Evolutionary Computation in Python

Cosijopii García García[1]

[1]Intelligent systems and software development Lab, Universidad del Istmo Campus Ixtepec. contacto@cosijopii.com
https://www.cosijopii.com/

November 8, 2024

Outline

Section 1

Introduction to Genetic Algorithms

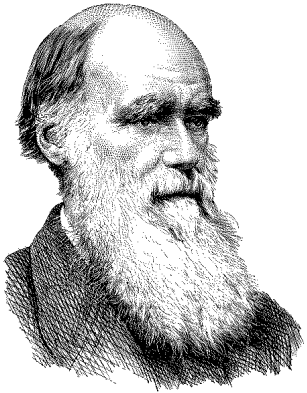
Section 1

Introduction to Genetic Algorithms

1.1 History

History

- Evolutionary computing is a research area within computer science. As the name suggests, it is a special flavour of computing, which draws inspiration from the process of **natural evolution**[1].
- The power of evolution in nature is evident in the diverse species that make up our world, each tailored to survive well in its own niche.



Charles Robert Darwin

[1] Eiben and Smith, *Introduction to Evolutionary Computing*, 2015.

History

- As early as 1948, Turing proposed "genetical or evolutionary search"
- Bremermann conducted computer experiments on "optimization through evolution and recombination" in 1962 [2].
- In the 1960s three different implementations of the basic idea were developed in different places[3].

[3] Eiben and Smith, *Introduction to Evolutionary Computing*, 2015.

History

- Fogel, Owens, and Walsh introduced **evolutionary programming**
- Holland called his method a **genetic algorithm**
- Rechenberg and Schwefel invented **evolution strategies**

- After 15 years that methods conforms the **Evolutionary Computation**[4](EC)
- Finally in 90's a fourth stream following the general ideas emerged, **genetic programming** by Koza.

John Henry Holland

[4] Eiben and Smith, *Introduction to Evolutionary Computing*, 2015.

Section 1

Introduction to Genetic Algorithms

1.2 Genetic algorithm and their components

Genetics algorithms

- The genetic algorithm (GA) is one of the most popular evolutionary algorithms. Its research methodology is based on natural evolution, and was initially conceived by Holland as a way to study adaptive behavior[5].
- The canonical or simple genetic algorithm, has a binary representation, fitness proportionate selection, and low mutation probability[6].
- Over the years, new characteristics were developed, one of the most important is elitism, as well as different types of recombination and mutation

[5] Eiben and Smith, *Introduction to Evolutionary Computing*, 2015.

[6] Sastry, Goldberg, and Kendall, *Search Methodologies*, 2014.

Genetic algorithm

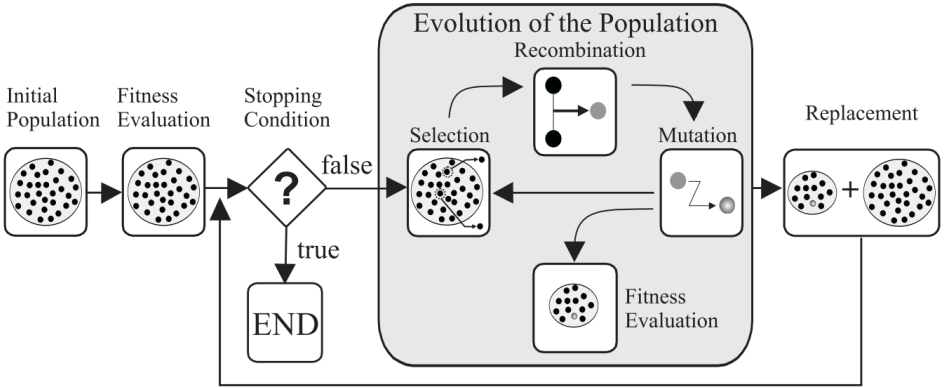


Figure 1: GA[7]

[7] Alba and Dorronsoro, *Cellular Genetic Algorithms*, 2008.

Genetics algorithms

- **Initialization**: The initial population of solutions is randomly generated across the search space.
- **Evaluation**: Once the population is created, the fitness value of every solution in the population is calculated.
- **Selection**: At the selection stage solutions chosen according to their fitness value. There are several forms of selection procedures
- **Recombination**: Information from two or more parents are combined to create a new possible better solution.
- **Mutation**: Locally and randomly modifies a solution. It involves one or more ways of adding small perturbations to an individual.
- **Replacement**: Offspring created by selection, recombination, and mutation replaces the original population by some criterion.

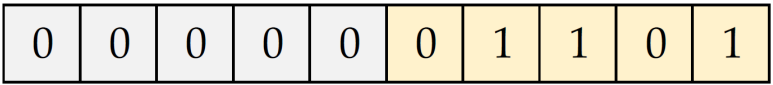
Representation

- There are different ways to represent the problems, depending on the type of representation that is used the evolutionary operators, the mutation and recombination or crossover can vary[8].
- Different types of representation exist in evolutionary algorithms and in genetics algorithms. The most famous representations are: **binary**, **integer**, and **real**.

[8] Brabazon, O'Neill, and McGarraghy, *Natural Computing Algorithms*, 2015.

Representation-Binary

- Binary representation was the first to be used, and historically many genetic algorithms (GA), used this representation regardless of the context of the problem to be solved, this representation is based on strings of bits that represent the genotype. This bit string represents how long the bit string will be, and how it will be mapped to the phenotype of the solution.



Representation-Integer

- Integer representation, it is mostly used when it is required to find optimal values for a set of variables in the domain of integer values. These values can be unrestricted or restricted to some finite set. For example, if we try to find a route in a square grid, and we are restricted to the set {0,1,2,3} which represents {North, East, South, West} in this case using an integer coding, is better than a binary, another example of this can be the representation of networks.

Representation-Real

- For many problems, genotype representation with real-valued is the most natural way of representation and current optimization applications use real-valued coding[9]. This occurs when variables values come from a continuous distribution rather than a discrete distribution. An example of this consists of physical quantities representation as length, width, height, or weight, some of these components can be real number values.

0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3

[9] Brabazon, O'Neill, and McGarraghy, *Natural Computing Algorithms*, 2015.

Evaluation

- The role of the evaluation function is to represent the requirements the population should adapt to meet. It forms the basis for selection, and so it facilitates improvements. More accurately, it defines what improvement means.
- The evaluation function is commonly called the **fitness** function in EC.
- Typically, this function is composed from the inverse representation (to create the corresponding phenotype) followed by a quality measure in the phenotype space.

Selection

- Selection is one of the main operators used in evolutionary algorithms. Its main objective is to find the best solutions in a population[10].
- Selection criteria determine selection pressure, which is the degree to which good **fitness** solutions are selected. If selection pressure is too **low**, information from good parents will be spread too slowly throughout the population, If selection pressure is too **high**, population will be stuck in a local optima.
- Selection techniques can be classified in two categories **Fitness proportionate** and **Ordinal selection**.

[10] Fogel, Bäck, and Michalewicz, *Evolutionary computation. Vol. 1, Basic algorithms and operators*, 2000.

Selection - Fitness proportionate

- **Roulette-wheel selection**: The principal idea of this method is to divide the candidates by their fitness. The greater the fitness of a solution, the more likely it is to be chosen.
- **Universal stochastic selection**: Is a slightly modified version of the roulette wheel instead of using a single selection point and turning the roulette wheel again and again until all needed individuals have been selected, we turn the wheel only once and use multiple selection points that are equally spaced around the wheel. This way, all the individuals are chosen at the same time[11].

[11] Wirsansky, *Hands-on genetic algorithms with Python*.

Selection - Fitness proportionate

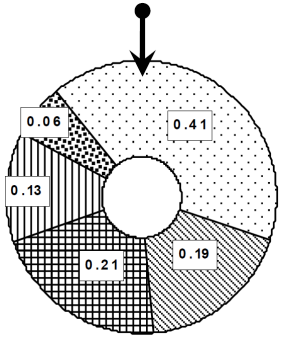


Figure 2: Roulette-wheel selection[12]

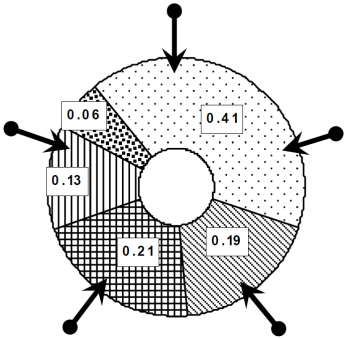


Figure 3: Universal stochastic selection

[12] Younes, Elkamel, and Areibi, "Genetic Algorithms in Chemical Engineering : A Tutorial", 2008.

Selection - Ordinal selection

- **Tournament selection**: p solutions are selected and enter into a tournament against each other. An individual with higher fitness in a group of p solutions wins the tournament and is selected as a parent. The most used tournament size is $p = 2$.
- **Truncation selection**: truncation selection, individuals are ordered according to their fitness value and top $(1/p)$ best ones are chosen to perform recombination.

Recombination

- Recombination is the process by which a new solution is created using information from two or more parents. It is considered one of the most important characteristics in evolutionary algorithms. A traditional way in which recombination operators perform "crossover" is by marking sub-segments in parents genomes to later assemble into a new individual[13].
- Recombination focuses on **exploration**, trying to search for promising new zones in search space.

[13] Eiben and Smith, *Introduction to Evolutionary Computing*, 2015.

Recombination

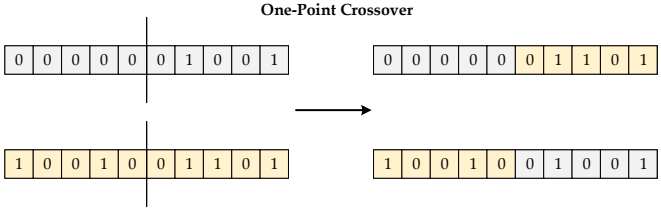


Figure 4: Caption

Recombination

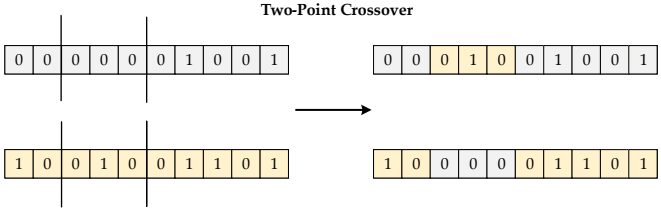


Figure 5: Caption

Mutation

- Mutation is a mechanism in which only one solution is involved. Solutions selected as parents or offspring solutions are mutated by slightly modifying their genetic information[14]
- The most used scheme is to perform a bitwise in the string-bit with a certain probability pm.
- Mutation focuses on **exploitation**, trying to find new solutions in the unexplored areas by recombination.

[14] Fogel, Bäck, and Michalewicz, *Evolutionary computation. Vol. 1, Basic algorithms and operators*, 2000.

Mutation

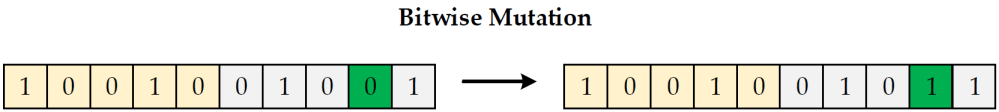


Figure 6: bitwise

Replacement

- Survivors selection mechanism is responsible for managing a reduction process in a EA's population from a set of μ parents and λ offspring to a set of μ individuals that form the next generation.
- There are many ways to do this, but a main one is to decide based on **individuals fitness**. Another technique is based on population's age, this method is used in the *SGA*, where each individual exists only one cycle and parents are discarded to be replaced by offspring. This criterion is known as a **generational population model**.

Replacement

- **Replace the worst**: In this replacement scheme μ worst parents are replaced. This can lead to a premature convergence, when solutions get stuck in a limited search space zone.
- **$(\mu + \lambda)$** : comes from evolutionary strategies. It refers to the case in which both sets of parents and offspring are ranked in a same set. Then top μ solutions are kept for the next generation. This strategy can be seen as a generalization of replacing the worst criterion.
- **Elitism**: Elitism is used to maintain the best solution(s) in the population. Thus if the best solution is chosen to be replaced, and any offspring is worse or equal. The offspring is discarded and the best individual is kept.

Genetic algorithm - resume

Algorithm 1: Simple or Canonical GA

1 $\mathbf{t} = 0$;
2 Initialize $\mathbf{Population}(\mathbf{t})$;
3 Evaluate $\mathbf{Population}(\mathbf{t})$;
4 **while** *Termination condition not satisfied* **do**
5 $\quad \mathbf{t} = \mathbf{t} + 1$;
6 $\quad$ Select $\mathbf{m}(\mathbf{t})$ Parents from $\mathbf{Population}(\mathbf{t} - \mathbf{1})$;
7 $\quad$ Recombine and mutate solutions in $\mathbf{m}(\mathbf{t})$;
8 $\quad$ Create offspring population $\mathbf{m'}(\mathbf{t})$;
9 $\quad$ Evaluate $\mathbf{m'}(\mathbf{t})$;
10 $\quad$ Select individuals for next generation;
11 **end while**

Section 2

Hands-On Genetic Algorithms

Section 2

Hands-On Genetic Algorithms

2.1 Single-objective optimization problem

Single-objective optimization problem

- x is a **decision variables** The vector **x** of **n** decision variables is represented by:

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]^T \tag{1}$$

- A **general single-objective optimization problem**[15] is defined as minimizing (or maximizing) $f(x)$ subject to $g_i(x) \leq 0, i = \{1, \ldots, m\}$, and $h_j(x) = 0, j = \{1, \ldots, p\} x \in \Omega$.

[15] Coello et al., *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2007.

Single-objective optimization problem

- **Constraints** are imposed by environment characteristics or resources and occur in most optimization problems. They are expressed in the form of mathematical equalities or inequalities, $h_j(x) = 0, j = \{1, \ldots, p\}$ and $g_i(x) \leq 0, i = \{1, \ldots, m\}$. If the number of equality constraints is greater than the number of decision variables, the problem is over constrained so there are not enough degrees of freedom for optimization[16]

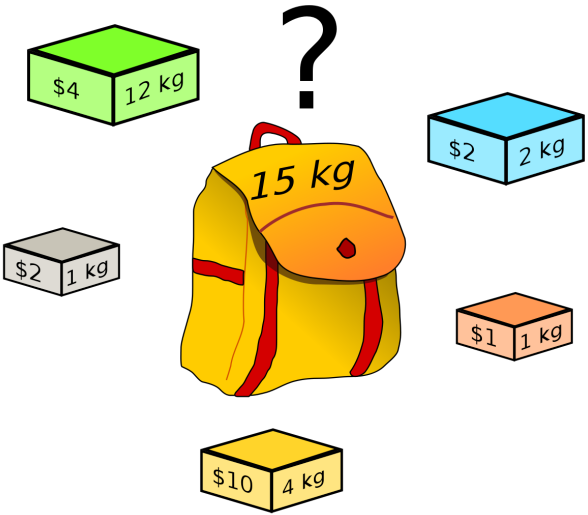
[16] Coello et al., *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2007.

Section 2

Hands-On Genetic Algorithms

2.2 Knapsack problem

The knapsack problem



0–1 knapsack problem

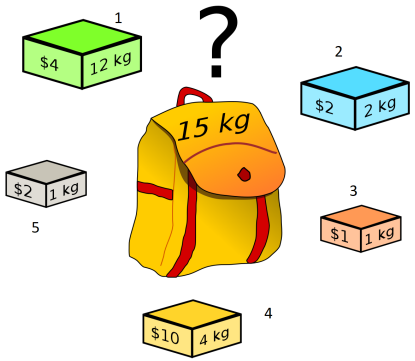
- We are given a set of n items, each of which has attached to it some value v_i, and some weight w_i. The task is to select a subset of those items that maximizes the sum of the values, while keeping the summed weight within some capacity C_{max}.
- The goal is to:

$$\text{Maximize : } \sum_{i=1}^{n} v_i \cdot x_i \tag{2}$$

$$\text{Subject to the constraints: } \sum_{i=1}^{n} w_i \cdot x_i \leq C_{max} \tag{3}$$

The knapsack problem

Input: Nonnegative integers $n, v_1, \ldots, v_n, w_1, \cdots, w_n$ and W.
Task: Find a subset $S \subseteq \{1, \cdots, n\}$ such that $\sum_{j \in S} w_j \leq C_{max}$ and $\sum_{j \in S} v_j$ is maximum.



Box	V	W	X
1	$v_1 = 4$	$w_1 = 12$	$x_1 = 0$
2	$v_2 = 2$	$w_2 = 2$	$x_2 = 1$
3	$v_3 = 1$	$w_3 = 1$	$x_3 = 1$
4	$v_4 = 10$	$w_4 = 4$	$x_4 = 1$
5	$v_5 = 2$	$w_5 = 1$	$x_5 = 1$

0–1 knapsack problem

- It is a natural idea to represent candidate solutions for this problem as binary strings of length n, where a 1 in a given position indicates that an item is included and a 0 that it is omitted.
- When solving a problem it is also important to identify the information that is needed, in this case, it seems clear the objective function but also need the values(v_i) of each item as well as its weight(w_i), in addition to this we need to define a maximum capacity C_{max}.

Hands-on

Open code in **Colab**

Section 2

Hands-On Genetic Algorithms

2.3 The Multiobjective Optimization Problem

Multiobjective optimization

- **The Multiobjective Optimization Problem** can then be defined (in words) as the problem of finding[17]:
- *"A. **vector of decision variables** which satisfies **constraints** and optimizes a **vector function** whose elements represent the objective functions. These functions form a mathematical description of performance criteria which are usually in **conflict** with each other. Hence, the term "optimize" means finding such a solution which would give the values of all the objective functions acceptable to the decision maker.*

[17] Coello et al., *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2007.

Multiobjective optimization

$$\left.\begin{aligned} \text{Minimize/Maximize } \mathbf{f}_m(\mathbf{x}), m = 1, 2, \ldots, k; \\ \text{subject to } g_j(\mathbf{x}) \geq 0, j = 1, 2, \ldots, m; \\ h_k(\mathbf{x}) = 0, k = 1, 2, \ldots, p; \\ x_i^{(L)} \leq x_i \leq x_i^{(U)}, i = 1, 2, \ldots, t; \end{aligned}\right\} \quad (4)$$

- with k objectives, m and p are the number of inequality and equality constraints. A solution $\mathbf{x} \in \mathbf{R}^n$ is a vector of n decision variables: $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, which satisfy all constraints and variable bounds[18].

[18] Coello et al., *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2007.

Multiobjective optimization

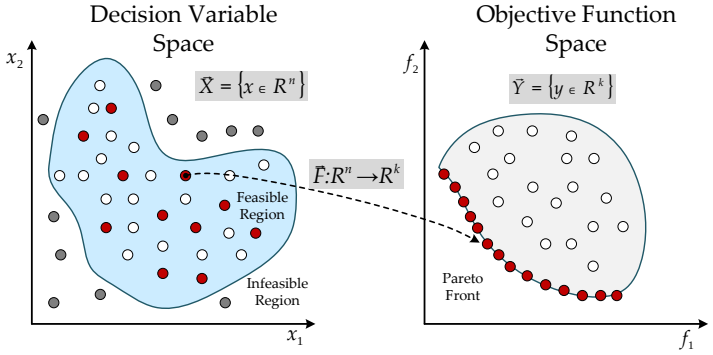


Figure 7: Mapping of decision variables space to the objective function space. Feasible solutions and zone are marked in blue. In the decision variable space, the Pareto optimal set is marked with red solutions and its mapping to the objective function space creates the Pareto front.

Multiobjective optimization

- **Pareto dominance**[19]: A vector $\mathbf{u} = (u_1, u_2, \ldots, u_k)$ is said to dominate another vector $\mathbf{v} = (v_1, v_2, \ldots, v_k)$ (denoted by $\mathbf{u} \preceq \mathbf{v}$) if and only if $\mathbf{u}$ is partially less than $\mathbf{v}$, this is specified as follows:
  $\forall i \in \{1, \ldots k\}, u_i \leq v_i$ and $\exists i \in \{1, \ldots k\} : u_i) < v_i$.
- **Pareto Optimal Set**: For a given MOP and $F(x)$, the POS $\mathcal{P}^*$ is determined by:

$$\mathcal{P}^* = \{\mathbf{x} \in \Omega \mid \neg\exists \mathbf{x}' \in \Omega \; F(\mathbf{x}') \preceq F(\mathbf{x})\} \tag{5}$$

- **Pareto Front** :
  For a given MOP, $F(X)$ and POS, $\mathcal{P}^*$, the Pareto Front $\mathcal{PF}^*$ can be expressed as:

$$\mathcal{PF}^* = \{\mathbf{u} = F(\mathbf{x}) \mid \mathbf{x} \in \mathcal{P}^*\} \tag{6}$$

[19] Coello et al., *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2007.

Resolving a MOP in Pymoo

- **input:** x
- **Task:** Find the vector x that minimize the follow:

$$\begin{cases} f_1(\mathbf{x}) = 100(x_1^2 + x_2^2), \\ f_2(\mathbf{x}) = (x_1 - 5)^2 + x_2^2, \\ \textbf{Subject to:} \\ g_1(\mathbf{x}) \equiv 2(x_1 - 0.1)(x_1 - 0.9)/0.18 \leq 0, \\ g_2(\mathbf{x}) \equiv -20(x_1 - 0.4)(x_1 - 0.6)/4.8 \leq 0, \\ -2 \leq x_1 \leq 2, \\ -2 \leq x_2 \leq 2 \end{cases} \tag{7}$$

Hands-on

Open code in **Colab**

Questions?

Section 3

References

References I

Alba, Enrique and Bernabé Dorronsoro. *Cellular Genetic Algorithms*. Vol. 42. Operations Research/Computer Science Interfaces Series. Boston, MA: Springer US, 2008. ISBN: 978-0-387-77609-5. DOI: 10.1007/978-0-387-77610-1. URL: http://link.springer.com/10.1007/978-0-387-77610-1.

Brabazon, Anthony, Michael O'Neill, and Seán McGarraghy. *Natural Computing Algorithms*. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. ISBN: 978-3-662-43630-1. DOI: 10.1007/978-3-662-43631-8. URL: www.springer.com/series/http://link.springer.com/10.1007/978-3-662-43631-8.

Coello, Carlos A Coello et al. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation Series. Boston, MA: Springer US, 2007, p. 315. ISBN: 978-0-387-33254-3. DOI: 10.1007/978-0-387-36797-2. URL: http://link.springer.com/10.1007/978-0-387-36797http://link.springer.com/10.1007/978-0-387-36797-2.

Eiben, A.E. and J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. ISBN: 978-3-662-44873-1. DOI: 10.1007/978-3-662-44874-8. URL: http://link.springer.com/10.1007/978-3-662-44874-8.

References II

Fogel, David B., Thomas Bäck, and Zbigniew. Michalewicz. *Evolutionary computation. Vol. 1, Basic algorithms and operators*. Institute of Physics Pub, 2000, p. 384. ISBN: 0750306645.

Sastry, Sastry, David E. Goldberg, and Graham Kendall. *Search Methodologies*. Ed. by Edmund K. Burke and Graham Kendall. Boston, MA: Springer US, 2014. Chap. 4, p. 716. ISBN: 978-1-4614-6939-1. DOI: 10.1007/978-1-4614-6940-7. arXiv: arXiv:1011.1669v3. URL: https://link.springer.com/content/pdf/10.1007%2F978-1-4614-6940-7.pdfhttp://link.springer.com/10.1007/978-1-4614-6940-7.

Wirsansky, Eyal. *Hands-on genetic algorithms with Python*. ISBN: 9781838557744.

Younes, Abdunnaser, Ali Elkamel, and Shawki Areibi. "Genetic Algorithms in Chemical Engineering : A Tutorial". In: 2008.