Evolutionary Computation in Python: Open Source Frameworks

Alicia Morales Reyes[1] Cosijopii García García[1] José Alejandro Cornejo Acosta[1]

[1]Computer Science Lab, Instituto Nacional de Astrofísica, Óptica y Electrónica
{a.morales,cosijopii, alexcornejo}@inaoep.mx

September 26, 2022

Outline

Section 1

Introduction to Genetic Algorithms

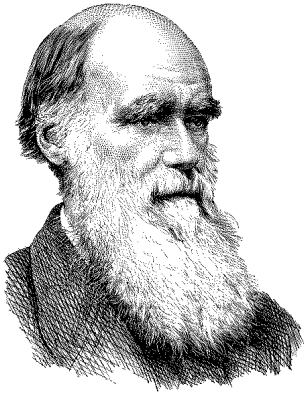
Section 1

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

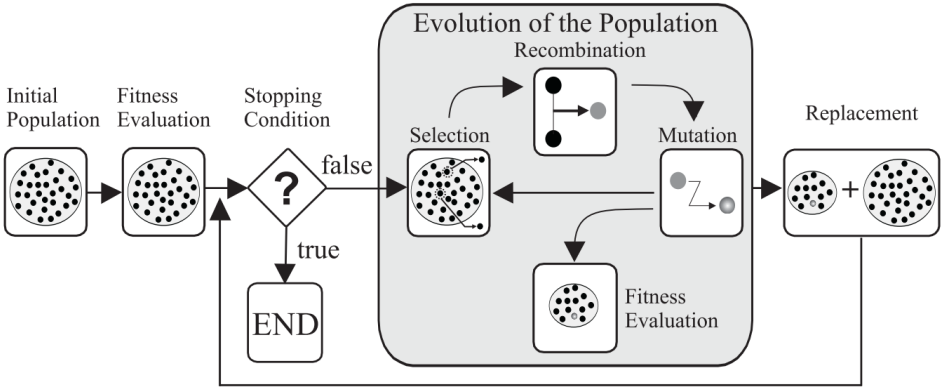


Figure 1: GA[7]

[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.

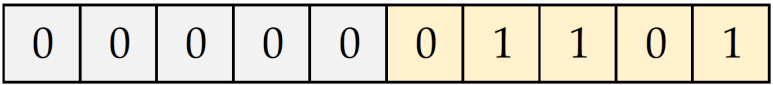
Representation

- 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

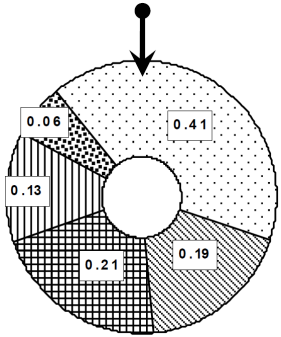


Figure 2: Roulette-wheel selection[12]

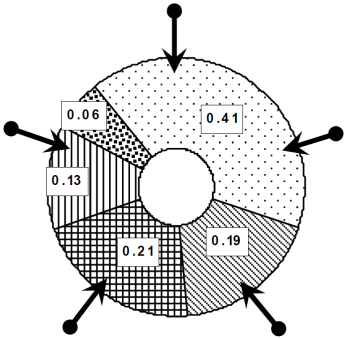


Figure 3: Universal stochastic selection

[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

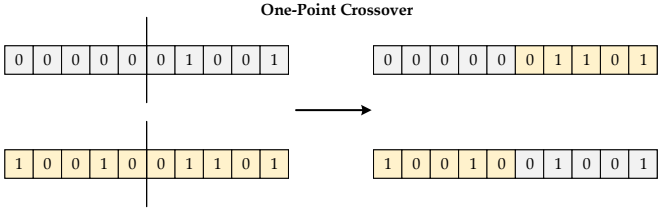


Figure 4: Caption

Recombination

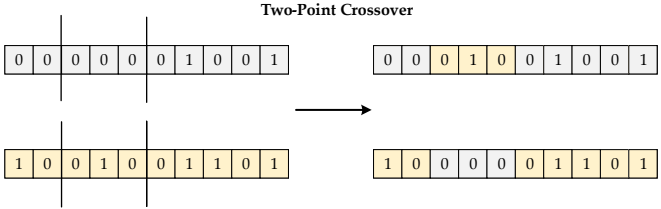


Figure 5: Caption

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

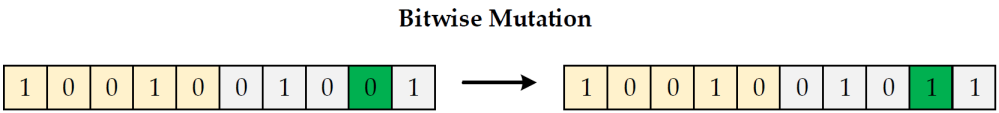


Figure 6: bitwise

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 1

Introduction to Genetic Algorithms

1.3 Beyond genetics algorithms

Differential evolution

- The differential evolution (DE) algorithm is one of the most popular evolutionary algorithms. It was proposed in 1997[15]
- DE differs from GA in the reproduction mechanism and shares most stages of evolutionary algorithms.
-

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{if } (randb(j) \leq CR) \text{ or } j = rnbr(i) \\ x_{ji,G} & \text{if } (randb(j) > CR) \text{ or } j \neq rnbr(i) \end{cases} \quad (1)$$

[15] Storn and Price, *Differential Evolution-A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*, 1997.

Differential evolution

Algorithm 2: Differential Evolution Algorithm

Create initial population $x_{i,G}, i = 1, 2, \ldots, NP$;
Evaluate fitness of each solution;
while *Termination condition not satisfied* **do**
    for *each vector* $x_{i,G}$ **do**
        Select $x_{r_1,G}, x_{r_2,G}, x_{r_3,G}$ from population where $r_1, r_2, r_3 \neq i$;
        Apply $x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G})$ and create mutant vector $v_{i,G+1}$;
        Combine target vector $x_{i,G}$ and mutant vector $v_{i,G+1}$ to produce trial vector $u_{i,G+1}$;
        Trial vector fitness evaluation;
        if *trial vector has higher fitness than target vector* **then**
            Replace target vector with trial vector;
        end if
    end for
end while

Genetic programming

- Genetic programming is a relatively young member of the evolutionary algorithm family. It differs from other EA strands in its application area as well as the particular representation (using trees as chromosomes).
- GP could instead be positioned in machine learning. In terms of the different problem types.

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

Figure 7: Sketch of GP[16]

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

Evolutionary Strategies

- The earliest ES's were simple two-membered algorithms denoted (1+1) ES, working in a vector space. An offspring is generated by the addition of a random number independently to each to the elements of the parent vector and accepted if fitter.
- In the 1970s the concept of multi-membered evolution strategies was introduced, with the naming convention based on μ individuals in the population and λ offspring generated in one cycle.
- The resulting $(\mu + \lambda)$and (μ, λ) ES's gave rise to the possibility of more sophisticated forms of step-size control, and led to the development of a very useful feature in evolutionary computing: self-adaptation of strategy parameters[17].

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

Evolutionary Strategies

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	Deterministic elitist replacement by (μ, λ) or $(\mu + \lambda)$
Speciality	Self-adaptation of mutation step sizes

Figure 8: Sketch of ES[18]

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

Particle swarm optimization

- Particle swarm optimization (PSO) was launched in 1995, when Kennedy and Eberhart published their seminal paper about a "concept for the optimization of nonlinear functions using particle swarm methodology"[19].
- Similarly to DE, the distinguishing feature of PSO is a twist to the usual reproduction operators in EC: PSO does **not use crossover** and its mutation is defined through a vector addition.

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

$$\overline{x}_i' = \overline{x} + \overline{v}_i' \tag{2}$$

$$\overline{v}_i' = w \cdot \overline{v}_i + \phi_1 U_1 \cdot (\overline{b}_i - \overline{x}_i) + \phi_2 U_2 \cdot (\overline{c} - \overline{x}_i) \tag{3}$$

$$\overline{b}_i' = \begin{cases} \overline{x}_i' & \text{if} f(\overline{x}_i') < f(\overline{b}_i) \\ \overline{b}_i & otherwise \end{cases} \tag{4}$$

where w and ϕ_i are the weights (w is called the inertia, ϕ_1 is the learning rate for the personal influence and $\phi_2$is the learning rate for the social influence), while U_1 and U_2 are randomizer matrices that multiply every coordinate of $(\overline{b}_i - \overline{x}_i)$ and $(\overline{c} - \overline{x}_i)$ by a number drawn from the uniform distribution.

Section 2

Open source and alternative for Genetic Algorithms

Section 2

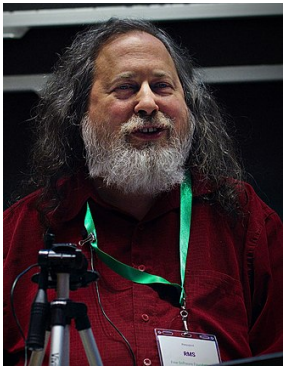
Open source and alternative for Genetic Algorithms

2.1 What is open source?

Free software

According to the **Free Software Foundation (FSF)**, a *software is free* if the program's users have the four essential freedoms:

1. Run the software
2. Study, change and improve
3. Copy
4. Distribute

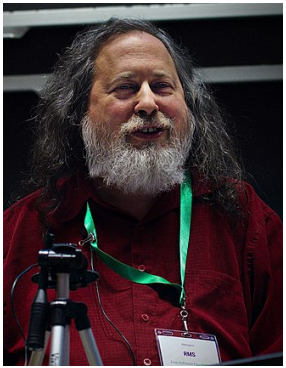


FREE SOFTWARE FOUNDATION

Free software

According to the **Free Software Foundation (FSF)**, a *software is free* if the program's users have the four essential freedoms:

1. Run the software
2. Study, change and improve
3. Copy
4. Distribute



FREE SOFTWARE FOUNDATION

Open source software (OSS)

It's software that is distributed with its source code.

1. It's code that is designed to be publicly accessible.
2. It's developed in a decentralized and collaborative way.
3. It typically includes a license.

Why to use it?

1. It's usually cheaper.
2. Peer review.
3. Transparency and Reliability.
4. Open collaboration.
5. Flexibility (source code is available for use, modification, and distribution).

Why to use it?

1. It's usually cheaper.
2. Peer review.
3. Transparency and Reliability.
4. Open collaboration.
5. Flexibility (source code is available for use, modification, and distribution).

Why to use it?

1. It’s usually cheaper.
2. Peer review.
3. Transparency and Reliability.
4. Open collaboration.
5. Flexibility (source code is available for use, modification, and distribution).

Why to use it?

1. It's usually cheaper.
2. Peer review.
3. Transparency and Reliability.
4. Open collaboration.
5. Flexibility (source code is available for use, modification, and distribution).

Why to use it?

1. It's usually cheaper.
2. Peer review.
3. Transparency and Reliability.
4. Open collaboration.
5. Flexibility (source code is available for use, modification, and distribution).

A good open source project (for GAs)

1. Who maintains it?
2. Is it up to date and well documented?
3. Does it have research papers (cites)?

A good open source project (for GAs)

1. Who maintains it?
2. Is it up to date and well documented?
3. Does it have research papers (cites)?

A good open source project (for GAs)

1. Who maintains it?
2. Is it up to date and well documented?
3. Does it have research papers (cites)?

A good open source project (for GAs)

1. Who maintains it?
2. Is it up to date and well documented?
3. Does it have research papers (cites)?

Section 2

Open source and alternative for Genetic Algorithms

2.2 Software for C/C++ and Java

Jenetics library[20]

- Developed and maintained mainly by Franz Wilhelmstötter, a senior software engineer.
- A Java library for Evolutionary Algorithms (Genetic Algorithms, Genetic Programming, etc).
- Supports parallel processing.
- Cited by research papers and frequent updates (very active community).

[20] Wilhelmstötter, *JENETICS LIBRARY USER'S MANUAL 6.3.0*, 2021.

Jenetics library[20]

- Developed and maintained mainly by Franz Wilhelmstötter, a senior software engineer.
- A Java library for Evolutionary Algorithms (Genetic Algorithms, Genetic Programming, etc).
- Supports parallel processing.
- Cited by research papers and frequent updates (very active community).

[20] Wilhelmstötter, *JENETICS LIBRARY USER'S MANUAL 6.3.0*, 2021.

Jenetics library[20]

- Developed and maintained mainly by Franz Wilhelmstötter, a senior software engineer.
- A Java library for Evolutionary Algorithms (Genetic Algorithms, Genetic Programming, etc).
- Supports parallel processing.
- Cited by research papers and frequent updates (very active community).

[20] Wilhelmstötter, *JENETICS LIBRARY USER'S MANUAL 6.3.0*, 2021.

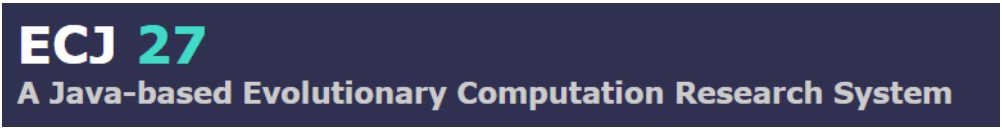
Jenetics library[20]

- Developed and maintained mainly by Franz Wilhelmstötter, a senior software engineer.
- A Java library for Evolutionary Algorithms (Genetic Algorithms, Genetic Programming, etc).
- Supports parallel processing.
- Cited by research papers and frequent updates (very active community).

[20] Wilhelmstötter, *JENETICS LIBRARY USER'S MANUAL 6.3.0*, 2021.

ECJ library[21]

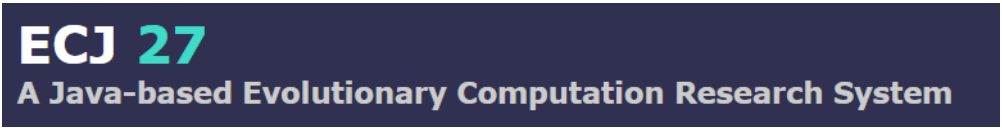
- A Java-based Evolutionary Computation Research System.
- Developed at George Mason University's ECLab Evolutionary Computation Laboratory.
- Extensive catalog of algorithms (Supports Genetic Algorithms, EDAs, Ant Colony Optimization, Genetic Programming, etc).



[21] Scott and Luke, "ECJ at 20: Toward a General Metaheuristics Toolkit", 2019.

ECJ library[21]

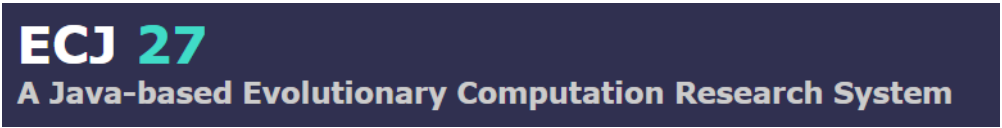
- A Java-based Evolutionary Computation Research System.
- Developed at George Mason University's ECLab Evolutionary Computation Laboratory.
- Extensive catalog of algorithms (Supports Genetic Algorithms, EDAs, Ant Colony Optimization, Genetic Programming, etc).



[21] Scott and Luke, "ECJ at 20: Toward a General Metaheuristics Toolkit", 2019.

ECJ library[21]

- A Java-based Evolutionary Computation Research System.
- Developed at George Mason University's ECLab Evolutionary Computation Laboratory.
- Extensive catalog of algorithms (Supports Genetic Algorithms, EDAs, Ant Colony Optimization, Genetic Programming, etc).



[21] Scott and Luke, "ECJ at 20: Toward a General Metaheuristics Toolkit", 2019.

Paradiseo[22]

- A C++ evolutionary computation framework for stochastic optimization algorithms.
- Started by the Geneura Team at the University of Granada. Current maintainers INRIA, University of the Littoral Opal Coast and Institut Pasteur.
- It focus on the efficiency and speed.
- Supports GA, ES, SA, local search mechanism.

[22] Dreo et al., "Paradiseo: From a Modular Framework for Evolutionary Computation to the Automated Design of Metaheuristics: 22 Years of Paradiseo", 2021.

Paradiseo[22]

- A C++ evolutionary computation framework for stochastic optimization algorithms.
- Started by the Geneura Team at the University of Granada. Current maintainers INRIA, University of the Littoral Opal Coast and Institut Pasteur.
- It focus on the efficiency and speed.
- Supports GA, ES, SA, local search mechanism.

[22] Dreo et al., "Paradiseo: From a Modular Framework for Evolutionary Computation to the Automated Design of Metaheuristics: 22 Years of Paradiseo", 2021.

Paradiseo[22]

- A C++ evolutionary computation framework for stochastic optimization algorithms.
- Started by the Geneura Team at the University of Granada. Current maintainers INRIA, University of the Littoral Opal Coast and Institut Pasteur.
- It focus on the efficiency and speed.
- Supports GA, ES, SA, local search mechanism.

[22] Dreo et al., "Paradiseo: From a Modular Framework for Evolutionary Computation to the Automated Design of Metaheuristics: 22 Years of Paradiseo", 2021.

Paradiseo[22]

- A C++ evolutionary computation framework for stochastic optimization algorithms.
- Started by the Geneura Team at the University of Granada. Current maintainers INRIA, University of the Littoral Opal Coast and Institut Pasteur.
- It focus on the efficiency and speed.
- Supports GA, ES, SA, local search mechanism.

[22] Dreo et al., "Paradiseo: From a Modular Framework for Evolutionary Computation to the Automated Design of Metaheuristics: 22 Years of Paradiseo", 2021.

Section 2

Open source and alternative for Genetic Algorithms

2.3 Software for Python

DEAP framework[23]

- Developed at the Computer Vision and Systems Laboratory (CVSL) at Université Laval, in Quebec city, Canada.
- A Python framework for Evolutionary Algorithms (Genetic Algorithms and Genetic Programming).
- Supports parallelization.
- Very easy to use.

DISTRIBUTED
EVOLUTIONARY
ALGORITHMS IN
PYTHON

[23] Fortin et al., "DEAP: Evolutionary Algorithms Made Easy", 2012.

DEAP framework[23]

- Developed at the Computer Vision and Systems Laboratory (CVSL) at Université Laval, in Quebec city, Canada.
- A Python framework for Evolutionary Algorithms (Genetic Algorithms and Genetic Programming).
- Supports parallelization.
- Very easy to use.

DISTRIBUTED
EVOLUTIONARY
ALGORITHMS IN
PYTHON

[23] Fortin et al., "DEAP: Evolutionary Algorithms Made Easy", 2012.

DEAP framework[23]

- Developed at the Computer Vision and Systems Laboratory (CVSL) at Université Laval, in Quebec city, Canada.
- A Python framework for Evolutionary Algorithms (Genetic Algorithms and Genetic Programming).
- Supports parallelization.
- Very easy to use.

DISTRIBUTED
EVOLUTIONARY
ALGORITHMS IN
PYTHON

[23] Fortin et al., "DEAP: Evolutionary Algorithms Made Easy", 2012.

DEAP framework[23]

- Developed at the Computer Vision and Systems Laboratory (CVSL) at Université Laval, in Quebec city, Canada.
- A Python framework for Evolutionary Algorithms (Genetic Algorithms and Genetic Programming).
- Supports parallelization.
- Very easy to use.

DISTRIBUTED
EVOLUTIONARY
ALGORITHMS IN
PYTHON

[23] Fortin et al., "DEAP: Evolutionary Algorithms Made Easy", 2012.

pymoo framework[24]

- Powered by the Python research community **anyoptimization**. Maintained by Julian Blank & Kalyanmoy Deb (Computational Optimization and Innovation Laboratory (COIN), Michigan State University in East Lansing, Michigan, USA).
- A Python framework for Evolutionary Algorithms (single and multiobjective).
- Performance indicator.
- Wide range of algorithms (GA, DE, NSGAs, PSO, ES, etc).

[24] Blank and Deb, "pymoo: Multi-Objective Optimization in Python", 2020.

pymoo framework[24]

- Powered by the Python research community **anyoptimization**. Maintained by Julian Blank & Kalyanmoy Deb (Computational Optimization and Innovation Laboratory (COIN), Michigan State University in East Lansing, Michigan, USA).
- A Python framework for Evolutionary Algorithms (single and multiobjective).
- Performance indicator.
- Wide range of algorithms (GA, DE, NSGAs, PSO, ES, etc).

[24] Blank and Deb, "pymoo: Multi-Objective Optimization in Python", 2020.

pymoo framework[24]

- Powered by the Python research community **anyoptimization**. Maintained by Julian Blank & Kalyanmoy Deb (Computational Optimization and Innovation Laboratory (COIN), Michigan State University in East Lansing, Michigan, USA).
- A Python framework for Evolutionary Algorithms (single and multiobjective).
- Performance indicator.
- Wide range of algorithms (GA, DE, NSGAs, PSO, ES, etc).

[24] Blank and Deb, "pymoo: Multi-Objective Optimization in Python", 2020.

pymoo framework[24]

- Powered by the Python research community **anyoptimization**. Maintained by Julian Blank & Kalyanmoy Deb (Computational Optimization and Innovation Laboratory (COIN), Michigan State University in East Lansing, Michigan, USA).
- A Python framework for Evolutionary Algorithms (single and multiobjective).
- Performance indicator.
- Wide range of algorithms (GA, DE, NSGAs, PSO, ES, etc).

[24] Blank and Deb, "pymoo: Multi-Objective Optimization in Python", 2020.

Section 3

Hands-On Genetic Algorithms

Section 3

Hands-On Genetic Algorithms

3.1 Single-objective optimization problem

Single-objective optimization problem

- A **general single-objective optimization problem**[25] is defined as minimizing (or maximizing) $f(x)$ subject to $g_i(x) \leq 0, i = \{1, \ldots, m\}$, and $h_j(x) = 0, j = \{1, \ldots, p\} x \in \Omega$.
- x is a **decision variables** The vector **x** of **n** decision variables is represented by:

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]^T \tag{5}$$

[25] 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[26]

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

Single-objective optimization problem

- The method for finding the **global optimum** (may not be unique) of any function is referred to as Global Optimization. In general, the global minimum of a single objective problem is presented in the next definition[27]:

$$f : \Omega \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}, \Omega \neq \varnothing, \text{ for } \mathbf{x} \in \Omega \text{the value } f^* \triangleq f(\mathbf{x}^*) > -\infty \tag{6}$$

- is called a **global minimum** if and only if

$$\forall \mathbf{x} \in \Omega : f(\mathbf{x}^*) \leq f(\mathbf{x}) \tag{7}$$

- $\mathbf{x}^*$ is by definition the global minimum solution, f is the objective function, and the set Ω is the feasible region of $\mathbf{x}$. The goal of determining the global minimum solution(s) is called the global optimization problem for a single-objective problem

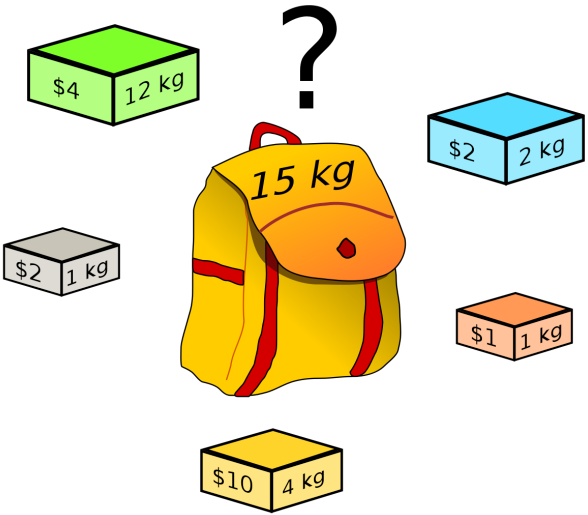
[27] Coello et al., *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2007.

Section 3

Hands-On Genetic Algorithms

3.2 Knapsack problem

The knapsack problem



0–1 knapsack problem

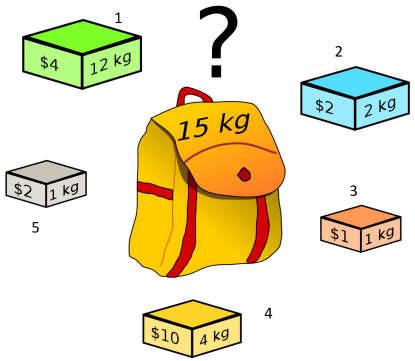
- We are given a set of n items, each of which has attached to it some value v_i, and some weight w_i. The task is to select a subset of those items that maximizes the sum of the values, while keeping the summed weight within some capacity C_{max}.
- The goal is to:

$$\text{Maximize : } \sum_{i=1}^{n} v_i \cdot x_i \tag{8}$$

$$\text{Subject to the constraints: } \sum_{i=1}^{n} w_i \cdot x_i \leq C_{max} \tag{9}$$

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 https://github.com/alex-cornejo/ENC2022/tree/main/knapsack.

Section 3

Hands-On Genetic Algorithms

3.3 Traveling Salesman Problem and n-queens

The Traveling Salesman Problem (TSP)

Given a **set of cities**, find the shortest tour for a salesman that visits each city exactly once, starting and ending his tour in the same city.

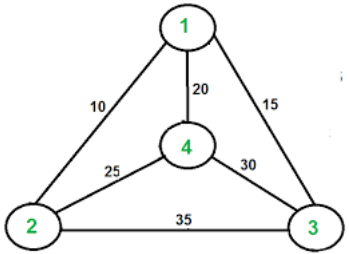
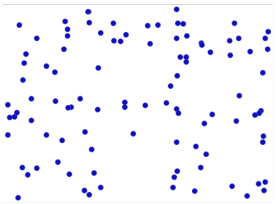


Figure 10: TSP instance with 4 cities.

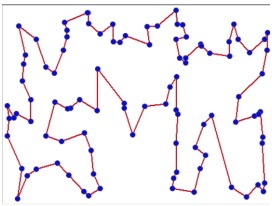
The Traveling Salesman Problem

Input: A complete weighted graph $G = (V, E)$.

Task: Find a *Hamiltonian Cycle* of minimum weight.



(a) Instance.



(b) Optimal solution.

Applications of TSP

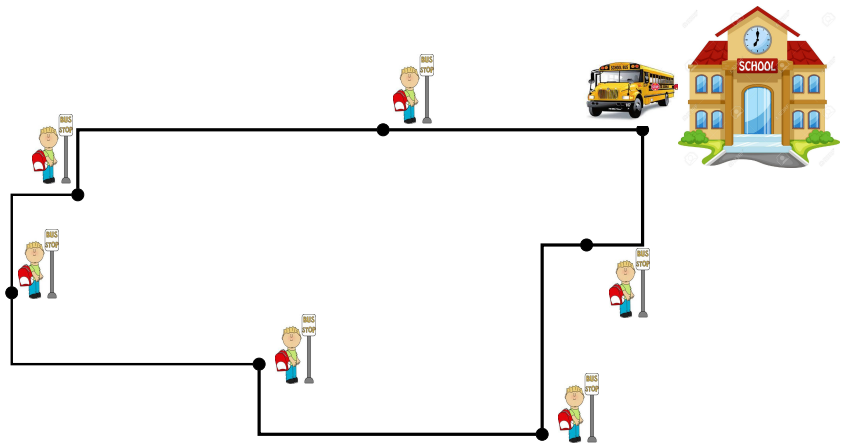


Figure 12: Scholar bus.

Applications of TSP

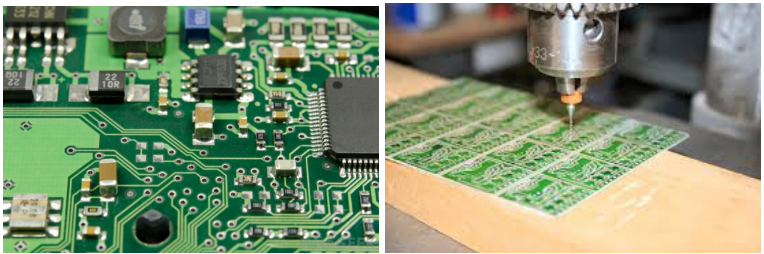


Figure 13: Printing circuit boards.

Brute force for the TSP

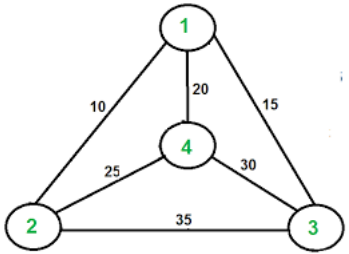


Figure 14: TSP instance with 4 cities.

1.- (1,2,3,4)
2.- (1,2,4,3)
?.- ...

Brute force for the TSP

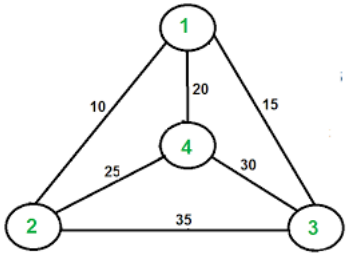


Figure 14: TSP instance with 4 cities.

1.- (1,2,3,4)
2.- (1,2,4,3)
?.- ...

Brute force for the TSP

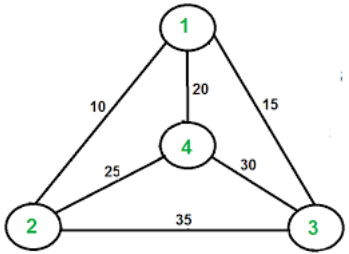


Figure 14: TSP instance with 4 cities.

1.- (1,2,3,4)
2.- (1,2,4,3)
?.- ...

Brute force for the TSP

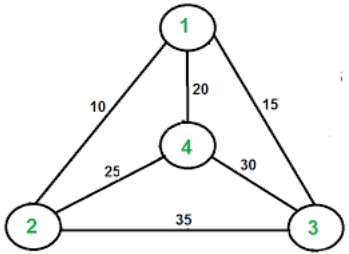


Figure 14: TSP instance with 4 cities.

1.- (1,2,3,4)
2.- (1,2,4,3)
?.- ...

Brute force for the TSP

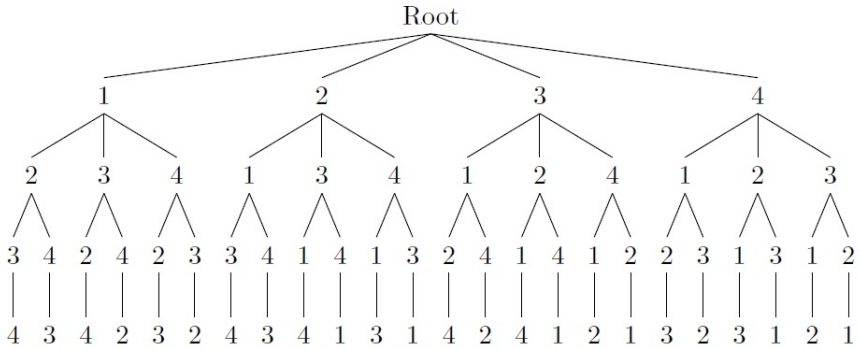


Figure 15: For 4 cities there are 4! permutations.

Genetic Algorithm for the TSP

1. Representation.
2. Genetic operators.
   - Initialization: random permutations
   - Selection: roulette wheel | tournament
   - Mutation: Swap indices
   - Recombination: ?

Genetic Algorithm for the TSP

1. Representation.

2. Genetic operators.
   - Initialization: random permutations
   - Selection: roulette wheel | tournament
   - Mutation: Swap indices
   - Recombination: ?

Genetic Algorithm for the TSP

1. Representation.

2. Genetic operators.
   - Initialization: random permutations
   - Selection: roulette wheel | tournament
   - Mutation: Swap indices
   - Recombination: ?

Genetic Algorithm for the TSP

1. Representation.

2. Genetic operators.
   - Initialization: random permutations
   - Selection: roulette wheel | tournament
   - Mutation: Swap indices
   - Recombination: ?

Genetic Algorithm for the TSP

1. Representation.

2. Genetic operators.
   - Initialization: random permutations
   - Selection: roulette wheel | tournament
   - Mutation: Swap indices
   - Recombination: ?

Genetic Algorithm for the TSP

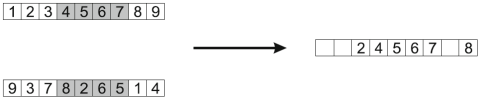
1. Representation.

2. Genetic operators.
   - Initialization: random permutations
   - Selection: roulette wheel | tournament
   - Mutation: Swap indices
   - Recombination: ?

Partially Mapped Crossover (PMX)

1. Choose 2 random points and copy the segment from P1 to the offspring.

2. Copy elements **in segment** from P2 to the offspring by *following a strategy*.

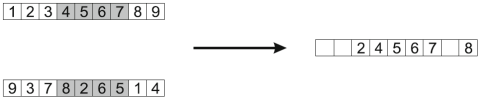


3. Copy remaining elements from P2 into same positions in offspring.

Partially Mapped Crossover (PMX)

1. Choose 2 random points and copy the segment from P1 to the offspring.

2. Copy elements **in segment** from P2 to the offspring by *following a strategy*.

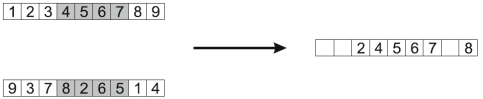


3. Copy remaining elements from P2 into same positions in offspring.

Partially Mapped Crossover (PMX)

1. Choose 2 random points and copy the segment from P1 to the offspring.

2. Copy elements **in segment** from P2 to the offspring by *following a strategy*.



3. Copy remaining elements from P2 into same positions in offspring.

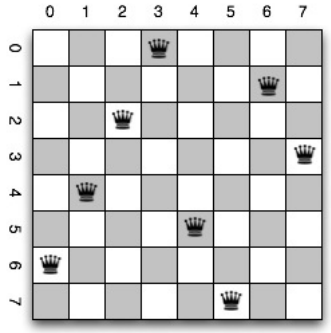
Hands-on

Open code https://github.com/alex-cornejo/ENC2022.

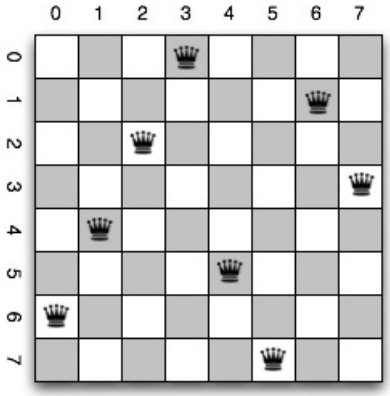
The n-queens problem

Input: A positive integer $n \in \mathbb{Z}^+$.

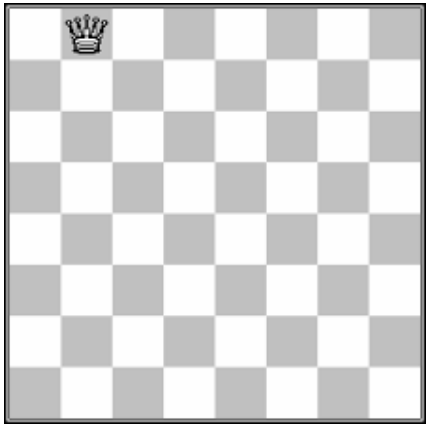
Task: Place n queens on a $n \times n$ chessboard so that no two of them can check each other.



Which representation to use?



Fitness function?



Section 3

Hands-On Genetic Algorithms

3.4 The Frequency Assignment Problem (FAP)

The Frequency Assignment Problem (FAP)

Input: A double weight undirected graph $G = (V, E, D, P)$ with a vertices set V, and edges set E, two edge weight sets D (distance constraints) and P (penalization), and a set F of consecutive frequencies.

Task: Find a mapping $f: V \to F$ such that the following objective function is minimized.

$$\min \sum_{(i,j)\in E; |f_i - f_j| \leq d_{ij}} p_{ij} \tag{10}$$

Application of FAP

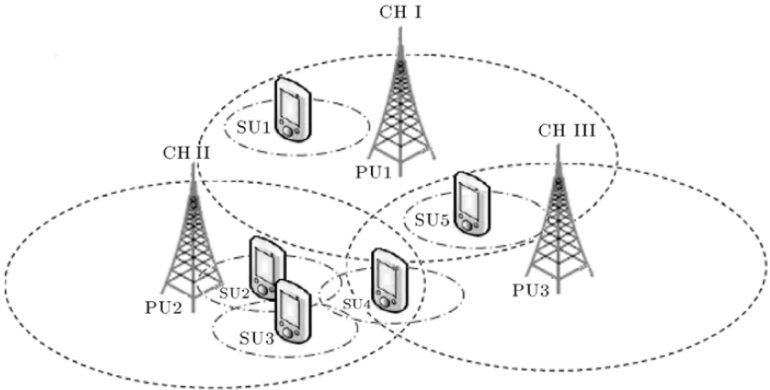
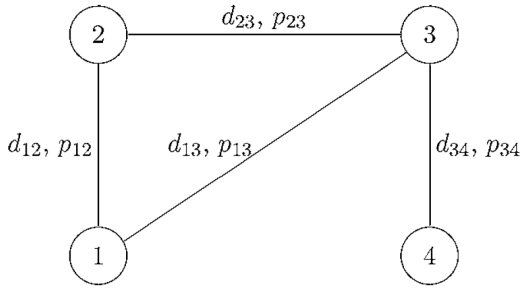


Figure 16: Taken from[28].

[28] Koroupi, Talebi, and Salehinejad, "Cognitive radio networks spectrum allocation: An ACS perspective", 2012.

The Frequency Assignment Problem (FAP)



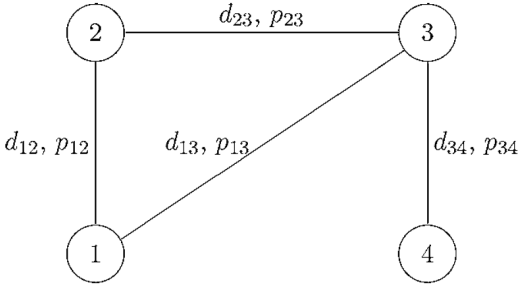
$$E = \{\{1,2\},\{1,3\},\{2,3\},\{3,4\}\}$$

$$V = \{1,2,3,4\} \quad F = \{0,1,\ldots,|F|-1\}$$

Figure 17: Taken from[29].

[29] Montemanni, Smith, and Allen, "Lower Bounds for Fixed Spectrum Frequency Assignment", 2001.

The Frequency Assignment Problem (FAP)



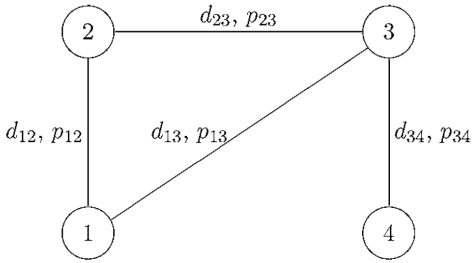
$$E = \{\{1,2\},\{1,3\},\{2,3\},\{3,4\}\}$$

$$V = \{1,2,3,4\} \quad F = \{0,1,\ldots,|F|-1\}$$

E	D	P
$(1,2)$	$d_{12} = 1$	$p_{12} = 3$
$(1,3)$	$d_{13} = 2$	$p_{13} = 2$
$(2,3)$	$d_{23} = 1$	$p_{23} = 1$
$(3,4)$	$d_{34} = 3$	$p_{34} = 4$

Genetic Algorithm for the FAP

1. Representation.
2. Genetic operators.
   - Initialization: random integers
   - Selection: roulette wheel | tournament
   - Mutation: Integer Uniform
   - Recombination: 2-points

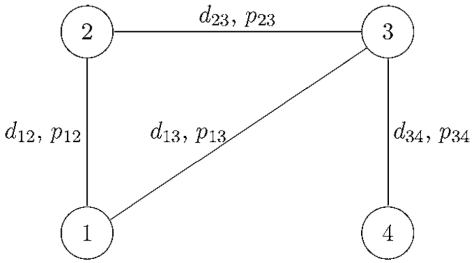


$$E = \{\{1,2\},\{1,3\},\{2,3\},\{3,4\}\}$$

$$V = \{1,2,3,4\} \quad F = \{0,1,\ldots,|F|-1\}$$

Genetic Algorithm for the FAP

1. Representation.
2. Genetic operators.
   - Initialization: random integers
   - Selection: roulette wheel | tournament
   - Mutation: Integer Uniform
   - Recombination: 2-points



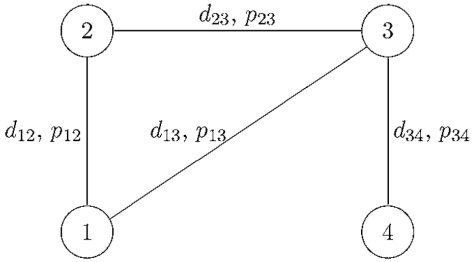
$$E = \{\{1,2\},\{1,3\},\{2,3\},\{3,4\}\}$$

$$V = \{1,2,3,4\} \quad F = \{0,1,\ldots,|F|-1\}$$

Genetic Algorithm for the FAP

1. Representation.

2. Genetic operators.
   - Initialization: random integers
   - Selection: roulette wheel | tournament
   - Mutation: Integer Uniform
   - Recombination: 2-points



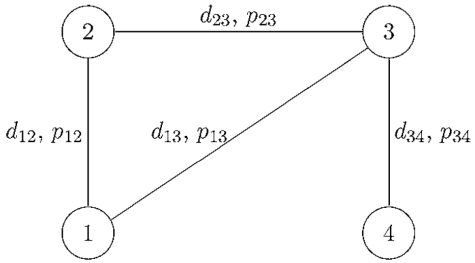
$$E = \{\{1,2\},\{1,3\},\{2,3\},\{3,4\}\}$$

$$V = \{1,2,3,4\} \quad F = \{0,1,\ldots,|F|-1\}$$

Genetic Algorithm for the FAP

1. Representation.

2. Genetic operators.
   - Initialization: random integers
   - Selection: roulette wheel | tournament
   - Mutation: Integer Uniform
   - Recombination: 2-points



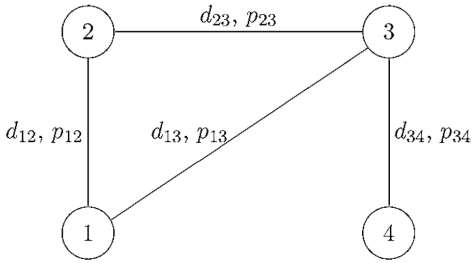
$$E = \{\{1,2\},\{1,3\},\{2,3\},\{3,4\}\}$$

$$V = \{1,2,3,4\} \quad F = \{0,1,\ldots,|F|-1\}$$

Genetic Algorithm for the FAP

1. Representation.

2. Genetic operators.
   - Initialization: random integers
   - Selection: roulette wheel | tournament
   - Mutation: Integer Uniform
   - Recombination: 2-points



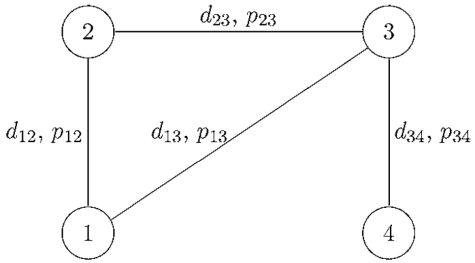
$$E = \{\{1,2\},\{1,3\},\{2,3\},\{3,4\}\}$$

$$V = \{1,2,3,4\} \quad F = \{0,1,\ldots,|F|-1\}$$

Genetic Algorithm for the FAP

1. Representation.

2. Genetic operators.
   - Initialization: random integers
   - Selection: roulette wheel | tournament
   - Mutation: Integer Uniform
   - Recombination: 2-points



$$E = \{\{1,2\},\{1,3\},\{2,3\},\{3,4\}\}$$

$$V = \{1,2,3,4\} \quad F = \{0,1,\ldots,|F|-1\}$$

Hands-on

Open code https://github.com/alex-cornejo/ENC2022.

Section 3

Hands-On Genetic Algorithms

3.5 The Multiobjective Optimization Problem

Multiobjective optimization

- **The Multiobjective Optimization Problem** can then be defined (in words) as the problem of finding[30]:
- *"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.*

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

Multiobjective optimization

$$\left.\begin{aligned} \text{Minimize/Maximize } \mathbf{f}_m(\mathbf{x}), m = 1, 2, \ldots, k; \\ \text{subject to } g_j(\mathbf{x}) \geq 0, j = 1, 2, \ldots, m; \\ h_k(\mathbf{x}) = 0, k = 1, 2, \ldots, p; \\ x_i^{(L)} \leq x_i \leq x_i^{(U)}, i = 1, 2, \ldots, t; \end{aligned}\right\} \quad (11)$$

- 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[31].

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

Multiobjective optimization

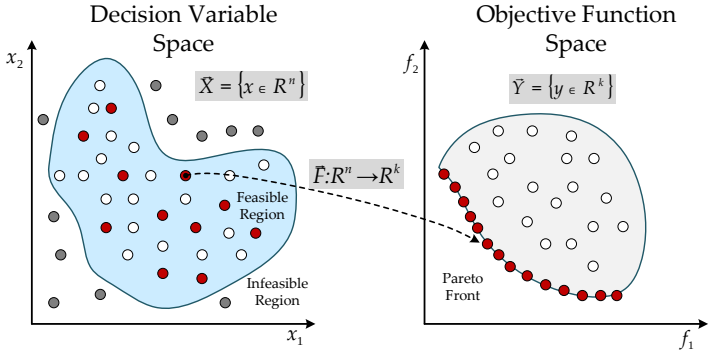


Figure 18: Mapping of decision variables space to the objective function space. Feasible solutions and zone are marked in blue. In the decision variable space, the Pareto optimal set is marked with red solutions and its mapping to the objective function space creates the Pareto front.

Multiobjective optimization

- **Pareto dominance**[32]: 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:

$$\mathcal{P}^* = \{\mathbf{x} \in \Omega \mid \neg\exists \mathbf{x}' \in \Omega \; F(\mathbf{x}') \preceq F(\mathbf{x})\} \tag{12}$$

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

$$\mathcal{PF}^* = \{\mathbf{u} = F(\mathbf{x}) \mid \mathbf{x} \in \mathcal{P}^*\} \tag{13}$$

[32] 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:

$$\begin{cases} f_1(\mathbf{x}) = 100(x_1^2 + x_2^2), \\ f_2(\mathbf{x}) = (x_1 - 5)^2 + x_2^2, \\ \textbf{Subject to:} \\ g_1(\mathbf{x}) \equiv 2(x_1 - 0.1)(x_1 - 0.9)/0.18 \leq 0, \\ g_2(\mathbf{x}) \equiv -20(x_1 - 0.4)(x_1 - 0.6)/4.8 \leq 0, \\ -2 \leq x_1 \leq 2, \\ -2 \leq x_2 \leq 2 \end{cases} \tag{14}$$

Hands-on

Open code https://github.com/alex-cornejo/ENC2022/tree/main/MOO.

Section 4

Final remarks

Questions?

Section 5

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.

Blank, J. and K. Deb. "pymoo: Multi-Objective Optimization in Python". In: *IEEE Access* 8 (2020), pp. 89497–89509.

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.

References II

Dreo, Johann et al. "Paradiseo: From a Modular Framework for Evolutionary Computation to the Automated Design of Metaheuristics: 22 Years of Paradiseo". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1522–1530. ISBN: 9781450383516. URL: https://doi.org/10.1145/3449726.3463276.

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.

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.

Fortin, Félix-Antoine et al. "DEAP: Evolutionary Algorithms Made Easy". In: *Journal of Machine Learning Research* 13 (2012), pp. 2171–2175.

Koroupi, F., S. Talebi, and H. Salehinejad. "Cognitive radio networks spectrum allocation: An ACS perspective". In: *Scientia Iranica* 19.3 (2012), pp. 767–773. ISSN: 1026-3098. DOI: https://doi.org/10.1016/j.scient.2011.04.029. URL: https://www.sciencedirect.com/science/article/pii/S1026309811002847.

References III

Montemanni, Roberto, D.H. Smith, and Stuart Allen. "Lower Bounds for Fixed Spectrum Frequency Assignment". In: *Annals of Operations Research* 107 (Oct. 2001), pp. 237–250. DOI: 10.1023/A:1014911401612.

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.

Scott, Eric O. and Sean Luke. "ECJ at 20: Toward a General Metaheuristics Toolkit". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 1391–1398. ISBN: 9781450367486. DOI: 10.1145/3319619.3326865. URL: https://doi.org/10.1145/3319619.3326865.

Storn, Rainer and Kenneth Price. *Differential Evolution-A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*. Tech. rep. 1997, pp. 341–359.

Wilhelmstötter, Franz. *JENETICS LIBRARY USER'S MANUAL 6.3.0*. 2021. URL: https://jenetics.io/.

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

References IV

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