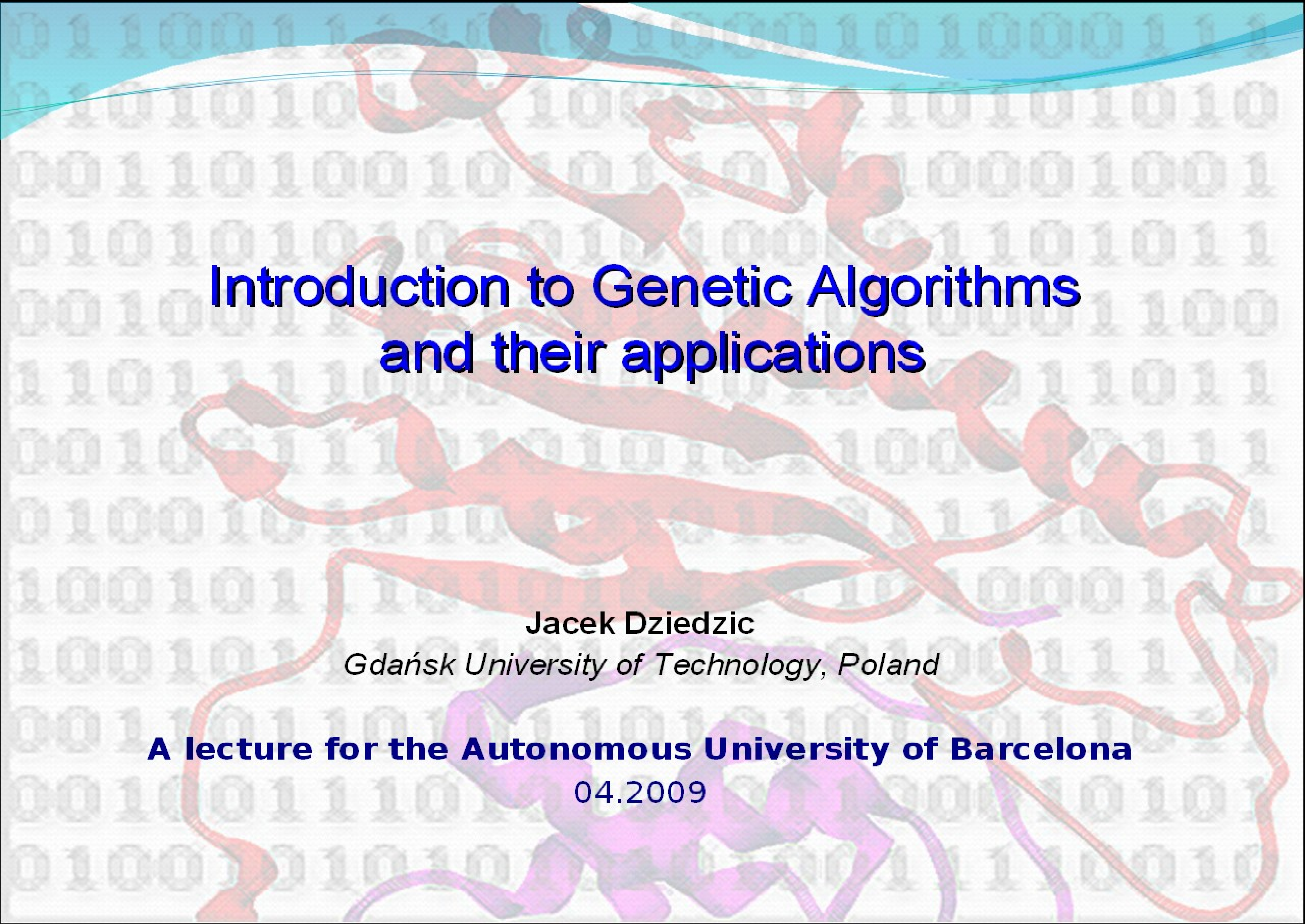


Introduction to Genetic Algorithms and their applications

Jacek Dziedzic

Gdańsk University of Technology, Poland

A lecture for the Autonomous University of Barcelona

04.2009

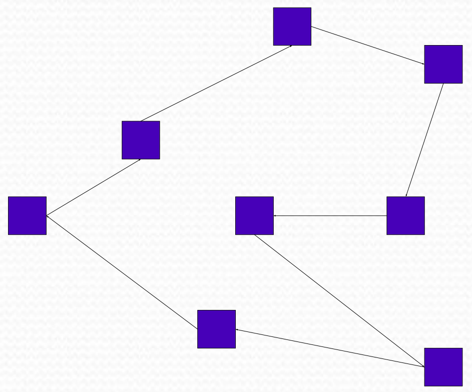
Parameter optimization problem

- One of the most common problems in engineering.
- There is a set of variables, we are looking for optimum values of the variables, i.e. values that minimize or maximize a target function.
  - *Trivial example: We need to build a cylindrical container that will hold 400 dm^3 of fuel. What are the dimensions of the container, so that the amount of material is minimized?*

$S(r, h)$ = min, with $\pi r^2 h=400$. r =?, h=?

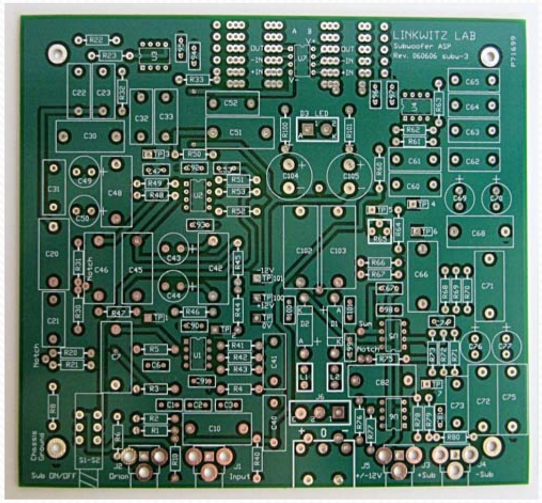
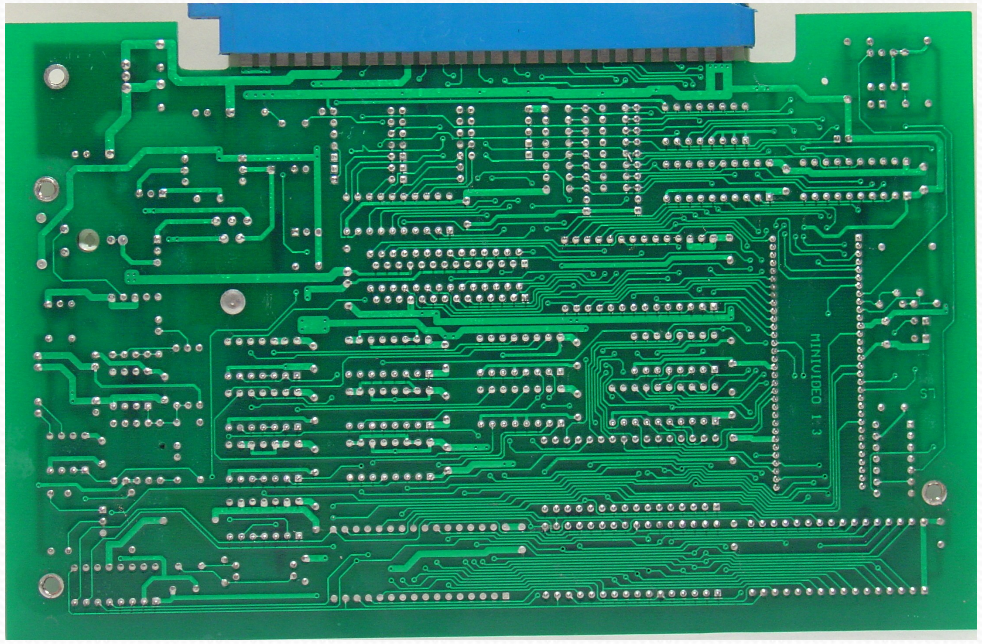
Parameter optimization problem

- One of the most common problems in engineering.
- There is a set of variables, we are looking for optimum values of the variables, i.e. values that minimize or maximize a target function.
  - ***Travelling salesman problem****. What is the least-cost trip that visits every city exactly once and returns to the one we start from?*
  - *The target function to be minimized is the total distance to travel. The variables are more difficult to define, because this is a combinatorial problem.*



Parameter optimization problem

- *Another version of the travelling salesman problem: a robotic drill drills holes in a PCB. What is the shortest path to drill all the holes?*

- Of course the direct search through all the permutations, the number of which grows as $O(n!)$, is infeasible for larger n.

Parameter optimization problem

- One of the most common problems in engineering.
- There is a set of variables, we are looking for optimum values of the variables, i.e. values that minimize or maximize a target function.
  - ***Knapsack problem***. *Given a set of items of defined value and weight, find out the number of each item to include, to obtain max total value, while not exceeding a total weight of W.*
  - *The target function f() to be maximized could be the total value. The (integral) variables x_1, x_2, .., x_n could be the counts of items. To imply the bound of max weight, we could assume f = 0 if we overloaded.*

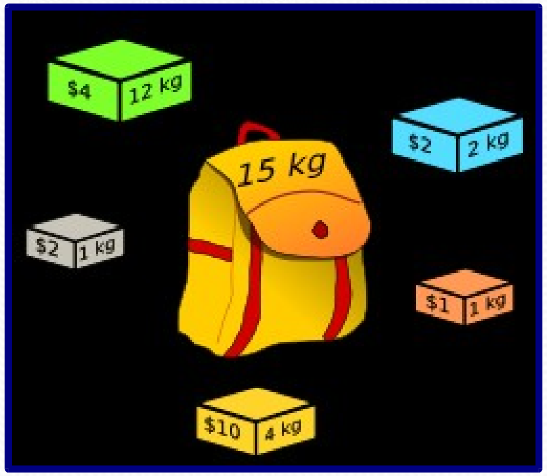
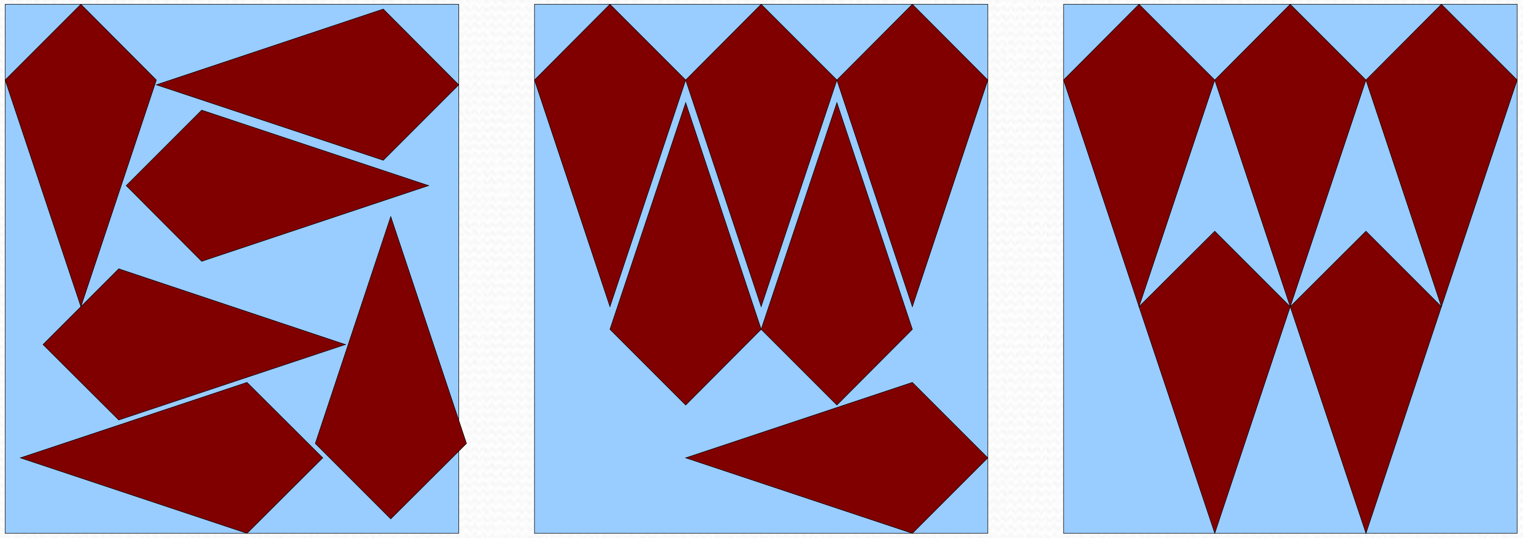


Image by Wikipedia

Other optimization problems

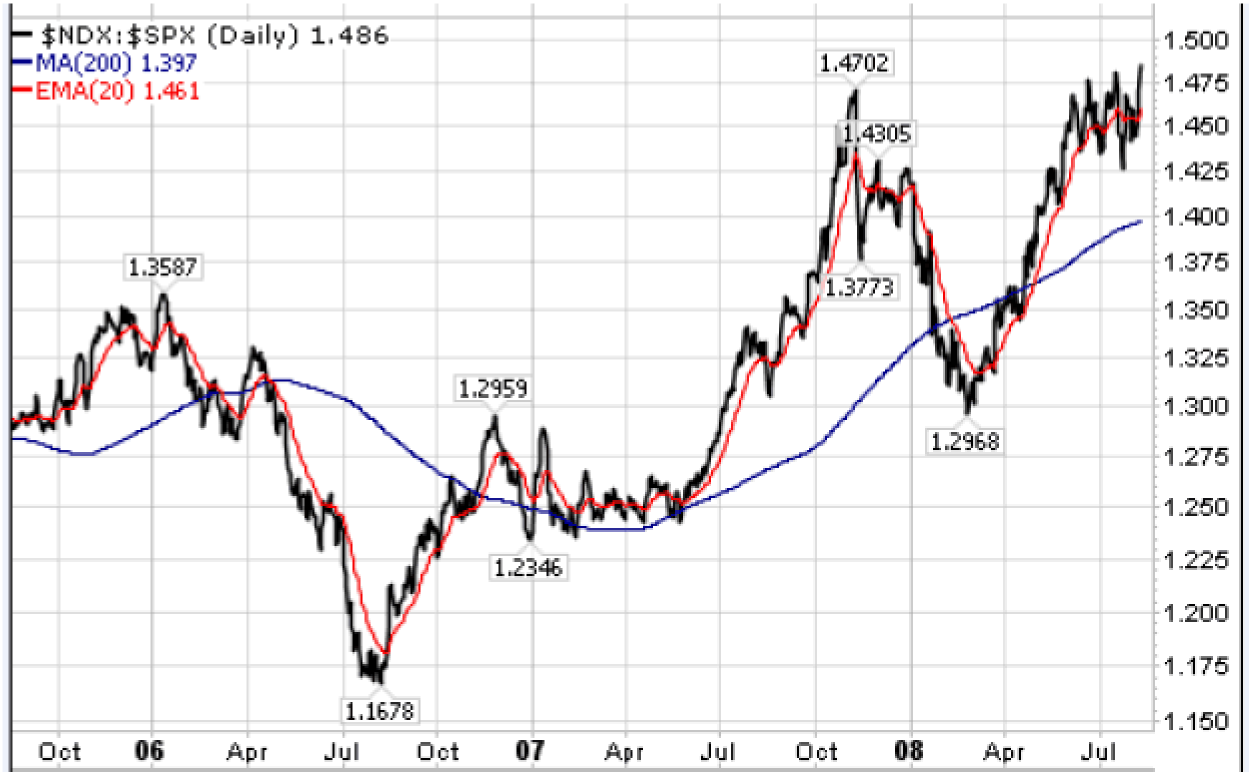
- *Given a large sheet of metal and a shape, how should we cut out the shapes so that the least metal is wasted?*

?



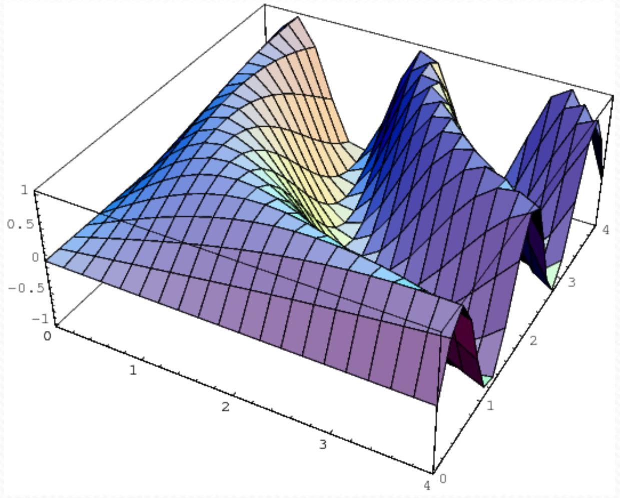
Other optimization problems

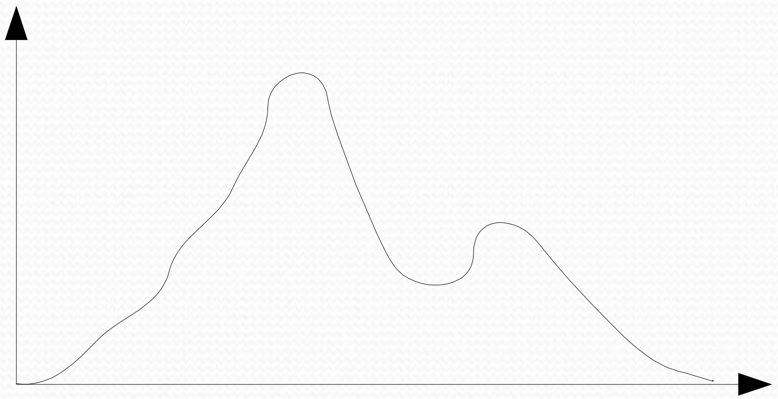
- *Given the past data from financial markets (e.g. foreign currency), find out a set of equations that best models the observed trends.*



Parameter optimization problem

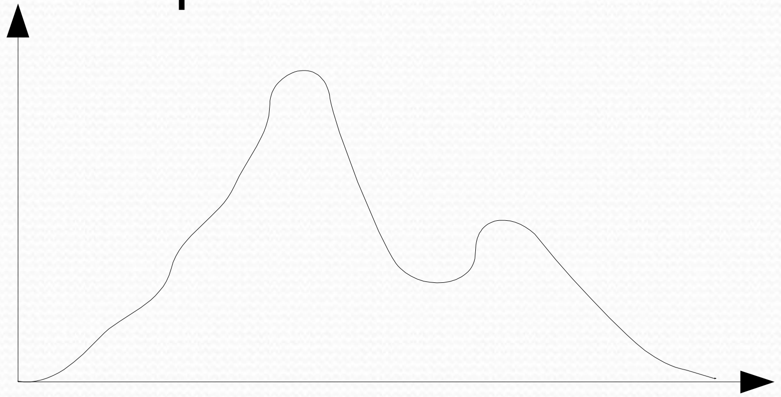
- In the end it all boils down to finding a global maximum (or minimum) of a certain, usually multivariable, function.
- Approach #1 – brute force search. Sample the function at *n* points, distributed randomly or uniformly (or in some other way) and return the best point found. This only works if the dimensionality is low or the problem is discrete and the number of possible solutions is small.





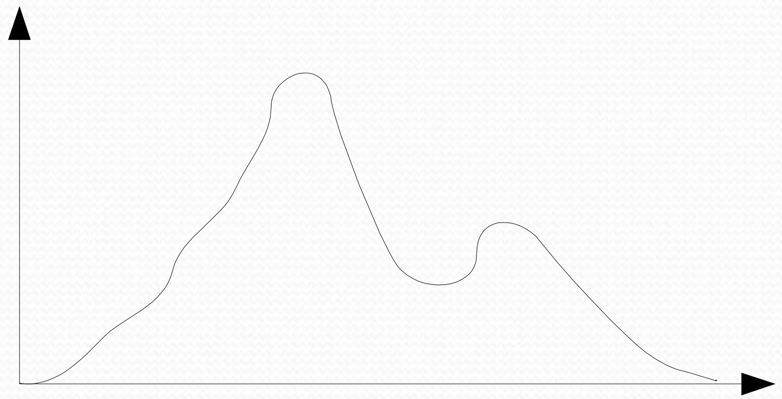
Parameter optimization problem

- In the end it all boils down to finding a global maximum (or minimum) of a certain, usually multivariable, function.
- Approach #2 – gradient ("hill-climbing") methods. Use the information on function gradient to ascend along the gradient and find the “hilltop”.
  - Need gradient information, whether analytical or numerical.
  - If the function is multimodal, may get stuck in a local maximum. Need a starting “guess” point.
  - Fails if the function is not well-behaved (derivative does not exist, function not continuous, etc.)



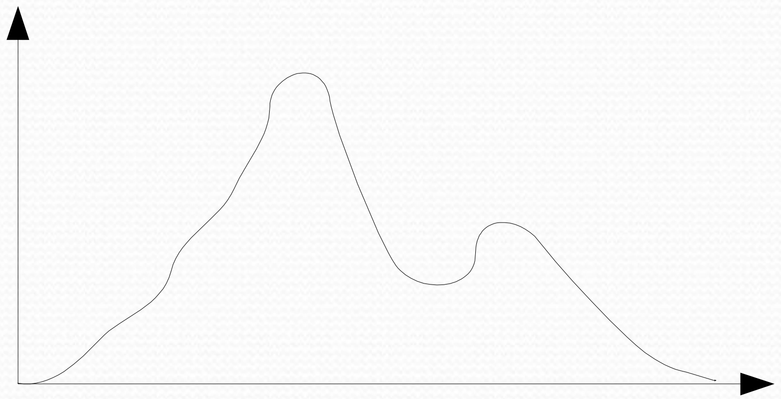
Parameter optimization problem

- In the end it all boils down to finding a global maximum (or minimum) of a certain, usually multivariable, function.
- Approach #3 – iterated search. Combines the first two approaches. Repeatedly climbs hills starting with different, randomly chosen points.
  - Each trial is carried out in isolation, so it does not collect information on the overall shape of the function – the parts of domain that are of low interest are still sampled.

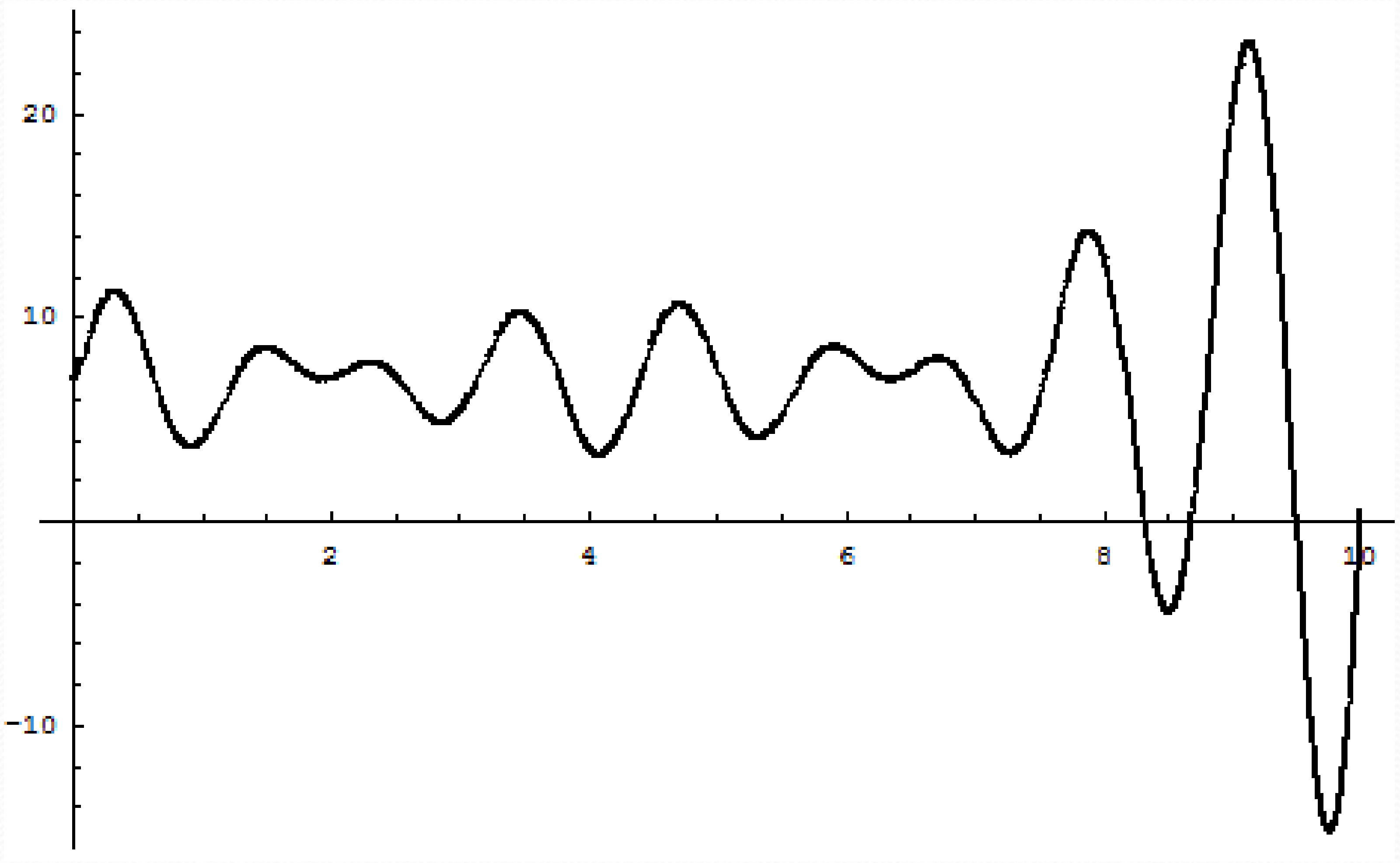


Parameter optimization problem

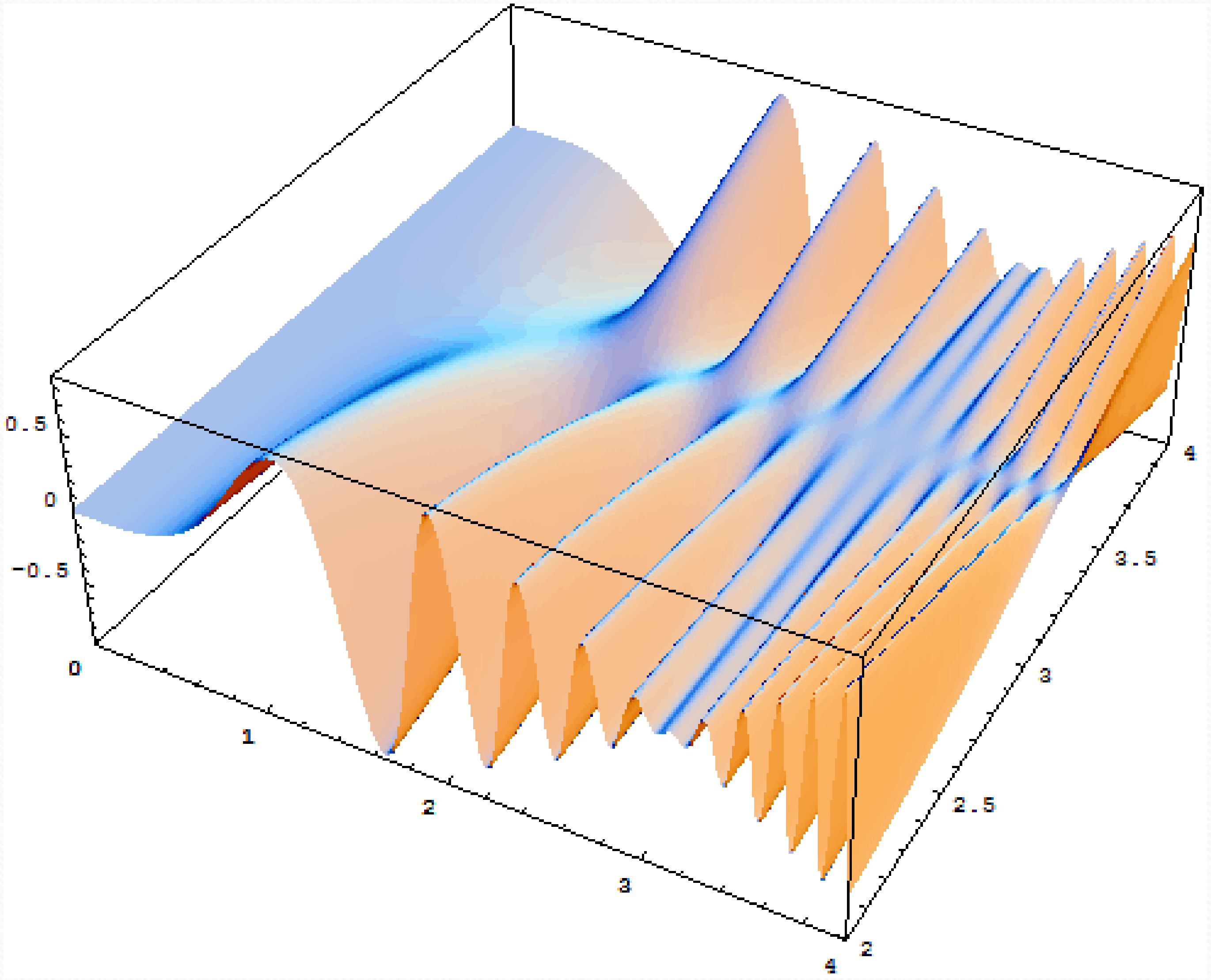
- In the end it all boils down to finding a global maximum (or minimum) of a certain, usually multivariable, function.
- Approach #4 – simulated annealing. A modified version of hill-climbing. We start from random point and try to make a random move. If the move takes us to a higher point – it is accepted. If not, it is accepted with a probability of *p*. The probability *p* varies with time – starting from 1 and slowly going towards 0.
  - Still works with only one candidate solution.



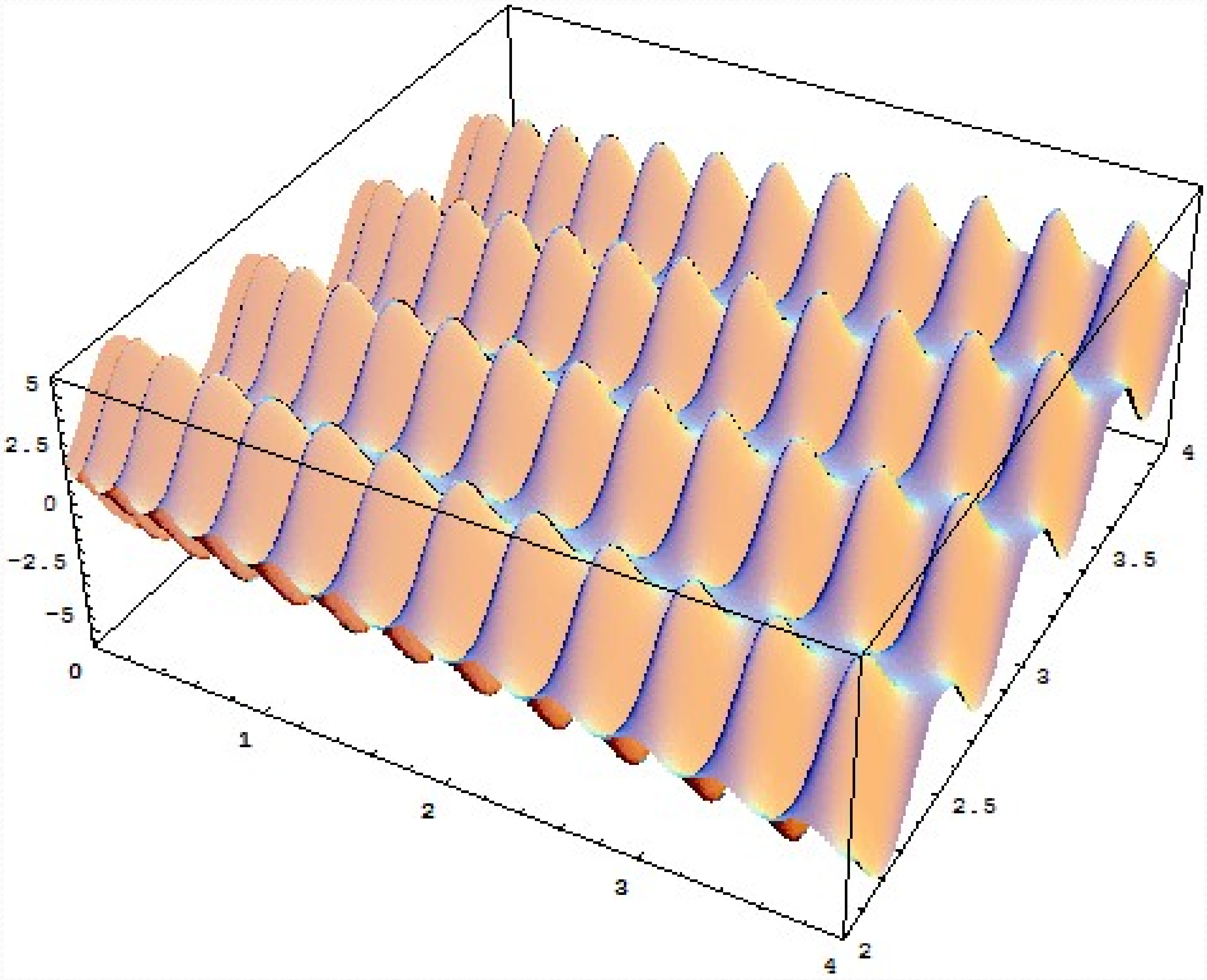
Some difficult functions



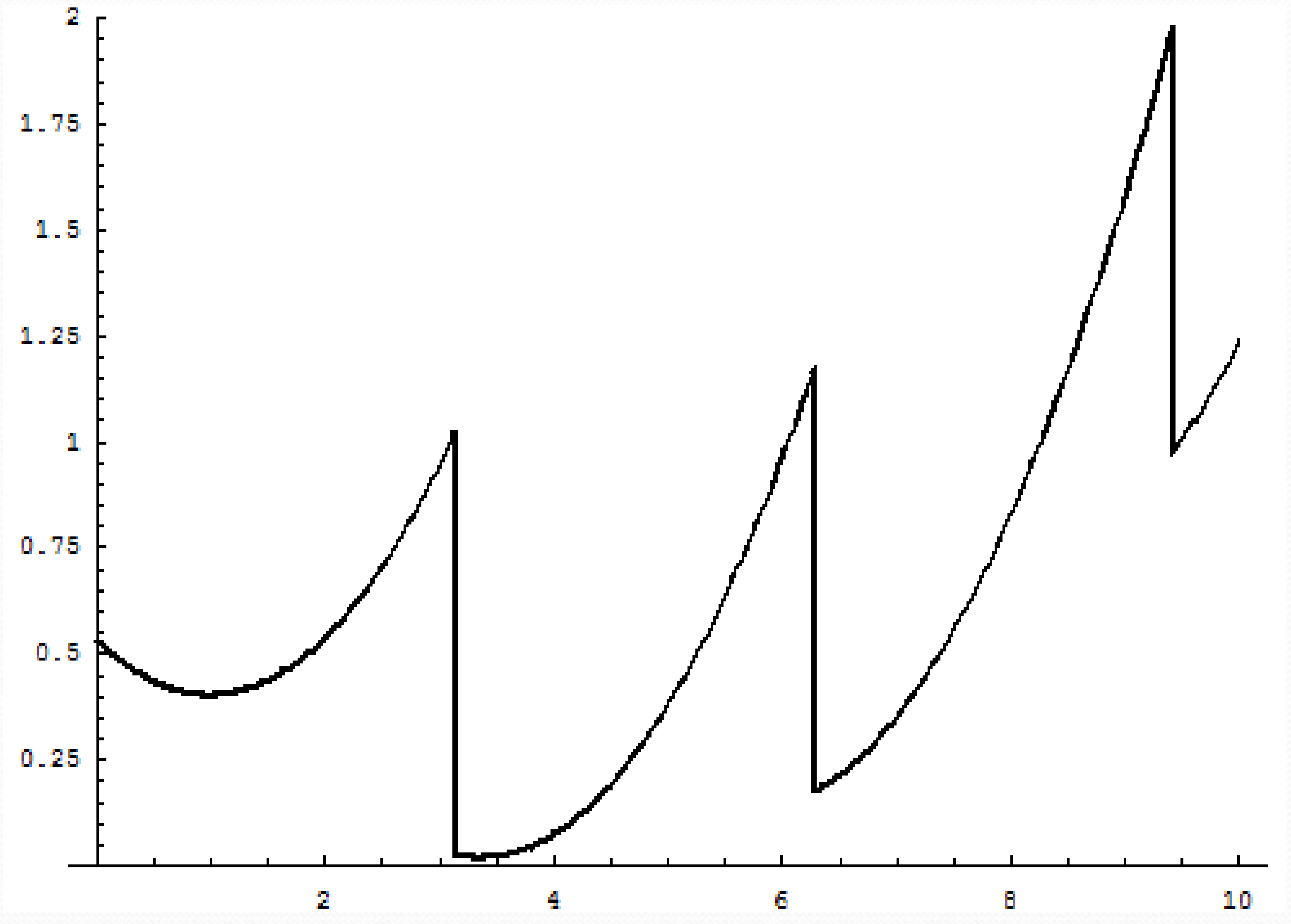
Some difficult functions



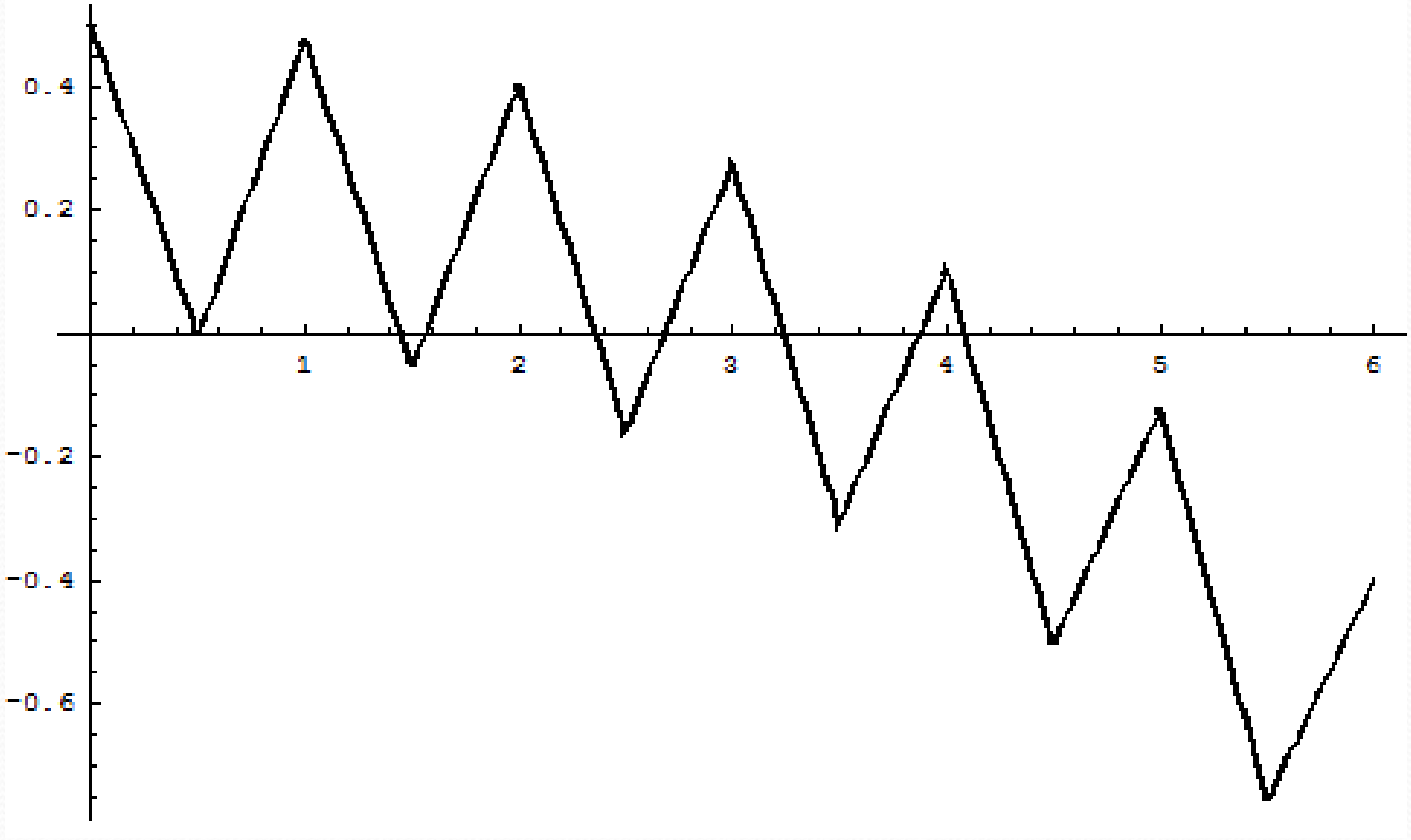
Some difficult functions



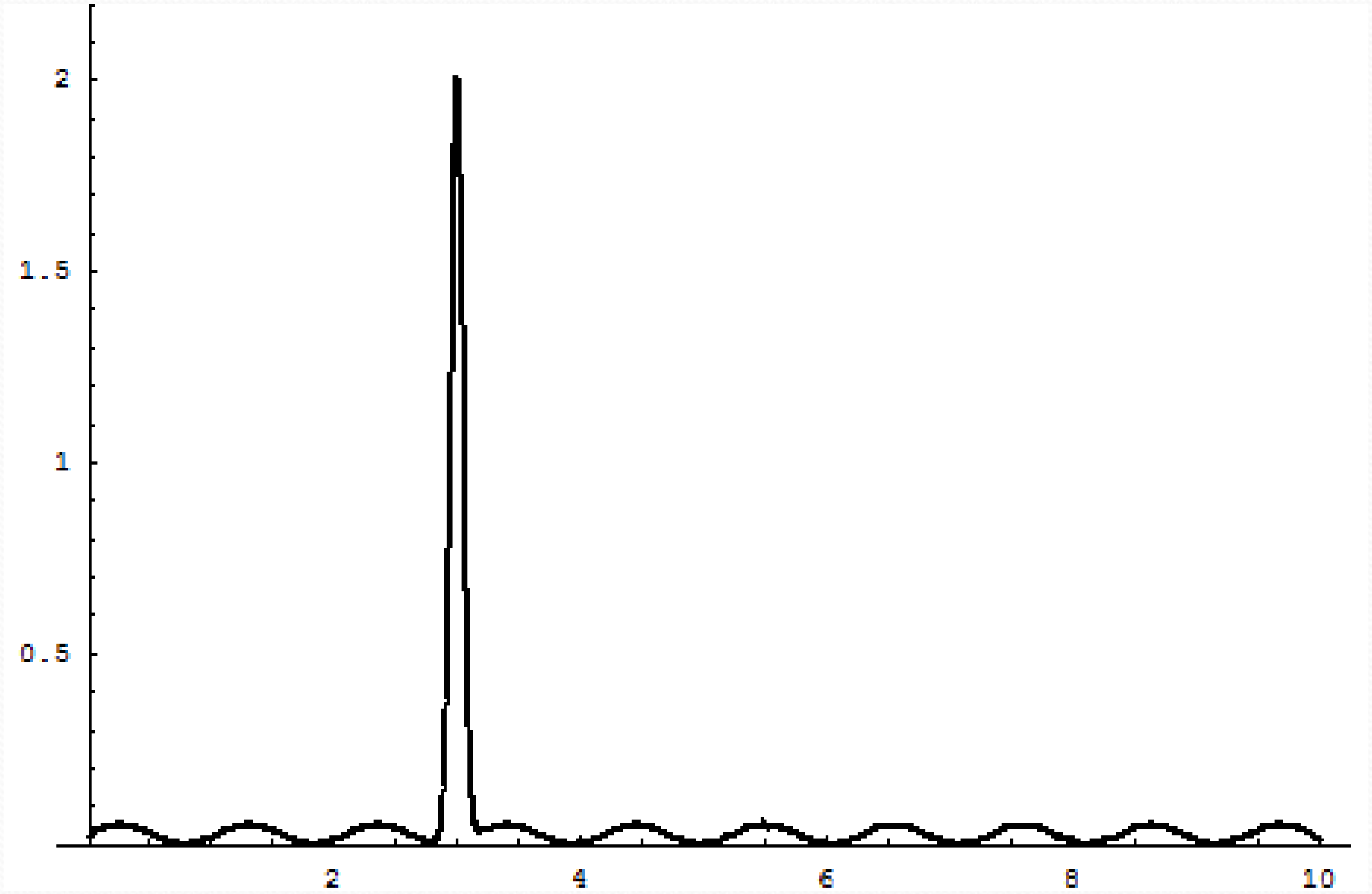
Some difficult functions



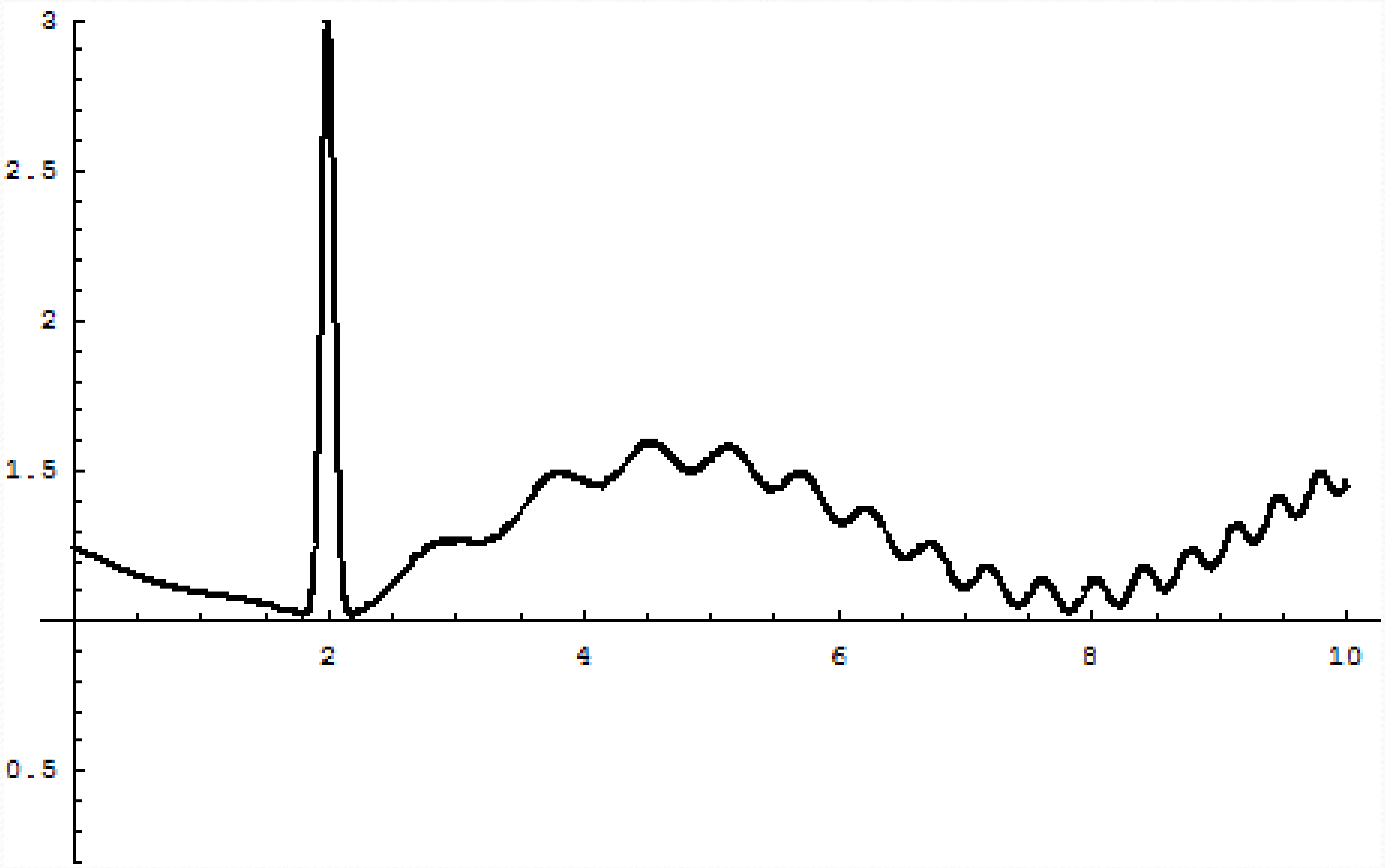
Some difficult functions



Some difficult functions



Some difficult functions



Some difficult functions

- Most functions of high dimensionality.
- The search space for $f(x_1, x_2, ..., x_{400})$ would be huge.

Basic idea behind genetic algorithms

- The previous approaches suffered from:
  - investing all effort into one candidate solution only, or
  - searching with many candidates, but blindly.
- How about a method which will try out many candidates and continually try to improve them?
- This is what evolution does – trying many genetic variations. Those which are successful are more likely to survive until reproductive stage. Failed "experiments" of nature do not live long (or do not reproduce). Thus successful genes reproduce, getting more chances to improve in the next generation.
- Set of possible genetic sequences ↔ search space.
- Good solutions ↔ highly fit organisms.

A bit of biology

- All living organisms consist of *cells*.
- Each cell*) contains (inside the nucleus) one or more *chromosomes* – long strings of DNA that are the “blueprints” or “plans” for the organism.
- Human cells contain 23 pairs = 46 chromosomes.
- DNA strings are ca. 1-2 m (!) long.

*) Red blood cells have no nucleus and contain no DNA.

52 CHAPTER 3 The Cellular Level of Organization

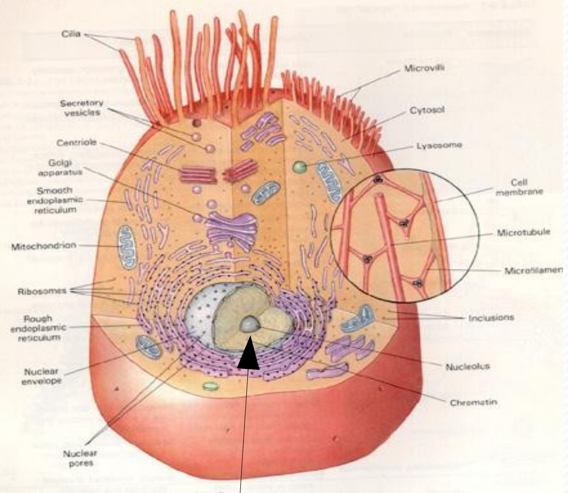


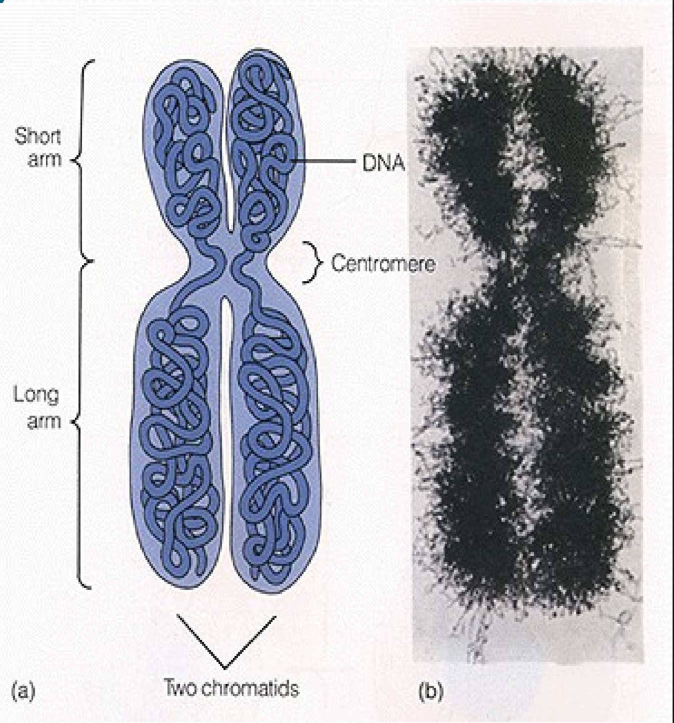
FIGURE 3-3: Anatomy of a "typical" cell
This composite figure should be examined in combination with Table 3-1.

A “typical” cell
(picture stolen)

Here reside the chromosomes!

A bit of biology

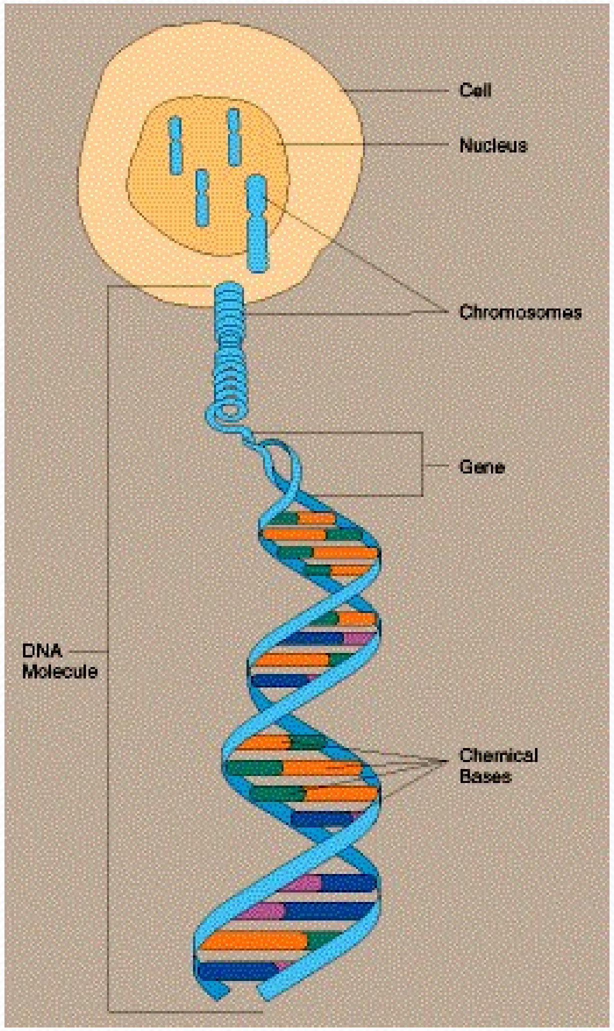
- Chromosomes can be conceptually thought as consisting of *genes* – functional blocks of DNA, each of which encodes a particular protein. Roughly, we can imagine that each gene encodes a trait, e.g. eye colour.
- Different "settings" for a gene (like brown, blue, green for eye colour) are called *alleles*.
- Each gene is located at a particular position – *locus*, on the chromosome.



A chromosome
(picture stolen)

A bit of biology

- DNA encodes information as sequences of bases which act like letters of the alphabet. There are four bases – adenine (A), thymine (T), cytosine (C) and guanine (G).
- The order in which the bases occur in the DNA defines the information carried.
- A typical gene is made up of 10^3-10^5 bases.
- A typical chromosome is made up of 50-250 million bases.
- Typical number of genes: 50-100 thousand.

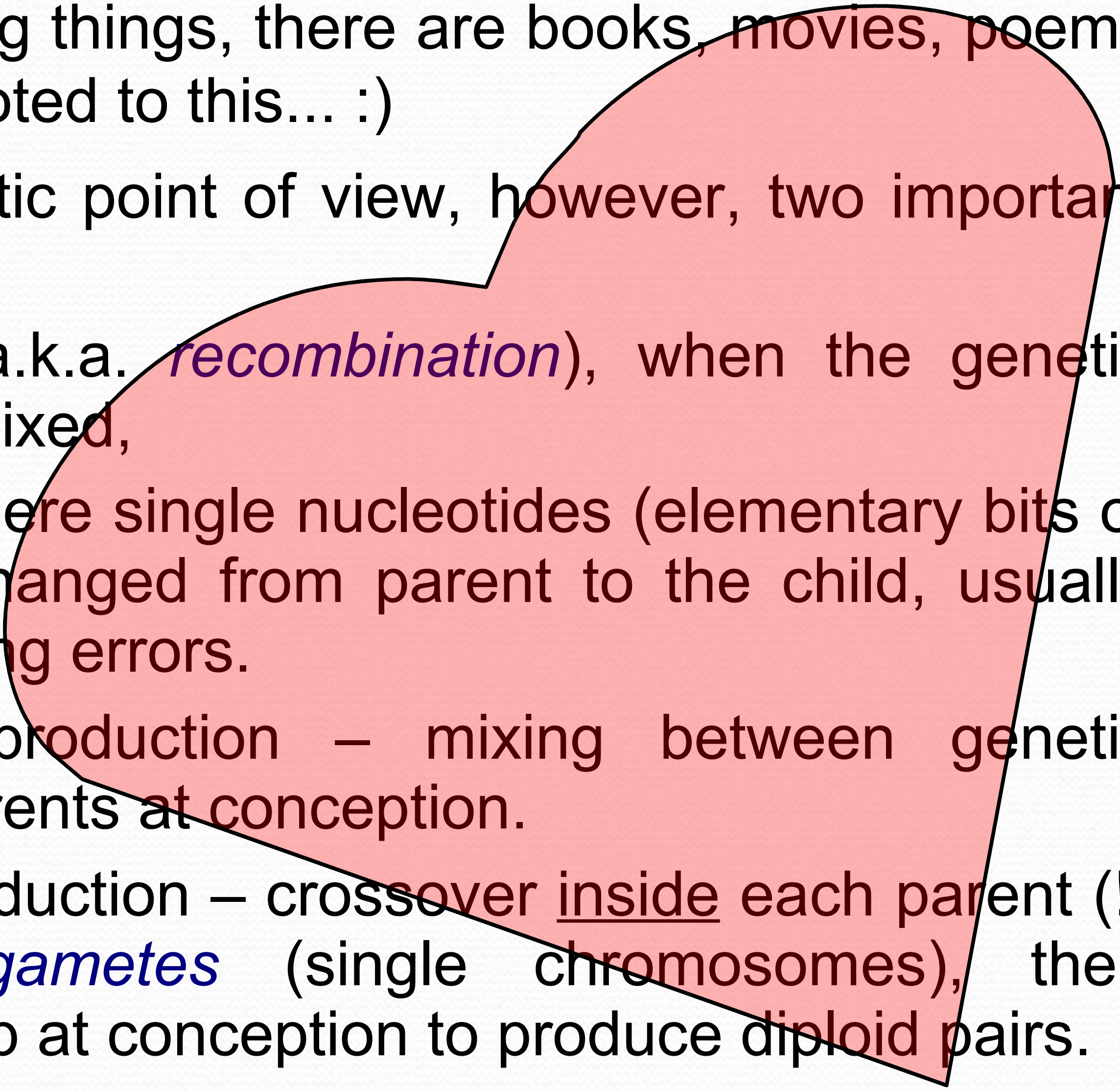


Overview of the situation
(picture stolen)

A bit of biology

- The complete genetic make-up of an individual (his total "genetic package") is called the *genotype*. Different people would have differing genotypes.
- The physical and mental characteristics of an organism (like eye colour, brain size, height, intelligence, etc.) is called the *phenotype*. The phenotype is the result of inherited genotype and the action of environment (during fetal and later development).

What happens during sexual reproduction?



- Many interesting things, there are books, movies, poems and songs devoted to this... :)
- From the genetic point of view, however, two important things happen:
  - *crossover* (a.k.a. *recombination*), when the genetic material is mixed,
  - *mutation*, where single nucleotides (elementary bits of DNA) are changed from parent to the child, usually due to copying errors.
- In haploid reproduction – mixing between genetic materials of parents at conception.
- In diploid reproduction – crossover inside each parent (!) to produce *gametes* (single chromosomes), then gametes pair up at conception to produce diploid pairs.

So much for biology

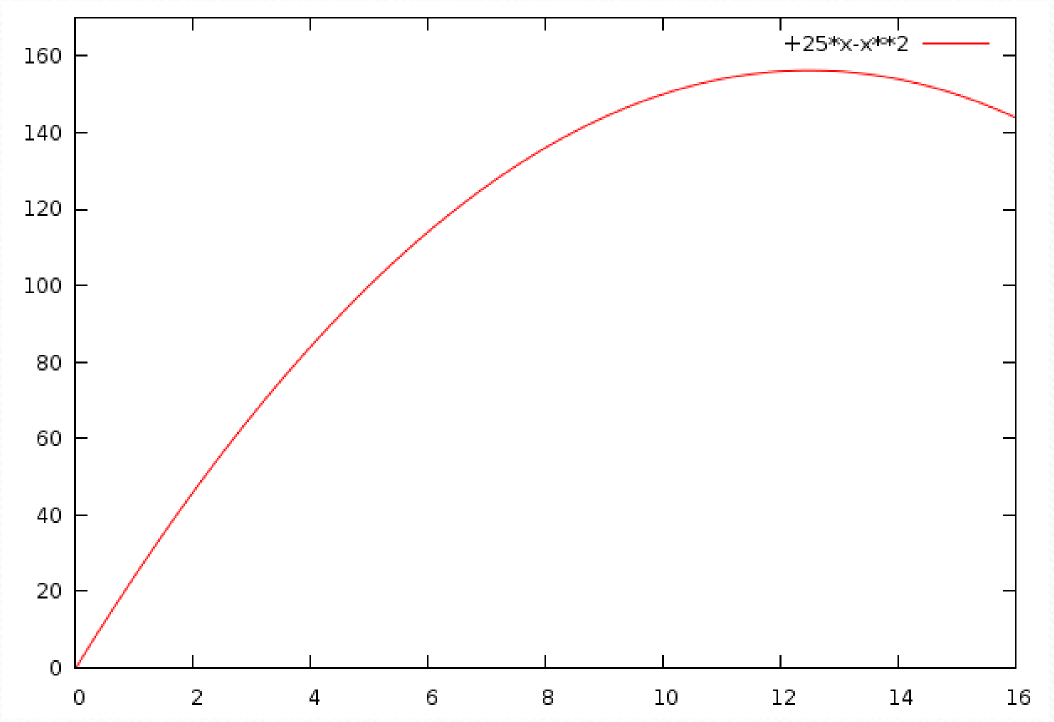
- The moral of the story – evolution looks for fittest individuals, trying to solve a giant optimization problem of finding the best possible genetic make-up.
- Fittest individuals are given chance to reproduce, passing useful genes to their offspring.
- Thus the gene pool constantly improves, i.e. evolution is converging to better and better solutions.
- There are very many candidate solutions (like you and me and the neighbour's cat) evaluated all the time.
- This is the behaviour that we try to roughly mimic with genetic algorithms.

The artificial analog: encoding

- The first thing to do is to find a way to encode each possible solution to the problem we're solving as a set of "genes".
- Remember that a gene is nothing more than a string of symbols.
- In the artificial world typically only two symbols are used – 0 and 1.
- We need to find a way to encode parameter values as strings of 0s and 1s.
- The easiest way is to discretize the (scaled) parameters and treat them as binary numbers.

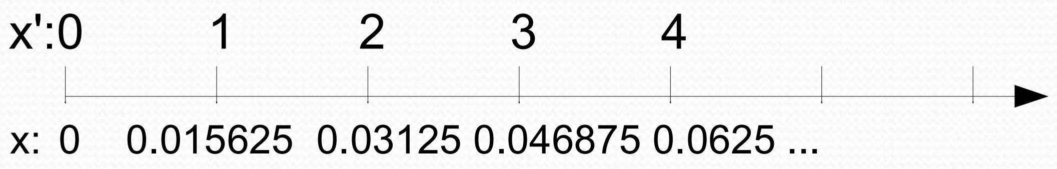
The artificial analog: encoding

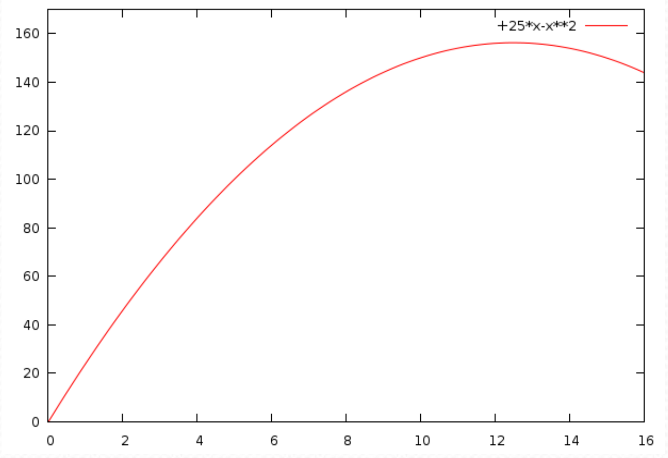
- Let's concentrate on a simple example – we will try to find the maximum of a function of one variable: $f(x) = -x^2+25x$ on the interval $x \in <0, 16>$.



The artificial analog: encoding

- There is an infinite number of values for *x* that need to be tested, but we have to concentrate on a finite subset, i.e. perform a discretization.
- Let's split the interval $x \in \langle 0, 16 \rangle$ into, say, 1024 equal parts (subintervals), each being 0.015625=1/64 in length.
- Let's take *x*'=*x**64 as our new parameter.

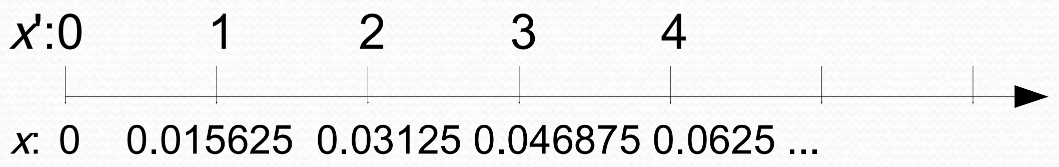


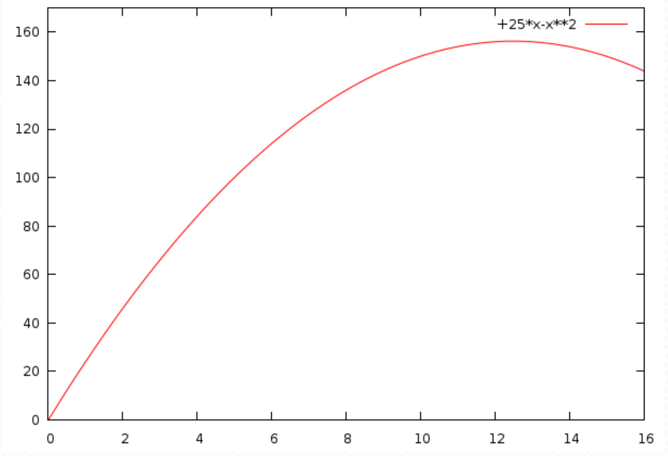


The artificial analog: encoding

- We now have $f(x')$, where $x' \in \langle 0, 1023 \rangle$.
- $x'=0$ corresponds to $x=0$,
- $x'=1023$ corresponds to $x=15.984375$

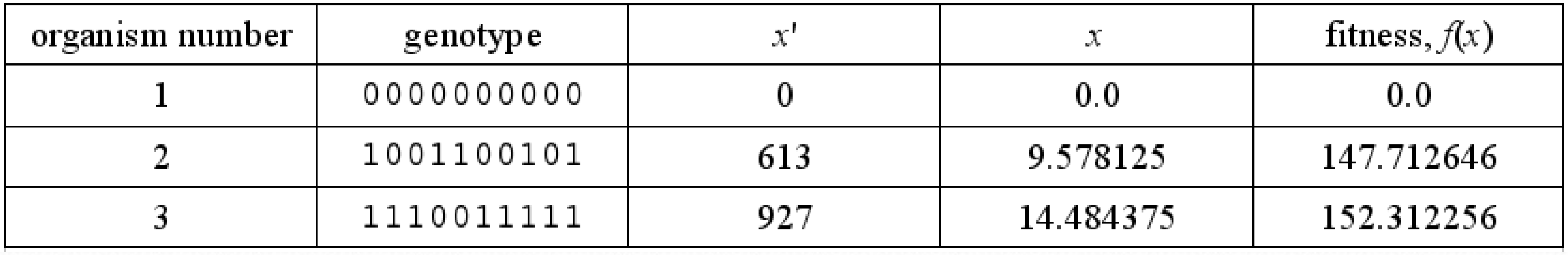
  (we've lost the rightmost point of the doman).
- There are now $1024=2^{10}$ possible values for x'.
- That means that we can encode x' as a binary number: $x' \in \langle 0000000000_2, 1111111111_2 \rangle$.





The artificial analog: encoding

- That is, a chromosome of a length of 10 and an alphabet of “0” and “1” can store the (discretized) value of our parameter *x*.
- Candidate solutions (organisms) have various bitstrings as their chromosomes which represent various *x*'s, e.g.



organism number	genotype	x'	x	fitness, $f(x)$
1	0000000000	0	0.0	0.0
2	1001100101	613	9.578125	147.712646
3	1110011111	927	14.484375	152.312256

- A *fitness function* tells us how good a solution an organism represents. Here we have simply chosen the fitness function to be equal to *f*(*x*), but in general we must ensure that it is nonnegative (usually scaling suffices).

The artificial analog: general procedure

Create an initial population:

1. Generate initial population of *N* organisms, giving them random genotypes.
2. Compute the fitness of each individual.
3. *Produce a new generation:*
   - 3.1 Select two organisms from old generation for mating, **selection is biased towards choosing the fitter ones**.
   - 3.2 Recombine the genetic material of two organisms with a probability p_c, mutate the genetic material with a very small probability, creating a pair of children.
   - 3.3 Compute the fitness of children, insert them into new population.
   - 3.4 If the size of the new population is still smaller than the old population, go to 3.1
4. Replace the old population with the new one.
5. If satisfied with the fittest organism (solution), finish. If population converged (over 95% of individuals have the same genotype), finish. Otherwise go to 3.

The artificial analog

- This is exactly what evolution does, but there are several simplifications:
    - children replace parents after a generation change,
    - every time two children are produced,
    - recombination (crossover) does not look at gene boundaries, genetic material is split blindly.
- Initial population need not have random genotypes – if we know part of the search space is more important, we may create more individuals with appropriate genotypes.
- **The most important point – selection for mating is based on fitness – the fitter an individual is, the greater chances it has to reproduce.**

Why this works, in simple words

- The basic two assumptions are:
  - only the fittest individuals get a chance to reproduce,
  - because of *crossover*, children will usually / often / sometimes be fitter that their parents. (Those which are not will likely not reproduce).
- That way the population evolves towards better (fitter) solutions.
- Thus the search is not blind – the solutions that are better are investigated more in the next generations.
- "*Only the strongest will survive*".

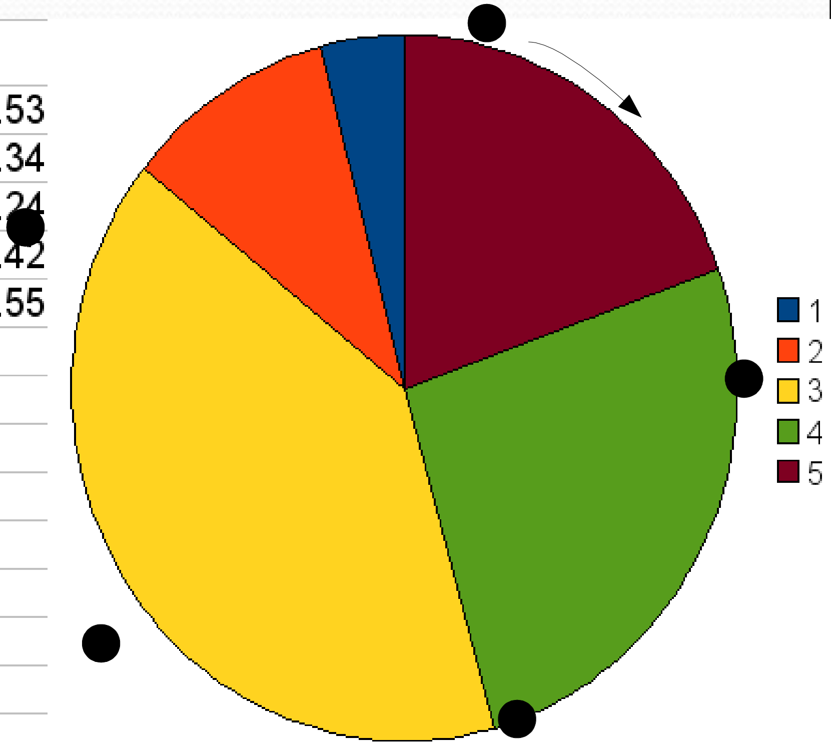
Choosing the fittest – The fitness bias

- We need to select individuals for mating with a bias towards the fittest ones.
- Average mating count must be 1, because new population size must be the same as old population size.
- One simple way:
  - calculate average fitness: $\bar{f} = \frac{\sum_{i=1}^{N} f_i}{N}$,
  - let individual *i* mate $f_i / \bar{f}$ times.
  - the problem is, this number will generally not be an integer. How do you allow an organism to mate 1.65 times?
- The canonical way is *Stochastic Universal Sampling*.

Stochastic Universal Sampling

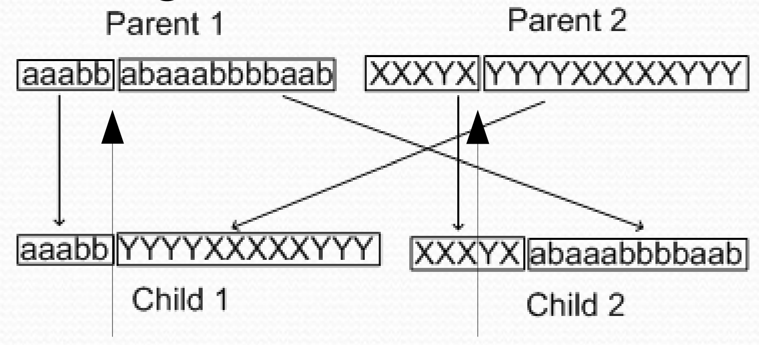
- Assume the population is randomly laid on a pie graph, with the space on the pie graph being proportional to fitness.
- Place an outer roulette wheel around the pie graph with equally spaced *N* points.
- Spin the roulette wheel.
- The *N* individuals that get to mate are instantane-ously selected.

individual, i	fitness, f_i
1	0.53
2	1.34
3	5.24
4	3.42
5	2.55



Details of *1-point crossover*

- The mixing of the genetic material of the two parents.



crossover point

crossover point

- The chromosome is split into two parts at a random point. The resulting parts of chromosomes are interchanged.
- Crossover point may lie at gene boundary by chance, but in general it will not.
- Crossover occurs with a set probability p_c (usually $0.6 < p_c < 1$). If there is no crossover, children are clones of their parents.

Other ways to *cross-over*

- *2-point crossover*.
- *n-point (multipoing) crossover.*
- *Uniform crossover*.

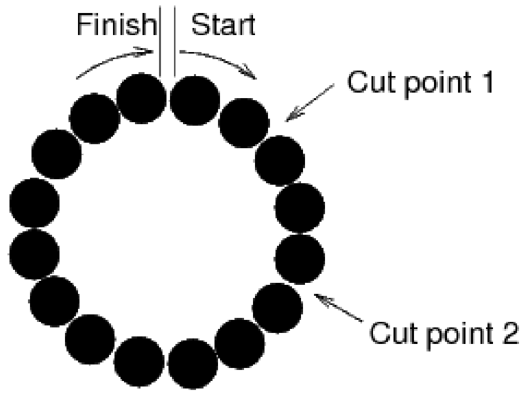


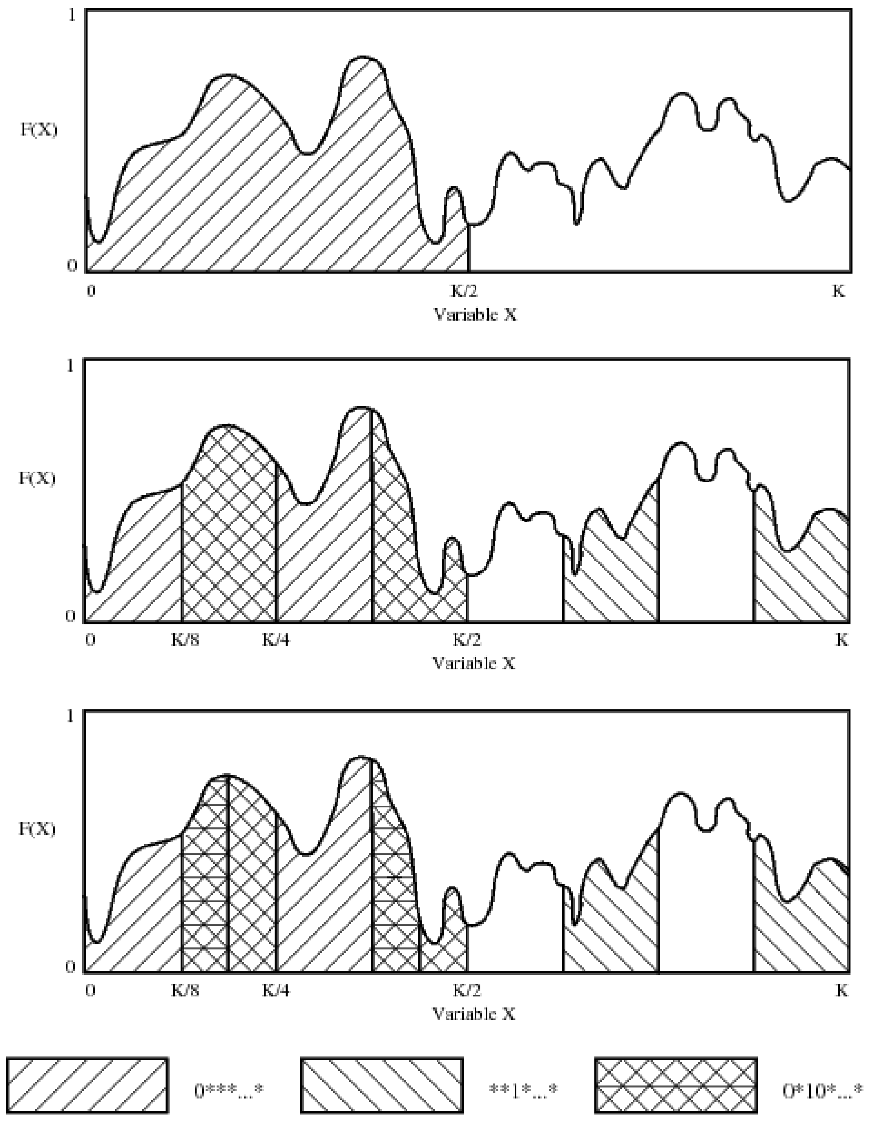
Illustration of 2-point crossover.
Note how the genetic material wraps around.
Image taken from [2].

- No agreement on whether a single crossover method is apparently superior to the other methods. Some are more disruptive than others, which may or may not be a good thing, depending on the problem solved.
  Detailed analysis: [2], p. 2-3.
- Note how shorter genes are less likely to be disrupted by crossover (non-uniform), while longer sequences will be split with a higher probability.

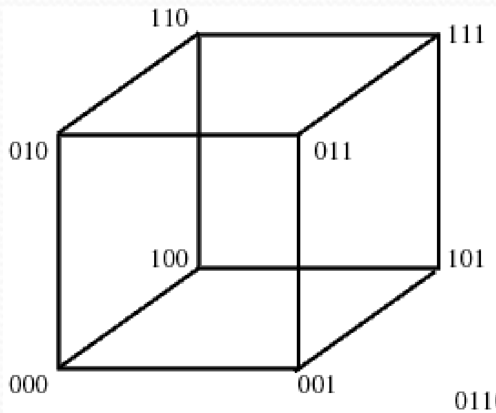
Why GAs work
– a more mathematical approach

- Described on the blackboard:
  - *schema*, *instance* of a schema.
  - *defined bits* of a schema, *defining length* of a schema, *order* of a schema.
  - How many *schemata* of length *L* are there?
  - How many *schemata* does a bit string represent?
  - How many *schemata* does a population represent?
  - Implicit calculation of estimated average fitness of every H present in the population.
  - Low-order as the key to surviving mutation.
  - Short defining length as the key to surviving crossover.
  - ***The Schema Theorem***.

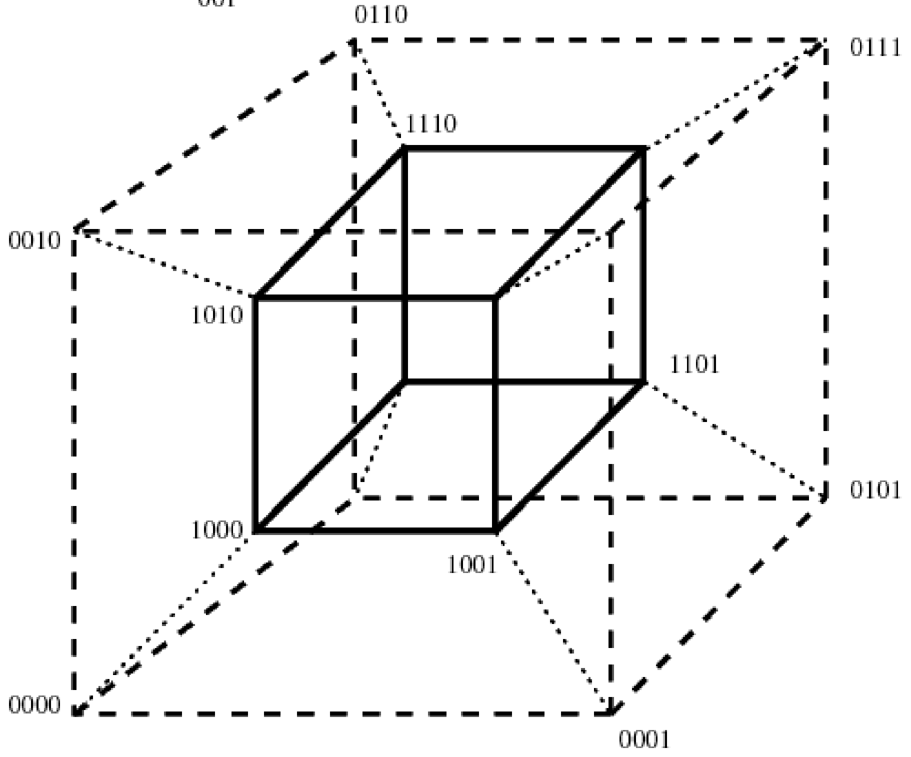
Visualizing schemata



Visualizing schemata

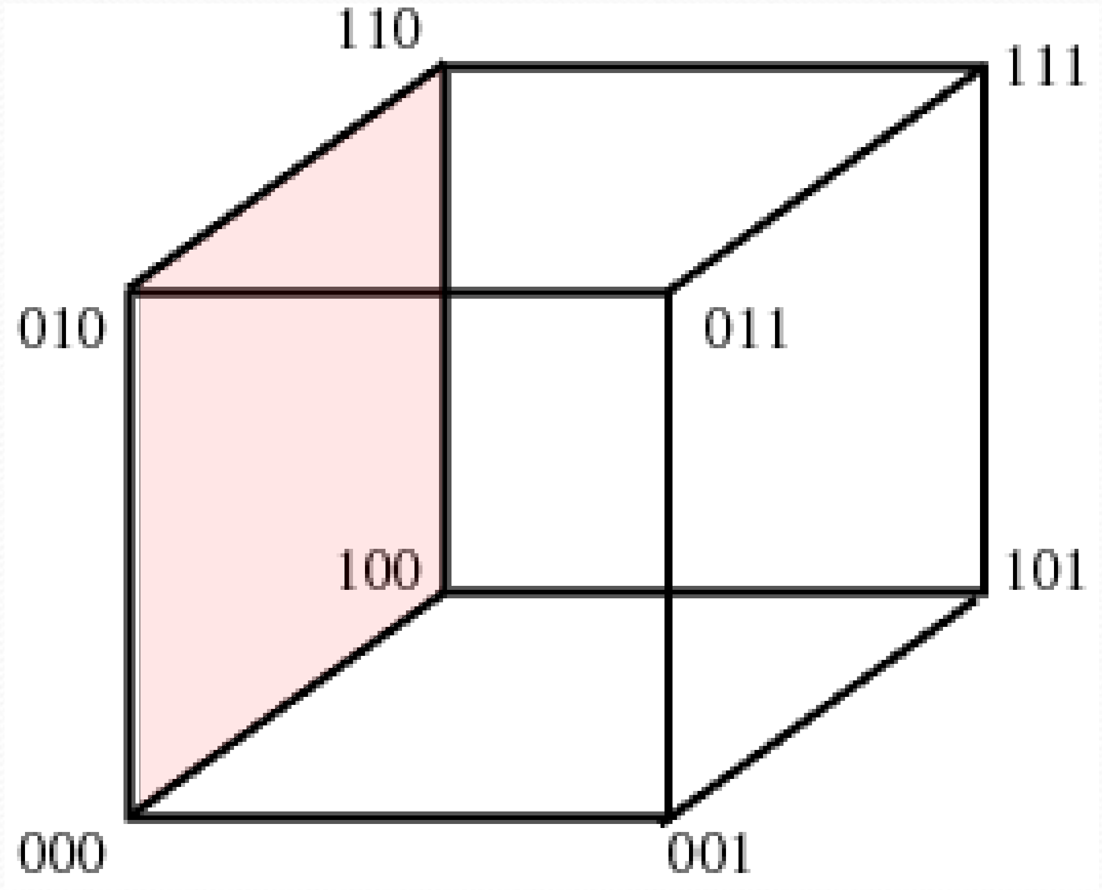


All length-three strings can be arranged on the corners of a 3D cube.



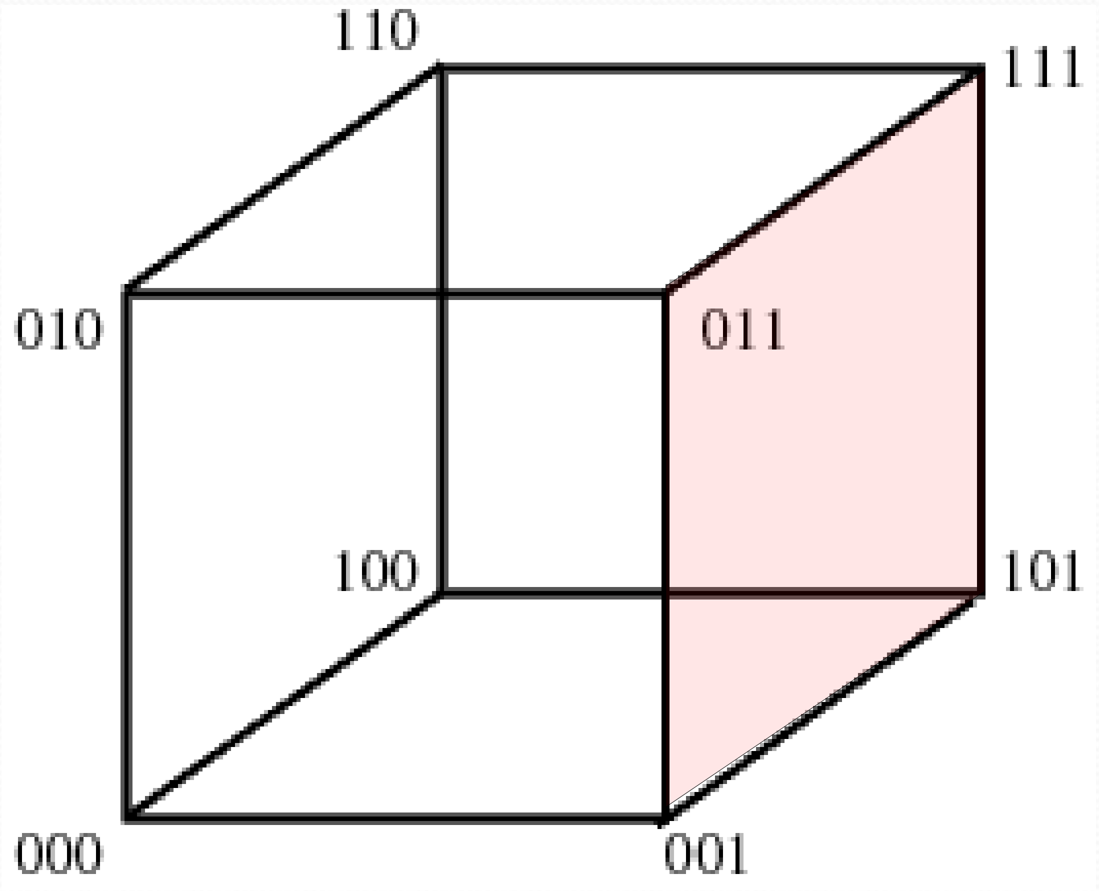
All length-four strings can be arranged on the corners of a 4D cube (a hypercube).

Visualizing schemata



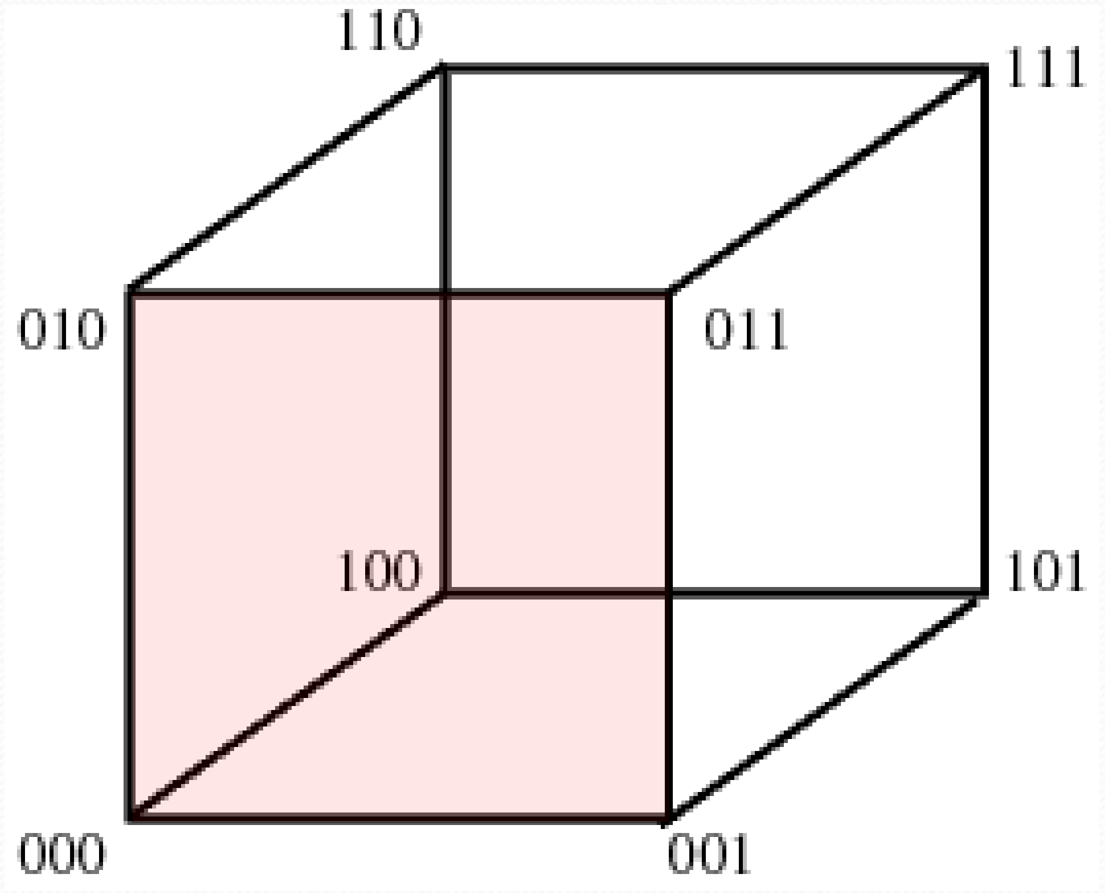
$* * 0$

Visualizing schemata



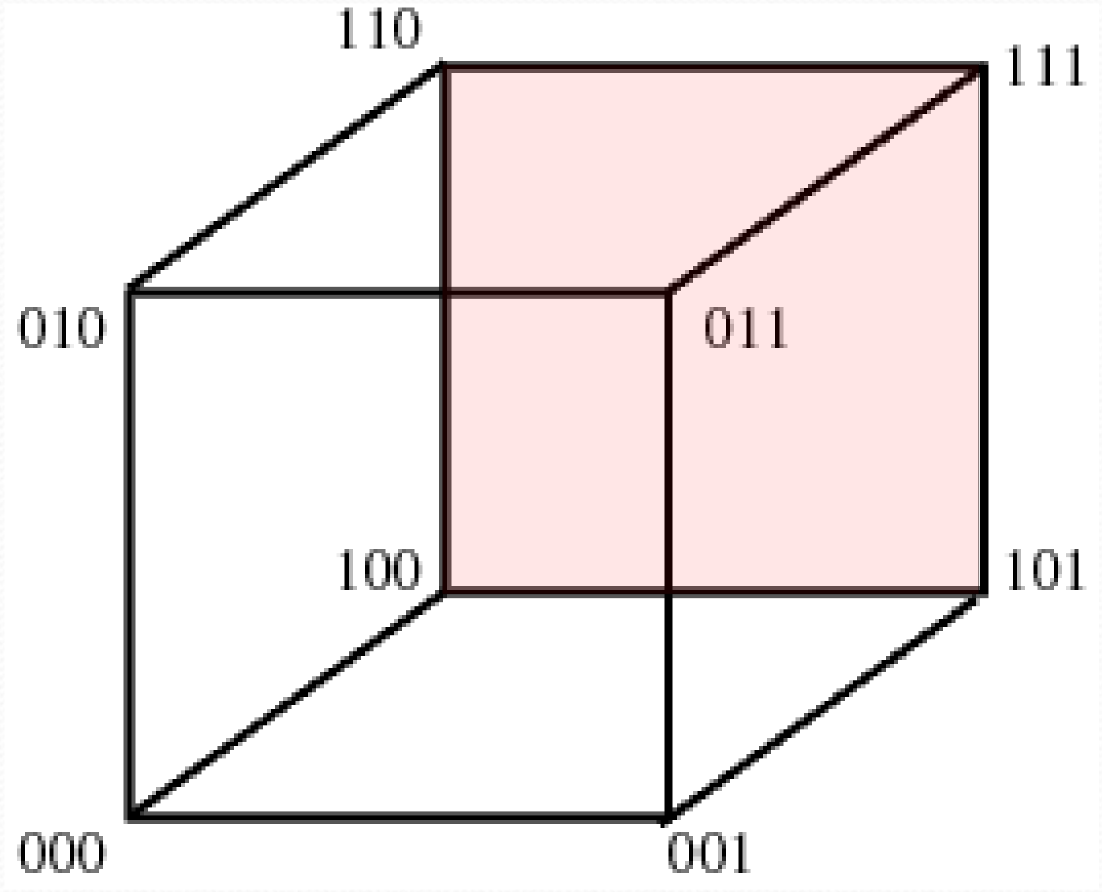
$**1$

Visualizing schemata



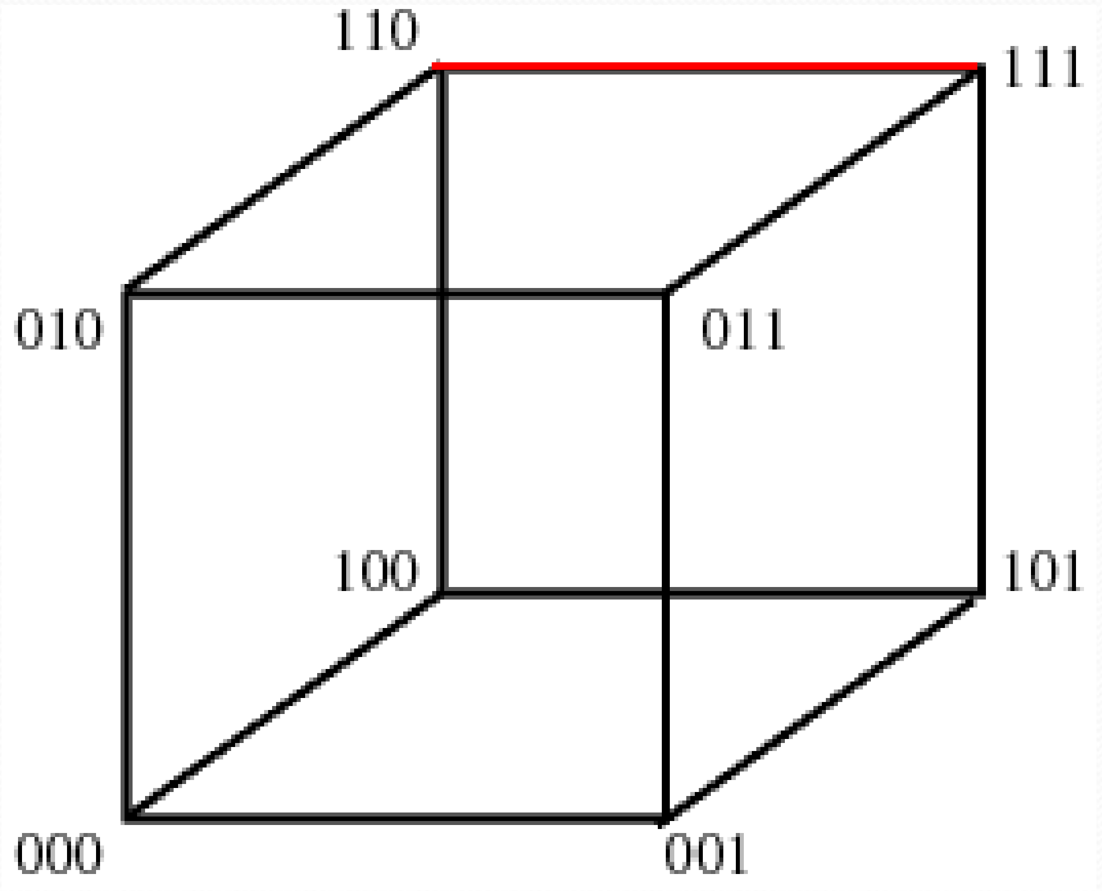
0∗∗

Visualizing schemata



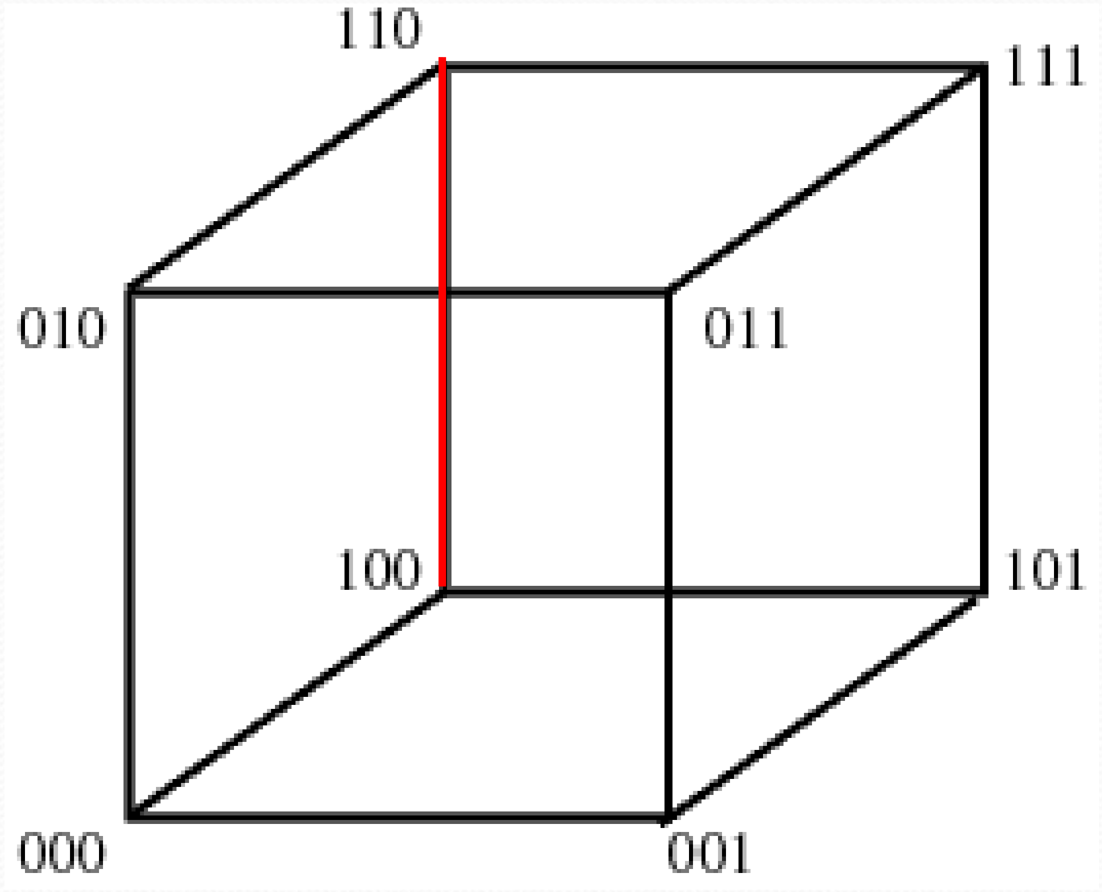
1∗∗

Visualizing schemata



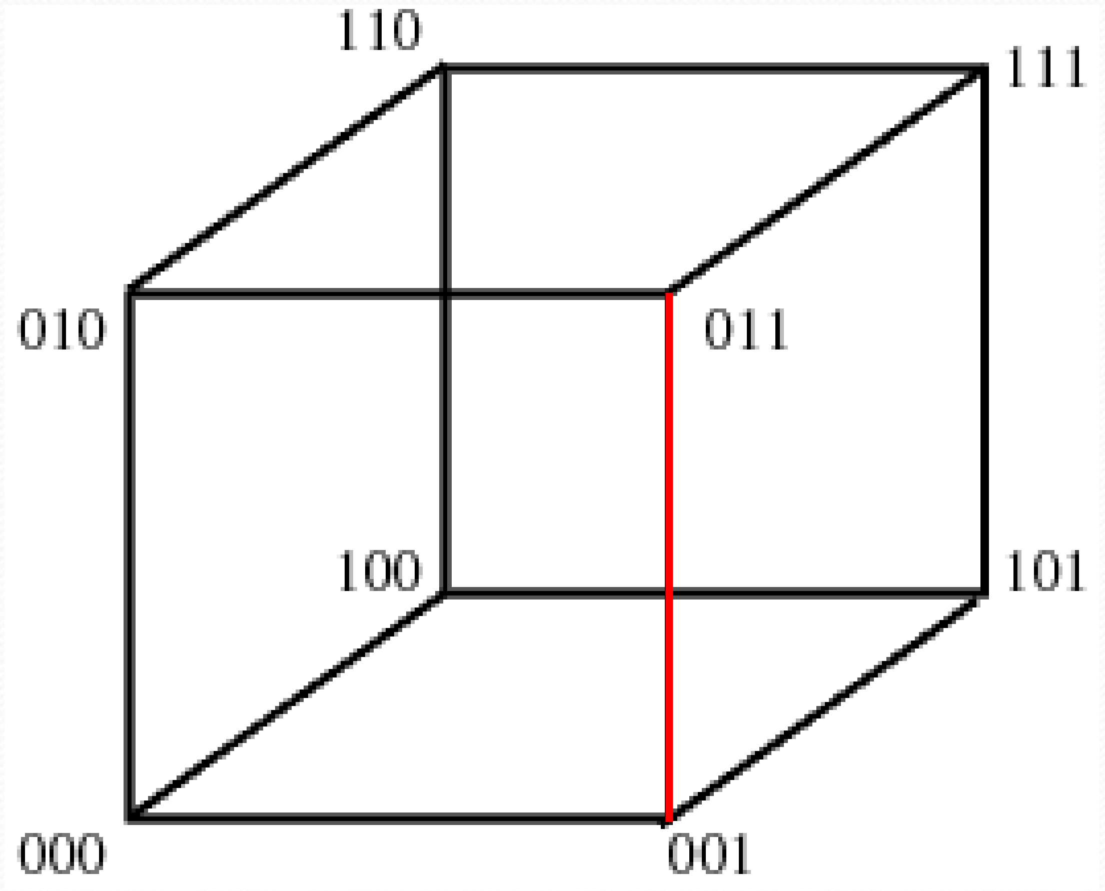
11*

Visualizing schemata



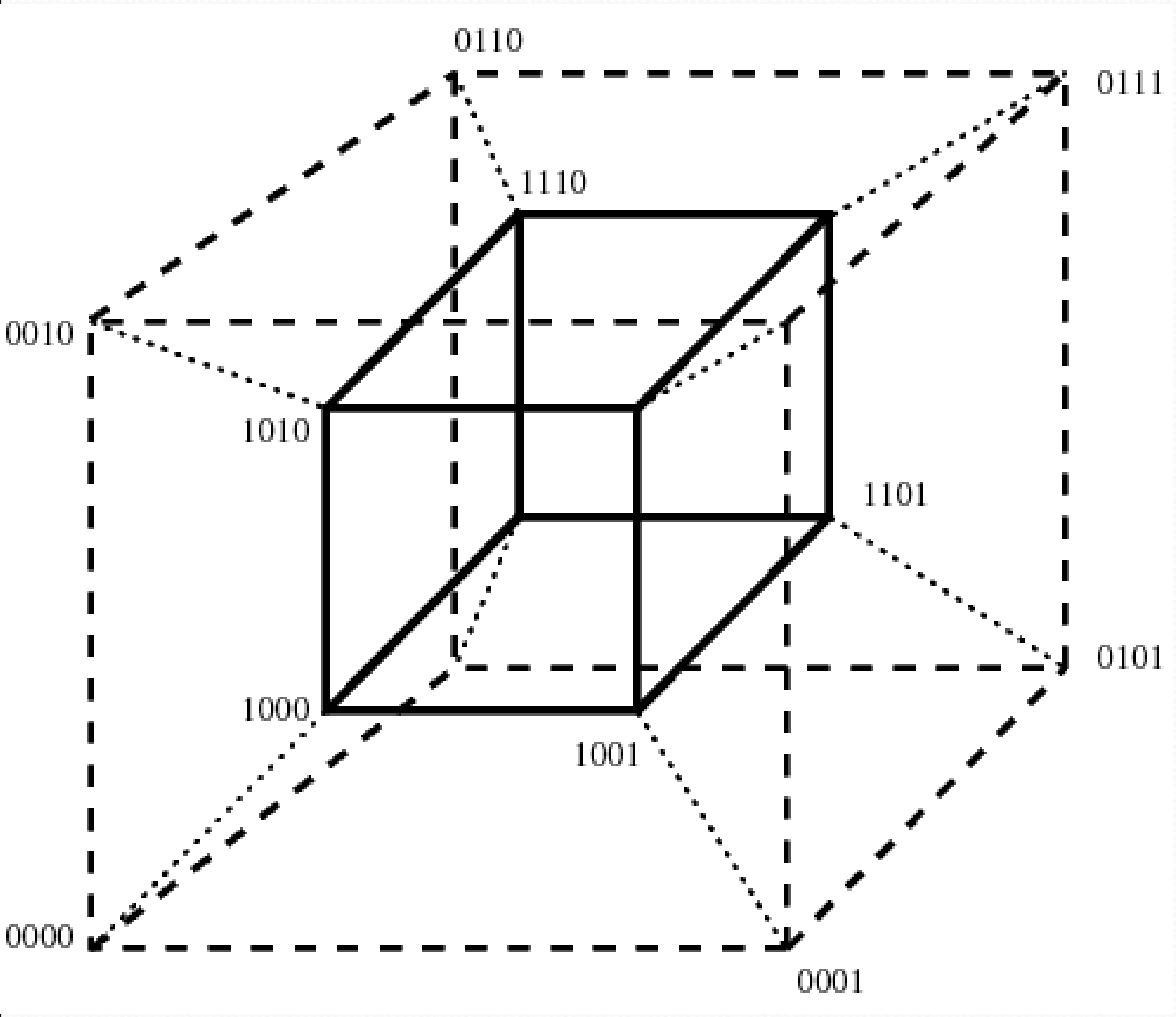
$1*0$

Visualizing schemata



$0 * 1$

Visualizing schemata



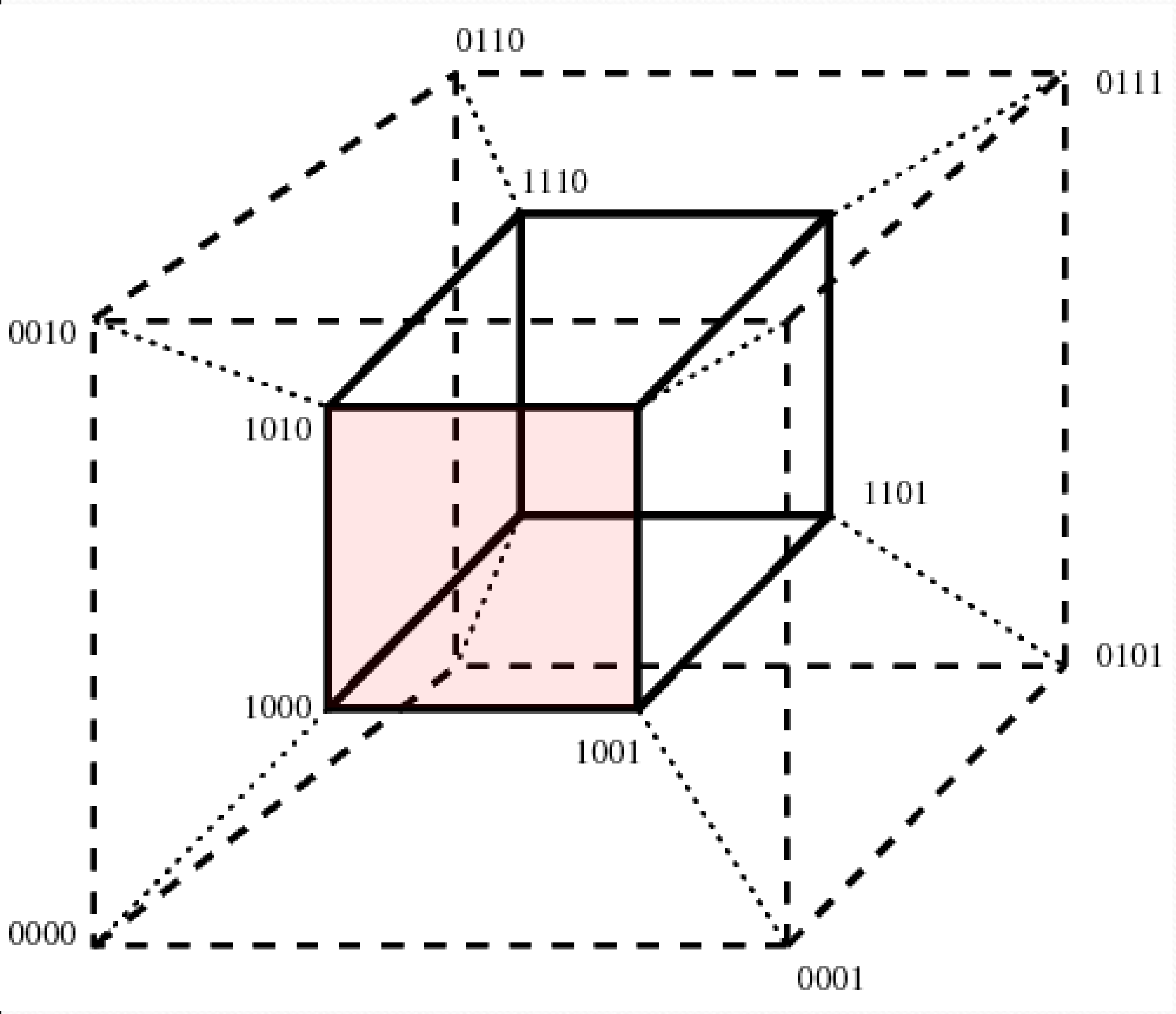
$0***$

is the outer cube

$1***$

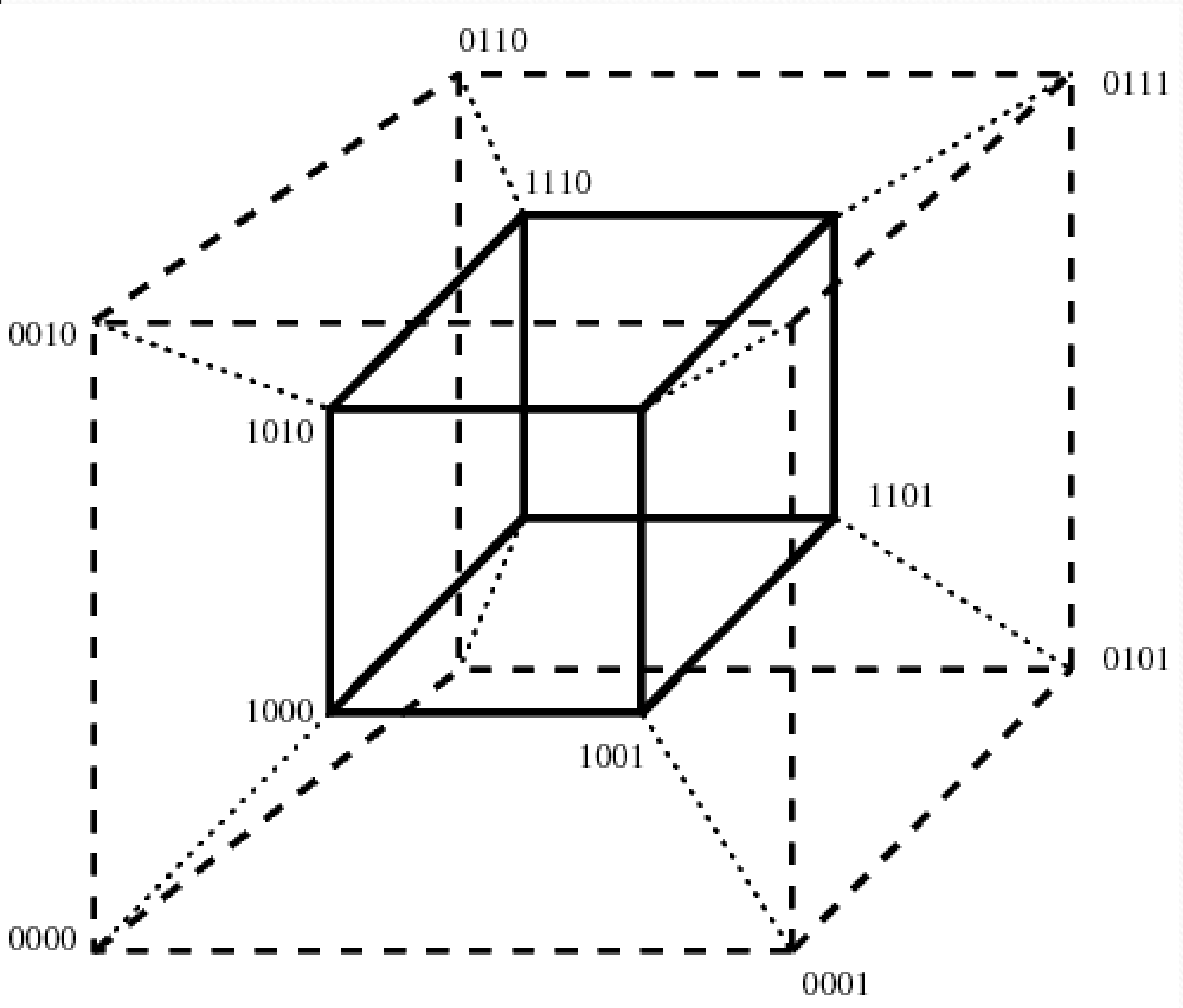
is the inner cube

Visualizing schemata



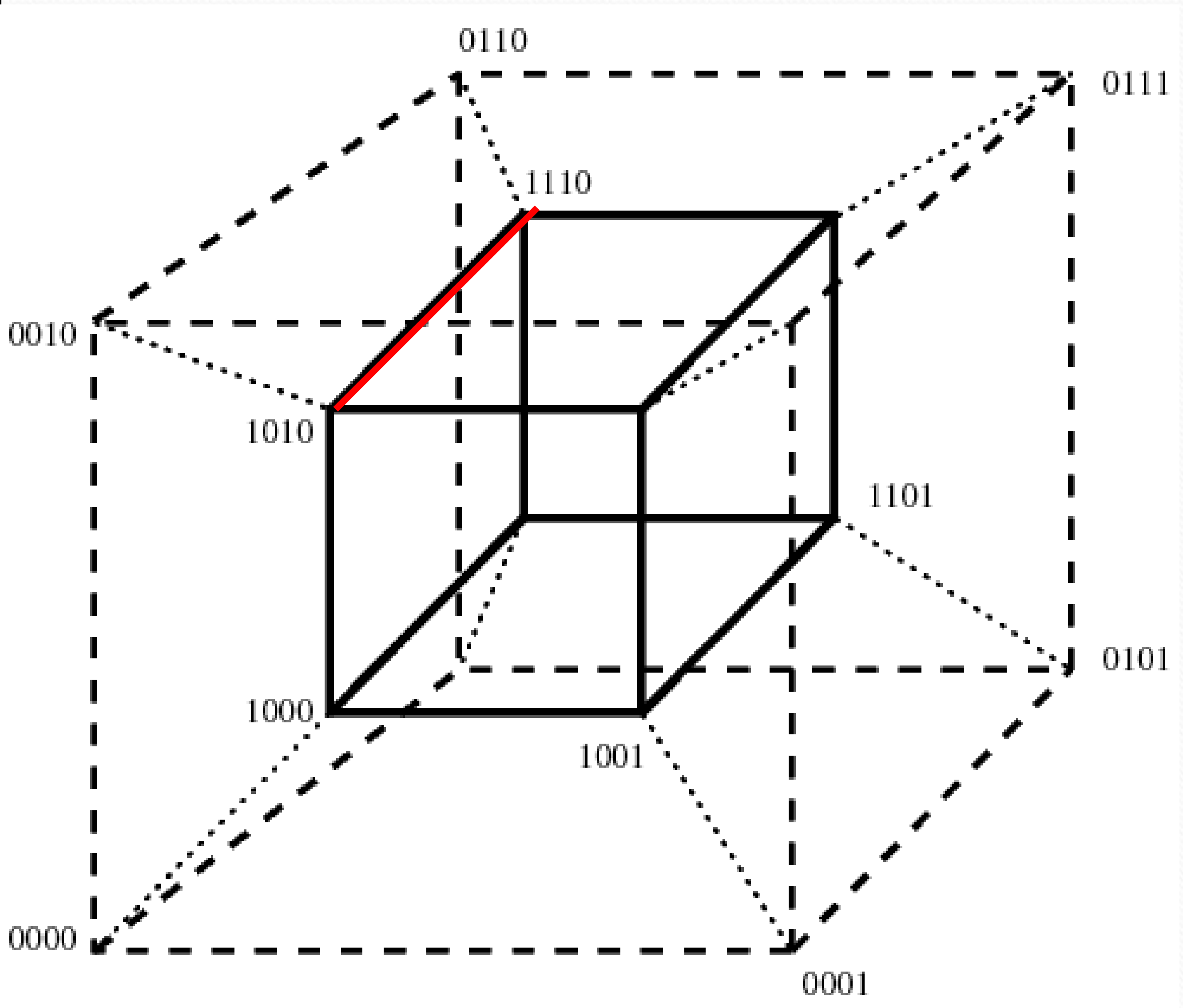
$10**$

Visualizing schemata



$11**$

Visualizing schemata



$1 * 10$

Thus 1-star schemata can be visualized as lines, 2-star schemata as planes, 3-star schemata as “3D-planes” (hyperplanes) and so on...

Then, if we know that the GA samples many schemata at the same time (remember *implicit parallelism*) and each schema corresponds to many points (bitstrings)... we can visualize where the power of GA comes from.

Why GAs work
– a more mathematical approach

- Interpretation of the Schema Theorem (simplified):

Short defining-length, low-order schemata whose average fitness is above the average receive exponentially larger number of samples (instances evaluated) over time – because their number increases by a factor of $\frac{\hat{u}(H,t)}{\bar{f}(t)}$ *in each generation.*

- Remember that we have only considered the destructive power of crossover and mutation, whereas they also have constructive power – especially crossover tries to combine good schemata to obtain even better schemata.

Why GAs work
– a more mathematical approach

- Thus, when evaluating the fitness of the population of *N* strings, the GA implicitly evaluates the average fitnesses of all schemata that are present in the population, and increases or decreases their representation according to the Schema Theorem.
- Selection acts in such a way, that more and more samples are rapidly given to the schemata whose fitness is (estimated to be) above average.
- Good schemata get exponentially increasing number of samples in subsequent generations.
- Poor schemata get exponentially decreasing number of samples in subsequent generations.

Why GAs work
– a more mathematical approach

- The short defining length, low-order schemata with fitnesses above average are called *building blocks* – they act as partial solutions to the problem, as the solution candidates are composed of them.
- We found out that between 2^l and $N \cdot 2^l$ schemata are represented in a population. How many *useful schemata evaluations* (evaluations of schemata that will survive to the next generation with a set, constant probability p_s) are made at each time?
- It can be proven that this number is proportional to N^3 with the proportionality constant depending on p_s.
- This is a suprising result if we keep in mind that we only need computer memory and time proportional to N.

Why GAs work

– a more mathematical approach

- The conclusion that we get $c \cdot N^3$ useful schemata evaluations with a population of only N is termed *implicit parallelism*, reflecting the fact that GA implicitly evaluates many schemata simultaneously.
- The *Building Block Hypothesis* is a less formal way of explaining the power of GAs. It simply states that GAs *work by discovering, promoting and recombining good building blocks, in a highly parallel fashion*. Here the building blocks can be simply thought of as "combinations of bits which confer higher fitness on the strings that contain them". These building blocks do not necessarily need to have schemata representations, but schemata are easiest to think about.

Why GAs work
– a more mathematical approach

- The role of the various parts of the GA algorithm can be summarized as follows:
  - *Selection* increasingly focuses the search on subsets of search space which have estimated above-average fitness. It simply pushes the search towards the regions which we think are more likely to contain good solutions.
  - *Crossover* tries to combine good building blocks on the same bit string to build strings of better and better fitness.
  - *Mutation* is an insurance policy against irreversibly losing genetic diversity at a certain locus. This is important if the population is too quickly dominated by copies of superfit individuals.

Exploration *vs.* exploitation

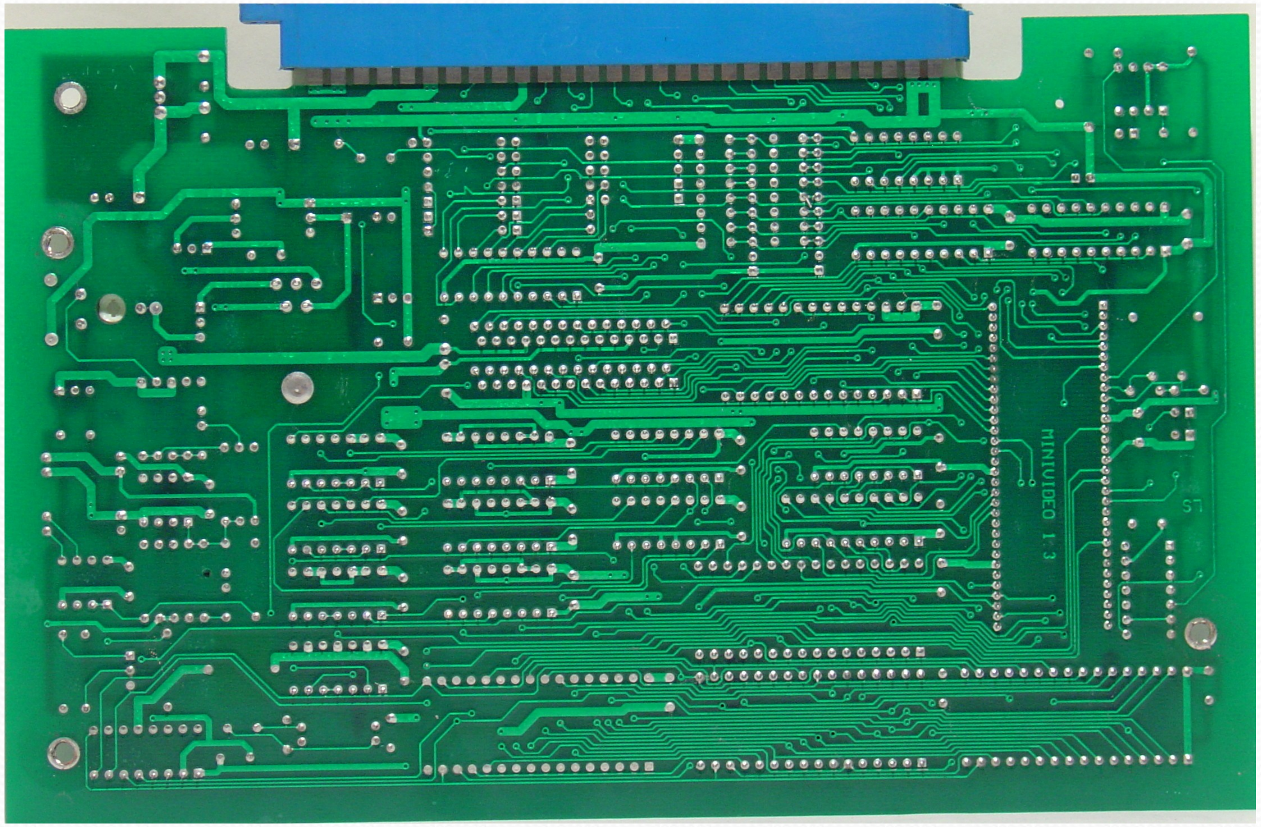
- The search is a constant struggle between two forces:
  - *exploration* – the search for new, useful solutions,
  - *exploitation* – using and propagating (exploiting) the useful solutions that have already been found.
- The struggle comes about because any move towards exploration (trying new solutions in the parts of search space that have so far given poor results) takes away time from exploitation (concentrating on the good solutions found already).
- The algorithm must balance these two forces:
  - if we move too far towards exploration, we search blindly, not paying attention to the good results we have found already
  - if we move too far towards exploitation, we may overadapt (get stuck in a local maximum and forget about trying new possibilities).

Why GAs work
– a more mathematical approach

- The competition then moves from low-order schemata, to higher-order schemata as time proceeds.
- The fact that selection rules change with time (because they depend on the current average fitness, $\bar{f}(t)$) complicates the matters even more.
- In fact, until today we only understand the “rough general picture” of how it works.

Applications
– quick overview

- Optimization, especially with otherwise difficult problems. We now understand well why they are applicable there. The PCB layout problem is a classic example.



Applications
– quick overview

- Design. This can involve a mix of combinatorial and functional optimization. For example the structure of a bridge must be designed so that it withstands a set of loads, has defined length and must be as cheap as possible. The combinatorial part involves designing the shape and then functional optimization deals with the dimensions of the components. GAs often try things that a human designer would never have thought of – they do not have preconceived ideas ("*this is usually done like that*" or "*that is supposed to look like this*") and are not afraid to experiment. Many alternative designs can be produced quickly, which are then assessed by human engineers.

Applications
– quick overview

- Automatic programming. Evolving computer programs for specific tasks. The genetic algorithm can be used to evolve the rules of a cellular automaton with time, to create rules that better reproduce the desired behaviour.
- Population genetics. As a model for evolution. Attempts to answer questions like “under what conditions will a gene favouring recombination of other genes be evolutionally viable?”.
- Social systems, especially their evolutionary aspects. How does the social behaviour in insect colonies appear? Under what conditions will the agents in a multi-agent system communicate and cooperate?

Applications
– quick overview

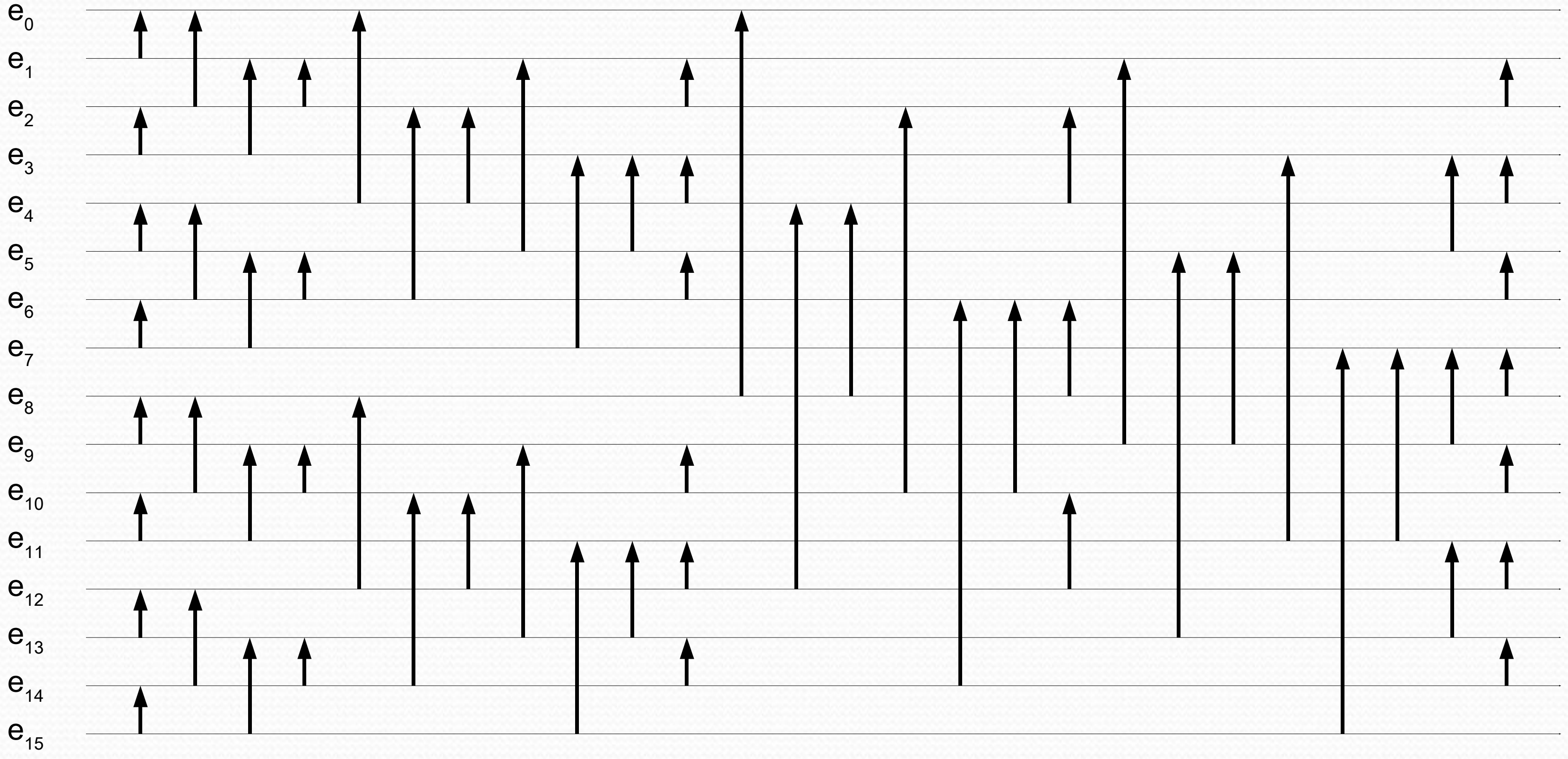
- Combinatorial problems. Recall the travelling salesman problem. GAs can find near-optimum solutions with cases of hundreds of cities. This can be used to find routes that are shortest or cheapest or fastest (or close-to-best) for train-journey planners, GPS navigationdevices, etc..
- Combinatorial problems – typically require different encodings than typical optimization (because of the discreteness).
- Combinatorial problems. Job-shop scheduling – there is a set of a resources in a company (such as machines, people and rooms) and a set of tasks, like manufacturing some batches goods. The resources must be assigned to task with the constraint that the same machine/person/room cannot be used to do two things at the same time. Cheapest, fastest, most efficient or otherwise “best” or “close-to-best” arrangements can be sought for with GAs.

More applications – parallel sort

- Sorting elements is fundamental to computer science.
- Many efficient method exist (e.g. QuickSort, HeapSort, MergeSort), but in the traditional formulations they are serial, i.e. work on a single processor.
- A parallel sorting technique, which could utilize many processors would be of interest, especially nowadays when parallel computing is becoming more popular.
- An example parallel implementation, which sorts 16 elements follows.

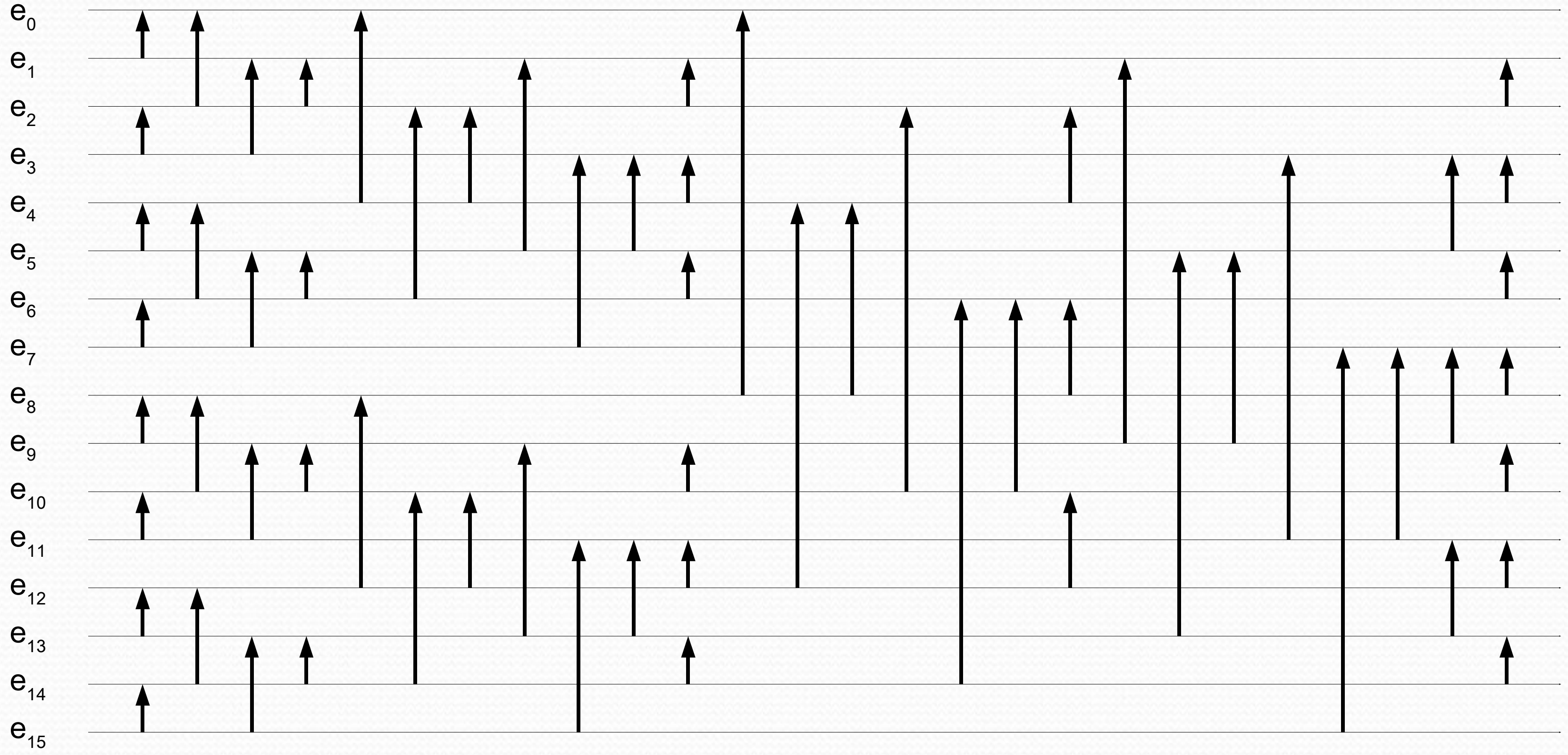
More applications – parallel sort

↑ means: "*compare the two elements and swap them if they are not in the right order*"



- A parallel-sorting network ("*Batcher sort*") for a list of n=16 elements. It takes k = 63 compare-and-maybe-swap operations to sort the list. Comparisons that are in the same column can be made in parallel, so the time to sort would be that of (roughly) t = 26 comparisons.

More applications – parallel sort



- Important questions:
  - What is the minimal sorting network for this case (minimize *k*)?
  - What is the minimal parallel sorting network for this case (minimize *t*)?
  - How, in general, does one find minimal (or close-to-minimal) networks?

More applications – parallel sort

- In 1962 Bose and Nelson developed a method of designing sorting networks for n = 16, assuming k = 65 comparisons. They were fairly convinced that this was a minimum value.
- In 1964 Batcher discovered the network with k = 63 (the one shown in the picture). He was fairly convinced that this was the minimum value.
- In 1969 Shapiro constructed a network with k = 62. He knew better not to assume this was the minimum value.
- The same year Green found a network with k = 60. There is no proof that this is a minimal network.
- In late 1980's Hillis tried a GA on the problem.

More applications – parallel sort – Hillis' experiment

- Hillis tried to find the minimum network, not the most parallel version (so he optimized for *k*, not for *t*).
- The network can be encoded as ordered list of pairs, like (2,5), (4,2), (7,14), ... – meaning first compare-and-maybe-swap elements 2 and 5, then 4 and 2, then 7 and 14 and so on.
- Every element number can be encoded in four bits: $<0, 15> \rightarrow <0000_2, 1111_2>$, so a pair can be encoded in 8 bits.
- Hillis used a diploid representation. Each 8-bit gene coded one pair, but since the representation was diploid, two values (pairs) for each gene were stored. Draw on the blackboard.

More applications – parallel sort – Hillis' experiment

- If the alleles for the gene in both chromosomes were equal ("*homozygous*"), then it meant the (one) pair coded in the gene must be added to the network.
- If the alleles for the gene in both chromosomes were different ("*heterozygous*"), then it mean **both** (!) pairs must be added to the network (so no *dominance*).
- The length of the chromosome was 60 genes.
- That way networks for *k* between 60 (all homozygous) and 120 (all heterozygous) were tested, Hillis hoped that the network will evolve to the *k* = 60 best-known network by gradually becoming all-homozygous.

More applications – parallel sort – Hillis' experiment

- Hillis cheated (or “*used the knowledge about the problem domain to help the GA*”) by starting from a population where the initial comparisons were encoded “right” – he noticed that almost all $k = 60$ sorting networks start from the same pattern of 32 comparisons.
- Note that a great majority of individuals will not represent correct networks – that is, they will describe networks that do not correctly sort numbers. Awarding all of them a fitness of 0 would be a mistake, it is better to give “partial credit” for getting part of the job done.
- Thus Hillis tested the networks on a set of input test-cases and awarded partial credit for the % of cases sorted correctly.

More applications – parallel sort – Hillis' experiment

- Notice how the fitness function depends only on "how correct the network is" and not on how short it is. Hillis claimed that shorter networks will be favoured implicitly, because a good comparison encoded in a heterozygous gene can be lost during crossover, whereas a homozygous gene will not lose it (he used more complicated crossover than we have used, since it was a diploid model).
- Also, Hillis placed his individuals on a 2D lattice, trying to improve *speciation* (differentiation) in the population – he hoped that groups of individuals living in different parts of his "world" would have genetic similarities "in the family", but not across families. In this way he attempted to escape the scenario in which all (almost all) individuals represent very similar networks.
- Mating occured only within a "neighbourhood".

More applications – parallel sort – Hillis' experiment

- Hillis had access to a large (at the time) parallel machine, so he could afford using large populations (he tried between 2^9=512 and 2^{20}=1048576 individuals). He ran the experiment for 5000 generations.
- His GA discovered the *k*=65 sorting network, but could not find the better ones (remember, best known has *k*=60).
- He was disappointed with the result and looked for reasons that could have led to GA getting stuck in one of the local optima.
- One convincing reason was that the **test-cases were not difficult enough** – once the GA found a network that worked on all test-cases, the selection pressure dropped radically. Nature thought “It works, so why improve it?”.

More applications – parallel sort – Hillis' experiment

- So Hillis took another hint from biology and looked at biological “arms races”, such as host-parasite interactions.
- When a parasite is successful in exploiting weaknesses in host organisms, the host tries to evolve a defence to the parasite. As soon as it evolves, the parasite tries to evolve a way to go around the new defence, a better way to enter the host and this “arms race spiral” continues.
- Using an analogy, Hills made the test cases live on the same 2D grid that his networks lived on, and made them evolve to become more difficult as the networks got better. So the test cases got harder and harder and tried to exploit any weaknesses in the sorting networks.
- That forced the population to keep changing.

More applications – parallel sort – Hillis' experiment

- The result was the discovery of a k = 61 sorting network...
- Thus, a major improvement when compared to the first model, yet a disappointing one comparison from the best human result. So “*close, but no cigar*.”

More applications – study of evolution, the Baldwin Effect

- More applications of GA to study biological processes.
- Two well-known adaptive processes in biology:
  - evolution – adapting life *per se* on Earth in the long-term, across 10^9 years,
  - individual learning – adapting the behaviour of every individual during the course of its life, across 10^0-10^2 years.
- Do these two processes interact? If so, how?

More applications – study of evolution, the Baldwin Effect

- The Lamarckian hypothesis (1801).
- Lamarck postulated that traits acquired during the lifetime of an organism can be transmitted to the organism's offspring.
- E.g. shortening a dog's tail would cause its offspring to have shorter tails.
- Or a giraffe reaches for leaves in high trees and, consequently, its offspring has longer necks.

A rendition of Lamarck, image taken from Wikipedia

More applications – study of evolution, the Baldwin Effect

- Notice that Lamarck had this idea before Darwin "discovered" evolution and long before we understood the mechanisms of genetics.
- Today biologists agree that the Lamarckian hypothesis cannot be true – there is no conceivable way in which the fact that we have shortened a dog's tail would influence its genetic material and we firmly believe that the genetic code is the only way in which information can be transferred from a parent to a newborn.
- We can generalize a bit and instead of physical traits (like "short tail") think about **things learned during the life of an individual**.

More applications – study of evolution, the Baldwin Effect

- We can generalize a bit and instead of physical traits (like “short tail”) think about **things learned during the life of an individual**.
- For example one capybara discovers that a certain plant is poisonous (after eating it, it gets very sick, but it survives). If the capybara can learn, it will never touch that plant again.
- The question is – will the offspring of the capybara know?

More applications – study of evolution, the Baldwin Effect

- The question is – will the offspring of the capybara know?
- Lamarck would say yes, but we disagree with Lamarck.
- **Yet, maybe there is a way in which what we learn can influence evolution** (i.e. we don't talk about the capybara warning its children about the plant "in language", but maybe, in some way, the information that the plant is poisonous can be transferred genetically)?

More applications – study of evolution, the Baldwin Effect

- Baldwin (1896) proposed a less-direct mechanism for this (a non-Lamarckian mechanism in which individual learning could have an effect on evolution). He claimed:
- If learning helps survival, then the individual with good learning skills will have a larger chance of having children.
- Thus, the gene responsible for learning will have a larger chance of reproduction.
- Thus, the number of copies of this gene will increase (in GA lingo: “gene for learning is a good schema”).
- Thus, the individuals that *learn* have a larger chance of developing the gene that encodes the trait that originally had to be learned.
- So the intelligent capybara will have more offspring, which will have more offspring, ..., allowing evolution more tries to find the gene that codes “*don't eat this plant*” and further capybaras will not have to *learn* this, they will *know*.

More applications – study of evolution, the Baldwin Effect

- Having the right behaviour encoded genetically gives an organism a direct advantage over the other organisms, because *learning* is less reliable than *knowing* (genetically) (e.g. once in a while the capybara could die after eating the poisonous plant).
- What is more, genetic information is available right from the point of birth, whereas learning takes time.
- In summary: learning increases the chance of survival, giving nature more chances to independently discover the same trait *genetically*.

More applications – study of evolution, the Baldwin Effect

- Opponents of Baldwin argue that it is not clear how **the correlation** between what is learned and what has to be discovered genetically **takes place**.
- The randomness of genetic variation is the central principle of modern evolutionary theory, so we would expect the “lucky intelligent capybaras” to have the same chance of evolving a different eye colour, better kind of fur as evolving a phobia of the poisonous plant.
- Just because there are more of them, doesn't imply that they will learn what we think they should learn.
- We cannot claim that genetic variation can be directed by individual (phenotypic) trait – this is a disguised version of Lamarckian hypothesis.

More applications – study of evolution, the Baldwin Effect

- More complicated models that allow for the Baldwin effect (i.e. learning influencing evolution) were formulated (e.g. Waddington, 1942), but biologists still cannot agree.
- A perfect spot for a Genetic Algorithm to model the behaviour and to see what happens!
- In 1987 Hinton and Nowlan employed a GA to try to demonstrate the Baldwin effect empirically and to measure its magnitude.
- Let us study their experiment.

More applications – study of evolution, the Baldwin Effect

- Each individual in the model is described by 20 binary values describing specific traits. We can hold a simplified view that this is a set of 20 rules for “eat this, don't eat that” (20 commandments?).
- It is assumed that there is only one set of “correct” values. **The fitness of an individual that got even one value wrong is zero**.
- Note how this is similar to Weinberg's reaction constants for *E. coli*.
- Thus our search space includes $2^{20}=1048576$ possible configurations, of which only one (!) is correct (“a needle in a haystack problem”).

More applications – study of evolution, the Baldwin Effect

- How do we include *learning* in the picture?
- Hinton and Nowlan allow a value in an organism to have one of three values “1”, “0” and “?” (meaning “learnable”).
- They use the simplest way to learn – random guessing. Thus on each learning trial, an individual tries to guess the right value for every “?” in its genetic make-up.

10011011101111001101 – (the genotype for) an individual that “genetically knows” everything (if got it right – maximum fitness, if got it wrong, fitness =0).

100110??1011??00?101 – (the genotype for) an individual that “genetically knows” 15 of 20 traits, but can (try to) learn the remaining 5.

???????????????????? – (the genotype for) an individual that doesn't “genetically know” anything, but can (try to) learn it.

More applications – study of evolution, the Baldwin Effect

- In each generation Hinton and Nowlan gave every individual maximum 1000 chances to guess the correct values for “?”. Let us denote by *n* the number of learning trials that remained out of the maximum 1000 given to an individual to guess (learn) the correct solution.
- The fitness of an individual was assumed to be:

$$f = 1 + \frac{19n}{1000}.$$

- Thus, an individual that already had the correct genetic make-up (did not need to guess) had a fitness of 1+19000/1000 = 20.
- An individual that never learned the right solution got a fitness of 1.

More applications – study of evolution, the Baldwin Effect

- Notice, how the addition of *learning* changes this from a "needle in a haystack problem" to a "climb a hill by a combination of learning and evolution" problem).
- We now have a trade-off between efficiency ("knowing genetically") and plasticity ("learning" or "having a lot of '?'").
- If you have a lot of "?", you can survive, but the more "?" you have, the more difficult it is to find the right solution.
- If you have few "?", you "know a lot" genetically, but if you "know it wrong", you have a fitness of zero.
- Of course, to avoid Lamarck's perspective, the learned information was NOT passed genetically to the next generation.

More applications – study of evolution, the Baldwin Effect

- Hinton and Nowlan managed to show that, without learning, the average fitness did not improve with time.
- Yet, with learning (even though the learned information was not passed genetically), evolution quickly eliminated the wrong alleles [4, Fig. 3.4].
- That way, they proved that learning can guide evolution, even if the learned information is not passed genetically. Even with this simple form of learning (guessing) evolution was able to discover individuals with all values set correctly.
- Thus, they have confirmed the Baldwin effect.

More applications – study of evolution, the Baldwin Effect

- Notice, however, that the although the wrong alleles disappeared completely, the frequency of “?” (undecided) values did not drop below ca. 40%.
- So the wrong behaviours were eliminated by evolution, but there was not enough selection pressure to make the individuals *know* all the answers genetically.
- Increasing the length of the experiment 10-times allowed evolution to reduce the number of undecided alleles to 30%, but the convergence was very slow.
- Thus, it was confirmed that learning **can** guide evolution, at least on a very simple model.
- Yet, it only forced evolution to eliminate the wrong notions, did not eliminate the possibility of learning.

Genetic algorithms – key points

- **Powerful tool in search for optimimum values of parameters (optimization). Works well where traditional methods fail (difficult functions, many-dimensional problems).**
- Tool to study models of evolution.
- Tool to evolve programs (evolware).
- Robust technique, usually the basic algorithm works surprisingly well.
- Only the basics of the underlying maths are understood, still an object of research.
- Several techniques exist to improve the capabilities, they are usually needed in specific cases (e.g. dynamically changing fitness landscape).

Genetic algorithms – key points

- Powerful tool in search for optimimum values of parameters (optimization). Works well where traditional methods fail (difficult functions, many-dimensional problems).
- **Tool to study models of evolution.**
- Tool to evolve programs (evolware).
- Robust technique, usually the basic algorithm works surprisingly well.
- Only the basics of the underlying maths are understood, still an object of research.
- Several techniques exist to improve the capabilities, they are usually needed in specific cases (e.g. dynamically changing fitness landscape).

Genetic algorithms – key points

- Powerful tool in search for optimimum values of parameters (optimization). Works well where traditional methods fail (difficult functions, many-dimensional problems).
- Tool to study models of evolution.
- **Tool to evolve programs (evolware).**
- Robust technique, usually the basic algorithm works surprisingly well.
- Only the basics of the underlying maths are understood, still an object of research.
- Several techniques exist to improve the capabilities, they are usually needed in specific cases (e.g. dynamically changing fitness landscape).

References

- [1] Beasley D., Bull D., Martin R. "*An overview of Genetic Algorithms: Part 1, Fundamentals*", 1993.
- [2] Beasley D., Bull D., Martin R. "*An overview of Genetic Algorithms: Part 2, Research Topics*", 1993.
- [3] Goldberg DE, "*Genetic Algorithms in Search, Optimization & Machine Learning*", 2006.
- [4] Mitchell M, "*An Introduction to Genetic Algorithms*", 1998.

- [1] and [2] are freely downloadable from the Net – either google for them or use the external links below the Wikipedia article "Genetic Algorithm".