Design of Bioinspired Mathematical Algorithms (MA2015)

Fundamentals of Genetic Algorithms

José Carlos Ortiz Bayliss
jcobayliss@tec.mx

Contents

Tecnológico de Monterrey

The evolutionary computation metaphor

- During the four billion year history of the earth, biological life was born, perhaps as a result of a series of rare chance chemical and physical reactions of molecules.

The evolutionary computation metaphor

- During the four billion year history of the earth, biological life was born, perhaps as a result of a series of rare chance chemical and physical reactions of molecules.
- Over time, more and more complex forms of biological life evolved. Evolutionary algorithms are computer models based on genetics and evolution in biology.

The evolutionary computation metaphor

- During the four billion year history of the earth, biological life was born, perhaps as a result of a series of rare chance chemical and physical reactions of molecules.
- Over time, more and more complex forms of biological life evolved. Evolutionary algorithms are computer models based on genetics and evolution in biology.
- The basic elements of evolutionary algorithms are: selection of solutions based on how well they adapt to the environment, reproduction for crossover of genes, and mutation for random change of genes.

Tecnológico de Monterrey

From natural evolution to evolutionary computation

- **In natural evolution**: A given environment is filled with a population of individuals that strive for survival and reproduction. The fitness of these individuals (determined by the environment) relates to their chances of survival and multiplying.

From natural evolution to evolutionary computation

- **In natural evolution**: A given environment is filled with a population of individuals that strive for survival and reproduction. The fitness of these individuals (determined by the environment) relates to their chances of survival and multiplying.
- **In evolutionary computation**: We have a collection of candidate solutions. Their quality (how well they solve the problem) determines the chance that they will be kept and used as seeds for constructing further candidate solutions.

Natural evolution

As long as the environment is “nice” to the species, no significant changes are observed in the population.

Natural evolution

When the environment changes, it forces the species to adapt to survive. The fittest individuals are more like to survive and, as a consequence, to reproduce.

Natural evolution

Given the proper selective pressure from the environment, the characteristics of the fittest individuals will be spread over the population.

Natural evolution

Once again, changes to the environment modify the fitness of the individuals. For example, a feature that was once considered a benefit may now be an undesirable condition. If the change is too drastic, the species might get extinct.

Tecnológico de Monterrey

Natural evolution

Sometimes, random changes in the individuals turn out to be useful characteristics that increase their chances to survive and reproduce.

Natural evolution

The features that represent an advantage are now spread over the population.

Why do we need evolutionary computation?

- Developing automated problem solvers (algorithms) is one of the central themes of mathematics and computer science.

Why do we need evolutionary computation?

- Developing automated problem solvers (algorithms) is one of the central themes of mathematics and computer science.
- Similarly to engineering, where looking at nature's solutions has always been a source of inspiration, copying "natural problem solvers" is a stream within these disciplines.

Why do we need evolutionary computation?

- Developing automated problem solvers (algorithms) is one of the central themes of mathematics and computer science.
- Similarly to engineering, where looking at nature's solutions has always been a source of inspiration, copying "natural problem solvers" is a stream within these disciplines.
- There is a need for algorithms that are applicable to a wide range of problems, do not need much tailoring for specific problems, and deliver good (not necessarily optimal) solutions within acceptable time.

Genetic algorithms

- A genetic algorithm is a stochastic search method based on the mechanics of natural selection and the Darwinian idea of survival according to fitness.

Genetic algorithms

- A genetic algorithm is a stochastic search method based on the mechanics of natural selection and the Darwinian idea of survival according to fitness.
- In a genetic algorithm, the possible solutions to a problem are strings in a reduced alphabet (usually binary) that look like real life chromosomes from the individuals.

Genetic algorithms

- A genetic algorithm is a stochastic search method based on the mechanics of natural selection and the Darwinian idea of survival according to fitness.
- In a genetic algorithm, the possible solutions to a problem are strings in a reduced alphabet (usually binary) that look like real life chromosomes from the individuals.
- A genetic algorithm evolves a population of these individuals applying genetic operators like selection, crossover, and mutation.

The canonical genetic algorithm

```
procedure GENETICALGORITHM(n, p_c, p_m)
    population ← INITIALIZE(n)
    EVALUATE(population)
    do
        nextPopulation ← INITIALIZE(0)
        for i = 0 to n / 2 do
            parents ← SELECT(population, 2)
            offspring ← COMBINE(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← MUTATE(population[i], p_m)
        end for
        EVALUATE(population)
        SORT(population)
    while stopping condition is met
    return population[0]
end procedure
```

The canonical genetic algorithm - Initialization

```
procedure GENETICALGORITHM(n, p_c, p_m)
    population ← INITIALIZE(n)
    EVALUATE(population)
    do
        nextPopulation ← INITIALIZE(0)
        for i = 0 to n / 2 do
            parents ← SELECT(population, 2)
            offspring ← COMBINE(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← MUTATE(population[i], p_m)
        end for
        EVALUATE(population)
        SORT(population)
    while stopping condition is met
    return population[0]
end procedure
```

- The role of the population is to keep the representation of possible solutions. Then, a population is a set of genotypes.

Tecnológico de Monterrey

The canonical genetic algorithm - Initialization

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- The role of the population is to keep the representation of possible solutions. Then, a population is a set of genotypes.
- Initialization deals with how the first population is created.

The canonical genetic algorithm - Initialization

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- The role of the population is to keep the representation of possible solutions. Then, a population is a set of genotypes.
- Initialization deals with how the first population is created.
- Most of the times, the first population is seeded by randomly generated individuals.

The canonical genetic algorithm - Initialization

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- The role of the population is to keep the representation of possible solutions. Then, a population is a set of genotypes.
- Initialization deals with how the first population is created.
- Most of the times, the first population is seeded by randomly generated individuals.
- Sometimes, problem-specific heuristics can be used in this step, to create an initial population with higher fitness.

The canonical genetic algorithm - Fitness evaluation

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- The evaluation function is a way to represent the requirements the population should adapt to (it defines what "improvement" means).

The canonical genetic algorithm - Fitness evaluation

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- The evaluation function is a way to represent the requirements the population should adapt to (it defines what "improvement" means).
- Technically, it is a function or procedure that assigns a quality measure to genotypes.

The canonical genetic algorithm - Fitness evaluation

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- The evaluation function is a way to represent the requirements the population should adapt to (it defines what "improvement" means).
- Technically, it is a function or procedure that assigns a quality measure to genotypes.
- In evolutionary algorithms, we usually call it "fitness function" (although it might cause a counterintuitive terminology if the original problem requires minimisation).

Tecnológico de Monterrey

The canonical genetic algorithm - Selection

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- Selection discriminates individuals based on their quality, and in particular, to allow the better individuals to become parents of the next generation.

The canonical genetic algorithm - Selection

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- Selection discriminates individuals based on their quality, and in particular, to allow the better individuals to become parents of the next generation.
- Selection is responsible for pushing quality improvements.

The canonical genetic algorithm - Selection

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- Selection discriminates individuals based on their quality, and in particular, to allow the better individuals to become parents of the next generation.
- Selection is responsible for pushing quality improvements.
- Selection is typically probabilistic. Thus, high-quality individuals have more chances of becoming parents than those with low quality.

Tecnológico de Monterrey

The canonical genetic algorithm - Crossover

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- Crossover merges information from two parent genotypes into one or two offspring genotypes.

The canonical genetic algorithm - Crossover

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- Crossover merges information from two parent genotypes into one or two offspring genotypes.
- The rationale is simple: by mating two individuals with different but desirable features, we can produce an offspring that combines both of those features.

The canonical genetic algorithm - Crossover

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- Crossover merges information from two parent genotypes into one or two offspring genotypes.
- The rationale is simple: by mating two individuals with different but desirable features, we can produce an offspring that combines both of those features.
- Recombination is a stochastic operator: the choices of what parts of each parent are combined, and how this is done, depend on random choices.

The canonical genetic algorithm - Mutation

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- Mutation is applied to one genotype and delivers a (slightly) modified mutant, the offspring.

The canonical genetic algorithm - Mutation

```
procedure GENETICALGORITHM(n, p_c, p_m)
    population ← INITIALIZE(n)
    EVALUATE(population)
    do
        nextPopulation ← INITIALIZE(0)
        for i = 0 to n / 2 do
            parents ← SELECT(population, 2)
            offspring ← COMBINE(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← MUTATE(population[i], p_m)
        end for
        EVALUATE(population)
        SORT(population)
    while stopping condition is met
    return population[0]
end procedure
```

- Mutation is applied to one genotype and delivers a (slightly) modified mutant, the offspring.
- A mutation operator is always stochastic: its output (the offspring) depends on the outcomes of a series of random choices. In general, mutationis supposed to cause a random, unbiased change.

Tecnológico de Monterrey

The canonical genetic algorithm - Mutation

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- Mutation is applied to one genotype and delivers a (slightly) modified mutant, the offspring.
- A mutation operator is always stochastic: its output (the offspring) depends on the outcomes of a series of random choices. In general, mutationis supposed to cause a random, unbiased change.
- Mutation should guarantee that the space is connected. In other words, if we had infinite time we could reach the optimal solution by just using mutation (but we do not have infinite time, do we?).

The canonical genetic algorithm - Mutation

```
procedure GENETICALGORITHM(n, p_c, p_m)
    population ← INITIALIZE(n)
    EVALUATE(population)
    do
        nextPopulation ← INITIALIZE(0)
        for i = 0 to n / 2 do
            parents ← SELECT(population, 2)
            offspring ← COMBINE(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← MUTATE(population[i], p_m)
        end for
        EVALUATE(population)
        SORT(population)
    while stopping condition is met
    return population[0]
end procedure
```

- Mutation is applied to one genotype and delivers a (slightly) modified mutant, the offspring.
- A mutation operator is always stochastic: its output (the offspring) depends on the outcomes of a series of random choices. In general, mutationis supposed to cause a random, unbiased change.
- Mutation should guarantee that the space is connected. In other words, if we had infinite time we could reach the optimal solution by just using mutation (but we do not have infinite time, do we?).
- The role of mutation has historically been different in various evolutionary algorithms.

Tecnológico de Monterrey

The canonical genetic algorithm - Stopping condition

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- A threshold fitness value has been reached.

The canonical genetic algorithm - Stopping condition

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- A threshold fitness value has been reached.
- The maximally allowed CPU time elapses.

The canonical genetic algorithm - Stopping condition

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- A threshold fitness value has been reached.
- The maximally allowed CPU time elapses.
- The total number of fitness evaluations reaches a given limit.

The canonical genetic algorithm - Stopping condition

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- A threshold fitness value has been reached.
- The maximally allowed CPU time elapses.
- The total number of fitness evaluations reaches a given limit.
- The fitness improvement remains under a threshold value for a given period of time (i.e, for a number of generations or fitness evaluations).

The canonical genetic algorithm - Stopping condition

```
procedure GeneticAlgorithm(n, p_c, p_m)
    population ← Initialize(n)
    evaluate(population)
    do
        nextPopulation ← Initialize(0)
        for i = 0 to n / 2 do
            parents ← Select(population, 2)
            offspring ← combine(parents, p_c)
            nextPopulation ← nextPopulation ⋃ offspring
        end for
        population ← nextPopulation
        for i = 0 to n do
            population[i] ← mutate(population[i], p_m)
        end for
        evaluate(population)
        sort(population)
    while stopping condition is met
    return population[0]
end procedure
```

- A threshold fitness value has been reached.
- The maximally allowed CPU time elapses.
- The total number of fitness evaluations reaches a given limit.
- The fitness improvement remains under a threshold value for a given period of time (i.e, for a number of generations or fitness evaluations).
- The population diversity drops under a given threshold.

Representation

- We need a way to link the "real world" to the "evolutionary algorithm world". In other words, to set up a bridge between the original problem context and the space where evolution takes place.

Representation

- We need a way to link the "real world" to the "evolutionary algorithm world". In other words, to set up a bridge between the original problem context and the space where evolution takes place.
- The representation of each solution for an evolutionary algorithm is up to us. Although string representation of a solution is common, other forms of representation may be more convenient for other problems.

Tecnológico de Monterrey

Representation

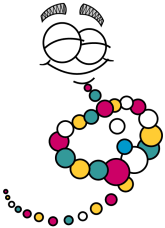
- We need a way to link the “real world” to the “evolutionary algorithm world”. In other words, to set up a bridge between the original problem context and the space where evolution takes place.
- The representation of each solution for an evolutionary algorithm is up to us. Although string representation of a solution is common, other forms of representation may be more convenient for other problems.
- In practice, some genetic operators only work for specific representations.

Tecnológico de Monterrey

Genotype vs. Phenotype

Genotype

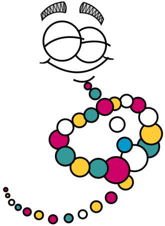
The set of genes that an individual carries.



Genotype vs. Phenotype

Genotype

The set of genes that an individual carries.



Phenotype

All of its observable characteristics.

Converting from genotype to phenotype

Genotype	Phenotype
`0110`	6

Converting from genotype to phenotype

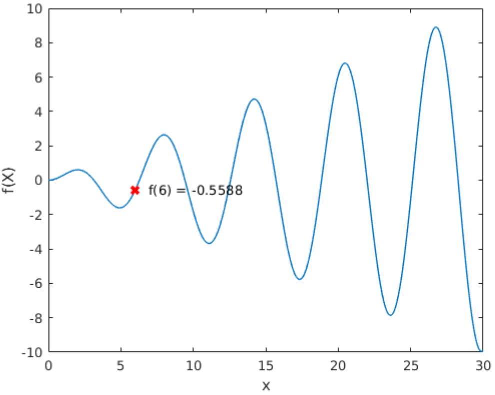
Genotype	Phenotype
`0110`	6
`0110`	4

Converting from genotype to phenotype

Genotype	Phenotype
0110	6
0110	4

0110

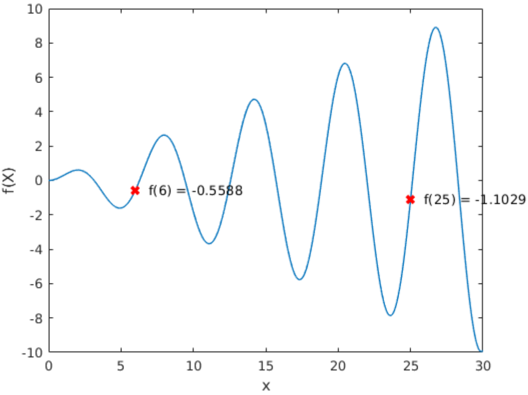
Converting from genotype to phenotype

Genotype	Phenotype	Evaluation
00110	6	

Converting from genotype to phenotype

Genotype	Phenotype
00110	6
11001	25

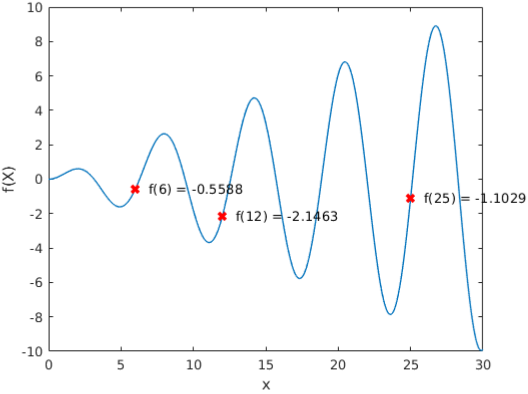
Evaluation



Converting from genotype to phenotype

Genotype	Phenotype
00110	6
11001	25
01100	12

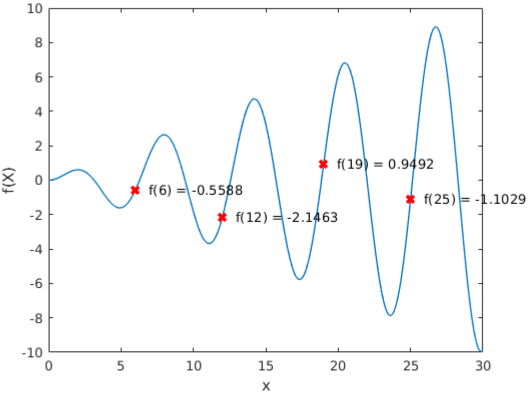
Evaluation



Converting from genotype to phenotype

Genotype	Phenotype
00110	6
11001	25
01100	12
10011	19

Evaluation



Tecnológico de Monterrey

Binary representation

- This is one of the earliest representations, and historically it is the most used one.

Binary representation

- This is one of the earliest representations, and historically it is the most used one.
- For a particular application we have to decide how long the string should be, and how we will interpret it to produce a phenotype.

Binary representation

- This is one of the earliest representations, and historically it is the most used one.
- For a particular application we have to decide how long the string should be, and how we will interpret it to produce a phenotype.
- We must be sure that the encoding allows that all possible bit strings denote a valid solution to the given problem and that, vice versa, all possible solutions can be represented.

Tecnológico de Monterrey

Crossover operators for binary representation

- One-point crossover.

Crossover operators for binary representation

- One-point crossover.
- Two-point crossover.

Crossover operators for binary representation

- One-point crossover.
- Two-point crossover.
- n-point crossover.

Crossover operators for binary representation

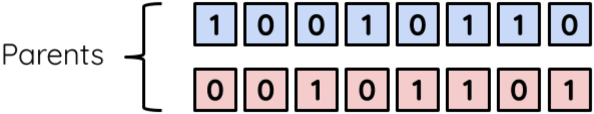
- One-point crossover.
- Two-point crossover.
- n-point crossover.
- Uniform crossover.

One-point crossover

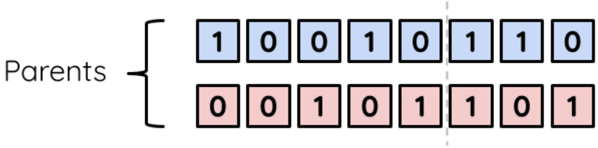
Given two parents:

1. Select a random cross point.
2. Interchange positions after the cross point.

One-point crossover

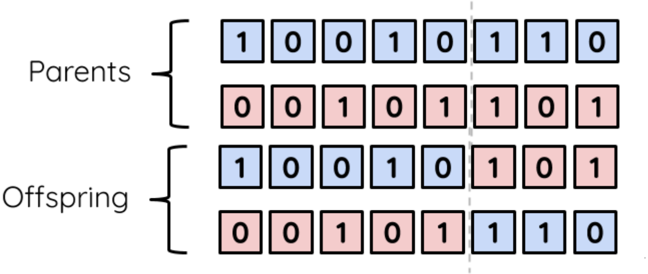


One-point crossover



Tecnológico de Monterrey

One-point crossover



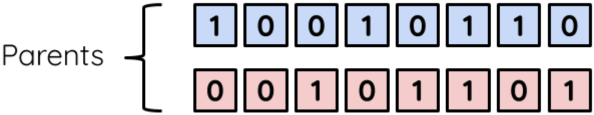
Tecnológico de Monterrey

Two point crossover

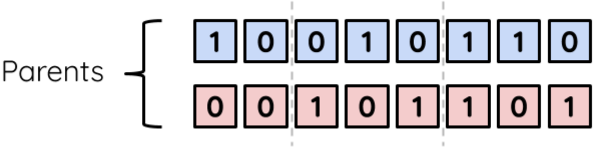
Given two parents:

1. Select two random cross points.
2. Interchange positions between the cross points.

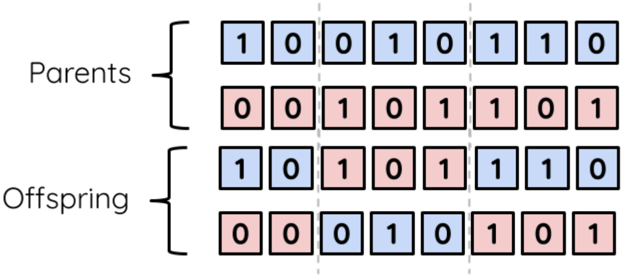
Two-point crossover



Two-point crossover



Two-point crossover

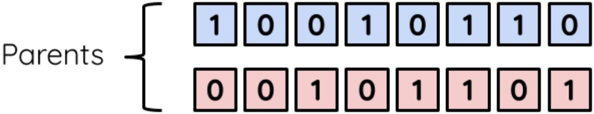


n-point crossover

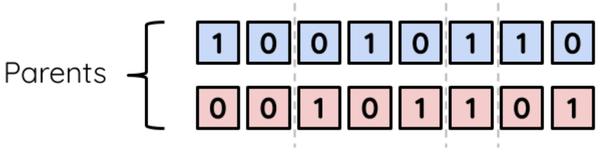
Given two parents:

1. Select n random cross points.
2. Interchange positions between the cross points.

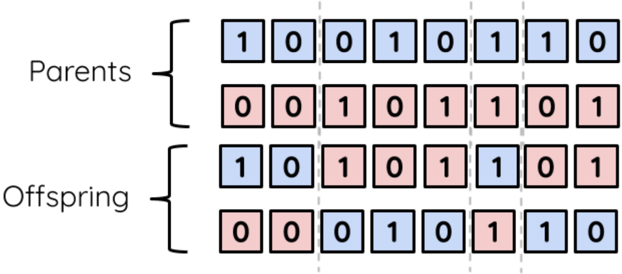
n-point crossover



n-point crossover



n-point crossover



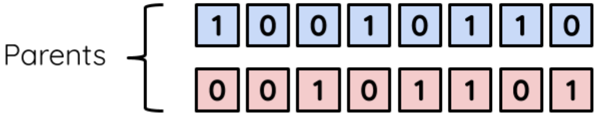
Tecnológico de Monterrey

Uniform crossover

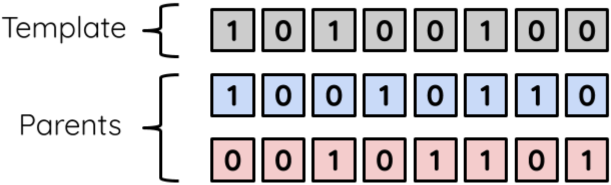
Given two parents:

1. A random binary template is created with probability p of generating a zero.
2. Positions are interchanged according to the template.

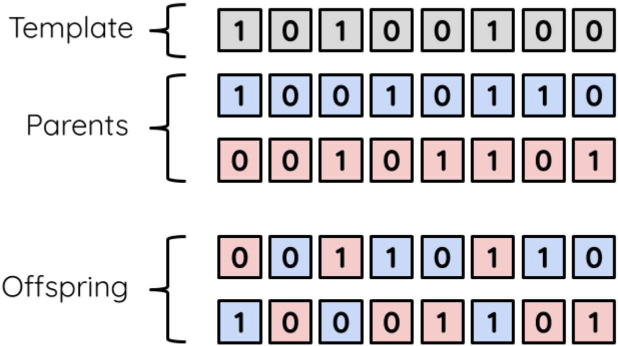
Uniform crossover



Uniform crossover



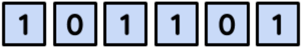
Uniform crossover



Tecnológico de Monterrey

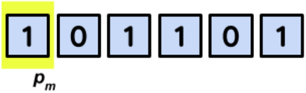
Mutation operator for binary representation

With probability p_m, it flips the value to each gene.



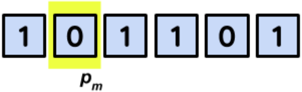
Mutation operator for binary representation

With probability p_m, it flips the value to each gene.



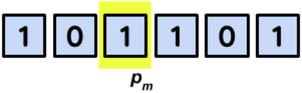
Mutation operator for binary representation

With probability p_m, it flips the value to each gene.



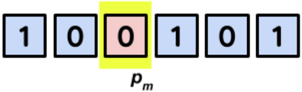
Mutation operator for binary representation

With probability p_m, it flips the value to each gene.



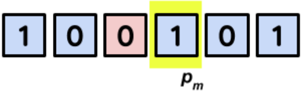
Mutation operator for binary representation

With probability p_m, it flips the value to each gene.



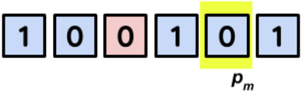
Mutation operator for binary representation

With probability p_m, it flips the value to each gene.



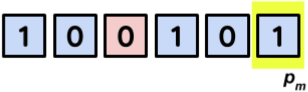
Mutation operator for binary representation

With probability p_m, it flips the value to each gene.



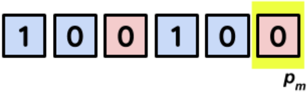
Mutation operator for binary representation

With probability p_m, it flips the value to each gene.



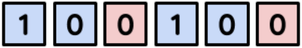
Mutation operator for binary representation

With probability p_m, it flips the value to each gene.



Mutation operator for binary representation

With probability p_m, it flips the value to each gene.

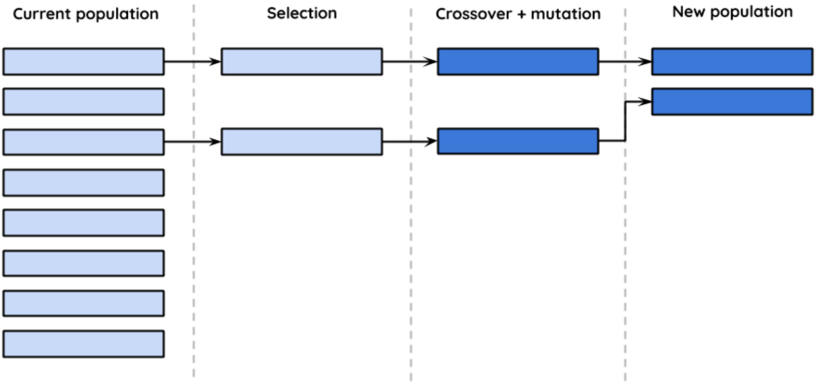


Generational selection

The whole population is substituted every generation (parents die and do not coexist with their offspring).

Generational selection

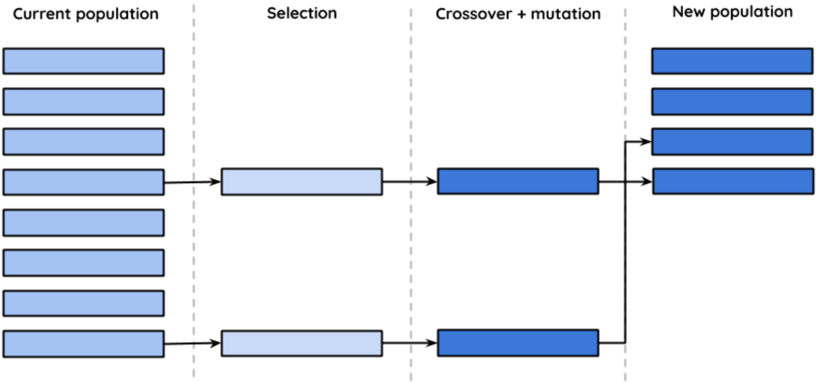
The whole population is substituted every generation (parents die and do not coexist with their offspring).



Tecnológico de Monterrey

Generational selection

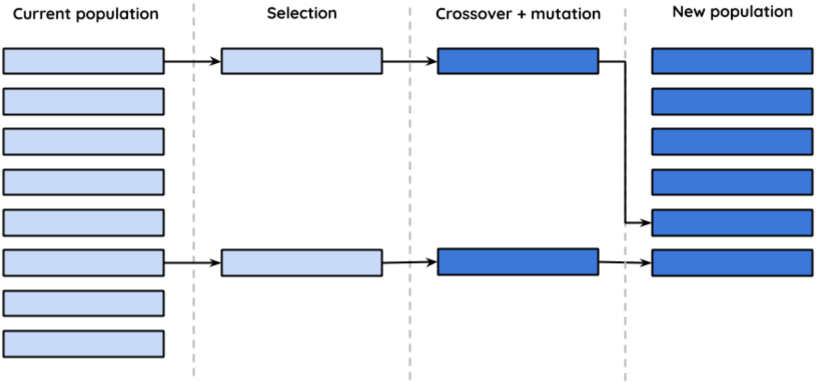
The whole population is substituted every generation (parents die and do not coexist with their offspring).



Tecnológico de Monterrey

Generational selection

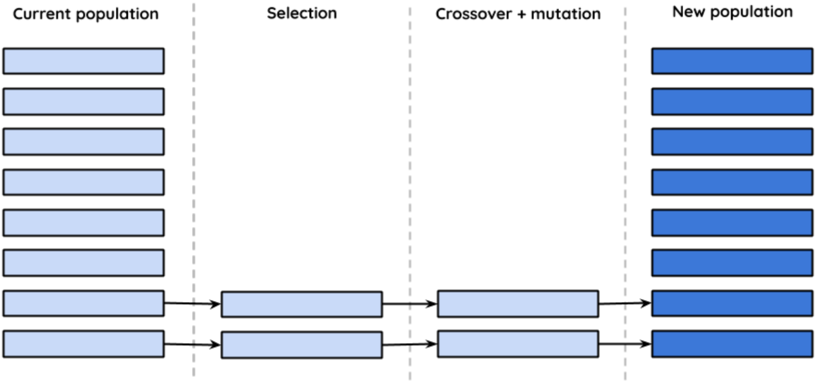
The whole population is substituted every generation (parents die and do not coexist with their offspring).



Tecnológico de Monterrey

Generational selection

The whole population is substituted every generation (parents die and do not coexist with their offspring).



Tournament selection

- Every time a parent is to be selected, m individuals are randomly selected.

Tournament selection

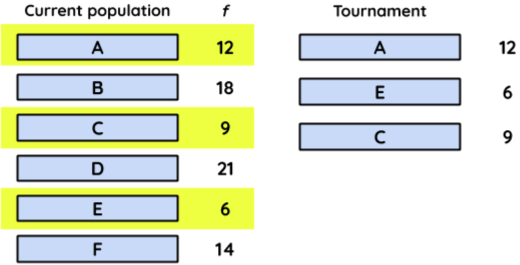
- Every time a parent is to be selected, m individuals are randomly selected.
- The individual with the best fitness among those m individuals will be chosen as the a parent.

Tournament selection

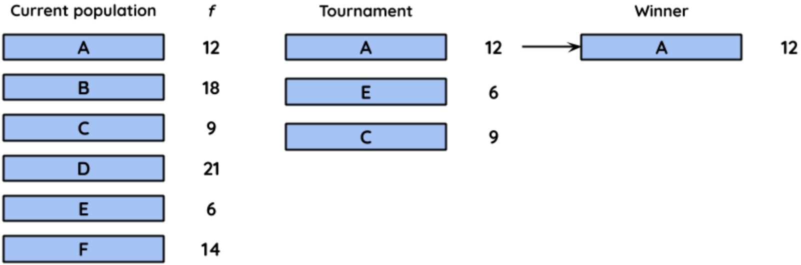
Current population	f
A	12
B	18
C	9
D	21
E	6
F	14

Tecnológico de Monterrey

Tournament selection



Tournament selection



Bibliography

[Eiben and Smith, 2015] Eiben, A. E. and Smith, J. E. (2015).
Introduction to Evolutionary Computing.
Springer Publishing Company, Incorporated, 2nd edition.

[Munakata, 2008] Munakata, T. (2008).
Fundamentals of the New Artificial Intelligence: neural, evolutionary, fuzzy and more.
Springer, London.

Tecnológico de Monterrey