POLITECNICO
MILANO 1863

AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

Genetic Algorithms Introduction

Andrea Bonarini
andrea.bonarini@polimi.it

Artificial Intelligence and Robotics Lab
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

What are Genetic Algorithms?

Algorithms to learn (sub-) optimal models

inspired by the biological genetic model

Each model can be considered as the solution of a problem

Looking for solutions of a problem

Looking for solutions of a problem is different from looking for data in a database: the potential solutions might be so many that it might not be possible even to list all of them.

We would like to generate candidate solutions to be evaluated, by applying some criteria to reduce this generation to the ones that could be good enough to be considered.

Many algorithms to search for solutions exist, each applicable in particular conditions, and showing a different performance on different problems (hill climbing, A*, tabu search, simulated annealing, ...)

Search space

We say that we search a solution in a **search space**.

The quality of the solution is called "fitness" in analogy with the biological terminology.

Genetic algorithms are an effective tool to search in very large deterministic search spaces (e.g., 10^{70}), also when these show irregularities.

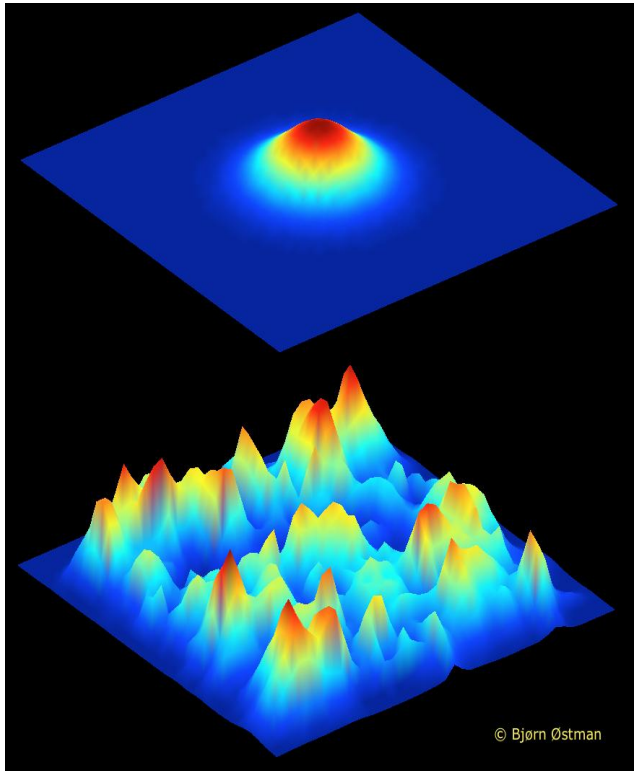
GA recombine parts of solutions to generate better ones. There is no guarantee to get to the optimum, but it is proved that the solutions are improving with the iterations of the algorithm.

Fitness Landscape

The fitness landscape is a spatial representation of the quality of the solutions

If we have two values to represent the solution, we can visualize the fitness landscape in a three-dimensional space, where we could see hills and valleys.

Genetic algorithms are quite good also when the fitness landscape has many local minima and maxima.



Short history

Genetic Algorithms are a part of evolutionary computation, and they are inspired by Darwin's theory of evolution: problems are solved by an evolutionary process that mimics natural evolution in looking for a best (fittest) solution (survivor).

We can trace a brief history of evolutionary computation:

1960: Ingo Rechenberg introduces the idea of evolutionary computing in his work "Evolution strategies"

1975: John Holland proposes Genetic Algorithms and publishes his book "Adaption in Natural and Artificial Systems"

1992: John Koza uses genetic algorithms to evolve programs to perform certain tasks. He called his method Genetic Programming

1995: Stewart Wilson re-defines learning classifier systems with XCS: GA to learn rules...

Applications of Genetic Algorithms

Optimization: circuit layout, job shop scheduling...

Automatic programming: evolving computer programs, inventions, …

Classification: classifying etities from theri features

Prediction: whether forecast, proteins ...

Economy: bidding strategies, market evolution, ...

Ecology: biological arm races, host-parasite coevolution, ...

...

Terminology

In biology

Chromosome: information that characterizes an individual

Gene: elementary information contained in chromosomes (e.g.: eye color); each gene has a specific position in the chromosome

Allele: value for a gene (e.g.: brown, blue, ...)

Genome: the complete collection of the genetic material (all chromosomes)

Terminology (2)

Genotype: the specific set of genes contained in the genome; two individuals with the same genome have the same genotype

Fenotype: specific set of genes for an individual (physical features)

Diploid: individual with paired chromosomes

Haploid: individual with non-paired chromosomes

Terminology (3)

Crossover: in sexual reproduction, paired chromosomes exchange genetic material to generate a gamete that combines with that of the other parent to generate the diploids of the offspring. In the haploid reproduction, the chromosomes of the parents mix with each other to obtain the offspring chromosome

Mutation: a gene changes during reproduction

Fitness: probability of life and of reproduction, or, also, function of the generated offspring

From biology to genetic algorithms

Chromosome: candidate solution for a problem, usually represented by a bit string or a character string

Gene: single bit, or set of bits, that characterize a solution

Allele: in a bit string is either 0 or 1; in general: an element of the alphabet used to represent a chromosome

In general, reproduction is haploid

How does a genetic algorithm work?

1. [Start] Generate a random population of n chromosomes (suitable solutions)
2. [Fitness] Evaluate the fitness f(x) of each chromosome x in the population
3. [Test] If the end condition is satisfied, return the best solution in current population, otherwise
4. [New population] Create a new population by repeating the following steps until the new population is complete
    a) [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
    b) [Crossover] With a crossover probability cross over the parents to form new offspring. If no crossover was performed, offspring is the exact copy of parents.
    c) [Mutation] With a mutation probability mutate new offsprings at each locus
    d) [Accepting] Place new offsprings in the new population
5. [Replace] Use the new generated population for a further run of the algorithm
6. [Loop] Go to step 2

Designing genetic algorithm applications

There are many parameters and settings that can be implemented differently to face various problems:

- How to define a fitness function
- How to create chromosomes and what type of encoding do we have to choose
- How to select parents for crossover in the hope that the better parents will produce better offspring
- How to define crossover and mutation, the two basic operators of GA

Fitness function

.

f(x) is used to evaluate the quality (fitness) of an individual (solution). It *defines* the solution to be found and *drives* the search.

The evaluation is used to sort the individuals, fundamental operation for genetic evolution

Data model

Sets (populations) of chromosomes, each representing a candidate solution for the proposed problem.

The first step in developing a genetic algorithm is defining how to encode a solution:

- A chromosome should contain information about the solution that it represents
- The encoding depends mainly on the problem to be solved (e.g., integer or real numbers, strings, permutation, parsing trees, . . .)

E.g.:

0 1 1 0 1 1 0 0 1 1

Low High Medium Low Low

1.234 67.345 899.00 78.786

Binary encoding

Binary encoding is the most common one, mainly because, when all configurations are used, it guarantees the maximum exploitation of the information representation.

- In binary encoding, every chromosome is a string of bits (0 or 1)
- Simple implementation of the genetic operators
- Not natural for many problems

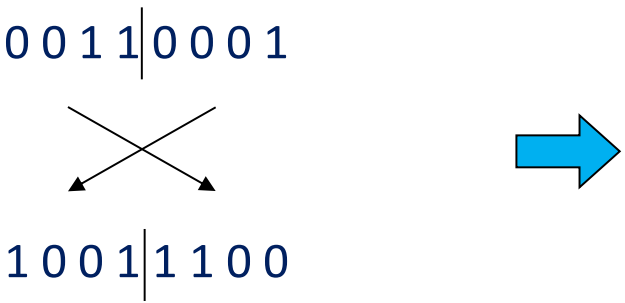
E.g.:

0 1 1 0 1 1 0 0 1 1

Binary encoding: operators

For binary encoding we have many operators.

Single point crossover: one crossover point is selected, the binary string from the beginning of the chromosome to the crossover point is copied from the first parent, the rest is copied from the other parent



Binary encoding: operators

- **Two point crossover**: two crossover points are selected, binary string from the beginning of the chromosome to the first crossover point is copied from the first parent, the part from the first to the second crossover point is copied from the other parent and the rest is copied from the first parent again
- **Uniform crossover**: bits are randomly copied from the first or from the second parent
- **Arithmetic** crossover: some arithmetic operation is performed to make a new offspring (e.g., logic AND)

- **Mutation**: inversion of bits selected with a given probability

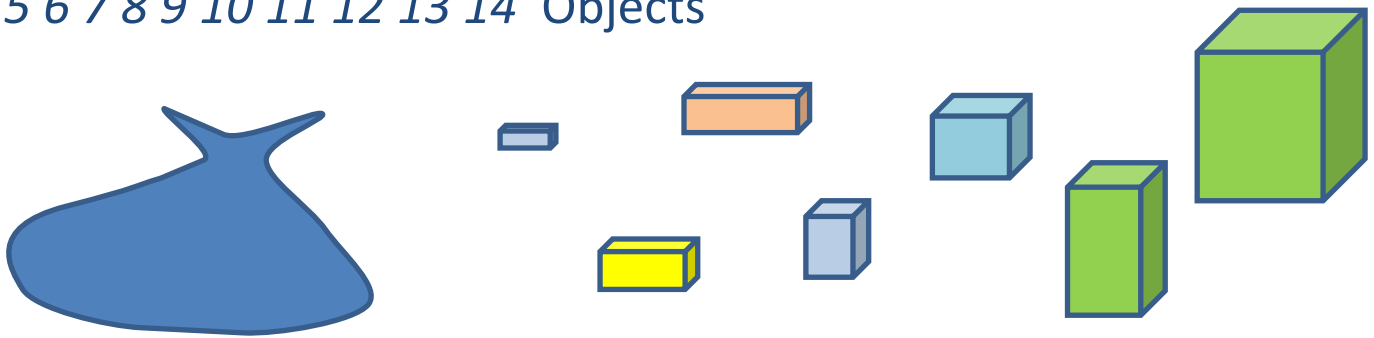
Binary encoding example

Knapsack problem

- There are things with given value and size. The knapsack has given capacity. Select things to maximize the value of what is in the knapsack, but do not exceed the knapsack capacity.
- Each bit says whether the corresponding thing is in the knapsack.

1 0 1 1 1 1 0 0 0 0 1 1 0 0 Chromosome

1 2 3 4 5 6 7 8 9 10 11 12 13 14 Objects



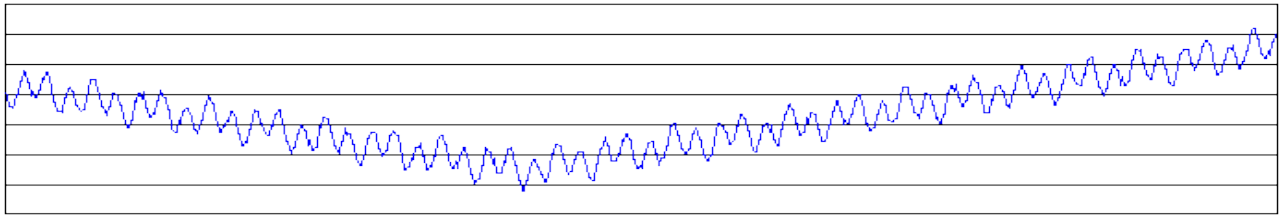
Another example

In this simple example we are looking for the extreme of a function defined over a search space.

1. Search Space: An interval of the real line
2. Fitness Function: The value of the function we are “exploring”

Why should we use genetic algorithms for this?

Because functions may get quite nasty ;-)



An example

Our chromosome may look like these:

Cromosome 1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0

Cromosome 2 1 0 1 1 1 1 0 0 0 1 1 1 1 0 0 1

Each chromosome is represented as the binary code of a real number

We apply single point crossover and standard flipping value mutation

Another nice demo to play with is on
http://math.hws.edu/eck/jsdemo/jsGeneticAlgorithm.html

Permutation encoding

Permutation encoding can be used in **ordering** problems

- Every chromosome is a string of numbers each representing a position in a sequence that could be modified to optimize some fitness
- Crossover and mutation must be designed to leave the chromosome consistent (i.e., representing a sequence with all the original elements)

E.g.:

- Cromosome 1 15 7 8 3 5 13 10 11 16 12 1 14 2 4 6 9
- Cromosome 2 2 9 14 1 11 5 8 15 13 6 12 16 7 3 10 4

Permutation encoding operators

For permutation encoding we have to preserve consistency of the solution, i.e. all the elements should appear only once, and be all present in the offspring

One point ordered crossover: one crossover point is selected, the permutation is copied from the first parent till the crossover point, then the other parent is scanned looking the other numbers

(1 2 3 4 5 6 7 8 9) x (4 5 3 6 8 9 7 2 1) ⇒ (1 2 3 4 5 6 8 9 7)

Section ordered crossover: a section of one parent is preserved and the oder places are taken in order from the second parent

Order changing mutation: two numbers are selected and exchanged

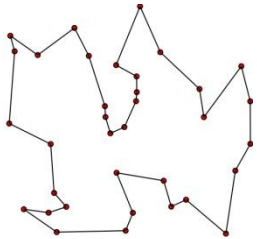
(1 2 3 4 5 6 8 9 7) ⇒ (1 8 3 4 5 6 2 9 7)

Permutation encoding example

Traveling salesman problem (TSP)

- There are cities and given distances between them. Traveling salesman has to visit all of them, but he does not want to travel more than necessary. Find a sequence of cities with a minimal travelled distance
- The chromosome describes the order of cities

An example: https://www.youtube.com/watch?v=94p5NUogClM



Direct value encoding

Direct value encoding can be used in problems where some more complicated values are required

Every chromosome is a sequence of some values connected to the problem, such as (real) numbers, chars or any objects

Good choice for some special problems, but necessary to develop some specific crossover and mutation

A B H Y V V

2.5678 1.4361 3.3426 7.8761

close open walk back

Direct value encoding: operators

For real value encoding we can use the same crossover used for binary encoding

Creep mutation: a small number is added (or subtracted) to selected values

(1.29 5.68 2.86 4.11 5.55) ⇒ (1.29 5.68 2.73 4.22 5.55)

Direct encoding example

Finding weights for a neural network

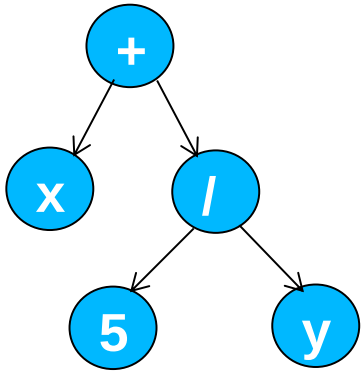
- A neural network is given with defined architecture. Find weights between neurons to get the desired output from the network
- Real values in chromosomes represent weights in the neural network
- It is a different way of finding the optimal weights w.r.t. the standard NN learning

Tree encoding

Tree encoding is used mainly for evolving programs or expressions (i.e., **genetic programming**)

- Every chromosome is a tree of some objects, such as functions or commands in programming language.
- Programming language LISP is often used for this purpose, given its simple syntax, so crossover and mutation can be done relatively easily.

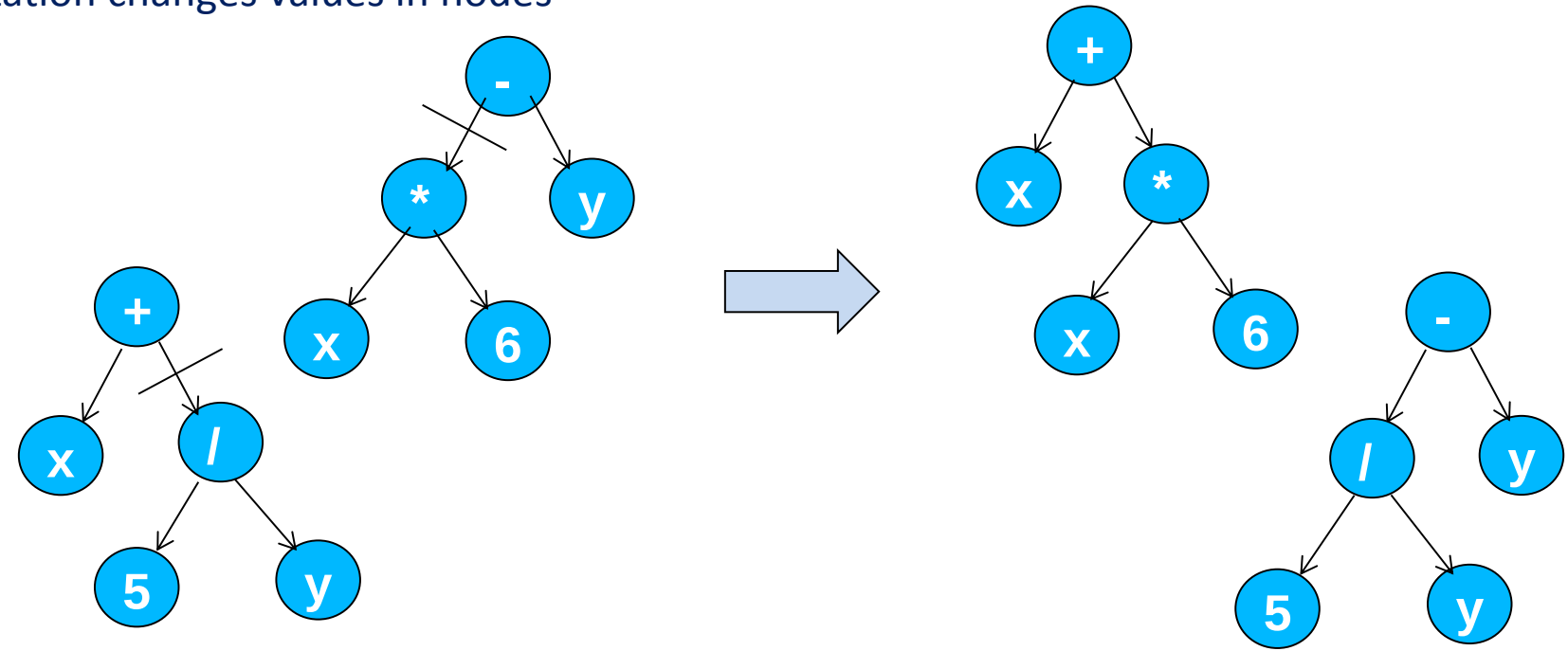
E.g., Cromosome (+ x (/ 5 y)), i.e. x+5/y



Tree encoding: operators

Crossover cuts a link and exchanges material.

Mutation changes values in nodes



https://kahoot.com

Tree encoding example

Finding a function that would best match given pairs of values (approximant function)

Input and output values are given. The task is to find a function that will give the best outputs for all inputs.

Chromosome are functions represented in a tree

Selection of the parents

- According to Darwin's theory of evolution the best chromosomes survive to create new offspring.
- The best individuals should be selected to contribute to the new population so that it improves. They are the ones that mate to generate hopefully better offsprings.
- Genetic operators are applied to pairs of good individuals to generate offsprings to be inserted in the population.

Selection methods

There are many methods to select the best chromosomes:

- Roulette wheel selection,
- Rank selection
- Tournament selection
- Boltzmann selection
- . . .

Roulette wheel selection

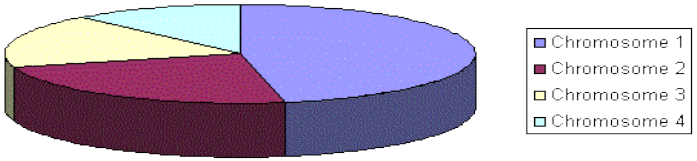
Parents are selected proportionally to their fitness.

The better they are, the more chances to be selected they have.

Imagine a roulette wheel where all the chromosomes in the population are placed

The size of the section in the roulette wheel is proportional to the value of the fitness function of every chromosome - the bigger the value , the larger the section

A marble is thrown in the roulette wheel and the chromosome where it stops is selected

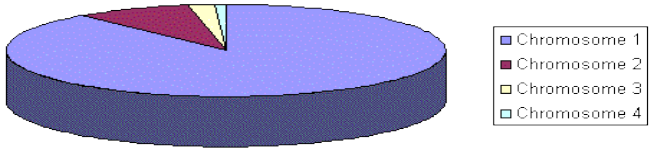


Problems may arise if there is a large difference of fitness from the best to the worst, which could hardly be selected.

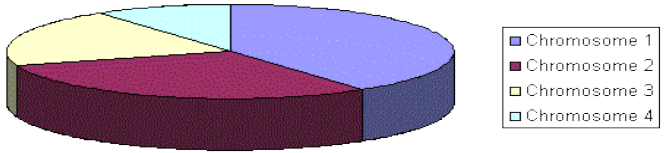
Rank selection

Parents are ranked and the selection probability is proportional to the rank.

Roulette before ranking



Roulette after ranking



Possible problems: slow convergence, due to small difference between best and worst parents.

Tournament selection

Parents are pooled and a tournament is held within the pool(s).

Pseudocode:

1. Choose k (the tournament size) individuals from the population at random
   a) Choose the best individual from pool/tournament with probability p
   b) Choose the second best individual with probability $p * (1 - p)$
   c) Choose the third best individual with probability $p * ((1 - p)^2)$
   d) and so on until all the individuals of the pool has been considered or the size of population is reached
2. If the population size is reached then stop
3. Otherwise go to 1.

Efficient implementation, easy to adjust.

Boltzmann selection

Parents are selected with a probability that favors exploration at the beginning of learning and tends to stabilize and select the best solutions as generations proceed.

Let f_{MAX} be the fitness value of the best individual so far obtained, and $f(X_i)$ be the fitness of the current string X_i. If $X_i > f_{MAX}$ then the current string is taken, otherwise it is taken with a probability

$$p = e^{-(fmax - f(Xi))/T}$$

where

$$T=T_0(1-\alpha)^k$$

and $\alpha \in [0, 1]$, $T_0 \in [5, 100]$, $k = 1 + 100 * g/G$

where g is the generation number and G the maximum number of generations

Reproduction

How are the new individuals generated?

10101110 (0.98)
01000100 (0.95)
11101000 (0.88)
00101011 (0.75)
01010100 (0.56)
00101000 (0.42)

?

Copy operator

When creating a new population, we may have a chance to miss the best chromosome.

Elitism is the name of the method that first copies the best chromosome (or few best chromosomes) to the new population. It can rapidly increase the performance, because it prevents a loss of the so–far best found solution, which might give good contributions.

The copy operator simply copies the individual in the new population.

Why does crossover work?

Hypothesis: an individual is good because it contains good sequences (*building blocks hypothesis*)

There is some probability that crossover composes good solution parts obtaining an individual better than its parents

Crossover point is randomly selected in order to avoid blocking the dimension of the building blocks to be found

Schemata

Holland (1975) formalized the idea of *building block*

A *schema* is a set of strings described by a template made of 0, 1, and *, where * is a wildcard character and stands for any of the other two

E.g.,

H=1****1

represents any 6-bit long string that starts and ends by 1.

The *order* of the schema is the number of defined bits (in this case 2)

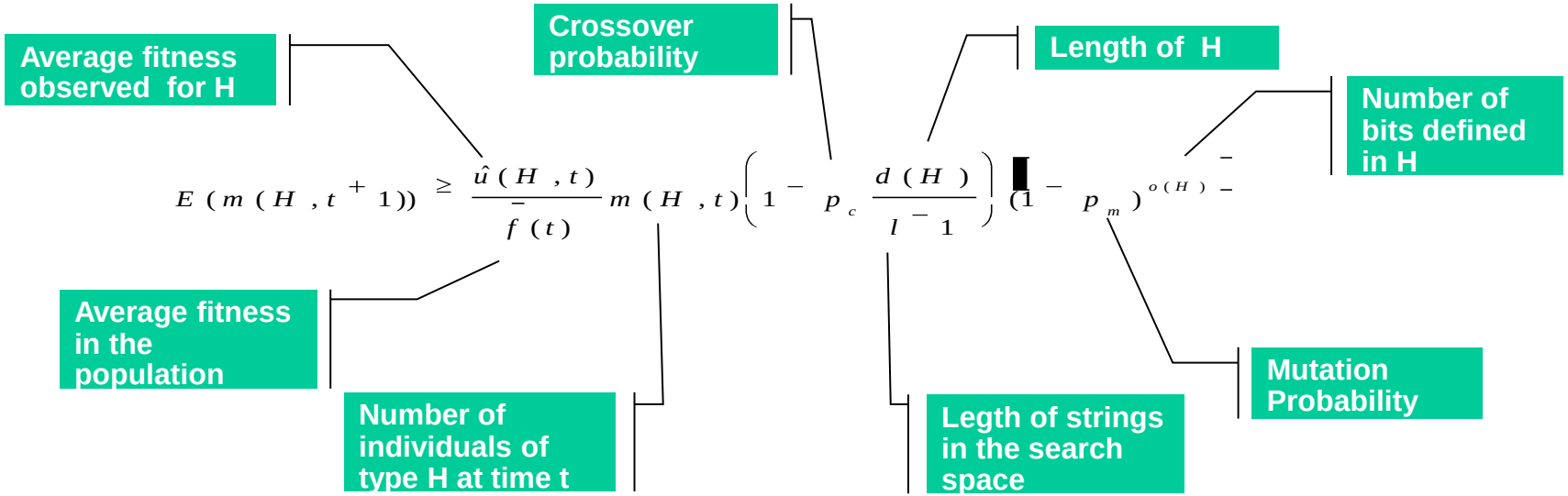
Any bit string of length l is an individual made of 2^l schemata.

Any string evaluation actually evaluates many schemata at the same time.

The more the evaluated strings for a schema, the more the average value of fitness of the schema reliably represents its value

Schema Theorem

The expected number of individuals belonging to the same schema, at the next step is bounded by:



Consequence of the schema theorem

Short schemata, of low order, whose fitness is higher than the average fitness of the population have a number of samples evaluated that grows exponentially, given that they are less damaged by crossover and mutation

Convergence

Given that many schemata are evaluated at the same time, we can say that Genetic Algorithms exploit implicit parallelism

When should we use genetic algorithms?

- Large, non-unimodal, non-smooth deterministic search space
- A global optimum is not strictly required, but a sub-optimal solution found in a short time is acceptable
- The fitness function is noisy

If…

- the search space is small => exhaustive search
- the search space is unimodal => steepest ascent (e.g., gradient)
- the search space is known => heuristics
- The search space is stochastic => Reinforcement Learning

Some general modeling criteria

- Shorter strings are easier to learn
- The chromosome should describe completely the solution
- Binary representation gives the highest possibility to develop building blocks
- Aim at supporting the presence of significant building blocks

Rules of thumb

These “rules of thumb” are often results of empiric studies performed on binary encoding only, but they often work fine . . .

- Crossover rate should be high, generally about 80% − 95% (however some results show that for some problems crossover rate about 60% is the best)
- Mutation rate should be very low. Good rates might be about 0.5% − 1% per allele
- Very big population size usually does not improve performance (in the sense of speed of finding solution). Good population size is about 20 − 30, however sometimes sizes 50 − 100 are reported as the best
- Basic roulette wheel selection can always be used, but sometimes rank or tournament selection can converge faster. Elitism should be used for sure if you do not use other methods to save the best found solution

Some examples

Learning to fight: https://www.youtube.com/watch?v=u2t77mQmJiY

Learning to jump over a ball: https://www.youtube.com/watch?v=Gl3EjiVlz_4

Learning the shape of a car (in 2D): https://www.youtube.com/watch?v=FKbarpAlBkw

https://kahoot.com