Introduction to Evolutionary Computation

Claus Aranha

2024-10-22

Self-Introduction

- Name: Claus Aranha
- Origin: Brazil
- Research: Evolutionary Computation and Artificial Life (EC Laboratory)
- Hobbies: Game Programming, Spiders

Learn more at: https://conclave.cs.tsukuba.ac.jp

Outline

In this lecture we talk about Evolutionary Computation, which is one kind of artificial Intelligence.

We introduce an exercise to evolve the shape of simulated robots using Genetic Algorithms.

Parts of this lecture:

- What is Evolutionary Computation
- Evolving Robot Shape with Genetic Algorithms
- Other Evolutionary Algorithms
- Ideas to improve the simulation

Artificial Intelligence and Evolutionary Computation

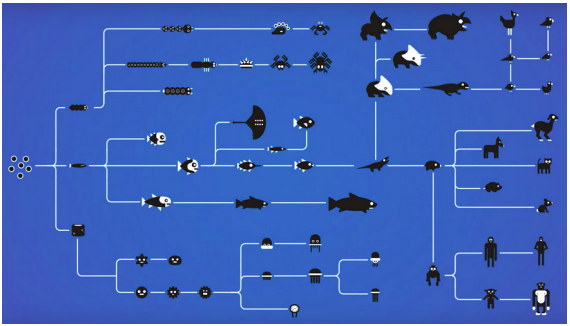
Artificial Intelligence takes inspiration from many sources of intelligence:

- Representation Knowledge and Planning: Inspired on human reasoning systems;
- Expert Systems: Inspired by human knowledge and experience;
- Neural Networks: Inspired by brain biology;
- Evolutionary Computation: Inspired by evolution of life in nature;

Evolutionary Computation is different from other forms of AI, because it is not inspired by human beings.

Why search for AI in natural Evolution?

- Natural Evolution is amazing:
  - Evolution created a large variety of creatures;
  - Each creature is well adapted to their environment;
  - Nature does not know what each creature needs

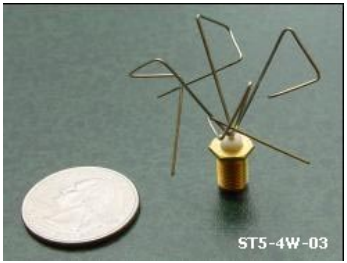


- Can we use evolution to find artificial intelligence?

Evolutionary Computation is Useful

Evolutionary Computation is a General Problem Solver.

- It can be applied to many different kinds of problems; (discrete, continous, symbolic, non-linear, etc)
- You don't need a lot of domain knowledge;
- It is specially useful for design problems;
- It can produce surprising results;



What is Evolution?

Natural evolution consists of three parts:

- Natural Selection: Creatures that are well adapted to their environment survive and have many children.
- Genes: Creatures transmit their genes to their children. Children are similar to their parents.
- Crossover and Mutation: Sexual reproduction causes a child to receive genes from two parents. Mutation causes new genes (good and bad) to appear.

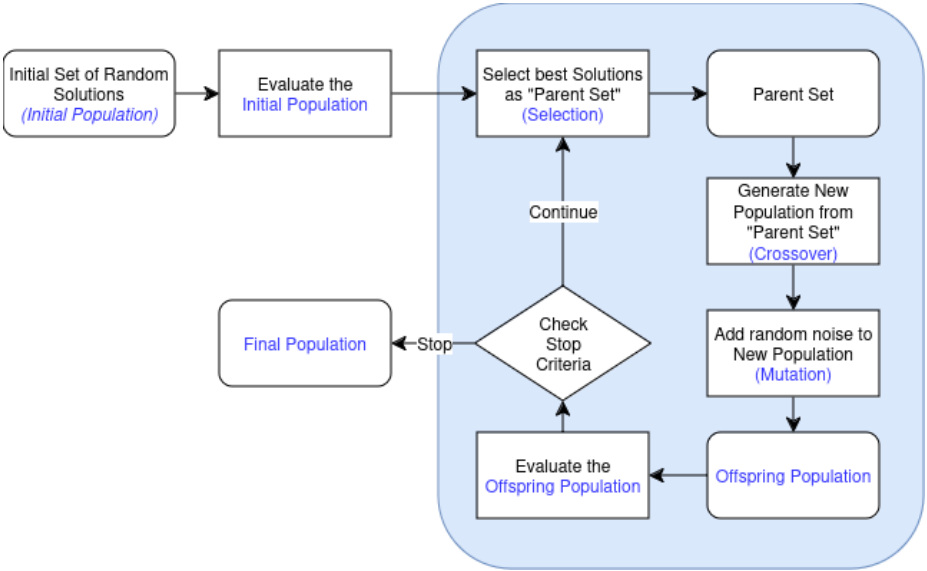
These three components, over millions of years, let to all the different species today.

Evolutionary Computation

To create artificial evolution, we have to reproduce the three components of natural evolution:

- Natural Selection → Fitness Function: The fitness function evaluating the quality of a solution represents the environment. The higher the score, the more offspring a solution will have.
- Genes → Data Structure: The data representing the solution is the "Gene" for the evolutionary algorithm. In most cases, the Gene and the Solution are identical.
- Crossover and Mutation → Variation Operators: To create new solutions, we define functions that modify existing one with random noise.

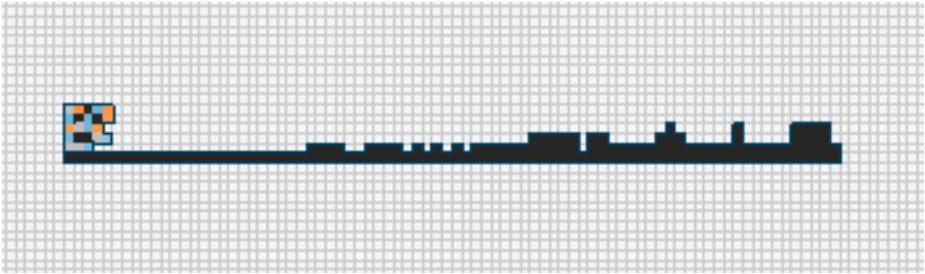
The Evolutionary Computation Loop



Problem Setting

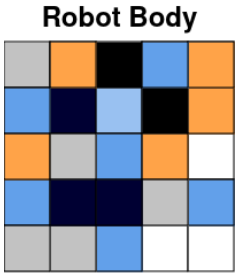
Let's illustrate how Evolutionary Computation can be used to solve a simple simulated robotics problem.

In this problem, a simulated robot has to move through difficult terrain. We will use Genetic Algorithm to evolve the best robot body for this task.

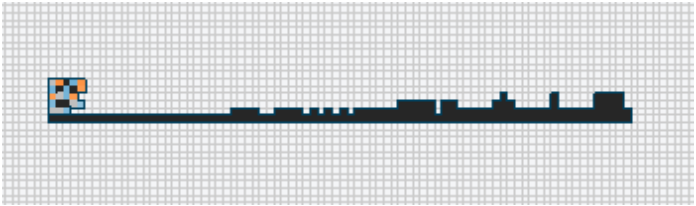


You can download the code for the exercise in this lesson in the following link:
https://codeberg.org/caranha/YASRE/

Simulated Robot Body



The body of the robot is a $n \times n$ grid. ($n = 5$)



Each cell is one of five types:

Color	Type	Explanation
White	Empty	Nothing
Black	Rigid Cell	Passive, does not deform
Gray	Soft Cell	Passive, does deform
Red	Horizontal Muscle	Active, expands vertically
Blue	Vertical Muscle	Active, expands horizontally

Simulated Robot Controller

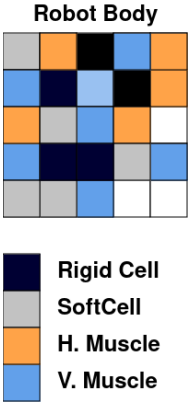
The two muscle cells (Red and Blue) can contract and expand.

The simulator expects a floating point expansion value bigger than 0.

The expansion value is defined as:

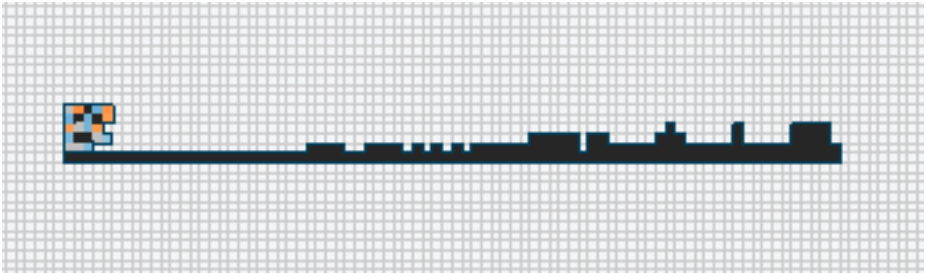
$$D_i = \sin(\text{step}/3 + 0.1 * i) + 1$$

t is the number of the steps since the start of the simulation.
i is the ID of the muscle cell.



This equation gives a cyclic expansion from 0 to 2, with a small offset for each actuator.

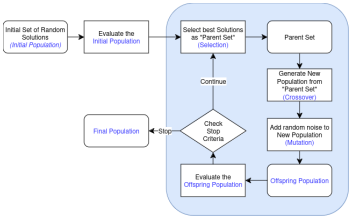
Robot Score



At the end of the simulation, the robot score is calculated as the euclidean distance between the robot's center of mass and its starting point.

(Note: Theoretically, the robot could get a large score by jumping really high too)

Genetic Algorithm for the Robot

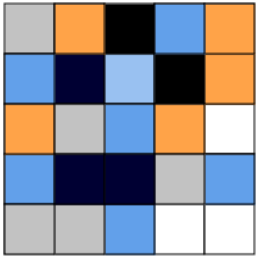


We will use the Genetic Algorithm to find a good robot body for this task:

1. Generate 20 random robots as the initial population;
2. Evaluate the population; ← simulation score
3. Select the best individuals of the population to create 20 offspring;
4. Create offspring using crossover and mutation;
5. Set the offspring as the new population;
6. Return to (2)

Robotic Genes

Robot Body



Robot Genes

2	3	1	4	3
4	1	4	1	3
3	2	4	3	0
4	1	1	2	4
2	2	4	0	0

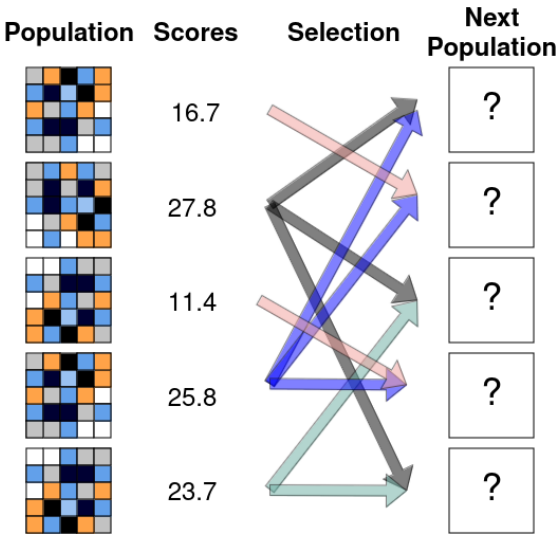
Data Structure

2	3	1	4	3	4	1	4	1	3	3	2	4	3	0	4	1	1	2	4	2	2	4	0	0

Each cell is represented by an integer between 0 and 4.

This data format can be easily manipulated by the program.

Selection of the Fittest

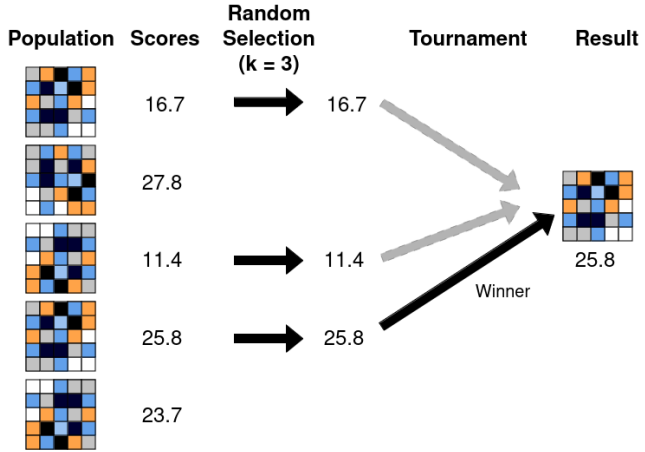


Individuals with higher fitness have higher probability to have offspring.

Individuals with lower fitness have lower probability to have offspring.

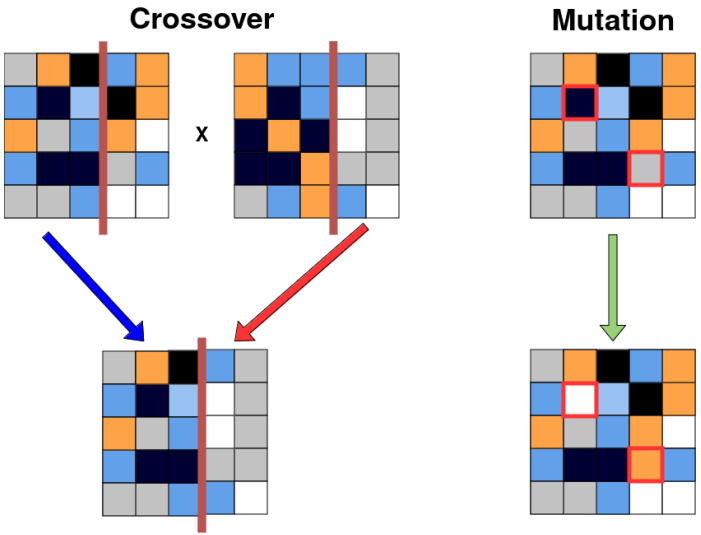
Selection of the Fittest – Tournament Selection

The tournament selection simulates creatures competing for mating.



The fittest individual has an offspring.

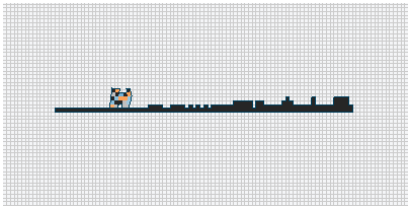
Creating a new Robot: Crossover and Mutation



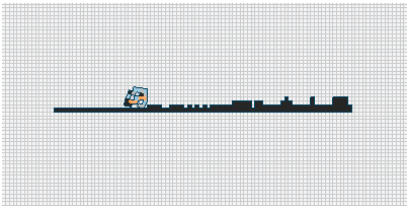
Results of the Genetic Algorithm

Look at the GIF folder

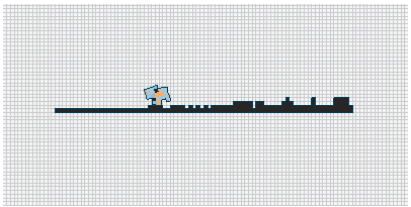
Generation 0



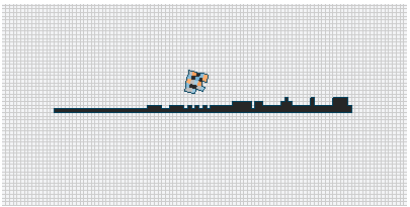
Generation 60



Generation 100



Generation 400



The results aren't very good... How to improve?

Choose better parameters

Genetic Algorithm has many parameters: Population size, Selection K, Mutation Chance... If we change these parameters, we could obtain a better result.

Improve the Algorithm

The crossover and mutation are a bit too simple.

We could also keep the best individual every generation (Elite Strategy).

Run for more time

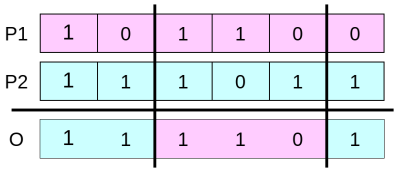
Sometimes evolution just takes time... Pay attention to convergence!

Other EC Algorithms: Genetic Algorithm

Holland, 1975

Key ideas:

- Population tracks several solutions in parallel;
- Selection bias search to good solutions;
- Crossover exchanges building blocks;



These three ideas imply that good building blocks spread through the population (schemata theory)

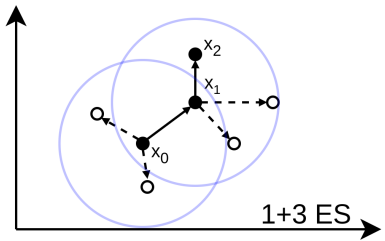
Genetic Algorithm

Require: Binary Representation of Solution
Require: $P \in \{1,0\}^{d \times p}$ (Initial Population)
1: while not terminate condition do
2: Evaluate fitness of population, $f(P)$
3: for $i : 0 \rightarrow p$ do
4: Select parents x_a,x_b by fitness
5: $o_i \leftarrow$ crossover(x_a, x_b)
6: $o_i \leftarrow$ mutation(o_i)
7: Insert o_i in P_{new}
8: end for
9: $P \leftarrow P_{\text{new}}$
10: end while
11: Output best $x \in P$

Other EC Algorithms: Evolution Strategy (ES)

Rechenberg and Schwefel, 1971

Key idea: Offspring with beneficial mutations replace the parent.



Later versions adapt the mutation step s during the optimization.

Evolution Strategy (1+λ-ES)

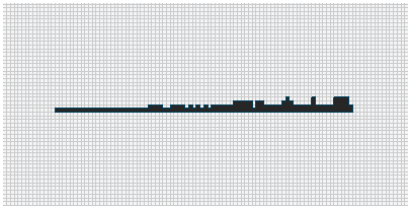
Require: $x \in \mathbb{R}^d$ (Initial Solution)
Require: λ: number of offspring
Require: s: step size

1: while not terminate condition do
2: for $i : 0 \rightarrow \lambda$ do
3: $o_i \leftarrow x + N(0,1)^d * s$
4: end for
5: $o_b \leftarrow$ best o_i
6: If o_b better than x: $x \leftarrow o_b$
7: end while
8: Output x

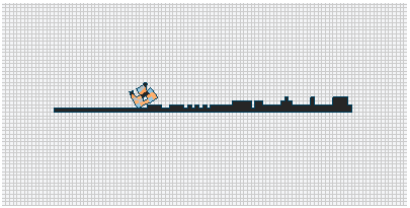
Results of the Evolution Strategy

Look at the GIF folder

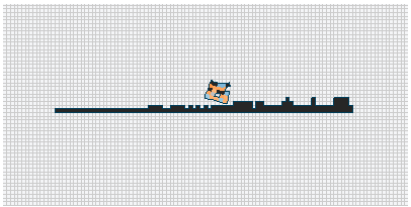
Generation 0



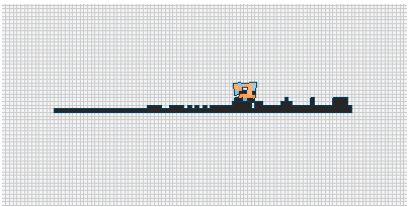
Generation 26



Generation 271



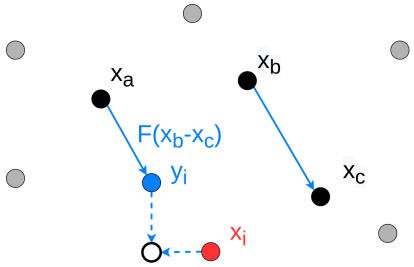
Generation 486



Other EC Algorithms: Differential Evolution

Storn and price, 1997

Key idea: Mutation based on the difference of two random solutions means that the mutation step automatically adapts to the size of the population.



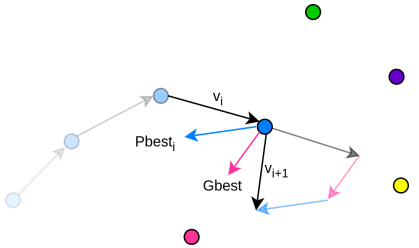
Differential Evolution

Require: $P \in \mathbb{R}^{d \times p}$ (Initial Population)
Require: Parameters F, Cr
1: while not terminate condition do
2: for $i : 0 \rightarrow p$ do
3: Choose random $x_a, x_b, x_c \in P$
4: $y_i = x_a + F(x_b - x_c)$
5: for $j : 0 \rightarrow d$ do
6: $y_{i,j} \leftarrow x_{i,j}$ with prob Cr
7: end for
8: If y_i better than x_i: $x_i \leftarrow y_i$
9: end for
10: end while
11: Output best $x \in P$

Other "EC" Algorithms: Particle Swarm Optimization

Dorigo 1992

Key idea: Solutions move through the search space like particles, influenced by the current local and global optima. Particles exchange information and track multiple optima.



Particle Swarm Optimization

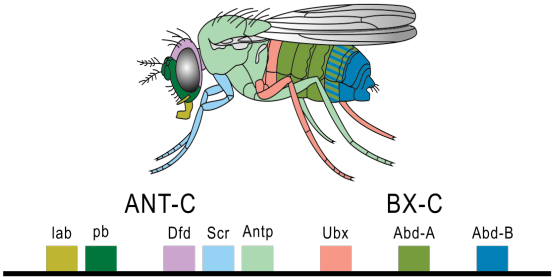
Require: $P \in \mathbb{R}^{d \times p}$ (Initial Population)
Require: $V \in \mathbb{R}^{d \times p}$ (Speed Vector)
1: while not terminate condition do
2: Pbest $\leftarrow P$, Gbest $\leftarrow$ best(P)
3: for $i : 0 \rightarrow p$ do
4: $V_i = V_i + w_p * \text{Pbest}_i, w_g * \text{Gbest}$
5: $P_i = P_i + V_i$
6: Update Pbest_i
7: end for
8: Update Gbest
9: end while
10: Output best $x \in P$

Other Improvements: Changing the gene to a structured representation

In the current gene representation, each cell has an individual gene.

However, the relations between genes and body shape in animals is modular (arms, legs, etc).

Would it improve the results if the Genetic Algorithm used a modular gene architecture?



Other Improvements: Changing the gene to include control elements

In the current gene representation, only the type of the cell is encoded.
However, What if we also encoded the offset and multiplier of the control sine wave?

$$D_i = \sin(\text{step}/M_i + O_i) + 1$$

- M_i is the sine multiplier of cell i. It indicates how fast the cell expands.
- O_i is the sine offset of cell i. It indicates when the cell expands.

Cell Type	1	2	3	3	1	4	1	0	0	1	4	0
Offset	0.1	0.3	0.2	0.3	0.7	0.8	0.5	0.3	0.4	0.1	0.3	0.3
Multiplier	2	3	2.5	1	4	3.3	2.2	1	3	1	3	3

One of my students found that adding this to the gene improved the solution by a lot.

Other Improvements

There are many more things that can be improved in this simulation:

- How can we make the robots good for many scenarios?
  (Robustness)
- Can we have many scenarios, and the robots choose the best one for them?
  (Niching)
- Can we have multiple robots interact and evolve together?
  (Co-evolution)

All these ideas exist in nature. Which ones are necessary to evolve intelligent creatures?

The End – Key ideas

- Evolutionary Computation is a group of algorithms based on Natural Evolution
- We showed how to evolve a Robot using Genetic Algorithms
- I hope that you can download the code, and improve it as you wish!

`https://codeberg.org/caranha/YASRE`