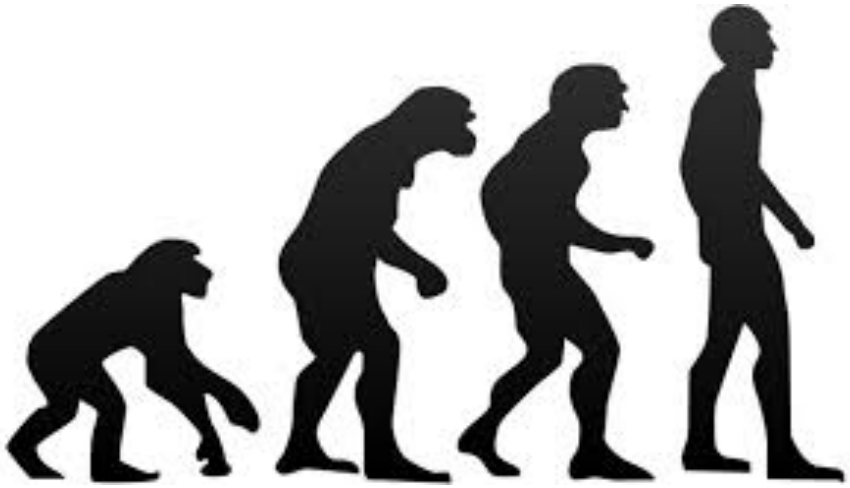
Genetic Algorithm

Alexandre Bergel
DCC - University of Chile
http://bergel.eu
04-12-2025

Goal of today

Give an overview of *genetic algorithm*

Examples of *problems* that can be *addressed* with GA



Premise of Genetic Algorithm

Natural selection is pioneered by *Charles Darwin* (1809-1882), biologist who worked on *natural evolution*

"On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life", 1859

"One general low, leading to the advancement of all organic beings, namely, vary, let the strongest live and the weakest die"

— Charles Darwin

“Guess the 3-letter word I have in mind”

Secret word: *cat*

“Guess the 3-letter word I have in mind”

“For each try, I tell you the number of correct letters”

Secret word: *cat*

"cow"

1

"poz"

0

"gaz"

1

Secret word: *cat*

gaz_{1}

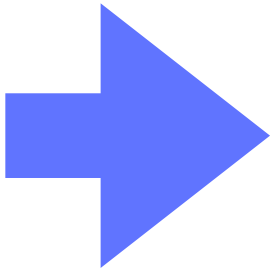
cow_{1}

poz_{0}

gaz_1

cow_1

poz_0

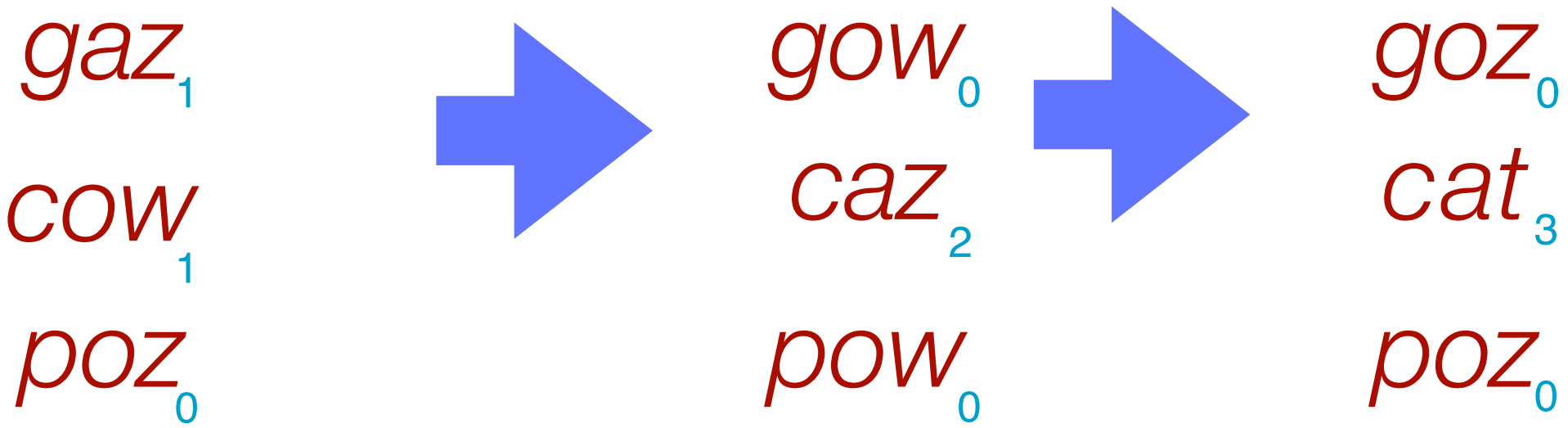


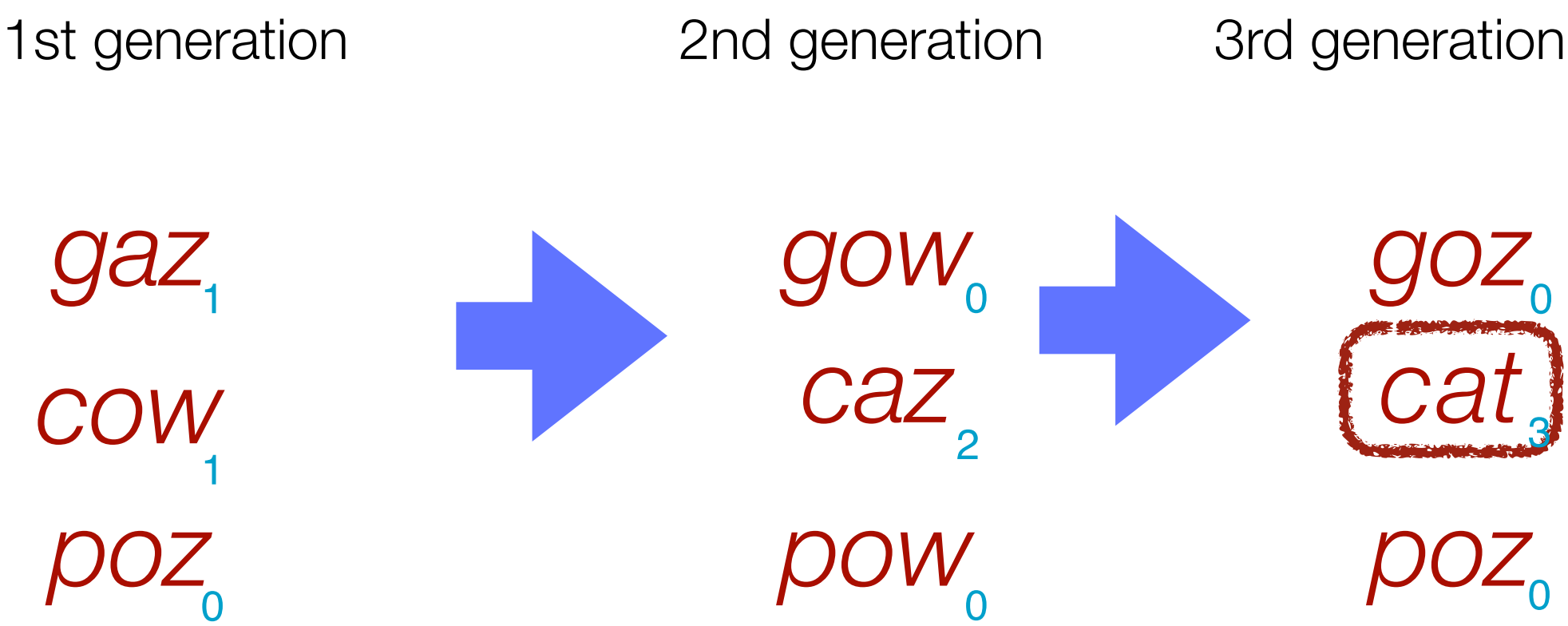
Using genetic operators, the words are combined and some letters are randomly modified

gow_0

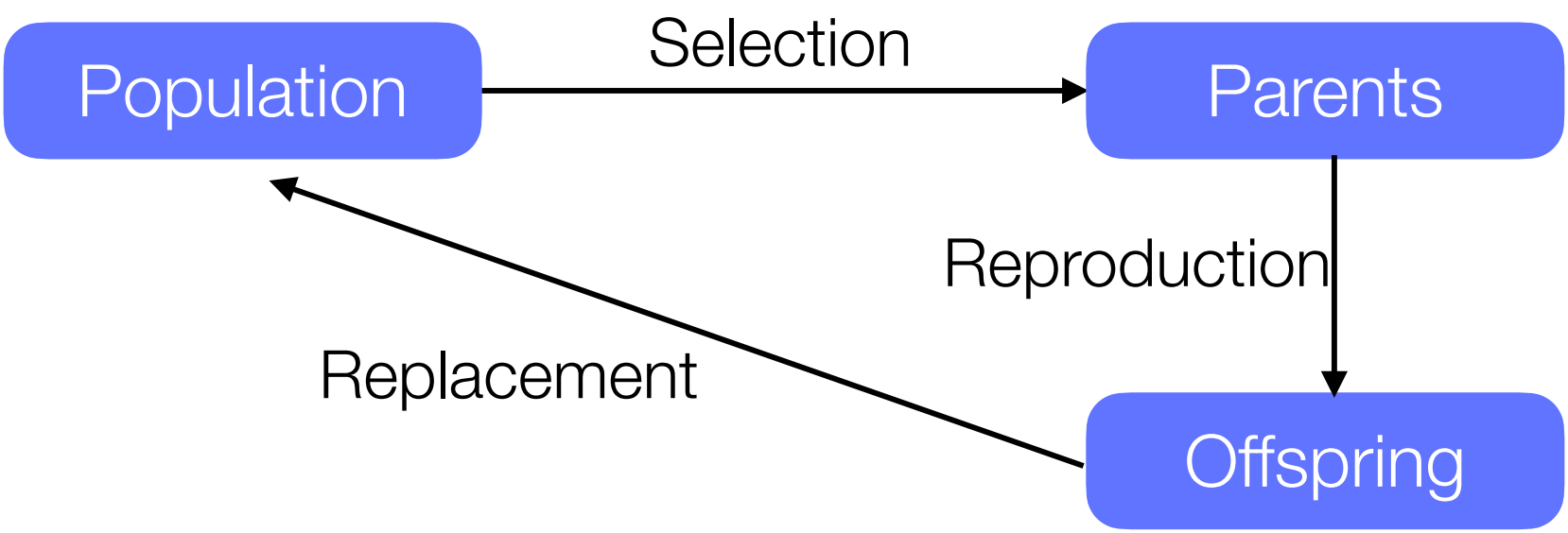
caz_2

pow_0

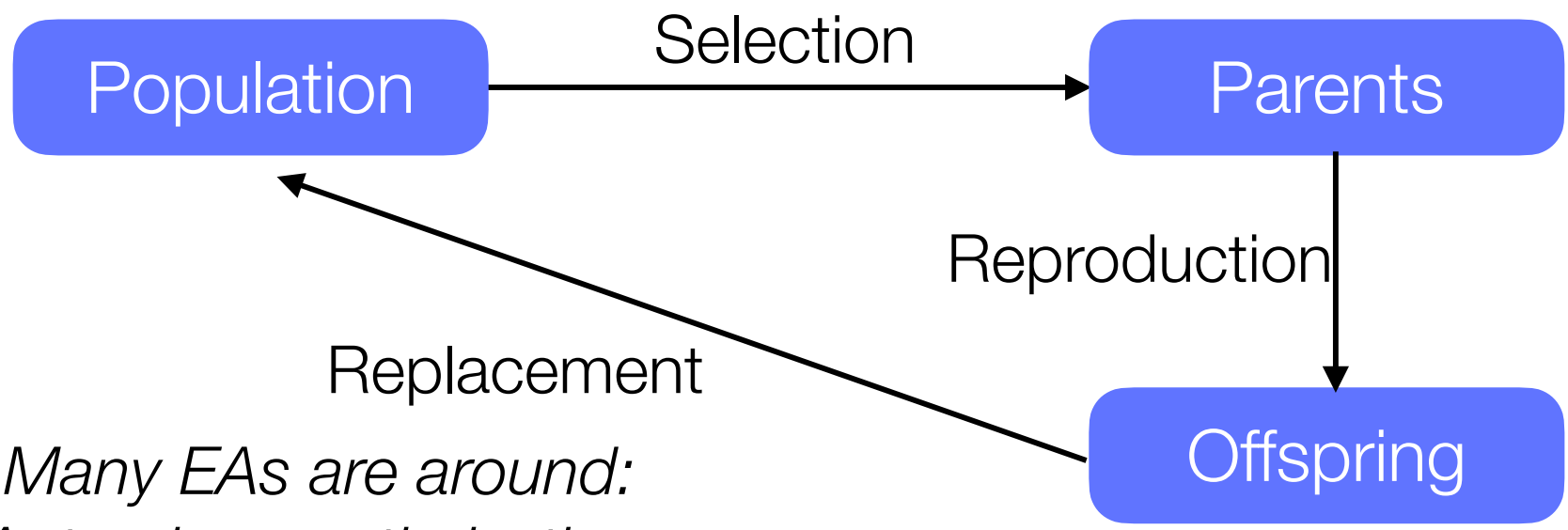

gaz 1
cow 1
poz 0
gow 0
caz 2
pow 0
goz 0
cat 3
poz 0


1st generation
2nd generation
3rd generation
gaz1
cow1
poz0
gow0
caz2
pow0
goz0
cat3
poz0

Flow chart of an evolution algorithm

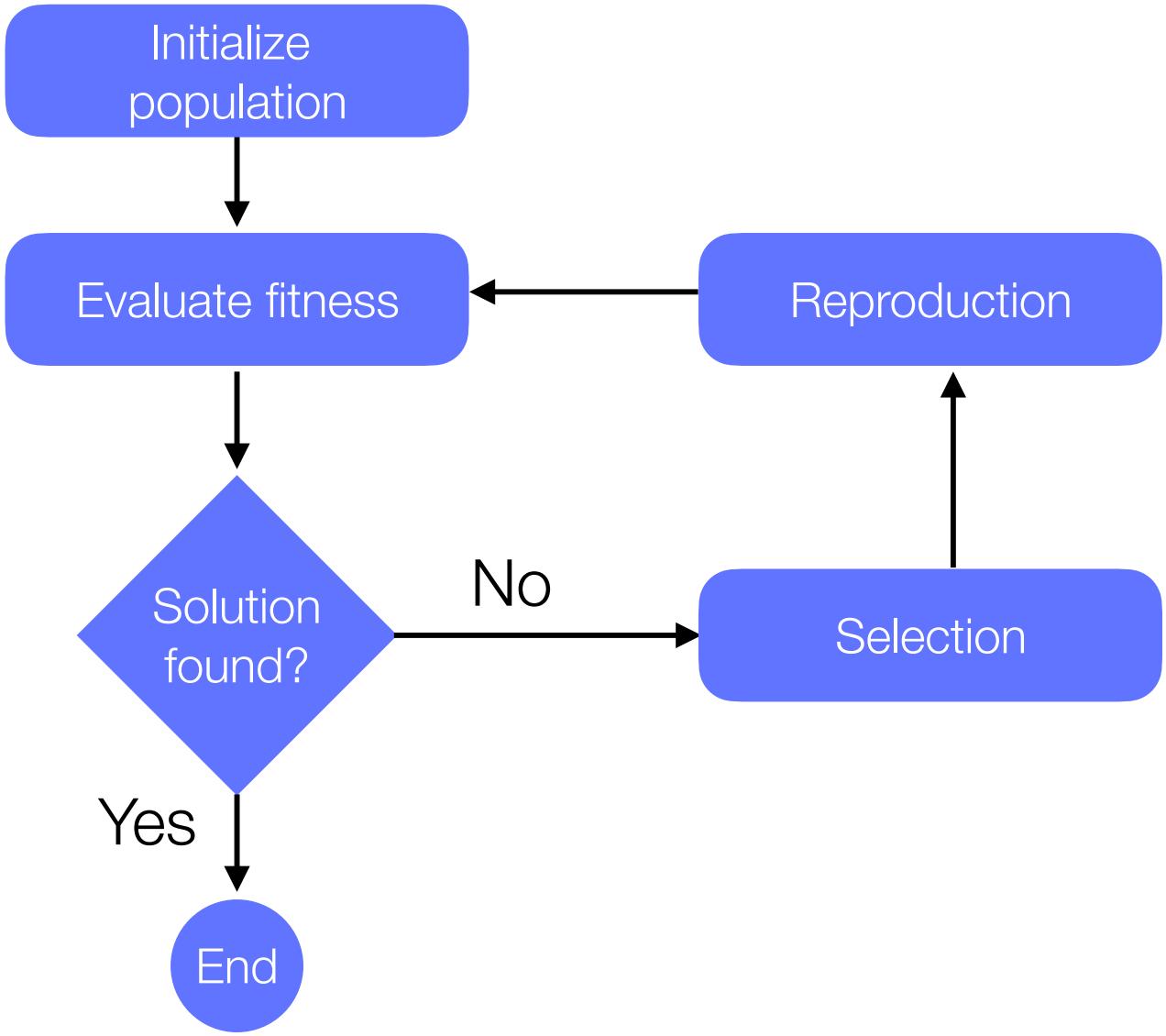


Flow chart of an evolution algorithm

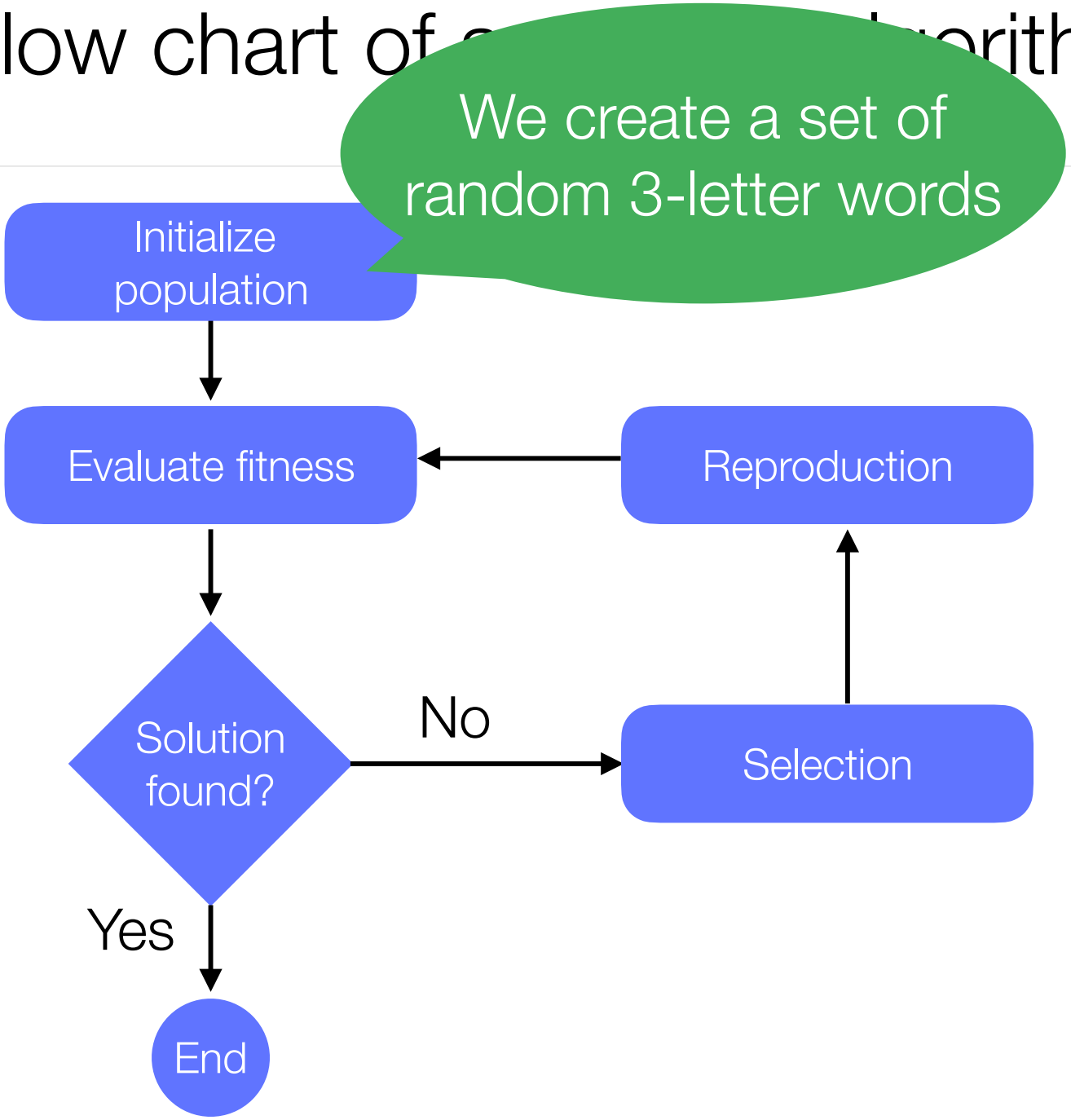


Many EAs are around:
Ant colony optimization,
Artificial immune system,
Cultural algorithms,
Genetic Algorithm,
Genetic Programming,
Grammatical evolution,
…

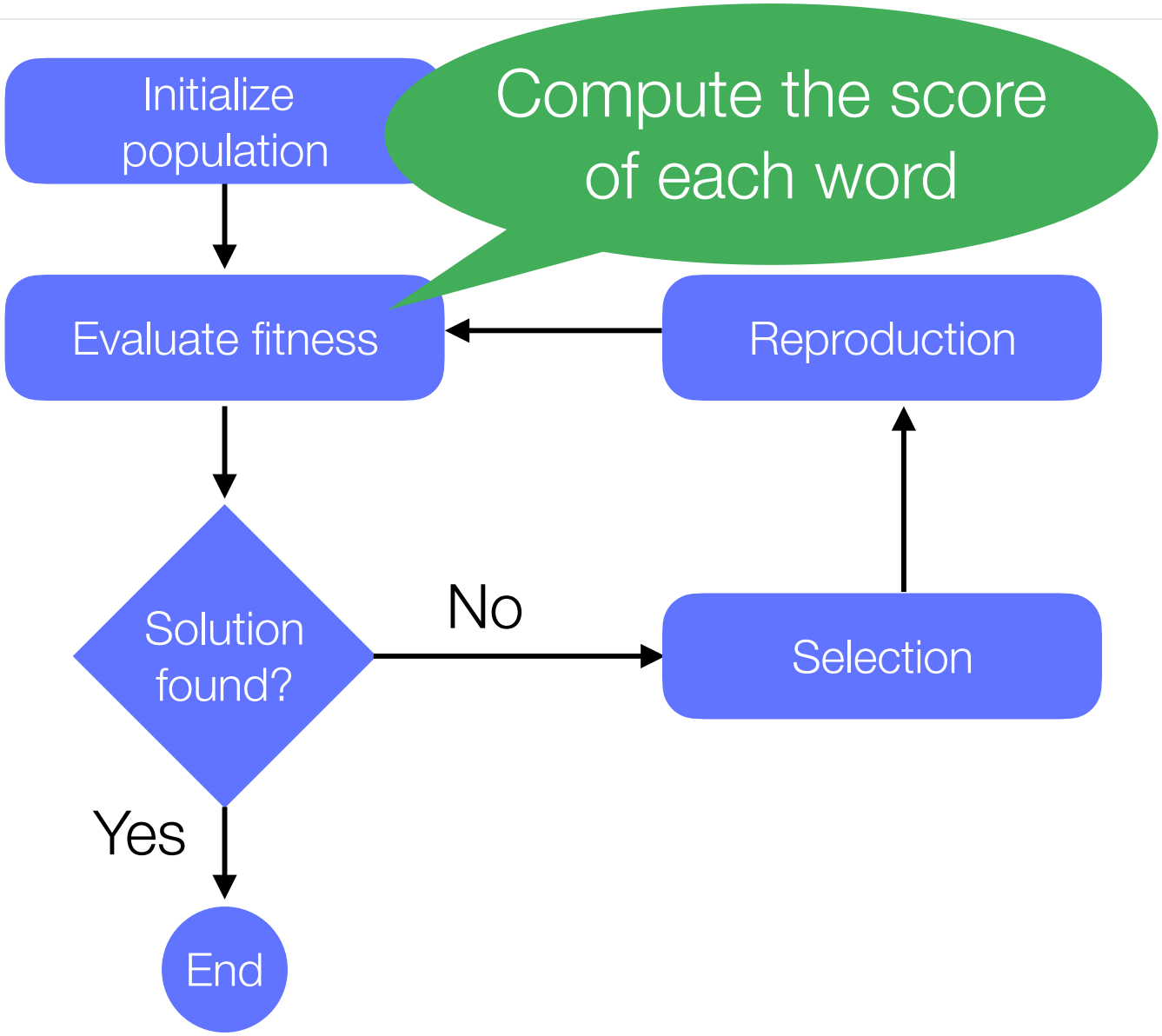
Flow chart of a genetic algorithm



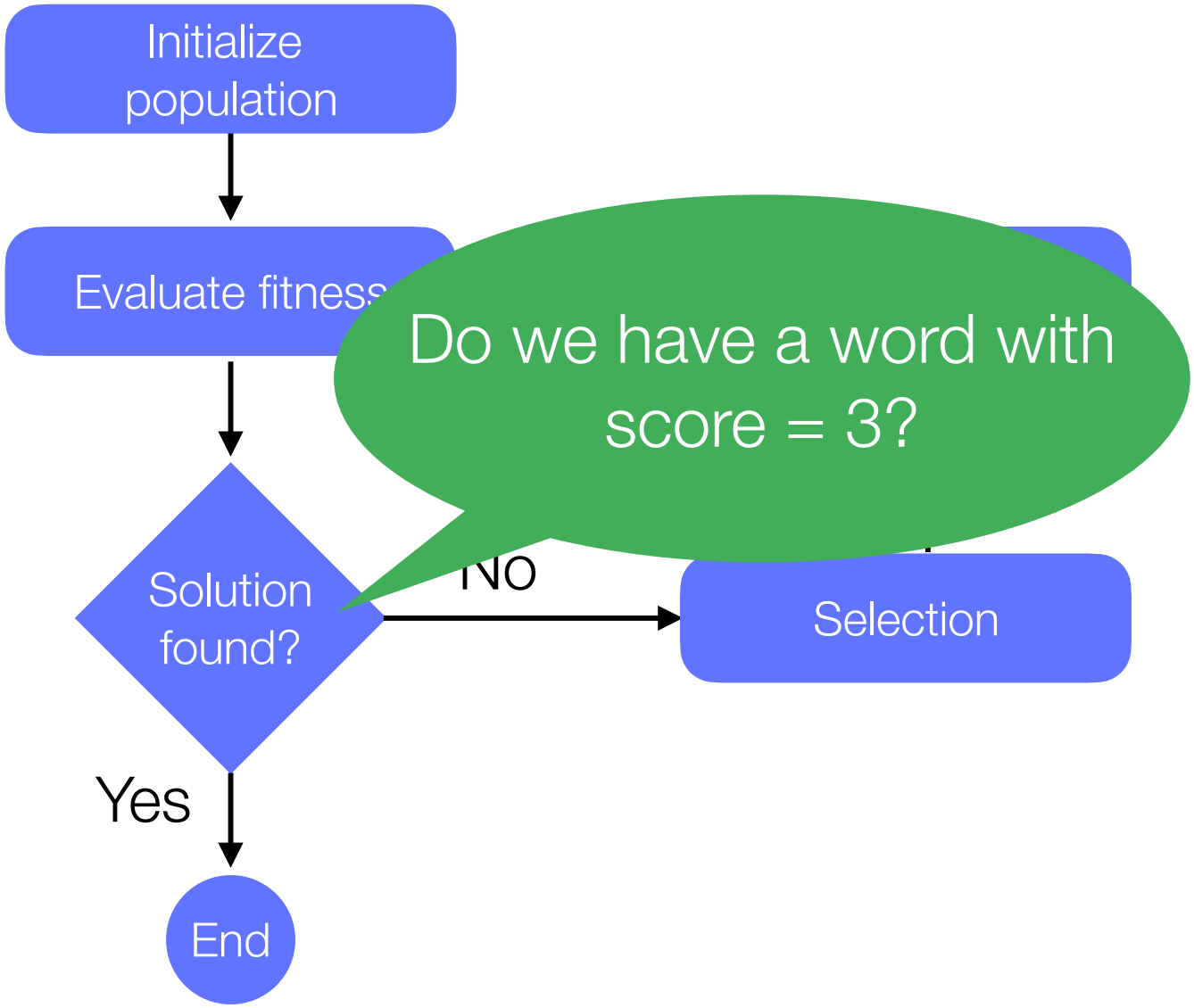
Flow chart of a genetic algorithm



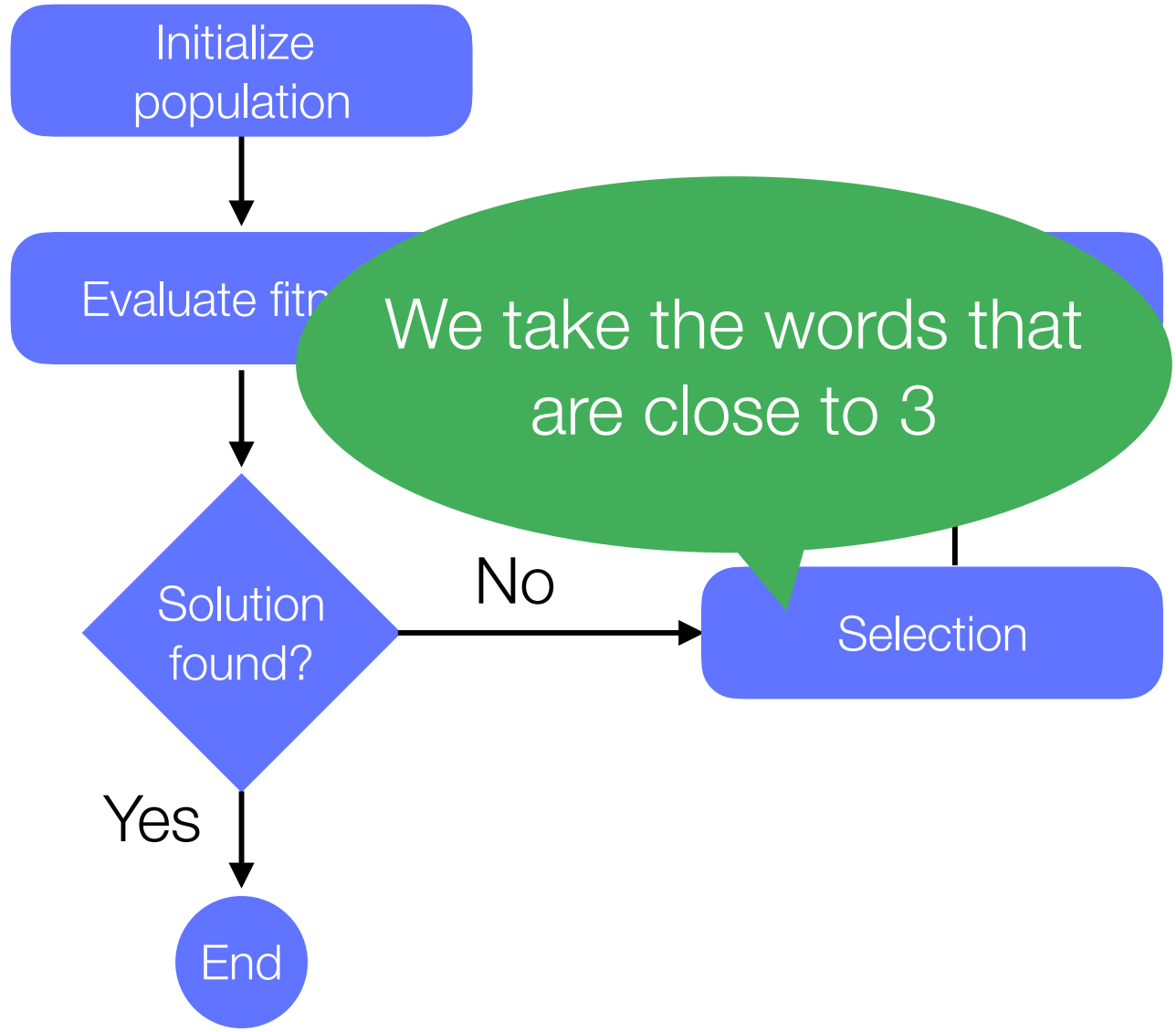
Flow chart of a genetic algorithm



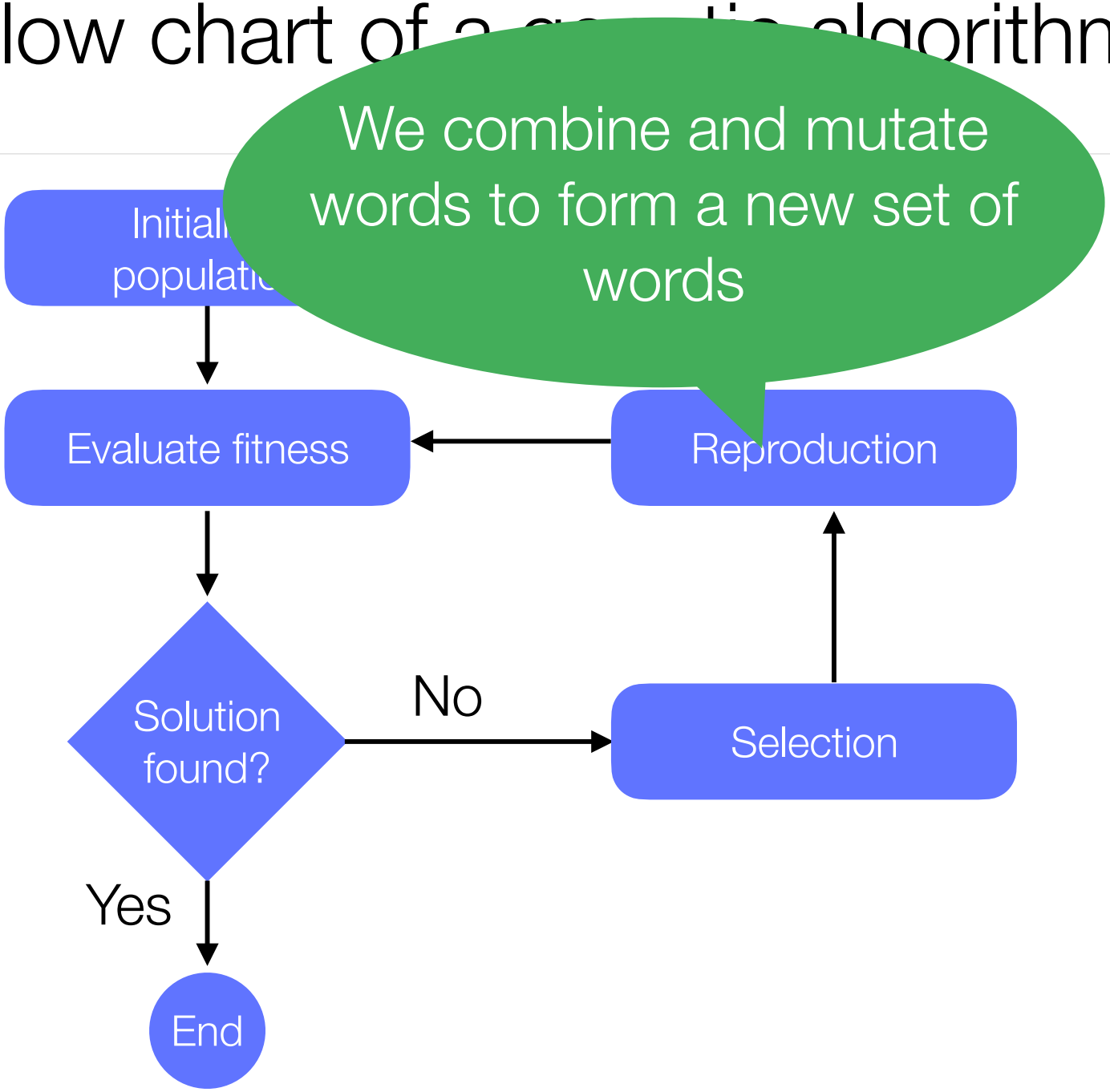
Flow chart of a genetic algorithm



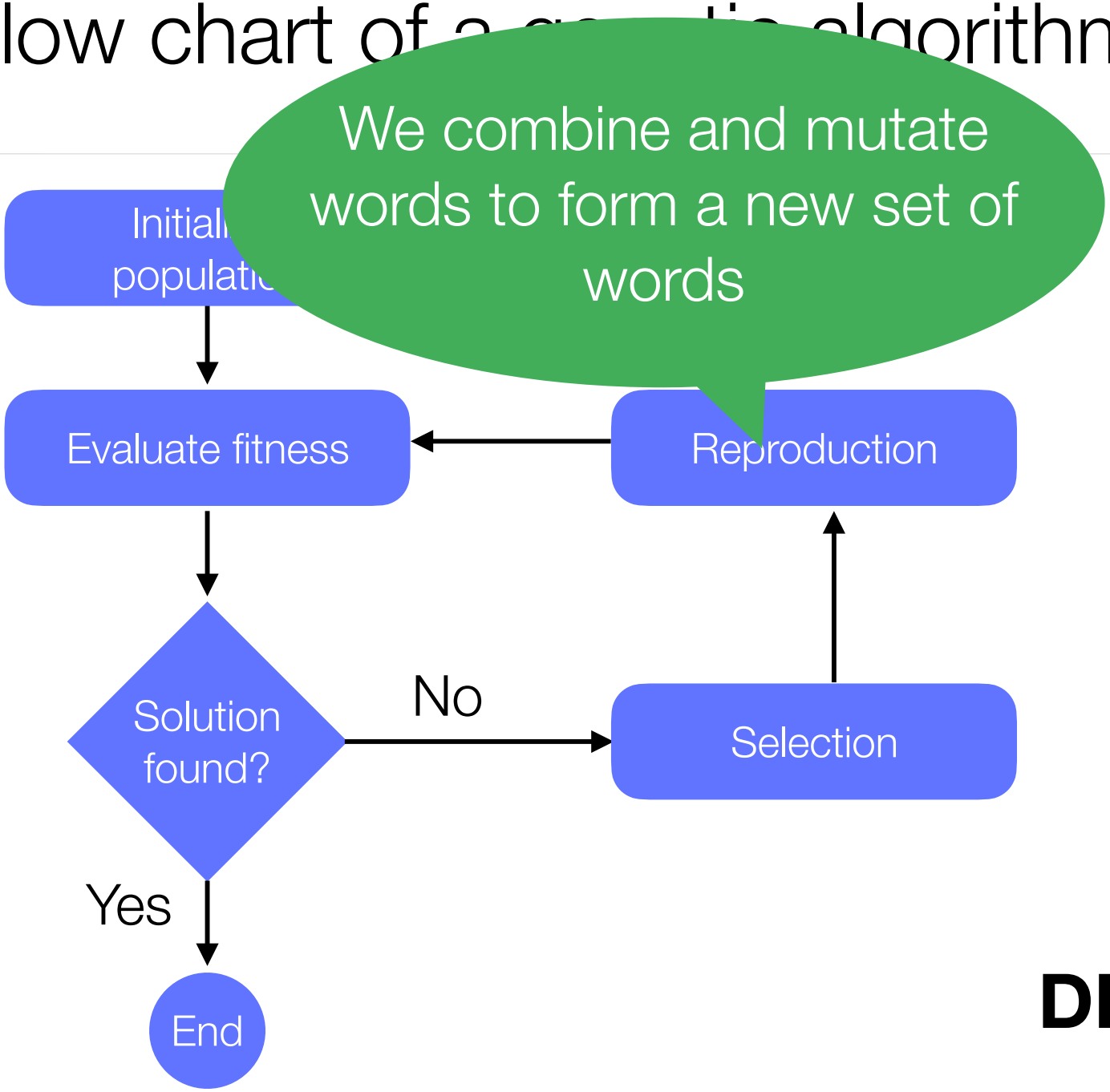
Flow chart of a genetic algorithm



Flow chart of a genetic algorithm



Flow chart of a genetic algorithm



DEMO

Genetic Algorithm in a Nutshell

Evolutionary computation technique that automatically solves problems without specifying the form or structure of the solution in advance

Generally speaking, *genetic algorithms are simulations of evolution*, using biological genetic operations

Finding x such as $f(x) = y$ is maximal, x is a tuple of any arbitrary dimension and domain value, $y \in \mathbb{R}$

Terminology: Individual

An *individual* represents an element in a population

Each individual has a chromosome, composed of *genes*

Individual

Gene = index in a chromosome

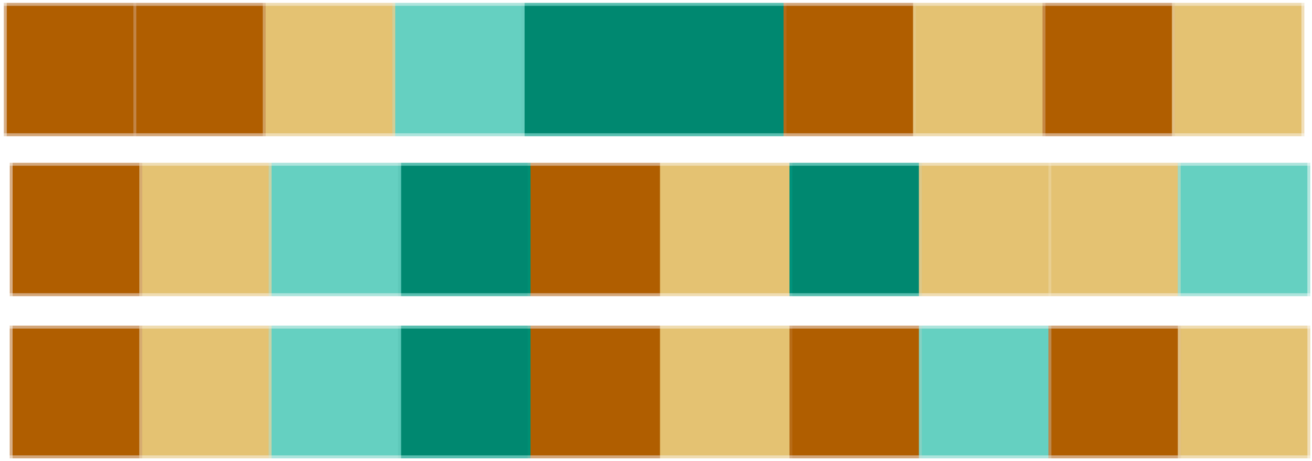
Allele = value of a gene

Terminology: Population

A *population* is a set of individuals

Generally, the *population size is fixed* over time. Individuals are replaced at each generation, but the number of individuals remains constants.

All individuals of the population *have the same size*



Terminology: Fitness function

The fitness function evaluates how *fit* an individual is

$f($  $) = y \qquad y \in \mathbb{R}$

The whole idea of genetic algorithm is to search for the individual that maximizes the fitness function

Example: optimizing a server

Consider a server running in Java

The Java virtual machine, which has over 200 options

What are the options that consume the least amount of memory to answer HTTP requests

$$f(x) = y$$

x is a set of options, e.g., `[ -Xgc:parallel, -ms32m, -mx200m]`

y is the amount of memory (in bytes)

$f(x)$ is computed by launching the server using the options and sending 1000 requests

Example: optimizing a server

Consid

Th ons

Wh unt of memory to answe

We look for the optimal x that minimize f(x)

$$f(x) = y$$

x is a set of options, e.g., `[ -Xgc:parallel, -ms32m, -mx200m]`

y is the amount of memory (in bytes)

$f(x)$ is computed by launching the server using the options and sending 1000 requests

Example: optimizing a server

Consider a server running in Java

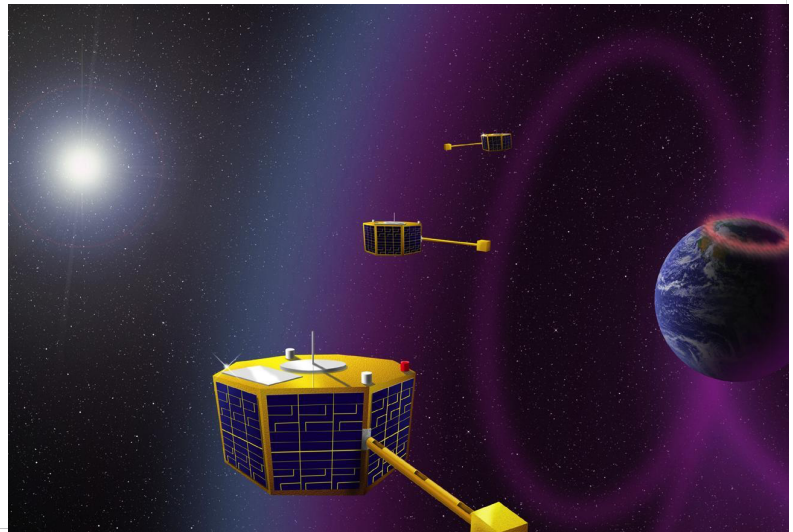
The Java virtual machine, which has over 200 options

Optimization of Java Virtual Machine Flags Using Feature Model and Genetic Algorithm -- Felipe Canales, Geoffrey Hecht, Alexandre Bergel. *Proceedings of 12th International Conference on Performance Engineering - Work-in-Progress/Vision track at ICPE (ACM/SPEC ICPE-WIP'21)*
http://bergel.eu/MyPapers/Cana21a-JVMGlagsAndGA.pdf

$f(x)$ is computed by launching the server using the options and sending 1000 requests

Example: space antenna

Evolved antenna, produced by NASA in 2006.
Used in 3 satellites that take measuremen
Earth magnetosphere.
Satellites used this antenna to communicate with the ground



Example: space antenna

List of commands:
forward(length, radius)
rotate-x(angle)
rotate-y(angle)
rotate-z(angle)

$f(x) = y$

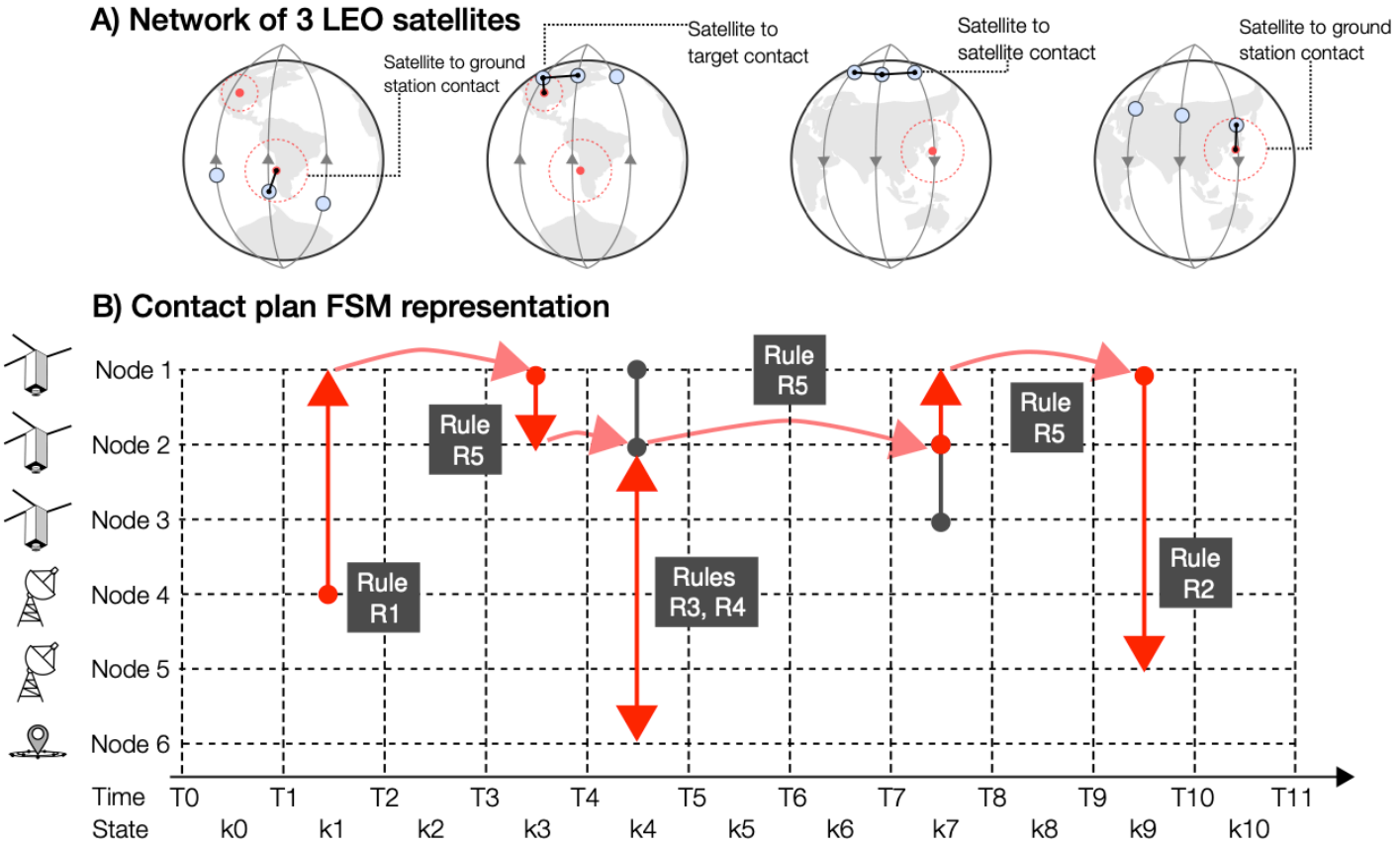
voltage and impedance for radio waves

Automated Antenna Design with Evolutionary Algorithms

Gregory S. Hornby* and Al Globus
University of California Santa Cruz, Mailtop 269-3, NASA Ames Research Center, Moffett Field, CA
Derek S. Linden
JEM Engineering, 8683 Cherry Lane, Laurel, Maryland 20707
Jason D. Lohn
NASA Ames Research Center, Mail Stop 269-1, Moffett Field, CA 94035

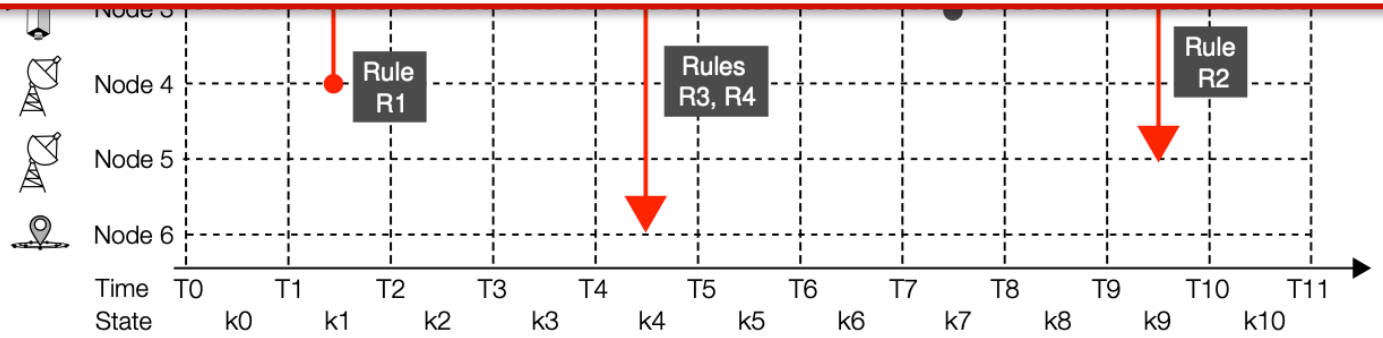
https://ntrs.nasa.gov/api/citations/20060024675/downloads/20060024675.pdf

Example: identifying communication routes in a constellation of satellites



Example: identifying communication routes in a constellation of satellites

Nanosatellite constellation control framework using evolutionary contact plan design -- Carlos E. Gonzalez, Alexandre Bergel, Marcos A. Diaz. *Proceedings of 8th IEEE International Conference on Space Mission Challenges for Information Technology - The Space-Terrestrial Internetworking workshop (SMC-IT-STINT)*
http://bergel.eu/MyPapers/Gonz21-Constellation.pdf



Example: software testing

$$f(x) = y$$

Actions on the user interface (e.g., pressing a button, entering a value in the text field, clicking on a menu)

Number of tested functionalities

Randomly testing an application has many applications:

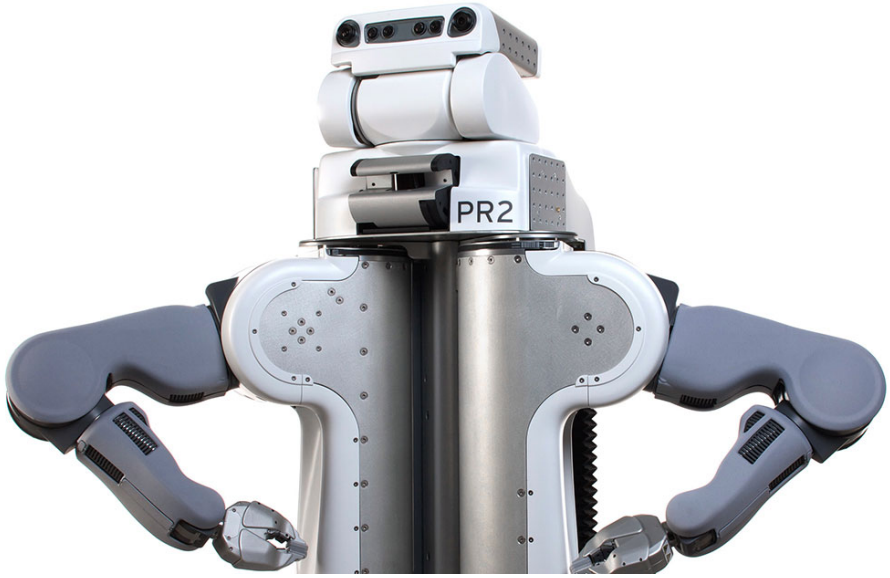
- Finding functional bugs (i.e., which actions crashes my application)
- Finding dead code (i.e., which part of my application is not used)

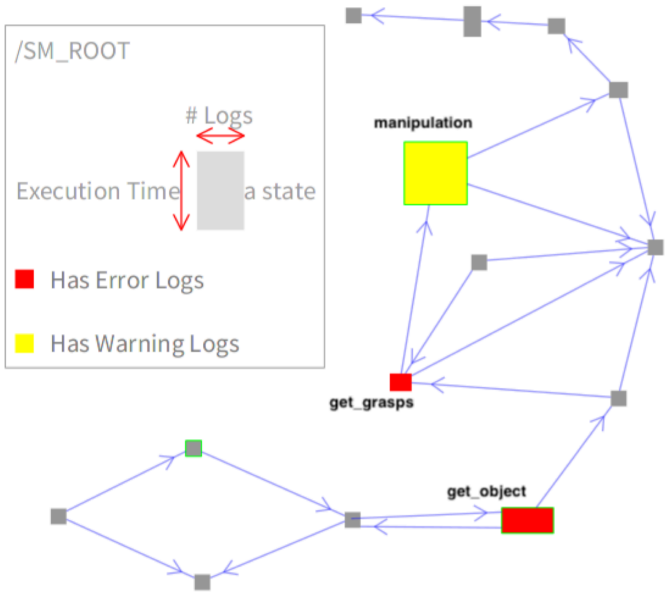
Example: robot testing

$$f(x) = y$$

List of random inputs of a robot state

Number of raised warning/errors

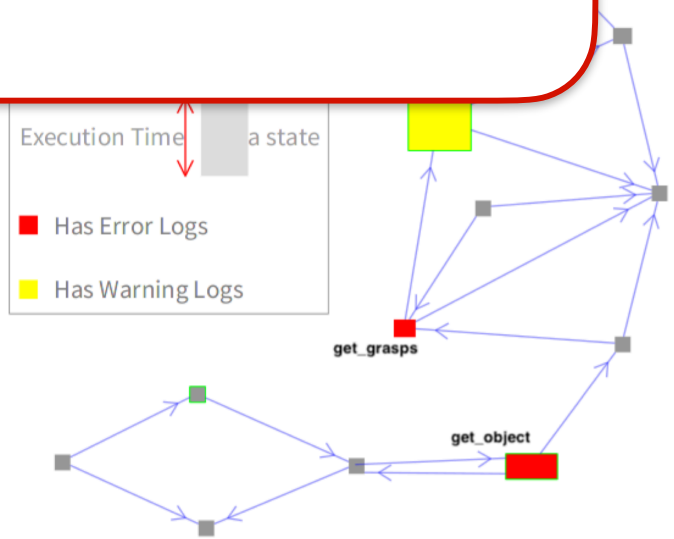




Example: robot testing

$$f(x) = y$$

Fuzz Testing in Behavior-Based Robotics -- Rodrigo Delgado, Miguel Campusano, Alexandre Bergel. *Proceedings of IEEE International Conference on Robotics and Automation (IEEE ICRA'21)*
http://bergel.eu/MyPapers/Delg21a-FuzzTesting.pdf



An Implementation of Genetic Algorithm

The problem to solve is finding a sequence of bits

```
SEQUENCE_OF_BITS_TO_FIND = '10101010101010101010'
```

We need to know the number of genes each individual must have

```
NUMBER_OF_GENES = len(SEQUENCE_OF_BITS_TO_FIND)
```

An Implementation of Genetic Algorithm

We need a way to create a gene:

```
def gene_factory():
  if(random.random() > 0.5):
    return '1'
  else:
    return '0'
```

We need a way to create an individual:

```
def sequence_bit_factory():
    return [ gene_factory() for i in range(NUMBER_OF_GENES)]
```

An Implementation of Genetic Algorithm

We need a way to compute the fitness of each individual:

```
def fitness_bits(anIndividual):
  result = 0
  for a,b in zip(anIndividual, SEQUENCE_OF_BITS_TO_FIND):
    if (a==b):
      result = result + 1
 return result
```

The function zip is useful to combine two collections into a single one:

```
>>> a =[1,2,3]
>>> b = [10,20,30]
>>> zip(a,b)
[(1, 10), (2, 20), (3, 30)]
```

Using Genetic Algorithm

We run the algorithm

```
ga = GA(pop_size=100,
        mutation_rate=0.1,
        fitness=fitness_bits,
        individual_factory=sequence_bit_factory,
       gene_factory=gene_factory,
      termination_condition = lambda f : f == NUMBER_OF_GENES,
        silent = True)
```

Using Genetic Algorithm

We run the algorithm

```
ga = GA(pop_size=100,
        mutation_rate=0.1,
        fitness=fitness_bits,
        individual_factory=sequence_bit_factory,
       gene_factory=gene_factory,
      termination_condition = lambda f : f == NUMBER_OF_GENES,
        silent = True)
```

Indicate the size of the population

Using Genetic Algorithm

We run the algorithm

```
ga = GA(pop_size=100,
        mutation_rate=0.1
        fitness=fitness_bits,
        individual_factory=sequence_bit_factory,
        gene_factory=gene_factory,
        termination_condition = lambda f : f == NUMBER_OF_GENES,
        silent = True)
```

Indicate the mutation rate

Using Genetic Algorithm

We run the algorithm

```
ga = GA(pop_size=100,
        mutation_rate=0.1,
        fitness=fitness_bits,
        individual_factory=sequence_bit_factory,
        gene_factory=gene_factory,
        termination_condition = lambda f : f == NUMBER_OF_GENES,
        silent = True)
```

Provide a way to compute the fitness of an individual

Using Genetic Algorithm

We run the algorithm

Provide a way to create an individual. This is useful for the algorithm to create the initial population

```
ga = GA(pop_size=100,
        mutation_rate=0.1,
        fitness=fitness_bits,
        individual_factory=sequence_bit_factory,
        gene_factory=gene_factory,
       termination_condition = lambda f : f == NUMBER_OF_GENES,
        silent = True)
```

Using Genetic Algorithm

We run the algorithm

Provide a way to create a gene.
This is useful for the mutation operator

```
ga = GA(pop_size=100,
        mutation_rate=0.1,
        fitness=fitness_bits,
        individual_factory=sequence_bit_factory,
        gene_factory=gene_factory,
        termination_condition = lambda f : f == NUMBER_OF_GENES,
        silent = True)
```

Using Genetic Algorithm

We run the algorithm

```
ga = GA(pop_size=100,
        mutation_rate=0.1,
        fitness=fitness_bits,
        individual_factory=sequence_bit_factory,
        gene_factory=gene_factory,
        termination_condition = lambda f : f == NUMBER_OF_GENES,
        silent = True)
```

Provide a way to terminate the algorithm. Any complex condition can be formulated here

Using Genetic Algorithm

We run the algorithm

```
ga = GA(pop_size=100,
        mutation_rate=0.1,
        fitness=fitness_bits,
        individual_factory=sequence_bit_factory,
       gene_factory=gene_factory,
      termination_condition = lambda f : f == NUMBER_OF_GENES,
       silent = True)
```

Write some logs during the execution of the algorithm

An implementation of genetic algorithm

We consider two genetic operations.

Crossover:

```
    # crossover operation
    # crossover([1,2,3,4], [10,20,30,40])
    # => [1, 20, 30, 40]
    def crossover(self, individual1, individual2):
        _tmp = random.randint(1, len(individual1))
        return individual1[:_tmp] + individual2[_tmp:]
```

b = [10,20,30]
b[:2] => [10, 20]
b[2:] => [30]

An implementation of genetic algorithm

We consider two genetic operations.

Mutation:

```
    # Mutate an individual
     # Note that this method does a side effect. I.e.,
it does not create a new individual
     def mutate(self, an_individual):
         index = random.randrange(len(an_individual))
         an_individual[index] = self.gene_factory()
```

An implementation of genetic algorithm

For two given parents, we can create a new individual:

```
    # create a new individual from two parents elements
    def create_new_individual(self, parent1, parent2):
        final_child = self.crossover(parent1, parent2)
        # mutation
        if random.random() < self.mutation_rate:
            self.mutate(final_child)
        return final_child
```

An implementation of genetic algorithm

The selection mechanism is built as:

```
    # Obtain the best individual from a tournament on the
population
    def tournament(self, aPopulation, k = 5):
        best = None
        best_fitness = -1
        for _ in range(1, k):
            ind = random.choice(aPopulation)
            if(best is None or (self.fitness_function(ind) >
best_fitness)):
                best = ind
                best_fitness = self.fitness_function(ind)
        return best
```

An implementation of genetic algorithm

```
    # main method to actually run the algorithm
    # return a tupple best_fitness_list, avg_list, best_individual
    def run(self):
        current_iteration = 0
        best_fitness_list = list()
        avg_list = list()

        # We generate the initial population
        self.population = self.generate_population()

        # We compute all the fitnesses. It is a collection of fitness
values, of the same size than the population
        self.individual_fitnesses = self.get_fitness(self.population)

        # We get the best_individual of the current population
        best_individual = self.get_best_individual()
  …
```

An implementation of genetic algorithm

```
…
        # We loop if we have not reached the maximum number of iterations and if the
termination condition is not met
        while current_iteration <= self.max_iterations and not
self.termination_condition(self.fitness_function(best_individual)):
            self.log("iter {} of {}".format(current_iteration, self.max_iterations))

            best_fitness_list.append(max(self.individual_fitnesses))
            avg_list.append(np.mean(self.individual_fitnesses))

            self.log("best is: {}\nwith {} acc. Avg: {}".format(best_individual,
best_fitness_list[-1], avg_list[-1]))

            # we create a new population
            new_population = []
            for _ in range(self.population_size):
                parent1 = self.tournament(self.population)
                parent2 = self.tournament(self.population)
                new_population.append(self.create_new_individual(parent1, parent2))

            self.population = new_population
            self.individual_fitnesses = self.get_fitness(self.population)
            current_iteration += 1
          best_individual = self.get_best_individual()
  …
```

An implementation of genetic algorithm

```
        …
        last_fitness = self.get_fitness(self.population)
        best_individual = self.population[last_fitness.index(max(last_fitness))]

        best_fitness_list.append(max(last_fitness))
        avg_list.append(np.mean(last_fitness))

        self.log("best found is: {}\nwith {} acc. Avg: {}".format(best_individual,
best_fitness_list[-1], avg_list[-1]))

        return best_fitness_list, avg_list, best_individual
```

Concluding words

Genetic Algorithm is a technique that *finds individuals* that *maximize the fitness*

Useful to find x such that $f(x)$ *is maximal*

Can be used in a wide range of different *optimization problems*

dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

www.dcc.uchile.cl

/ DCCUCHILE