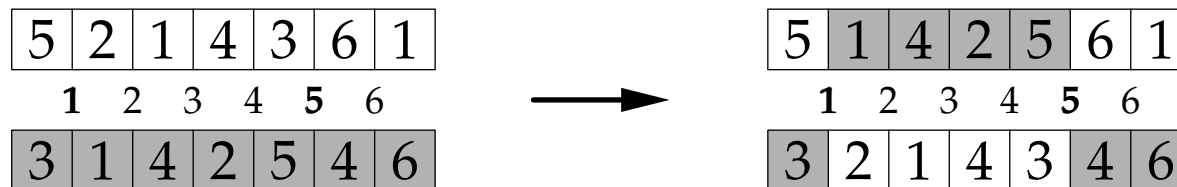
- **One-Point Crossover**

- A random cut point is determined.
- The gene sequence on one side of the cut point are exchanged.



- **Two-point Crossover**

- Two random cut points are determined.
- The gene sequences between the two cut points are exchanged.



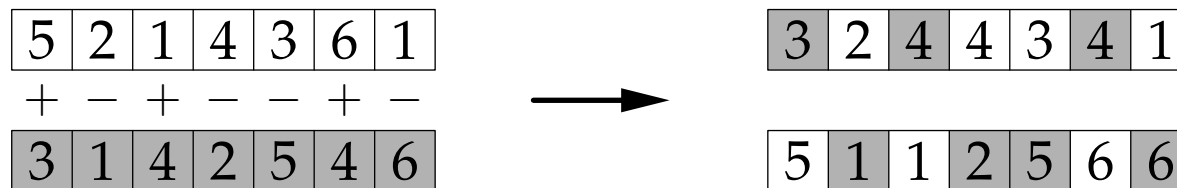
Genetic Two-parent Operators II

- **n-point Crossover**

- Generalization of one-point and two-point Crossover
- n random cut points are determined.
- Between the cut points, gene sequences are alternately exchanged and preserved (e.g., preserved before first, exchanged between 2nd and 3rd, etc.).

- **Uniform Crossover**

- For each gene it is determined independently whether it is exchanged or not (+: yes, -: no, *parameter*: probability p_x of an exchange).

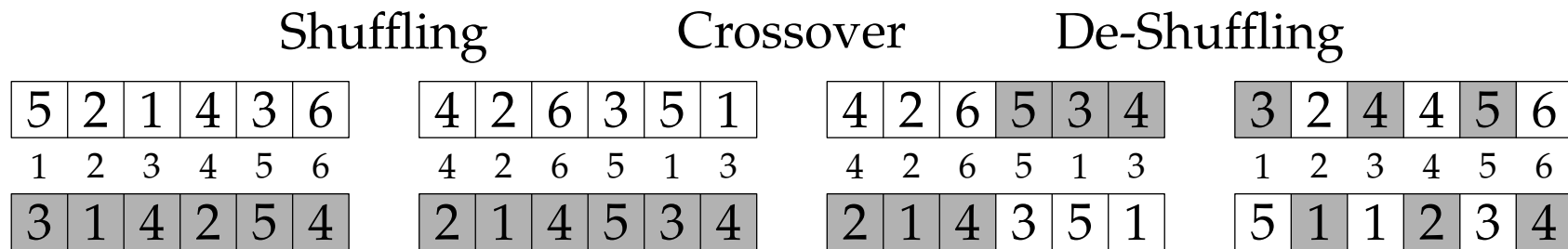


- **Note:** Uniform Crossover is **not** equivalent to $(\text{length}(\omega) - 1)$ -point Crossover! The number of crossover points is chosen randomly. (For n -point crossover it is fixed and predetermined.)

Genetic Two-parent Operators III

- **Shuffle Crossover**

- Before one-point crossover is applied, the genes are shuffled randomly, and after it has been applied, the genes are returned to their original position (de-shuffling of the genes).



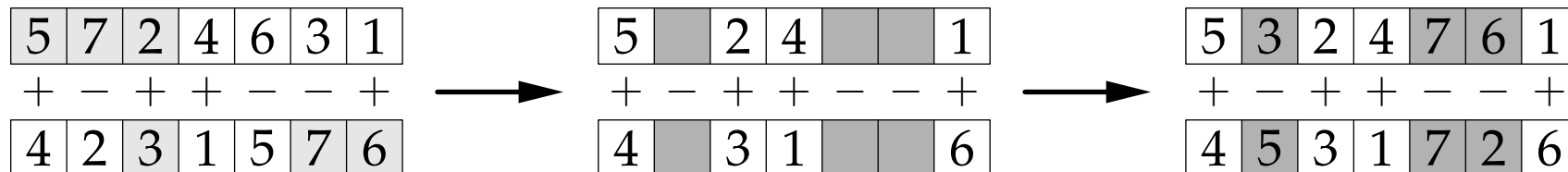
- **Note:**
Shuffle crossover is **not** equivalent to uniform crossover!
- With shuffle crossover any possible number of genes that may be exchanged between chromosomes has the same probability, with uniform crossover this number is binomially distributed with parameter p_x .
- Shuffle crossover is among the most recommendable crossover operators.

Genetic Two-parent Operators IV

Permutation Preserving Crossover Operators

- **Uniform Order-based Crossover**

- In analogy to standard uniform crossover it is decided for each gene individually, whether it is preserved or not (+: yes, -: no, *parameter*: probability p_k for preservation).
- The gaps are filled by the missing alleles, respecting the order of these alleles in the *other* parent chromosome.



- This procedure essentially preserved **order information**.
- *Alternative*: In one chromosome the genes marked with “+”, in the other chromosome those marked with “-” are preserved.

Genetic Two-parent Operators V

Permutation Preserving Crossover Operators

- **Edge Recombination** (specially designed for the traveling salesman problem)
 - A chromosome is interpreted as a graph (specifically as a chain or ring): Each gene possesses edges to its neighbors in the chromosome.
 - The edges of the graphs of two chromosomes are mixed, hence the name *edge recombination*.
 - This procedure preserves **neighborhood information** (i.e., information about the alleles to the left and right).
- **Procedure:**
 1. *Creating an Edge Table:*
 - For each allele its neighbors (in both parents) are listed. (The first and last gene of a chromosome may be considered neighbors.)
 - If an allele has the same neighbors in both parents (ignoring whether on the left or on the right), then this neighbor is listed only once, but marked.

Two-parent Operators: Edge Recombination

- **Procedure:**

- 2. *Creating a Child/Descendant:*

- The first allele is chosen randomly from one the parents.
 - A selected allele is deleted from the edge table (that is, from the list of neighbors of the alleles).
 - Each subsequent allele is chosen from the undeleted neighbors of the preceding allele, respecting the following **precedence list**:
 - a) marked neighbors (that is, neighbors that occur twice)
 - b) neighbors with shortest neighbor list (with marked neighbors counting only once)
 - c) random selection of a neighbor
 - A second child/descendant may be created in an analogous fashion from the second parent. However, this is usually not done.

Two-parent Operators: Edge Recombination

Example: **A:**

6	3	1	5	2	7	4
---	---	---	---	---	---	---

B:

3	7	2	5	6	1	4
---	---	---	---	---	---	---

Creating the Edge Table

allele	neighbors		joined together
	in A	in B	
1	3, 5	6, 4	3, 4, 5, 6
2	5, 7	7, 5	5*, 7*
3	6, 1	4, 7	1, 4, 6, 7
4	7, 6	1, 3	1, 3, 6, 7
5	1, 2	2, 6	1, 2*, 6
6	4, 3	5, 1	1, 3, 4, 5
7	2, 4	3, 2	2*, 3, 4

- Both chromosomes are seen as a ring (first and last gene are neighbors):
In **A** the left neighbor of 6 is 4, the right neighbor of 4 is 6; analogously in **B**.
- In both chromosomes the 5, 2 and 7 are side by side — this should be preserved (this also shows in the markings).

Two-parent Operators: Edge Recombination

Creating a Descendant

6	5	2	7	4	3	1
---	---	---	---	---	---	---

Allele	Neighbors	Choice: 6	5	2	7	4	3	1
1	3, 4, 5, 6	3, 4, 5	3, 4	3, 4	3, 4	3		
2	5*, 7*	5*, 7*	7*	7*	—	—	—	—
3	1, 4, 6, 7	1, 4, 7	1, 4, 7	1, 4, 7	1, 4	1	1	—
4	1, 3, 6, 7	1, 3, 7	1, 3, 7	1, 3, 7	1, 3	1, 3	—	—
5	1, 2*, 6	1, 2*	1, 2*	—	—	—	—	—
6	1, 3, 4, 5	1, 3, 4, 5	—	—	—	—	—	—
7	2*, 3, 4	2*, 3, 4	2*, 3, 4	3, 4	3, 4	—	—	—

- Start with the first allele of the chromosome **A**, that is, with 6, and then delete the 6 from all neighbor lists (third column).
- Since among the neighbors of 6 (those are 1, 3, 4, 5) the 5 has the shortest neighbor list, the 5 is chosen for the next gene. Then 2 follows, then 7 etc.

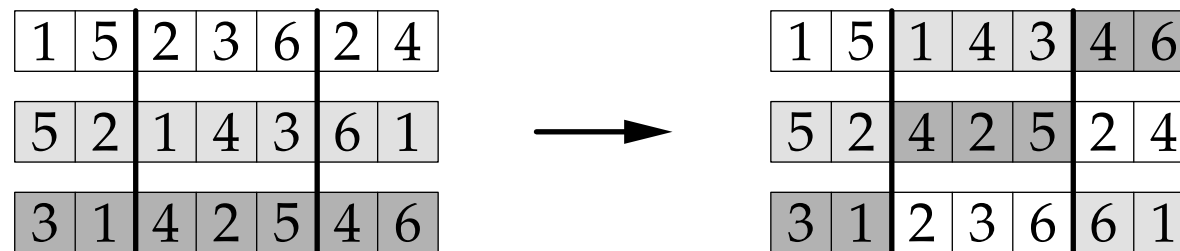
Two-parent Operators: Edge Recombination

- The child/descendant usually has one new edge (from the last to the first gene).
- Edge recombination may also be used if the first and last gene of a chromosome are not seen as neighbors:
The corresponding edges are simply not entered into the edge table.
- If the first and the last gene are seen as neighbors, the initial allele may be chosen from anywhere in the chromosome. However, if they are not seen as neighbors, the initial allele has to be one that is at the beginning or at the end of a chromosome.
- When creating a child/descendant, it may happen that the neighbor list of the allele just chosen is empty.
(The precedence rules serve the purpose to keep the probability low that this happens; however, they are not perfect.)
In this case the construction continues with an allele that is chosen randomly from the remaining alleles.
Note that this creates a new edge (not present in the parents).

Genetic Three- and Multi-Parent Operators

- **Diagonal Crossover**

- Similar to One-, Two- and n -Point Crossover, but for more parents.
- With three parents, two crossover points are chosen.
- At the cut points the gene sequences are diagonally and cyclically shifted across the chromosomes.



- A generalization to more than three parents suggests itself immediately. For k parents $k - 1$ crossover points are chosen.
- This procedure leads to a very good exploration of the search space, especially for a large number of parents (10–15 parents).

Characterization of Crossover Operators

- **Positional Bias**

- Positional bias is present if the probability that two genes are inherited / passed down together (stay together in the same chromosome or pass together into the other chromosome) depends on their relative position.
- Positional bias is not desirable, because then the order of the genes in a chromosome may have a decisive influence on the success or failure of the evolutionary algorithm (certain arrangements are more difficult to achieve).

- **Example: One-Point Crossover**

- Two genes are separated from each other (end up in different descendants) if the crossover point falls between them.
- The closer two genes are to each other in a chromosome, the fewer crossover points exist between them.
- *Therefore:* Genes that are located next to each other have a higher probability to end up in the same child/descendant than genes that are located far apart from each other.

Characterization of Crossover Operators

- **Distributional Bias**

- Distributional bias is present if the probability that a certain number of genes is exchanged between the chromosomes is not the same for all numbers.
- Distributional bias is often undesired, since then partial solutions of different size have different chances of getting into the next generation.
- Distributional bias is usually less critical/harmful (and thus more easily tolerable) than positional bias.

- **Example: Uniform Crossover**

- Since every gene is exchanged independently of all others with the probability p_x , the number k of exchanged genes is binomially distributed with the parameter p_x :

$$P(K = k) = \binom{n}{k} p_x^k (1 - p_x)^{n-k} \quad \text{where} \quad n \hat{=} \text{total number of genes.}$$

- *Therefore:* Very small and very large numbers are less probable, $n \cdot p_x$ exchanges are most probable.

Theoretical Consideration: Schema Theorem

Schema Theorem

- Question: **Why do evolutionary algorithms work?**
- **Approach to a Solution** by [Holland 1975]:
Consider chromosome schemata (that is, only partially fixed chromosomes) and examine how the number of chromosomes that fit a schema develop over time (over the generations),
- **Objective:** A rough statement, using mainly stochastic means, how evolutionary algorithms explore the search space.
- **Simplification** of the exposition: Restriction to
 - bit sequences (chromosomes consisting of zeros and ones) with a fixed length L
 - fitness proportional selection (roulette wheel selection)
 - standard mutation (changing/flipping a randomly chosen bit)
 - one-point crossover (cutting the chromosomes at some point and exchanging the genes on one side)

Schema Theorem: Schemata

- **Definition: Schema**

A **Schema** h is a character sequence of length L over an alphabet $\{0, 1, *\}$, that is, $h \in \{0, 1, *\}^L$.

The character $*$ is called **Joker Character** or **Don't Care Symbol**.

- **Definition: Fit**

A chromosome $c \in \{0, 1\}^L$ **fits a schema** $h \in \{0, 1, *\}^L$, written: $c \triangleleft h$, if it coincides with h at all positions, at which h contains a 0 or a 1. (Positions that contain a $*$ are not considered.)

- **Example:** $h = \quad **0*11*10*$ schema of length 10
 $c_1 = \quad 1100111100$ fits h , that is, $c_1 \triangleleft h$
 $c_2 = \quad 1111111111$ does not fit h , that is, $c_2 \not\triangleleft h$

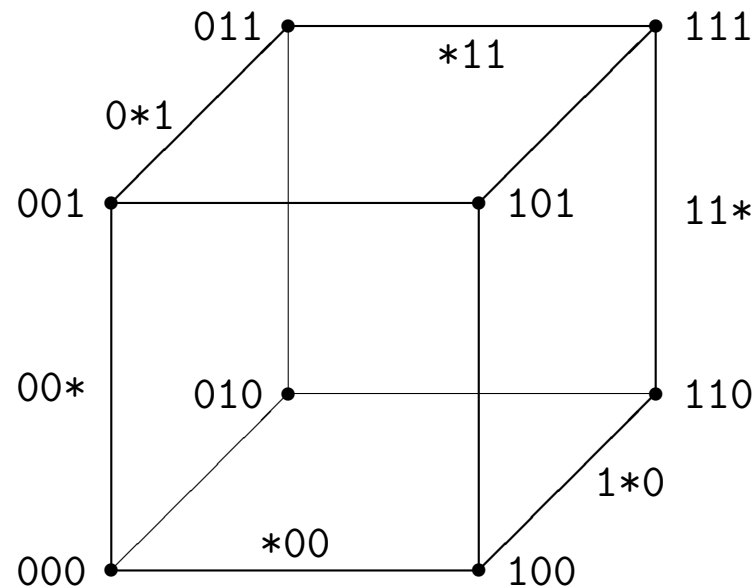
- There are 2^L chromosomes and 3^L schemata.

Each chromosome fits $\sum_{i=0}^L \binom{L}{i} = 2^L$ schemata.

The number of chromosomes that fit a schema depends on the number of $*$ s.

Schemata: Hyperplanes

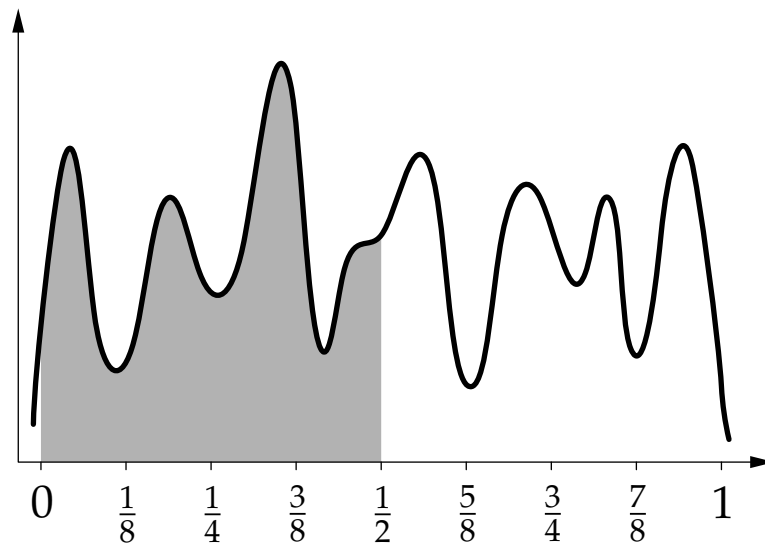
- Each schema describes a hyperplane in a unit hypercube (though only (hyper-)planes that are parallel or perpendicular to coordinate axes).



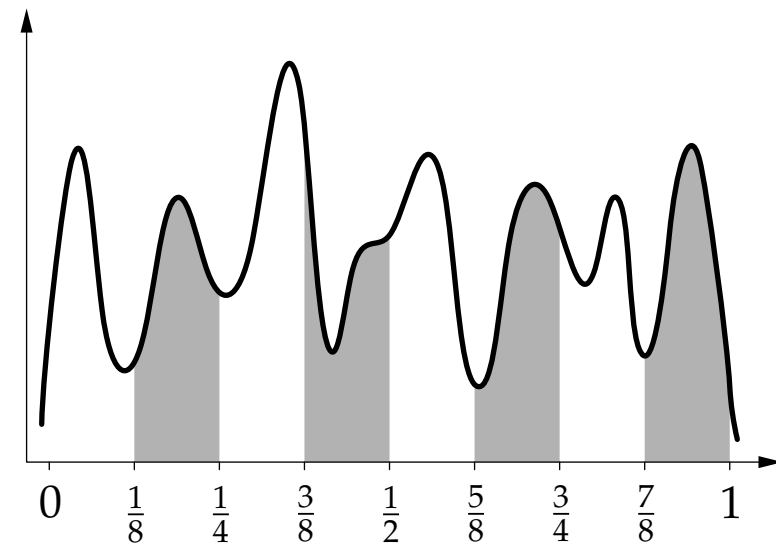
- Examples:**
 - $*00 \hat{=}$ edge from 000 to 100 (front bottom)
 - $1*0 \hat{=}$ edge from 100 to 110 (right bottom)
 - $0** \hat{=}$ left cube face
 - $**1 \hat{=}$ top cube face
 - $*** \hat{=}$ whole cube

Schemata: Domains of Functions

- Given: real-valued function $f : [0, 1] \rightarrow \mathbb{R}$
- Assumption: Binary encoding of the function's argument (no Gray code).
- Each schema corresponds to a “stripe pattern” in the domain of f :



schema 0**...*



schema **1*...*

- Schemata with Gray coding: see exercises

Schema Theorem: Influence of Selection

- **Objective:** Trace how chromosomes fitting a schema develop.
- In order to trace how chromosomes that fit a certain schema spread in a population, we have to examine what effects are caused by **selection** and the **genetic operators** (mutation and crossover).
- For the selection we have to examine what fitness chromosomes have that fit a schema h . Approach: Averaging over all chromosomes.

- **Definition: Average Fitness**

The **average relative fitness** of the chromosomes that fit schema h in a generation $\text{pop}(t)$ is

$$f_{\text{rel}}(h) = \frac{\sum_{c \in \text{pop}(t), c \triangleleft h} f_{\text{rel}}(c)}{|\{c \in \text{pop}(t) \mid c \triangleleft h\}|}$$

- The average number of children of a chromosome that fit a schema h is $f_{\text{rel}}(h) \cdot \text{popsize}$.
 \Rightarrow The expected number of chromosomes that fit a schema h after selection is:
(number of chromosomes fitting before selection) $\cdot f_{\text{rel}}(h) \cdot \text{popsize}$.

Schema Theorem: Influence of Selection

- Other considerations concerning the relative fitness of a schemata:

$$\begin{aligned}
 f_{\text{rel}}(h) \cdot \text{popsize} &= \frac{\sum_{c \in \text{pop}(t), c \triangleleft h} f_{\text{rel}}(c)}{|\{c \in \text{pop}(t) \mid c \triangleleft h\}|} \cdot \text{popsize} \\
 &= \frac{\sum_{c \in \text{pop}(t), c \triangleleft h} \frac{f(c)}{\sum_{c' \in \text{pop}(t)} f(c')}}{|\{c \in \text{pop}(t) \mid c \triangleleft h\}|} \cdot \text{popsize} \\
 &= \frac{\sum_{c \in \text{pop}(t), c \triangleleft h} f(c)}{|\{c \in \text{pop}(t) \mid c \triangleleft h\}|} = \frac{\overline{f_t(h)}}{\overline{f_t}}
 \end{aligned}$$

$\overline{f_t(h)}$ average fitness of the chromosomes that fit schema h in the t -th generation

$\overline{f_t}$ average fitness of all chromosomes of the t -th generation

- The average number of descendants thus can be expressed as the ratio of the average fitness of a schema to the overall average fitness (**fitness ratio**).

Schema Theorem: Influence of Mutation

- For the genetic operators we need measures with which we can state the probability that by applying them the fit to a schema is preserved or lost.
- We start with the influence of **(standard) mutation**.

- **Definition: Order** (for standard mutation, changing one bit)

The **Order** of a schema h is the number of zeros and ones in h , that is $\text{ord}(h) = \#0 + \#1 = L - \#*$ ($\#$: number of occurrences).

Example: $\text{ord}(**0*11*10*) = 5$.

- A standard mutation destroys the fit to a schema with probability $p_m^*(h) = \frac{\text{ord}(h)}{L}$, if the chosen bit is flipped ($0 \leftrightarrow 1$),
probability $p_m^*(h) = \frac{\text{ord}(h)}{2L}$, if the new value of the bit is chosen randomly.
- **Explanation:** Each of the L genes of a chromosome of length L is chosen to be mutated with the same probability.
Changing a bit where a schema h is $*$ cannot destroy the fit.

Schema Theorem: Influence of Crossover

- Consider now the influence of **One-Point Crossover**.

- **Definition: Defining Length** (for one-point crossover)

The **defining length** of a schema h is the difference between the position number of the last 0/1 and the position number of the first 0/1 in h .

Example: $dl(**0*11*10*) = 9 - 3 = 6$.

- The probability that the cut point of one-point crossover falls in such a way that two non-joker characters are separated from each other is $p_c^*(h) = \frac{dl(h)}{L-1}$.
- **Explanation:** For chromosomes of length L there are $L - 1$ possible cut points for one-point crossover, all of which are equally likely.

$dl(h)$ of these cut points lie in such a way that the genes that are fixed in the schema h (i.e., that are 0 or 1) end up in different descendants, which may destroy a fit.

- **Attention:** The fit *may* be destroyed, but this is not a necessary consequence.
→ for the later computation additional considerations are needed (later).

Schema Theorem: Definitions

- Reminder: Our objective is a rough statement, using mainly stochastic means, how evolutionary algorithms explore the search space.
- **Expected Value of the Number of Fitting Chromosomes**
 $N(h, t)$ is the expected value of the number of chromosomes that fit schema h in the t -th generation.
- **Expected Value after Selection**
 $N(h, t + \Delta t_s)$ is the expected value of chromosomes that fit schema h in the t -th generation after selection.
- **Expected Value after Crossover**
 $N(h, t + \Delta t_s + \Delta t_c)$ is the expected value of chromosomes that fit schema h in the t -th generation after selection and crossover.
- **Expected Value after Mutation**
 $N(h, t + \Delta t_s + \Delta t_c + \Delta t_m) = N(h, t + 1)$ is the expected value of chromosomes that fit schema h in the t -th generation after selection, crossover and mutation (and thus fit schema h in the $(t + 1)$ -th generation).

Schema Theorem: Selection

- **Desired:** A mathematical relationship between $N(h, t)$ and $N(h, t + 1)$.
- **Procedure:** We consider, step by step, the effects of selection, crossover and mutation with the help of the three measures introduced above, namely the *average fitness*, the *order* and the *defining length* of a schema.

- **Selection:**

The effects of selection are captured by the average fitness (since we are using fitness proportional selection):

$$N(h, t + \Delta t_s) = N(h, t) \cdot f_{\text{rel}}(h) \cdot \text{popsize}$$

$N(h, t) \cdot f_{\text{rel}}(h)$ probability that a chromosome is chosen that fits schema h ,

$f_{\text{rel}}(h) \cdot \text{popsize}$ average number of descendants of a chromosome that fits schema h .

- **Note:** The relative fitness $f_{\text{rel}}(h)$ is not exactly determined, because the number of chromosomes that fit schema h is only known as an expected value.

Schema Theorem: Crossover

- **Crossover:**

The effects of crossover are captured by:

$$N(h, t + \Delta t_s + \Delta t_c) = \underbrace{(1 - p_c) \cdot N(h, t + \Delta t_s)}_A + \underbrace{p_c \cdot N(h, t + \Delta t_s) \cdot (1 - p_{\text{loss}})}_B + C$$

p_c probability that a chromosome is subjected to crossover

p_{loss} probability that the fit of a chromosome to schema h is lost by applying a crossover operation to it

A expected value of the number of chromosomes that fit schema h and *do not* participate in a crossover operation

B expected value of the number of chromosomes that participate in a crossover and whose fit to schema h is *not* destroyed by this operation

C gains in the number of chromosomes that fit schema h by ...
(see exercises)

Schema Theorem: Crossover

Considerations concerning the probability p_{loss} :

- Examples:**

$h =$	$**0* 1*1*$		$**0*1*1*$	$= h$
$h \triangleright c_1 =$	$0000 1111$	\rightarrow	00000000	$= c'_1 \not\triangleleft h$
$h \not\triangleright c_2 =$	$1111 0000$	\rightarrow	11111111	$= c'_2 \not\triangleleft h$
$h =$	$**0* 1*1*$		$**0*1*1*$	$= h$
$h \triangleright c_1 =$	$0000 1111$	\rightarrow	00001010	$= c'_1 \triangleleft h$
$h \triangleright c_2 =$	$1101 1010$	\rightarrow	11011111	$= c'_2 \triangleleft h$

- Therefore:**

$$p_{\text{loss}} \leq \underbrace{\frac{dl(h)}{L-1}}_{A=p_c^*(h)} \cdot \left(1 - \underbrace{\frac{N(h, t + \Delta t_s)}{\text{popsize}}}_B \right)$$

A probability that the cut point falls between fixed genes

B probability that the second chromosome fits schema h

- Question:** Why do we have only \leq and not $=$? (see exercises)

Schema Theorem: Crossover

- Inserting the expression for p_{loss} yields:

$$\begin{aligned}
 & N(h, t + \Delta t_s + \Delta t_c) \\
 & \geq (1 - p_c) \cdot N(h, t + \Delta t_s) \\
 & \quad + p_c \cdot N(h, t + \Delta t_s) \cdot \left(1 - \frac{dl(h)}{L-1} \cdot \left(1 - \frac{N(h, t + \Delta t_s)}{\text{popsize}} \right) \right) \\
 & = N(h, t + \Delta t_s) \left(1 - p_c + p_c \cdot \left(1 - \frac{dl(h)}{L-1} \cdot \left(1 - \frac{N(h, t + \Delta t_s)}{\text{popsize}} \right) \right) \right) \\
 & = N(h, t + \Delta t_s) \cdot \left(1 - p_c \frac{dl(h)}{L-1} \cdot \left(1 - \frac{N(h, t + \Delta t_s)}{\text{popsize}} \right) \right) \\
 & \stackrel{(*)}{=} N(h, t) \cdot f_{\text{rel}}(h) \cdot \text{popsize} \cdot \left(1 - p_c \frac{dl(h)}{L-1} \cdot (1 - N(h, t) \cdot f_{\text{rel}}(h)) \right)
 \end{aligned}$$

- In the step (*) we used twice the relation

$$N(h, t + \Delta t_s) = N(h, t) \cdot f_{\text{rel}}(h) \cdot \text{popsize}$$

that we derived above.

Schema Theorem: Mutation

- **Mutation:**

The effects of mutation are captured by the order:

$$N(h, t + 1) = N(h, t + \Delta t_s + \Delta t_c + \Delta t_m) = N(h, t + \Delta t_s + \Delta t_c) \cdot (1 - p_m)^{\text{ord}(h)}$$

p_m mutation probability for each bit, that is,
each bit is mutated (flipped) with probability p_m
and remains unchanged with probability $(1 - p_m)$.

Explanation: In order for the fit to the schema h to be preserved,
none of the $\text{ord}(h)$ genes may be changed that are fixed (0 or 1) in schema h .

- **Alternative Models** are possible, for example:

Each chromosome is subjected to exactly one bit mutation.

$$N(h, t + 1) = N(h, t + \Delta t_s + \Delta t_c + \Delta t_m) = N(h, t + \Delta t_s + \Delta t_c) \cdot \left(1 - \frac{\text{ord}(h)}{L}\right)$$

$\frac{\text{ord}(h)}{L}$ probability that the bit mutation modifies a gene that is fixed (0 or 1)
in schema h (assumption: same probability for each bit to be chosen).

Schema Theorem

- In total (using the first mutation model):

$$N(h, t + 1) = f_{\text{rel}}(h) \cdot \text{popsize} \cdot \left(1 - p_c \frac{dl(h)}{L - 1} \cdot (1 - N(h, t) \cdot f_{\text{rel}}(h)) \right) \cdot (1 - p_m)^{\text{ord}(h)} \cdot N(h, t)$$

- Inserting the fitness ratio yields the **Schema Theorem**:

$$N(h, t + 1) = \frac{\overline{f_t(h)}}{\overline{f_t}} \left(1 - p_c \frac{dl(h)}{L - 1} \left(1 - \frac{N(h, t)}{\text{popsize}} \cdot \frac{\overline{f_t(h)}}{\overline{f_t}} \right) \right) (1 - p_m)^{\text{ord}(h)} \cdot N(h, t)$$

- **Interpretation:** Schemata with

- above average evaluation,
- short defining length and
- low order

proliferate particularly heavily (roughly exponentially).

Schema Theorem: Building Block Hypothesis

- The Schema Theorem says that the search space is explored particularly well in such hyperplanes (regions) that correspond to schemata with high average fitness, short defining length and low order.

(Because in these regions the chromosomes proliferate most heavily.)

- Such schemata are called **Building Blocks**; the above statement is therefore also referred to as the **Building Block Hypothesis**.
- **Note:** This form of the building block hypothesis holds only for bit sequences, fitness proportional selection, standard mutation and one-point crossover.

(Because we restricted our considerations to such a setup.)

If, for example, other genetic operators are used, building blocks may be characterized by different properties (see exercises).

(However, high average fitness is always a property of building blocks, because all selection methods favor chromosomes with high fitness, although they favor them differently strongly and not always in direct relationship to the fitness.)

Schema Theorem: Criticism

- In a strict sense, the Schema Theorem only holds for infinite population size. (Otherwise there may be deviations from the considered expected values that may not be negligible.)

Clearly, this assumption cannot be met in practice.

Hence there are always deviations from the ideal behavior (*stochastic drift*).

- It is assumed implicitly that there are almost no or only very few interactions between genes (i.e. low *epistasy*), that is, that the fitness of chromosomes that fit a schema is very similar.

- A third objection that is put forward often is:

It is assumed implicitly that interacting genes are located close to each other in a chromosome in order to form building blocks that are as small as possible.

However, this objection only refers to the restriction to one point crossover and not the approach as such. This objection can be met by using a different crossover operation and a different measure that captures the effects of this crossover operation.

Theoretical Consideration: The Two-Armed Bandit Argument

The Two-Armed Bandit Argument

- **Claim:** The schema theorem implies that a genetic algorithm achieves a **near-optimal balance between exploration** of the search space **and exploitation** of good solution candidates.
- As an argument for this claim, [Holland 1975] used the **two-armed bandit** model as an analogy, that is, a slot machine with two independent arms.
- The two arms have different expected (per trial) payoffs μ_1 and μ_2 with variances σ_1^2 and σ_2^2 , respectively, all of which are unknown.
Without loss of generality we may assume $\mu_1 > \mu_2$, though (since we can always reorder/relabel the payoffs).
- It is also unknown, however, which of the two arms of the slot machine has the higher payoff (that is, it is unknown whether μ_1 is assigned to the left or to the right arm).
- Suppose we may play N games with such a slot machine.
What is the best strategy to maximize our winnings?

The Two-Armed Bandit Argument

- If we knew which arm has the greater payoff, we would simply use that arm for all N games, thus clearly maximizing our expected winnings.
- Since we do not know which arm is better, we must invest some trials into gathering information about which arm might be the one with the higher payoff.
 - For example, we may choose to use $2n$ trials, $2n < N$, for this task, in which we play both arms equally often (that is, we use each arm n times).
 - Afterward we evaluate which arm has given us the higher average payoff per trial (exploration).
 - In the remaining $N - 2n$ trials we then exclusively play the arm that has the higher observed payoff (exploitation).
- Our original question can thus be reformulated as: How should we choose n relative to N in order to maximize our (expected) winnings?
- In other words, how should we balance **exploration** (initial $2n$ trials) and **exploitation** (final $N - 2n$ trials)?

The Two-Armed Bandit Argument

- Maximizing (expected) winnings is equivalent to minimizing (expected) loss relative to always having chosen the better arm.
- Clearly, there are **two types of losses** involved here:
 - Loss during information gathering, when we play the worse arm n times (regardless of which arm is worse, since we play both arms equally often).
 - Loss due to the fact that we determine the better arm only based on an empirical payoff estimate, which may point to the wrong arm.
- The former refers to the fact that in the $2n$ trials we devote to exploration we necessarily lose

$$L_1(N, n) = n(\mu_1 - \mu_2),$$

because we do n trials with the arm with lower payoff μ_2 instead of using the arm with the higher payoff μ_1 .

- The loss from the remaining $N - 2n$ trials can only be given in stochastic terms.

The Two-Armed Bandit Argument

- The loss from the remaining $N - 2n$ trials can only be given in stochastic terms.
- Let p_n be the probability that the average payoffs per trial, as determined empirically in the first $2n$ trials, actually identify the correct arm.

(The index n of this probability is due to the fact that it obviously depends on the choice of $2n$: the larger $2n$, the higher the probability that the empirical payoff estimate from the $2n$ exploration trials identifies the correct arm.)

- That is, with probability p_n we use the arm actually having the higher payoff μ_1 for the remaining $N - 2n$ trials, while with probability $1 - p_n$ we use the arm actually having the lower payoff μ_2 in these trials.
- In the former case there is no additional loss (beyond what is incurred in the exploration phase), while in the latter case we lose

$$L_2(N, n) = (N - 2n)(\mu_1 - \mu_2)$$

in the exploitation phase, because we choose the wrong arm (that is, the one actually having the lower payoff μ_2).

The Two-Armed Bandit Argument

- Therefore the **expected total loss** is

$$\begin{aligned} L(N, n) &= \underbrace{L_1(N, n)}_{\text{exploration loss}} + (1 - p_n) \underbrace{L_2(N, n)}_{\text{incorrect exploitation loss}} \\ &= n(\mu_1 - \mu_2) + (1 - p_n)(N - 2n)(\mu_1 - \mu_2) \\ &= (\mu_1 - \mu_2)(np_n + (1 - p_n)(N - n)). \end{aligned}$$

- The final form nicely captures that in case the better arm is correctly identified (probability p_n), we lose winnings from n times using the worse arm in the exploration phase.
- However, in case the better arm is incorrectly identified (probability $1 - p_n$), we lose winnings from $N - n$ trials with the worse arm (n of which happen in the exploration phase and $N - 2n$ happen in the exploitation phase).
- We now have to **minimize the loss function** $L(N, n)$ **w.r.t.** n .

The Two-Armed Bandit Argument

- We now have to **minimize the loss function** $L(N, n)$ **w.r.t.** n .
- The main problem here is to express the probability p_n in terms of n (since p_n clearly depends on n : the longer the exploration phase, the higher the chance that it yields a correct decision).
- Going into details is beyond the scope of this lecture, so we only present the final result [Holland 1975]: n should be chosen according to

$$n \approx c_1 \ln \left(\frac{c_2 N^2}{\ln(c_3 N^2)} \right),$$

where c_1 , c_2 and c_3 are certain positive constants.

- By rewriting this expression, we can turn it into

$$N - n \approx e^{n/2c_1} \sqrt{\frac{\ln(c_3 N^2)}{c_2}} - n.$$

The Two-Armed Bandit Argument

- Since with growing n the term $e^{n/2c_1}$ dominates the expression on the right hand side, this equation can be simplified (accepting further approximation) by

$$N - n \approx e^{cn}.$$

- In other words, the total number of trials $N - n$ executed with the arm that is observed to be better should increase exponentially compared to the number of trials n that are executed with the arm that is observed to be worse.
- This result, though obtained for a two-armed bandit, can be transferred to multi-armed bandits.
- In this more general form it is then applied to the schemata that are considered in the schema theorem:
the arms of the bandit correspond to different schemata,
their payoff to the (average) fitness of chromosomes matching them.
- A chromosome in a population that matches a schema is seen as a trial of the corresponding bandit arm.

The Two-Armed Bandit Argument

- The arms of the bandit correspond to different schemata, their payoff to the (average) fitness of chromosomes matching them.
- A chromosome in a population that matches a schema is seen as a trial of the corresponding bandit arm.
- Recall, however, that a chromosome matches many schemata, thus exhibiting an inherently parallel exploration of the space of schemata.
- The schema theorem states that the number of chromosomes matching schemata with better than average fitness grows essentially exponentially over several generations.
- The two- or multi-armed bandit argument now says that this is an optimal strategy to balance
 - **exploration of schemata** (playing all arms of the bandit) and
 - **exploitation of schemata** (playing the arm or arms that have been observed to be better than the others).

Theoretical Consideration: The Argument of Minimal Alphabets

The Argument of Minimal Alphabets

- The **principle of minimal alphabets** is sometimes invoked to claim that binary encodings, as used by genetic algorithms, are “optimal” in a certain sense.
- The core idea is that the number of possible schemata should be maximized relative to the size of the search space (or the population size), so that the parallelism inherent in the search for schemata is maximally effective.
- That is, with the chromosomes of the population a number of schemata should be covered that is as large as possible.
- If chromosomes are defined as strings of length L over an alphabet \mathcal{A} , then the ratio of the number of schemata to the size of the search space is

$$\frac{(|\mathcal{A}| + 1)^L}{|\mathcal{A}|^L}.$$

- Clearly, this ratio is maximized if the size $|\mathcal{A}|$ of the alphabet is minimized.
- As the smallest usable alphabet has $|\mathcal{A}| = 2$, binary codings optimize this ratio.

The Argument of Minimal Alphabets

A more intuitive form of this argument was put forward by [Goldberg 1989]:

- The larger the size of the alphabet, the more difficult it is to find meaningful schemata, because a schema is matched by a larger number of chromosomes.
- Since a schema averages over the fitness of the matching chromosomes, the quality of a schema may be tainted by some bad chromosomes (low fitness) that happen to match the same schema.
⇒ One should minimize the number of chromosomes matched by a schema.
- Since a schema h matches $|\mathcal{A}|^{(L-\text{ord}(h))}$ chromosomes, we should use an alphabet of minimal size, that is, $|\mathcal{A}| = 2$.
- Whether these arguments are convincing is debatable.
- At least one has to admit that there is a **trade-off** between maximizing the number of schemata relative to the search space and expressing the problem in a more natural manner with a larger alphabet [Goldberg 1991].

Evolutionary Algorithms for Behavioral Simulation

Evolutionary Algorithms for Behavioral Simulation

- Up to now: Evolutionary algorithms are used to solve (numeric or discrete) optimization problems.
- Now: Evolutionary algorithms are used to simulate behavior (populations dynamics) and to find behavioral strategies.
- **Basis: Game Theory** [von Neumann & Morgenstern 1944]
 - Serves as an analysis tool for social and economic situations.
 - Modeling of actions as moves (of a game) in a fixed framework.
 - One of the most important theoretical foundations of economics.
- **General Approach:**
 - Encode the behavioral strategy of an agent in a chromosome.
 - Let agents interact with each other and assess their success.
 - Success is measured as a *payoff* from one or multiple games played.
 - Agents multiply or go extinct depending on their achieved success.

The Prisoner's Dilemma

The best-known problem of game theory is the **prisoner's dilemma**.

- Two persons committed a bank robbery and were arrested.
- However, the evidence is insufficient to convict them for the robbery in a lawsuit based entirely on circumstantial evidence.
- The evidence only suffices to convict them for a lesser crime (for example: illegal possession of a firearm). (penalty: 1 year in prison)
- The prosecutor offers a deal: principal witness (crown's witness) rule
 - If one of the two defendants confesses the bank robbery, he becomes the chief witness (crown's witness) and is not convicted.
 - However, using the testimony of the chief witness, the other defendant is convicted with a full sentence (penalty: 10 years in prison)
 - Problem: If both confess, the chief witness rule no longer applies. However, since they both confessed, they are granted extenuating circumstances and thus receive a lower sentence. (penalty: 5 years in prison each)

The Prisoner's Dilemma: Nash Equilibrium

In order to analyze the prisoner's dilemma one uses a **payoff matrix** (**A** and **B** denote the two defendants):

		B	
		keeps silent	confesses
A	keeps silent	-1 -1	0 -10
	confesses	0 -10	-5 -5

- Cooperation (both keep silent) is the best option overall (minimum total number of years in prison).
- **However:** Double confession is the so-called **Nash equilibrium**. [Nash 1950]

Nash equilibrium: Neither of the two players can improve his/her payoff if only he/she changes his/her action.

Every payoff matrix has at least one Nash equilibrium (for mixed strategies).

The Prisoner's Dilemma: Nash Equilibrium

- A **pure strategy** is a strategy in which each player must decide on one option, that is, must assign a probability of 1 to playing it (and a probability of 0 to all others).
- In a **mixed strategy** probabilities other than 1 and 0 may be assigned to the different options. That is, a mixed strategy is an unrestricted probability distribution over the different options a player can choose from.
- Nash's theorem holds only if mixed strategies are permitted.
- If only pure strategies are permitted, there may not be a Nash equilibrium:

	B		
A		option 1	option 2
option 1		0	2
option 2		1	0
		1	3

If Player A chooses Option 1, Player B prefers Option 2 to 1, but if Player A chooses Option 2, Player B prefers Option 1 to 2.

Analogously for Player B.

The Prisoner's Dilemma: Nash Equilibrium

A \ B	option 1	option 2
option 1	1, 0	0, 2
option 2	0, 1	3, 0

However, if mixed strategies are permitted (that is, for probabilities other than 0 and 1 for the different options), there is a Nash equilibrium for this payoff matrix.

- Let p be the probability with which Player A chooses Option 1 and consequently $1 - p$ the probability that Player A chooses Option 2.
- Let q be the probability with which Player B chooses Option 1 and consequently $1 - q$ the probability that Player B chooses Option 2.
- Let $E(\cdot)$ denote the expected payoff.
- For a Nash equilibrium it must be

$$\begin{aligned} E(\text{Player A chooses Option 1}) &= E(\text{Player A chooses Option 2}) \quad \text{and} \\ E(\text{Player B chooses Option 1}) &= E(\text{Player B chooses Option 2}) \end{aligned}$$

The Prisoner's Dilemma: Nash Equilibrium

- For Player A we obtain:

$$E(\text{Player A chooses Option 1}) = q \cdot 1 + (1 - q) \cdot 0 = q,$$

$$E(\text{Player A chooses Option 2}) = q \cdot 0 + (1 - q) \cdot 3 = 3 - 3q.$$

For a Nash equilibrium it must be

$$E(\text{Player A chooses Option 1}) = E(\text{Player A chooses Option 2}),$$

so that Player A does not have an incentive to change its strategy.

It follows $q = 3 - 3q$ or $q = 3/4$.

- Likewise we obtain for Player B:

$$E(\text{Player B chooses Option 1}) = p \cdot 0 + (1 - p) \cdot 1 = 1 - p,$$

$$E(\text{Player B chooses Option 2}) = p \cdot 2 + (1 - p) \cdot 0 = 2p.$$

For a Nash equilibrium it must be

$$E(\text{Player B chooses Option 1}) = E(\text{Player B chooses Option 2}),$$

so that Player B does not have an incentive to change its strategy.

It follows $1 - p = 2p$ or $p = 1/3$.

The Prisoner's Dilemma

General **payoff matrix** for the prisoner's dilemma:

A \ B	cooperate	defect
cooperate	R / R	S / T
defect	T / S	P / P

R: Reward for mutual cooperation

P: Punishment for mutual defection

T: Temptation to defect

S: Sucker's payoff

- The exact values for **R**, **P**, **T** and **S** are not important.
- However, it must be $T > R > P > S$ and $2R > T + S$.
(If the second condition does not hold, alternating defect is preferable.)
- This general payoff matrix has a Nash equilibrium even if only pure strategies are permitted.

The Prisoner's Dilemma

- Many every day situations can be described (and thus analyzed theoretically) by the model of the prisoner's dilemma.
- **However:** Even though double defection is the Nash equilibrium, we (also) observe different (cooperative) behavior.
- Question therefore (according to [Axelrod 1980]):
Under what circumstances does cooperation develop in a world of self-interested individuals without a central authority?
- Answer by [Hobbes 1651] (Leviathan):
 - **Not at all!**
Before an organized state existed, the natural state was dominated by self-interested (egotistic) individuals, which competed with each other so recklessly that life was “solitary, poor, nasty, brutish, and short.”
 - **However:** On an international scale there is *de facto* no central authority, but nevertheless we observe (economic and political) cooperation of states. How can this observation be explained?

The Prisoner's Dilemma

- **Approach** by [Axelrod 1980]:

Consider the **iterated prisoner's dilemma**.

(The prisoner's dilemma is played multiple times, with each player knowing (remembering) the previous actions (moves) of the other player.)

- **Idea** of this approach:

- If the prisoner's dilemma is played *only once*, it is best to choose the Nash equilibrium.
- If the prisoner's dilemma is played *multiple times*, a player can react to an uncooperative behavior of the other player.
(Possibility of *retaliation* for disadvantages suffered)

- **Questions:**

Does the iterated prisoner's dilemma lead to cooperation?

What is the best strategy for the iterated prisoner's dilemma?

The Prisoner's Dilemma

[Axelrod 1980] fixed the following **payoff matrix**:

		B	
		cooperate	defect
A	cooperate	3, 3	0, 5
	defect	5, 0	1, 1

(This is the set of smallest non-negative integer numbers that satisfy the conditions.)

- Axelrod invited researchers from various disciplines (psychology, social and political sciences, economics, mathematics), to write programs that play the prisoner's dilemma.
- A program may store (remember) its own moves and those of its opponent. Hence it can know whether its opponent cooperated or defected in previous moves and thus may adapt its strategy accordingly.

The Prisoner's Dilemma: Tournaments

In order to answer the questions raised above, Axelrod conducted two tournaments:

- **1. Tournament:**

- 14 programs plus one random player (Fortran)
- Round robin tournament with 200 matches per pairing
- Winner: Anatol Rapoport with Tit-for-Tat.
(Program plays opponent's move from the previous match.)

- The programs and results of the first tournament were published, then an invitation for a second tournament was sent out.

Idea: An analysis of the results of the first tournament may enable competitors to write better programs.

- **2. Tournament:**

- 62 programs plus one random player (Fortran and Basic)
- Round robin tournament with 200 matches per pairing
- Winner: Anatol Rapoport with Tit-for-Tat.

The Prisoner's Dilemma: Tit-for-Tat

- The game strategy of **Tit-for-Tat** is *very* simple:
 - Cooperate in the first match (play C).
 - In all following matches, play the move of the opponent from the directly preceding match.
- **Note:** A pure Tit-for-Tat may not be the best strategy, if it is played against *single* other strategies.
 - Only if there are individuals in a population with whom Tit-for-Tat can cooperate, it achieves a very good overall score.
 - **Problem** of Tit-for-Tat: It is **susceptible for mistakes**. If Tit-for-Tat plays against Tit-for-Tat and one of the two players defects “by accident”, a sequence of mutual retaliation results.
- A very important alternative strategy is **Tit-for-Two-Tat**: Retaliate only after two defections by the opponent.

The reduces the chance of mutual retaliations considerably, but also increases the chances of being exploited.

The Prisoner's Dilemma: Evolutionary Algorithm

Encoding the Game Strategies: [Axelrod 1987]

- Consider all possible match sequences of length 3 ($2^6 = 64$ possibilities).
- Store for each match sequence the move to be made in the next match (C — cooperate or D — defect, encoded in one bit):

		1. match	2. match	3. match	
1. Bit:	reply to	(C,C),	(C,C),	(C,C):	C
2. Bit:	reply to	(C,C),	(C,C),	(C,D):	D
3. Bit:	reply to	(C,C),	(C,C),	(D,C):	C
	⋮	⋮		⋮	⋮
64. Bit:	reply to	(D,D),	(D,D),	(D,D):	D

(First element of each pair: own move, second element: opponent's move)

- In addition: 6 bits to encode the match sequence before the first move.
→ each chromosome consists of 70 binary genes (each C or D).

Evolutionary Algorithm: Procedure

- Initialization of an initial population with random bit sequences (70 bits).
- From the current population random pairs of individuals are selected. They play 200 matches of the prisoner's dilemma against each other.
For the first three matches (a part of) the initial match sequence (stored in the chromosome) is used to determine the move to be executed. (A missing or short history is replaced or filled, respectively.)
- Each individual plays against the same number of opponents. (due to computing limitations — 1987! — no full round robin tournament).
- Selection of individuals for the next generation:
above average result ($x \geq \mu + \sigma$): 2 descendants
average result ($\mu - \sigma < x < \mu + \sigma$): 1 descendant
below average result ($\mu - \sigma \geq x$): 0 descendants
- Genetic operators: standard mutation, one-point crossover

Evolutionary Algorithm: Result

- The evolved strategies are very similar to **Tit-for-Tat**.
- [Axelrod 1987] identified the following general patterns:
 - **Don't rock the boat.** Cooperate after three cooperations.
 $(C,C), (C,C), (C,C) \rightarrow C$
 - **Be provokable.** Play defect after a sudden defect of the opponent.
 $(C,C), (C,C), (C,D) \rightarrow D$
 - **Accept an apology.** Cooperate after mutual exploitation.
 $(C,C), (C,D), (D,C) \rightarrow C$
 - **Forget.** (Don't be resentful.)
Cooperate, after cooperation has been restored
after an exploitation (even without retaliation).
 $(C,C), (C,D), (C,C) \rightarrow C$
 - **Accept a rut.** Play defect after three defects by the opponent.
 $(D,D), (D,D), (D,D) \rightarrow D$

The Prisoner's Dilemma: Extensions

The prisoner's dilemma may be extended in various ways, in order to make it more realistic and to capture more situations:

- In practice the effects of actions are not always perfectly observable. Therefore: Not the move of the opponent, but only probabilities are known. (This does *not* mean that mixed strategies are considered!)
- Often more than two agents are involved in an interaction: Multiple agent prisoner's dilemma

In addition, the descriptions of the strategies may be extended:

- Longer history of matches (more than three matches).
- A random component may be added for the choice of the move: Probabilities for choosing C or D instead of a fixed move. (This means that mixed strategies are considered!)
- Description of a game strategies by Moore automata or even general programs, that are modified in an evolutionary algorithm.

Genetic Programming

Genetic Programming

- **Up to now:** Representation of solution candidates / game strategies by chromosomes of fixed length (arrays of genes).
- **Now:** Representation by functional expressions or programs.
 - **Genetic Programming**
 - more complex chromosomes of variable length
- Formal basis: a grammar to describe the language, symbols to form the expressions/programs
- Fixing two sets of symbols for the expressions/programs:
 - \mathcal{F} — set of function symbols and operators
 - \mathcal{T} — set of terminal symbols (constants and variables)
- The two sets \mathcal{F} and \mathcal{T} are problem specific.
They should not be too large (in order to restrict the search space), but rich enough to allow for finding a solution.

Genetic Programming

- **Example 1:** Learning a Boolean Function

- $\mathcal{F} = \{\text{and, or, not, if ... then ... else ...}, \dots\}$
- $\mathcal{T} = \{x_1, \dots, x_m, 1, 0\}$ or $\mathcal{T} = \{x_1, \dots, x_m, t, f\}$

- **Example 2:** Symbolic Regression

(regression: Finding a best fit function for a given data set by minimizing the sum of squared errors — *method of least squares*)

- $\mathcal{F} = \{+, -, *, /, \sqrt{}, \sin, \cos, \log, \exp, \dots\}$
- $\mathcal{T} = \{x_1, \dots, x_m\} \cup \mathbb{R}$

- **Important:** It must be possible to evaluate any chromosome that might be created in the process.

Therefore: If necessary, completion of the domain of a function, for example

- $\forall x \leq 0 : \log(x) = 0$ or = smallest representable floating point number
- $\tan\left(\frac{\pi}{2}\right) = 0$ or = largest representable floating point number

Genetic Programming

- The chromosomes are expressions that are composed of elements of $\mathcal{C} = \mathcal{F} \cup \mathcal{T}$ (if necessary, parentheses are added to express precedence).
- However: Restriction to “well-formed” symbolic expressions.
This can be achieved with a typical **recursive definition** (prefix notation):
 - constants and variable symbols are symbolic expressions.
 - If t_1, \dots, t_n are symbolic expressions and $f \in \mathcal{F}$ is (n -ary) function symbol, then $(f t_1 \dots t_n)$ is a symbolic expression.
 - No other sequences of symbols are symbolic expressions.
- **Examples** for this definition:
 - “ $(+ (* 3 x) (/ 8 2))$ ” is a symbolic expression.
Lisp- or Scheme-style syntax, meaning: $3 \cdot x + \frac{8}{2}$.
 - “ $2 7 * (3 /$ ” is not a symbolic expression.
- Alternative: postfix notation (does not need parentheses)

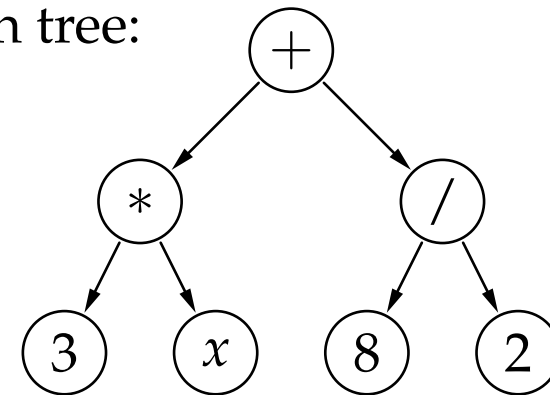
Genetic Programming

- For implementing genetic programming it is advantageous, to represent symbolic expressions by so-called **expression trees**. (expression trees are used, for example, in the front-end of a compiler in order to represent arithmetic expressions them and to optimize them afterward.)

symbolic expression:

$(+ (* 3 x) (/ 8 2))$

expression tree:



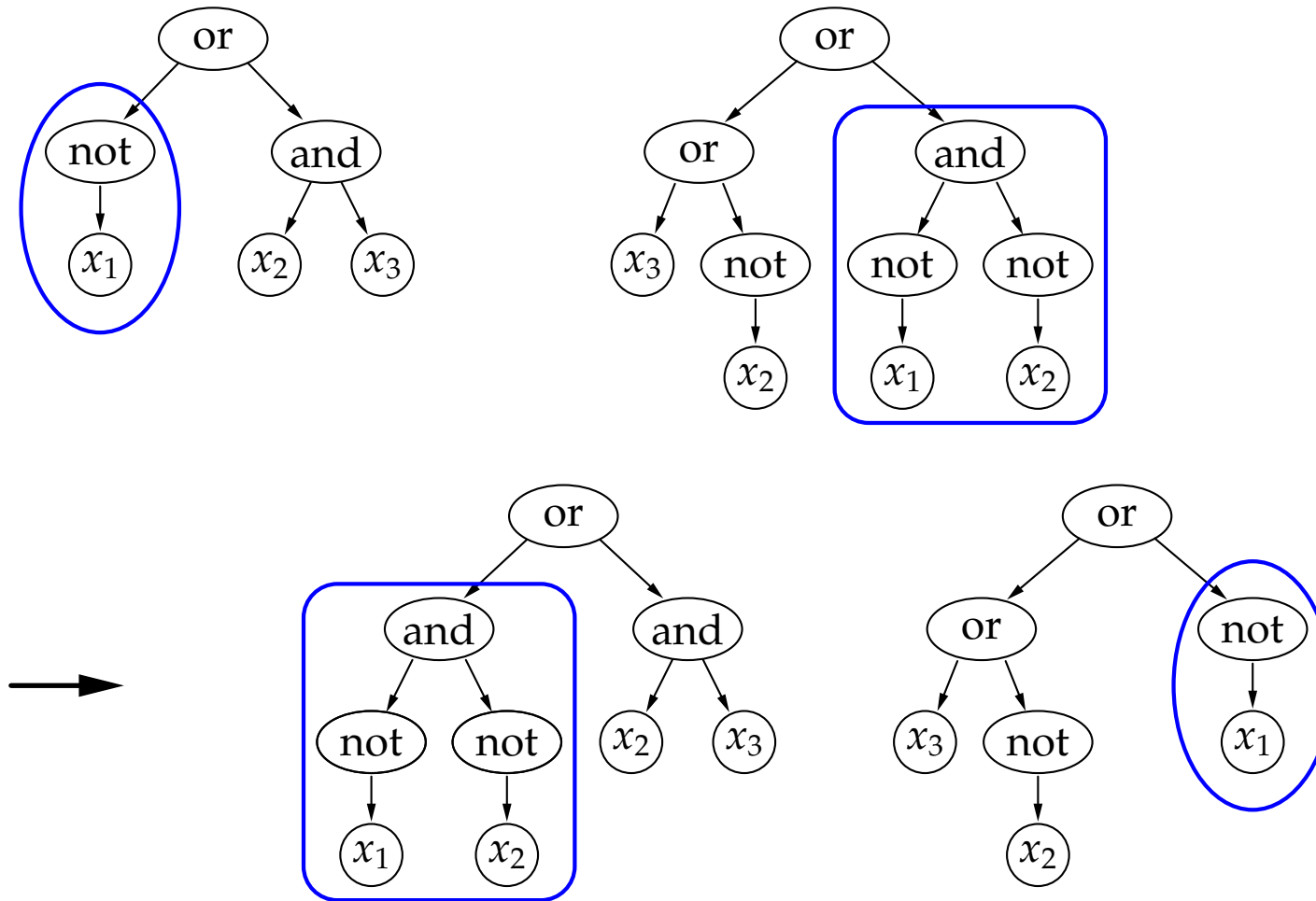
- In functional programming languages like Lisp or Scheme expressions are represented by (nested) lists:
 - The first element of a list is the function symbol or operator.
 - The subsequent elements are the arguments or operands.

Genetic Programming: Procedure

- Creating an **initial population** of random symbolic expressions.
Parameters of the creation process:
 - maximum nesting depth (maximum tree height)
 - probability for choosing a terminal symbol
- **Evaluation** of the expressions by computing the fitness.
 - Learning Boolean functions:
Number of correct results/outputs for all possible inputs or at least for a sufficiently large sample of the inputs.
 - Symbolic regression:
Sum of the squared errors over the given data points.
one-dimensional: data $(x_i, y_i), i = 1, \dots, n,$
fitness $f(c) = \sum_{i=1}^n (c(x_i) - y_i)^2$
- **Selection** with one of the known methods (work also for symbolic expressions).
- Application of **genetic operators**, usually only crossover.

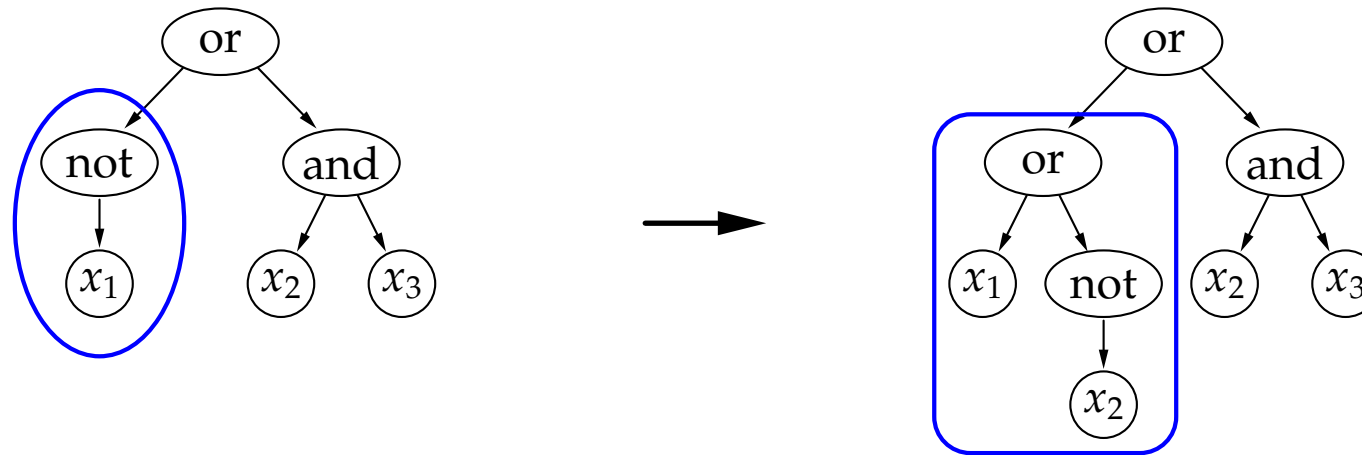
Genetic Programming: Crossover

- Crossover consists in exchanging sub-expressions (sub-trees)



Genetic Programming

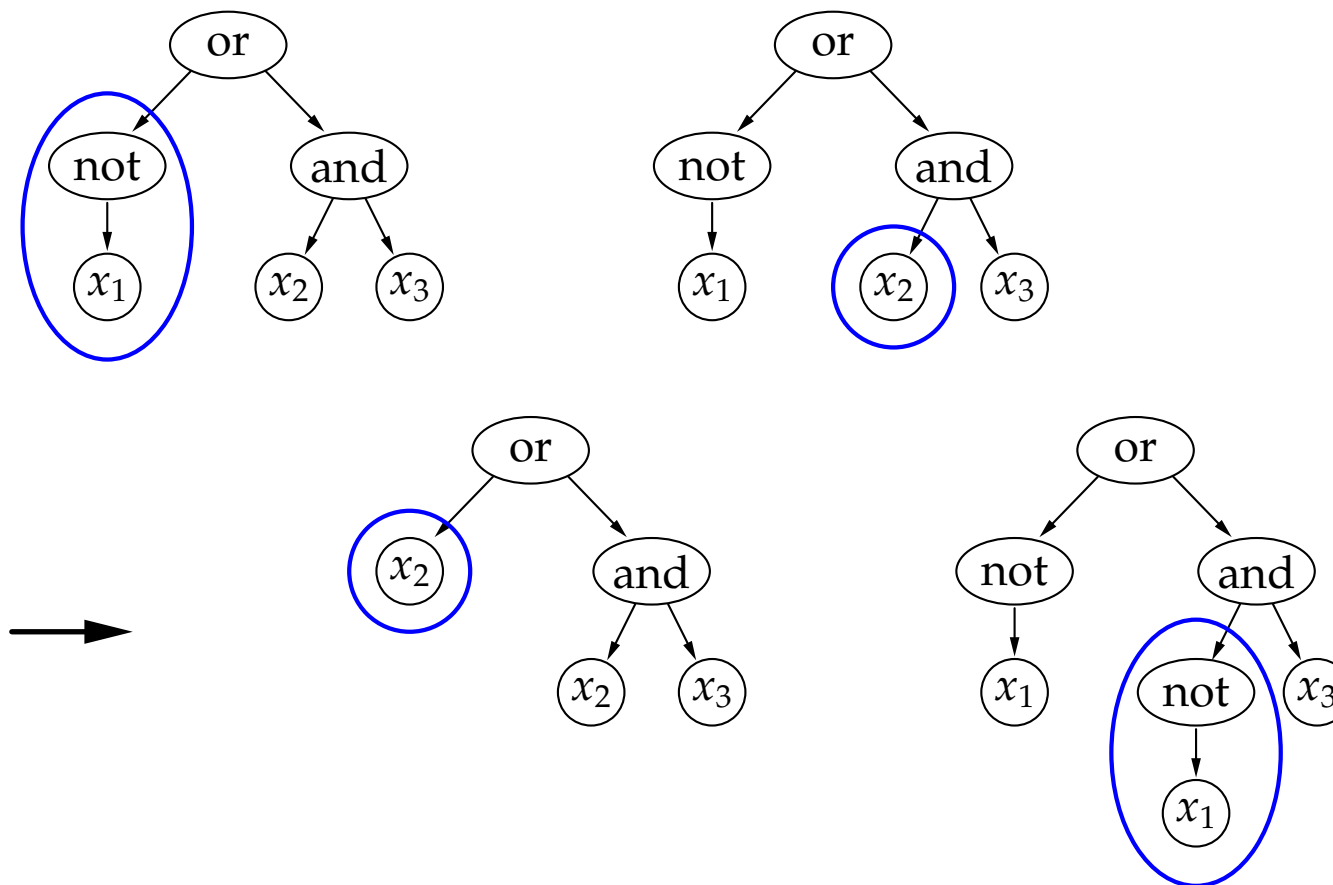
- Mutation consists in replacing a sub-expression (sub-tree) with a randomly created new expression (tree):



- It is advisable to replace only small sub-trees in order to limit the change that is effected.
- Often only crossover is used and no mutation.
- It should be taken care that the population is large enough, so that a sufficiently large supply of “genetic material” is available, from which new solution candidates can be combined. Diversity preserving procedure can help here.

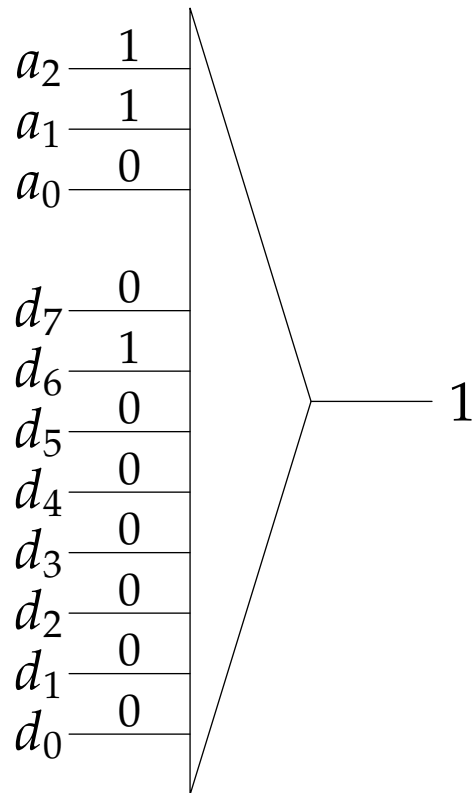
Genetic Programming

- **Important:** Crossover is more powerful than for arrays, because a crossover of identical parents can lead to new individuals. (It suffices to choose different sub-trees for exchange.)



Genetic Programming: 11-Multiplexer

Example: Learning a Boolean 11-Multiplexers [Koza 1992]



- Multiplexer with 8 data and 3 address lines.
(The state of the address lines specifies the data line to put through: multi-way switch.)
- $2^{11} = 2048$ possible input combinations with exactly one corresponding output.
- Definition of the symbol sets:
 - $\mathcal{T} = \{a_0, a_1, a_2, d_0, \dots, d_7\}$
 - $\mathcal{F} = \{\text{and, or, not, if}\}$
- Fitness function: $f(\omega) = 2048 - \sum_{i=1}^{2048} e_i$, where e_i is the error of the i -th input combination (i.e. fitness is number of correct outputs).

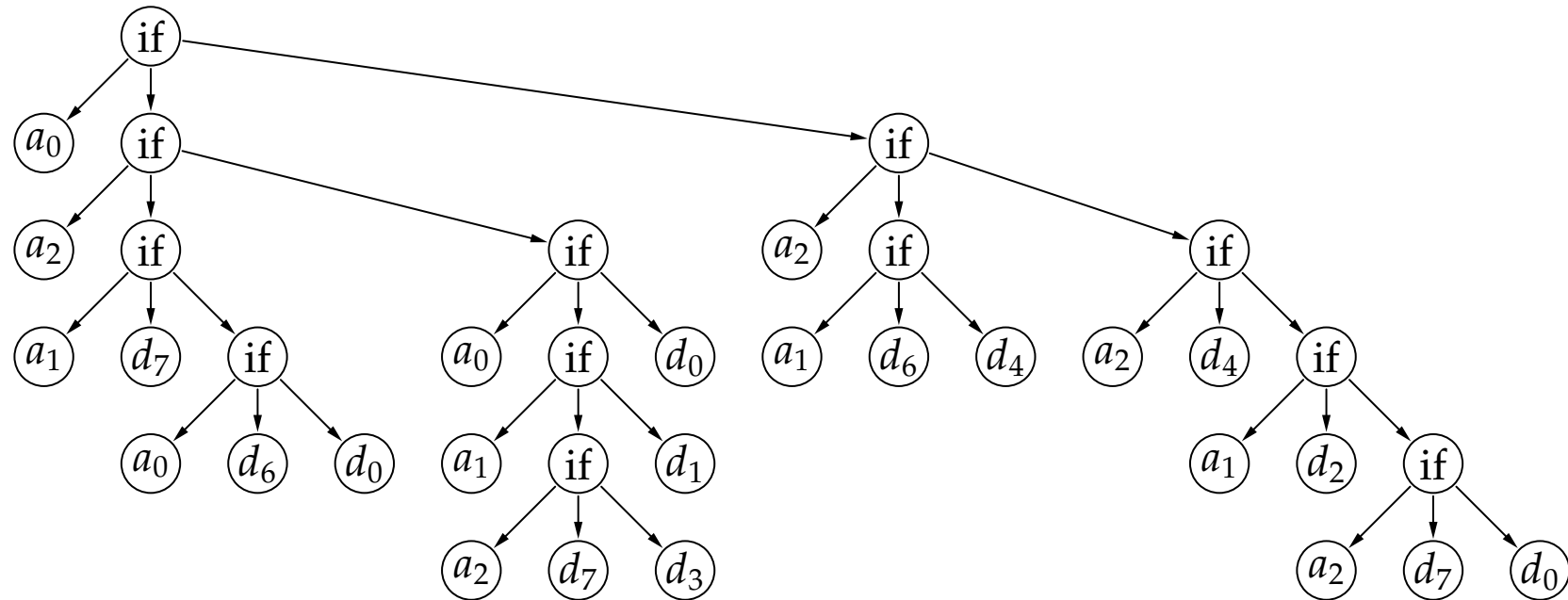
Genetic Programming: 11-Multiplexer

Example: Learning a Boolean 11-Multiplexers [Koza 1992]

- Population size: $\text{popsize} = 4000$
- Initial depth of the expression trees: 6, maximal depth: 17
- The fitness values in the initial population range from 768 to 1280, the average fitness is 1063.
(The expected value is 1024, since a random output should be correct, on average, in about half of the cases.)
- 23 expressions have a fitness of 1280, one of these represents a 3-multiplexer: $(\text{if } a_0 \ d_1 \ d_2)$
- Roulette wheel selection (fitness-proportional selection)
- 90% (3600) of the individuals are subjected to crossover, the remaining 10% are transferred unchanged.

Genetic Programming: 11-Multiplexer

- After only 9 generations the following solution was found: (fitness 2048):




- This result is surprisingly simple and therefore not plausible.
- It is possible that the result quality of genetic programming is exaggerated, which may be achieved by selecting (for reporting) the best of several runs. At least I usually had considerable trouble reproducing such results.

Genetic Programming: Stimulus Response Agent

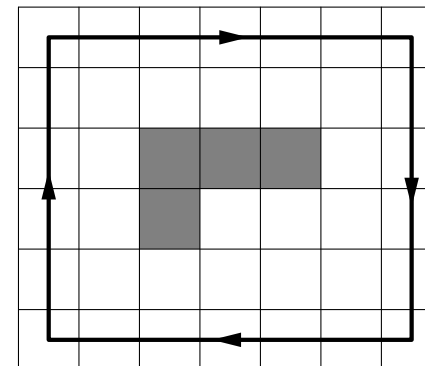
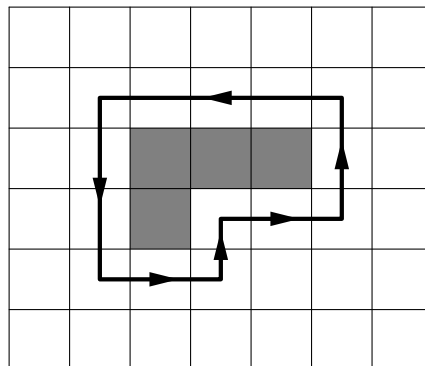
Example: Learning a Robot Control Program [Nilsson 1998]

- Consider a stimulus response agent in a grid world:

s_1	s_2	s_3
s_8		s_4
s_7	s_6	s_5

- 8 sensors s_1, \dots, s_8 that sense the state of the neighboring fields
- 4 actions: go east, go north, go west, go south
- Direct computation of the action (response) from the sensory input (stimulus), no memory

- Task:** Walk around an obstacle placed in the room or follow the walls of the room.



Genetic Programming: Stimulus Response Agent

- Symbol sets:
 - $\mathcal{T} = \{s_1, \dots, s_8, \text{east, north, west, south, 0, 1}\}$
 - $\mathcal{F} = \{\text{and, or, not, if}\}$

- Completion of the functions, for example, by

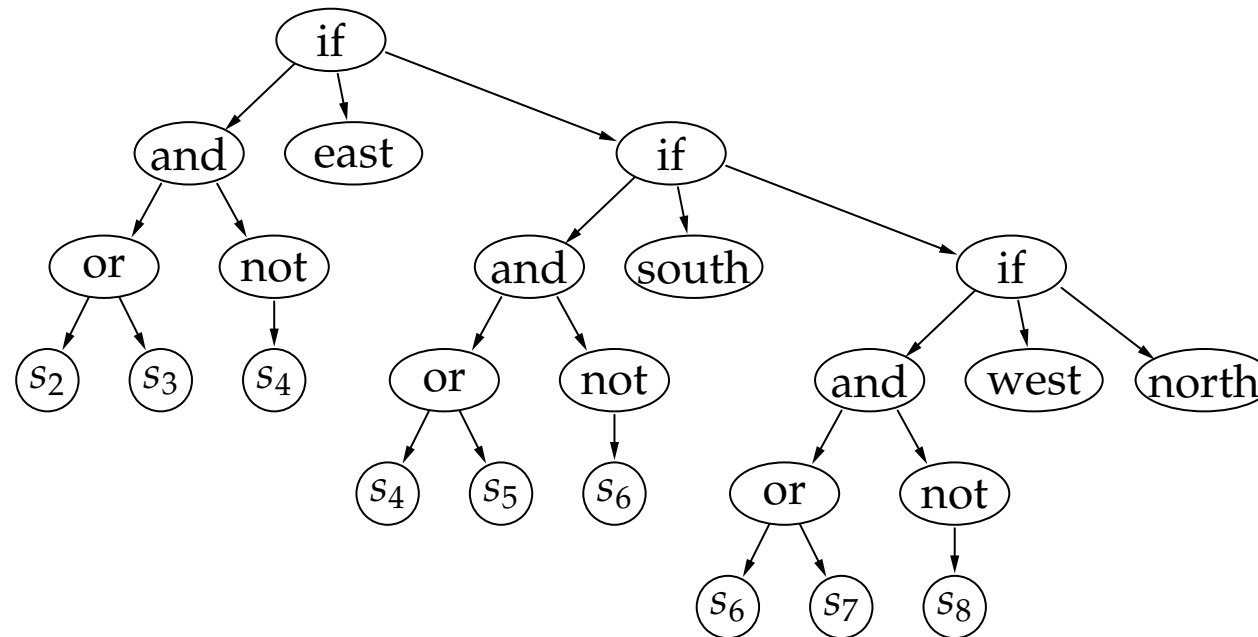
$$(\text{and } x \ y) = \begin{cases} \text{false,} & \text{if } x = \text{false,} \\ y, & \text{otherwise.} \end{cases}$$

(Note: In this way a Boolean operation may produce an action.)

- Population size: popsize = 5000, tournament selection with tournament size 5.
- Creation of the next population (next generation):
 - 10% (500) of the solution candidates are transferred unchanged.
 - 90% (4500) of the solution candidates are created by crossover.
 - <1% of the solution candidates are mutated.
- 10 generations (not counting the initial population) are computed.

Genetic Programming: Stimulus Response Agent

- Optimal, manually constructed solution:

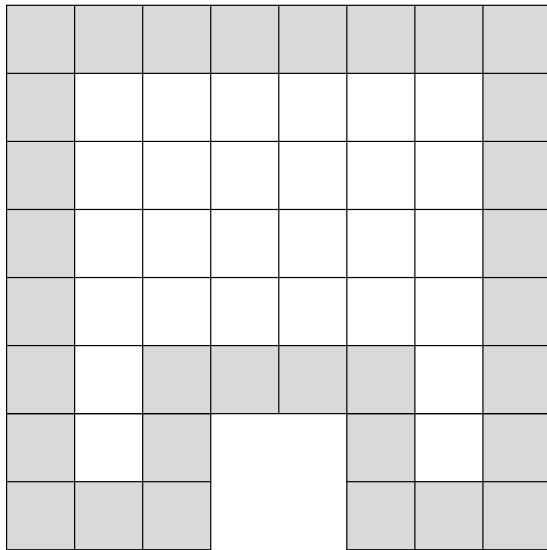


- It is highly unlikely that this solution is found.
- In order to keep the chromosomes simple, it may be advisable to use a penalty term, that penalizes the complexity of an expression.

However, the description of the experiments reported here does not mention such a penalty term.

Genetic Programming: Stimulus Response Agent

- Evaluation of a candidate solution with the help of a test room:

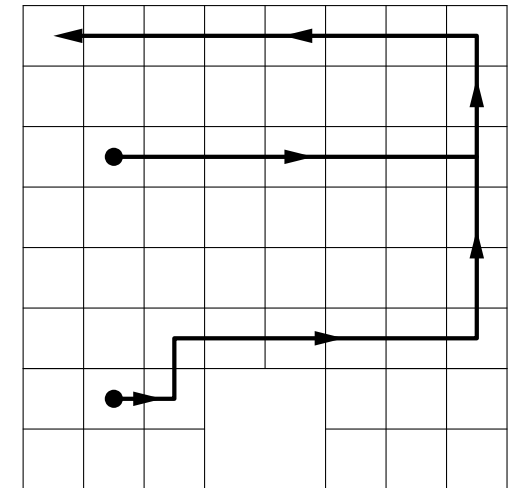


- A control program that works perfectly should steer the agent in such a way that it visits all of the fields marked in gray.
 - The initial field is chosen randomly.
 - If an action cannot be executed or instead of an action a truth value is produced, the execution of the control program is terminated.
- For each chromosome, 10 runs are executed.
 - In each run the agent is placed on a randomly chosen initial field and its movement is tracked.
 - The number of the border fields visited (shown in gray) is the fitness. (maximal fitness: $10 \cdot 32 = 320$)

Genetic Programming: Stimulus Response Agent

```
(and (not (not (if (if (not s1)
                  (if s4 north east)
                  (if west 0 south))
      (or (if s1 s3 s8) (not s7))
      (not (not north))))))
  (if (or (not (and (if s7 north s3)
                  (and south 1)))
        (or (or (not s6) (or s4 s4))
            (and (if west s3 s5)
                 (if 1 s4 s4))))))
    (or (not (and (not s3)
                 (if east s6 s2)))
        (or (not (if s1 east s6))
            (and (if s8 s7 1)
                 (or s7 s1))))))
    (or (not (if (or s2 s8)
                (or 0 s5)
                (or 1 east)))
        (or (and (or 1 s3)
                (and s1 east))
            (if (not west)
                (and west east)
                (if 1 north s8))))))
```

**Best Individual
in Generation 0:**

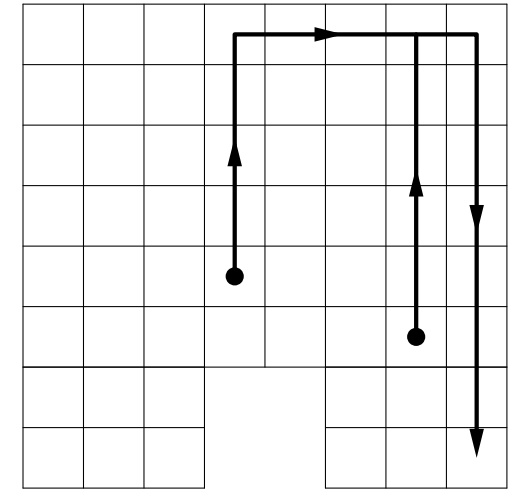


(movement from two
initial fields)

Genetic Programming: Stimulus Response Agent

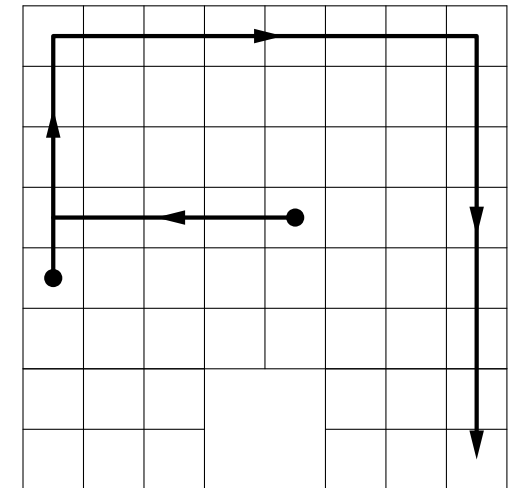
Best Individual in Generation 2:

```
(not (and (if s3
           (if s5 south east)
           north)
         (and not s4)))
```



Best Individual in Generation 6:

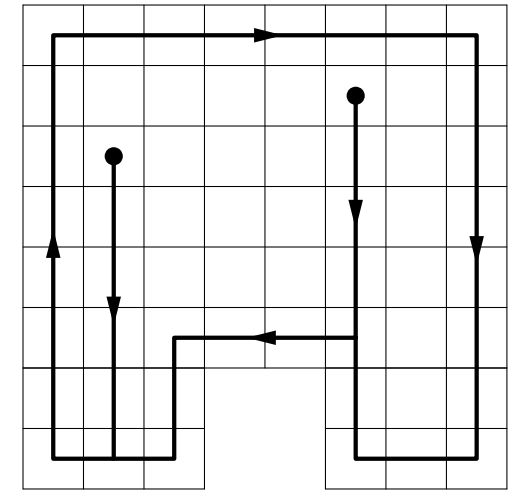
```
(if (and (not s4)
         (if s4 s6 s3))
    (or (if 1 s4 south)
        (if north east s3))
    (if (or (and 0 north)
           (and s4 (if s4
                     (if s5 south east)
                     north)))
        (and s4 (not (if s6 s7 s4)))
        (or (or (and s1 east) west) s1)))
```



Genetic Programming: Stimulus Response Agent

Best Individual in Generation 10:

```
(if (if (if s5 0 s3)
      (or s5 east)
      (if (or (and s4 0) s7)
          (or s7 0)
          (and (not (not (and s6 s5)))
              s5)))
    (if s8
        (or north
            (not (not s6)))
        west)
    (not (not (not (and (if (not south) s5 s8)
                       (not s2)))))))
```



- This result is surprisingly simple and therefore not plausible.
- It is possible that the result quality of genetic programming is exaggerated, which may be achieved by selecting (for reporting) the best of several runs. At least I usually had considerable trouble reproducing such results.

Genetic Programming: Stimulus Response Agent

Result found with the program available on the lecture page:

```
(or (if s3
      (if (not (and s4 s4))
            east
            (not south))
      (not south))
  (if (if north
        (not (and (not s7)
                  (or west south)))
        (and (not (not north))
              s4))
      (if (or (or (if (or east s4)
                    (and s8 s1)
                    (or s7 s1))
                (not (not south)))
            (not (or s5
                    (if north
                        north
                        north))))
          (if (or (if (and west south)
                    (and s3 east)
```

```
(if south
      s8
      s4))
  (not (or s3 s8)))
west
(or (or (or north s1)
      (not north))
    (and s4
          (and s1 s4))))
(and s4
  (not (and north s8)))
(or (if (if north
          s6
          s8)
      (or west west)
      (or south s7))
    (if (and s6 s4)
        s3
        s7)))
```

Genetic Programming: Extensions

- **Editing**

- Serves the purpose to simplify the chromosomes.
- Chromosomes or chromosome parts may represent a (very) simple function in an overly complex way.
- By transformations, which exploit logical and/or arithmetical equivalences, the expression is simplified.

- **Encapsulation / automatically defined functions**

- Potentially good sub-expressions should be protected against destruction by crossover or mutation.
- For a sub-expression (of a good chromosome) a new function is defined and a symbol representing this function is added to the set \mathcal{F} .
- The number of arguments of this new function equals the number of (distinct) leaves of the corresponding sub-tree.

Evolution Strategies

Evolution Strategies

- Up to now: Treatment of arbitrary (also discrete) optimization problems
Now : Restriction to **numeric optimization**

Given : Function $f : \mathbb{R}^n \rightarrow \mathbb{R}$

Desired : Minimum or maximum of f

Therefore : The chromosomes are **vectors of real numbers**.

- **Mutation** consists in the addition of a normally distributed **random vector** \vec{r} :

Each element r_i of the random vector is the realization of a normally distributed random variable with

- expected value 0 (independent of the element index i) and
- variance σ_i^2 or standard deviation σ_i .

The variance σ_i^2 may be dependent on or independent of the element index i and dependent on or independent of the generation number t .

- **Crossover** (recombination of two vectors) is often dispensed with, that is, often mutation is the only genetic operator.

Evolution Strategies: Selection

- **Strict Elitism:**
only the best individuals are passed into the next generation.
- Notation: μ – number of individuals in the parent generation
 λ – number of created offspring individuals (by mutation)
- **Two Fundamental Selection Strategies:**
 - **+ -Strategy** (Plus-Strategy, $(\mu + \lambda)$ -Strategy)
From the $(\mu + \lambda)$ individuals of the parent generation and the created offspring individuals the best μ chromosomes are selected.
(With this strategy it is usually $\lambda < \mu$.)
 - **, -Strategy** (Comma-Strategy, (μ, λ) -Strategy)
 $\lambda > \mu$ offspring individuals are generated,
from which the best μ chromosomes are selected.
(The chromosomes of the parent generation are definitely lost.)

Evolution Strategies: Selection

- **Example:** Consider the special case of the (1+1)-strategy
 - Initial “population”: \vec{x}_0 (randomly generated vector of real numbers)
 - Creation of the next generation:
Create a real-valued random vector \vec{r}_t and compute $\vec{x}_t^* = \vec{x}_t + \vec{r}_t$.
Then set
$$\vec{x}_{t+1} = \begin{cases} \vec{x}_t^*, & \text{if } f(\vec{x}_t^*) \geq f(\vec{x}_t), \\ \vec{x}_t, & \text{otherwise.} \end{cases}$$
 - Create additional generations until a termination criterion is satisfied.
- Obviously, this procedure is identical to the method of **hill climbing / random ascent** discussed earlier.
- Therefore the general +-strategy can be seen as a kind of *parallelized random ascent*, which is carried out at the same time at different locations in the search space, where always the most promising μ paths are followed.

Evolution Strategies

- Reminder (from the list of principles of organismic evolution):

Evolution-strategic Principles

Not only the organisms are optimized, but also the mechanisms of evolution: reproduction and death rates, life spans, susceptibility to mutations, mutations step widths, speed of evolution etc.

- Here: Adaptation of the variance of a random vector (mutation step width)
 - low variance → only small changes are made to the chromosomes
→ local search (exploitation)
 - high variance → large changes are made to the chromosomes
→ global search (exploration)
- Other possibilities to adapt parameters of the evolution process:
 - Choose the number of genes (vector elements) to modify.
 - Choose the number λ of the offspring individuals.

Evolution Strategies: Variance Adaptation

Global Variance Adaptation (chromosome-independent variance)

- Idea: Choose σ^2 or σ in such a way that the average convergence rate is as large as possible.
- Approach by [Rechenberg 1973]: Determine the optimal variance σ for
 - $f_1(x_1, \dots, x_n) = a + bx_1$ and
 - $f_2(x_1, \dots, x_n) = \sum_{i=1}^n x_i^2$,

by computing the probabilities for a successful (i.e. improving) mutation. These are

- for f_1 : $p_1 \approx 0.184$ and
- for f_2 : $p_2 \approx 0.270$.
- From this result the $\frac{1}{5}$ -**success rule** was derived heuristically:
In the +-strategy the mutation width has a good value, if about $\frac{1}{5}$ of the offspring individuals are better than the parents.

Evolution Strategies: Variance Adaptation

Global Variance Adaptation (chromosome-independent variance)

- Adaptation of the variance σ^2 on the basis of the $\frac{1}{5}$ -success rule:
 - If more than $\frac{1}{5}$ of the offspring individuals are better than the parents, enlarge the variance/standard deviation.

$$\sigma^{(\text{new})} = \sigma^{(\text{old})} \cdot c_{\text{inc}}, \quad c_{\text{inc}} \approx 1.22.$$

- If less than $\frac{1}{5}$ of the offspring individuals are better than the parents, reduce the variance/standard deviation.

$$\sigma^{(\text{new})} = \sigma^{(\text{old})} \cdot c_{\text{dec}}, \quad c_{\text{dec}} = \frac{1}{c_{\text{inc}}} \approx 0.83.$$

- For larger population sizes the $\frac{1}{5}$ -success rule is sometimes too optimistic, that is, it favors exploitation too much.
- In analogy to **simulated annealing** one may define a function with which the variance is reduced in the course of the generations.

Evolution Strategies: Variance Adaptation

Local Variance Adaptation (chromosome-specific variance)

- The variance/standard deviation is added to the chromosomes (as one or more additional genes):
 - one variance for all vector elements or
 - an individual variance for each vector element (double length of the vector/chromosome)
- **Note:** The additional vector elements for the variances have *no direct influence* on the fitness of a chromosome.
- **Expectation:** chromosomes with “bad” variances, that is,
 - too small: chromosomes do not develop quickly enough or
 - too large: chromosomes move too far away from their parents, create comparatively more “bad” offspring individuals. Therefore their genes (and hence their variances) die out more easily.

Evolution Strategies: Variance Adaptation

Local Variance Adaptation (chromosome-specific variance)

- The element-specific mutation step widths (standard deviations) are mutated according to the following schema:

$$\sigma_i^{(\text{new})} = \sigma_i^{(\text{old})} \cdot \exp(r_1 \cdot N(0, 1) + r_2 \cdot N_i(0, 1)).$$

$N(0, 1)$: normally distributed random number, one per chromosome

$N_i(0, 1)$: normally distributed random number, one for each vector element

- Recommended values for the parameter r_1 and r_2 are [Bäck and Schwefel 1993]

$$r_1 = \frac{1}{\sqrt{2n}}, \quad r_2 = \frac{1}{\sqrt{2\sqrt{n}}},$$

where n is the number of vector elements, or [Nissen 1997]

$$r_1 = 0.1, \quad r_2 = 0.2.$$

- Often a lower bound for the mutation step widths is introduced.

Evolution Strategies: Variance Adaptation

Extensions of Local Variance Adaptation

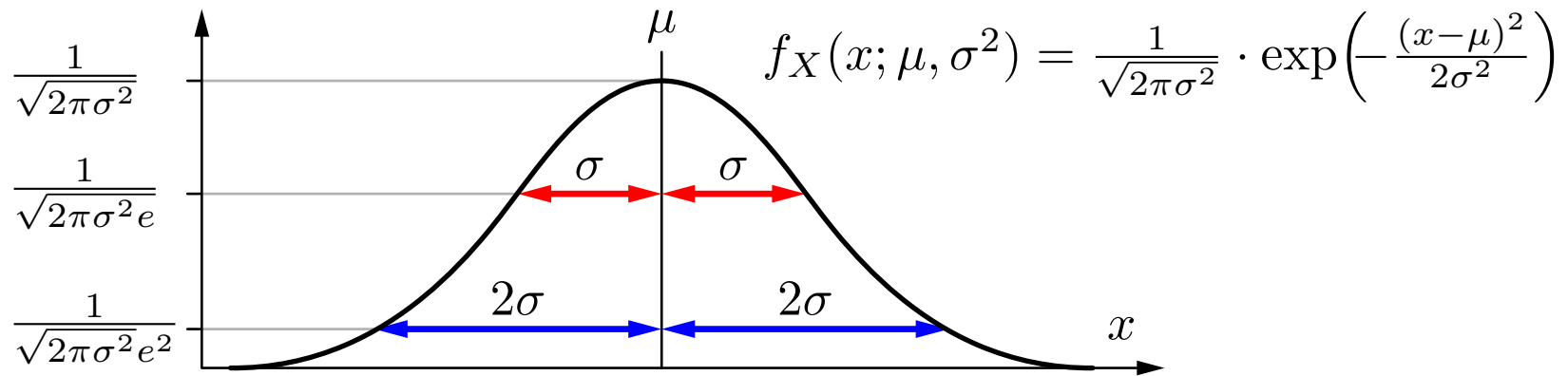
- In the standard form of local variance adaptation the variances of the different vector elements are independent of each other. (Formally: their covariance matrix is a diagonal matrix.)
- If the variations of a chromosome are to be created preferably in certain directions, this can be expressed with individual variances only if these directions are axes-parallel.
- **Example:** Variations of chromosomes with two genes are to be created preferably in the direction of the main diagonal, that is, in the direction $\pm(1, 1)$.
This cannot be expressed by individual variances alone.
- **Solution:** Use a covariance matrix with a large covariance, for example

$$\Sigma = \begin{pmatrix} 1 & 0.9 \\ 0.9 & 1 \end{pmatrix}.$$

Reminder: Variance and Standard Deviation

- **Special Case: Normal/Gaussian Distribution**

The variance/standard deviation provides information about the height of the mode and the width of the curve.



- μ : expected value, estimated by mean value \bar{x} ,
 - σ^2 : variance, estimated by (empirical) variance s^2 ,
 - σ : standard deviation, estimated by (empirical) standard deviation s .
- Important: the standard deviation has the same unit as the expected value.

Multivariate Normal Distribution

- A **univariate normal distribution** has the density function

$$f_X(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

- μ : expected value, estimated by mean value \bar{x} ,
- σ^2 : variance, estimated by (empirical) variance s^2 ,
- σ : standard deviation, estimated by (empirical) standard deviation s .

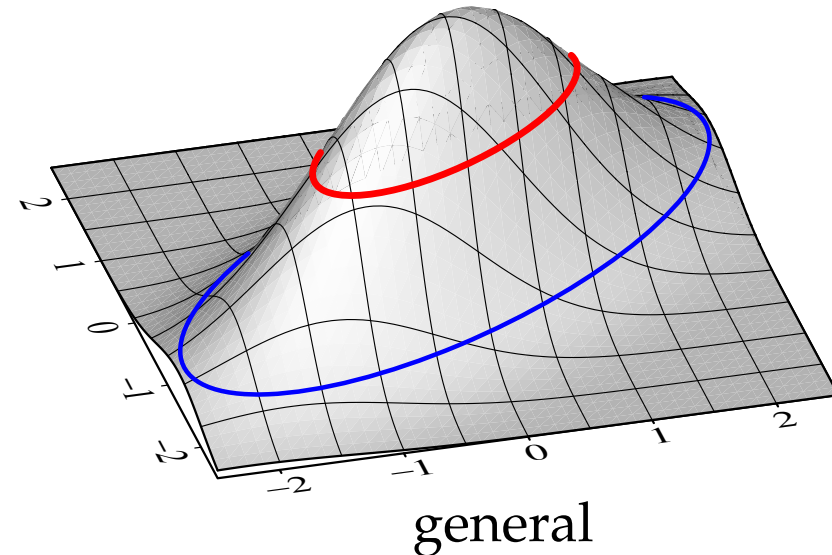
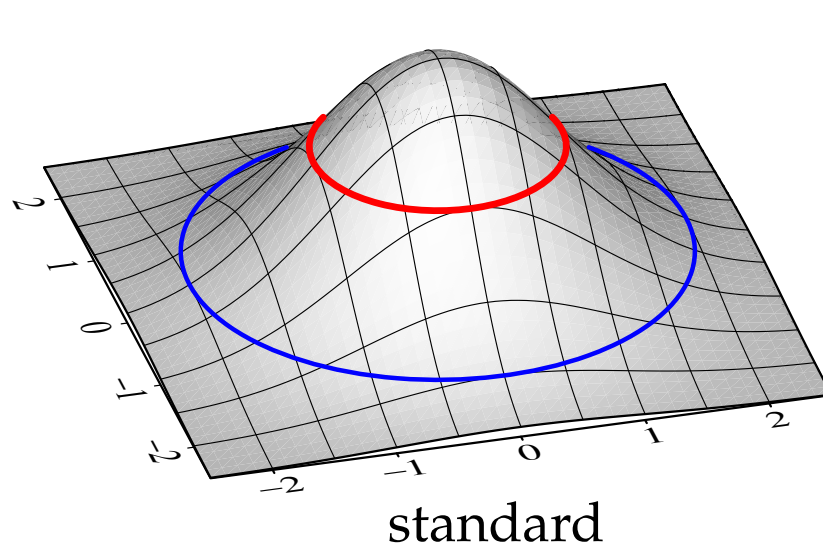
- A **multivariate normal distribution** has the density function

$$f_{\vec{X}}(\vec{x}; \vec{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^m |\Sigma|}} \cdot \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^\top \Sigma^{-1}(\vec{x} - \vec{\mu})\right)$$

- m : size of the vector \vec{x} (it is m -dimensional),
- $\vec{\mu}$: mean value vector, estimated by (empirical) mean value vector $\bar{\vec{x}}$,
- Σ : covariance matrix, estimated by (empirical) covariance matrix \mathbf{S} ,
- $|\Sigma|$: determinant of the covariance matrix Σ .

Interpretation of a Covariance Matrix

- The variance/standard deviation relates the spread of the distribution to the spread of a **standard normal distribution** ($\sigma^2 = \sigma = 1$).
- The covariance matrix relates the spread of the distribution to the spread of a **multivariate standard normal distribution** ($\Sigma = \mathbf{1}$).
- Example: bivariate normal distribution



- **Question:** Is there a multivariate analog of standard deviation?

Interpretation of a Covariance Matrix

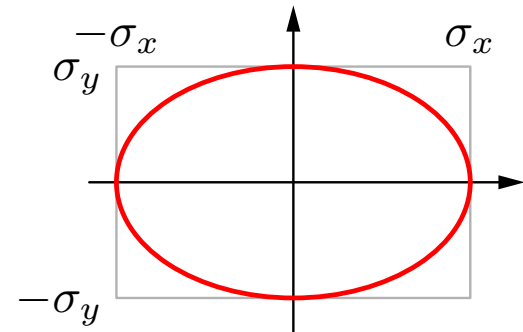
Question: Is there a multivariate analog of standard deviation?

First insight:

If all covariances vanish, the contour lines are axes-parallel ellipses.

The upper ellipse is inscribed into the rectangle $[-\sigma_x, \sigma_x] \times [-\sigma_y, \sigma_y]$.

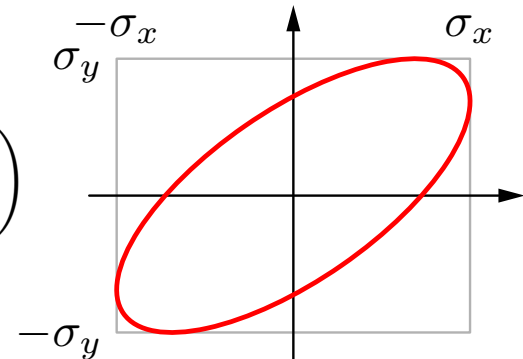
$$\Sigma = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix}$$



Second insight:

If the covariances do not vanish, the contour lines are rotated ellipses. Still the upper ellipse is inscribed into the rectangle $[-\sigma_x, \sigma_x] \times [-\sigma_y, \sigma_y]$.

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$$



Consequence: A covariance matrix describes a scaling and a rotation.

Cholesky Decomposition

- Intuitively: **Compute an analog of standard deviation.**
- Let \mathbf{S} be a symmetric, positive definite matrix (e.g. a covariance matrix). Cholesky decomposition serves the purpose to compute a “square root” of \mathbf{S} .
 - symmetric: $\forall 1 \leq i, j \leq m : s_{ij} = s_{ji}$
 - positive definite: for all m -dimensional vectors $\vec{v} \neq \vec{0}$ it is $\vec{v}^\top \mathbf{S} \vec{v} > 0$
- Formally: Compute a left/lower triangular matrix \mathbf{L} such that $\mathbf{L}\mathbf{L}^\top = \mathbf{S}$. (\mathbf{L}^\top is the transpose of the matrix \mathbf{L} .)

$$l_{ii} = \left(s_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \right)^{\frac{1}{2}}$$
$$l_{ji} = \frac{1}{l_{ii}} \left(s_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk} \right), \quad j = i + 1, i + 2, \dots, m.$$

Cholesky Decomposition

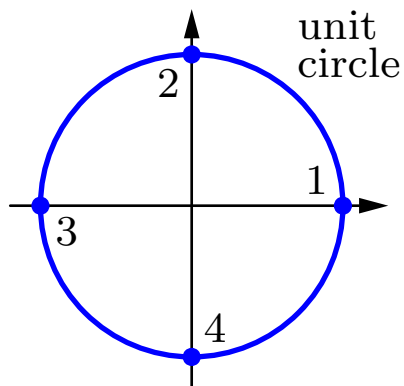
Special Case: Two Dimensions

- Covariance matrix

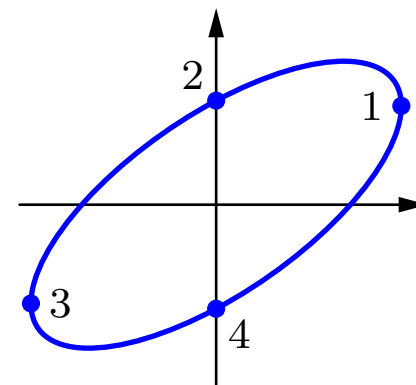
$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$$

- Cholesky decomposition

$$\mathbf{L} = \begin{pmatrix} \sigma_x & 0 \\ \frac{\sigma_{xy}}{\sigma_x} & \frac{1}{\sigma_x} \sqrt{\sigma_x^2 \sigma_y^2 - \sigma_{xy}^2} \end{pmatrix}$$



mapping with \mathbf{L}
 $\vec{v}' = \mathbf{L}\vec{v}$



Eigenvalue Decomposition

- Eigenvalue decomposition also yields an **analog of standard deviation**.
- It is computationally more expensive than Cholesky decomposition.
- Let \mathbf{S} be a symmetric, positive definite matrix (e.g. a covariance matrix).
 - \mathbf{S} can be written as

$$\mathbf{S} = \mathbf{R} \operatorname{diag}(\lambda_1, \dots, \lambda_m) \mathbf{R}^{-1},$$

where the $\lambda_j, j = 1, \dots, m$, are the eigenvalues of \mathbf{S}
and the columns of \mathbf{R} are the (normalized) eigenvectors of \mathbf{S} .

- The eigenvalues $\lambda_j, j = 1, \dots, m$, of \mathbf{S} are all positive
and the eigenvectors of \mathbf{S} are orthonormal ($\rightarrow \mathbf{R}^{-1} = \mathbf{R}^\top$).
- Due to the above, \mathbf{S} can be written as $\mathbf{S} = \mathbf{T} \mathbf{T}^\top$, where

$$\mathbf{T} = \mathbf{R} \operatorname{diag} \left(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_m} \right)$$

Eigenvalue Decomposition

Special Case: Two Dimensions

- Covariance matrix

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$$

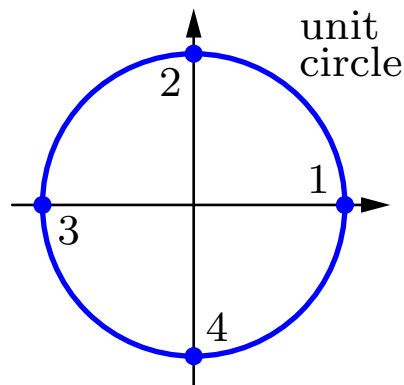
- Eigenvalue decomposition

$$\mathbf{T} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix},$$

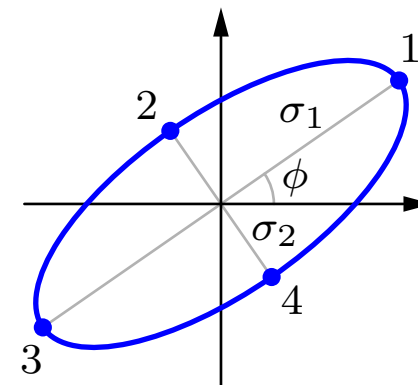
$$s = \sin \phi, c = \cos \phi, \phi = \frac{1}{2} \arctan \frac{2\sigma_{xy}}{\sigma_x^2 - \sigma_y^2},$$

$$\sigma_1 = \sqrt{c^2\sigma_x^2 + s^2\sigma_y^2 + 2sc\sigma_{xy}},$$

$$\sigma_2 = \sqrt{s^2\sigma_x^2 + c^2\sigma_y^2 - 2sc\sigma_{xy}}.$$



mapping with \mathbf{T}
 $\vec{v}' = \mathbf{T}\vec{v}$



Eigenvalue Decomposition

Eigenvalue decomposition enables us to write a covariance matrix Σ as

$$\Sigma = \mathbf{T}\mathbf{T}^\top \quad \text{with} \quad \mathbf{T} = \mathbf{R} \operatorname{diag} \left(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_m} \right).$$

As a consequence we can write its inverse Σ^{-1} as

$$\Sigma^{-1} = \mathbf{U}^\top \mathbf{U} \quad \text{with} \quad \mathbf{U} = \operatorname{diag} \left(\lambda_1^{-\frac{1}{2}}, \dots, \lambda_m^{-\frac{1}{2}} \right) \mathbf{R}^\top.$$

\mathbf{U} describes the inverse mapping of \mathbf{T} , i.e., rotates the ellipse so that its axes coincide with the coordinate axes and then scales the axes to unit length. Hence:

$$(\vec{x} - \vec{y})^\top \Sigma^{-1} (\vec{x} - \vec{y}) = (\vec{x} - \vec{y})^\top \mathbf{U}^\top \mathbf{U} (\vec{x} - \vec{y}) = (\vec{x}' - \vec{y}')^\top (\vec{x}' - \vec{y}'),$$

where $\vec{x}' = \mathbf{U}\vec{x}$ and $\vec{y}' = \mathbf{U}\vec{y}$.

Result: $(\vec{x} - \vec{y})^\top \Sigma^{-1} (\vec{x} - \vec{y})$ is equivalent to the squared **Euclidean distance** in the properly scaled eigensystem of the covariance matrix Σ .

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^\top \Sigma^{-1} (\vec{x} - \vec{y})} \quad \text{is called **Mahalanobis distance** .}$$

Eigenvalue Decomposition

Eigenvalue decomposition also shows that the determinant of the covariance matrix Σ provides a measure of the (hyper-)volume of the (hyper-)ellipsoid. It is

$$|\Sigma| = |\mathbf{R}| |\text{diag}(\lambda_1, \dots, \lambda_m)| |\mathbf{R}^\top| = |\text{diag}(\lambda_1, \dots, \lambda_m)| = \prod_{i=1}^m \lambda_i,$$

since $|\mathbf{R}| = |\mathbf{R}^\top| = 1$ as \mathbf{R} is orthogonal with unit length columns, and thus

$$\sqrt{|\Sigma|} = \prod_{i=1}^m \sqrt{\lambda_i},$$

which is proportional to the (hyper-)volume of the (hyper-)ellipsoid.

To be precise, the volume of the m -dimensional (hyper-)ellipsoid a (hyper-)sphere with radius r is mapped to with a covariance matrix Σ is

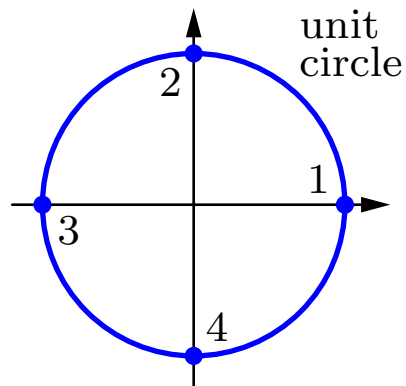
$$V_m(r) = \frac{\pi^{\frac{m}{2}} r^m}{\Gamma(\frac{m}{2} + 1)} \sqrt{|\Sigma|}, \quad \text{where} \quad \begin{aligned} \Gamma(x) &= \int_0^\infty e^{-t} t^{x-1} dt, \quad x > 0, \\ \Gamma(x+1) &= x \cdot \Gamma(x), \quad \Gamma(\frac{1}{2}) = \sqrt{\pi}, \quad \Gamma(1) = 1. \end{aligned}$$

Eigenvalue Decomposition

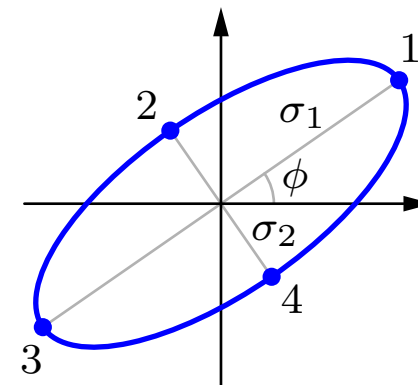
Special Case: Two Dimensions

- Covariance matrix and its eigenvalue decomposition:

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} \quad \text{and} \quad \mathbf{T} = \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}.$$



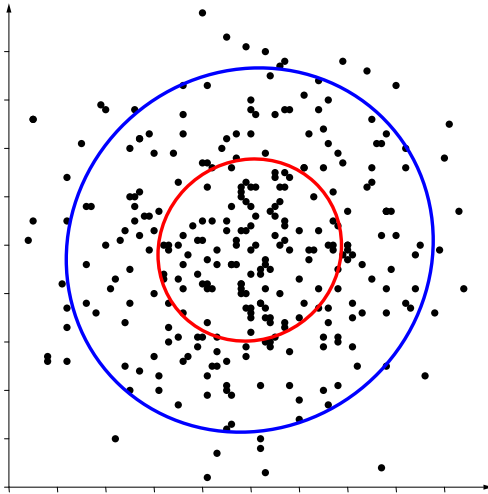
mapping with \mathbf{T}
 $\vec{v}' = \mathbf{T}\vec{v}$



- The area of the ellipse, to which the unit circle (area π) is mapped, is

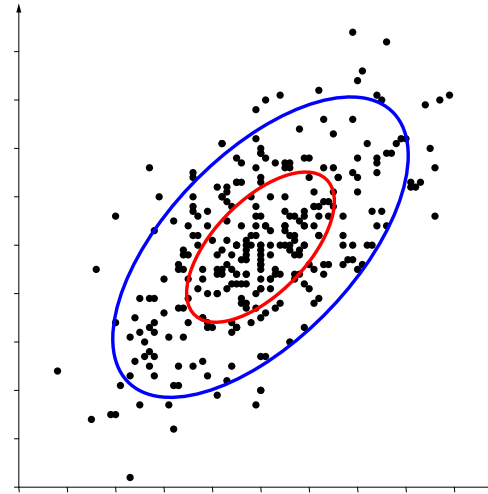
$$A = \pi\sigma_1\sigma_2 = \pi\sqrt{|\Sigma|}.$$

Covariance Matrices of Example Data Sets



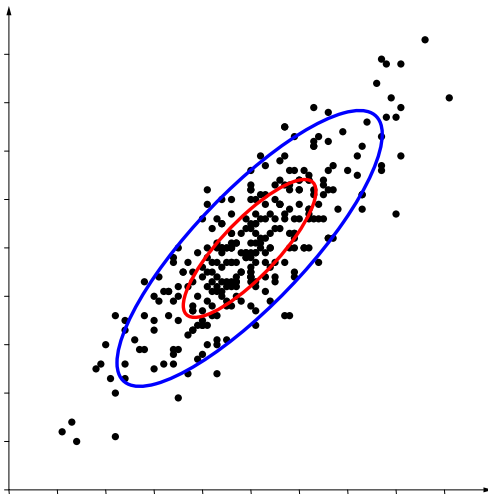
$$\Sigma \approx \begin{pmatrix} 3.59 & 0.19 \\ 0.19 & 3.54 \end{pmatrix}$$

$$\mathbf{L} \approx \begin{pmatrix} 1.90 & 0 \\ 0.10 & 1.88 \end{pmatrix}$$



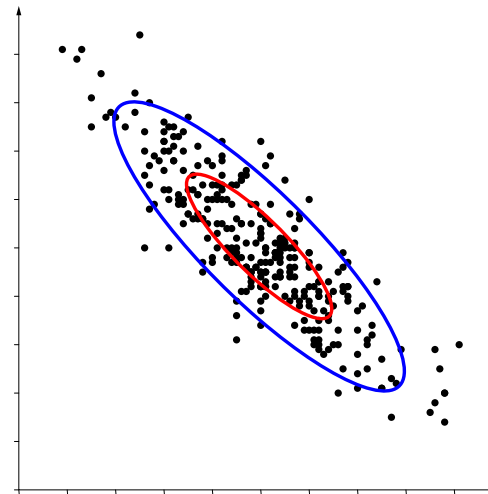
$$\Sigma \approx \begin{pmatrix} 2.33 & 1.44 \\ 1.44 & 2.41 \end{pmatrix}$$

$$\mathbf{L} \approx \begin{pmatrix} 1.52 & 0 \\ 0.95 & 1.22 \end{pmatrix}$$



$$\Sigma \approx \begin{pmatrix} 1.88 & 1.62 \\ 1.62 & 2.03 \end{pmatrix}$$

$$\mathbf{L} \approx \begin{pmatrix} 1.37 & 0 \\ 1.18 & 0.80 \end{pmatrix}$$



$$\Sigma \approx \begin{pmatrix} 2.25 & -1.93 \\ -1.93 & 2.23 \end{pmatrix}$$

$$\mathbf{L} \approx \begin{pmatrix} 1.50 & 0 \\ -1.29 & 0.76 \end{pmatrix}$$

Evolution Strategies: Variance Adaptation

- **Generally:** Correlated mutation is described by n variances and $\frac{n(n-1)}{2}$ rotation angles.

- In this case the covariance matrix

$$\Sigma = \left(\prod_{i=1}^n \prod_{k=i+1}^n R_{ik}(\varphi_{ik}) \right) \text{diag}(\sigma_1^2, \dots, \sigma_n^2) \left(\prod_{i=1}^n \prod_{k=i+1}^n R_{ik}(\varphi_{ik}) \right)^{-1}$$

is used, where

$$R_{ik}(\varphi_{ik}) = \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & \cos \varphi & & & & & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ \sin \varphi & & & & & & & \cos \varphi & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix}.$$

Evolution Strategies: Variance Adaptation

- Usually the covariance matrix Σ is chromosome-specific. (One chromosome then contains $n + \frac{n(n+1)}{2}$ genes.)
- The mutation of the covariances is executed on the rotation angles, not directly on the entries of the covariance matrix. As a mutation rule

$$\varphi_{ik}^{(\text{new})} = \varphi_{ik}^{(\text{old})} + r \cdot N(0, 1)$$

with $r \approx 0.0873$ ($\approx 5^\circ$) is used.

$N(0, 1)$ is a normally distributed random number computed anew in each step.

- **Disadvantages** of correlated mutation:
 - Many more parameters have to be adapted (than for uncorrelated mutation).
 - The variances and rotation angles have no direct influence on the fitness function; their adaptation is executed rather “on the side”. Hence it may be unclear whether the adaptation of the rotation angles follows the change of the parameters to optimize quickly enough.

Evolution Strategies: Crossover/Recombination

- **Random Selection of Components** from the parents:

$$\begin{array}{l} (x_1, x_2, x_3, \dots, x_{n-1}, x_n) \\ (y_1, y_2, y_3, \dots, y_{n-1}, y_n) \end{array} \Rightarrow (x_1, y_2, y_3, \dots, x_{n-1}, y_n)$$

This approach corresponds to **uniform crossover**.

In principle it is also possible to apply 1-, 2- or n -point crossover.

- **Averaging** (blending, intermediary recombination):

$$\begin{array}{l} (x_1, \dots, x_n) \\ (y_1, \dots, y_n) \end{array} \Rightarrow \frac{1}{2}(x_1 + y_1, \dots, x_n + y_n)$$

- **Attention:** Averaging may cause the risk of **Jenkin's Nightmare**.

(Reminder: *Jenkin's Nightmare* consists in the total disappearance of any diversity in a population.

It is favored by averaging/blending as a crossover operation, because then the genes tend to move towards average values.)

Evolution Strategies: Plus versus Comma Strategies

- Obvious **advantage** of the +-strategy:
 - Due to strict elitism, only improvements can occur.
(The next generation is never worse than the preceding one.)

- **Disadvantages:**

- Danger of getting stuck in local optima.
- For a $(\mu + \lambda)$ evolution strategy with

$$\frac{\mu}{\lambda} \geq \text{“best probability for a successful mutation”} \quad (\approx \frac{1}{5})$$

those chromosomes have a selection advantage, that keep their variance σ^2 as small as possible. However, this means that often mutations are insufficiently large to achieve a “true” improvement (“quasi-stagnation”).

Usual choice of the ratio of μ and λ : around 1:7.

- If for several generation no improvements have been observed, it is often switched temporarily from the +-strategy to the ,-strategy in order to increase the diversity of the population again.

Multi-Criteria Optimization

Multi-Criteria Optimization

- In many everyday situations not a single quantity is to be optimized, but several **different objectives** are to be satisfied to a degree that is as large as possible.
- **Example:** When buying a car, one would like to have
 - low purchase price,
 - low fuel consumption,
 - as much comfort as possible, e.g., air conditioning or electrical windows.
 - ...
- The different objectives one tries to reach are often not independent, but **complementary** or at least **competing**:
They cannot all be satisfied maximally at the same time.
- **Example:** When buying a car,
 - many special equipment items command an extra payment,
 - choosing e.g. air conditioning often requires a somewhat stronger motor and thus a higher price and fuel consumption.

Multi-Criteria Optimization

- **Formal Description:**

Let k criteria be given, each of which is associated with an objective function:

$$f_i : \Omega \rightarrow \mathbb{R}, \quad i = 1, \dots, k$$

- **Simple and Straightforward Solution:**

Combine the k objective functions into a single overall objective function, for example, by forming a weighted sum:

$$f(\omega) = \sum_{i=1}^k w_i f_i(\omega).$$

- **Choosing the Weights:**

- *Sign:* If the overall objective function is to be maximized, the signs of the weights w_i of those objective functions that are to be maximized must be positive, the signs of all other objective functions must be negative.
- *Absolute value:* The absolute values of the weights express the relative importance of the different criteria (also consider the possible range of values!).

Multi-Criteria Optimization

- **Problems** of using a weighted sum of individual objective functions:
 - Already at the start of the search one has to fix what relative importance the different criteria have.
 - It is not always easy to specify the weights in such a way that the preferences between the different criteria are expressed appropriately.
- The problems that occur with a linear combination of objective functions are even more fundamental:
 - In general, we face the important and often occurring problem of **aggregating preference orders**.
 - This problem also occurs, for example, with elections of people. (The candidate preferences of the voters have to be aggregated.)
 - **Arrow's Paradox / Impossibility Theorem** [Arrow 1951]:
There is no choice function that possesses all desirable properties:
universality, independence, monotonicity, non-imposition, non-dictatorship

Multi-Criteria Optimization

- Arrow's impossibility theorems [Arrow 1951] can, in principle, be circumvented by using **scaled preference orders**.
- **However:**
The scaling of the preference orders is an additional degree of freedom. It may be even more difficult to find a proper scaling than to choose the weights of a linear combination.
- **Alternative Approach:**
Try to find (sufficiently) many **Pareto-optimal** solutions.
- An element ω of the search space Ω is called **Pareto-optimal** w.r.t. the objective functions $f_i, i = 1, \dots, k$, if there is *no* element $\omega' \in \Omega$ such that
$$\forall i, 1 \leq i \leq k : f_i(\omega') \geq f_i(\omega) \quad \text{and}$$
$$\exists i, 1 \leq i \leq k : f_i(\omega') > f_i(\omega).$$
- **Intuitively:** The value of the objective function cannot be improved without worsening the value of another.

Multi-Criteria Optimization

- More detailed definition of the notion “Pareto-optimal”:

- An element $\omega_1 \in \Omega$ **dominates** an element $\omega_2 \in \Omega$ if

$$\forall i, 1 \leq i \leq k: f_i(\omega_1) \geq f_i(\omega_2).$$

- An element $\omega_1 \in \Omega$ **strictly dominates** an element $\omega_2 \in \Omega$ if ω_1 dominates ω_2 and

$$\exists i, 1 \leq i \leq k: f_i(\omega_1) > f_i(\omega_2).$$

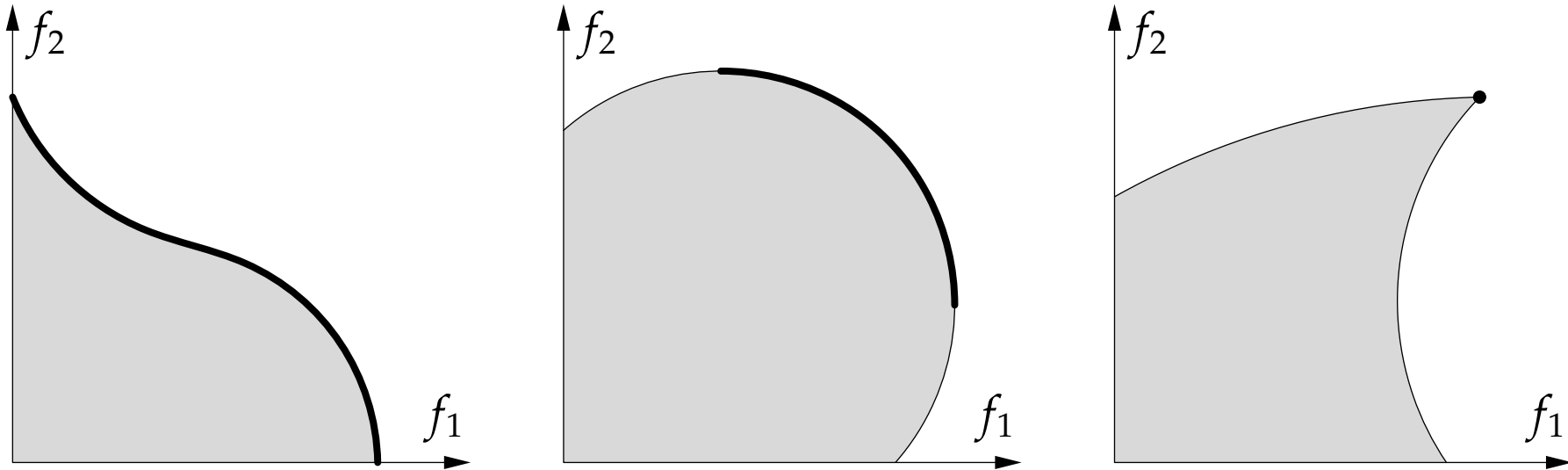
- An element $\omega_1 \in \Omega$ is called **Pareto-optimal** if it is strictly dominated by *no* element $\omega_2 \in \Omega$.

- **Advantage** of searching for Pareto-optimal solutions:

- The objective functions need not be combined; there is no need to specify weights.
- The search for different preferences has to be executed only once, since only afterward one of the found solutions is chosen.

Multi-Criteria Optimization

Illustration of Pareto-optimal Solutions:



- All points of the search space lie in the area shown in gray.
- Pareto-optimal solutions are located on the outlined part of the border.
- Note that, depending on the location of the solution candidates, the Pareto-optimal solution may be uniquely determined (see diagram on the right).

Multi-Criteria Optimization with Evolutionary Algorithms

- **Simplest Approach:**

Use a weighted sum of the individual objective functions as the fitness function.
(This has the mentioned disadvantages.)

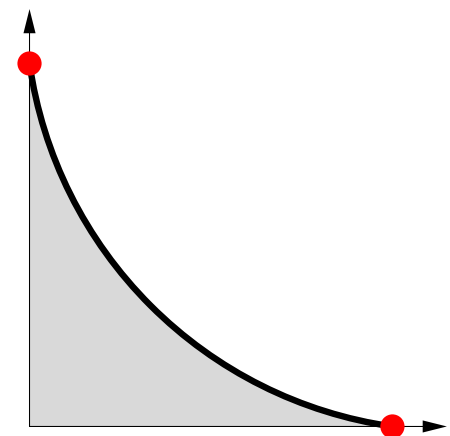
- Remark: This approach yields a Pareto-optimal solution, but only the specific one that is determined by the weights.

- **Natural Alternative:**

Let k criteria be given that are associated with objective functions $f_i, 1 \dots, k$.

$\forall i, 1, \dots, k$: Choose $\frac{\text{popsize}}{k}$ individuals based on the fitness function f_i .

- **Advantage:** simple, low computational overhead.
- **Disadvantage:** Solutions that satisfy all criteria fairly well, but none of them maximally, have a clear selection disadvantage.
- **Therefore:**
The search concentrates on border solutions.



Multi-Criteria Optimization with Evolutionary Algorithms

- **Better Approach:**
Use the notion of dominance for the selection.
- Build a **rank scale** for the individuals of a population:
 - Find all non-dominated solution candidates of the population.
 - Assign the highest rank to these solution candidates and remove them from the population.
 - Repeat determining and removing the non-dominated solution candidates for the subsequent ranks, until the population is empty.
- With the help of the found rank scale, execute **rank selection**.
- Usually this approach is combined with **niche techniques** (in order to distinguish between individuals having the same rank).
- **Alternative: tournament selection**
The tournament winner is determined by the notion of dominance and possibly niche techniques in addition.

Parallelization of Evolutionary Algorithms

Parallelization of Evolutionary Algorithms

- Evolutionary algorithms are comparatively **expensive optimization methods**, since often
 - a large population (some thousand up to some ten thousand individuals)
 - a large number of generations (some hundred)may be necessary to find a sufficiently good solution.
- This disadvantage may (sometimes) be compensated by a somewhat higher solution quality compared to other methods, but the runtime of an evolutionary algorithms can be inconveniently long.
- Possible solution approach: **Parallelization**, that is, distributing the necessary operations onto multiple processors (cores).
- Questions:
 - **Which steps may be parallelized?**
 - **Which other advantageous techniques may be used when parallelizing?**

Parallelization: What can be parallelized?

- **Creation of the Initial Population**

Usually very easy to parallelize, since generally the chromosomes of the initial population are chosen randomly and independently of each other.

Trying to avoid duplicates may impede parallelization.

However, parallelizing this step is of minor importance, because the initial population is created only once.

- **Evaluation of the Chromosomes**

Can be parallelized without problems, since the chromosomes are usually evaluated independently of each other (that is, the fitness depends only on the chromosome itself).

Even for the prisoner's dilemma chromosome pairs can be processed in parallel.

- **Computing the (relative) Fitness Values**

In order to compute relative fitness values of a rank scale of the chromosomes, the evaluations must be brought together.

Parallelization: What can be parallelized?

- **Selection**

Whether the selection step may be parallelized depends heavily on the employed selection method:

- **Expected Value Model and Elitism**

Both require a global consideration of the population and thus can be parallelized only with difficulties.

- **Roulette Wheel and Rank Selection**

After the relative fitness values or the ranks have been determined (which is difficult to do in parallel) the selection itself is easy to parallelize.

- **Tournament Selection**

Ideal for parallelizing selection, especially for small tournament sizes, since no global information about the population has to be determined: the comparison of fitness values is limited to the individuals of each tournament.

Parallelization: What can be parallelized?

- **Applying Genetic Operators**

Easy to execute in parallel, since only a single individual (mutation) or two chromosomes (crossover) are affected.

Together with tournament selection a steady state evolutionary algorithm is very convenient to parallelize.

- **Termination Criterion**

The simple test whether a user-specified maximum number of generations has been reached, creates no parallelization problems.

Termination criteria like

- the best individual of the population has a certain minimum quality or
- for a certain number of generations the best individual did not (significantly) improve

are less suited for a parallel execution, since they require a global view on the population.

Parallelization: Island Model and Migration

- Even if a selection method is employed that is difficult to parallelize, some parallelization may be achieved by processing several independent populations in parallel.
Each population is seen as inhabiting an island, hence the name **Island model**.
- A pure island model is equivalent to a repeated serial execution of the same evolutionary algorithm.
Usually it yields somewhat worse results than a single run with a (correspondingly) larger population.
- However, one may consider exchanging individuals between the different island populations at certain specified points in time (*not* in every generation).
This case is referred to as **migration**.
- Usually chromosomes from different islands are not recombined directly.
Only after a migration, which relocates some chromosomes to different islands, genetic information from one island is combined with that from another island.

Parallelization: Island Model and Migration

- **Controlling the Migration between Islands**

- **Random Model**

The two islands, between which individuals are to be exchanged, are determined randomly (chosen with equal probability).

In this way arbitrary islands may exchange individuals.

- **Network Model**

The islands are organized in some graph structure.

Individuals may migrate between islands only along the edges of the graph.

The edges, over which individuals are exchanged, are determined randomly.

- **Competition between Islands**

- The evolutionary algorithms that are applied on the different islands, differ from each other (in the procedure and/or in the parameters).

- The population size of an island is increased or reduced based on the average fitness of the individuals.

However, there is a certain minimum population size for an island.

Parallelization: Cellular Evolutionary Algorithms

(also called “isolation by distance”)

- A number of processors or workers are organized into a (rectangular) grid. This grid usually covers the surface of a torus (donut-shaped object).
- Selection and crossover are restricted to processors (workers) that are neighbors in the grid (that are connected by an edge). Mutation is limited to individual processors.
- Example: Every processor manages a single chromosome.
 - **Selection:** A processor chooses the best chromosome of its (four) neighbor processors or of these chromosomes randomly according to their fitness.
 - **Crossover:** The processor executes a crossover of the chosen neighbor chromosome and its own chromosome or it mutates its own chromosome. Of the two offspring individuals that result from a crossover, or of the parent and child of a mutation, it keeps the better individual.
- In this way groups of neighboring processors are formed that manage similar chromosomes. This mitigates the usually destructive effect of crossover.

Parallelization: Combination with Random Ascent

- **Combination** of evolutionary algorithms **with hill climbing/random ascent:**

Each individual executes local hill climbing/random ascent, that is,

- if a mutation improves the fitness, the parent is replaced,
- if a mutation worsens the fitness, the parent is kept.

This hill climbing/random ascent is easy to parallelize.

- Individuals search for a crossover partner in their neighborhood.
(For this, a definition of the distance of two individuals is needed — see also niche techniques, for example, *power law sharing*.)
- The offspring individuals (crossover products) execute (local) hill climbing/random ascent.
- The individuals of the next generation are chosen according to the **“local” elite principle**, that is, the two best among the two parents and the two optimized offspring individuals are kept.

Swarm-based Optimization Methods

Swarm Intelligence

picture not available in online version

“Where are we going?” — “Don’t know, swarm intelligence!”

The Wisdom of the Crowd

picture not available in online version

What many approve of, can only be good.

Swarm- and Population-based Optimization

Swarm Intelligence / Collective Intelligence

Area of Artificial Intelligence that develops intelligent multi-agent systems.

Inspired by the behavior of certain kinds of animals, especially

- social insects (e.g. ants, termites, bees etc.) and
- animals that live in swarms (or flock(ing)s, schools, herds, packs etc.) (e.g. fish, birds, ungulates, wolves, etc.).

Animals of such species can accomplish fairly complex tasks by working together (finding food, finding paths, building nests etc.).

Fundamental Ideas

- Individuals are usually fairly simple and thus have only limited capabilities,
- activity coordination is without central control, but by self-organization,
- exchange of information between individuals leads to cooperation.

Different methods can be classified according to how information is exchanged.

Swarm- and Population-based Optimization

- **Evolutionary Algorithms**

- Biological archetype: evolution of living creatures
- Information exchange: by recombining genotypes
- Each individual is a solution candidate

- **Particle Swarm Optimization**

- Biological archetype: Fish and birds looking for food
- Information exchange: by a simple aggregation of individual solutions
- Each individual is a solution candidate

- **Ant Colony Algorithms**

- Biological archetype: ants searching for paths to food sources
- Information exchange: by modifying the environment (stigmergy; “extended phenotype” according to Dawkins)
- Individuals construct solution candidates

Particle Swarm Optimization

Particle Swarm Optimization

pictures not available in online version

- Fish and birds search in swarms (schools, flocks) for abundant food sources.
- Apart from individual exploration (cognitive component) members of a swarm are also guided by other swarm members in their vicinity (social component).
- Living in a swarm usually also serves the purpose to protect the individuals against predators.

Particle Swarm Optimization

Particle Swarm Optimization was first suggested as an optimization method by [Kennedy and Eberhart 1995].

- Particle swarm optimization can be seen as a method that combines elements of trajectory-based search (for example gradient-based methods) and elements of population-based search (for example, evolutionary algorithms).
- **Approach:** Instead of only a single current solution candidate (like, for example, in hill climbing etc.) a “swarm” of solution candidates is employed.
- **Motivation:** Behavior of e.g. schools of fish when foraging (searching for food): Random fanning out, but also always return to the swarm. Information exchange between animals of the swarm.
- **Prerequisite:** The search space is real-valued, that is, $\Omega \subseteq \mathbb{R}^n$ and thus the function to optimize (w.l.o.g.: to maximize) is $f : \mathbb{R}^n \rightarrow \mathbb{R}$.
- **Procedure:** Each solution candidate is seen as a “particle” which possesses a location \vec{x}_i in the search space and a velocity $\vec{v}_i, i = 1, \dots, m$.

Particle Swarm Optimization

- **Update Formulae** for location and velocity of the i -th particle:

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t)$$

$$\vec{v}_i(t+1) = \alpha \cdot \vec{v}_i(t) + \beta_1 \cdot (\vec{x}_i^{(\text{local})}(t) - \vec{x}_i(t)) + \beta_2 \cdot (\vec{x}^{(\text{global})}(t) - \vec{x}_i(t))$$

- $\vec{x}_i^{(\text{local})}$ is the **local memory** of the individual (particle).

It is the best location in the search space that the particle has visited up to now,

$$\vec{x}_i^{(\text{local})}(t) = \vec{x}_i(\operatorname{argmax}_{u=1}^t f(\vec{x}_i(u))).$$

- $\vec{x}^{(\text{global})}$ is the **global memory** of the swarm.

It is the best location in the search space that any individual of the swarm has visited up to now (best solution candidate found up to now),

$$\vec{x}^{(\text{global})}(t) = \vec{x}_j^{(\text{local})}(t) \quad \text{where} \quad j = \operatorname{argmax}_{i=1}^m f(\vec{x}_i^{(\text{local})}(t)).$$

- Choice of parameters: β_1, β_2 randomly in each step, α decreasing over time.

Particle Swarm Optimization: Pseudo-Code

```
procedure pso; (* particle swarm optimization *)  
for each particle  $i$  do begin (* initialize the location of each particle *)  
    choose random  $\vec{x}_i; \vec{v}_i = \vec{0};$  (* randomly in the search space *)  
end;  
repeat (* move the particles of the swarm *)  
    for each particle  $i$  do begin (* traverse the particles *)  
         $y := f(\vec{x}_i);$  (* compute function at particle location *)  
        if  $y \geq f(\vec{x}_i^{(\text{local})})$  then  $\vec{x}_i^{(\text{local})} := \vec{x}_i;$   
        if  $y \geq f(\vec{x}^{(\text{global})})$  then  $\vec{x}^{(\text{global})} := \vec{x}_i;$   
    end; (* update local/global memory *)  
    for each particle  $i$  do begin (* traverse the particles again *)  
         $\vec{v}_i := \alpha \cdot \vec{v}_i + \beta_1 \cdot (\vec{x}_i^{(\text{local})} - \vec{x}_i) + \beta_2 \cdot (\vec{x}^{(\text{global})} - \vec{x}_i);$   
         $\vec{x}_i := \vec{x}_i + \vec{v}_i;$  (* update the velocity and *)  
    end; (* the location of each particle *)  
until termination criterion is met;
```

Particle Swarm Optimization: Extensions

- **Restricted Search Space**

If the search space is a proper subset of the \mathbb{R}^n (e.g. a hyperbox $[a, b]^n$), the particles are reflected at the boundaries of the search space.

- **Local Environment of a Particle**

Instead of the global memory of the swarm, the best local memory of only a part of the swarm is used, e.g., of those particles that are in the vicinity of the particle that is to be updated.

- **Automatic Parameter Adaptation**

E.g. adaptation of the swarm size: particles the local memory of which is significantly worse than that of particles in their vicinity are discarded.

- **Diversity Control**

Tries to prevent premature convergence to suboptimal solutions. This may be achieved, for example, by adding a random component to the update of the velocity, which increases the (location) diversity of the swarm.

Particle Swarm Optimization

PSODemo

File Actions Help

step: 29, best: $f(-0.100213, -0.001687) = 0.956989$

Select Function...

Function to optimize: eggbox 2

OK Apply Cancel

Create Particle Swarm...

Number of particles: 30

Tail length (history): 2

Seed for random numbers: 0

If the seed for the pseudo-random number generator is set to zero, the system time will be used instead.

Acceleration factor: 1

Initial deceleration factor: 1

Deceleration factor decay: 0.03

OK Apply Close

Run Optimization...

Number of epochs: 5,000

Delay between epochs: 100

OK Apply Close

Ant Colony Optimization

Ant Colony Optimization

pictures not available in online version

- Because found food has to be transported to the nest in order to feed offspring, ants usually construct “transport roads”.
- For this purpose the ants mark the paths to food sources with fragrant compounds (pheromones), so that other ants of the colony may also find the food sources.
- The paths lengths to the food sources are approximately minimized.

Ant Colony Optimization

Ant Colony Optimization was introduced by
[Dorigo *et al.* 1996, Dorigo and Gambardella 1997]

- **Motivation:**

Ants of certain species find shortest paths to food sources by placing and following pheromone markers (“scent marks”).

- Intuitively: Shorter paths receive more pheromone per unit of time.
- Paths are chosen randomly according to the available amount of pheromone. (The more pheromone is on a path, the more probable it is that it is chosen.)
- The amount of deposited pheromone can depend on the quality and the amount of the found food.

- **Fundamental Principle: Stigmergy**

In order to find paths, ants communicate indirectly via pheromone deposits.

Stigmergy (indirect communication by modifying the environment) allows for globally adapted behavior on the basis of locally available information.

Ant Colony Optimization

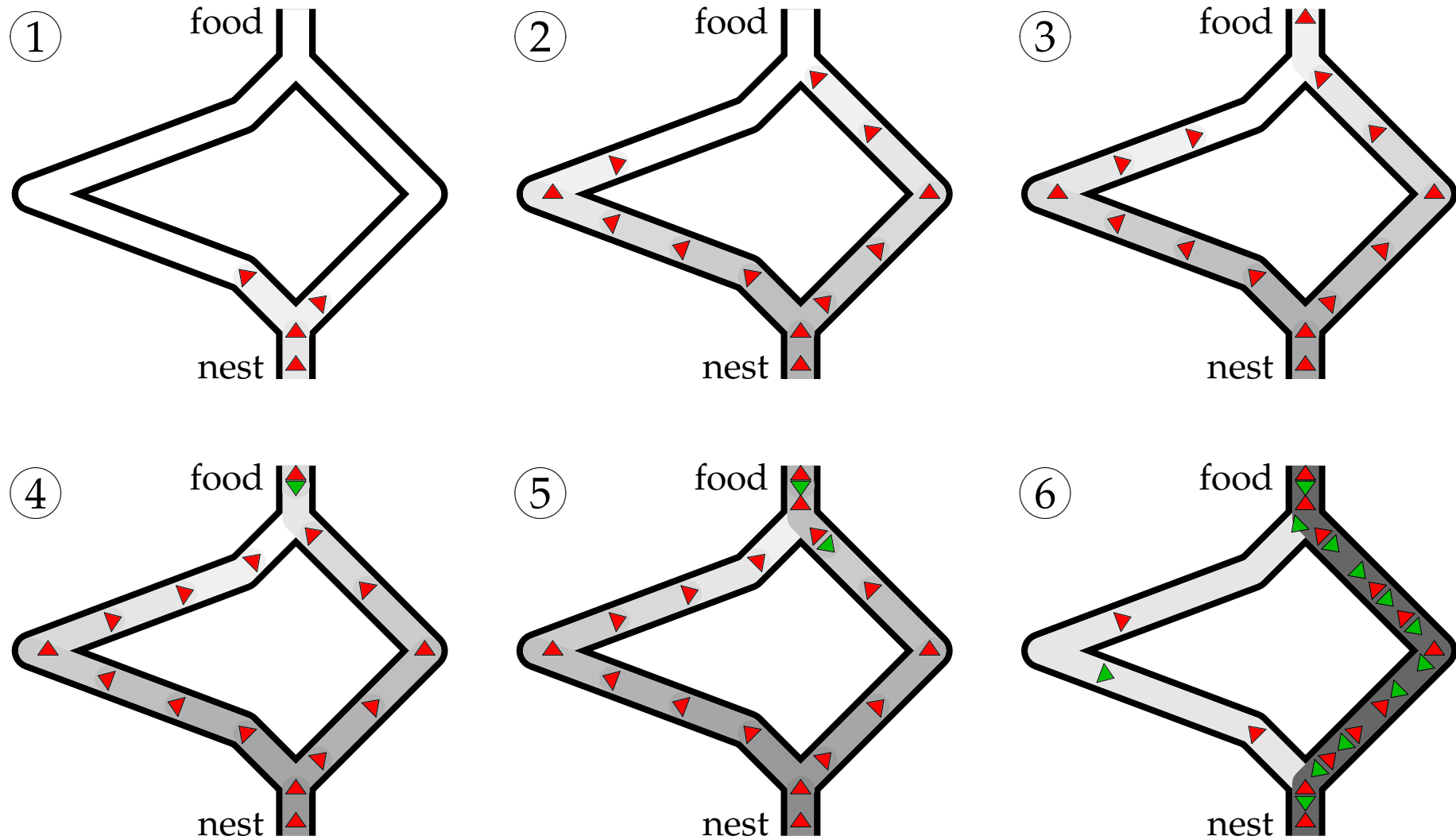
Double Bridge Experiment [Goss *et al.* 1989]

- An ant nest and a food source are connected by a double bridge. The two branches of the bridge have different length.
- The experiment was conducted with the Argentinian ant *Iridomyrmex Humilis*. This species of ants is practically blind (like almost all other species of ants). (Hence the ants cannot see which is the shorter branch/path.)
- In most trials almost all ants used the shorter path after only a few minutes.

Explanation

- On the shorter path the ants reach the food source faster. The end of the shorter path thus receives more pheromone (at the beginning).
- Due to this pheromone difference, on the way back the shorter path is chosen with a higher probability. This leads to an amplification of the pheromone difference.

Double Bridge Experiment



Double Bridge Experiment

- **Principle:** The shorter path is systematically amplified (autocatalysis):

more pheromone on path \rightleftarrows more ants choose path

- **Note:** The shorter path is found only, because the ants deposit pheromone on both the way to the food source *and* on the way back to the next.
- If, for example, the ants deposit pheromone only on the way to the food source:
 - On the way to the food source neither path can be preferred, because no pheromone difference exists or is created systematically.
 - At the rejoinder point of the bridge, the ratio of the pheromone deposits diminishes over time and finally disappears almost completely.
 - By random fluctuations in the choice of the path, the search may converge nevertheless (but randomly!) to one of the two branches of the bridge.
- Analogous arguments (symmetric situation) can be put forward if pheromone is deposited only on the way back to the nest.

Double Bridge Experiment

- **Note:** The shorter path is found, because at the beginning both branches of the bridge are available and both branches are free of pheromone.
The end of the shorter path is reached earlier by more ants.
This leads to different amounts of pheromone on the two paths, which initiates a self-amplifying process.
- **Question:** What happens if due to a change of the environment a new path becomes available that is shorter than the current one?
Do the ants change to the shorter path?
- **Answer:** *No!* [Goss *et al.* 1989]
If a path is established, it is maintained.
- Demonstration by a second double bridge experiment, in which at the start only the longer branch is available and the shorter branch is added only later.
The majority of the ants continue to use the longer path.
Only in rare instances they switch to the shorter path.

Natural and Artificial Ants

- Abstract the situation described by the function to optimize into a search for a best path in a weighted graph.
- **Problem:** Cycles that amplify themselves.
If an ant traverses a cycle, the deposited pheromone creates a tendency to traverse the cycle again.
Solution: Deposit pheromone only after a complete path has been constructed.
Cycles are removed before pheromone is deposited.
- **Problem:** The search may focus on solutions candidates constructed at the beginning of the procedure (premature convergence).
Solution: Evaporation of pheromone (has only minor influence in nature)
- **Useful Extensions/Improvements**
 - Deposited amount of pheromone depends on the solution quality.
 - Introducing heuristics into the choice of edges (e.g. edge weight).

Ant Colony Optimization

- **Prerequisite**
 - The problem to solve is a combinatorial optimization problem.
 - There is a constructive method to generate solution candidates.
- **Procedure:** Solution candidates are constructed with a sequence of random decisions, each of which extends a partial solution.
- The sequence of decisions can be seen as a path in a decision graph (also: construction graph).
- The ants are to explore paths through the decision graph and (hopefully) they may find the best (shortest, cheapest) path.
- The ants mark the used edges of the graph with pheromone. In this way other ants are guided to good solution candidates.
- Pheromone evaporates in each iteration, so that, once deposited, it does not influence the system arbitrarily long (“forgetting” of outdated information).

Ant Colony Optimization

Application to the Traveling Salesman Problem

- Representation of the problem by an $n \times n$ matrix $\mathbf{D} = (d_{ij})_{1 \leq i, j \leq n}$.
 n is the number of cities and d_{ij} the distance between cities i and j .
(Note: \mathbf{D} may be asymmetric, but $\forall i; 1 \leq i \leq n : d_{ii} = 0$.)
- Representation of the pheromone information as a $n \times n$ matrix $\mathbf{\Phi} = (\phi_{ij})_{1 \leq i, j \leq n}$.
A pheromone value ϕ_{ij} , $i \neq j$, states how desirable it is to visit city j immediately after city i . (the ϕ_{ii} are not needed.)
The matrix need not be (forced to be) symmetric.
- All matrix elements ϕ_{ij} are initialized with the same small value.
(At the beginning all edges carry the same amount of pheromone.)
- Ants traverse (with the help of the pheromones) Hamiltonian cycles.
They mark the edges of the traversed Hamiltonian cycle with pheromone, where the amount of deposited pheromone corresponds to the quality of the solution candidate.

Construction of Solution Candidates

Each ant has as a “memory” a set C that contains the indices of the cities that have not been visited yet. Each visited city is deleted from this set.

(This memory does not exist in the biological archetype!)

1. An ant is placed at a randomly chosen city.
This city is the start of the tour.
2. The ant chooses a city that has not yet been visited and moves to this city.
In city i the ant chooses the (unvisited) city j with the probability

$$p_{ij} = \frac{\phi_{ij}}{\sum_{k \in C} \phi_{ik}},$$

where C is the set of indices of the not yet visited cities and ϕ_{ij} is the amount of pheromone on the connection from city i to city j .

3. Repeat step 2 until all cities have been visited.

Pheromone Update

1. Evaporation

All pheromone values are reduced by a fraction η (called *evaporation*):

$$\forall i, j; 1 \leq i, j \leq n: \quad \phi_{ij} := (1 - \eta) \cdot \phi_{ij}$$

2. Amplification of Constructed Solutions

An additional amount of pheromone is deposited on the edges of the constructed solution candidates, which depends on the solution quality:

$$\forall \pi \in \Pi_t: \quad \phi_{\pi(i)\pi((i \bmod n)+1)} := \phi_{\pi(i)\pi((i \bmod n)+1)} + Q(\pi)$$

Π_t is the set of tours (permutations) constructed in step t .

As a quality function one may use, for example, the inverse tour length:

$$Q(\pi) = c \cdot \left(\sum_{i=1}^n d_{\pi(i)\pi((i \bmod n)+1)} \right)^{-1}$$

Intuitively:

The better the solution, the more pheromone is deposited onto its edges.

Traveling Salesman Problem: Pseudo-Code

```
procedure aco_tsp;           (* ant colony optimization for *)
                             (* the traveling salesman problem *)
initialize pheromone;        (* initialize all matrix elements  $\phi_{ij}$ , *)
repeat                      (*  $1 \leq i, j \leq n$ , to a small value  $\varepsilon$  *)
  for each ant do          (* construct solution candidates *)
     $C := \{1, \dots, n\};$     (* set of cities to visit *)
    choose  $i \in C;$          (* choose city to start at and *)
     $C := C - \{i\};$         (* remove it from the unvisited cities *)
    while  $C \neq \emptyset$  do begin (* as long as there are unvisited cities *)
      choose  $j \in C$  with probability  $p_{ij};$ 
       $C := C - \{i\};$       (* choose the next city of the tour, *)
       $i := j;$             (* delete it from the unvisited cities, *)
    end;                  (* and move to the chosen city *)
  endfor;                 (* update matrix  $\Phi$  according to *)
  update pheromone;         (* the solution quality *)
until termination criterion is met;
```

Extensions and Alternatives

- **Prefer Cities that are Close** (analogous to the nearest neighbor heuristic)
Move from city i to city j with probability

$$p_{ij} = \frac{\phi_{ij}^{\alpha} \tau_{ij}^{\beta}}{\sum_{k \in C} \phi_{ik}^{\alpha} \tau_{ik}^{\beta}}$$

where C is the set of indices of unvisited cities and $\tau_{ij} = d_{ij}^{-1}$.

- **Tendency to Choose the Best Edge** (greedy)
With probability p_{exploit} move from city i to city j_{best} with

$$j_{\text{best}} = \operatorname{argmax}_{j \in C} \phi_{ij} \quad \text{or} \quad j_{\text{best}} = \operatorname{argmax}_{j \in C} \phi_{ij}^{\alpha} \tau_{ij}^{\beta}$$

and use the probabilities p_{ij} with probability $1 - p_{\text{exploit}}$.

- **Amplify Best Known Tour** (elitist)
Deposit additional pheromone on the best tour found up to now.
This may be stated, for example, as a fraction of ants that traverse this best tour (in addition to ants constructing new tours).

Extensions and Alternatives

- **Rank-based Update**

Deposit pheromone only on the edges of the best m solutions of the preceding iteration (and possibly on the best tour found so far).

The amount of pheromone depends on the rank of the solution.

- **Strict Elitism**

- Deposit pheromone only on the edges of best solution of last iteration.

- Deposit pheromone only on the edges of best solution found so far.

- **Minimal/maximal Amount of Pheromone**

Limit the amount of pheromone on an edge to a maximum/minimum.

⇒ Minimal/maximal probability for the choice of an edge

⇒ better exploration of the search space, possibly slower/worse convergence

- **Limited Evaporation**

Pheromone only evaporates from edges that have been traversed in the last iteration.

⇒ better exploration of the search space

Local Improvements of the Tour

- A combination with **local solution improvement** is often advantageous: Before the pheromone is updated, a constructed tour is locally optimized, by testing simply modifications for improvements.
- Local optimization approaches may use, for example, the following operations:
 - **Recombination after Removing Two Edges** (2-opt) corresponds to a “reversal” of a partial tour
 - **Recombination after Removing Three Edges** (3-opt) corresponds to a “reversal” of two partial tours
 - **Restricted Recombination** (2.5-opt)
 - **Exchange of Consecutive Cities**
 - **Permutation of Consecutive Triplets**
- “Expensive” locale optimizations should be applied only to the best solution candidate found in the search up to now or only to the best solution candidate of the current iteration.

Ant Colony Optimization

The screenshot displays the ACODemo software interface. The main window shows a Traveling Salesman Problem (TSP) visualization with a set of vertices and edges. A red path highlights the current solution, and a grey path shows the previous iteration. The interface includes a menu bar (File, Actions, Help) and a status bar at the bottom left that reads "ACODemo is up and running."

Three dialog boxes are open on the right side of the window:

- Generate Random TSP...**
 - Number of vertices: 30
 - Seed for random numbers: 0
 - If the seed for the pseudo-random number generator is set to zero, the system time will be used instead.
 - Buttons: Ok, Apply, Close
- Create Ant Colony...**
 - Number of ants: 30
 - Seed for random numbers: 0
 - If the seed for the pseudo-random number generator is set to zero, the system time will be used instead.
 - Initial pheromone: 0
 - Exploitation probability: 0.2
 - Pheromone trail weight: 1
 - Inverse distance weight: 1
 - Evaporation fraction: 0.1
 - Trail laying exponent: 1
 - Elite enhancement: 0.1
 - Buttons: Ok, Apply, Close
- Run Optimization...**
 - Number of epochs: 5,000
 - Delay between epochs: 200
 - Buttons: Ok, Apply, Close

General Application to Optimization Problems

- **Fundamental Principle**

Formulate the problem as a search in a (decision) graph.
Solution candidates must be representable as edge sets.

(Note: Solution candidates need not be paths, though!)

- **General Description**

For specific problems, the following will be stated:

- Vertices and edges of the decision/construction graph
 - Constraints that have to be satisfied
 - Meaning of the pheromone on the edges (and possibly also on vertices)
 - Heuristically usable auxiliary information
 - Construction of a solution candidate
- The algorithmic procedure is essentially analogous to the procedure employed for the traveling salesman problem.

General Application to Optimization Problems

Traveling Salesman Problem

- *Vertices and edges of the decision/construction graph:*
the cities to visit and their connections,
the connections are weighted (distance, time, costs)
- *Constraints that have to be satisfied:*
each city must be visited exactly once
- *Meaning of the pheromone on the edges:*
how desirable it is to visit city j after city i
- *Heuristically usable auxiliary information:*
Distance of the cities; prefer cities that are “close”
- *Construction of a solution candidate:*
starting from a randomly chosen city, each steps moves to another,
yet unvisited city

General Application to Optimization Problems

Generalized Assignment Problem

A set of n tasks have to be assigned to m workers (people, machines),
The objective is to minimize the sum of assignment costs d_{ij} respecting
certain maximum capacities ρ_j for given capacity costs r_{ij} , $1 \leq i \leq n$, $1 \leq j \leq m$.

- Each task and each worker is a vertex of the construction graph, the edges carry the assignment costs d_{ij} .
- Each task has to be assigned to exactly one worker; the capacities of the workers may not be exceeded.
- The pheromone values on the edges describe how desirable the assignment of a task to a worker is.
- Inverse absolute or relative capacity costs or inverse assignment costs may be used as heuristics.
- Edges are selected incrementally (they need not form a path). Edges of already assigned tasks are skipped (maximum node degree 1). Solution candidates violating the constraints are penalized (costs are increased).

General Application to Optimization Problems

Knapsack Problem

From n objects with associated values w_i , weights g_i , and volumes v_i etc., $1 \leq i \leq n$, a subset of maximal value is to be selected, so that maximum values for weight, volume etc. are satisfied.

- Each object is a vertex of the construction graph.
The vertices carry the object values w_i ; edges are not needed.
- Maximum values for weight, volume, etc. have to be satisfied.
(The knapsack must not become too heavy, too large etc.)
- Pheromone values are assigned only to vertices.
They describe how desirable the choice of the object is.
- Heuristics: ratio of object value to relative weight, volume etc.
The ratios may take the maximum values into account.
- Vertices are selected incrementally, ensuring in each step that the maximum values for weight, volume etc. are not exceeded.

Convergence of the Search

How likely is it that Ant Colony Optimization finds a solution?

- Consider a “standard procedure” with the following properties:
 - Evaporation of pheromone with a constant factor from all edges.
 - Pheromone is deposited only the edges of the best solution candidate found so far (strict elitism).
 - There is a lower bound ϕ_{\min} for the amount of pheromone on the edges, which the pheromone value may not fall below.
- This standard procedure converges in probability to the solution. (That is, with the number of computed steps going to infinity, the probability that the solution is found approaches 1.)
- If the lower bound ϕ_{\min} for the amount of pheromone is reduced “sufficiently slowly” towards 0 ($\phi_{\min} = \frac{c}{\ln(t+1)}$ with the step number t and some constant c), it can be shown that for the number of steps going to infinity each ant of the colony constructs the solution with a probability that approaches 1.

Summary: Swarm-Based Optimization Methods

- Swarm- and population based algorithms are **Heuristics to Solve Optimization Problems**.
The purpose is to find approximate solutions.
- It is tried to mitigate the **Problem of Local Optima** (by better exploration of the search space).
- A core ingredient is the **information exchange** between the individuals.
Depending on the principle, different algorithm types may be distinguished.
- **Particle Swarm Optimization**
 - Optimization of a function with real-valued arguments.
 - Information exchange by taking neighbors into account.
- **Ant Colony Optimization**
 - Finding optimum paths (abstractly: in a decision graph).
 - Information exchange by modifying the environment (stigmergy).

Evolutionary Algorithms: Example Applications

1. Flight Route Optimization: ROGENA

Problemstellung: Flugroutenplanung

- Flugzeuge bewegen sich im Luftraum normalerweise
 - auf Standardrouten zwischen den Flughäfen,
 - mit einem vorgeschriebenen Mindestabstand auf der gleichen Route,
 - abhängig von ihrer Flugrichtung auf unterschiedlichen Höhen.
 - **Vorteile** dieser Lösung:
 - einfache Regeln/Vorschriften für den Flugverkehr,
 - leichte Kontrolle der Flugrouten durch die Fluglotsen,
 - große Sicherheit im Flugverkehr (i.w. Kollisionsvermeidung).
 - **Nachteile** dieser Lösung:
 - nur ein relativ kleiner Teil des Luftraums wird genutzt,
 - die Flugzeugdichte in der Nähe von Flughäfen ist relativ eng begrenzt.
- Diese Lösung ist dem steigenden Flugverkehr nicht mehr angemessen.

Luftraum in der Umgebung von Frankfurt

picture not available in online version

Lösungsidee: Fluglotsenunterstützung

- Der Luftraum zwischen den Standardrouten muß genutzt werden, um für zukünftige Luftverkehrssteigerungen gewappnet zu sein.
- **Problem** dieses Ansatzes:
 - Sich freier bewegende Flugzeuge sind schwerer zu koordinieren.
 - Dadurch steigt die Arbeitsbelastung für die Fluglotsen.
- **Aufgabenstellung:** Entwicklung eines Unterstützungswerkzeugs, das die Planung sicherer und effizienter Routen zwischen Eintritts- und Austrittspunkt in dem vom Lotsen kontrollierten Teil des Luftraumes übernimmt.
- **Vorgegebene Rahmenbedingungen:**
 - Eintrittszeitpunkt, Ein- und Austrittsort im kontrollierten Bereich,
 - Flugeigenschaften der kontrollierten Flugzeuge,
 - Bewegung weiterer Flugzeuge (Kollisionsvermeidung),
 - Sperrgebiete (Bebauung, militärische Nutzung, schlechtes Wetter).

Lösungsansatz mit genetischen Algorithms

- Der Suchraum für mögliche Flugrouten ist sehr groß; eine analytische Konstruktionsmethode ist schwer zu finden.
→ Suche mit genetischen Algorithms
- **ROGENA** (free ROuting with GENetic Algorithms) [Gerdes 1994, 1995]
- Die notwendigen Daten und Berechnungsformeln für die Beschreibung der Flugeigenschaften und des Flugverhaltens von Flugzeugen wurden der BADA-Datenbank der EUROCONTROL entnommen [Byrne 1995].
- Es wurde ein Ausschnitt von 200×200 nautischen Meilen (NM) mit einer Höhe von 0 bis 10000 Fuß (ft) aus dem Luftraum betrachtet.
- Mindestabstände zwischen Flugzeugen: standardmäßig 5 NM, beim Endanflug zwischen 2.5 und 6 NM in Abhängigkeit vom Gewicht der beiden Flugzeuge.
- Der genetische Algorithmus beginnt erst bei Flugroutenkonflikten (Unterschreitung der Sicherheitsabstände) zu arbeiten.

ROGENA: Kodierung der Lösungskandidaten

- Eine Flugroute wird als Sequenz von Linienstücken dargestellt (vgl. Übungsaufgabe zum Finden eines Weges durch ein Gebiet).
- Die Chromosomen haben variable Länge (variable Anzahl von Genen).

- Jedes Gen stellt einen zu überfliegenden Punkt dar; zusätzlich wird die Überfluggeschwindigkeit angegeben:

	x	y	z	v
Gen 1	86.2	15.8	1.65	258
Gen 2	88.9	24.7	1.31	252
⋮	⋮	⋮	⋯	⋮
Gen k	105.0	98.0	0.00	120

- Es müssen Nebenbedingungen eingehalten werden, z.B.
 - monoton fallende Flugroute (da Landeanflüge modelliert werden),
 - maximale Beschleunigung/Verzögerung zwischen Punkten,
 - minimaler Winkel zwischen Linienstücken (Kurvenradius) etc.

ROGENA: Fitnessfunktion

- Die Fitnessfunktion berücksichtigt folgende Eigenschaften einer Flugroute:
 - einzuhaltender Sicherheitsabstand zu anderen Flugzeugen,
 - kein Durchfliegen von gesperrtem Luftraum,
 - Länge der Flugroute bis zur Landebahn,
 - Pünktlichkeit des Fluges (planmäßige Ankunft),
 - möglichst geringe Abweichungen von der optimalen Sinkrate (berechnet nach BADA-Datenbank),
 - keine zu spitzen Winkel zwischen den Linienstücken, um Abweichungen von der tatsächlichen Flugbahn klein zu halten.
- Diese Eigenschaften gehen mit Gewichtungsfaktoren versehen in die Fitnessfunktion ein. Die Fitnessfunktion ist zu minimieren.
- Ein Benutzer kann die Gewichtungsfaktoren für die einzelnen Eigenschaften über Schieberegler beeinflussen.

ROGENA: Ablauf des genetischen Algorithm

- **Ziel:** Erzeugen einer sicheren und effizienten Flugroute für ein neu in den kontrollierten Luftraum eintretendes Flugzeug.
- **Basis:** modifizierte Form eines genetischen Algorithm (ein Chromosom wird entweder Crossover oder Mutation unterworfen)
- **Populationsgröße:** 60 Individuen/Chromosomen
- **Initialisierung der Anfangspopulation:**
Durch wiederholte zufällige Veränderung der Standardflugroute.
- **Selektionsverfahren:**
Im wesentlichen Glücksradauswahl, wobei aber Chromosomen, deren Fitness über einem Schwellenwert liegt, nicht in die Berechnung der Selektionswahrscheinlichkeit eingehen (entspricht der Forderung einer Mindestgüte).
Der Schwellenwert wird im Laufe der Generationen gesenkt, und zwar in Abhängigkeit vom Fitnessdurchschnitt der Population.
(Kombination aus Glücksradauswahl und Sintflutalgorithm)

ROGENA: Anwendung genetischer Operatoren

- 20 Chromosomen werden unverändert übernommen, darunter die 5 besten Chromosomen (Elitismus).
- 20 Chromosomen werden einem **2-Punkt-Crossover** unterworfen.
- 20 Chromosomen werden einem speziellen **Mutationsoperator** unterworfen:
 - entweder völlig zufällige Koordinatenänderung (globale Suche)
 - oder mittlere Veränderung in einen Punkt “in der Nähe”
 - oder kleine Veränderung in einen Punkt “in der Nähe” (Zufallsaufstieg).
 - Zusätzlich Veränderung der Genanzahl (mit geringer Wahrscheinlichkeit). Der Lösch- oder Einfügeort im Chromosom wird zufällig bestimmt, der neue Punkt zufällig in der Nähe seiner Nachbarn initialisiert.
- **Reparaturmechanismen:**

Eine Abfolge von Sink- und Steigvorgängen ist unökonomisch, ebenso eine Abfolge von Beschleunigungs- und Bremsvorgängen.

ROGENA: 2-Punkt-Crossover

picture not available in online version

Diese Form des Crossover wurde gewählt, um Teile von Routen auszutauschen und auf diese Weise nützliche Teile verschiedener Routen zu kombinieren.

ROGENA: Benutzeroberfläche

picture not available in online version

ROGENA: Testläufe und Ergebnisse

- **Benutzeroberfläche:**

- Originalroute in rot (direkte Verbindung von Eintrittsort und Zielpunkt)
 - beste aktuelle Route in grün
 - alternative Routen in grau
 - Höhendigramm der Route in getrenntem Fenster
 - Links: Zeitleiter mit Landezeiten aller Flugzeuge
- Es wurden mehrere **Simulationsläufe** mit realen Daten durchgeführt (beschreiben Eintrittsort/-zeitpunkt und tatsächliche Flugbahn).
 - Die von ROGENA erzeugten Flugrouten sind deutlich kürzer als die tatsächlichen, ohne daß dadurch Konflikte mit anderen Flugzeugen erzeugt wurden.
 - Wirkungsvolle Unterstützung eines Fluglotsen bei der Koordinierung, die Verkürzung der Routen ermöglicht einen höheren Flugzeugdurchsatz.

Anwendungsbeispiele

2. Erlernen von Fuzzy-Reglern

Kurzeinführung Fuzzy-Theorie

- **Klassische Logik:** nur Wahrheitswerte *wahr* und *falsch*.
Klassische Mengenlehre: entweder *ist Element* oder *ist nicht Element*.
- Die Zweiwertigkeit der klassischen Theorien ist oft nicht angemessen.

Beispiel zur Illustration: **Sorites-Paradoxon** (griech. *sorites*: Haufen)

- Eine Milliarde Sandkörner sind ein Sandhaufen. (*wahr*)
- Wenn man von einem Sandhaufen ein Sandkorn entfernt, bleibt ein Sandhaufen übrig. (*wahr*)

Es folgt daher:

- 999 999 999 Sandkörner sind ein Sandhaufen. (*wahr*)

Mehrfache Wiederholung des gleichen Schlusses liefert schließlich

- 1 Sandkorn ist ein Sandhaufen. (*falsch!*)

Bei welcher Anzahl Sandkörner ist der Schluß nicht wahrheitsbewahrend?

Kurzeinführung Fuzzy-Theorie

- Offenbar: Es gibt keine genau bestimmte Anzahl Sandkörner, bei der der Schluß auf die nächstkleinere Anzahl falsch ist.
- Problem: Begriffe der natürlichen Sprache (z.B. “Sandhaufen”, “kahlköpfig”, “warm”, “schnell”, “hoher Druck”, “leicht” etc.) sind **vage**.
- Beachte: Vage Begriffe sind zwar *unexakt*, aber trotzdem nicht *unbrauchbar*.
 - Auch für vage Begriffe gibt es Situationen/Objekte, auf die sie *sicher anwendbar* sind und solche auf die sie *sicher nicht anwendbar* sind.
 - Dazwischen liegt eine **Penumbra** (lat. für *Halbschatten*) von Situationen, in denen es unklar ist, ob die Begriffe anwendbar sind, oder in denen sie nur mit Einschränkungen anwendbar sind (“kleiner Sandhaufen”).
 - Die Fuzzy-Theorie versucht, diese Penumbra mathematisch zu modellieren (“weicher Übergang” zwischen *anwendbar* und *nicht anwendbar*).

Fuzzy-Logik

- Die **Fuzzy-Logik** ist eine Erweiterung der klassischen Logik um Zwischenwerte zwischen *wahr* und *falsch*.
- Als Wahrheitswert kann jeder Wert aus dem reellen Intervall $[0, 1]$ auftreten, wobei $0 \hat{=} falsch$ und $1 \hat{=} wahr$.
- Folglich notwendig: **Erweiterung der logischen Operatoren**
 - Negation klassisch: $\neg a$, fuzzy: $\sim a$ Fuzzy-Negation
 - Konjunktion klassisch: $a \wedge b$, fuzzy: $\top(a, b)$ t -Norm
 - Disjunktion klassisch: $a \vee b$, fuzzy: $\perp(a, b)$ t -Konorm
- **Grundprinzipien** der Erweiterung:
 - Für die Extremwerte 0 und 1 sollen sich die Operationen genauso verhalten wie ihre klassischen Vorbilder (Rand-/Eckbedingungen).
 - Für die Zwischenwerte soll das Verhalten monoton sein.
 - Soweit möglich, sollen die Gesetze der klassischen Logik erhalten werden.

Fuzzy-Negationen

Eine **Fuzzy-Negation** ist eine Funktion $\sim: [0, 1] \rightarrow [0, 1]$, die die folgenden Bedingungen erfüllt:

- $\sim 0 = 1$ und $\sim 1 = 0$ (Randbedingungen)
- $\forall a, b \in [0, 1] : a \leq b \Rightarrow \sim a \geq \sim b$ (Monotonie)

Gelten in der zweiten Bedingung statt \leq und \geq sogar die Beziehungen $<$ und $>$, so spricht man von einer *strikten* Negation.

Weitere Bedingungen, die manchmal gestellt werden, sind:

- \sim ist eine stetige Funktion.
- \sim ist *involutiv*, d.h. $\forall a \in [0, 1] : \sim \sim a = a$.

Involutivität entspricht dem klassischen *Gesetz der Identität* $\neg \neg a = a$.

Die obigen Bedingungen legen die Fuzzy-Negation nicht eindeutig fest.

Fuzzy-Negationen

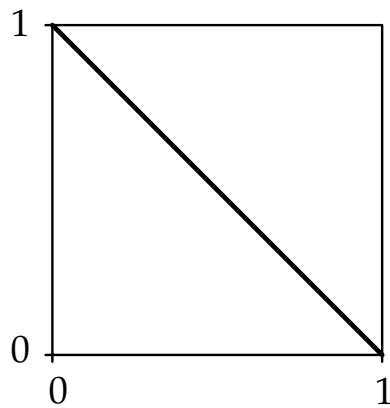
Standardnegation: $\sim a = 1 - a$

Schwellenwertnegation: $\sim(a; \theta) = \begin{cases} 1, & \text{falls } x \leq \theta, \\ 0, & \text{sonst.} \end{cases}$

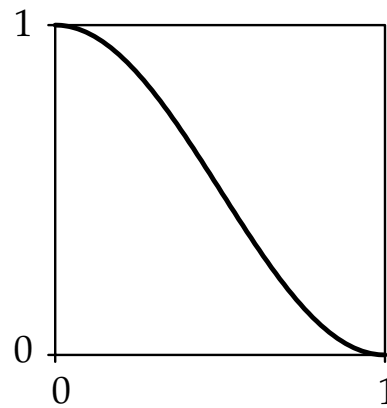
Kosinusnegation: $\sim a = \frac{1}{2}(1 + \cos \pi a)$

Sugeno-Negation: $\sim(a; \lambda) = \frac{1 - a}{1 + \lambda a}$

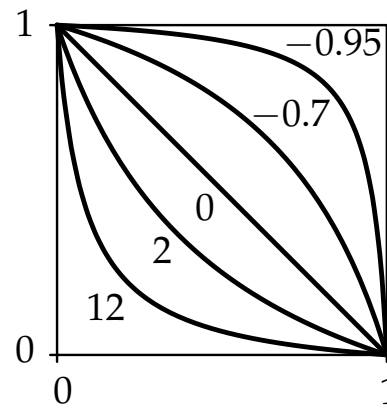
Yager-Negation: $\sim(a; \lambda) = (1 - a^\lambda)^{\frac{1}{\lambda}}$



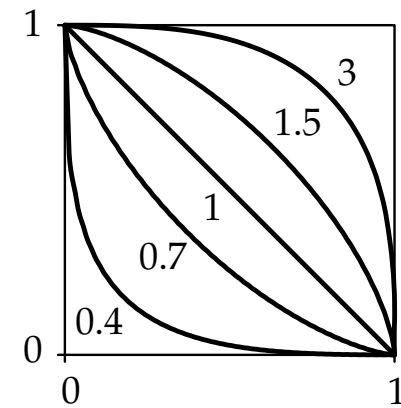
Standard



Kosinus



Sugeno



Yager

t-Normen / Fuzzy-Konjunktionen

Eine **t-Norm** oder **Fuzzy-Konjunktion** ist eine Funktion $\top : [0, 1]^2 \rightarrow [0, 1]$, die die folgenden Bedingungen erfüllt:

- $\forall a \in [0, 1] : \top(a, 1) = a$ (Randbedingung)
- $\forall a, b, c \in [0, 1] : b \leq c \Rightarrow \top(a, b) \leq \top(a, c)$ (Monotonie)
- $\forall a, b \in [0, 1] : \top(a, b) = \top(b, a)$ (Kommutativität)
- $\forall a, b, c \in [0, 1] : \top(a, \top(b, c)) = \top(\top(a, b), c)$ (Assoziativität)

Weitere Bedingungen, die manchmal gestellt werden, sind:

- \top ist eine stetige Funktion (Stetigkeit)
- $\forall a \in [0, 1] : \top(a, a) < a$ (Subidempotenz)
- $\forall a, b, c, d \in [0, 1] : a < b \wedge c < d \Rightarrow \top(a, b) < \top(c, d)$ (strikte Monotonie)

Die ersten beiden dieser Bedingungen (zusätzlich zu den ersten vier) definieren die Teilklasse der sogenannten *Archimedischen t-Normen*.

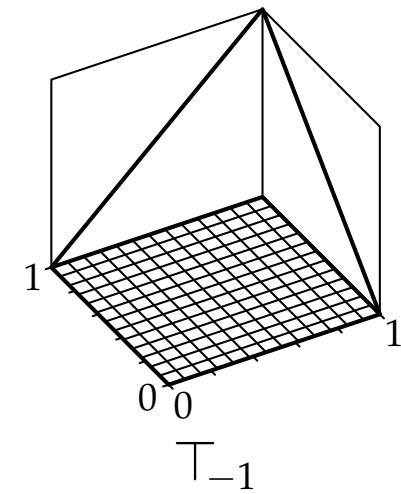
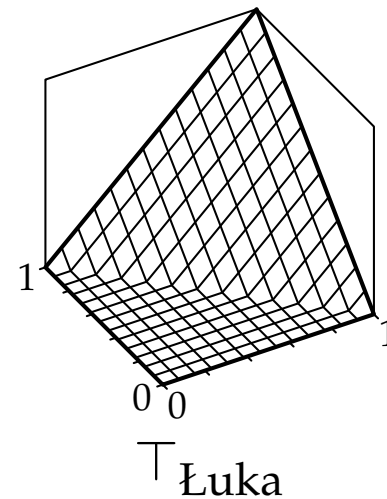
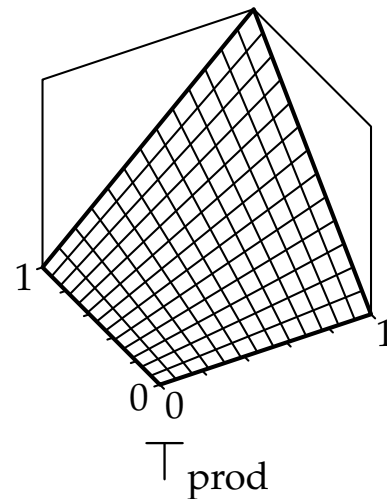
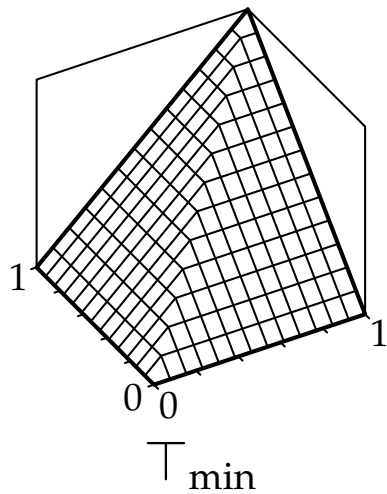
t-Normen / Fuzzy-Konjunktionen

Standardkonjunktion: $\top_{\min}(a, b) = \min\{a, b\}$

Algebraisches Product: $\top_{\text{prod}}(a, b) = a \cdot b$

Łukasiewicz: $\top_{\text{Łuka}}(a, b) = \max\{0, a + b - 1\}$

Drastisches Product: $\top_{-1}(a, b) = \begin{cases} a, & \text{if } b = 1, \\ b, & \text{if } a = 1, \\ 0, & \text{otherwise.} \end{cases}$



t-Konormen / Fuzzy-Disjunktionen

Eine **t-Konorm** oder **Fuzzy-Disjunktion** ist eine Funktion $\perp : [0, 1]^2 \rightarrow [0, 1]$, die die folgenden Bedingungen erfüllt:

- $\forall a \in [0, 1] : \perp(a, 0) = a$ (Randbedingung)
- $\forall a, b, c \in [0, 1] : b \leq c \Rightarrow \perp(a, b) \leq \perp(a, c)$ (Monotonie)
- $\forall a, b \in [0, 1] : \perp(a, b) = \perp(b, a)$ (Kommutativität)
- $\forall a, b, c \in [0, 1] : \perp(a, \perp(b, c)) = \perp(\perp(a, b), c)$ (Assoziativität)

Weitere Bedingungen, die manchmal gestellt werden, sind:

- \perp ist eine stetige Funktion (Stetigkeit)
- $\forall a \in [0, 1] : \perp(a, a) > a$ (Superidempotenz)
- $\forall a, b, c, d \in [0, 1] : a < b \wedge c < d \Rightarrow \perp(a, b) < \perp(c, d)$ (strikte Monotonie)

Die ersten beiden dieser Bedingungen (zusätzlich zu den ersten vier) definieren die Teilklasse der sogenannten *Archimedischen t-Konormen*.

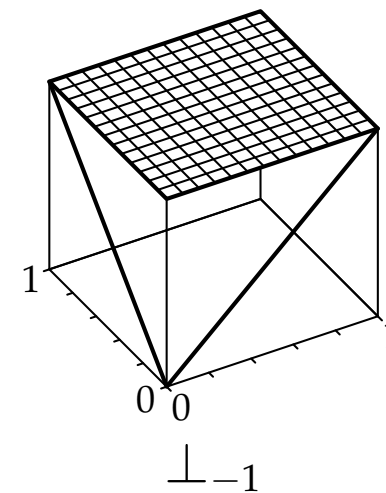
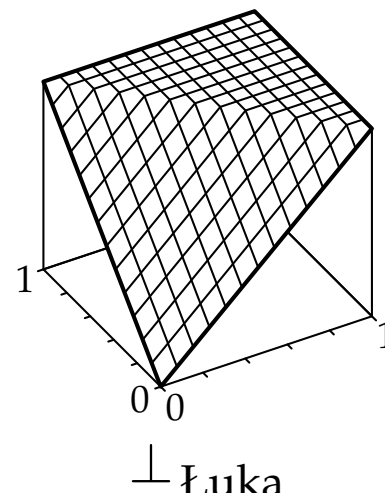
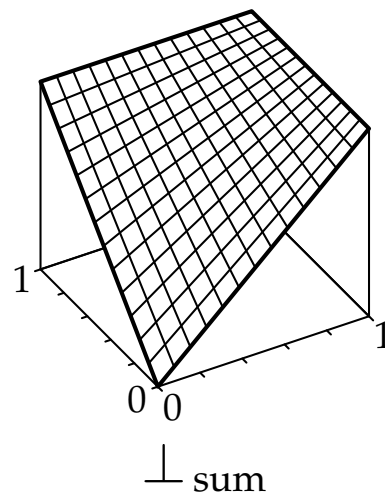
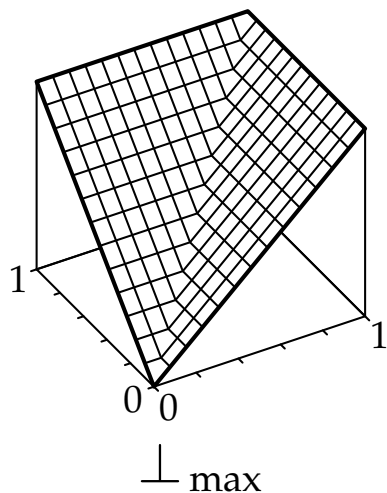
t-Konormen / Fuzzy-Disjunktionen

Standarddisjunktion: $\perp_{\max}(a, b) = \max\{a, b\}$

Algebraische Summe: $\perp_{\text{sum}}(a, b) = a + b - a \cdot b$

Łukasiewicz: $\perp_{\text{Łuka}}(a, b) = \min\{1, a + b\}$

Drastische Summe: $\perp_{-1}(a, b) = \begin{cases} a, & \text{if } b = 0, \\ b, & \text{if } a = 0, \\ 1, & \text{otherwise.} \end{cases}$



Zusammenspiel der Fuzzy-Operatoren

- Es gilt $\forall a, b \in [0, 1] : \top_{-1}(a, b) \leq \top_{\text{Łuka}}(a, b) \leq \top_{\text{prod}}(a, b) \leq \top_{\text{min}}(a, b)$.
Auch alle anderen denkbaren t -Normen liegen zwischen \top_{-1} und \top_{min} .
- Es gilt $\forall a, b \in [0, 1] : \perp_{\text{max}}(a, b) \leq \perp_{\text{sum}}(a, b) \leq \perp_{\text{Łuka}}(a, b) \leq \perp_{-1}(a, b)$.
Auch alle anderen denkbaren t -Konormen liegen zwischen \perp_{max} und \perp_{-1} .
- Beachte: Es gilt i.a. *weder* $\top(a, \sim a) = 0$ *noch* $\perp(a, \sim a) = 1$.
- Ein Operatorsatz (\sim, \top, \perp) bestehend aus einer Fuzzy-Negation \sim , einer t -Norm \top und einer t -Konorm \perp heißt **duales Tripel**, wenn mit diesen Operatoren die Verallgemeinerungen der DeMorganschen Gesetze gelten, d.h.

$$\forall a, b \in [0, 1] : \sim \top(a, b) = \perp(\sim a, \sim b)$$

$$\forall a, b \in [0, 1] : \sim \perp(a, b) = \top(\sim a, \sim b)$$

- Der am häufigsten benutzte Operatorsatz ist das duale Tripel $(\sim, \top_{\text{min}}, \perp_{\text{max}})$ mit der Standardnegation $\sim a \equiv 1 - a$.

Fuzzy-Mengenlehre

- Die klassische Mengenlehre basiert auf dem Begriff “*ist Element von*” (\in). Alternativ kann man die Zugehörigkeit zu einer Menge mit einer *Indikatorfunktion* beschreiben:

Sei X eine Menge. Dann heißt

$$I_M : X \rightarrow \{0, 1\}, \quad I_M(x) = \begin{cases} 1, & \text{falls } x \in X, \\ 0, & \text{sonst,} \end{cases}$$

Indikatorfunktion der Menge M bzgl. der Grundmenge X .

- In der Fuzzy-Mengenlehre wird die Indikatorfunktion durch eine *Zugehörigkeitsfunktion* ersetzt:

Sei X eine (klassische/scharfe) Menge. Dann heißt

$$\mu_M : X \rightarrow [0, 1], \quad \mu_M(x) \hat{=} \text{Zugehörigkeitsgrad von } x \text{ zu } M,$$

Zugehörigkeitsfunktion (*membership function*) der **Fuzzy-Menge** M bzgl. der *Grundmenge* X .

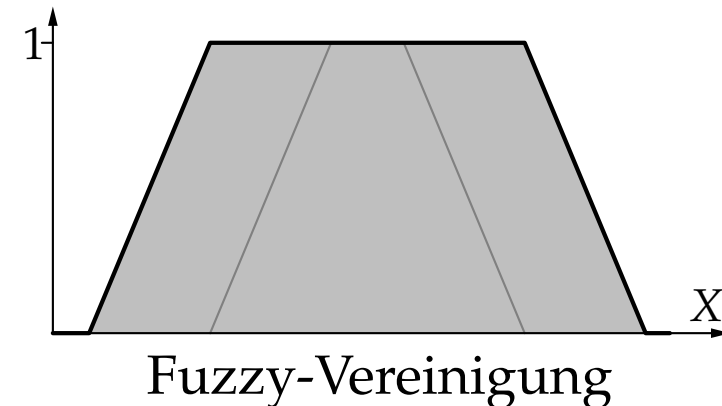
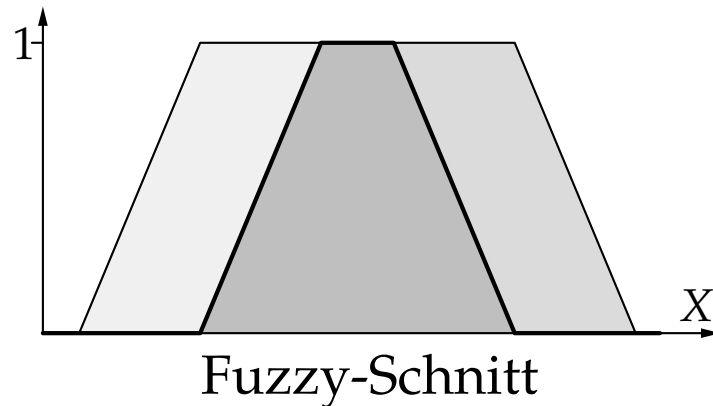
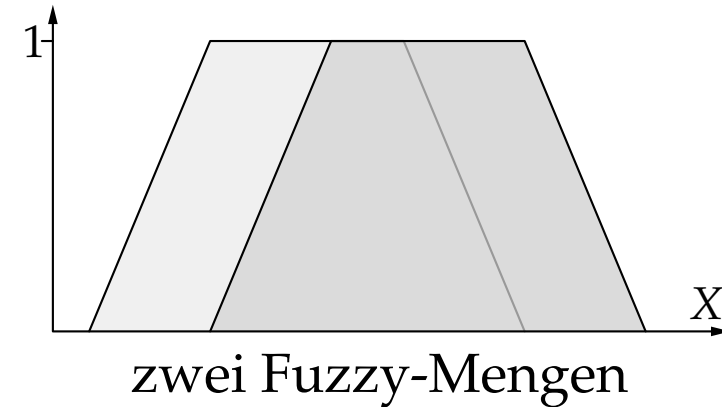
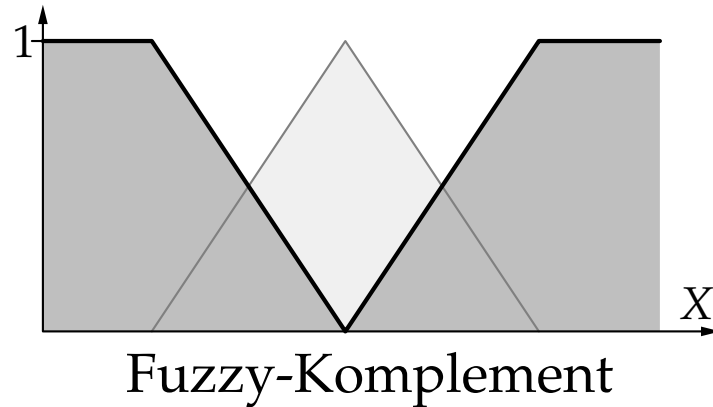
Meist wird die Fuzzy-Menge mit ihrer Zugehörigkeitsfunktion identifiziert.

Fuzzy-Mengenlehre: Operationen

- Wie beim Übergang von der klassischen Logik zur Fuzzy-Logik ist beim Übergang von der klassischen Mengenlehre zur Fuzzy-Mengenlehre eine Erweiterung der Operationen nötig.
- **Grundprinzip dieser Erweiterung:**
Greife auf die logische Definition der Operationen zurück.
⇒ elementweise Anwendung der logischen Operatoren
- Seien A und B (Fuzzy-)Mengen über der Grundmenge X .

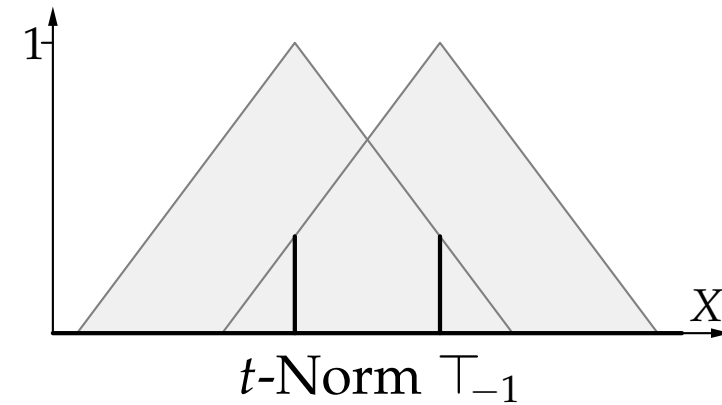
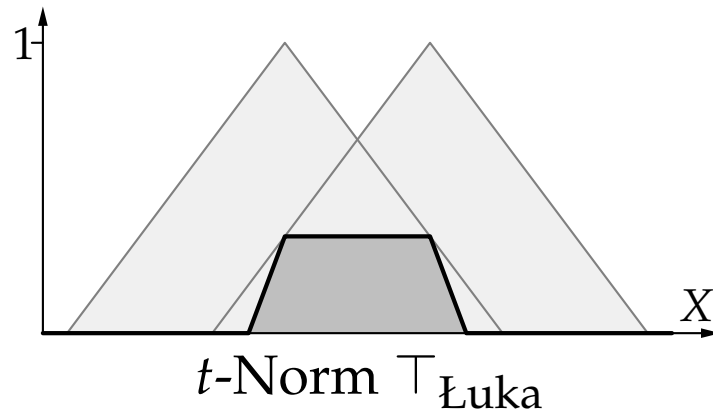
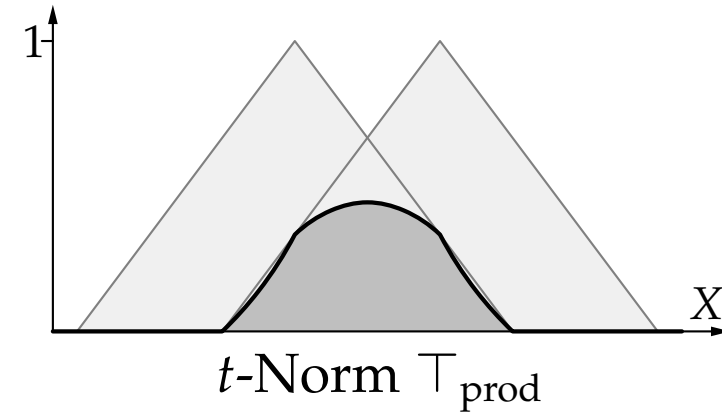
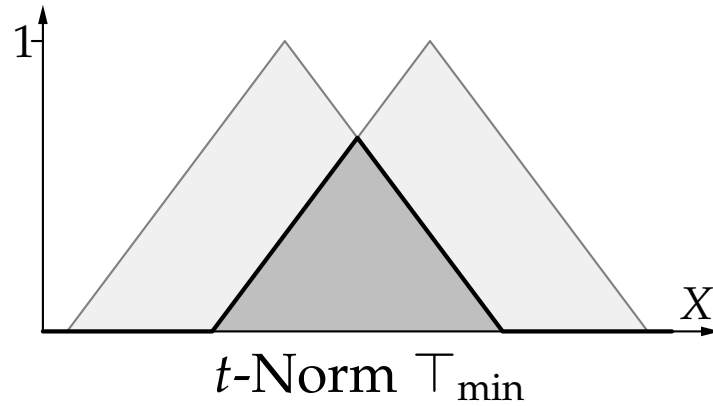
Komplement	klassisch	$\bar{A} = \{x \in X \mid x \notin A\}$
	fuzzy	$\forall x \in X: \mu_{\bar{A}}(x) = \sim\mu_A(x)$
Schnitt	klassisch	$A \cap B = \{x \in X \mid x \in A \wedge x \in B\}$
	fuzzy	$\forall x \in X: \mu_{A \cap B}(x) = \top(\mu_A(x), \mu_B(x))$
Vereinigung	klassisch	$A \cup B = \{x \in X \mid x \in A \vee x \in B\}$
	fuzzy	$\forall x \in X: \mu_{A \cup B}(x) = \perp(\mu_A(x), \mu_B(x))$

Fuzzy-Mengenoperationen: Beispiele



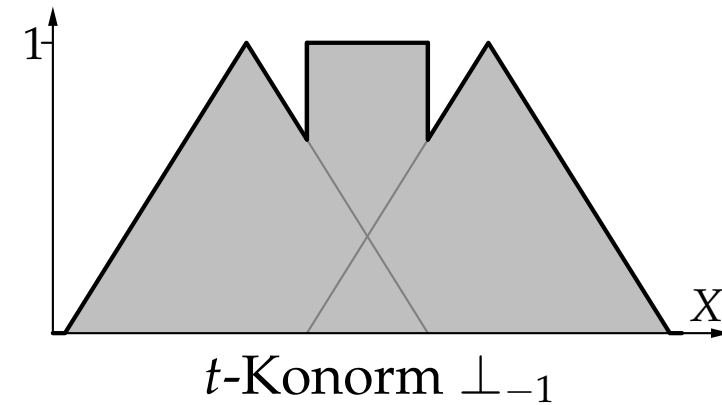
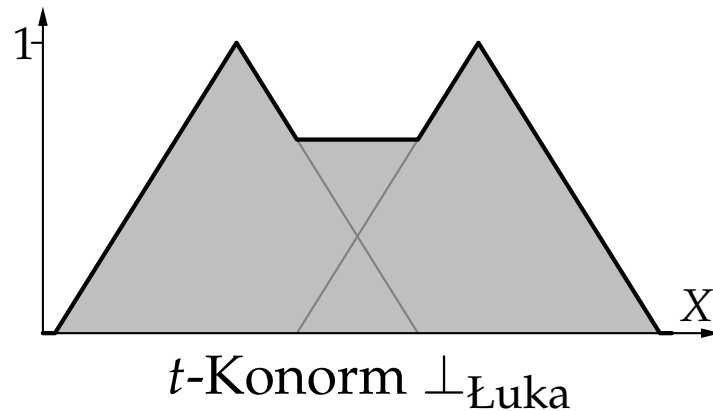
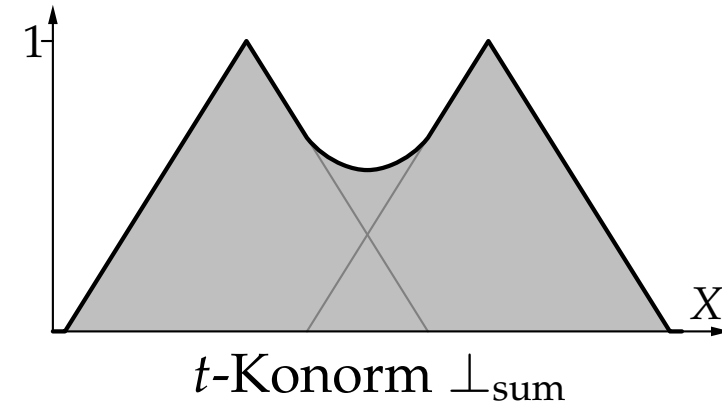
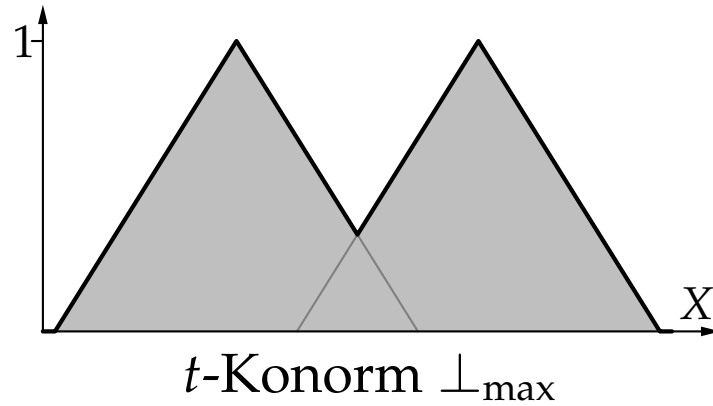
- Der links gezeigte Fuzzy-Schnitt und die rechts gezeigte Fuzzy-Vereinigung sind unabhängig von der gewählten t -Norm bzw. t -Konorm.

Fuzzy-Schnitt: Beispiele



- Man beachte, daß alle Fuzzy-Schnitte zwischen dem oben links und dem unten rechts gezeigten liegen.

Fuzzy-Vereinigung: Beispiele



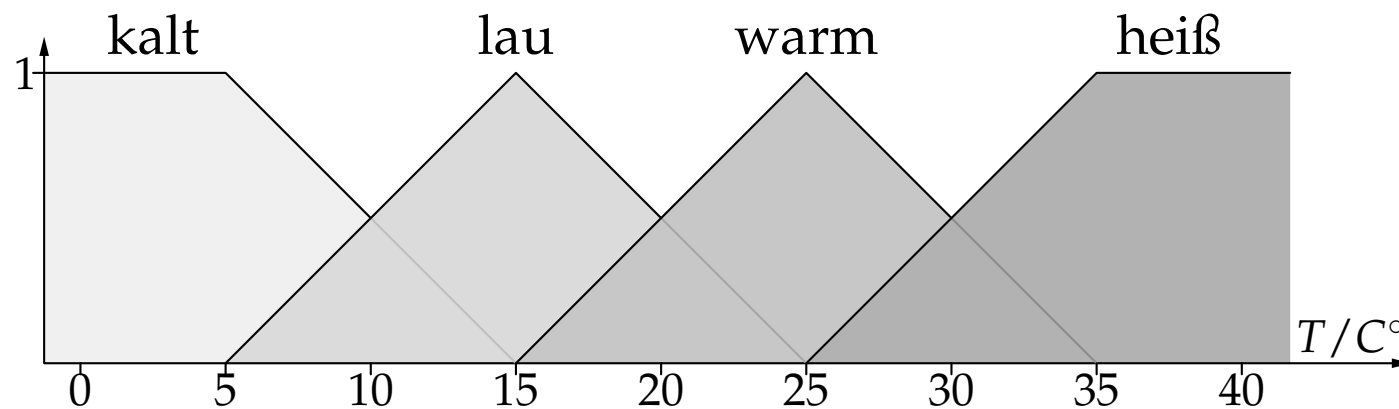
- Man beachte, daß alle Fuzzy-Vereinigungen zwischen der oben links und der unten rechts gezeigten liegen.

Fuzzy-Partitionen und Linguistische Variablen

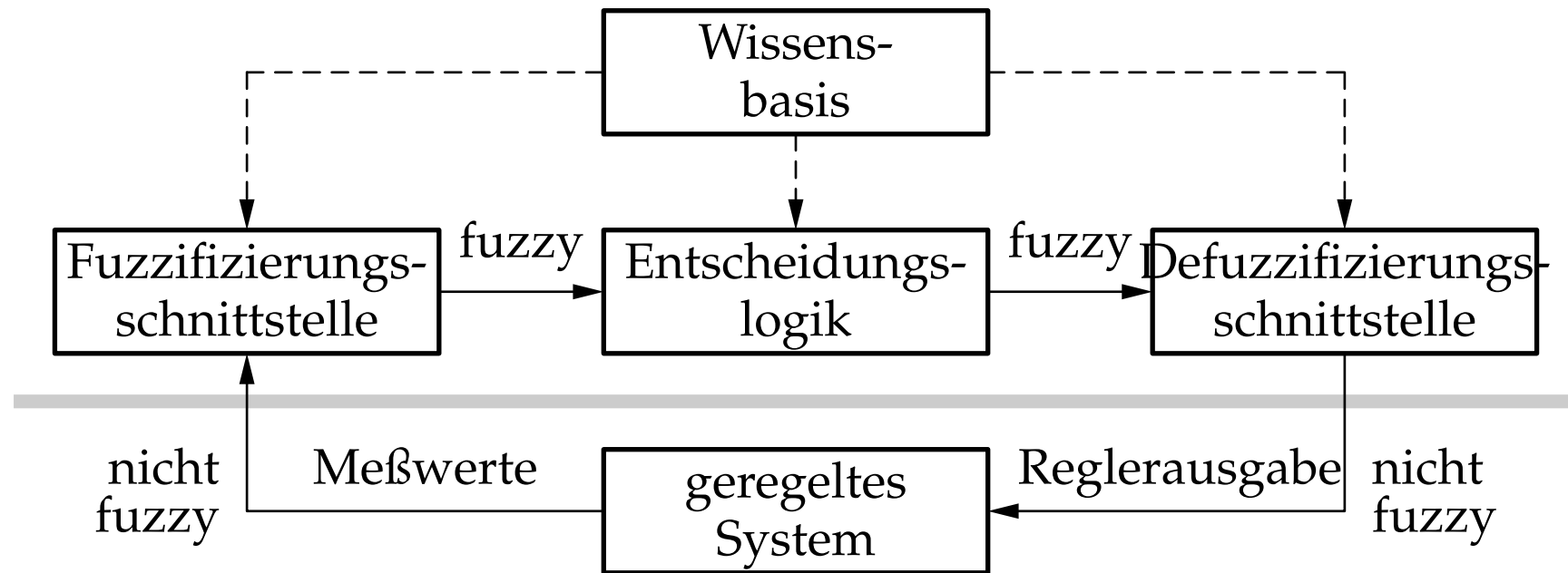
- Um Wertebereiche durch sprachliche (linguistische) Ausdrücke beschreiben zu können, werden sie mit Hilfe von Fuzzy-Mengen fuzzy-partitioniert.
Jeder Fuzzy-Menge der Partitionierung ist ein linguistischer Term zugeordnet.
- Übliche Bedingung: An jedem Punkt müssen sich die Zugehörigkeitsgrade aller Fuzzy-Mengen zu 1 addieren (*partition of unity*).

Beispiel: Fuzzy-Partitionierung für Temperaturen

Wir definieren eine linguistische Variable mit den Werten *kalt*, *lau*, *warm* und *heiß*.

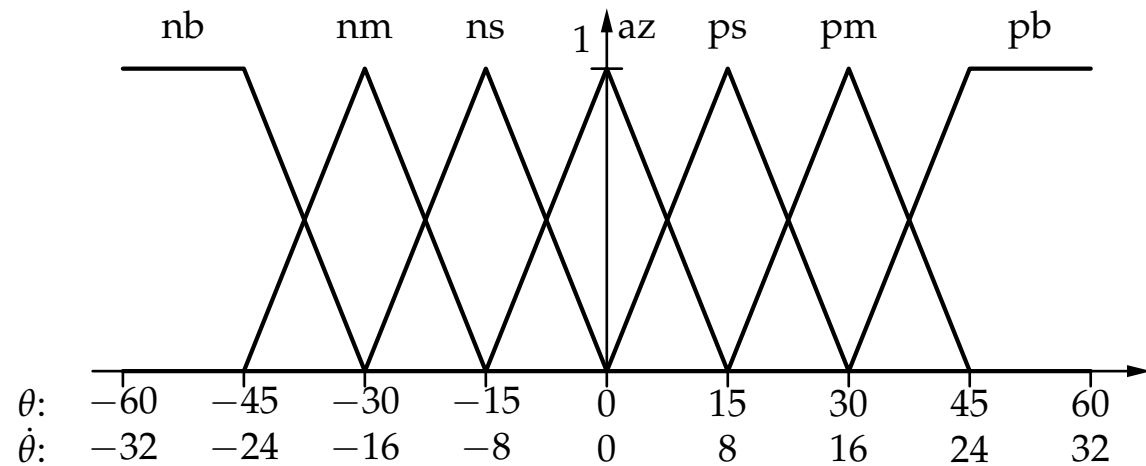
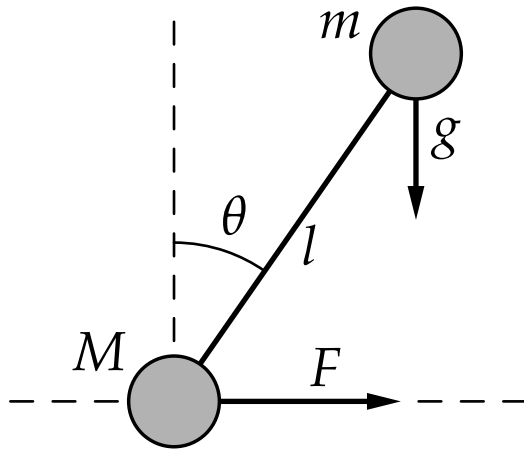


Architektur eines Fuzzy-Reglers



- Die Wissensbasis enthält die Fuzzy-Regeln für die Steuerung und die Fuzzy-Partitionen der Wertebereiche der Variablen.
- Eine Fuzzy-Regel lautet: **if X_1 is $A_{i_1}^{(1)}$ and ... and X_n is $A_{i_n}^{(n)}$ then Y is B .**
 X_1, \dots, X_n sind die Meßgrößen und Y ist die Stellgröße.
 $A_{i_k}^{(k)}$ und B sind linguistische Terme, denen Fuzzy-Mengen zugeordnet sind.

Beispiel-Fuzzy-Regler: Stabbalance

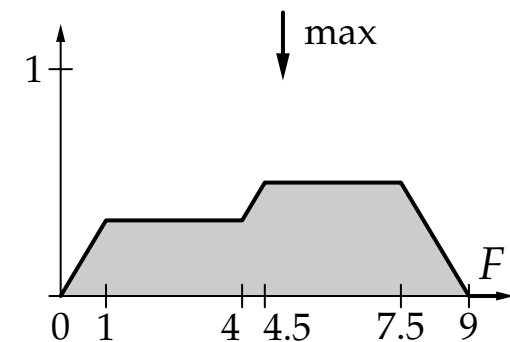
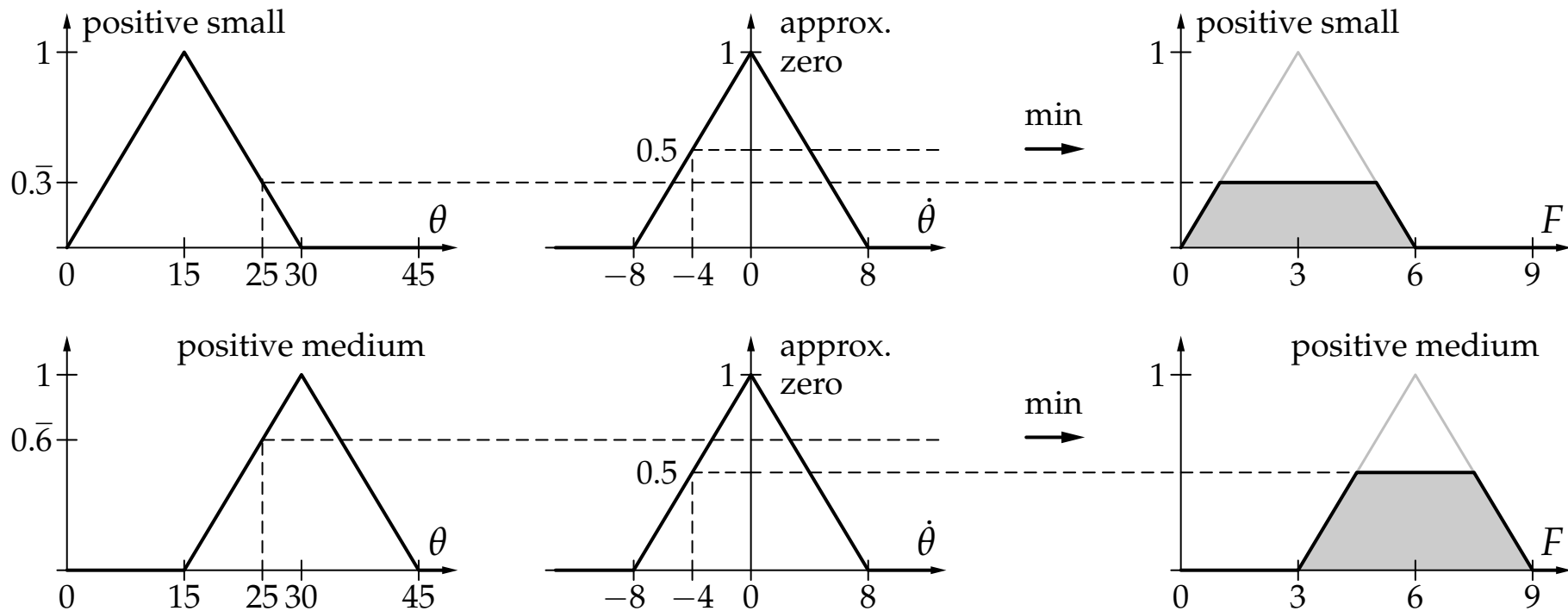


Abkürzungen

- pb – positive big
- pm – positive medium
- ps – positive small
- az – approximately zero
- ns – negative small
- nm – negative medium
- nb – negative big

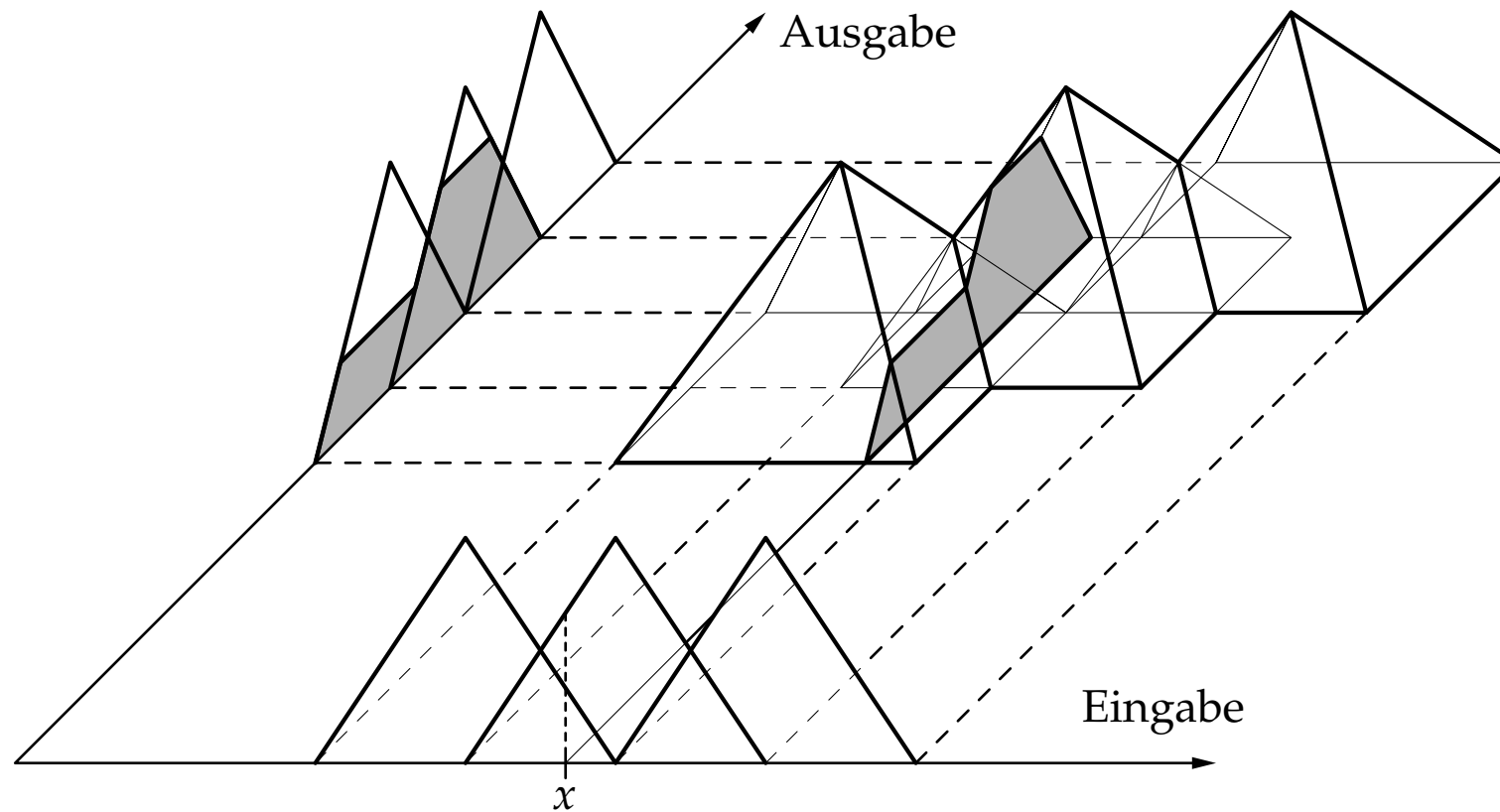
$\dot{\theta} \backslash \theta$	nb	nm	ns	az	ps	pm	pb
pb			ps	pb			
pm				pm			
ps	nm		az	ps			
az	nb	nm	ns	az	ps	pm	pb
ns				ns	az		pm
nm				nm			
nb				nb	ns		

Fuzzy-Regelung nach Mamdani–Assilian



Regelauswertung in einem Mamdani–Assilian-Regler.
 Das Eingabetupel (25, -4) führt zu der rechts gezeigten un-
 scharfen Ausgabe. Aus dieser Fuzzy-Menge wird der ent-
 sprechende Ausgabewert durch Defuzzifizierung bestimmt,
 z.B. durch die Mean-of-Maxima-Methode (MOM) oder die
 Schwerpunktmethod (COG).

Fuzzy-Regelung nach Mamdani-Assilian



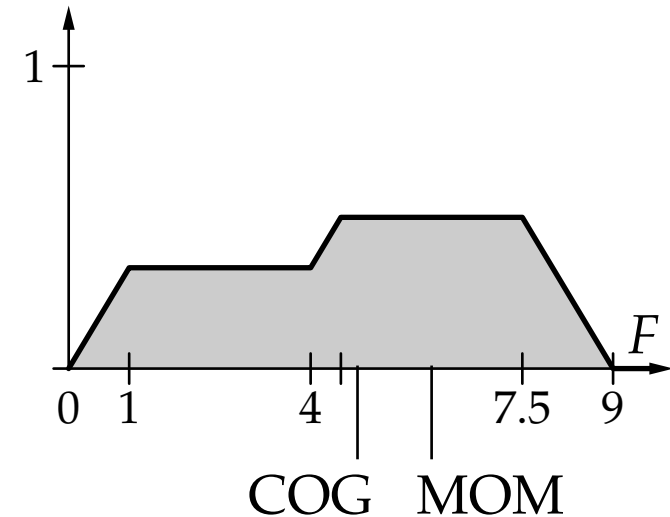
Ein Fuzzy-Regelsystem einer Meß- und einer Stellgröße und drei Fuzzy-Regeln.
Jede Pyramide wird durch eine Fuzzy-Regel spezifiziert.
Der Eingabewert x führt zu der grau gezeichneten unscharfen Ausgabe.

Defuzzifizierung

Die Auswertung der Fuzzy-Regeln liefert eine **Ausgabe-Fuzzy-Menge**.

Die Ausgabe-Fuzzy-Menge muß in einen **scharfen Stellwert** umgewandelt werden.

Dieser Vorgang heißt **Defuzzifizierung**.



Die wichtigsten Defuzzifizierungsmethoden sind:

- **Schwerpunktmethode** (Center of Gravity, COG)
Der Schwerpunkt der Fläche unter der Ausgabe-Fuzzy-Menge.
- **Flächenmittelpunktmethode** (Center of Area, COA)
Der Punkt, der die Fläche unter der Ausgabe-Fuzzy-Menge in gleich große Teile teilt.
- **Maxima-Mittelwert-Methode** (Mean of Maxima, MOM)
Das arithmetische Mittel der Stellen mit maximalem Zugehörigkeitsgrad.

Erzeugen/Optimieren von Fuzzy-Reglern mit GA

- Bei einem Mamdani-Assilian-Regler kann optimiert werden:
 - Die **Regelbasis** (welche Regeln, welche Ausgaben).
 - Die **Fuzzy-Mengen/Fuzzy-Partitionen** (Form, Lage, Ausdehnung, ggf. Anzahl der Fuzzy-Mengen).
 - Die **t-Norm** bzw. **t-Konorm** für die Regelauswertung (selten).
 - Parameter der **Defuzzifizierungsmethode** (falls vorhanden; selten).
 - Welche **Eingangsgrößen** in den Regeln verwendet werden. (Merkmalsauswahl, engl. *feature selection*)
- Wir betrachten nur die Optimierung der Regelbasis und der Fuzzy-Mengen bei fester Wahl der Eingabe-/Meßgrößen.
- Die Regelauswertung geschieht (wie gezeigt) über Minimum und Maximum, die Defuzzifizierung über die Schwerpunktmethode.

Mögliche Vorgehensweisen:

- *Die Regelbasis und die Fuzzy-Partitionen werden gleichzeitig optimiert.*

Nachteil: Sehr große Zahl von Parametern muß gleichzeitig optimiert werden.

- *Erst werden die Fuzzy-Partitionen bei vorgegebener Regelbasis optimiert, dann wird die Regelbasis mit den besten Fuzzy-Partitionen optimiert.*

Nachteil: Zum Aufstellen der Regelbasis muß Expertenwissen verfügbar sein.
(Wahl einer zufälligen Regelbasis ist wenig erfolgversprechend.)

- *Erst wird die Regelbasis für vorgegebene Fuzzy-Mengen optimiert, dann werden die Fuzzy-Partitionen mit der besten Regelbasis optimiert.*

Die Fuzzy-Mengen können z.B. äquidistant (gleichmäßig) verteilt werden. In diesem Fall muß ein Anwender lediglich die Zahl der Fuzzy-Mengen je Meßgröße und für die Stellgröße vorgeben.

→ Hier betrachten wir nur diese Möglichkeit.

Erzeugen/Optimieren von Fuzzy-Reglern: Fitnessfunktion

- Ein guter Regler sollte verschiedene Kriterien erfüllen:
 - Aus jeder möglichen Situation sollte der Sollzustand erreicht werden.
 - Der Sollzustand sollte möglichst schnell erreicht werden.
 - Der Sollzustand sollte mit geringem (Energie-)Aufwand erreicht werden.
- Der Regler wird mehrfach testweise auf das zu regelnde System angewandt.

(hier: Simulation des Stabbalance-Problems in einem Rechner, mehrere, zufällig gewählte Anfangssituationen)

Je nach Regelerfolg/Regelgüte erhält der Regler Punkte (Anzahl Situationen, Dauer erfolgreicher Regelung, Energieaufwand).

- **Beachte:**

In dieser Anwendung ist die Bewertung der Individuen die mit Abstand aufwendigste Operation (benötigte Rechenleistung). Jedes Individuum muß über eine gewisse Mindestzahl von Zeitschritten zur Regelung eingesetzt werden.

Bewertung des Regelerfolgs

- Weicht der Istwert zu stark vom Sollwert ab, wird abgebrochen (Fehlschlag). (z.B. Stabbalance-Problem: der Istwert muß im Intervall $[-90^\circ, 90^\circ]$ bleiben.)
- Nach einer bestimmten Dauer sollte der Istwert in der Nähe des Sollwertes liegen und dort verbleiben (Toleranzbereich). Ist dies nicht der Fall, wird ebenfalls abgebrochen (Fehlschlag).
- Der Toleranzbereich wird im Laufe der Generationen verringert (langsames Hinführen auf das gewünschte Ziel).
 - In den ersten Generationen ist es hinreichend, wenn der zu balancierende Stab nicht umfällt.
 - Später muß der Stab in einem immer engeren Winkelbereich um die senkrechte Lage gehalten werden.
- Die Beträge der Stellwerte werden aufsummiert und als Strafterm verwendet. (In der Gleichgewichtslage hat ein schnelles Umschalten zwischen großen Kräften gleiche Wirkung wie eine Regelung mit geringen Kräften. Hohe Kräfte sollen vermieden werden.)

Erzeugen/Optimieren der Regelbasis: Kodierung

- Es werden nur vollständige Regelbasen betrachtet (zu jeder Kombination von Eingabe-Fuzzy-Mengen gibt es eine Regel).
- Für jede Kombination von Eingabe-Fuzzy-Mengen muß lediglich der linguistische Term der Stellgröße festgelegt werden (i.w. Ausfüllen einer Tabelle).

Beispiel: Regelbasis für Stabbalance-Regler:

$\dot{\theta} \backslash \theta$	nb	nm	ns	az	ps	pm	pb
pb	az	ps	ps	pb	pb	pb	pb
pm	ns	az	ps	pm	pm	pb	pb
ps	nm	ns	az	ps	pm	pm	pb
az	nb	nm	ns	az	ps	pm	pb
ns	nb	nm	nm	ns	az	ps	pm
nm	nb	nb	nm	nm	ns	az	ps
nb	nb	nb	nb	nb	ns	ns	az

schematisch

Kodierung der Regelbasis

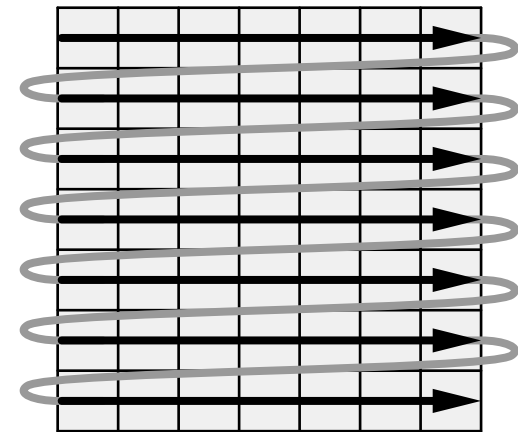
Darstellung der Regelbasis als Chromosom

- **Linearisierung** (Umwandlung in Vektor)

Die Tabelle wird in einer willkürlichen, aber festen Reihenfolge durchlaufen und die Tabelleneinträge werden in einem Vektor aufgelistet.

Beispiel: zeilenweise Auflistung

Problem: Nachbarschaftsbeziehungen
zwischen den Zeilen gehen verloren
(benachbarte Einträge sollten ähnliche
linguistische Terme enthalten; dies sollte
z.B. beim Crossover berücksichtigt werden)



- **Tabelle** (direkte Verwendung des Schemas)

Es werden zwei- oder mehrdimensionale Chromosomen erzeugt.
(In diesem Fall werden spezielle genetic operators benötigt.)

Genetic Operators für die Regelbasis

- **Mutation:** (analog zu Standardmutation)
 - Eine Regel (ein Tabelleneintrag) wird zufällig gewählt.
 - Der linguistische Term der Ausgabe wird zufällig verändert.
 - Es können mehrere Regeln/Tabellenfelder gleichzeitig geändert werden.
- Es ist ggf. günstig, die Mutation einer Regelbasis so einzuschränken, daß ein Tabelleneintrag nur auf linguistische Terme geändert werden kann, die dem vorhandenen Eintrag benachbart oder hinreichend nahe sind.

Beispiele: “positive small” kann nur auf “approximately zero” oder “positive medium” geändert werden.

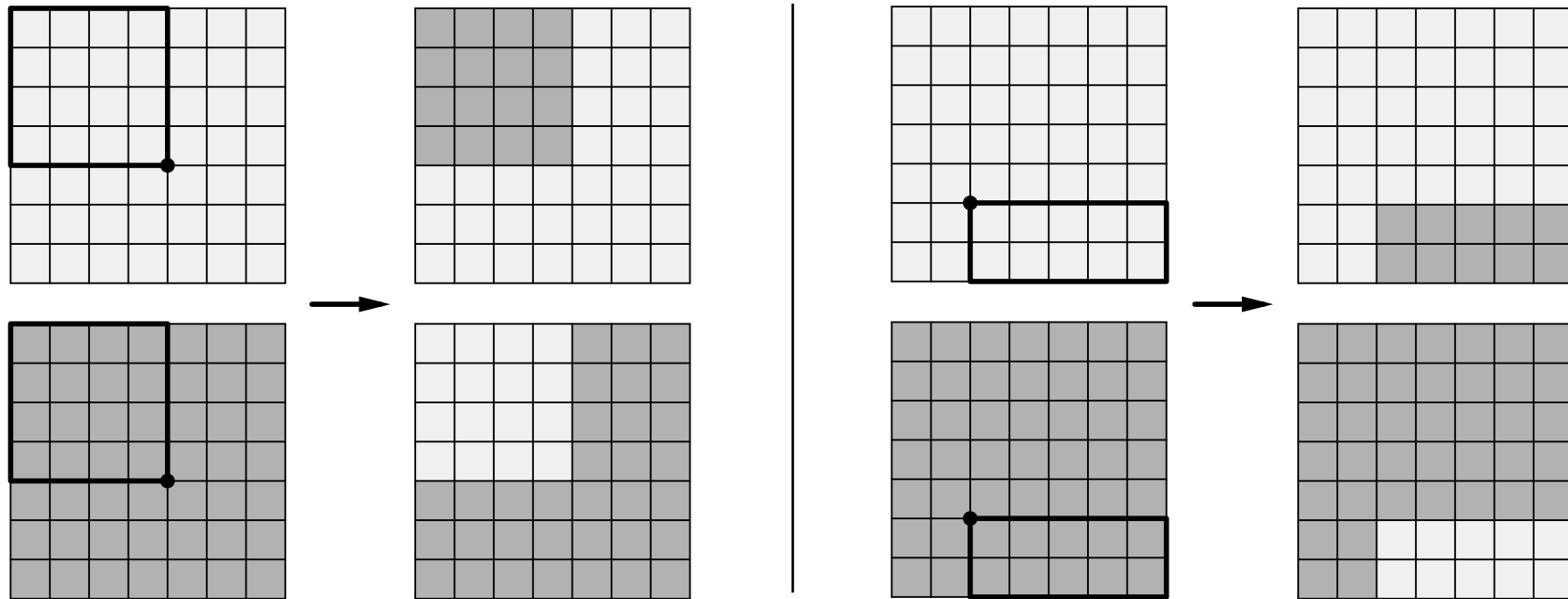
“negative big” kann nur auf “negative medium” oder ggf. “negative small” geändert werden.

(Dies verhindert ein zu schnelles “Zerfließen” gesammelter Information; die Regelbasen werden nur “vorsichtig” verändert.)

Genetic Operators für die Regelbasis

- **Crossover:** (Ein-Punkt-Crossover)

Wähle zufällig einen inneren Gitterpunkt der Tabelle und eine Ecke. Tausche die so definierte Teiltabellen zwischen zwei Eltern aus.

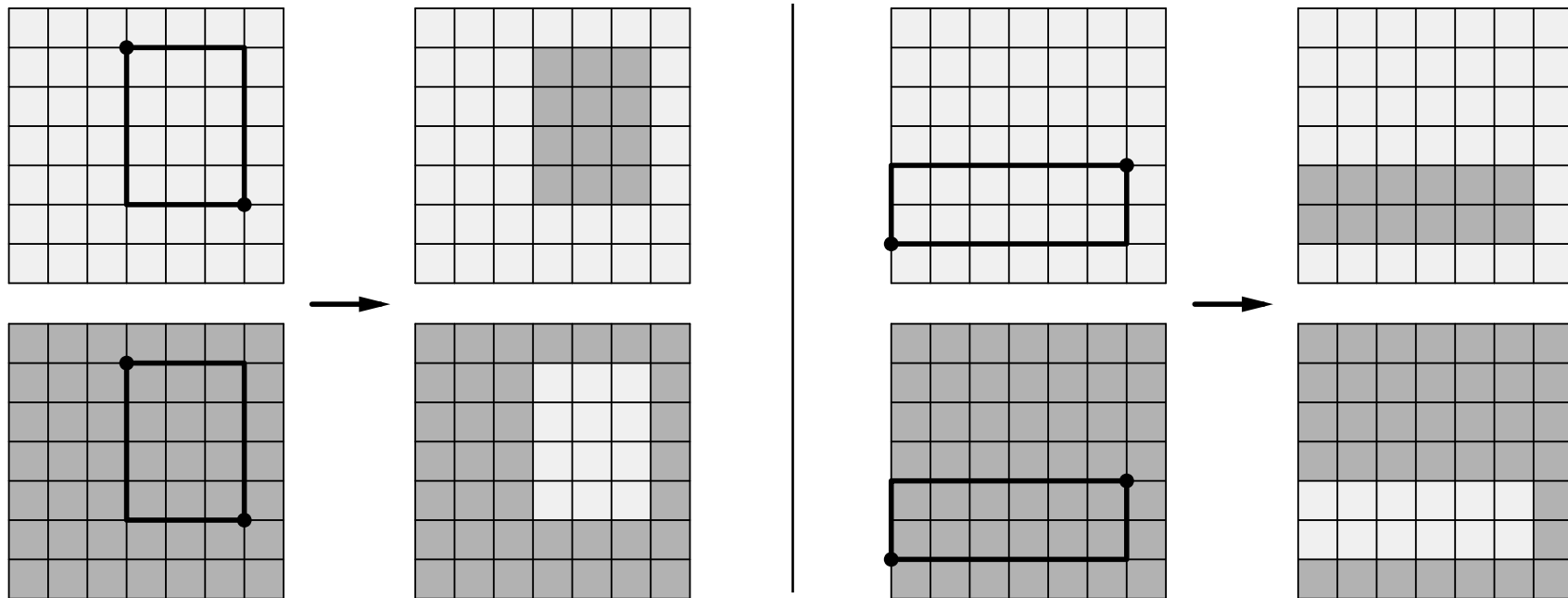


- **Beachte:** Um bevorzugt benachbarte Regeln zusammen zu vererben, *sollte* das Crossover ortsabhängige Verzerrung zeigen!

Genetic Operators für die Regelbasis

- **Crossover:** (Zwei-Punkt-Crossover)

Wähle zufällig zwei Gitterpunkte der Tabelle (auch Randpunkte).
Tausche die so definierte Teiltabellen zwischen zwei Eltern aus.



- Zwei-Punkt-Crossover ist besser geeignet als Ein-Punkt-Crossover, da Teillösungen flexibler ausgetauscht werden können.

Optimierung der Fuzzy-Mengen

- **Gegeben:** optimierte Regelbasis mit fest gewählten und nicht veränderten äquidistanten (gleichmäßig verteilten) Fuzzy-Mengen.
- **Gesucht:** weitere Verbesserung des Reglerverhaltens durch Anpassen der Fuzzy-Mengen bei fester Regelbasis (“Fine-Tuning”)
- **Kodierung der Fuzzy-Mengen:** (erste Möglichkeit)
 - Wähle die Form der Fuzzy-Mengen.
(z.B. Dreieck, Trapez, Gaußglocke, Parabel, Spline etc.)
 - Liste die definierenden Parameter der Fuzzy-Mengen auf.
(z.B. Dreieck: linker Rand, Mitte, rechter Rand)

Beispiel Stabbalance-Regler mit dreiecksförmigen Fuzzy-Mengen (Ausschnitt):

...	nm			ns			az			ps			...
...	-45	-30	-15	-30	-15	0	-15	0	15	0	15	30	...

Optimierung der Fuzzy-Mengen

Nachteile dieser Kodierung:

- Kodierung ist sehr “starr” bzgl. der Form der Fuzzy-Mengen:
Es wird z.B. vorher festgelegt, ob Dreiecke oder Trapeze verwendet werden.
- Genetic Operators können die Ordnung der Parameter zerstören
(es muß bei Dreiecken z.B. gelten: linker Rand \leq Mitte \leq rechter Rand).
- Mögliche “Überholvorgänge” zwischen Fuzzy-Mengen:
Durch Mutation und Crossover kann die sinnvolle Reihenfolge der Fuzzy-Mengen zerstört werden (es sollte z.B. gelten: n_s liegt rechts von p_s).
- Die Bedingung “Summe der Zugehörigkeitsgrade = 1” wird ggf. verletzt
(kann durch einmalige Darstellung identischer Parameter behandelt werden).

...	-45	-15	15	-15	15	30	15	30	45	30	45	60	...
...	-45	-30	-20	-30	-20	-10	-20	-10	0	-10	10	30	...

Kodierung der Fuzzy-Partitionen

An einer Reihe von gleichmäßig über den Wertebereich verteilten Stützstellen werden die Zugehörigkeitsgrade der verschiedenen Fuzzy-Mengen angegeben.

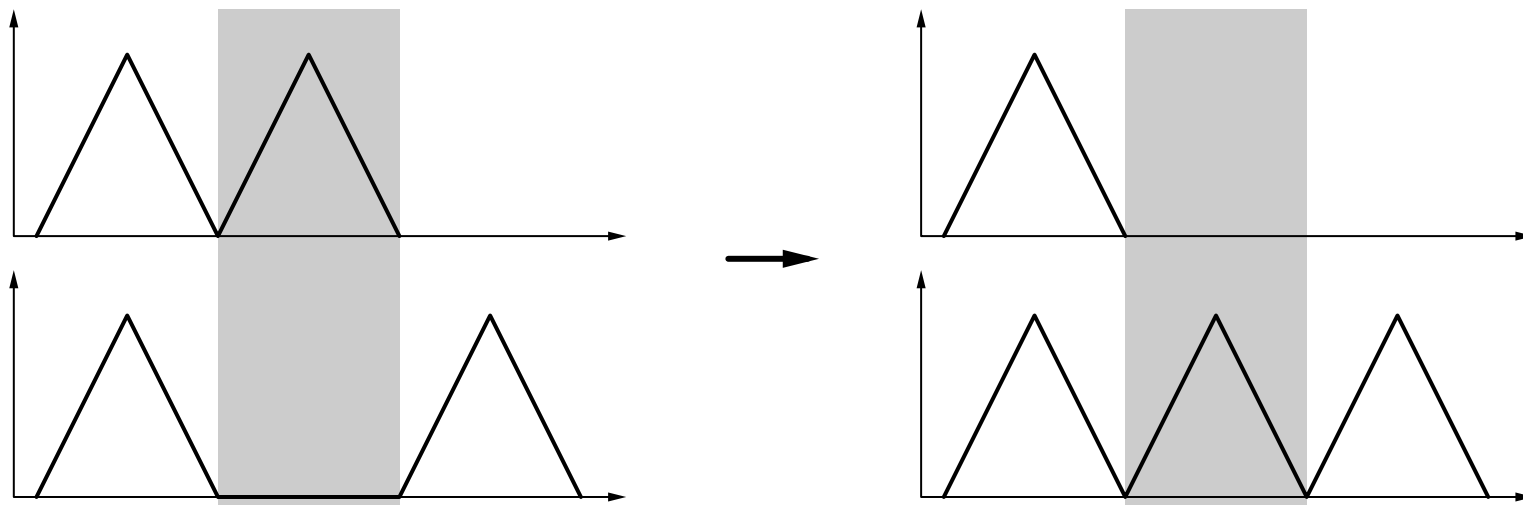
$$\underbrace{\begin{pmatrix} \mu_1(x_1) \\ \vdots \\ \mu_n(x_1) \end{pmatrix}}_{\text{Gen 1}} \cdots \underbrace{\begin{pmatrix} \mu_1(x_i) \\ \vdots \\ \mu_n(x_i) \end{pmatrix}}_{\text{Gen } i} \cdots \underbrace{\begin{pmatrix} \mu_1(x_m) \\ \vdots \\ \mu_n(x_m) \end{pmatrix}}_{\text{Gen } m}$$

Kodierung mit $m \times n$ Zahlen aus $[0, 1]$

pb	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	1	1	1
pm	0	0	0	0	0	0	0	0	0	0	0	0.5	1	0.5	0	0	0
ps	0	0	0	0	0	0	0	0	0	0.5	1	0.5	0	0	0	0	0
az	0	0	0	0	0	0	0	0.5	1	0.5	0	0	0	0	0	0	0
ns	0	0	0	0	0	0.5	1	0.5	0	0	0	0	0	0	0	0	0
nm	0	0	0	0.5	1	0.5	0	0	0	0	0	0	0	0	0	0	0
nb	1	1	1	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0
	-60	-45	-30	-15	0	15	30	45	60								

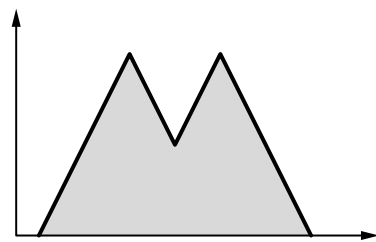
Genetic Operators

- **Mutation:** analog zur Standardmutation
Ein zufällig ausgewählter Eintrag wird zufällig geändert.
Es ist sinnvoll, die Größe der Änderung zu begrenzen,
z.B. durch Festlegen eines Intervalls oder durch eine Normalverteilung.
- **Crossover:** Einfaches Ein-Punkt- oder Zwei-Punkt-Crossover.
- Beachte: Durch Crossover können Fuzzy-Mengen ausgelöscht werden.

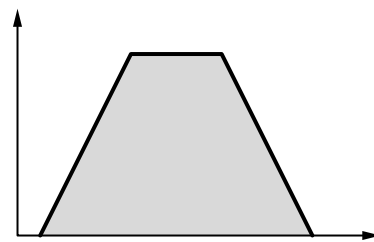


Reparatur der Fuzzy-Mengen

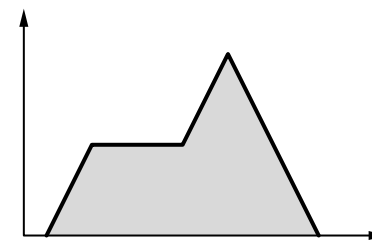
- Durch Mutation/Crossover kann der Fall eintreten, daß die Zugehörigkeitsfunktionen der Fuzzy-Mengen nicht mehr *unimodal* sind (d.h., die Zugehörigkeitsfunktion hat nur ein lokales Maximum.)
Multimodale Fuzzy-Mengen sind schwerer zu interpretieren als unimodale.
- Daher werden die Fuzzy-Mengen ggf. repariert (unimodal gemacht).



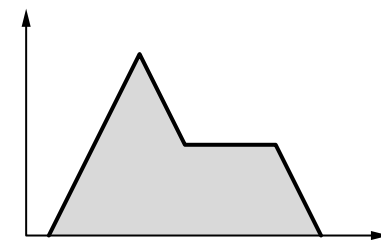
bimodale
Fuzzy-Menge



Verbinden
der Maxima



Abschneiden des
linken Maximums



Abschneiden des
rechten Maximums

- Ggf. sind auch Fuzzy-Mengen zu verbreitern oder abzuschneiden, damit der gesamte Wertebereich mit Fuzzy-mengen überdeckt ist, aber nicht zu viele Fuzzy-Mengen den gleichen Bereich abdecken.

Erzeugen/Optimieren von Fuzzy-Reglern mit GA

- Erzeugen/Optimieren in zwei Schritten:
 - Optimieren der Regelbasis bei festen Fuzzy-Partitionen.
 - Optimieren der Fuzzy-Partitionen mit erzeugter Regelbasis.
- In Experimenten gelang es, mit einem genetischen Algorithmus funktionsfähige Fuzzy-Regler für das Stabbalance-Problem zu erzeugen.
- Diese Art der Reglererzeugung ist zwar sehr aufwendig (Rechenzeit), hat aber den Vorteil, daß man kaum Hintergrundwissen benötigt.
- Weitere Anforderungen, die man in der Fitnessfunktion berücksichtigen kann:
 - **Kompaktheit:** geringe Anzahl von Regeln und Fuzzy-Mengen
 - **Vollständigkeit:** Abdeckung relevanter Bereiche im Eingaberaum
 - **Konsistenz:** keine Regeln mit sehr ähnlichem Antezedens und verschiedenem Konsequens
 - **Interpretierbarkeit:** Beschränkte Überlappung von Fuzzy-Mengen