Rapport

Passing vehicle search

Élèves :
Yassine TAHIRI ALAOUI

Enseignant :
Rachid ELLAIA
Maryam LAMI
Imane BRIKI

26 juin 2024

Gratitude I would like to thank you for allowing us to work on this project. It has guided us very well in our learning and allowed us to fully develop our knowledge.

1 General Overview

Optimization methods are crucial for tackling complex problems in various fields such as engineering, computer science, economics, and finance. These methods aim to find values for variables that either minimize or maximize an objective function while satisfying specific constraints. Traditional approaches include gradient descent and Newton-Raphson methods, whereas evolutionary methods like genetic algorithms also play a significant role. Optimization problems can be continuous or discrete, with some requiring stochastic or multi-objective techniques. These methods are widely applied in system design, logistics planning, and mathematical modeling, and must be chosen based on the problem's characteristics, especially for NP-hard problems.

1.1 Metaheuristic

Metaheuristic methods are a class of optimization algorithms offering innovative and flexible approaches to solving complex problems that traditional methods struggle with. Inspired by natural phenomena, these techniques rely on evolutionary mechanisms, swarm behaviors, or animal-inspired processes to find high-quality solutions within a search space. Common metaheuristics include genetic algorithms, ant colony optimization, particle swarm optimization, and simulated annealing. Their ability to effectively explore complex search spaces makes them powerful tools for solving realistic optimization problems. Metaheuristics are particularly useful in situations where classical methods fall short, such as combinatorial problems, scheduling issues, and other complex scenarios. Although they do not always guarantee convergence to the optimal solution, their adaptability and robustness make them valuable for real-world problems, where trade-offs between efficiency and accuracy are often necessary.

1.2 Advantages of Optimization Methods

Metaheuristic optimization methods offer several advantages for solving complex problems. Their adaptability and robustness allow them to adjust to various problem types without major modifications. These generic approaches are applicable to a wide range of problems, whether continuous, discrete, single-objective, or multi-objective.

A key feature of metaheuristics is their ability to combine global exploration of the search space with local exploitation of promising solutions. This duality promotes the discovery of new potential solutions while refining existing ones.

Another significant advantage is that metaheuristics do not require detailed knowledge of the derivatives of objective functions and constraints, distinguishing them from traditional optimization methods that often rely on such information. Metaheuristics excel in handling complex, nonlinear, and non-convex problems, where the topology of the objective function may be difficult to describe. Additionally, some metaheuristics can be parallelized, facilitating the resolution of complex problems using distributed computing resources.

In terms of performance, while convergence to the optimal solution is not guaranteed, metaheuristics often provide high-quality solutions within a reasonable timeframe. Their

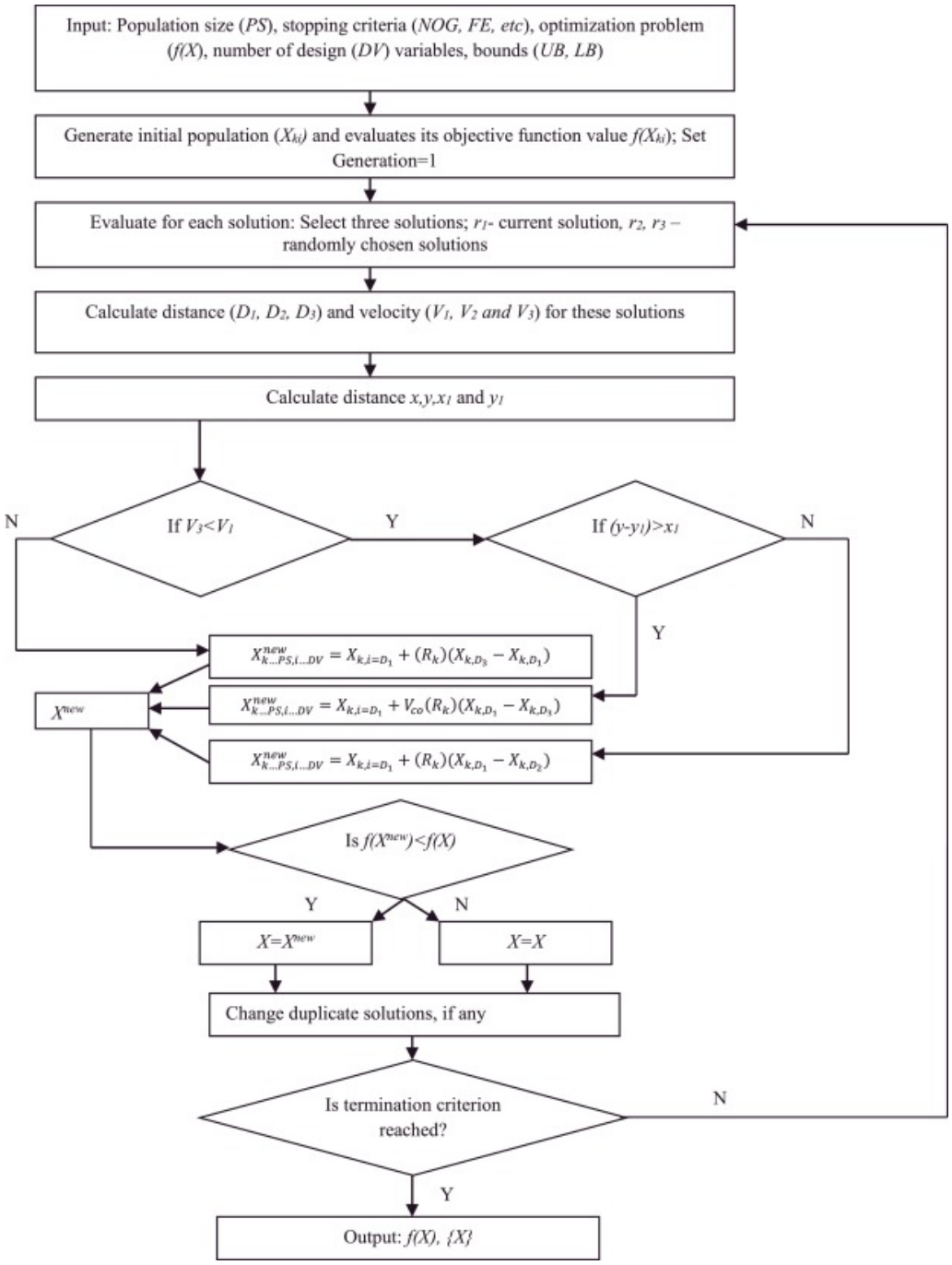
relative simplicity in implementation makes them accessible to a wide audience, even those without expertise in mathematical optimization.

In conclusion, although the choice of method depends on the specific problem, metaheuristics stand out for their versatility and ability to tackle real-world problems effectively. These characteristics make them valuable tools in the field of optimization.

2 Mono Objective Passing Vehicle Search

In many developed and developing countries, two-lane roads are prevalent in primary road networks. Studies indicate that two-lane roads constitute over 65% of urban and rural routes in the United States and about 54% in India. Additionally, more than 60% of accidents occur on two-lane rural roads. The key consideration for overtaking on a two-lane road is identifying safe passing opportunities, which depend on complex and interdependent factors such as the availability of gaps in oncoming traffic, individual vehicle speed and acceleration characteristics, traffic and driver behaviors, and road and weather conditions. Numerous studies have simulated traffic on two-lane roads, considering factors like overtaking, convoys, delays, traffic quality, and speed distribution.

Overtaking on a two-lane road involves using the lane for oncoming traffic, influenced by the demand and availability for passing at specific times and locations. This maneuver depends on speed differences between vehicles and requires sufficient distance from oncoming vehicles and adequate observable distance for the overtaking vehicle. Driver perspective also plays a role, as preferred speeds vary based on journey type, vehicle type, weather, road geometry, traffic conditions, and driver personality. Three major scenarios can occur when approaching a slower vehicle : (a) overtaking the slower vehicle, (b) following the slower vehicle until a passing opportunity arises, or (c) following the slower vehicle without intending to overtake. If vehicles do not overtake a slower vehicle, convoys form, affecting the preferred speed of the vehicles. Consequently, mathematical theories on two-lane traffic are challenging to establish and analyze.



3 Implementation of the PVS Algorithm in Python :

[7]:
```python
import numpy as np

def PVS(problem, params):

    FitnessFunction = problem["FitnessFunction"]
    DV = problem["DV"]
    LB = problem["LB"]
```

```
UB = problem["UB"]
VarSize = (1, DV)


Gmax = params["Gmax"]
PS = params["PS"]


empty_vehicle = {"Position": None, "Fitness": None}
FE = 0
g = 1


pop = [empty_vehicle.copy() for _ in range(PS)]

for i in range(PS):

    pop[i]["Position"] = np.random.uniform(LB, UB, VarSize)


    pop[i]["Fitness"] = FitnessFunction(pop[i]["Position"])
    FE += 1


pop = sorted(pop, key=lambda x: x["Fitness"])


BestSol = pop[0]


BestFitness = np.zeros((Gmax, PS))


BestFitnessInGeneration = np.zeros(Gmax)


while g <= Gmax:
    for i in range(PS):
        r = np.zeros(3, dtype=int)

        r[0] = i
        r1, r2, r3 = pop[r[0]], pop[r[1]], pop[r[2]]

        r[1] = r[0]
        while r[1] == r[0]:
            r[1] = np.random.randint(0, PS)
```

```
            r[2] = r[0]
            while r[2] == r[0] or r[2] == r[1]:
                r[2] = np.random.randint(0, PS)

            D = np.array([r[k] / PS for k in range(3)])
            V = np.random.rand(3) * (1 - D)

            y = np.abs(r2["Position"] - r3["Position"])
            x = np.abs(r2["Position"] - r1["Position"])

            x1 = V[2] * x / (V[0] - V[2])
            y1 = V[1] * x / (V[0] - V[1])

            if V[0] > V[2]:
                if (y - y1).all() > x1.all():
                    Vco = V[0] / (V[0] - V[2])
                    X_new = r1["Position"] + Vco * np.random.
↪rand(*VarSize) * (r1["Position"] - r3["Position"])
                else:
                    X_new = r1["Position"] + np.random.rand(*VarSize) *␣
↪(r1["Position"] - r2["Position"])
            else:
                X_new = r1["Position"] + np.random.rand(*VarSize) *␣
↪(r3["Position"] - r1["Position"])


            X_new = np.maximum(X_new, LB)
            X_new = np.minimum(X_new, UB)


            F_new = FitnessFunction(X_new)
            FE += 1


            if F_new < r1["Fitness"]:
                pop[r[0]]["Position"] = X_new
                pop[r[0]]["Fitness"] = F_new


            for k in range(PS - 1):
                if np.array_equal(pop[k]["Position"], pop[k +␣
↪1]["Position"]):
                    R = np.random.randint(0, DV)
                    pop[k + 1]["Position"][0, R] = np.random.
↪uniform(LB[0, R], UB[0, R])
```

```
            BestFitness[g - 1, i] = pop[r[0]]["Fitness"]


        pop = sorted(pop, key=lambda x: x["Fitness"])

        BestFitnessInGeneration[g - 1] = pop[0]["Fitness"]
        g += 1

    out = {"population": pop, "BestSol": pop[0]["Position"],␣
↪"BestFitness": pop[0]["Fitness"]}

    return out
```

[25]:
```
###For a single-objective optimization benchmark function, let's␣
↪consider the Sphere Function, a simple and widely used benchmark␣
↪function.
def sphere_function(x):
    return np.sum(x**2)

problem = {
    "FitnessFunction": sphere_function,
    "DV": 5,
    "LB": -5.12,
    "UB": 5.12
}

params = {
    "Gmax": 50,
    "PS": 20


result = PVS(problem, params)


print("Best Solution:", result["BestSol"])
print("Best Fitness:", result["BestFitness"])
```

```
Best Solution: [[ 0.18221628 -0.08650219 -0.26172758  0.31670193 -0.
↪11753342]]
Best Fitness: 0.22330095146847875
```

These results represent an optimal solution found by the PVS optimization algorithm for a single-objective optimization test function. Here is the interpretation of these results :

Best Solution : The best solution corresponds to the values of the decision variables that were found to be optimal by the algorithm. In this case, the values of the decision variables are [0.18221628, -0.08650219, -0.26172758, 0.31670193, -0.11753342]. This means that these values either minimize or maximize the objective function, depending on the problem.

Best Fitness : The best fitness corresponds to the value of the objective function evaluated with the best solution found. In this case, the best fitness is 0.22330095146847875. This indicates that, for the given values of the decision variables in the best solution, the value of the objective function is equal to 0.22330095146847875. In the case of minimization, this suggests that it is the best value the algorithm has reached, and in the case of maximization, it indicates the best value reached for the objective function.

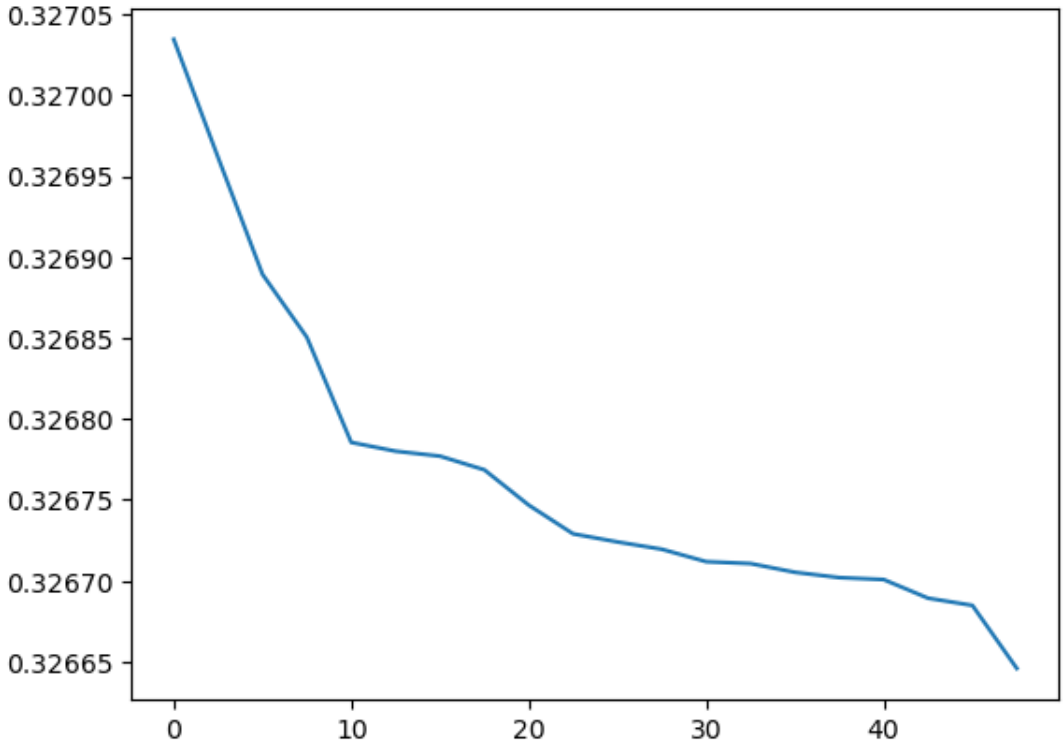
[5]:
```
fitness=[]
for i in result["population"]:
    fitness.append(i['Fitness'])

fitness.reverse()
```

[6]:
```
g=[]
for i in range (20):
    g.append(i*2.5)
```

[7]:
```
import matplotlib.pyplot as plt
plt.plot(np.transpose(g),np.transpose(fitness))
```

[7]:
```
[<matplotlib.lines.Line2D at 0x26182fb6450>]
```



Finally, here's a visualization of the fitness function over generations. The fitness list is reversed because the results are sorted in ascending order. This graph shows the fluctuation in fitness values across generations, illustrating how the Principal Variation Search (PVS) method tends to minimize the objective function.

4 Multi-Objective Optimization

Multi-objective optimization (MOOP) involves optimizing several objective functions simultaneously. These functions can represent various performance criteria, costs, or metrics that need to be minimized or maximized. Each objective function addresses a different aspect of the problem, and the goal is to find a set of solutions that offer the best trade-offs among these conflicting objectives.

4.1 pareto optimality :

Definition : Pareto optimality, named after economist Vilfredo Pareto, refers to a state where no solution can be improved in one objective without worsening another. A solution is Pareto optimal if there is no alternative that provides a better trade-off among conflicting objectives.

Pareto Front The set of all Pareto optimal solutions forms the Pareto front, representing the best trade-offs between conflicting objectives. This front is used as a reference for comparing different solutions.

Non-Domination Sorting Identifying Pareto optimal solutions typically involves non-domination sorting, categorizing solutions into different Pareto fronts based on their dominance relationships.

Visualization and Algorithms Visualization : Visualizing the Pareto front is crucial for understanding trade-offs in MOOP. Techniques like scatter plots, parallel coordinates, or radar charts can be used to represent the Pareto front and highlight the distribution of solutions.

Evolutionary Algorithms : Evolutionary algorithms, such as genetic algorithms, particle swarm optimization, or evolutionary strategies, are commonly used to find Pareto optimal solutions in MOOP. These algorithms evolve a population of solutions towards the Pareto front iteratively..

4.2 Multi-Objective Passing Vehicle Search (MOPVS)

The MOPVS algorithm introduces an innovative approach to multi-objective optimization, inspired by the dynamic interactions between overtaking vehicles on a road. The main goal is to identify a set of non-dominated solutions in a specified search space.

Initially, the algorithm generates a population of individuals with random positions in the search space. Each individual's fitness is evaluated by calculating the objective function values. In the algorithm's main loop, each individual undergoes an update process involving selecting a random individual from the population, calculating a new velocity for each dimension, and updating the position accordingly.

4.3 Crowding Function in Multi-Objective Optimization

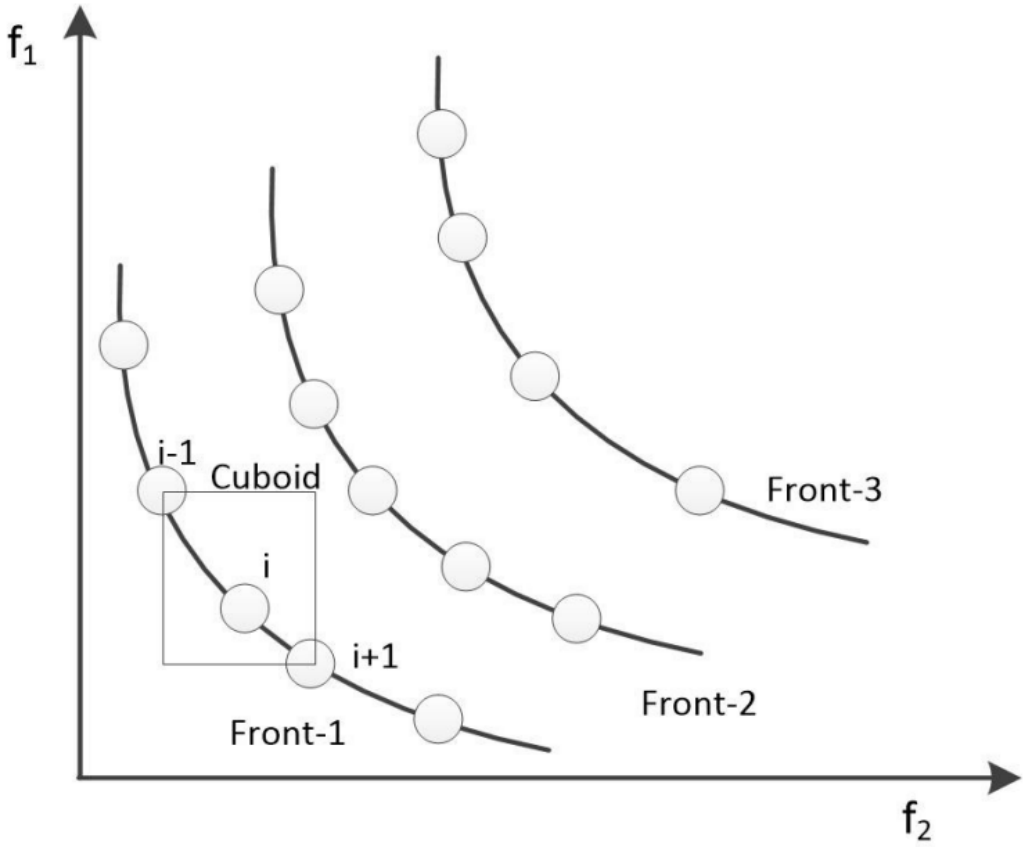
Crowding Function : The crowding function helps maintain solution diversity and balance the population along the Pareto front. It measures the density of solutions in the objective space, prioritizing less dense regions for exploration.

For each solution, calculate the crowding distance by summing the normalized differences in objective values between adjacent solutions. The crowding distance for a solution in objective space can be computed as follows :

$$CD_i = \sum_{m=1}^{M} \frac{f_m(i+1) - f_m(i-1)}{f_m^{\max} - f_m^{\min}}$$

where :

- M is the number of objectives.
- $f_m(i+1)$ is the value of the objective m for the next solution in the sorted list.
- $f_m(i-1)$ is the value of the objective m for the previous solution in the sorted list.
- $f_m^{\max}$ is the maximum value of the objective m among all solutions.
- $f_m^{\min}$ is the minimum value of the objective m among all solutions.



5 Multi-objective Passing Vehicle Search (MOPVS) Algorithm

The objective of the Multi-objective Passing Vehicle Search (MOPVS) algorithm is to find a set of non-dominated solutions within a search space defined by the lower and upper bounds for each dimension.

1. **Population Initialization :** The algorithm begins by creating a population of individuals randomly distributed within the search space. The fitness of each individual is evaluated by calculating their objective function values.

2. **Main Loop :** During each iteration of the main loop, each individual is updated. This involves selecting a random individual from the population and calculating a new velocity for each dimension based on the current velocity, the difference between the current position and the position of the randomly selected individual, and a velocity factor. Each individual's position is then updated according to the new velocity, ensuring it remains within the search space. If the new position dominates the individual's previous position, it replaces the old position.
3. **Non-dominated Frontiers :** Finally, the non-dominated frontiers of the population are calculated, and the non-dominated solutions are stored in a list, which is returned as the output of the algorithm.

The MOPVS algorithm is a simple and efficient method for finding a set of non-dominated solutions in a multi-objective optimization problem. Its inspiration from the concept of vehicles overtaking each other makes it an interesting and intuitive approach to optimization.

Proposed Pseudocode

1. **Define Parameters :** Define the parameters such as population size (PS), stopping criteria (Number Of Generations (NOG), Maximum function evaluations (FE), and error), number of design variables (DV), and bounds on design variables (LB, UB).
2. **Generate Initial Solutions :** Generate an initial set of solutions randomly and evaluate the objective function values for each solution.
3. **Set Generation Counter :** Set generation counter $t = 1$.
4. **Non-dominated Sorting :** Use a fast non-dominated sorting algorithm to group the solutions into different Pareto fronts. Assign a rank to each solution based on the number of solutions that dominate it. Solutions with higher ranks are closer to the Pareto front.
5. **Calculate Crowding Distance :** For each Pareto front, calculate the crowding distance for each solution using the provided steps.
6. **Selection :** Select two solutions from the current Pareto front using a tournament selection method based on the crowding distance. The solution with a higher crowding distance is preferred.
7. **Generate New Solution :** Generate a new solution by perturbing the design variables of the selected solutions. Evaluate the objective function values for the new solution.
8. **Update Population :** Use the Pareto dominance relation to determine whether the new solution should be added to the population. If the new solution is dominated by any existing solution, discard it. Otherwise, add it to the population and update the Pareto fronts, crowding distances, and velocities accordingly.
9. **Termination :** Repeat steps 4 to 8 until the termination criteria are satisfied, else go to Step 3.

Where :

— D_k = Normalized distance between the k-th vehicle and its nearest neighbor (Eq. 1).

— N_k = Number of solutions that dominate the k-th solution.
— V_k = Velocity of the k-th vehicle (Eq. 2).
— R_k = Random number between 0 and 1.
— PS = Population size.
— LB_i and UB_i = Lower and upper bounds on the i-th design variable.

5.1 Results refer to the Multi-objective Passing Vehicle Search (MOPVS)

5.1.1 Test functions

Test functions are essential for evaluating optimization algorithms. DTLZ1, DTLZ4, and ZDT3 are commonly used multi-objective test functions. We will use these test functions to evaluate the PVS (Passing Vehicle Search) method for optimization. By applying PVS to these well-established benchmarks, we can effectively assess its performance in solving complex multi-objective optimization problems.

5.1.2 Library Used in the Code

The pymoo library is a comprehensive Python library designed for multi-objective optimization. It provides a wide range of functionalities, including the definition of complex optimization problems, application of state-of-the-art algorithms, and visualization of results. pymoo supports various benchmark problems like ZDT3, facilitating the evaluation of optimization methods. Its powerful minimize function streamlines the optimization process by integrating seamlessly with different algorithms, such as the widely used NSGA2 (Non-dominated Sorting Genetic Algorithm II). Additionally, pymoo offers robust visualization tools, enabling intuitive representation of Pareto fronts and aiding in the analysis of algorithm performance. This library's versatility and ease of use make it an invaluable tool for researchers and practitioners in the field of multi-objective optimization.

5.1.3 Metrics

In evaluating the performance of optimization algorithms, several metrics are crucial for providing a comprehensive assessment.
Mean Squared Error (MSE) and Mean Absolute Error (MAE) are commonly used to measure the accuracy of predictions, with MSE focusing on the average squared difference between predicted and actual values, and MAE on the average absolute differences. Root Mean Squared Error (RMSE) offers a more interpretable metric by providing the square root of MSE, making it easier to relate to the original data scale. Generational Distance (GD) measures the average distance between the obtained solutions and the true Pareto front, indicating how close the solutions are to the optimal set. Hypervolume (HV) evaluates the volume of the objective space dominated by the obtained solutions, providing insight into both convergence and diversity. Finally, Spacing to Extent (STE) assesses the uniformity of the distribution of solutions along the Pareto front, ensuring that the solutions are evenly spread. Together, these metrics offer a robust framework for evaluating the quality, accuracy, and diversity of solutions produced by multi-objective optimization algorithms.

5.1.4 Results

DTLZ1
DTLZ1 is a commonly used multi-objective test function designed to evaluate the performance of optimization algorithms. The Pareto front in this case represents a set of non-dominated solutions where improvement in one objective would lead to the deterioration of at least one other objective. This visualization aids in understanding the distribution and convergence of solutions achieved by the optimization method under consideration, such as the Multi-objective Passing Vehicle Search (MOPVS) method.

Definition

$$
\begin{aligned}
&\text{Minimize} \quad f_1(x) = \frac{1}{2} x_1 x_2 \cdots x_{M-1}(1 + g(x_M)),\\
&\text{Minimize} \quad f_2(x) = \frac{1}{2} x_1 x_2 \cdots (1 - x_{M-1})(1 + g(x_M)),\\
&\quad \vdots\\
&\text{Minimize} \quad f_{M-1}(x) = \frac{1}{2} x_1 (1 - x_2)(1 + g(x_M)),\\
&\text{Minimize} \quad f_M(x) = \frac{1}{2}(1 - x_1)(1 + g(x_M)),\\
&\text{subject to} \quad 0 \le x_i \le 1, \quad \text{for} \quad i = 1, 2, \ldots, n.
\end{aligned}
$$

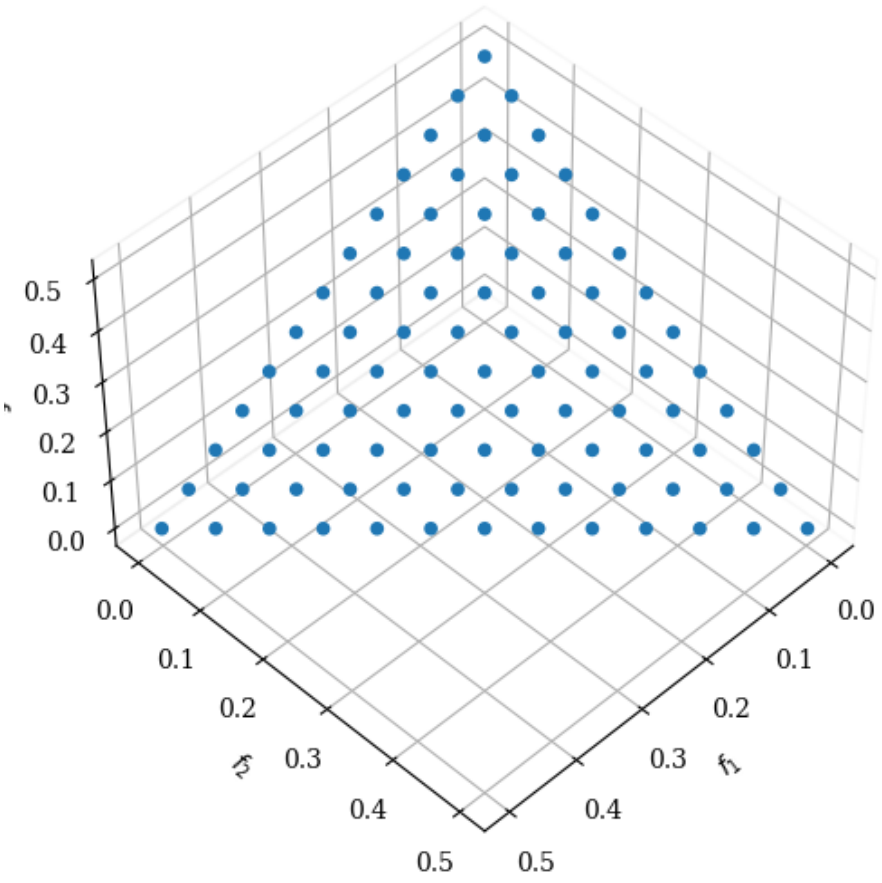
The last $k = (n - M + 1)$ variables are represented as x_M. The functional $g(x_M)$ requires $|x_M| = k$ variables and must take any function with $g \geq 0$.

We suggest the following :

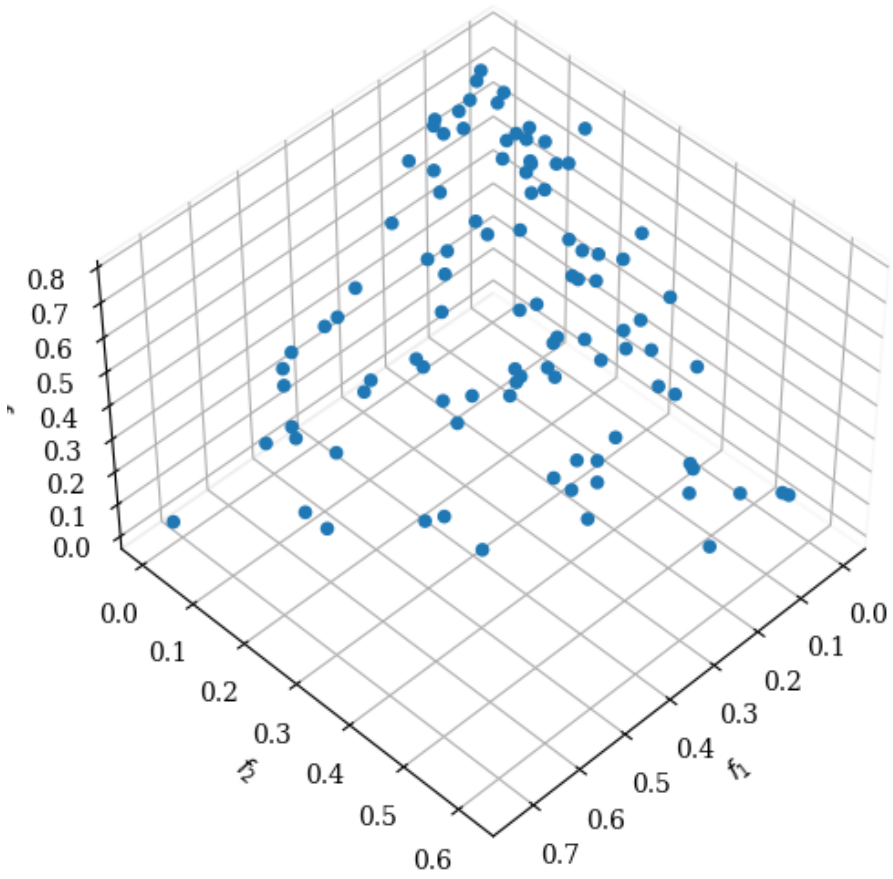
$$
g(x_M) = 100 \left[|x_M| + \sum_{x_i \in x_M} (x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5)) \right]
$$

Optimum

The Pareto-optimal solution corresponds to $x_i = 0.5$ (for all $x_i \in x_M$) and the objective function values lie on the linear hyper-plane : $\sum_{m=1}^{M} f_m^* = 0.5$. The figure above illustrates the Pareto front of the DTLZ1 test function.



The figure above illustrates the Pareto front obtained using the Multi-objective Passing Vehicle Search (MOPVS) algorithm. This Pareto front represents a set of non-dominated solutions discovered by the MOPVS method when applied to the DTLZ1 test function. Each point on the plot corresponds to a solution where no objective can be improved without compromising another, indicating a balanced optimization across multiple objectives. This visualization demonstrates the effectiveness of the MOPVS algorithm in identifying a diverse set of optimal solutions for complex multi-objective optimization problems.



Optimization Metrics

In this optimization application, we measure the performance of our algorithm using the following metrics :

- **Average Mean Squared Error (MSE)** : 0.06836899481720811
- **Average Mean Absolute Error (MAE)** : 0.20244957662477817
- **Average Root Mean Squared Error (RMSE)** : 0.2553438957045833
- **Generational Distance (GD)** : 0.09947396342625855

Financial Optimization Problem

Imagine a financial scenario where an investor is trying to allocate resources (represented by x) between two different investment opportunities to maximize returns.

First Function

$$f_1(x) = -x^3$$

Financial Interpretation

Investment Type 1 : represent a high-risk, high-reward investment opportunity, such as venture capital or early-stage startup investment.

Returns Profile : The returns diminish rapidly as more resources are allocated. This is because the function $-x^3$ increases negatively as x increases. Hence, the more you invest in this opportunity, the higher the risk of diminishing returns, possibly due to over-concentration or market saturation.

Second Function

$$f_2(x) = -(x-2)^2$$

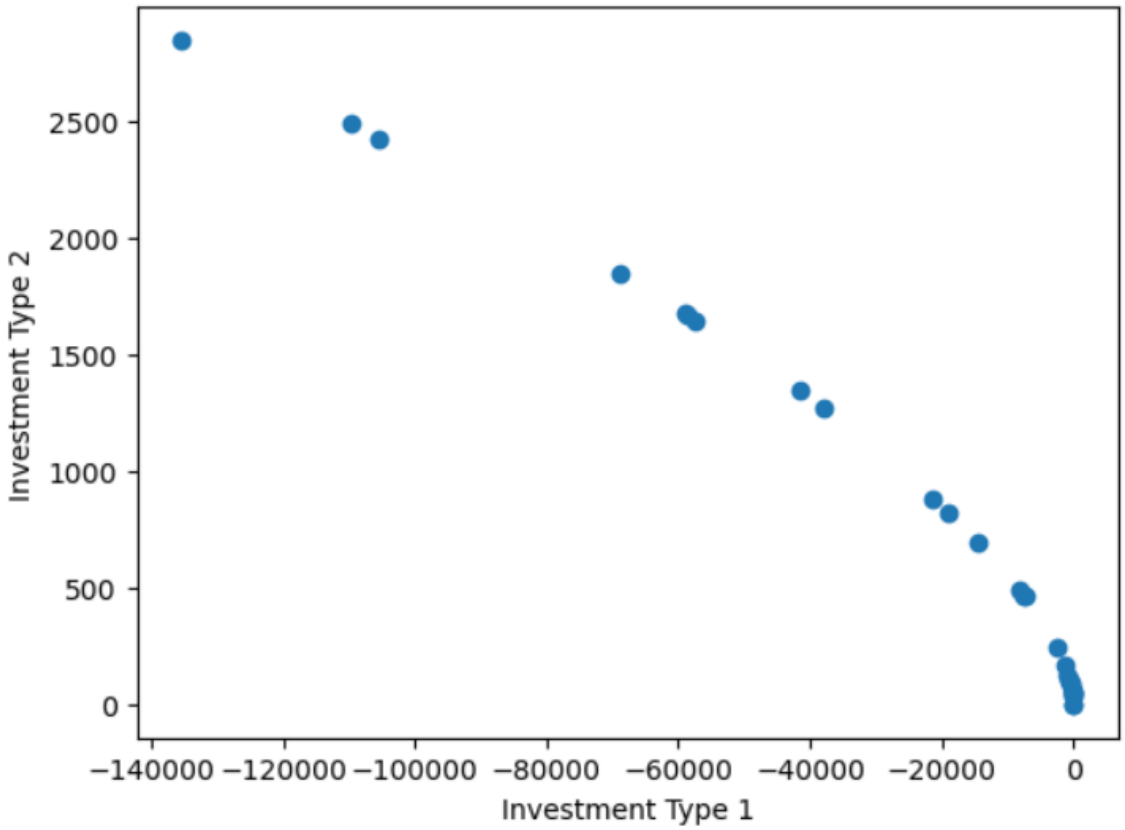
Financial Interpretation

Investment Type 2 : represent a moderate-risk, steady-return investment opportunity, such as investing in a well-established company.

Returns Profile : The returns are maximized when the investment is around $x = 2$. This suggests an optimal investment amount that provides the best return, and deviations from this optimal amount (either more or less) result in decreased returns. This can be interpreted as a sweet spot in the investment amount that maximizes returns, with diminishing returns on either side.

6 Results

To solve this problem of maximizing the returns of our portfolio based on two types of investments, we will employ the Multi-Objective Passing Vehicle Search (MOPVS) method of optimization. The results of our optimization are presented in the figure above, which illustrates the Pareto front.



The Pareto front illustrated in the figure represents the trade-off between the two investment types. Each point on the Pareto front indicates a potential portfolio allocation where increasing the return from one type of investment results in a decrease in the return from the other type, and vice versa.

Here are key observations :

Trade-off Relationship : The negative slope of the Pareto front suggests an inverse relationship between the returns of the two investment types. As the returns from Investment Type 1 increase (less negative), the returns from Investment Type 2 decrease, and vice versa.

Optimal Points : Points on the Pareto front are considered optimal because they offer the best possible returns for a given level of investment in either type. Any point not on the Pareto front would mean there is another allocation that offers better returns for at least one type of investment without worsening the returns for the other type.

Investment Type 1 Dominance : The wide range of negative values for Investment Type 1 indicates that this investment type is associated with a higher level of risk and potentially high returns, but also substantial losses.

Investment Type 2 Stability : The returns for Investment Type 2, on the other hand, are positive and range between 0 and approximately 2500, indicating a more stable and less risky investment compared to Investment Type 1.

Decision Making : Investors would choose a point on the Pareto front based on their risk tolerance and return expectations. For a more aggressive investment strategy, an investor might choose a point with higher returns from Investment Type 1 despite the higher risk. Conversely, a more conservative investor might prefer a point with higher returns from Investment Type 2, accepting lower overall returns for reduced risk.

The Pareto front helps in understanding the trade-offs and making informed decisions to balance the portfolio according to the investor's preferences and risk appetite.

7 References

https://github.com/sahutkarsh/NSGA-II/blob/master/NSGA-II.ipynb
https://pymoo.org/problems/many/dtlz.html
https://en.wikipedia.org/wiki/Multi-objective_optimization
https://www.sciencedirect.com/science/article/pii/S0307904X15007027#:~:text=Similar%20to%20other%20metaheuristic%20methods,on%20a%20two%2Dlane%20highway.