ME 557 – Computer Applications to Industrial Problems

Genetic Algorithms

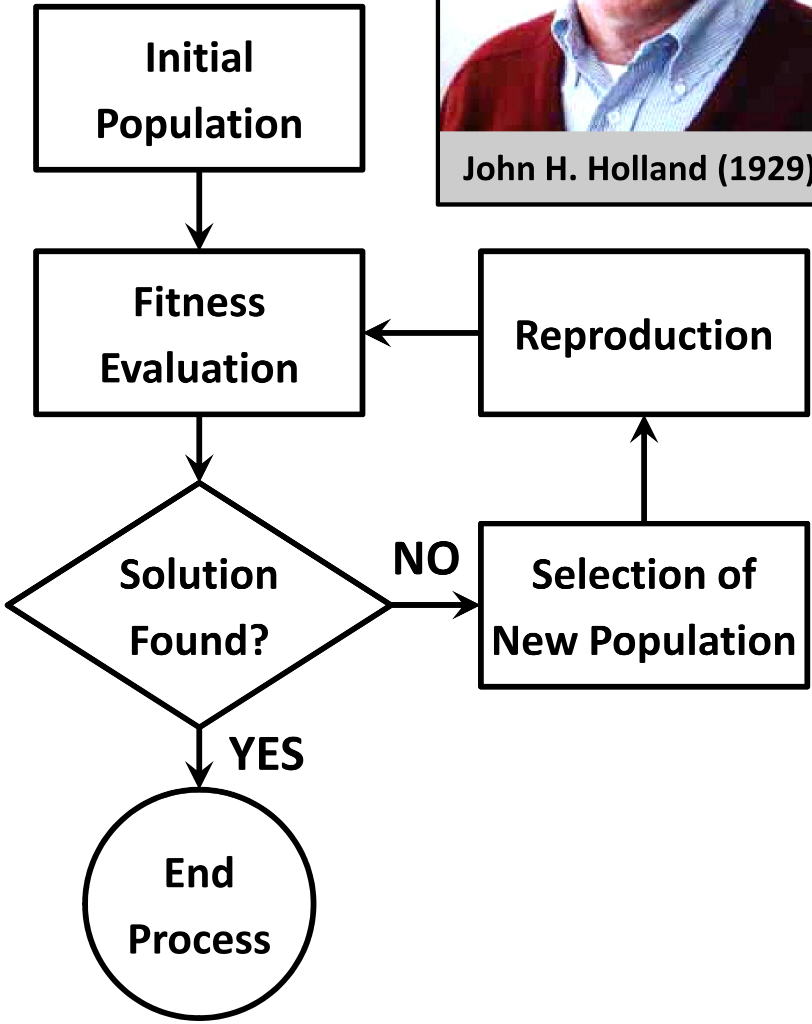
Mechanical Engineering
University of Gaziantep

Dr. Ali Tolga Bozdana
Dr. Sadık Olguner

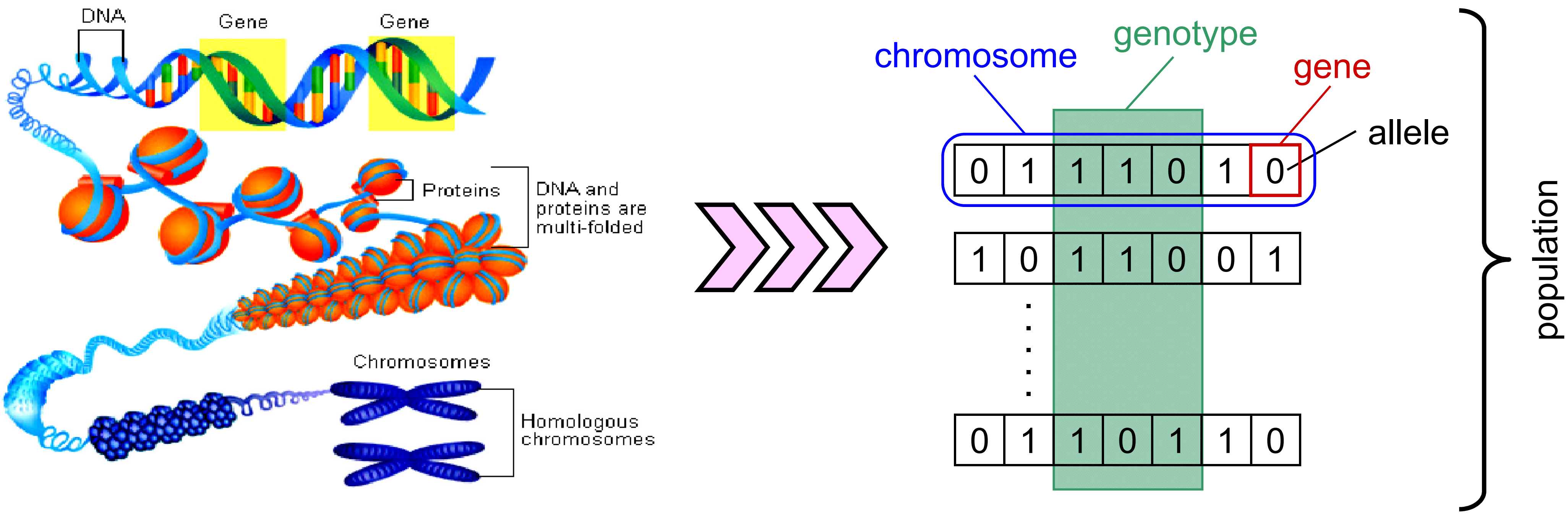
- **Genetic Algorithm (GA)** was initially developed by **John Henry Holland** (American scientist, professor of psychology and professor of electrical engineering & computer science at University of Michigan) in **1970's**.

John H. Holland (1929)

- It is a directed-search algorithm based on mechanics of Darwin's biological evolution theory.
- In other words, it is based on survival of strong ones in the population.
- GAs provide an efficient optimization methodology for highly complex search spaces.
- They are used in the best way when the objective (fitness) function:
  - is discontinuous
  - is highly nonlinear
  - is stochastic (statistically random)
  - has unreliable or undefined derivatives



- Any feasible possible solution: **individual**
- Group of all individuals: **population**
- Blueprint (string) of an individual: **chromosome**
- Possible feature (aspect) of an individual: **gene**
- Possible setting (black, blond, etc.) of a gene: **allele** (1 – dominant, 0 – recessive)
- Collection of all chromosomes for an individual : **genome**
- Particular set of genes in a genome: **genotype**
- Pyhsical characteristics (smart, beautiful, healthy, etc.) of a genotype: **phenotype**



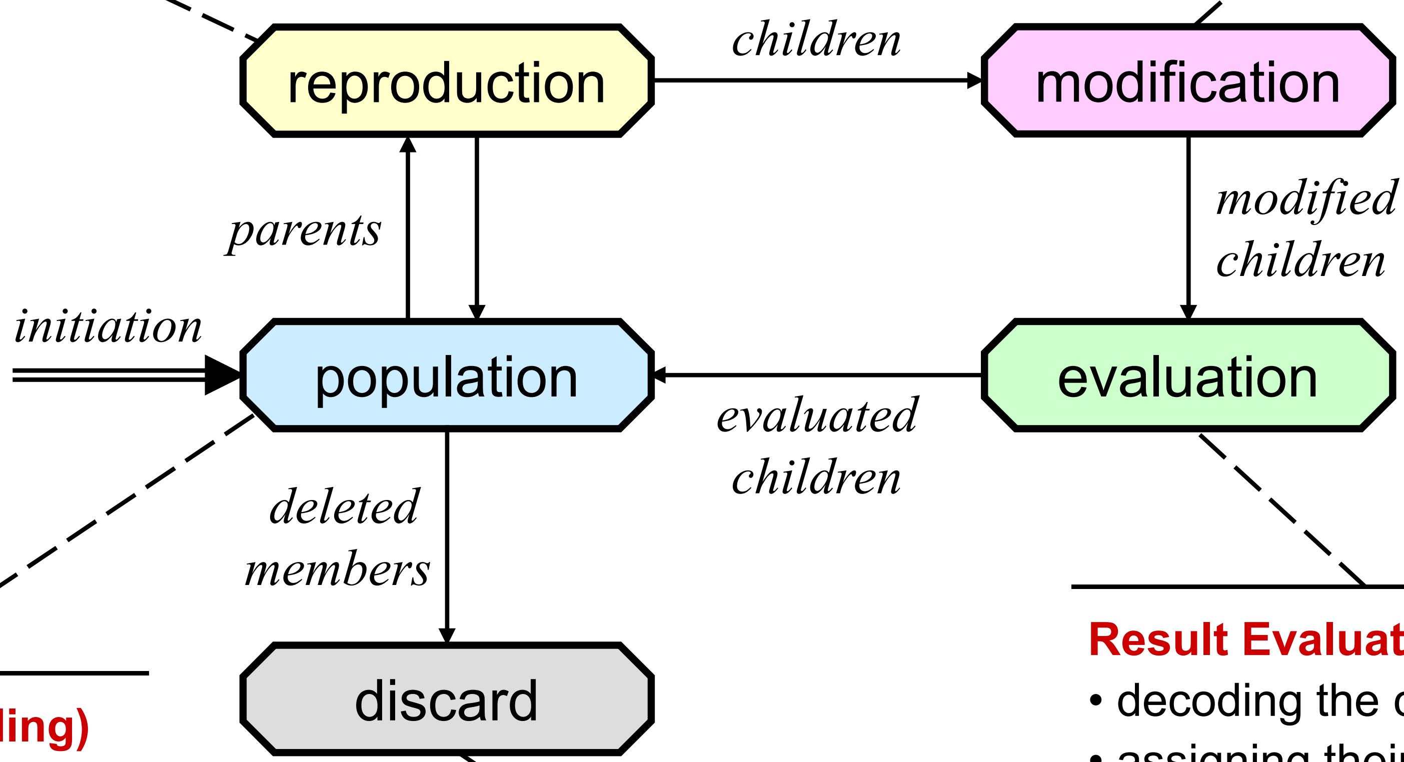
Evolutionary Cycle

Parental Selection

- Random (roulette wheel)
- Best-fitting (tournament)
- Individual fitness (weighted average)
- Rank-based (linear, exponential, ...)

Chromosome Modification

- Crossover (recombination of genes)
- Mutation (replacement of local genes)



Representation (Coding)

- Bit strings (0101, 1100)
- Real numbers (43.2, -33.1)
- Text (“high”, “red”)
- any data structure

Elimination

- discarding bad solutions (having low fitness)

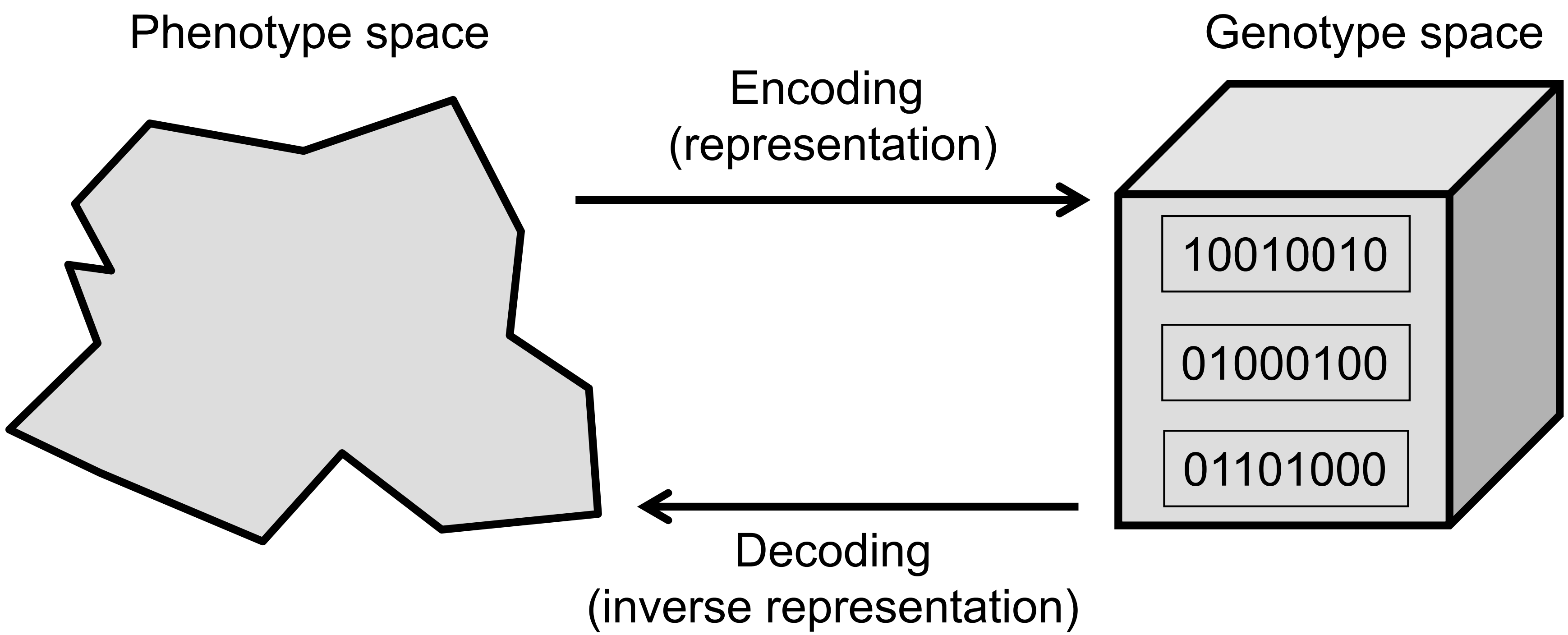
Result Evaluation

- decoding the chromosomes
- assigning their fitness values
- fitness ranking

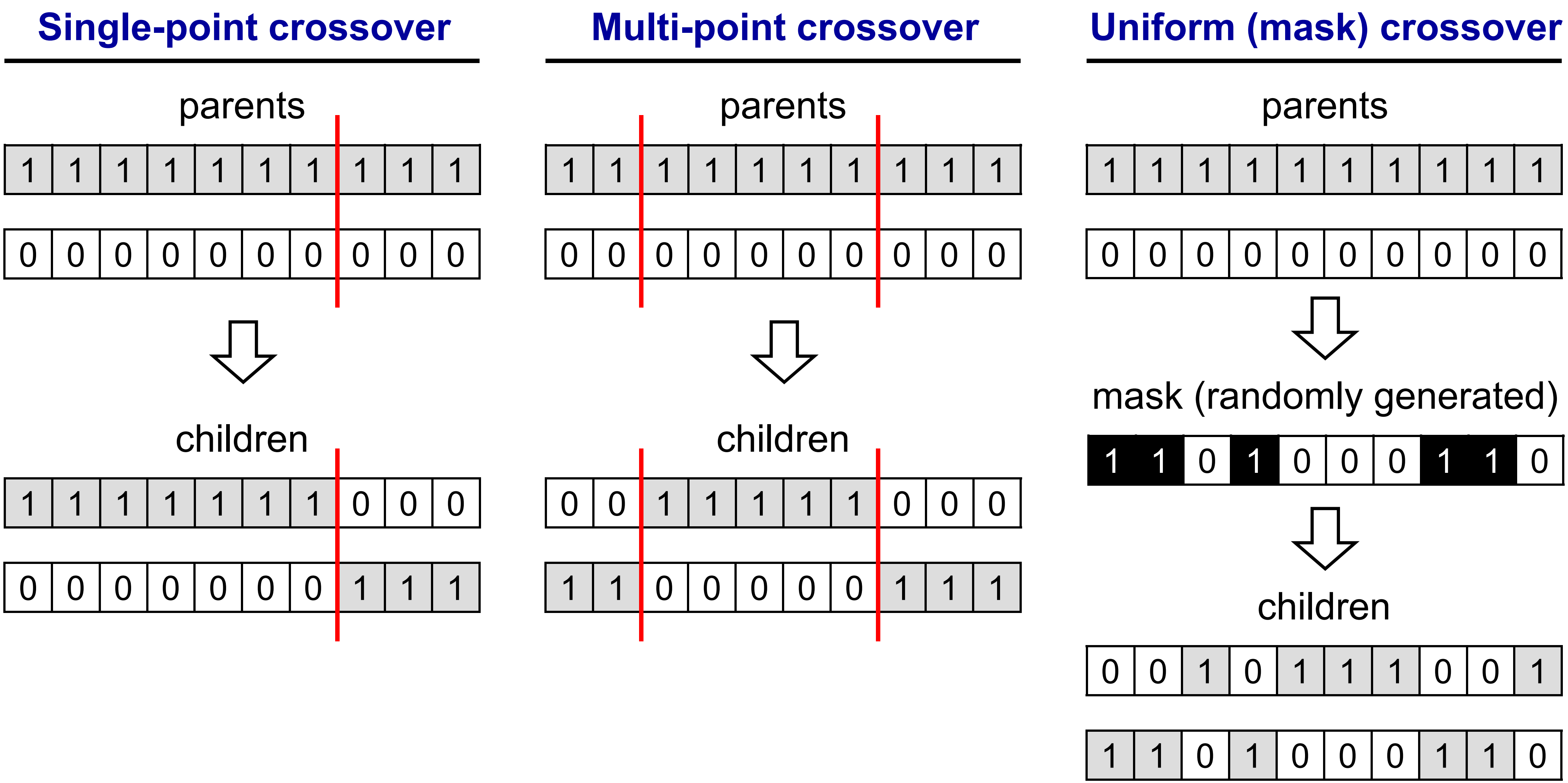
- Parameters of the solution (genes) are bound to form a string (chromosome).
- Good coding is probably the most important factor for the performance of a GA.
- Any alphabet can be used (numbers, characters, etc.), but **binary alphabet** is preferred.
- See the following links for conversion of decimal & binary numbers:

  http://en.wikipedia.org/wiki/Binary_numeral_system

  http://www.mathsisfun.com/numbers/convert-base.php



- Two parents produce two children (offspring).
- Chromosomes of parents are copied unmodified or randomly recombined.
- The chance (probability) of crossover is **generally between 0.6 and 1**.



- Section(s) of a randomly selected chromosome is permitted to mutate (change).
- Mutation probability is in the range of 1/population size & 1/chromosome length.
- Hence, the chance of mutation is **generally quite low (about 0.001)**.

parent

1	1	1	1	1	1	1	1	1	1

child

1	1	0	1	1	1	1	0	1	0

Crossover or Mutation?

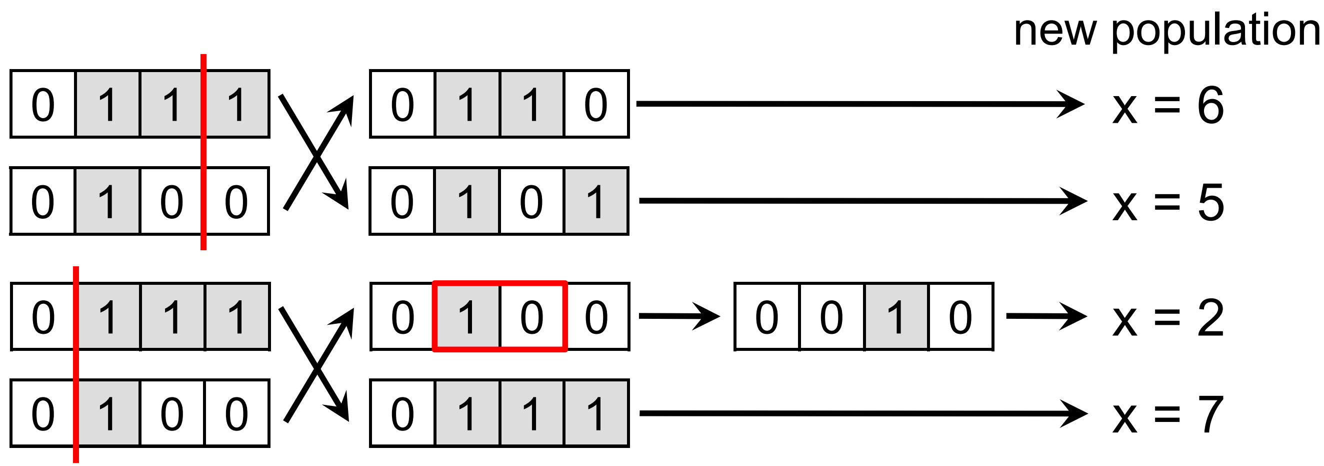
- It depends on the problem. In general, it is good to have both.
- There is **co-operation AND competition** between them:
  - ✓ Crossover is explorative: it makes a big jump to an area somewhere "in between" two (parent) areas.
  - ✓ Mutation is exploitative: it creates random small diversions, thereby staying "near (in the area of)" the parent.
- "Only crossover" can combine information from two parents. On the other hand, "only mutation" can introduce new information (alleles).

- Find the nearest integer value of x for $0 \le x \le 12$ to maximize $y(x) = -0.2(x^2) + 2.5(x)$
- Use population size of 4 having 4-bit chromosomes

1st run	
x	y
4	6.8
1	2.3
7	7.7
10	5.0

pick two x values (giving y_{max}) and convert into 4-bit chromosomes:

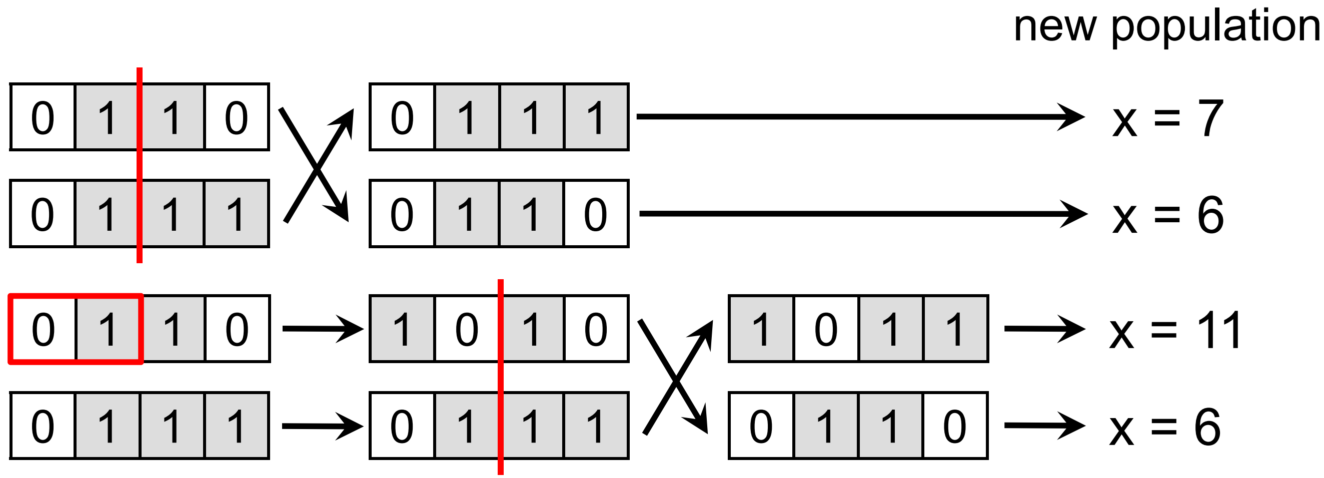
x	4-bit
7	0 1 1 1
4	0 1 0 0



2nd run	
x	y
6	7.8
5	7.5
2	4.2
7	7.7

pick two x values (giving y_{max}) and convert into 4-bit chromosomes:

x	4-bit
6	0 1 1 0
7	0 1 1 1



3rd run					
	x	7	6	11	6
	y	7.7	7.8	3.3	7.8

The optimum solution:
x = 6

Convergence to solution depends upon choice of population size, x values, crossover, mutation, etc.

Pros of GA

☺ Concept is easy to understand.

☺ Modular (i.e. separate from application).

☺ Supports multi-objective optimization.

☺ Good for “noisy” environment.

☺ Always results in an answer, which becomes better and better with time.

☺ Can easily run in parallel.

☺ Fitness function can be changed from iteration to iteration, which allows incorporating new data in the model if it becomes available.

Cons of GA

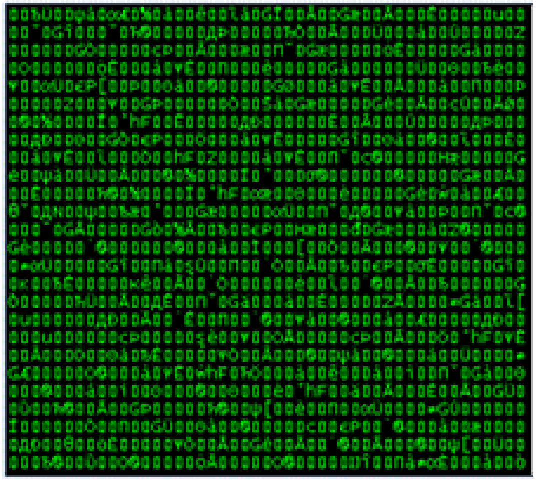
☹ Issues with choosing parameters:

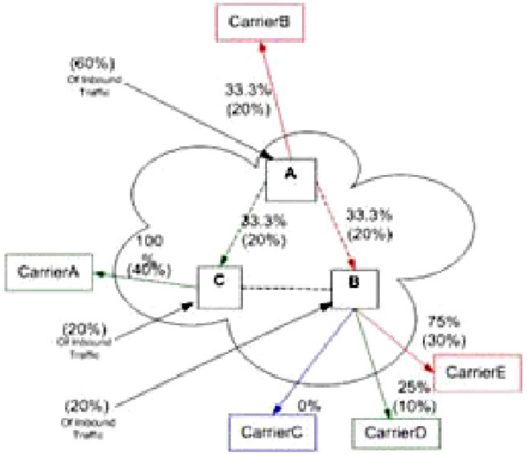
- Population size
- Crossover and mutation probabilities
- Selection

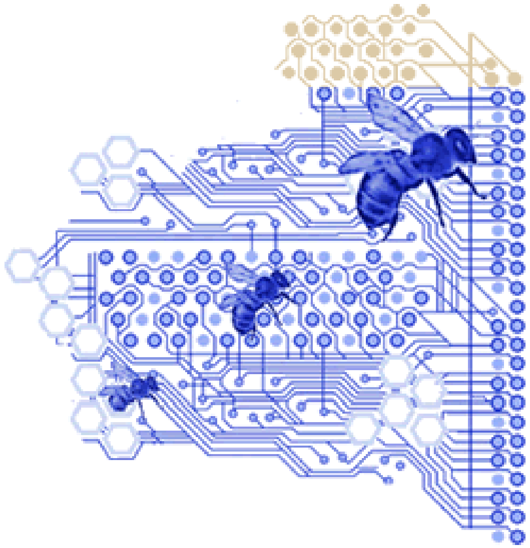
☹ Issues with performance:

- can be too slow (a large search space)
- only as good as the fitness function

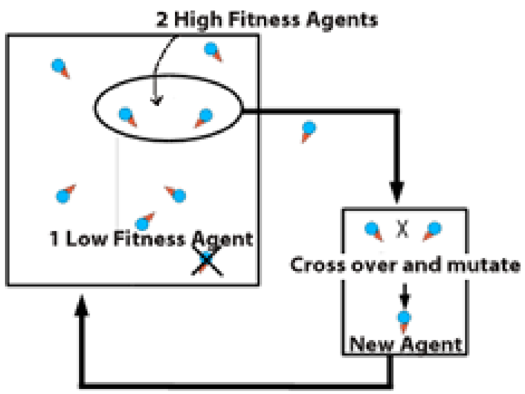
Applications

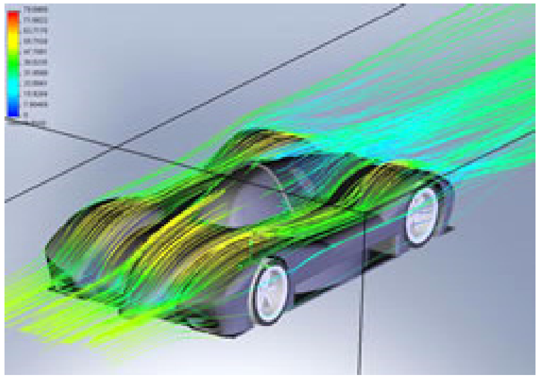


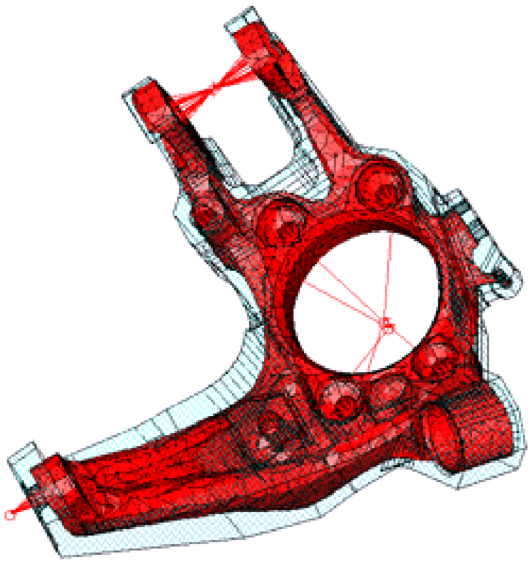


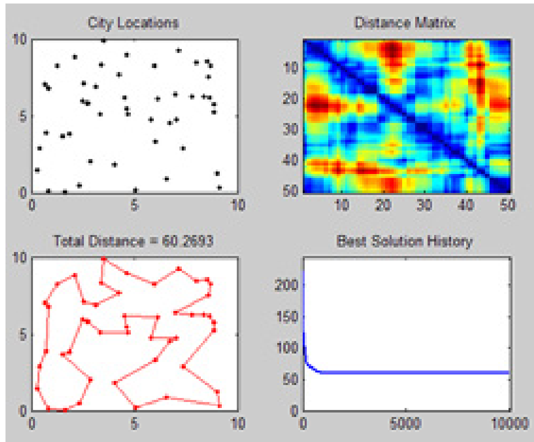


Encryption & Code Breaking | Gaming & Game Theory | Finance & Investment Strategies | Engineering Design | Robotics & Trajectory Planning | Evolvable Hardware | Routing & Scheduling







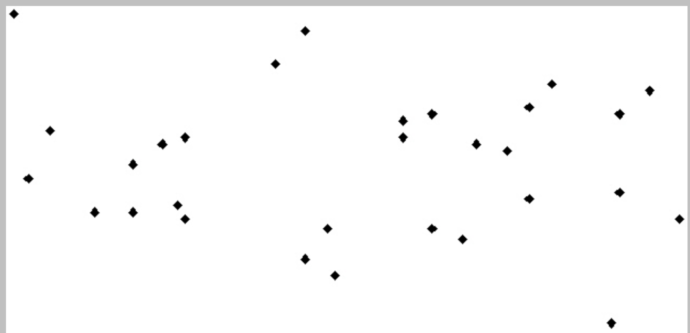


http://brainz.org/15-real-world-applications-genetic-algorithms/

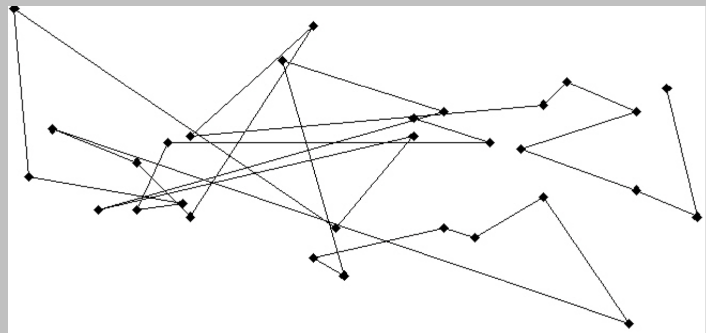
Example: Travelling Sales Person (TSP) Problem

- **Problem:** find a tour of given set of cities (e.g. for 30 cities, there are 30! ≈ 10^{32} possibilities)
- **Objective:** the total distance travelled will be minimized
- **Constraint:** each city must be visited only once
- **Cities:** 1) London 2) Istanbul 3) Beijing 4) New York 5) Rome 6) Barcelona 7) ......
- **Initial population:** (3 5 2 1 6 4 7 ...), (1 5 7 4 6 3 2 ...), (7 6 2 1 3 5 4 ...), ...

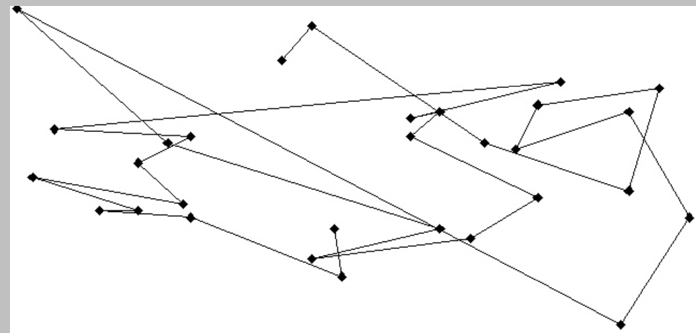
TSP30 (the problem)



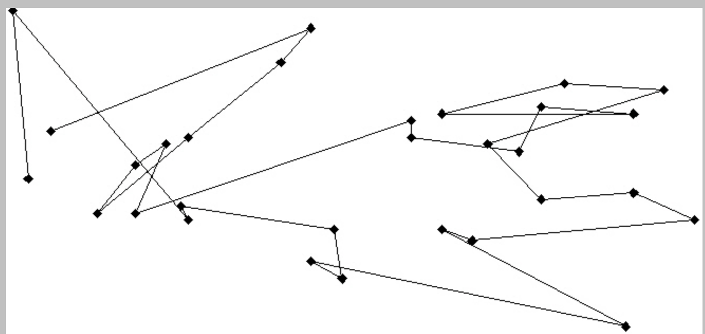
TSP30 (distance = 941)



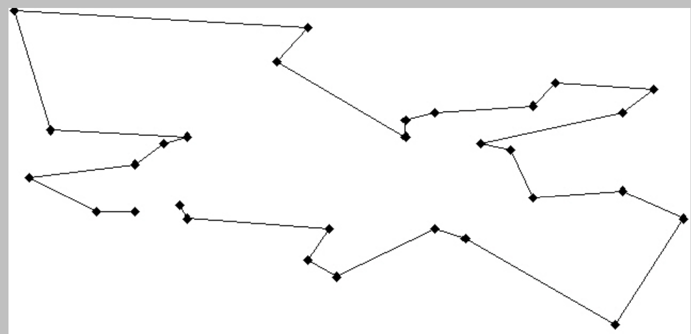
TSP30 (distance = 800)



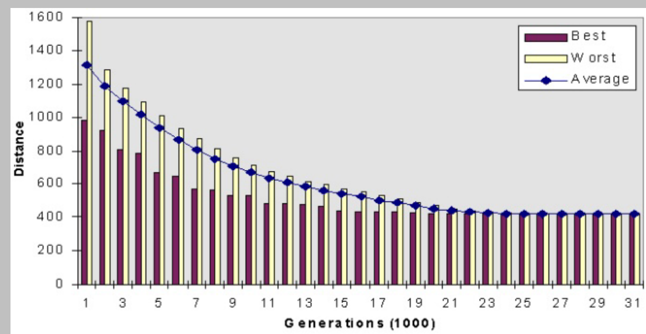
TSP30 (distance = 652)



TSP30 (distance = 420)

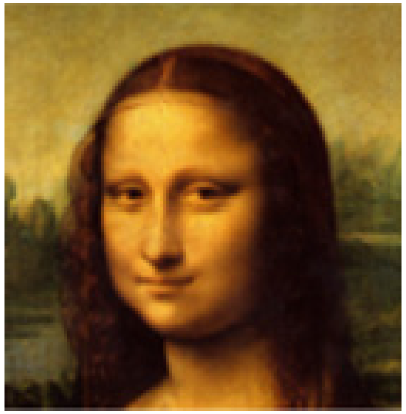


TSP30 (overall performance)

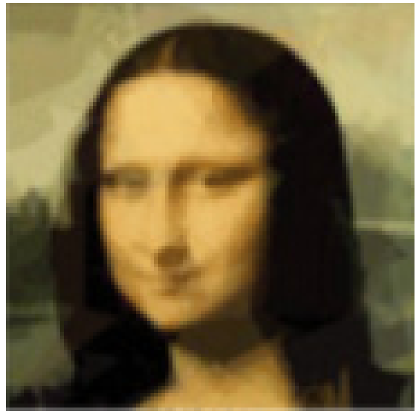


Example: Genetic Vectorizing

- This GA algorithm was developed by **Roger Alsing**.
- The idea is to regenerate an image using polygons.
- Polygons are created in a population of chromosomes, and they try to evolve in order to fit the source image.
- The image was regenerated with **50 semi-transparent polygons** after **904314 iterations**.
- As the regeneration is based on vectors, future works may enable recreating objects with a 3D printer.

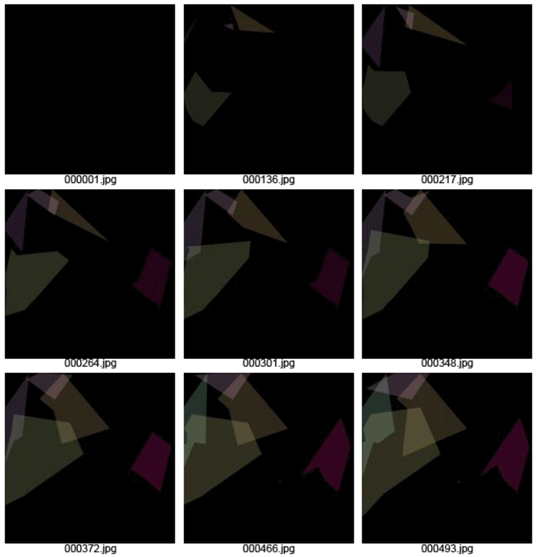


ORIGINAL

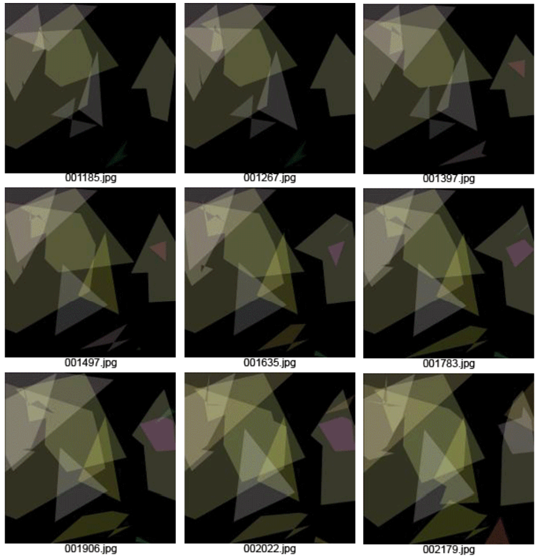


REGENERATED

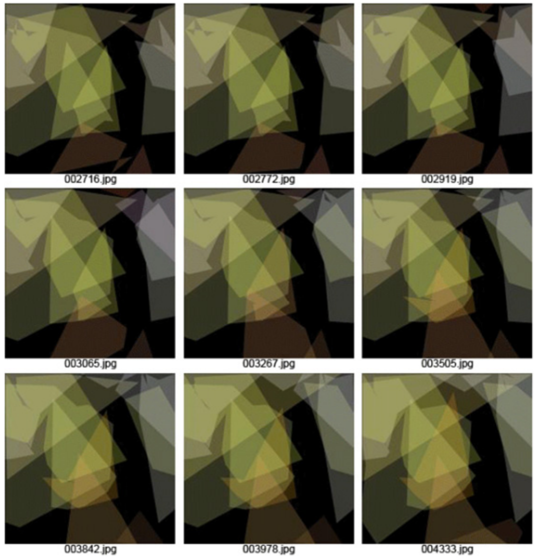
http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/



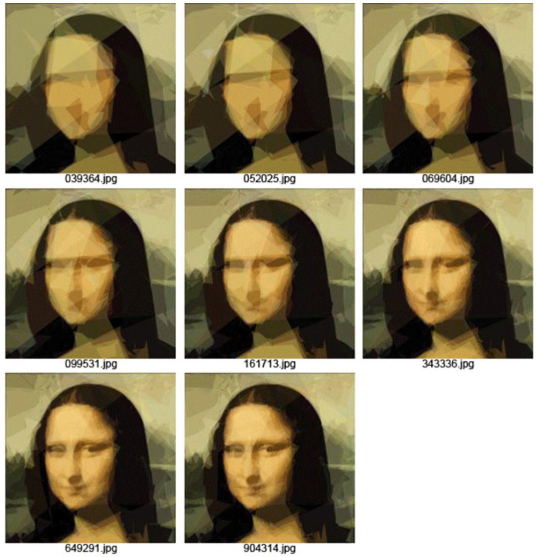
The image starts with random polygons.



Each attempt is scored against the real image.



The iterative process results in incremental improvements.



…and viola! Evolution yields the Mona Lisa “code”.

THANK YOU