Project report - BDR Thermea Group

Quentin Dumont
under the supervision of Aurélien Massein

22 août 2021

Table des matières

1 Introduction

1.1 Presentation of the company

BDR Thermea Group is a leading european company (third in Europe) dedicated to the manufacture and sale of domestic and industrial heating appliances.
Created in 2009 by the merger of several leading european brands (De Dietrich, Remeha ..), It offers a great many market-leading brands includes Baxi, De Dietrich, Remeha, Brötje ... and are among the first choice for installers, with strong local presence and support.
With a head office in Apeldoorn (Netherlands), BDR groups serve customers in more than 100 countries, employ 6300 people across the globe and has 1.8 billion euros in revenues.
More over, The group inovate to fight against climate change, it manufactures products with a near-zero carbon footprint and it developps innovative products to save energy and cut carbon emissions.

1.2 Internship

The company BDR Thermea Group located at Mertzwiller initialy welcome me in early march for a small project about the optimization. I had the chance to continue the research on the company during a two-month intership that is in the continuity of the previously project.

2 Project

2.1 Project context and reformulation

A thermo-dynamical water-Heater is composed of a water tank, a heat pump and a control system as shown below 2.1.

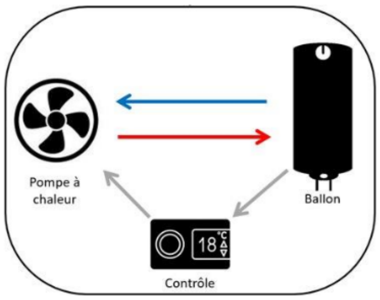


Figure 1 – Thermo-Dynamical Water-Heater

Amoung all product performances, we focus particulary on two critera of interest, being :

- **Coefficient of performance**(COP) of domestic hot water (DHW) noted by COP_{DHW} and defined by the EN16147 :2017 norm [4] that specifies the test conditions for determining the performances of a thermodynamical water heater
- **Star notation** defined by the LCIE 103-15/C specifications [3] calculated based on the performances measured during the EN16147 :2017 test protocol.

Indeed, these two critera are essential to the sale of products in european union which request strict quotas and to improve the sales of the company. In this subject, I'm looking to maximize these both criteria of performance to optimally design a product and to guarantee its quality

In order to solve this complex problem, I based the approach on the use of evolutionnary algorithms such as NSGA-II, NSGA-III (etc ...), that I implemented in python using the JmetalPy library [9]

2.2 Products tested

We will refer in this report to two different products when testing the tools we implemented : IDU3FS[19] in figure [2] and TWH3001[20] in figure [3]

FIGURE 2 – IDU3FS
Source: BDR Thermea [19]

FIGURE 3 – TWH3001
Source: BDR Thermea [20]

2.2.1 IDU3FS

IDU3FS[19] is a triple service air-source [1]heat pump dedicated to the new market, compact and easy to integrate into any type of configuration, even in a closet and offer an automatic heating / cooling as needed, while guaranteeing the domestic hot water (DHW) production.

Charactestics :
- Compatible with the RT2012 [2] regulation and standard
- Compact : 560x586x1950 mm

2.2.2 TWH

The thermodynamic water heater (THW)[20] operates too on the principle of the air source heat pump. It extracts the heat from the inlet air which could be either from oudoor or indoor. The various models can be used to extract energy either from the ambient air (a cellar, for example), from the outside air or when connected to a mechanical ventilation system.

Charactestics :
- Plenty of domestic hot water (214 -270 litres)
- hot water heating up to 62°C with PAC
- Dimension : 610x610x1690 mm

2.3 Roadmap

— Step 1 : Organization and research(article, report ...) about the subject.
— Step 2 : get familiar with the different multi-objectives algorithms.
  — Understand how evolutionnary algorithms works (NSGA2,NSGA3,MOPSO,SMOPSO, ...)
  — Explore JmetalPy (functions, implementation, ...)

1. Two heating circuit appliance and a dhw appliance (we focused on the dhw service)
2. Focusing on the new market when building new homes to meet ecological requirements

- Step 3 : optimization with a data set using a genetic algorithm
  - Define the optimization problem
  - Try different functions : stopping criterion, mutation, ...
  - Make a statistic study of various parameters for the algorithm
- Step 4 : compare different algorithms
  - Select different evolutionary algorithms
  - Make a statistic study of each one on multiple data sets
  - Determine the best parameters for different algorithms (for any data set)
- Step 5 : Select one
  - run simulation with the good parameter of the algorithm chosen
  - Make a statistic study

2.4 Dymola and FMU

Dymola originates from "The Dynamic Modeling Language" [16] and was implemented in Simula 67 (nowadays, re-implemented in Pascal and C++). It is a complete tool for modeling and simulation of complex systems used in many areas as robotic, aeronautics, thermodynamic, ... It is based on the Modelica language and can model and simulate any physical components which could be described by differential equations (or lowest level's algebraic equations). these compenents could be for example : **thermo-dynamic, thermal, electric, mecanic, hydraulic etc..**
In our case, these products were modelled in Dymola and exported in FMU, which can be called and exploited without the need of the Symola software. FMUs are indispensable to simulate the product behaviour and apply to it the norm process, in order to simulate a thermo-dynamic water-heater operation and to obtain valuable data (detailed later). After the models (IDU3FS and TWH3001) are built by Dymola, we can use a Python library called **FMpy**[30] in order to speed up the simulation and to launch it in our code.
Indeed, **FMpy** is a free Python library to simulate **Functional Mock-up Units**(FMUs) that :

- supports FMI 1.0 and 2.0
- supports Co-Simulation and Model exchange
- has a command line, graphical user interface

2.5 The parameters, objectives and data

2.5.1 Parameters

Several simulations of the thermo-dynamical water heater's model were performed to obtain a number of important data (COP_{DHW}, $CPUtime$,...). For this, 3 parameters are indispensable to permit the proper functioning of the simulation : $T_{set}, \Delta T_{hysteresis}, H_{gauge}$.

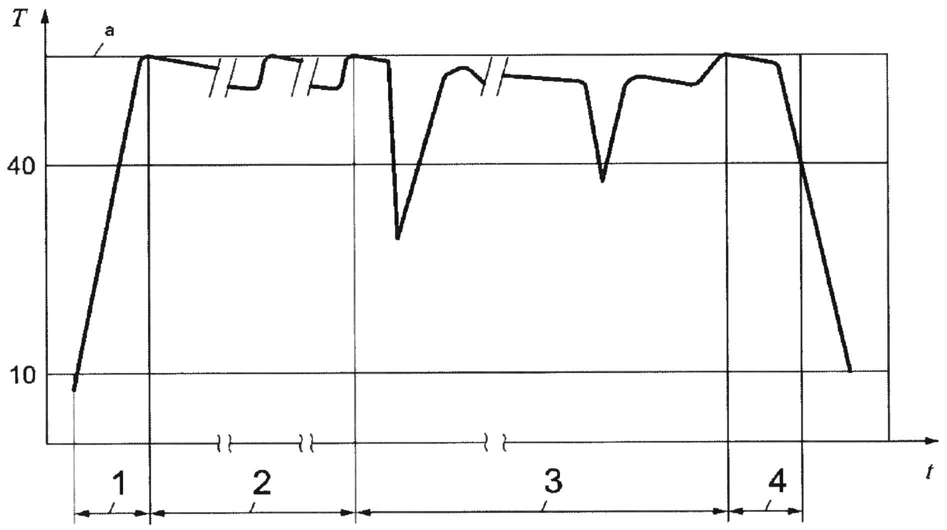
- T_{set} is the setpoint temperature, the maximal temperature to reach in order to stop the heating process
- $\Delta T_{hysteresis}$ is the temperature hysteresis, as $T_{min} = T_{set} - \Delta T_{hysteresis}$ is the minimal temperature to reach in order to start the heating process
- H_{gauge} is the height-location (from the top) of the temperature sensor mount onto the water heater which measures and regulates the tank's water temperature

These three parameters with the influence of the time-depending runing's simulation determine with precision the result of outputs.
Identifying these parameters value is the core of our optimisation problem and the design of products.

2.5.2 Objectives

Originally, a product is evaluated in real conditions to obtain its COP_{DHW} and to value its star notation (as well as other objectives). This test is long, it takes roughly 14 days (this may vary depending on the

parameters). The test in real conditions requires different specific steps and evaluates the quality of the products as described before.
Here is a detailed example :



Légende

1	[Étape C] remplissage et période de mise en température (voir 7.7)	T	température
2	[Étape D] Puissance absorbée en régime stabilisé (voir 7.8)	t	temps
3	[Étape E] Puisages d'eau (voir 7.9)	a	température de consigne
4	[Étape F] Eau mitigée à 40 °C et température d'eau chaude de référence (voir 7.10)		

Figure 1 — Étapes et ordre des essais

FIGURE 4 – A test in real conditions
Source: EN 16147 norm [4]

Explication :

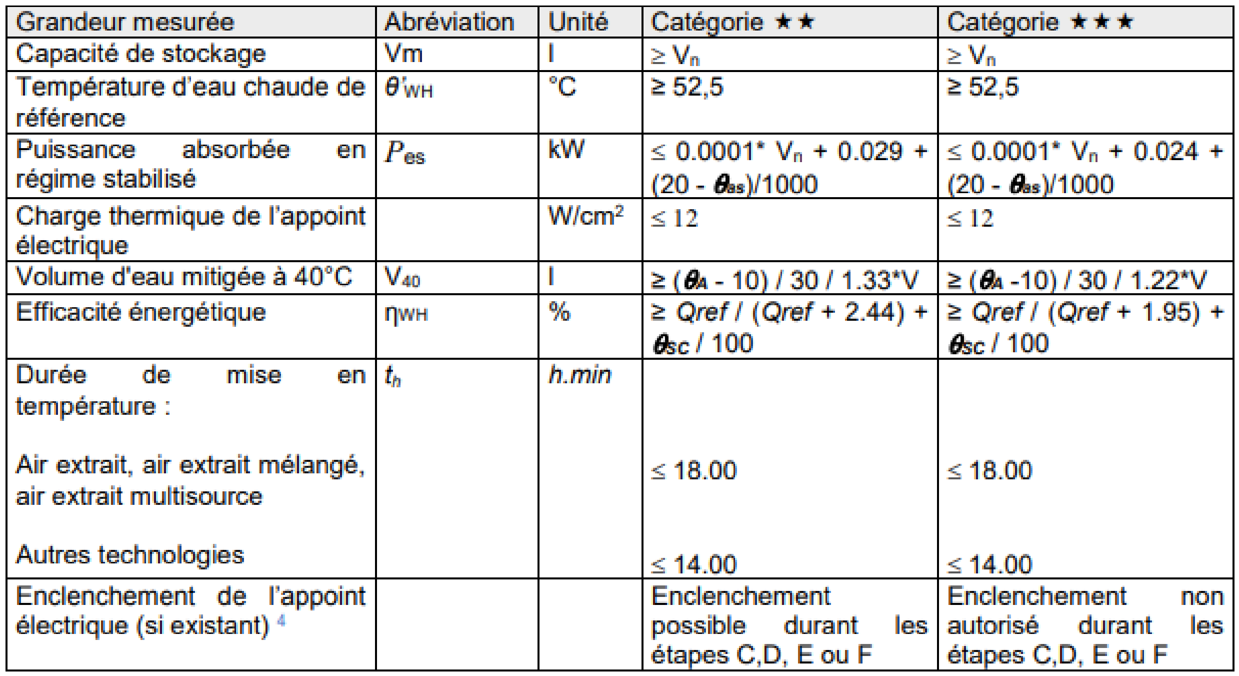
- **Step C :** From 10-degree water filled water tank, the product should heat the water until a set point temperarature noted T_{set}.
- **Step D :** Product is in idle state, there is not DHW drawings and the natural heat losses will affect the stored hot water in the tank and the product will need to heat it when a given minimal temperature is reached, noted $T_{min}= T_{set} - \Delta T_{hysteresis}$
- **Step E :** Perform a 24-hour withdrawing's cycle L or M and then, wait until the product is running (again) and finishes to heat the water tank.
- **Step F :** Empty water-heater tank completely and quantify the stored hot water volume equivalent to 40 °C ; during that step Θ'_{WH} measure corresponds to the average withdrawn outlet temperature.

To speed up the process, the tests are simulated with FMUs (Function Mockup Units). A test simulated this way takes around 44 minutes. Methods such as metamodelisation [21] can't be used because the model is still very complex (around 40000 equations), involves proprietary DLLs with unknown code, and each step of the test is itself described with complex algorithms specified and imposed by the norm EN16147.
Thanks to the FMU, A very large amount of information (outputs) is computed. These information contain

different values as :

- COP_{DHW}, V_{40}, P_{es}, θ'_{wh}, t_h, $CPUtime$, ...

They correspond respectively to Coefficient of Performance, amount of mixed water at 40°C, an absorptive power at steady-state, reference hot water temperature, warm-up period, simulation time. **Stars** can be determined with these values by criteria for obtaining as follows :



Grandeur mesurée	Abréviation	Unité	Catégorie ★★	Catégorie ★★★
Capacité de stockage	Vm	l	≥ V_n	≥ V_n
Température d'eau chaude de référence	θ'_{WH}	°C	≥ 52,5	≥ 52,5
Puissance absorbée en régime stabilisé	P_{es}	kW	≤ 0.0001* V_n + 0.029 + (20 - θ_{as})/1000	≤ 0.0001* V_n + 0.024 + (20 - θ_{as})/1000
Charge thermique de l'appoint électrique		W/cm²	≤ 12	≤ 12
Volume d'eau mitigée à 40°C	V_{40}	l	≥ (θ_A - 10) / 30 / 1.33*V	≥ (θ_A -10) / 30 / 1.22*V
Efficacité énergétique	η_{WH}	%	≥ *Qref* / (*Qref* + 2.44) + θ_{SC} / 100	≥ *Qref* / (*Qref* + 1.95) + θ_{SC} / 100
Durée de mise en température :	t_h	*h.min*		
Air extrait, air extrait mélangé, air extrait multisource			≤ 18.00	≤ 18.00
Autres technologies			≤ 14.00	≤ 14.00
Enclenchement de l'appoint électrique (si existant) [4]			Enclenchement possible durant les étapes C,D, E ou F	Enclenchement non autorisé durant les étapes C,D, E ou F

FIGURE 5 – criteria of stars
Source: LCIE 103-15 [23]

The most problematic condition is the reference hot water temperature θ'_{WH} because it is quite difficult to achieve contrary to t_h which is much easier. The product need at least 1 star notation to be sold.
By convention, we state that the 1 star notation corresponds to passing the EN16147 norm. We need at "1 star" to sell the product in the european market.

2.5.3 Discretizing the data

In order to have "manucfaturable" parameters and to reduce our optimisation problem complexity, we need to discretise our parameters. Such discretion is a crucial step.
Indeed, a discretization too weak needless risks of losing valuable information while conversely, a too strong discretization risks to significantly increase computation time and to not respect the physical conditions of the product.
In that result, We define the parameters' range and design space of our optimisation by a cubic-grid, having $N = 16 \times 29 \times 19 = 8816$ possible designs [3], as :

— $T_{set} \in [315.15, 330.15]$ every 1 Kelvin degree, as $dim(\mathbb{T}_{set}) = 16$ => [42.0, 57.0] °C
— $\Delta T_{hysteresis} \in [2, 30]$ every 1 degree, as $dim(\Delta\mathbb{T}_{hysteresis}) = 29$
— $H_{gauge} \in [0, 0.9]$ every 0.05 meter, as $dim(\mathbb{H}_{gauge}) = 19$

3. Some of these designs are not physically possible

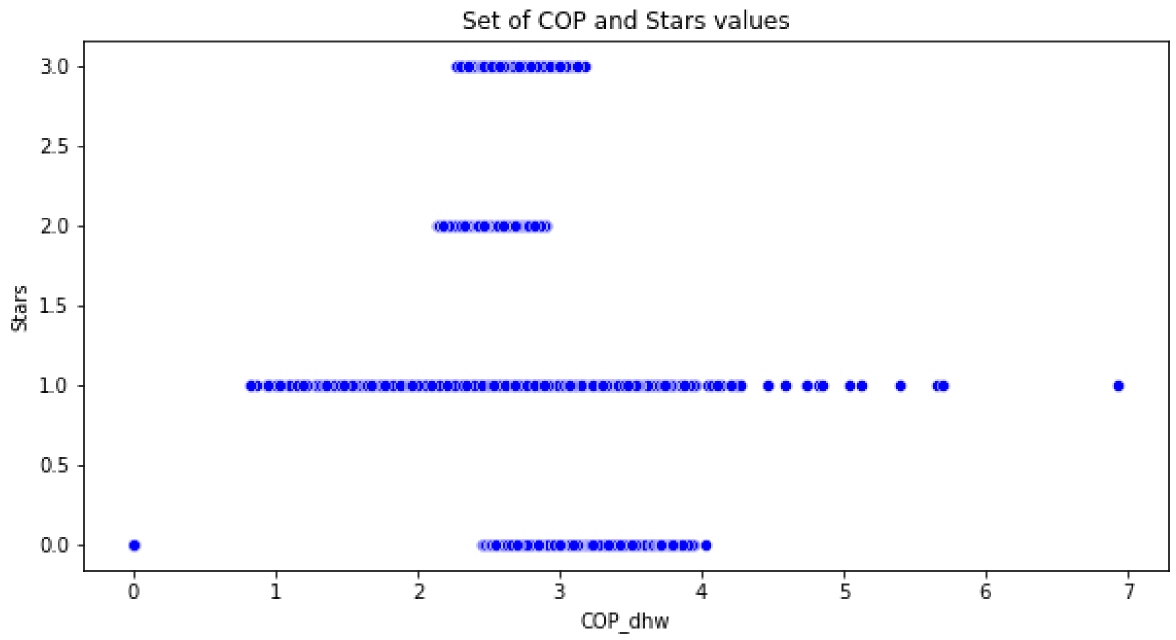
So the main optimization problem can be formulated as such :

$$\underset{u \in \mathcal{X}}{\operatorname{argmax}}(COP_{DHW}(u), stars(u)) \tag{1}$$

where $\mathcal{X} = \mathbb{T}_{set} \times \Delta\mathbb{T}_{hysteresis} \times \mathbb{H}_{gauge}$

2.5.4 Data

For each combination in this discretized set, the COP_{DHW} and $Stars$ values was already computed with an FMU, so any simulations have to be done.
We obtain a visualization of all possible COP_{DHW} and $Stars$ values :



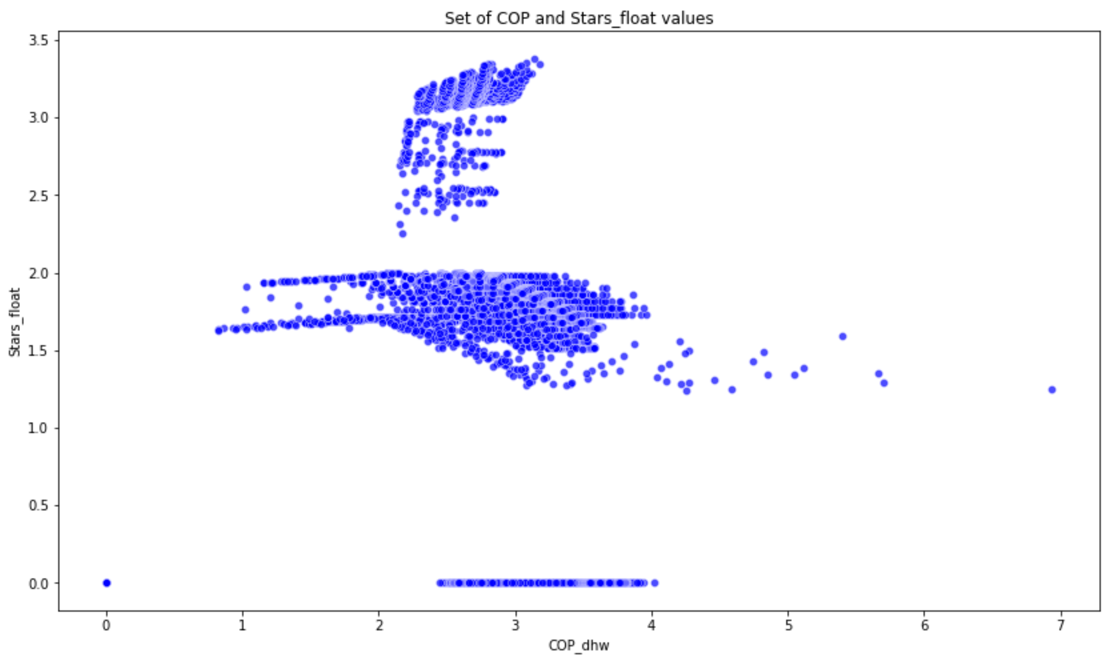
The result of the graphic's visualization 2.5.4 has tended to be mixed as a result of an earning of points on a straight line.
The problem of this metric it's flattening the requirements and how were are close to it. Also it may not be adequate for the optimisation algorithm.
It was so necessary to create what is called **"floating star notation"** in the following manner :
By the figure 5, we find different thresholds according to variables V_{40}, P_{es}, θ_{wh}, t_h, ... (which allow to have 2 or 3 stars).
These thresholds tell us if we have 2 or 3 stars or how close we are. Thus, if we are close to them, we want to have float values close to 2^- or 3^- stars. We opt for a method of projection on [0,1] as detailed as follows :

Notation :
$i \in \{P_{es}, V_{40}, \theta_{wh}, T_{ref\_V_{40}}\}$
x_i : value of i
y_c^i : *current threshold*
y_n^i : *next threshold*

- ☐ **Stars = 0**
  - Do nothing
- ☐ **Stars = 1**
  - $y_c^i = 0$ and $y_n^i = threshold_2^i$
  - if $\leq$: $X_i = min(1, max(0, \frac{y_c^i - x_i}{y_c^i - y_n^i}))$
  - if $\geq$: $X_i = min(1, max(0, \frac{x_i - y_c^i}{y_n^i - y_c^i}))$
  - $\alpha_i = \{6, 0, 3, 12\}$ [a]
  - $stars\_float = stars + \sum_{i=1}^{4} \frac{\alpha_i X_i}{\alpha_i}$
- ☐ **Stars = 2**
  - $y_c^i = threshold_2^i$ and $y_n^i = threshold_3^i$
  - if $\leq$: $X_i = min(1, max(0, \frac{y_c^i - x_i}{y_c^i - y_n^i}))$
  - if $\geq$: $X_i = min(1, max(0, \frac{x_i - y_c^i}{y_n^i - y_c^i}))$
  - $\alpha_i = \{6, 0, 3, 0\}$ [b]
  - $stars\_float = stars + \sum_{i=1}^{4} \frac{\alpha_i X_i}{\alpha_i}$
- ☐ **Stars = 3**
  - $y_c^i = threshold_3^i$
  - $X_i = min(3, \frac{y_c^i}{x_i} - 1)$
  - $stars\_float = stars + \sum_{i=1}^{4} \frac{X_i}{4}$

[a] weight to give more importance to certain values
[b] weight to give more importance to certain values

In this way, we obtain this graphic :



We can observe a better distribution of the points. The difficulty was to find a good way to distribuate these points.
Now that *stars_float* is computed, the visualization of the *stars_float* values of a product in function of the parameters as well as the solution of 1 have great interest to us :

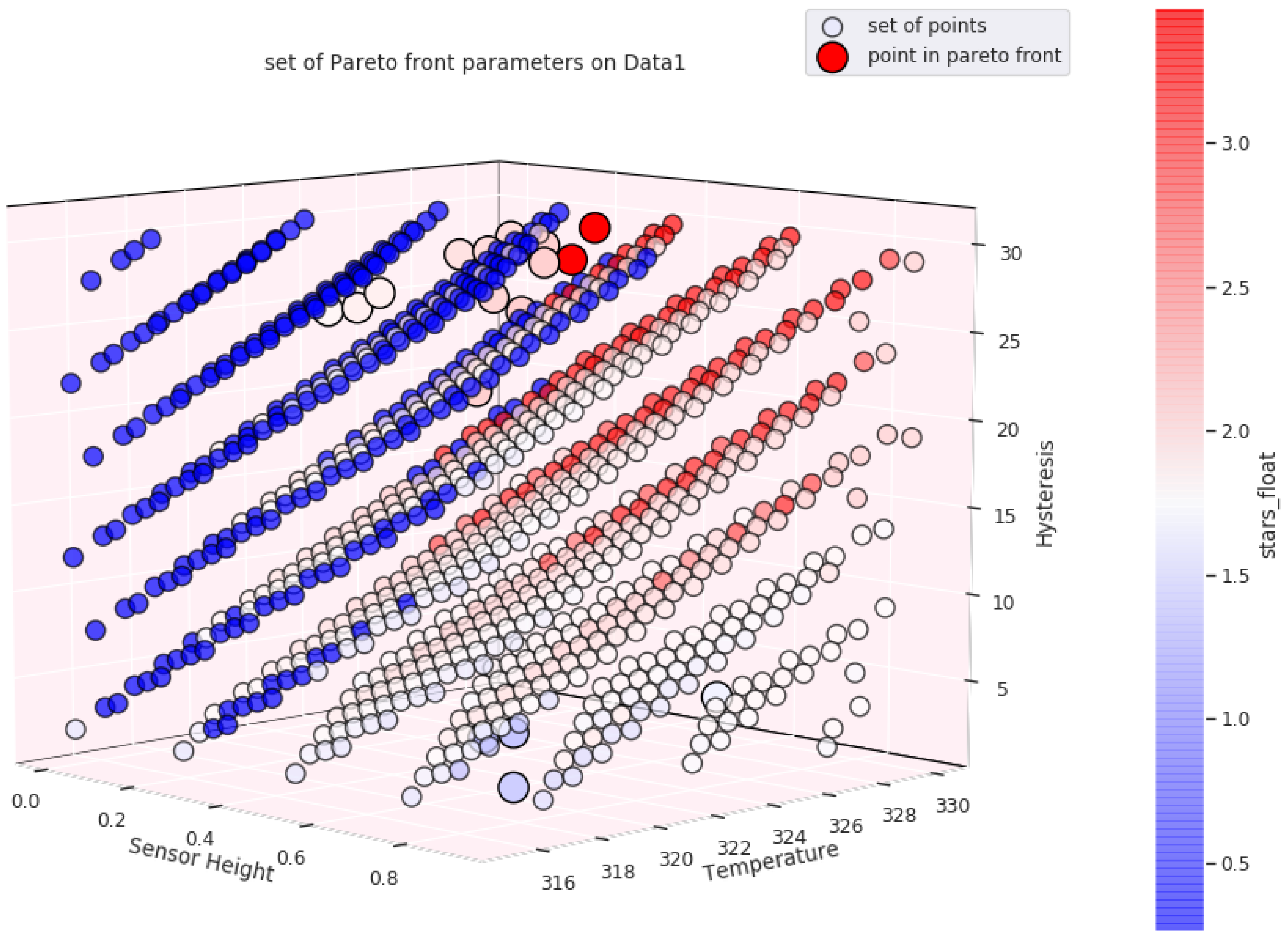


FIGURE 6 – COP_{DHW} value of IDU3FS [19] depending on T_{set}, $\Delta T_{hysteresis}$ and H_{gauge}

We observe that the parameters are an important impact on getting stars. Having a low value of sensor height and temperature provide a low stars value and having a high value of temperature and hysteresis provide a high stars value.

2.6 Multiple objectives optimization

Multi-objective optimization problems deals with conflicting objectives. This comes from the fact that we cannot say which is "best".
Here is an example :
we are interested in the following mathematical problem :

— f_m an function of objective ($COP(x)$ and $stars(x)$ for example)
— $x = (x_1, x_2, .., x_n)^T$ a vector

$$min_{x \in K}(f_m(x)), \quad m \in \{1, 2\}, \quad K\ compact$$

with constraints :

$$g_j(x) \geq 0, \quad j = 1, 2, ...n_1$$

and illustrated by :

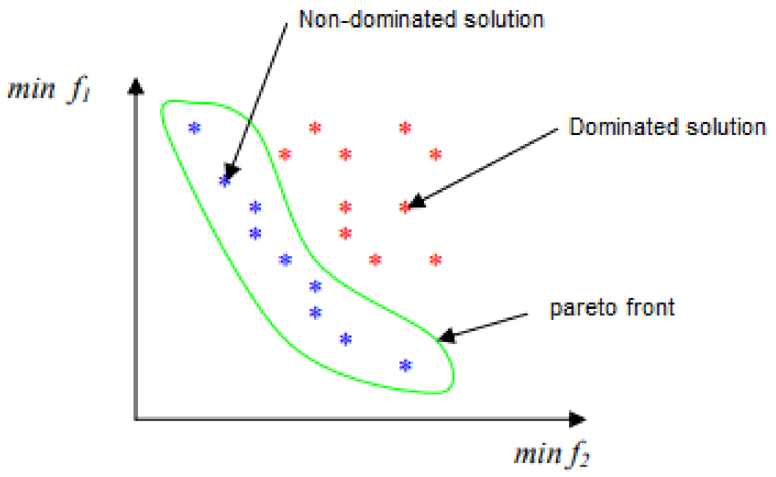


FIGURE 7 – Pareto front
Source: tel.archives-ouvertes[12]

We can see in the figure 7 the presence of *dominated solution*, *non−dominated solution* and *paretofront* defined by :

Definition 1 *: **Domination***
A solution x_i dominates x_j if both condition 1 and 2 below are true :
— *Condition 1 : x_i is no worse than x_j for all objectives*
— *Condition 2 : x_i is strictly better than x_j in at least one objective*
Mathematical notation : $x_j \preceq x_i$

Definition 2 *: **Non-dominated solution***
Among a set of solutions M, the non-dominated solutions are those that are not dominated by any member of this set. The others are so called "dominated solution"

Definition 3 *: **Pareto front***
the non-dominated set of solutions is called "pareto front"

So, to solve a problem of multiple objectives optimization defined by our equation 1, we aim to find the pareto front which holds only pareto-optimal solutions. In order to solve this complex problem, we will use metaheuristic algorithms.

Here is the pareto front for each datafiles :

FIGURE 8 – Pareto front of the datafile number 1

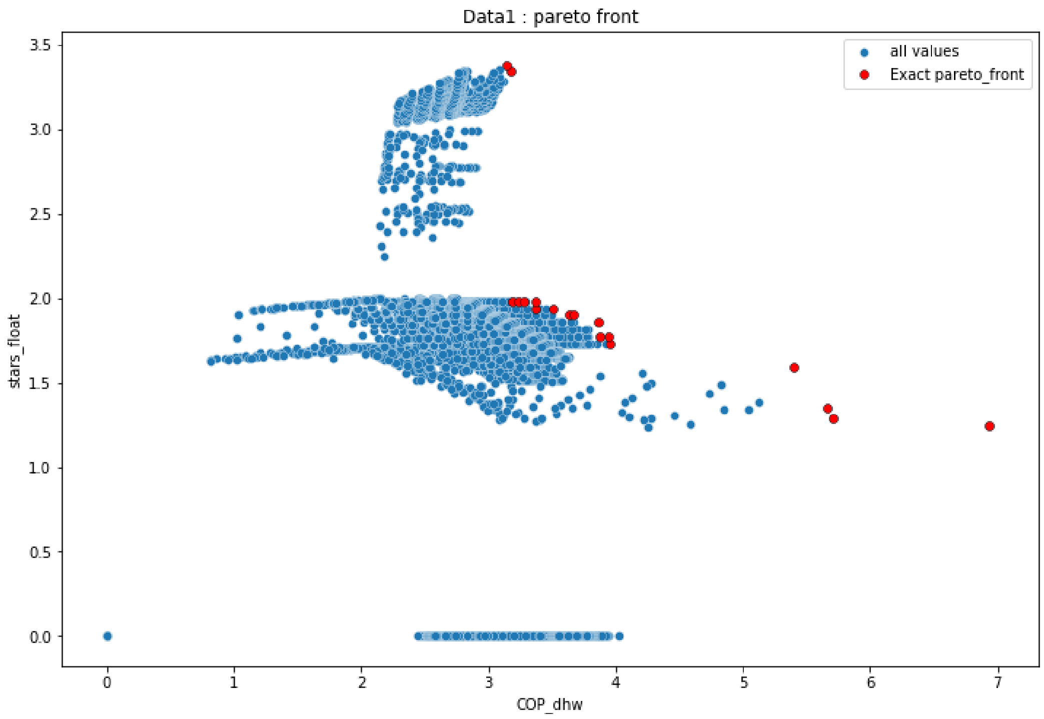


FIGURE 9 – Pareto front of the datafile number 2

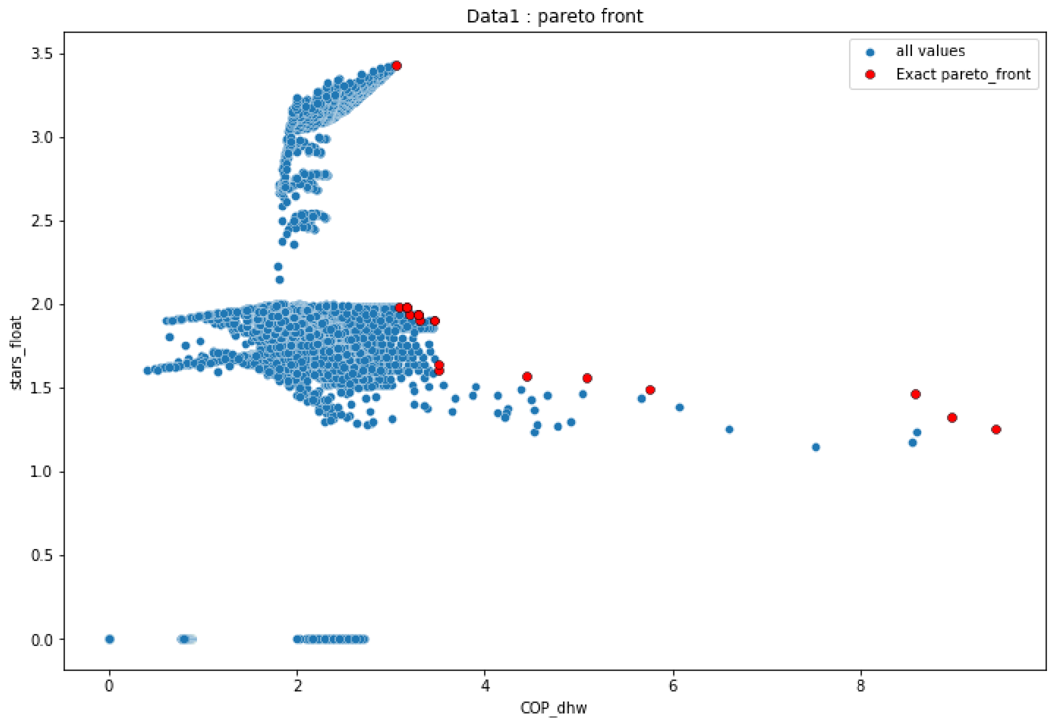


Figure 10 – Pareto front of the datafile number 3

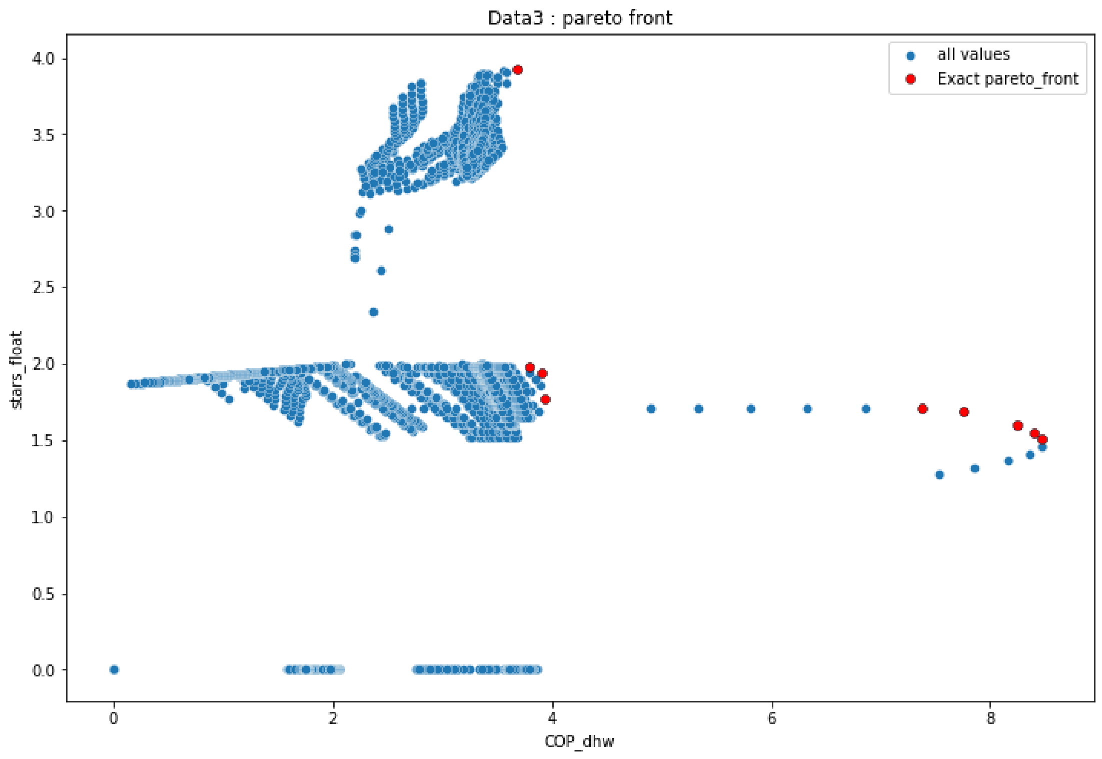
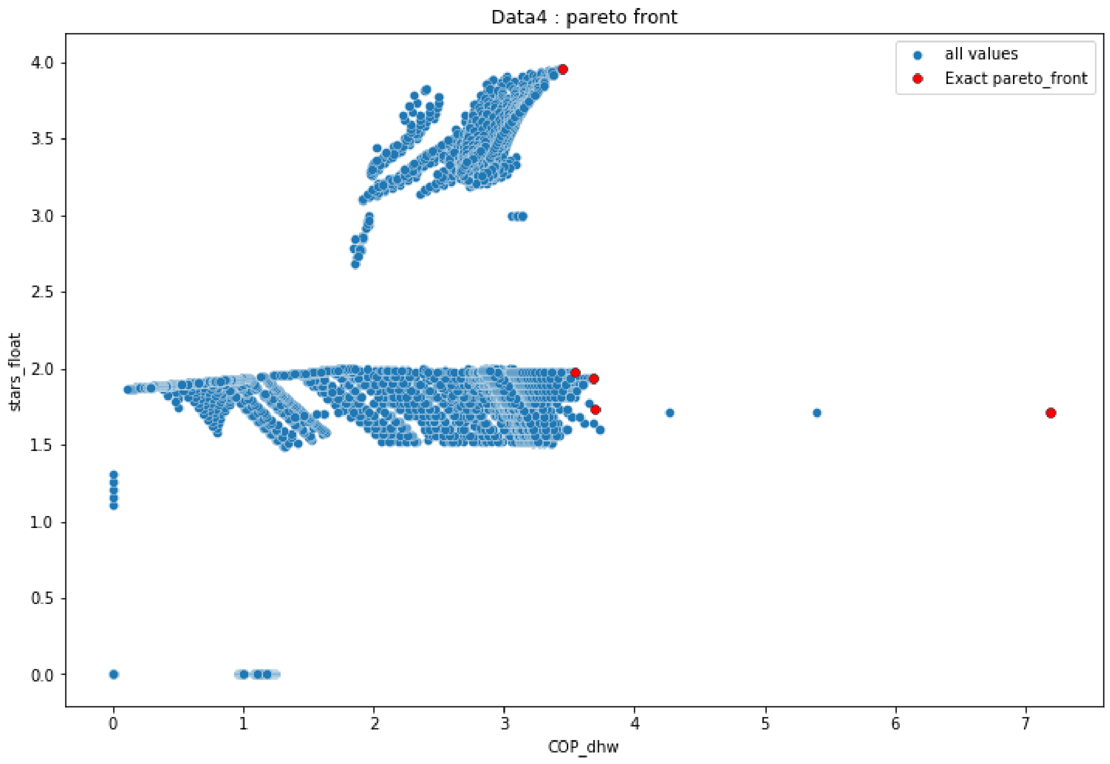


Figure 11 – Pareto front of the datafile number 4



2.7 Genetic Algorithms and Metaheuristic

As we have seen, computing the COP_{DHW} and $stars\_float$ value for only one set of parameters takes a lot of time (12 days on average in laboratory), even in a simulation (44 min on average), so it is not doable to explore all possibilites in a reasonable amount of time. Therefore we need metaheuristics[21] to wisely explore the design space. The main idea behind a genetic algorithm is to consider the set of all possible parameters as a population of individuals, each with their own attributes (the parameters) and fitness (objectives). An individual is considered to be fitter than an other individual if he has better objectives. The algorithm starts with a population of random individuals that will evolve through selection (getting us closer to a satisfying solution) and mutations (preventing us from being stuck around a non-optimal solution of our problem). In the end, there will be a best individual that will likely be a good solution for the optimization problem.
In our case, a set given of attributes $(x_1, x_2, x_3) \in \mathbb{T}_{set} \times \Delta\mathbb{T}_{hysteresis} \times \mathbb{H}_{gauge}$ is viewed as an individual with attributes (x_1, x_2, x_3) and the corresponding couple (COP_{DHW},$stars\_float$) value is its fitness.

2.7.1 Genetic algorithms

Genetic algorithms were popularised by John H. Holland [6] from 1975. They are inspired from the theory of evolution, the process of natural selection and involve randomness and it is useful to solve a problem too complex to be dealt with using classical methods.
The idea is to evolve a set of solutions to a given problem and by using them, we should be able to find reasonable solutions even without exploring or knowing all data.

The steps of an genetic algorithm could look like this[7] :

— Step 1 : Generate an initial population of random individuals (generation 0)
— Step 2 : Select the fittest individuals to breed the next generation
— Step 3 : Mate the individuals selected. We get a new set of individuals called the offspring.
— Step 4 : Apply random mutations to the the whole population (offspring included)
— Step 5 : Repeat steps 2, 3, 4 until a stopping criterion is satisfied.
— Step 6 : At the end, select the fittests individual among all generations

2.7.2 Particle swarm optimization

Particle swarm optimization is one of the most famous too methaeuristic optimization technique based on swarm, which was proposed by Eberhart and Kennedy[28] from 1995. This algorithms inspired from swarm behavior such as bird flocking in nature.
It simulates animal's social behavior and cooperative way to find food, and each member in the swarms keeps changing the search pattern according to the learning experiences of its own and other members.

The steps of an particle swarm optimization could look like this[29] :
Considering a swarm with M particles, a position vector $X_i^t = \begin{pmatrix} x_{i1} \\ x_{i2} \\ ... \\ x_{in} \end{pmatrix}$ and a velocity vector $V_i^t = \begin{pmatrix} v_{i1} \\ v_{i2} \\ ... \\ v_{in} \end{pmatrix}$ at a t iteration.
We note : $pbest_i$ the best position of the particle i and $gbest_i$ the best position of all particles.

— Step 1 : Generate an initial swarm population of random individuals (generation 0)
— Step 2 : Initialize X_i and V_i randomly, $\forall\ i \in$ M.
— Step 3 : Evaluate the fitness with X_i, $\forall\ i \in$ M
— Step 4 : Initialize $pbest_i$ with X_i and $gbest_i$ with the best fitness of all X_i, $\forall\ i \in$ M
— Step 5 : Repeat these following steps until a stopping criterion is satisfied :
— Step 6 : Update V_i^t and X_i^t $\forall\ i \in$ M
— Step 7 : Evaluate the fitness with X_i, $\forall\ i \in$ M

— step 8 : if the fitness of X_i^t is better than $pbest_i$, replace $pbest_i$ by X_i^t , $\forall$ $i \in$ M
— Step 9 : if the fitness of X_i^t is better than $gbest_i$, replace $gbest_i$ by X_i^t , $\forall$ $i \in$ M

2.8 Mating and Mutation

As described previously, there is an idea of natural selection in genetics[10] represented by :

• **Mating/Crossover** :
An individual is encoded as a sequence of genes. an individual can pass or not his gene combination to the next generation by reproducing with a partner. If that happens, the offspring will have a combination of genes from the mother and the father. So for the mating, we choose a crossover, it executes a one point crossover on the input sequence individuals represented by the probability **cx_pb** in our algorithm.

• **Mutation** :
There is a second effect that will determine the gene pool of the new generation. Due to many random circumstances, there are some genes that will just randomly change in any new individual. The probability of mutation is noted by **mut_pb**.
ind_pb represents the probability to change one specific attribute of an individual knowing that he is mutating.

2.9 Implementation

The implementation of genetic-algorithms and particle swarm optimization's algorithms will be done by the JmetalPy's librairy[9]. It is an object-oriented Python-base framework for multi-objective optimization with metaheuristic techniques and have a very large amount of choice of metaheuristic algorithms. It was created by Antonio Benitez-Hidalgo, Antonio J.Nebro, José Garcia-Nieto, ...[32] in 2006 and has been continuously evolving since then.
These avantages make it a very good choice because he had a full redesign from scratch in 2015, an easy use of parallel computing and as said before, a large amount of metaheuristic algorithms.
That's how I chose 4 "candidates" of metaheuristic algorithms (2 **genetic algorithms** and 2 **particle swarm optimization's algorithms**) and 1 algorithm of random search (**our comparative algorithm**) in JmetalPy.
These 4 "candidates" are as follows :

- $NSGA-II$ (genetic-algorithm)
- $NSGA-III$ (genetic-algorithm)
- $MOPSO$ (particle swarm optimization)
- $SMPSO$ (particle swarm optimization)

Our motivation is twofold. First, we want to compare each these multi-objective optimization's algorithms consisting in analysing the performance. Secondly, we want to study the convergence speed with the idea of number of individuals computed and we hope that the found pareto front will be close enough to the true pareto front.

2.9.1 NSGA-II

Because of its proven effectiveness, we have chosen to implement NSGA-II(Non-dominated Sorting Genetic Algorithm) whose operating principle is shown in the figure 12 and which follows the general step of genetic algorithm 2.7.1.

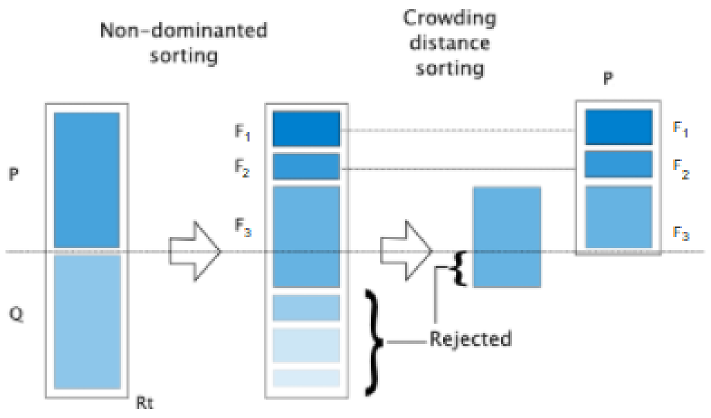


FIGURE 12 – Operating principle
Source: pymoo.org[8]

First, as described in the general step, we generate an initial population of random individuals which then, a sorting using the concept of non-domination is done. Each solution has a rank equal to the level of non-dominance (1= bests, 2 =next levels, ..., n = worsts). The reproduction consists of a tournament for the selection of parents. When two individuals of the population are chosen randomly in the population, the tournament is based on a comparison of the domination of the two individuals. At the generation t, we create a population R_t containing 2n individuals : P_t (n individuals) and Q_t (n individuals) defined as follows :

— P_t population selected at the generation t

— Q_t children population of P_{t-1} generated with crossover and mutation operators

As before, we sort R_t with the concept of non-domination. In this way, we have groups of individuals represented by nondominated fronts such as F1 which represents individuals of rank 1, F2 individuals of rank 2, ... Now, we need to reduce the number of individuals (2n) of R_t because to continue a new generation, we need a population P_{t+1} of size n.
Several cases are to expected :
While the number of individuals conserved does not exceed the size n, we retaines in this order F_1,F_2,.., F_i.For example, in the previously figure 12, F1 and F2 are fully conserved but with all the individuals of F3, we exceeds the size n. So, we have to keep a part of F3. In this case, NSGA2 use the mechanism of "crowding distance" which is a mechanism for preserving the diversity of the population as described in this figure.

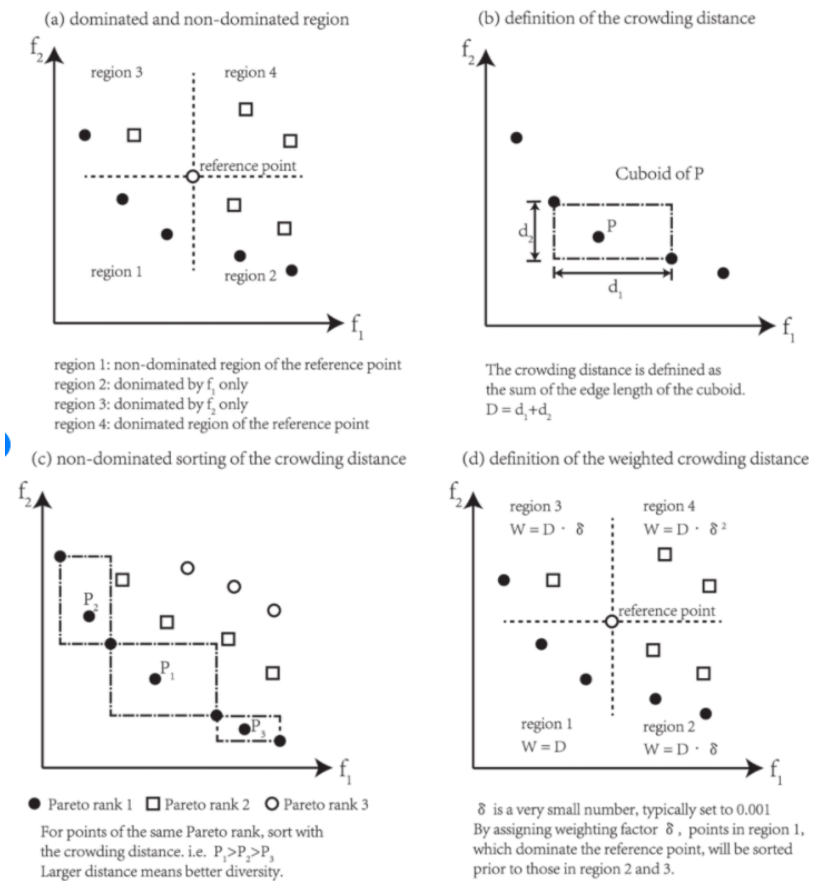

(a) dominated and non-dominated region
region 3
region 4
reference point
region 1
region 2
region 1: non-dominated region of the reference point
region 2: donimated by f_1 only
region 3: donimated by f_2 only
region 4: donimated region of the reference point
(b) definition of the crowding distance
Cuboid of P
The crowding distance is defined as the sum of the edge length of the cuboid.
$D = d_1 + d_2$
(c) non-dominated sorting of the crowding distance
● Pareto rank 1 □ Pareto rank 2 ○ Pareto rank 3
For points of the same Pareto rank, sort with the crowding distance. i.e. $P_1 > P_2 > P_3$
Larger distance means better diversity.
(d) definition of the weighted crowding distance
region 3 $W = D \cdot \delta$
region 4 $W = D \cdot \delta^2$
region 1 $W = D$
region 2 $W = D \cdot \delta$
δ is a very small number, typically set to 0.001
By assigning weighting factor δ, points in region 1, which dominate the reference point, will be sorted prior to those in region 2 and 3.

FIGURE 13 – Crowding distance
Source: [33]

This method complete the population P_{t+1}. We continue iteratively until the satisfaction of stop criteria.

2.9.2 NSGA-III

As its predecessor, NSGA-III is one of the most famous genetic algorithms which adds the idea of reference points as an improvement of NSGA-II. The both are similar, except in the selection step. Whereas NSGA-II use a comparaison operation to select and maintain diversity, NSGA-III applies well-distributed reference points to maintain diversity. In this respect, the "crowding distance" is replaced by a reference points method.
Here is an explication of reference points method with the same example than before :
As in NSGA-II, the non-dominated sorting is done which could be illustrated by this figure :

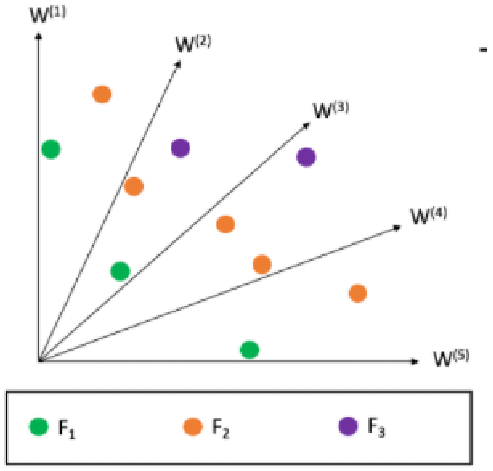


FIGURE 14 – example
Source: pymoo NSGA3[17]

We have here 3 groups F_1, F_2 and F_3 as the previously example. Now, we can imagine that we only want to retain F_1 and a part of F_2. The selection is done as follows :

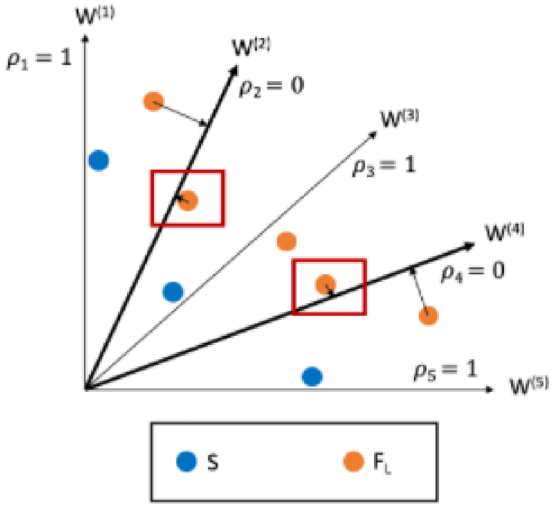


FIGURE 15 – reference points
Source: pymoo NSGA3[17]

The smallests perpendicular distance between the solution and the segment of the reference points ($W^{(1)}$, ..., $W^{(i)}$) is surviving.

Disavantage : the number of objectives is limited (Used for solving two and three objective optimization problems)

2.9.3 MOPSO and SMPSO

$MOPSO$ and $SMPSO$ are one of the most effectiveness particle swarm optimization whose principle resides in the limitation of the velocity of particles.$MOPSO$ (multi-objective particle swarm optimization) is the basic algorithm of particle swarm and $SMPSO$ (Speed-constrained Multi-objective) which adds a velocity constriction procedure is an update of $MOPSO$.
In the particle swarm optimization algorithms, each canditate solution is a particle (population called "swarm" is represented by $X = (x_1, ..., x_i, ..x_n)^T$.
A particle $\vec{x_i}$ will be update at the generation t with the formula :

$$\vec{x_i}(t) = \vec{x_i}(t-1) + \vec{v_i}(t)$$

where $\vec{v_i}$ is the velocity represented by :

$$\vec{v_i}(t) = w.\vec{v_i}(t-1) + C_1.r_1.(\vec{x}_{p_{best}} - \vec{x_i}) + C_2.r_2.(\vec{x}_{g_{best}} - \vec{x_i})$$

with w the inertia weight of the particle, $\vec{x}_{p_{best}}$ and $\vec{x}_{g_{best}}$ as seen before [29], r_1 and r_2 two uniformly random numbers in [0,1] and C_1,C_2 two positive constant parameters which are acceleration coefficients.

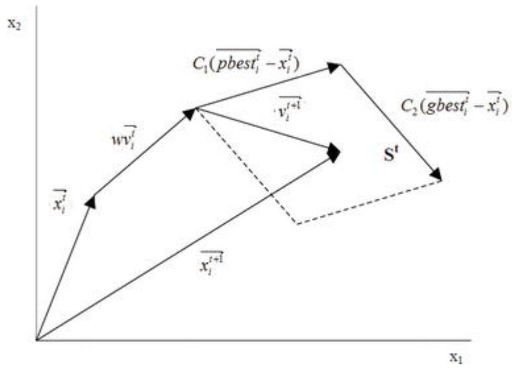


FIGURE 16 – $\vec{x_i}(t+1)$
Source: [26]

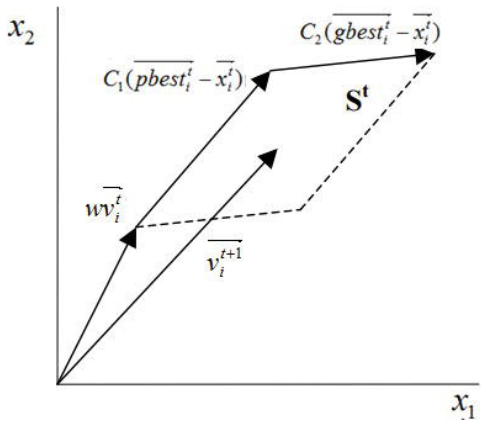

x_2
$C_2(\overline{gbest_i^t - x_i^t})$
$C_1(\overline{pbest_i^t - x_i^t})$
S^t
$\overline{wv_i^t}$
$\overline{v_i^{t+1}}$
x_1

FIGURE 17 – $\vec{v_i}(t+1)$
Source: [26]

The algorithms use the crowding distance of NSGA-II to keep the best particles.
As SMPSO is an update of MOPSO, we'll go looking into some differences between both.
First, the possible values of C_1 and C_2 may take. In $MOPSO$, C_1 and $C_2 \in [1.5, 2.0]$ and in $SMPSO$, C_1 and $C_2 \in [1.5, 2.5]$ to control the effect of $\vec{x}_{p_{best}}$ and $\vec{x}_{g_{best}}$. An other difference is the mutation operators. $MOPSO$ use a conbination of uniform (30 %) and non-uniform (30%) mutation to the particle swarm but $SMPSO$ use a polynomial mutation.

2.10 Operator

All these algorithms cited previously are already implemented in JmetalPy. They are defined by different class which requires various parameters :

Class $NSGA2$ and $NSGA3$ require :
- **Class Problem** : Problem
- **population size** : int
- **Class Mutation** : IntegerPolynomalMutation(probability=mut_pbs).
- **Class Crossover** : IntegerSBXCrossover(probability=cx_pbs).
- **Class Selection** : BinaryTournamentSelection(MultiComparator([FastNonDominatedRanking.get_-comparator(), CrowdingDistance.get_comparator()])).
- **Class Termination_criterion** : StoppingByQualityIndicator(problem,InvertedGenerationalDistance().
- **reference_directions** : UniformReferenceDirectionFactory() (NSGA-III)

.
and Class $MOPSO$ and $SMPSO$ require :
- **Class Problem** : Problem
- **Swarm_size** : int
- **Class Mutation** : FloatPolynomalMutation(probability=mut_pbs) (SMPSO).
- **Class Uniform_mutation** : UniformMutation(probability=mut_pbs,perturbation) (MOPSO).
- **Class Non_uniform_mutation** : NonUniformMutation(probability=mut_pbs,perturbation) (MOPSO).
- **Class Termination_criterion** : StoppingByQualityIndicator(problem,InvertedGenerationalDistance().
- **epsilon** : float (MOPSO)

2.10.1 Mutation

- **Uniform Mutation** : This operator replaces a random gene from our chromosome (x_i for example) and assing with a uniform random value selected between the upper and lower bounds for that gene.
- **Non-Uniform Mutation** : this operator either lower the mutation rate as the population gets fitter or make the mutations smaller as time progresses.

2.10.2 Crossover

SBXCrossover(simulated binary crossover) simulates the one-point crossover operator of the binary-coded GAs represented in this figure :

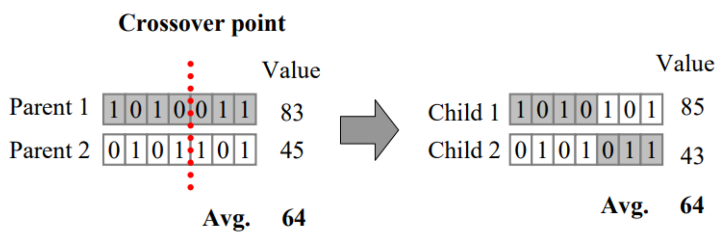


FIGURE 18 – One-point Crossover
Source: [14]

The properties of SBXCrossover are :
— Gene values of children have same distance from the average gene value of parents
— Each point of the chromosome has the same probability to be selected as a crossover point
— children are more likely to be near the parents
When two parents x_1 and x_2 are selected, an offspring can be computed as in this formula :

$$x_1^{offspring} = \frac{1}{2}[(1+\beta)x_1 + (1-\beta)x_2]$$

$$x_2^{offspring} = \frac{1}{2}[(1-\beta)x_1 + (1+\beta)x_2]$$

with :

$$\beta = \begin{cases} (2u)^{\frac{1}{n_{index}+1}} & if\ u \leq 0.5 \\ (\frac{1}{2(1-u)})^{\frac{1}{n_{index}+1}} & else \end{cases} \tag{2}$$

n_{index} is the distribution index and $u \in [0,1[$. When n_{index} is large, children are close to the parents and conversely.

2.10.3 Termination criterion

We opted for Inverted Generational Distance (IGD) as a criteria to stop our algorithm. This measure use the true pareto front as a reference and compare each elements of the true pareto front with the pareto front returned by our algorithm.
This measure is defined by :

$$IGD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n}$$

where n is the number of elements in the true pareto front and d_i, the euclidean distance between the true pareto front's points and the nearest points of the pareto front found by our algorithm.
When $IGD = 0$, all the points generated by our algorithm are in the true pareto front. So, we chose an $IGD \leq 0.001$ which seem to be very close of the true pareto front.

2.10.4 Problem

The class problem represents the implementation of the problem that we want to optimize. It have a function evaluate() which requires an individual and it returns the objectives values and constraints values of this individual.

2.10.5 Reference point

Generate evenly spaced numbers over a specified edges of our objectives problem unitary plan as follows :

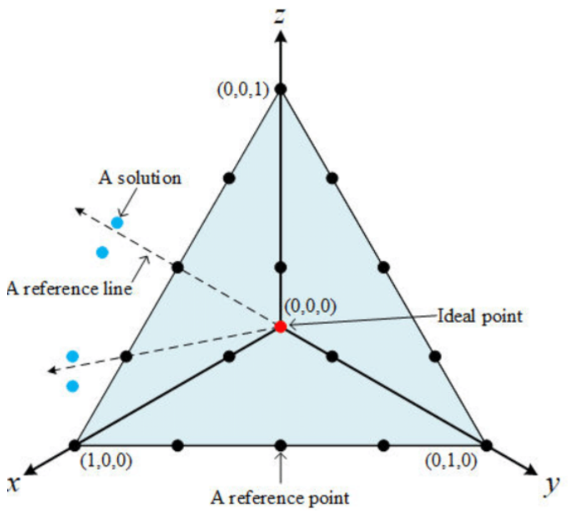


FIGURE 19 – The reference points of NSGA-III in three-objective problem
Source: [31]

2.11 Evaluating parameters and algorithms

Once implementation of algorithms are done, we can run them on the discretized dataset to solve our multi-objective problem 1. The objective is to find the algorithm and its parameters that converge quicker to that best known individual.

2.11.1 Parameters of algorithms

Each algorithm need a specific set of parameters to be runned :
— NSGA2 : paramater p = [problem, seeds, mut_pbs, index_pbs , n_inits, cx_pbs]
— NSGA3 : parameter p = [problem, seeds, mut_pbs, index_pbs , n_inits, cx_pbs]
— MOPSO : parameter p = [problem, seeds, mut_pbs, perturbation,n_inits]
— SMPSO : parameter p = [problem , seeds, mut_pbs, index_pbs, n_inits]
Where n_inits is the number of individuals in the population and seed is a number that defines a unique scenario for a pseudo-random sequence.

We want to decide on the overall best set of parameters for a given algorithm `alg`. To do this, the first goal is to know an "interval" of parameters where the sucess rate is better, we take so a big set of parameters with a small number of seed :

— `problem` : a set of 1 problem
— `seeds` : a set of 10 different seeds
— `datafiles` : a set of 1 datafile to run the algorithms on
— `n_inits` : a set of 3 different values for the parameter `n_inits` :`n_inits` $= \{50, 150, 250\}$
— `cx_pbs` : a set of 11 different values for the parameter `cx_pb` : `cx_pbs` $= \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$
— `mut_pbs` : a set of 11 different values for the parameter `mut_pb` : `mut_pbs` $= \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$
— `index_pbs` : a set of 8 different values for the parameter `index_pb` : `ind_pbs` $= \{10, 30, 50, 200, 300, 500, 1000, 50000\}$
— `perturbation` : a set of 11 different values for the parameter `perturbation` : `perturbation` $= \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$

We could then run our algorithm with every set of parameters (plus seeds and datafiles) from the set `params`. This would make for each algorithms, a total of :
— NSGA2 : $10 \times 1 \times 11 \times 8 \times 3 \times 11 = 29040$ executions
— NSGA3 : $10 \times 1 \times 11 \times 8 \times 3 \times 11 = 29040$ executions

— MOPSO : $10 \times 1 \times 11 \times 1 \times 3 \times 6 = 1980$ executions
— SMPSO : $10 \times 1 \times 11 \times 8 \times 3 = 2640$ executions

This step allows to do a pre-treatment to know an interval of the best parameters of each algorithms. We observed that :

— index_pbs not have much influence on the sucess rate[4] of the convergence for $NSGA2$ and $NSGA3$
— the rate sucess is better when n_inits = 250 for $NSGA2$ and $NSGA3$ or n_inits = 50, 150 for $MOPSO$ and $SMPSO$

Moreover, We too studied the influence of mut_pbs and cx_pbs on the sucess rate.
Here is the result :

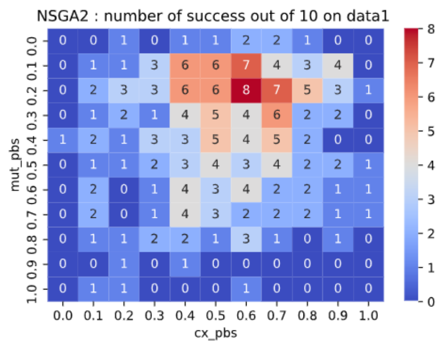


FIGURE 20 – NSGA2 : number of success out of 10 on data1

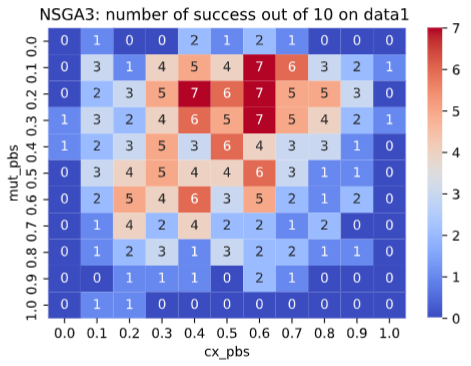


FIGURE 21 – NSGA3 : number of success out of 10 on data1

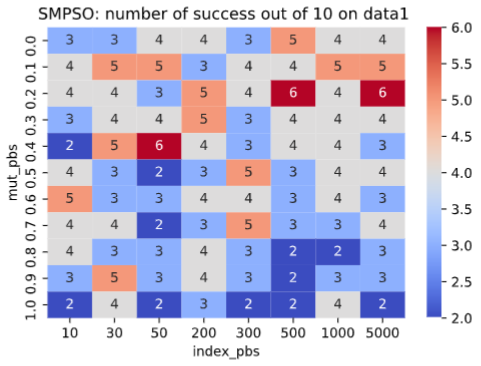


FIGURE 22 – SMPSO : number of success out of 10 on data1

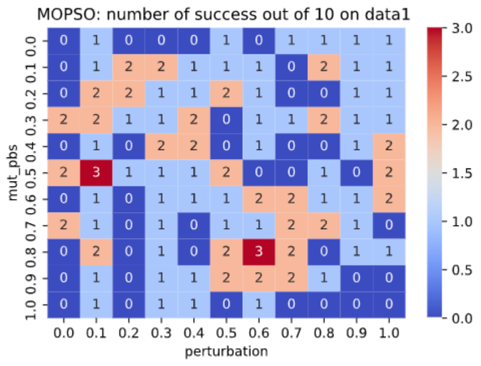


FIGURE 23 – MOPSO : number of success out of 10 on data1

Thanks to this analysis, we can observe high number of sucess aeras for each algorithms. These aeras allows to select a small interval of possible values and to discretize better our set of parameters. Because of the weakly convergence on $MOPSO$, I decide to remove it.
We can now reduce considerably the number of execution's algorithms by selecting :

— `problem` : a set of 1 problem
— `seeds` : a set of 100 different seeds
— `datafiles` : a set of 1-4 datafiles to run the algorithms on
— `n_inits` : a set of 3 different values for the parameter `n_inits` :`n_inits` $= \{50, 150\}$ (PSO) and $\{250\}$ (NSGA)

4. Sucess rate : convergence with an IGD ≤ 0.001

— `cx_pbs` : a set of 7 different values for the parameter `cx_pb` : `cx_pbs` $= \{0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.70\}$
— `mut_pbs` : a set of 7 different values for the parameter `mut_pb` : `mut_pbs` $= \{0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45\}$ (NSGA)
— `mut_pbs` : a set of 9 different values for the parameter `mut_pb` : `mut_pbs` $= \{0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.5, 0.6\}$ (PSO)
— `index_pbs` : a set of 8 different values for the parameter `index_pb` : `ind_pbs` $= \{10, 30, 50, 200, 300, 500, 1000, 50000\}$ (PSO) or $\{30\}$ (NSGA)

We obtain the same graphic than before with 100 seeds :

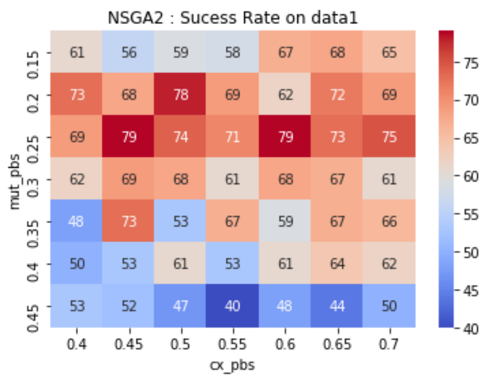


FIGURE 24 – Sucess rate with NSGA2 on the file 1

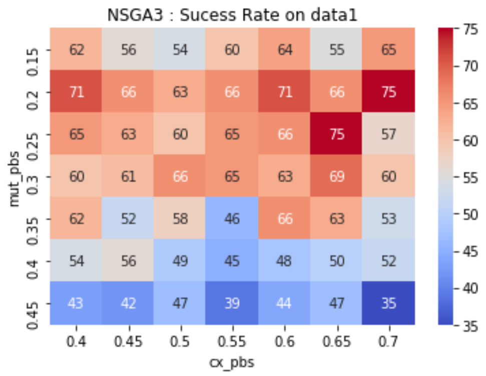


FIGURE 25 – Sucess rate with NSGA3 on the file 1

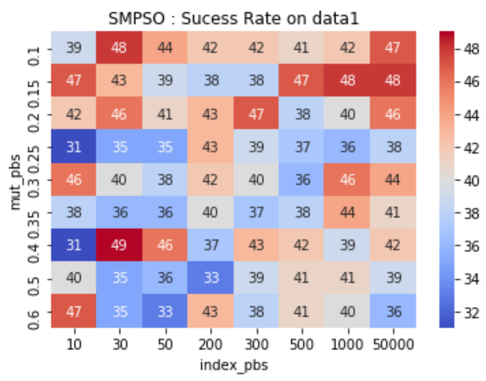


FIGURE 26 – Sucess rate with SMPSO on the file 1

There is an high influence of parameters on the sucess rate. the extreme values of mut_pbs close to 1.0 or 0.0 have a lower sucess rate than the median values. cx_pbs meanwhile, it influences more weakly on the sucess rate than mut_pbs.

Let's increase now the number of datafile for an indepth analysis : Instead of looking the sucess rate of $IGD \leq 0.001$, we focus on the number of point found of the true pareto front in the following way :

- Data number 1 has 18 points on the true front pareto, so we want to find at least 17 points
- Data number 2 has 18 points on the true front pareto, so we want to find at least 17 points
- Data number 3 has 9 points on the true front pareto, so we want to find at least 8 points
- Data number 4 has 5 points on the true front pareto, so we want to find at least 3 points

The sucess rate become :
Notation : file 1 : IDU3FS cycle L, file 2 : IDU3FS cycle M, file 3 : TWH3001 cycle L and file 4 : TWH3001 cycle M

Figure 27 – Sucess rate with NSGA2 on the file 1

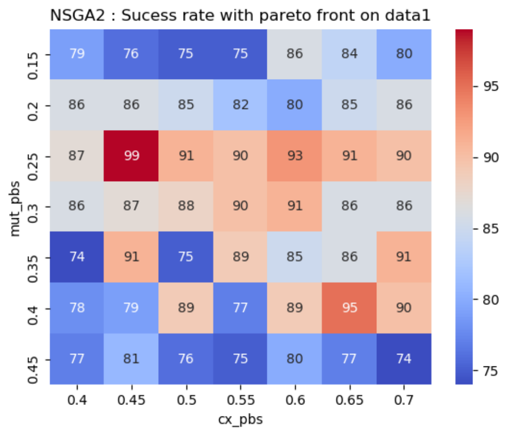


Figure 28 – Sucess rate with NSGA2 on the file 2

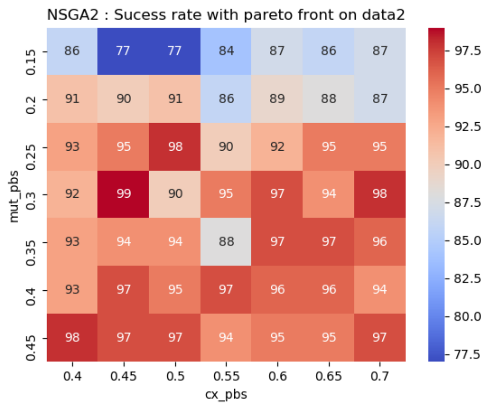


Figure 29 – Sucess rate with NSGA2 on the file 3

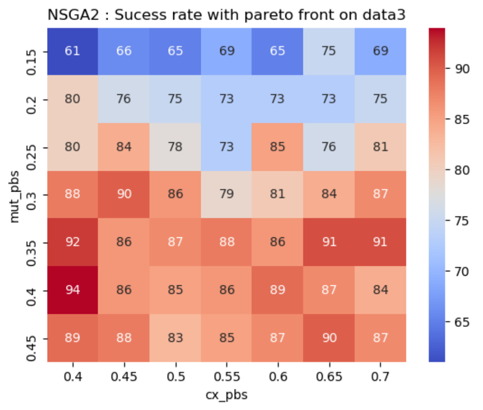


Figure 30 – Sucess rate with NSGA2 on the file 4

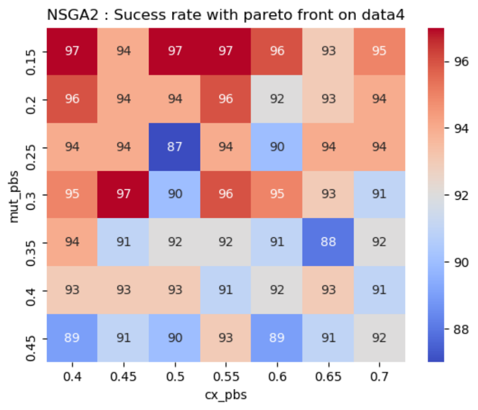


Figure 31 – Success rate with NSGA3 on the file 1

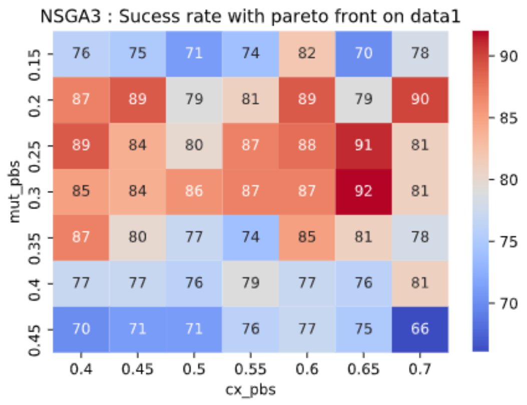


Figure 32 – Sucess rate with NSGA3 on the file 2

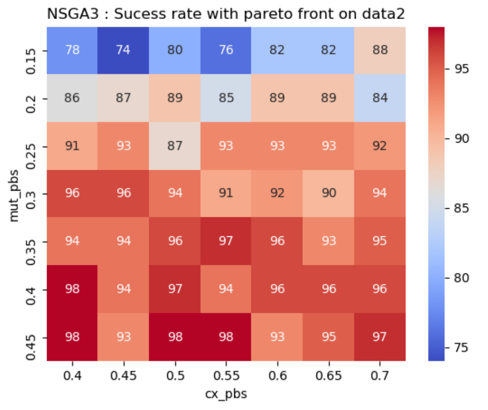


Figure 33 – Sucess rate with NSGA3 on the file 3

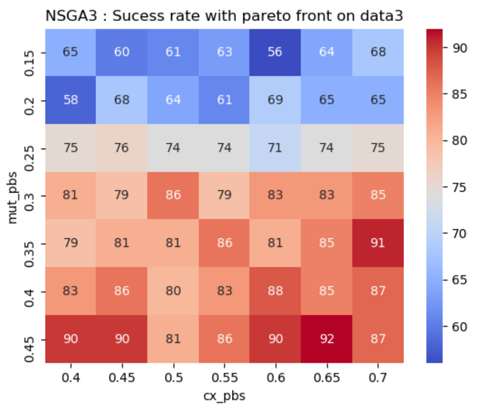


Figure 34 – Sucess rate with NSGA3 on the file 4

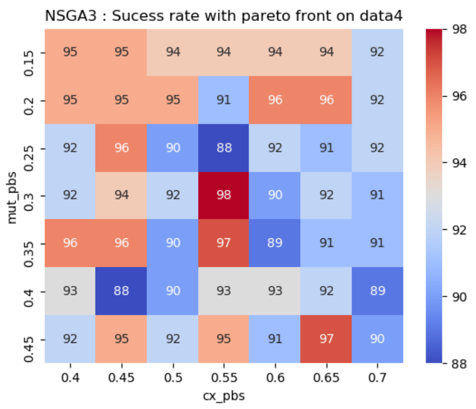


Figure 35 – Sucess rate with SMPSO on the file 1

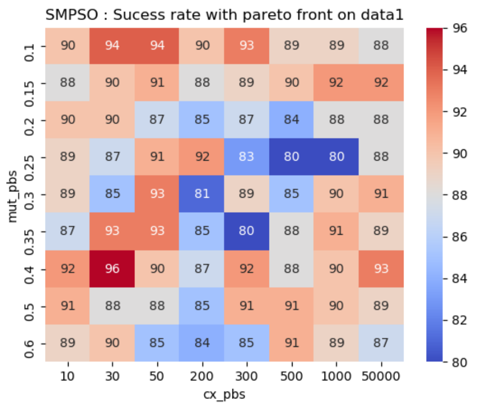


Figure 36 – Sucess rate with SMPSO on the file 2

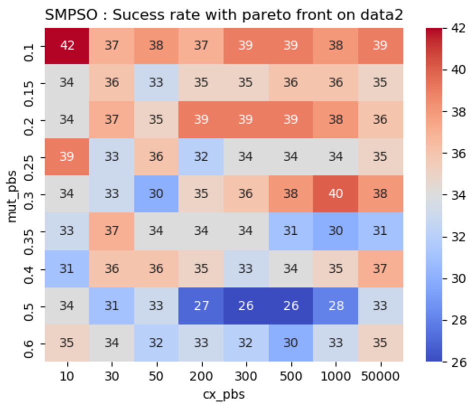


Figure 37 – Sucess rate with SMPSO on the file 3

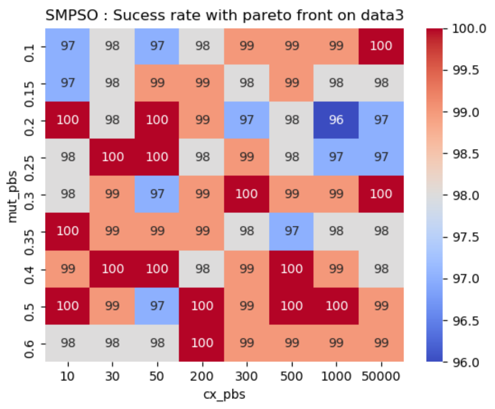
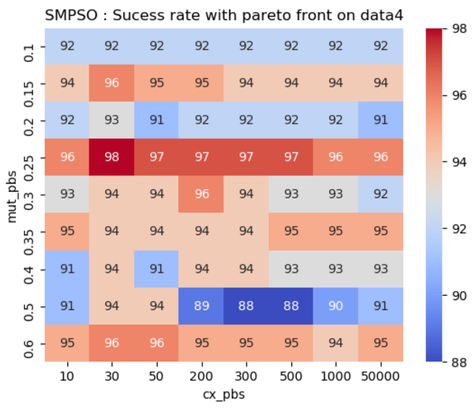


Figure 38 – Sucess rate with SMPSO on the file 4



We can see different phenomens :
First, for each algorithms, different various areas of high sucess rate for each datafiles, making it very difficult choice for the selection of best parameter.
Secondly, SMPSO have a weakly sucess rate for the file number 2 (IDU3FS-cycle M)36

2.12 Classification :

If we consider an algorithm `alg` running on a data file `datafile` with a set of parameters p = [`cx_pb, mut_pb, ind_pb`], we want to define a meaningful metric to evaluate the efficiency of the algorithm. The one we chose is the number of unique individuals that were evaluated during the execution of the algorithm. This number will never be the same because of the random nature of genetic algorithms, therefore we can decide on a seed to ensure the reproductibility of the measure.
For a given seed `seed`, we can store the quantity `unique_evals`(`alg`, `datafile`, p, `seed`) in a database. Now, we can repeat this process for a big amount of seeds (we did it with 100 different seeds) to get a better idea of the overall efficiency of this set of parameters. In the end, we store a quantile (90%) of `unique_evals` over all these executions of the algorithm and make a ranking.
We can use this notation : $q_{90}(\texttt{alg},\ \texttt{datafile},\ p)$ (this value depends on the set of seeds that we used).

First option : a "minmax" classification For each set of parameters p, we store its worst performance over all the datafiles, $q_{90}(\texttt{alg},\ \texttt{datafile}_{i_{\text{worst}}},\ p) = \max_{i=1,2} q_{90}(\texttt{alg},\ \texttt{datafile}_i,\ p)$. Then we take the parameter $p_{j_{\text{best}}}$ that minimizes the worst quantiles : $p_{j_{\text{best}}} = \min_{j=1,\dots 175}\left(\max_{i=1,2} q_{90}(\texttt{alg},\ \texttt{datafile}_i,\ p_j)\right)$. This way, we are guaranteed that the chosen parameter will perform at least as good as expected in all cases : we minimize the risk.

Second option : a "points" classification For each data file `datafile`, we classify the sets of parameters from the one with the smallest (best) quantile to the one with the biggest (worst) quantile. With this ranking, each set of parameters gets a number of points equal to its position in the list (a high quantile is worth more points). We then add up the points of each set of parameters accross all the data files to get a final ranking, and we choose the one which got the fewest points. This way, we know that this set of parameters is the best in average (it could still have performed quite badly on some data files).

Third option : a "sucess rate" classification For each data file `datafile`, we count the convergence's sucess rate of each sets of parameters as follows :

- α = sucess rate of n points of the true pareto front found
- β = sucess rate of n-1 points of the true pareto front found

and the score become :

$$score = 2\alpha + \beta$$

We then add up the score of each set of parameters accross all the data files to get a final ranking, and we choose the one which got the fewest score. This way, we know that this set of parameters is the best in average for the convergence.

2.12.1 General comparaison

We can sum up all the classification with tables 2.12.1. We computed further rankings in order to complete the uncolored cells.

NSGA2 parameters	TWH-L	TWH-M	IDU3FS-L	IDU3FS-M	minmax	points	sucess rate
$p_1 = (0.25, 0.6)$	4st	7st	2st	23st	1st (3552)	1st	10st
$p_2 = (0.3, 0.4)$	5st	14st	17st	20st	2st (3640)	3st	11st
$p_3 = (0.3, 0.5)$	12st	9st	1st	8st	3st (3768)	2st	9st
$p_4 = (0.3, 0.45)$	16st	15st	3st	22st	4st (3805)	3st	8st
$p_5 = (0.4, 0.65)$	22st	26st	27st	41st	10st (3864)	9st	1st

FIGURE 39 – Best parameters for NSGA2 (notation p_i) through various classifications

NSGA3 parameters	TWH-L	TWH-M	IDU3FS-L	IDU3FS-M	minmax	points	sucess rate
$p_1 = (0.3, 0.5)$	22st	1st	4st	17st	1st (3923)	1st	1st

FIGURE 40 – Best parameters for NSGA3 (notation p_i) through various classifications

SMPSO parameters	TWH-L	TWH-M	IDU3FS-L	IDU3FS-M	minmax	points	sucess rate
$p_1 = (0.35, 30, 50)$	16st	13st	121st	83st	1st (3766)	12st	5st
$p_2 = (0.25, 500, 50)$	26st	15st	59st	56st	2st (3774)	3st	8st
$p_3 = (0.25, 200, 50)$	29st	14st	96st	56st	3st (3786)	8st	7st
$p_4 = (0.25, 1000, 50)$	20st	18st	45st	44st	4st (3791)	1st	6st
$p_5 = (0.25, 5000, 50)$	14st	20st	46st	50st	5st (3795)	2st	1st

FIGURE 41 – Best parameters for SMPSO (notation p_i) through various classifications

.

We can observe for NSGA2 :

- The worst quantiles are related to the TWH-L files
- The classification minmax and points are similar for ranking
- Sucess rate and others classification are really different

We can observe for NSGA3 :

- only one parameter verifies 90% of sucess rate for all datafiles

And we can observe for SMPSO :

- Each classification not give the same ranking
- It is more difficult to choice the best parameter

Application : classification on datafiles For each of our algorithms on each products, we keep best parameters of the "minmax" classification 2.12.1. We got the following violinplots for NSGA2, NSGA3 and SMPSO for NSGA-II. These values are always lower than 4000 because we did not allow our algorithms to explore more than half of the data sets (if we have to explore half the data to find the best individual, it means that we are doing worst than pure randomness...)

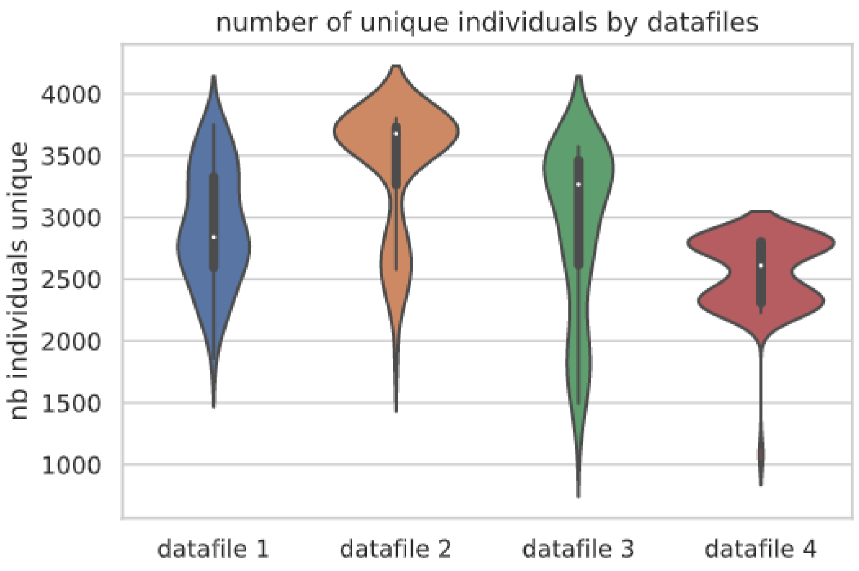


Figure 42 – NSGA2 : violinplots for each datafiles

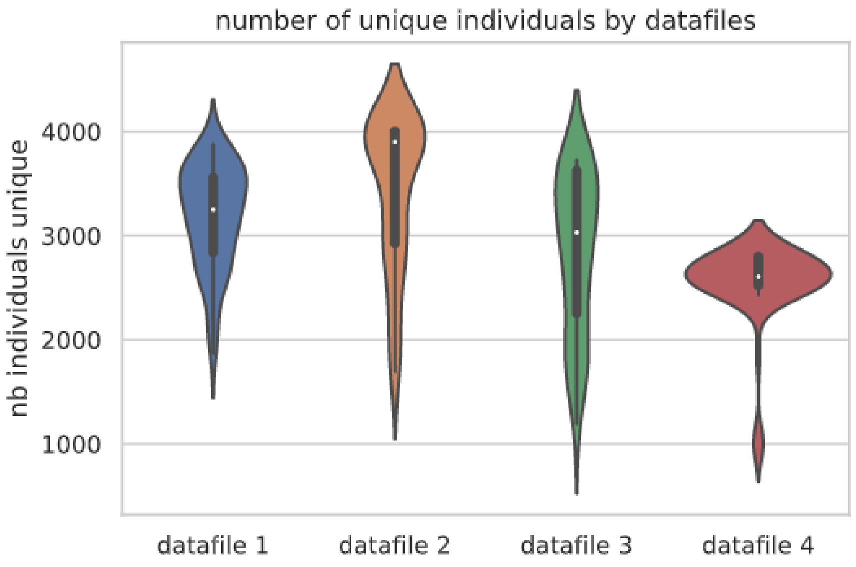


Figure 43 – NSGA3 : violinplots for each datafiles

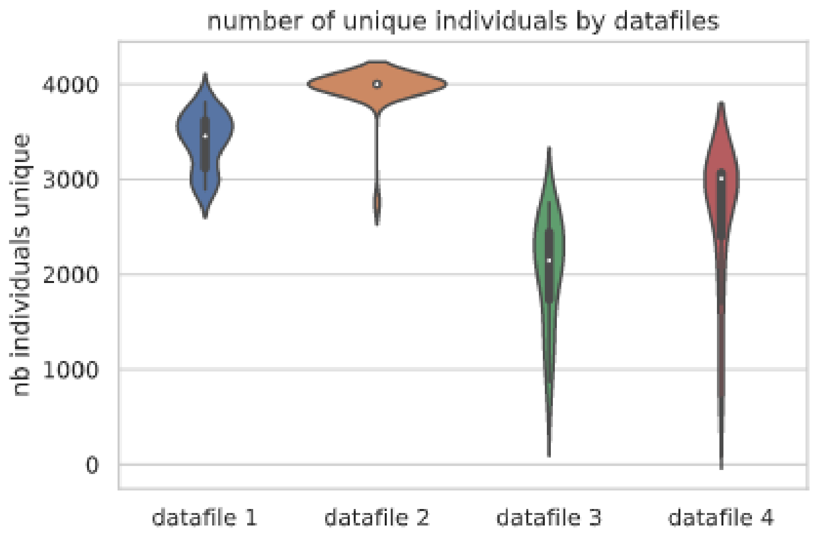


FIGURE 44 – SMPSO : violinplots for each datafiles

We remark that :

- On average, the number of unique individuals are lowest with NSGA2.
- Different forms of the violinplots for each datafiles
- Often, values of individuals unique are between 1500 and 4000.
- SMPSO with the datafile number 2 : problem of convergence related to the figure 36

2.13 Random Search

We want to compare a random search algoritm with metaheuristic algorithms. In this respect, we run the random search algorithms with 1000 seeds for each datafiles.
The random search follows the hypergeometric law :

$$X \sim \mathbf{H}(N, n, p)$$

with N : populatin size, n : sample size, p : the probability of the issue.
And :

$$P(X = k) = \frac{\binom{Np}{k}\binom{N(1-p)}{n-k}}{\binom{N}{n}}$$

In our case :

- N_i : population size for each datafile
- n_i : 4000 for each datafile
- p_i : probability to found k point of the true pareto front out of total points k^i_{total}

k varies from 1 to k^i_{total} for each datafiles
We can visualize theorical probability and experimental probability from randomsearch algorithm with 1000 seeds : $P(X \geq k)$

. .

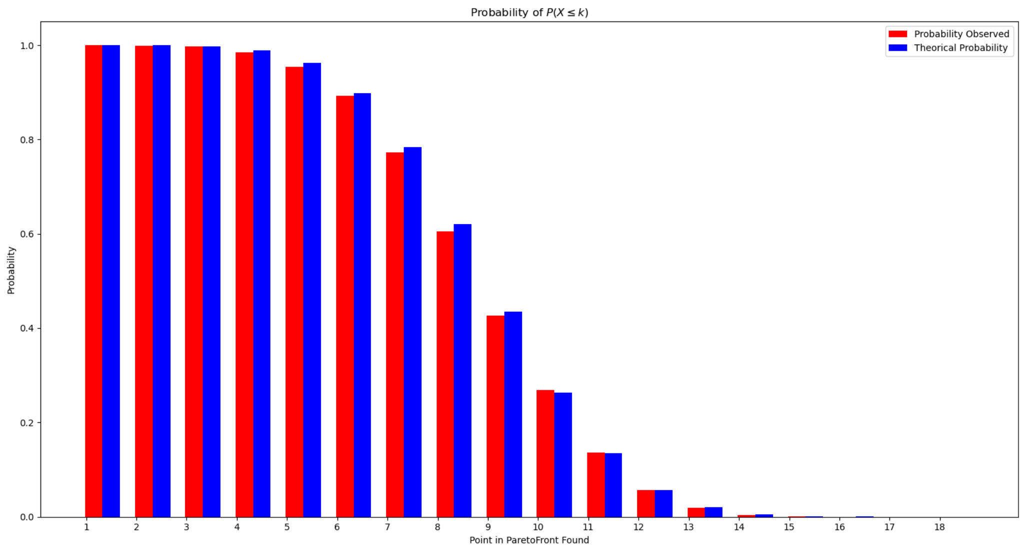


Figure 45 – Randomsearch on IDU3FS cycle L

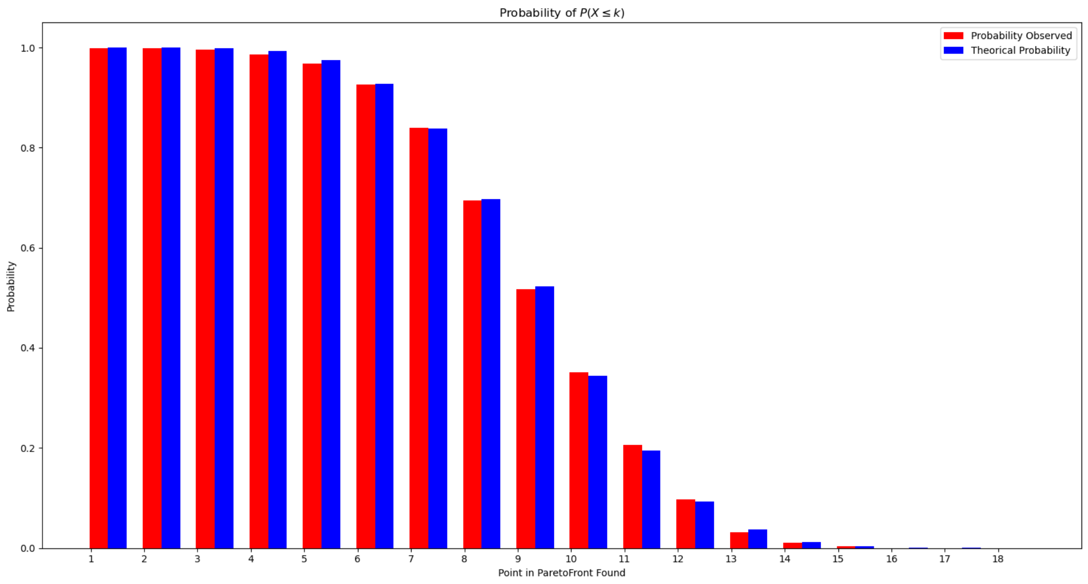


Figure 46 – Randomsearch on IDU3FS cycle M

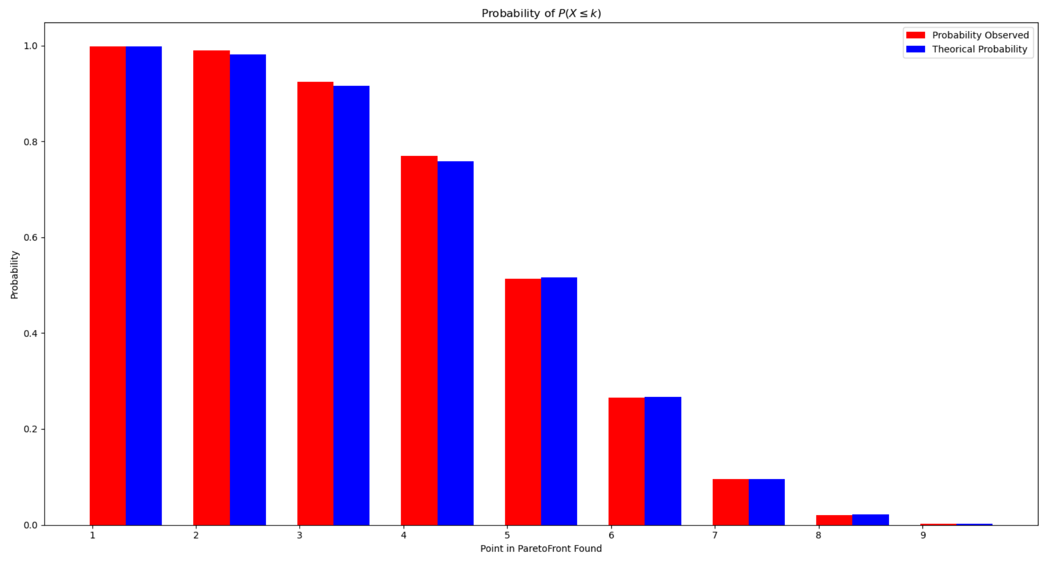


Figure 47 – Randomsearch on THW3001 cycle L

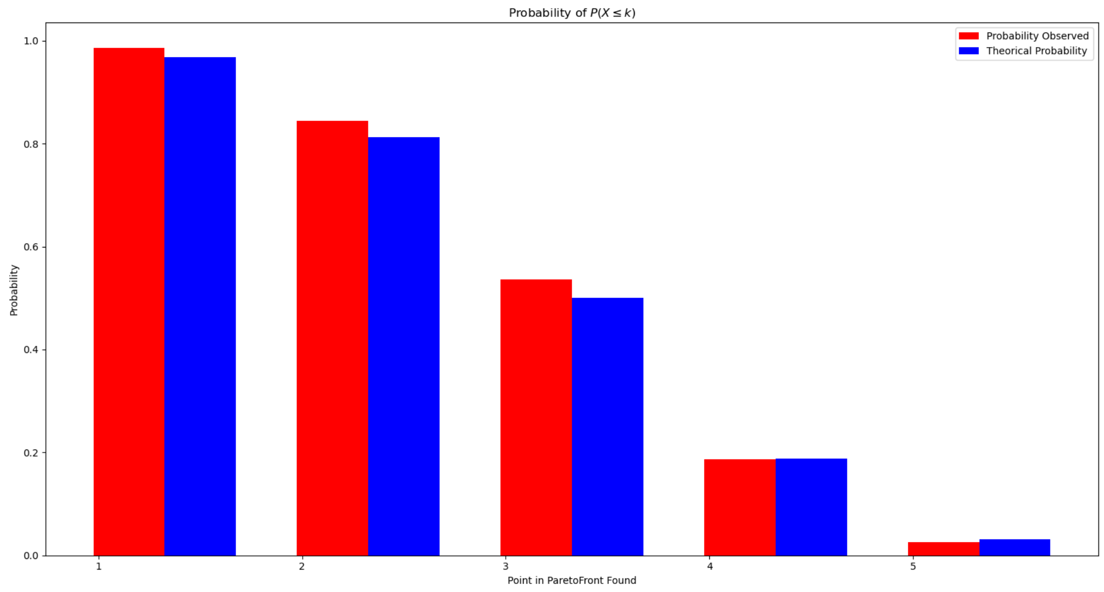


Figure 48 – Randomsearch on THW3001 cycle M

Found all the point in the true front pareto have a very low probability with a randomsearch algorithm and we can find the half in more or less 55% of cases.

2.13.1 Comparaison with our algorithm

We make a comparaison between Randomsearch and our 2 genetic algorithms.
The following graphics show the probability observed to found $k^i_{total} - 1$[5] for each datafiles :

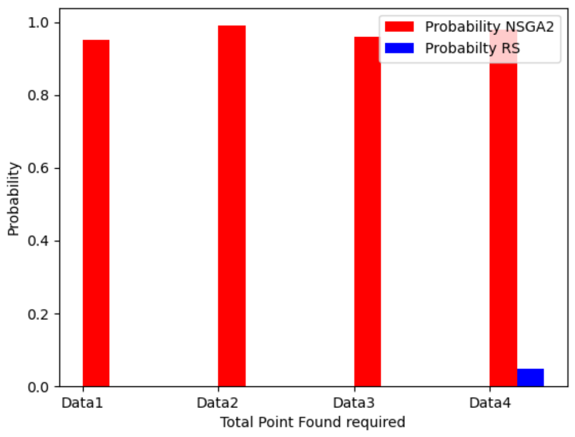


FIGURE 49 – $NSGA2 : P(X \geq k^i_{itotal} - 1)$

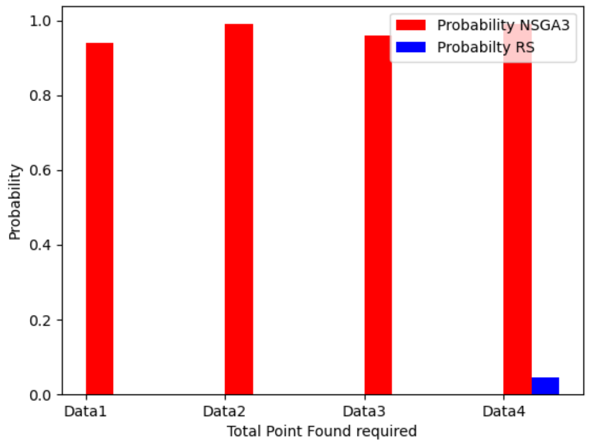


FIGURE 50 – $NSGA3 : P(X \geq k^i_{total} - 1)$

5. $k^i_{total} - 1$: 17 in the both IDU3FS-cycle, 8 in TWH3001 cycle L and 3 in TWH3001 cycle M

We can see that genetic algorithms are very effective to found $k^i_{total} - 1$ compared to randomsearch which is very inefficient.

2.14 Run FMUpy

To run the simulation with FMUpy in real conditions, we had to choose one algorithm with its set of parameters. Thans to its effectiveness, we choose NSGA2 with the set of parameters $\{mut\_pbs = 0.25, cx\_pbs = 0.6, n\_inits = 250, index\_pbs = 30\}$ which is the best parameter of the "minmax" classification and we runned the simulation during 1.5 days (35 cores) with the product IDU3FS cycle L.
We can see the evolution of points found in the pareto front for each generation of the algorithm with the evolution of the hypervolume metric :

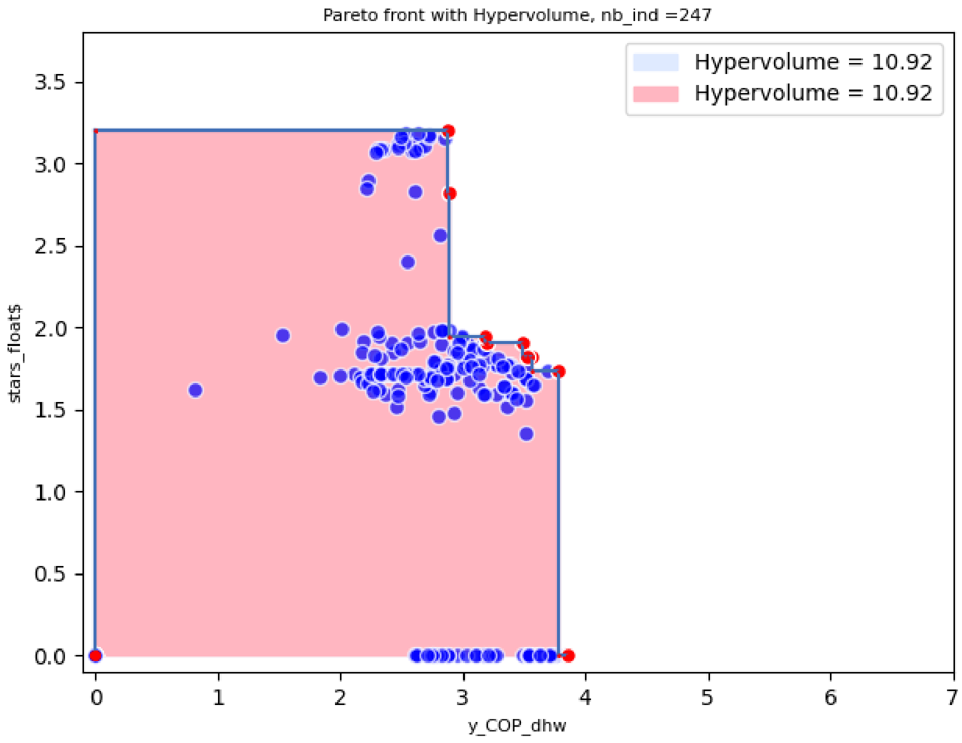


Figure 51 – Evolution : generation 1-247 individuals computed

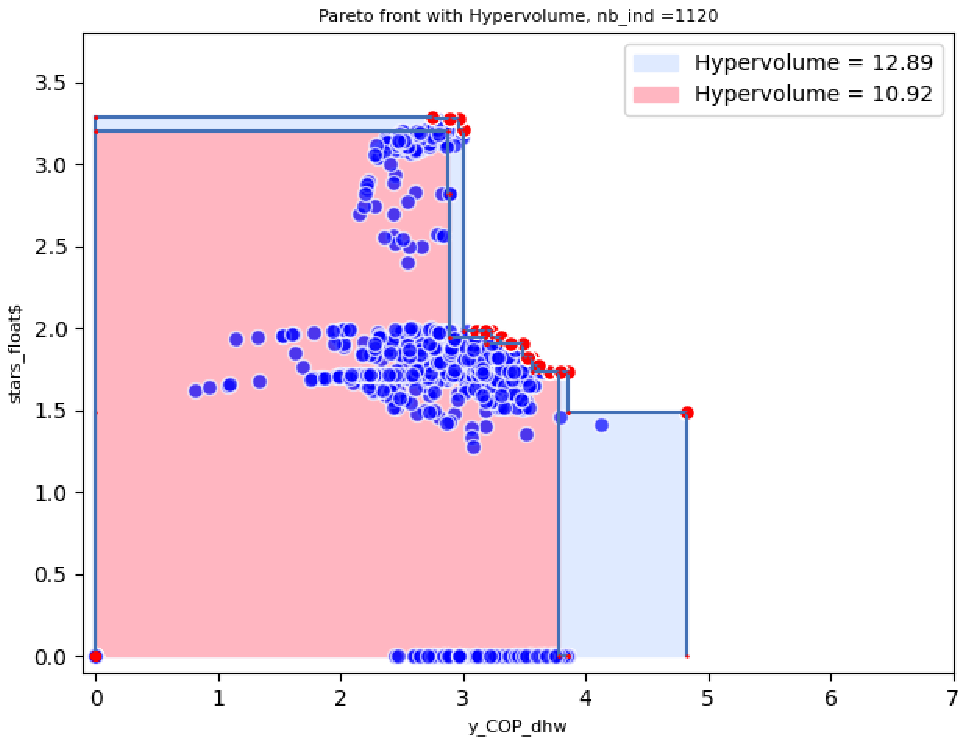


FIGURE 52 – Evolution : generation 6-1120 individuals computed

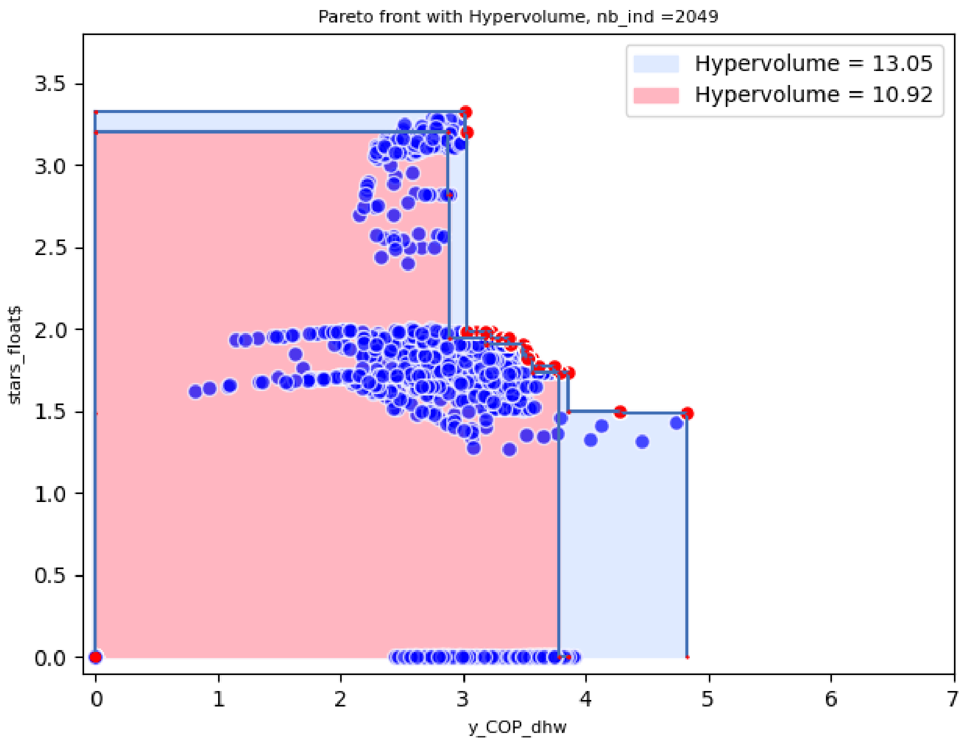


FIGURE 53 – Evolution : generation 12-2049 individuals computed

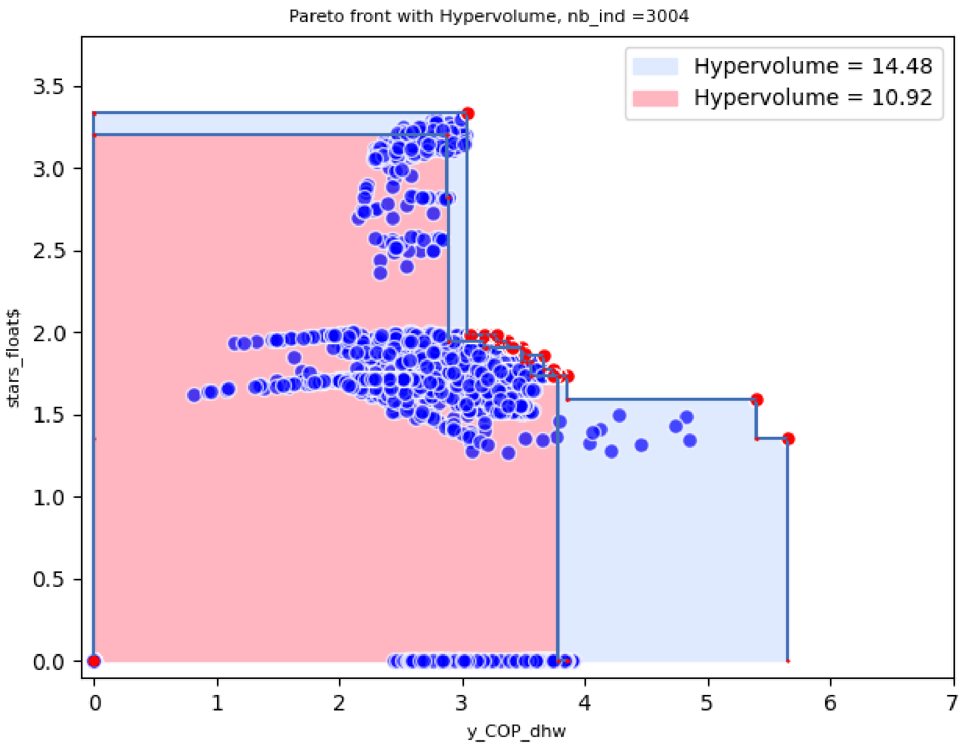


FIGURE 54 – Evolution : generation 20-3004 individuals computed

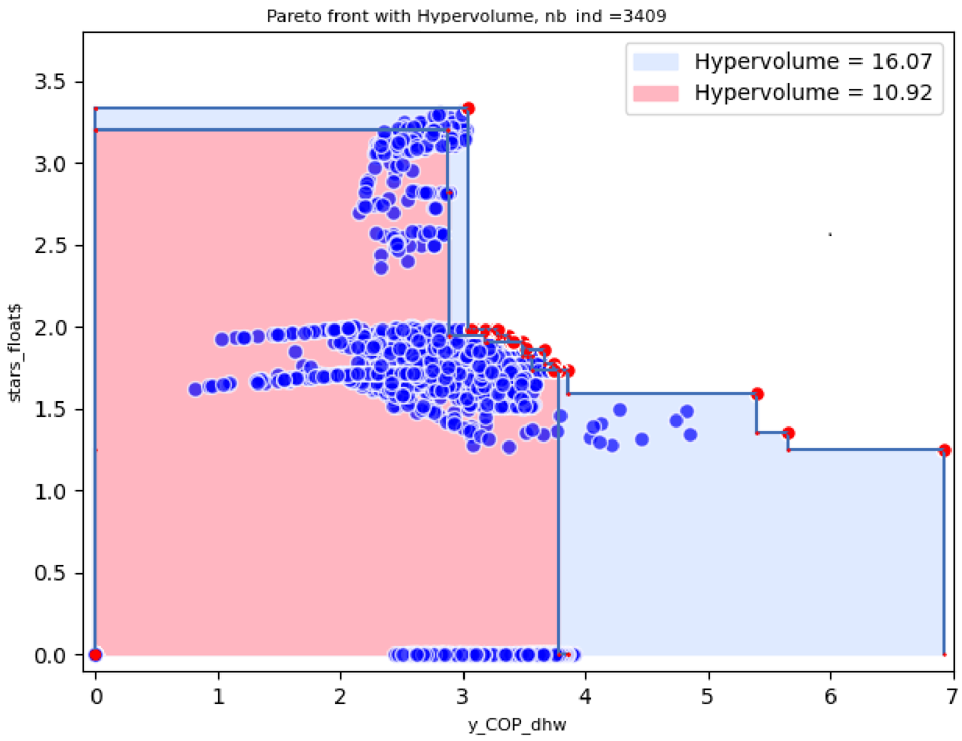


FIGURE 55 – Evolution : generation 22-3409 individuals computed

We can observe that :

- faster convergence to the points in the middle of the pareto front
- faster convergence to the points in the top of the pareto front
- hypervolume metric increases at each generation
- a tendency to explore to the right

To compute these 3409 individuals, It took us 1.08 days (3896 minutes) with 35 cores versus 3.07 days (11054 minutes) to explore all individuals with 35 cores, i.e a reduction of 64.8% of the computation time. More over, we could choose the hypervolume metric as a stopping criterion for our algorithms for the simple reason that it has the particularity to increase until to get all the points in the pareto front.

2.15 Conclusion :

In conclusion to this project, we have realized that metaheuristic algorithms are a promising method for solving complex optimization problems such as the maximization of the (COP_{DHW},$Stars\_float$). They allowed us to find a solution by exploring roughly a quarter of the design space and win a lot of time for the simulations.
We have seen that the choice of set of parameters are decisive to have a good convergence and could be choose amoung our different classifications.
Despite all, they are still a lot of areas to explore like other algorithms, stopping criterion by hypervolume metric, more operators etc..

3 Bibliography

Références

[1] FanScout online manual
https://ebmpapst.atlassian.net/wiki/spaces/PS/pages/8716290/Online%2Bmanual

[2] Libloader API
https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/

[3] LCIE 103-15/C specifications
https://www.lcie.fr/medias/cdc_103-15c_chauffe-eau_thermodynamique_autonome_a_accu-mulation_2018_06_version_française.pdf

[4] EN 16147 norm
https://www.boutique.afnor.org/norme/nf-en-16147/pompes-a-chaleur-avec-compresseur-entraine-par-moteur-electrique-essais-determination-des-performances-et-exigences-pour-le-m/article/905399/fa188149?utm_source=UNM&utm_medium=lien-texte&utm_campai-gnc=AffiliationUNM

[5] De Dietrich Thermique website
https://www.dedietrich-thermique.fr/

[6] Genetic Algorithms - John H ; Holland, 1992
http://bingweb.binghamton.edu/ apape/courses/e670/papers/holland-geneticalgorithms-1992.pdf

[7] Genetic Algorithms in Python using the DEAP library
https://towardsdatascience.com/genetic-algorithms-in-python-using-the-deap-library-e67f7ce4024c

[8] pymoo.org-algorithms-nsga2
https://pymoo.org/algorithms/nsga2.html

[9] JmetalPy GitHub repository
https://github.com/jMetal/jMetalPy/tree/master/jmetal

[10] DEAP tools
https://deap.readthedocs.io/en/master/api/tools.html

[11] DEAP examples
https://deap.readthedocs.io/en/master/examples/

[12] Pareto front
https://tel.archives-ouvertes.fr/tel-00959099/document

[13] DEAP NSGA-II example
https://github.com/DEAP/deap/blob/master/examples/ga/nsga2.py

[14] SBX Crossover
https://engineering.purdue.edu/ sudhoff/ee630/Lecture04.pdf

[15] Un modele base sur les algorithmes genetiques et le front de pareto pour l'optimisation multi-objectif des parametres de tournage
https://www.researchgate.net/profile/Idir-Belaidi-2/publication/279940732_-UN_MODELE_BASE_SUR_LES_ALGORITHMES_GENETIQUES_ET_LE_FRONT_DE_-PARETO_POUR_L%27OPTIMISATION_MULTI-OBJECTIF_DES_PARAMETRES_DE_-TOURNAGE/links/559ec1e508ae03c44a5cd5c2/UN-MODELE-BASE-SUR-LES-ALGORITHMES-GENETIQUES-ET-LE-FRONT-DE-PARETO-POUR-LOPTIMISATION-MULTI-OBJECTIF-DES-PARAMETRES-DE-TOURNAGE.pdf

[16] Dymola
https://www.mathworks.com/products/connections/product_detail/dymola.html

[17] pymoo NSGA3
https://pymoo.org/algorithms/nsga3.html

[18] Metaheuristic
https://en.wikipedia.org/wiki/Metaheuristic

[19] BDR Thermea IDU3FS product
https://www.dedietrich-thermique.fr/nos-produits/pompe-a-chaleur/strateo

[20] BDR Thermea TWH product
https://www.dedietrich-thermique.fr/nos-produits/chauffe-eau-thermodynamique/kaliko

[21] Metamodelisations
https://fr.wikipedia.org/wiki/M%C3%A9tamod%C3%A8le

[22] Genetic algorithms
https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

[23] Stars
https://www.lcie.fr/medias/cdc_103-15c_chauffe-eau_thermodynamique_autonome_a_accu-mulation_2018_06_version_fran%C3%A7aise.pdf

[24] Evolutionary-Algorithms
https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac

[25] PSO
https://www.sciencedirect.com/topics/engineering/particle-swarm-optimization

[26] principle of PSO

[27]

[28] Kennedy, J. and Eberhart, International Conference On Neural Networks, from 1995
https://ieeexplore.ieee.org/document/488968

[29] Step Detailed of PSO's algorithms
https://www.intechopen.com/chapters/69586

[30] FMpy
https://github.com/CATIA-Systems/FMPy

[31] reference point image
https://www.researchgate.net/figure/The-reference-points-of-NSGA-III-in-three-objective-problem_fig1_331088291

[32] Documentation detailed of JmetalPy
https://arxiv.org/pdf/1903.02915.pdf

[33] Crowding distance
https://www.researchgate.net/figure/The-definition-of-the-crowding-distance-and-the-weighted-crowding-distance_fig19_287992462