ESKİŞEHİR OSMANGAZİ ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Introduction To Heuristic Algorithms

2023 - 2024 Fall Term

Project Report

Group 2

Burak CAFERLER - 152120201144

Adem MAVANACI - 152120191040

İrem GÜNEŞ - 151320183048

Aylin ÜNAL - 152120181117

Dr. Öğr. Üyesi Emrah ATILGAN

Ocak 2024

1. Introduction..... 3
1.1. General Purpose and Scope..... 3
1.2. Technologies and Tools Used..... 3
1.3. Project Description..... 3
1.3.1 PyQt5..... 3
1.3.2. Heuristic Algorithms..... 4
1.3.3. Optimization Functions..... 5
2. Structure of Interface..... 6
3. Codes Explained..... 9
4. Tests Performed And Screen Photos..... 14
4.1. Results and Discussion..... 17
4.1.1. Project Results and Achievements..... 17
4.1.2. Challenges and Solutions..... 17
4.2. Future Studies..... 17
4.2.1. Development Potential of the Project..... 17
4.2.2. Additional Features and Improvements..... 17
References..... 18

1. Introduction

1.1. General Purpose and Scope

This project is designed to evaluate and compare the performance of various heuristic algorithms in optimization problems. These algorithms are intended for use in complex tasks such as the optimization of mathematical functions. The main objective of the project is to understand how effective and efficient these algorithms are in various scenarios.

1.2. Technologies and Tools Used

The project has been developed using Python and the PyQt graphical user interface library. Various heuristic algorithms (GA, GWO, HS, PSO, SA, WOA, ALO, SSA, SCA, MVO, MFO, JAYA, HHO, FFA, DE, CS, BAT) and mathematical functions (Ackley, Griewank, Schwefel, Rastrigin, etc.) have been utilized.

1.3. Project Description

Detailed Explanation of Technologies and Tools Used

1.3.1 PyQt5

PyQt5 is a versatile Python library used for creating cross-platform desktop applications with graphical user interfaces (GUIs). It provides bindings for the Qt toolkit and was chosen for our project due to its flexibility and ease of use.

Utilizing PyQt5, we designed the project's UI components, including buttons, labels, comboboxes, and stacked widgets. The Qt Designer tool, part of PyQt5, facilitated the visual layout of the main window and UI elements, resulting in an intuitive and user-friendly interface.

The integration of Matplotlib with PyQt5 allowed seamless embedding of Matplotlib plots, enabling real-time visualization of optimization results. PyQt5's signal and slot mechanism were employed to handle user interactions, ensuring a responsive UI that dynamically updated based on user input.

The dynamic nature of the project, with users selecting from various optimization algorithms and functions, required a multi-window UI. PyQt5 supported the creation of organized interfaces with different input fields for algorithm parameters and function selections.

In summary, PyQt5 played a crucial role in developing our project by providing a robust framework for UI design and implementation. Its integration with Matplotlib facilitated effective visualization, contributing to the creation of an interactive platform for heuristic algorithm and optimization function evaluation.

1.3.2. Heuristic Algorithms

- GA (Genetic Algorithm): An optimization technique that mimics biological evolutionary processes. It offers a general solution for various problems but carries the risk of getting trapped in local optima.

- GWO (Grey Wolf Optimizer): A method that models the hunting behavior of wolf packs. It performs well in complex search spaces.

- HS (Harmony Search): An approach inspired by musical composition processes. It requires little parameter adjustment and is noted for its simple structure.

- PSO (Particle Swarm Optimization): A method that imitates the movements of bird flocks or fish schools. It provides a fast and efficient search structure but may struggle in complex search areas.

- SA (Simulated Annealing): A method that mimics the thermal processing of metals. It yields good results in broad search areas, though its computational intensity is high.

- WOA (Whale Optimization Algorithm): Mimics the hunting behavior of whale pods.

- ALO (Ant Lion Optimizer): Models the hunting strategies of antlions.

- SSA (Salp Swarm Algorithm): Emulates the natural behaviors of salp swarms.

- SCA (Sine Cosine Algorithm): An optimization method based on mathematical sine and cosine functions.

- MVO (Multi-Verse Optimizer): Inspired by theories of the universe.

- MFO (Moth Flame Optimization): Mimics the flight pattern of moths towards a flame. JAYA (JAYA Algorithm): A simple, parameter-less, and effective optimization algorithm.

- HHO (Harris Hawks Optimization): Imitates the hunting strategy of Harris hawks.

- FFA (Firefly Algorithm): Mimics the light-emitting behavior of fireflies.

- DE (Differential Evolution): A powerful optimization method based on evolutionary strategy.
- CS (Cuckoo Search): Models the parasitic egg-laying behavior of cuckoo birds.
- BAT (Bat Algorithm): Imitates the echolocation behavior of bats.

1.3.3. Optimization Functions

- Ackley: A complex function with numerous local minima.
- Griewank: Often used to test algorithms' tendency to get trapped in local minima.
- Schwefel: Known for its wide search area and complex structure.
- Rastrigin: Features a wavy landscape with frequent local minima.
- Sphere: Simple and smooth structure, typically used to test basic performance of algorithms.
- Perm: A complex function involving multiple variables.
- Zakharov: Another function known for its complex structure.
- Rosenbrock: Characterized by a narrow valley, requiring algorithms to follow a specific path.
- Dixon-Price: Used to test the performance of evolutionary algorithms.
- Michalewicz: Often used for optimization problems with a focus on the efficient exploration of search spaces.
- Powell: A multidimensional optimization function often used for testing optimization algorithms.
- Powersum: Involves the sum of powers of different degrees, testing algorithms' ability to handle various power functions.
- Sum2: A simple function involving the sum of squares of variables.

- Trid: Known for its global convex shape with a narrow, diagonal valley.
- Ellipse: A function with an elliptic shape, testing algorithms' ability to handle non-spherical structures.
- Nesterov: Named after the mathematician Yurii Nesterov, it is used for testing optimization algorithms.
- Saddle: Represents a saddle-shaped surface, challenging algorithms to navigate through complex landscapes.
- Levy: A function with a global minimum in a Gaussian-shaped basin, testing algorithms' ability to locate the optimal point in a smooth landscape.

2. Structure of Interface

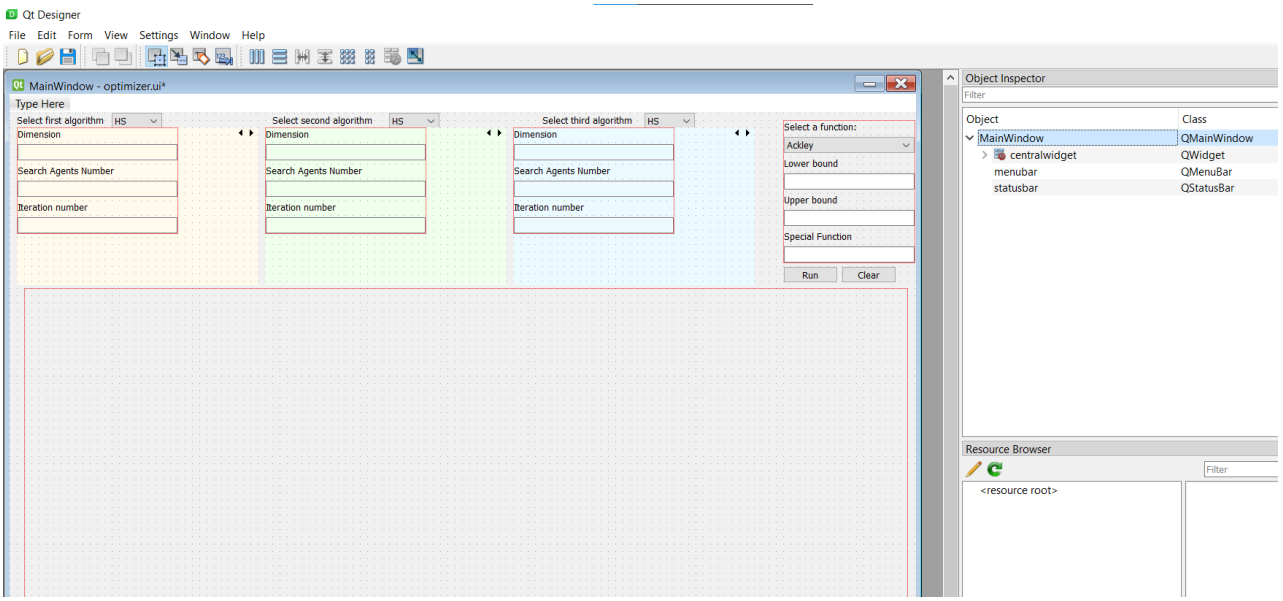

Figure 1. MainWindows, centralWidget, menubar, statusbar

QT designer has been used for interface design. In MainWindow, there are three properties called centralWidget, menubar and statusbar.

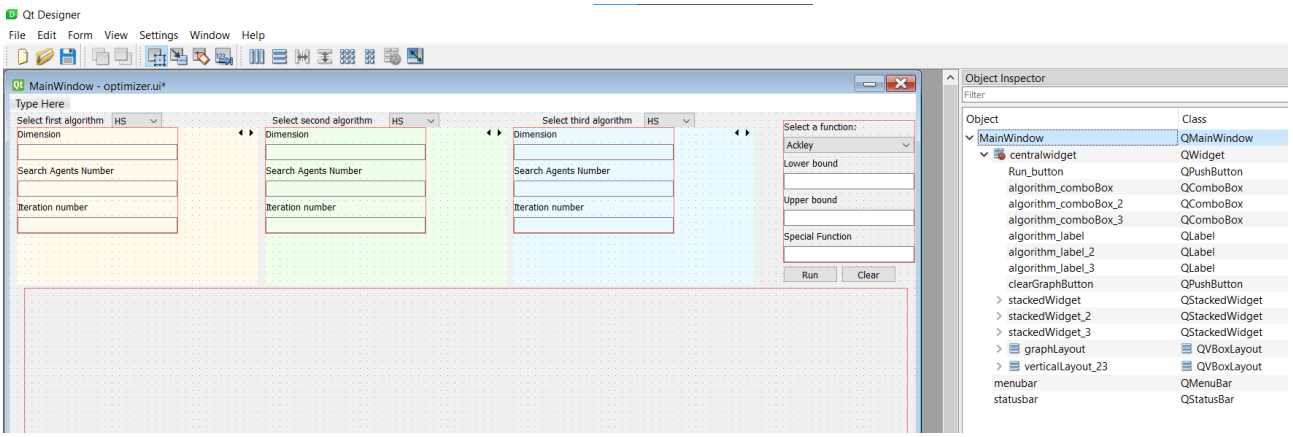

Figure 2. QPushButton, clearGraphButton, QComboboxes,

In centralWidget, there are two QPushButton which represents run_button to run algorithms and clearGraphButton to clear the graph that drawed before. To select three algorithms, there are three QCombobox called algorithm_comboBox, algorithm_comboBox_2, algorithm_comboBox_3.

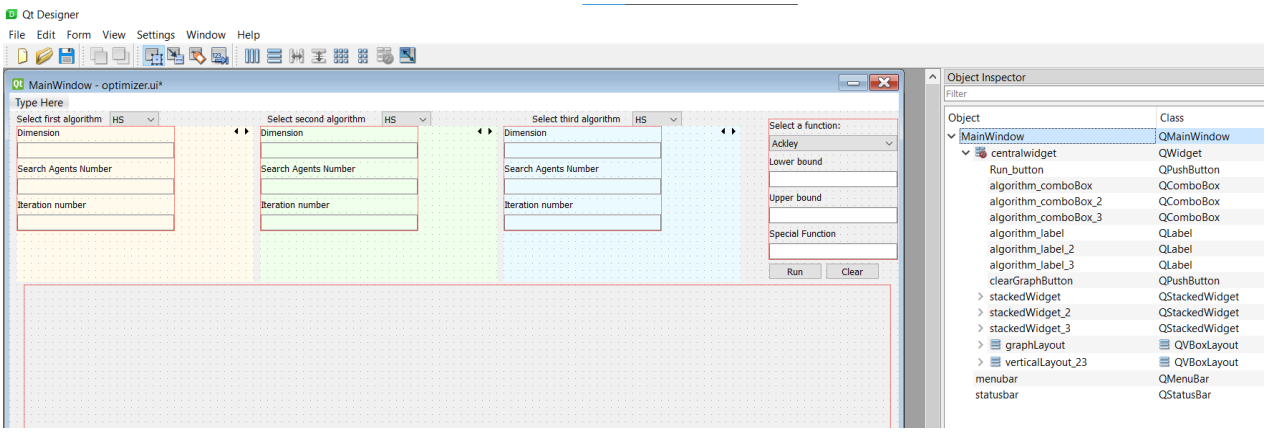

Figure 3. Labels, QStackWidgets

For each algorithms there are three label called algorithm_label, algorithm_label_2, algorithm_label_3. Each algorithms input values are stored in QStackedWidget called stackedWidget, stackedWidget_2, stackedWidget_3. Each QStackedWidget dynamically changes according to the selected algorithm in QComboBoxes.

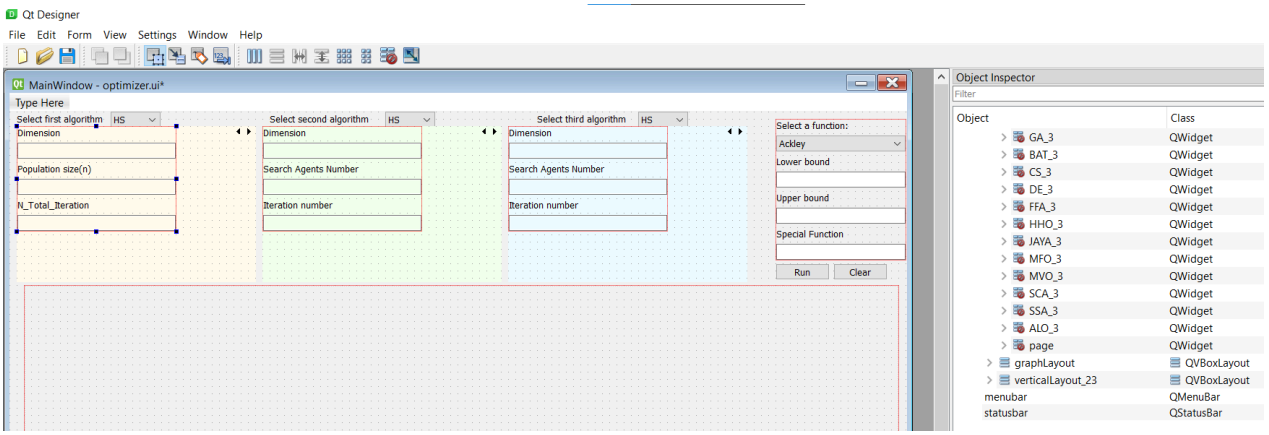

Figure 4. QWidgets, QVBox

Algorithms' inputs are stored in QWidget respective to their characteristic input values. To draw a graph related to selected functions and algorithms, there is QVBox called graphWidget. Which represents graph related to desired input values.

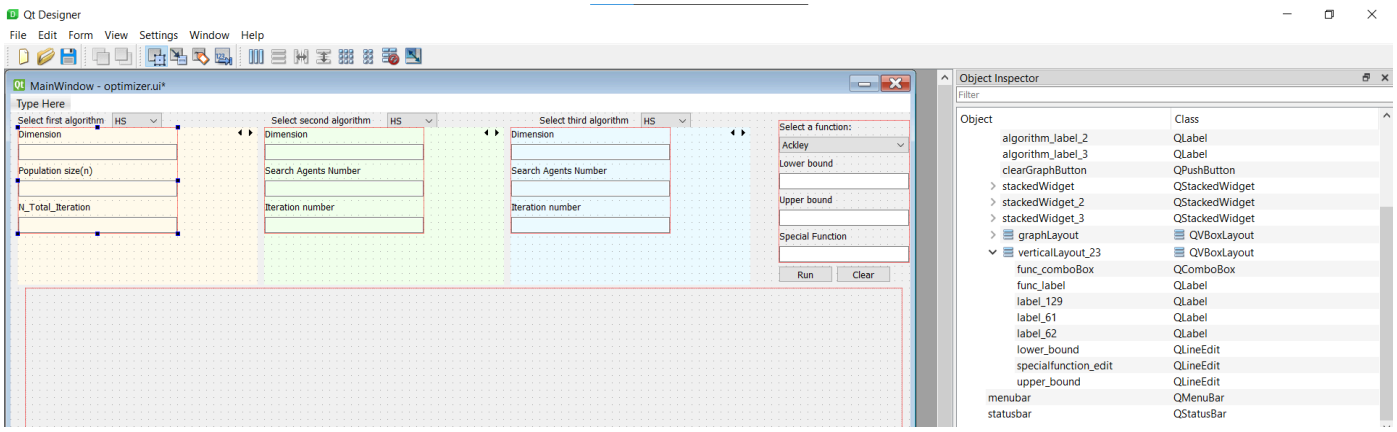

Figure 5. QVLayout, QLineEdits

Finally, there is QVLayout called verticalLayout_23. Inside verticalLayout_23, there are label and func_Combobox to select desired functions. Furthermore three QLineEdits to enter lower_bound, upper_bound with their labels.

3. Codes Explained

Our project consists of 19 optimization algorithms, namely GA, GWO, HS, PSO, SA, BAT, CS, DE, FFA, HHO, JAYA, MFO, MVO, SCA, SSA, ALO, and WOA. Each of these algorithms includes customized optimization methods that aim to optimize the relevant function effectively.

To provide an easy-to-use interface, we designed the graphical interface (UI) of our project using Qt Designer. The main window where the user can perform operations belongs to the MainWindow class, which extends the Ui_MainWindow class. Through this interface, the user can select various optimization algorithms and functions, set the relevant parameters, and run the algorithms with ease.

```
self.algorithm_comboBox.currentIndexChanged.connect(self.toggle_layer)
self.algorithm_comboBox_2.currentIndexChanged.connect(self.toggle_layer)
self.algorithm_comboBox_3.currentIndexChanged.connect(self.toggle_layer)
```

Figure 6. When Combobox Triggered where StackWidget changes

```
def toggle_layer(self):


        self.stackedWidget.setCurrentIndex(self.algorithm_comboBox.currentIndex())
        self.stackedWidget_2.setCurrentIndex(self.algorithm_comboBox_2.currentIndex())
        self.stackedWidget_3.setCurrentIndex(self.algorithm_comboBox_3.currentIndex())
```

Figure 7. Toggle Layer Function

The parameters of the algorithms can be entered in the fields located within a widget named StackWidget. Whenever the current index in the comboboxes changes, the function called toggle_layer is triggered. This function opens the field for the relevant algorithm in the StackWidget.

```
class MatplotlibWidget(FigureCanvas):
    def __init__(self, parent=None, width=5, height=4, dpi=100):
        fig, self.axes = plt.subplots(figsize=(width, height), dpi=dpi)
        super(MatplotlibWidget, self).__init__(fig)
        self.setParent(parent)
```

Figure 8. Class for MatplotlibWidget

The MatplotlibWidget class represents a PyQt5 canvas containing the Matplotlib figure. This canvas can be embedded within the PyQt5 application and integrate plots drawn by Matplotlib into the PyQt5 interface.

```
self.func_comboBox.currentIndexChanged.connect(self.bounds_select)
```

Figure 9. When function combobox triggered call bounds_select function

```
def bounds_select(self):
    index=self.func_comboBox.currentIndex()
    default_lower_bound = str(bounds[index][0])
    default_upper_bound = str(bounds[index][1])

    self.lower_bound.setText(default_lower_bound)
    self.upper_bound.setText(default_upper_bound)
```

Figure 10. bounds_select function

This code triggers the bounds_select method to execute when the user changes the selection of func_comboBox. The currentIndexChanged signal is used to detect the change. The bounds_select method extracts the lower and upper bounds values from the bounds list in the selected index and updates the lower_bound and upper_bound texts accordingly. The purpose of this method is to display the specific lower and upper limits of the selected function, making it easier for the user to visualize and understand. When the user chooses a function, this method automatically updates the limits on the UI.

```
self.Run_button.clicked.connect(self.button_clicked)
```

Figure 11. When run_button clicked

```
def button_clicked(self):
    combo_boxes = [self.algorithm_comboBox, self.algorithm_comboBox_2, self.algorithm_comboBox_3]

    for index, combo_box in enumerate(combo_boxes, start=1):
        if combo_box.currentText() == "PSO":
            self.run_algorithm(index, "PSO")

        elif combo_box.currentText() == "SA":
            self.run_algorithm(index, "SA")

        elif combo_box.currentText() == "GWO":
            self.run_algorithm(index, "GWO")

        elif combo_box.currentText() == "HS":
            self.run_algorithm(index, "HS")
```

Figure 12. button_clicked function

This block of code takes in the name of an algorithm and the type of combobox selected (1, 2, or 3) from the algorithm_comboBox, algorithm_comboBox_2, and algorithm_comboBox_3 comboboxes.

Once the relevant algorithm has been selected, the run_algorithm method is called to execute it. This feature allows the user to select multiple algorithms via different comboboxes and to run each one sequentially. The index of the relevant combobox and the name of the selected algorithm are passed to the run_algorithm method to execute the selected algorithm.

```
def run_algorithm(self, algorithm_type, algorithm_name):



    obj_func = self.function_select()

    if algorithm_name == "PSO" :  # PSO
        if algorithm_type == 1:
            pop_size = int(self.pso_pop_size.text())
            num_gen = int(self.pso_num_gen.text())
            dim=int(self.pso_dim.text())
        elif algorithm_type == 2:
            pop_size = int(self.pso_pop_size_2.text())
            num_gen = int(self.pso_num_gen_2.text())
            dim=int(self.pso_dim_2.text())
        else:
            pop_size = int(self.pso_pop_size_3.text())
            num_gen = int(self.pso_num_gen_3.text())
            dim=int(self.pso_dim_3.text())

        lower_bound = float(self.lower_bound.text())
        upper_bound = float(self.upper_bound.text())
        self.sol = PSO(obj_func, lower_bound, upper_bound, dim, pop_size, num_gen)
        self.update_graph(algorithm_type)
```

Figure 13. run_algorithm function

This piece of code contains the run_algorithm function, which is called with a specific algorithm type (index indicating which combobox the relevant algorithm is selected from) and name. Depending on the selected algorithm type and name, this function initializes the corresponding optimization algorithm and then updates the solution graph in the graphical interface.

Each if block in this code represents a type of optimization algorithm and initializes the corresponding algorithm by taking the necessary parameters depending on the algorithm selected by the user. These parameters are taken from the corresponding text boxes in the user interface. This function creates a solution object (self.sol) containing the results of the initialized algorithm and then updates the solution graph in the graphical interface by calling the update_graph function.

```
def function_select(self):
        func=self.func_comboBox.currentIndex()
        if func==0:
            return functions.selectFunction(Functions.ackley)
        elif func==1:
            return functions.selectFunction(Functions.griewank)
        elif func==2:
            return functions.selectFunction(Functions.schwefel)
        elif func==3:
            return functions.selectFunction(Functions.rastrigin)
        elif func==4:
            return functions.selectFunction(Functions.sphere)
```

Figure 14. Function_select function

This feature enables users to choose an optimization target function based on a function they select in the interface. The "self.func_comboBox.currentIndex()" method retrieves the index of the selected function in the drop-down list and then selects a suitable target function based on that index.

```
def update_graph(self,algorithm_type):
    sns.lineplot(x=self.sol.x, y=self.sol.y, ax=self.matplotlibWidget.axes, label=self.get_algorithm_label(algorithm_type))

    self.matplotlibWidget.axes.set_xlabel('Iteration count')
    self.matplotlibWidget.axes.set_ylabel('Fitness value')

    self.matplotlibWidget.axes.legend(loc="upper right")

    self.matplotlibWidget.draw()
```

Figure 15. update_graph function

This code is used to create a graph that shows the solution obtained from certain algorithms. By utilizing the Seaborn library, the solution is visually represented using data from self.sol.x (the number of iterations) and self.sol.y (fitness values). This enables us to observe how the algorithm's solution changes with the number of iterations.

The graph includes labels for both the x and y axes. To complete the drawing, Seaborn's legend function is used. The label of the algorithm that was used for the drawing is added with the self.get_algorithm_label(algorithm_type) function, and then positioned. Finally, the updated graph is displayed using self.matplotlibWidget.draw().

```python
def get_algorithm_label(self,algorithm_type):
    if algorithm_type == 1:
        return f"{self.algorithm_comboBox.currentText()}"
    elif algorithm_type == 2:
        return f"{self.algorithm_comboBox_2.currentText()}"
    elif algorithm_type == 3:
        return f"{self.algorithm_comboBox_3.currentText()}"
```

Figure 16. get_algorithm_label function

According to the algorithm_type provided as a parameter, it returns the label by taking the algorithm name from the relevant ComboBox. If algorithm_type is 1, it selects the algorithm name from self.algorithm_comboBox, if 2, it selects the algorithm name from self.algorithm_comboBox_2, and if 3, it selects the algorithm name from self.algorithm_comboBox_3.

```python
self.clearGraphButton.clicked.connect(self.clear_graph)
```

Figure 17. When clear_graph button is triggered

```python
def clear_graph(self):
        # Clear the graph in the MatplotlibWidget
        self.matplotlibWidget.axes.clear()
        self.matplotlibWidget.draw()
```

Figure 18. Clear_graph function

This function is triggered when the clear button is clicked and clears the canvas in the interface.

4. Tests Performed And Screen Photos

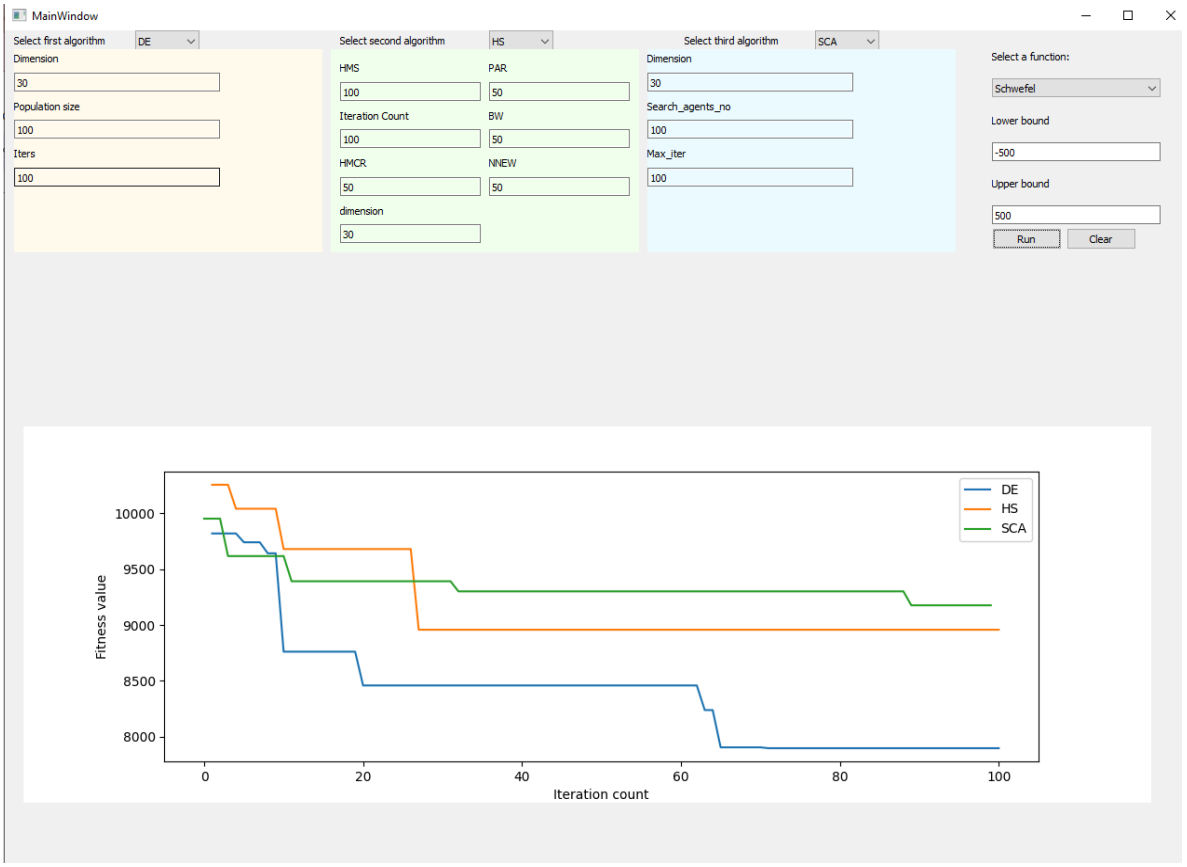

Figure 19. DE-HS-SCA

In Figure 19, Differential Evolution, Harmony Search and Sine Cousine Algorithms were run.

For DE: Population Size=100; Iters=100; Dimension=30,

For HS: HMS=100; PAR=50; Iteration Count=100; BW=50; HMCR=50; NNEW=50; Dimension=30,

For SCA: Dimension=30, Search Agents No=100; Iteration Count is given as 100.

All three given algorithms were run in the range of lower bound=-500, upper bound=500 using the Schwefel Function.

All 3 algorithms were run at 100 iterations, 30 dimension and 100 population size. The algorithm that reached the lowest and best value was the Differential Evolution Algorithm. They did not show any improvement after the Harmony Search Algorithm was fixed in the 20-40 iteration range. Harmony Search Algorithm was fixed in the 80-100 iteration range. Harmony Search Algorithm gave the worst result.

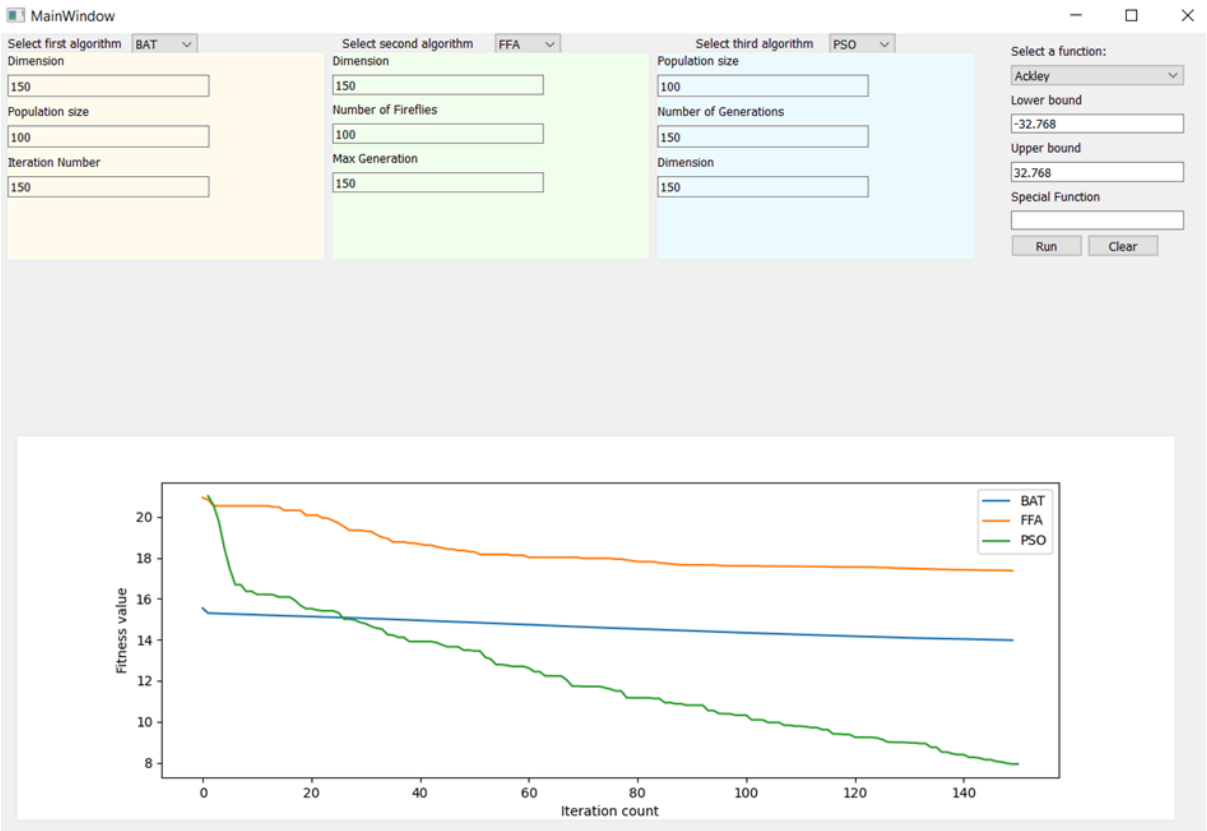

Figure 20. BAT-FFA-PSO

In Figure 20, Bat, Firefly and Particle Swarm Optimization Algorithms were run.

For BAT: Dimension=150; Population Size=100; Iteration Number=150,

For FFA: Dimension=150; Number of Fireflies=100; Iteration Number=150,

For PSO: Dimension=150; Population Size=100; Iteration Number=150.

All three given algorithms were run in the range of lower bound=-32.768, upper bound=32.768 using the Ackley Function.

All three algorithms were run at 150 iterations, 150 dimension value and 100 population size value. The algorithm that achieved the lowest and best value was the Particle Swarm Optimization Algorithm. The Bat Algorithm showed slight improvement over 150 iterations. While the Firefly Algorithm decreased rapidly in the 0-40 iteration range, it showed a slight improvement in the 40-150 iteration range.

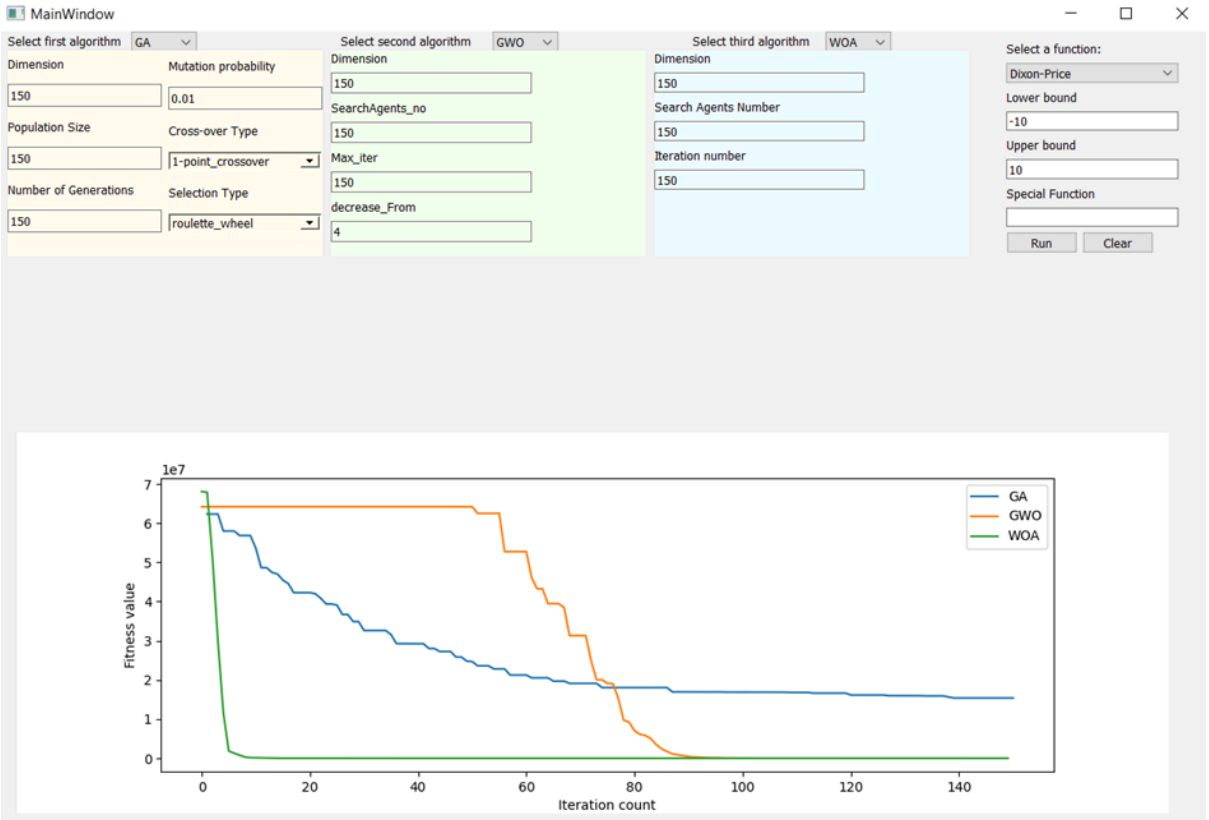

Figure 21. GA-GWO-WOA

In Figure 21, Genetic, Gray Wolf and Whale Optimization Algorithms were run.

For GA: Dimension=150; Population Size=150; Iteration Number=150; Mutation Probability=0.01; Cross-Over Type=1-point cross-over; Selection Type=Roulette Wheel,

For GWO: Dimension=150; Search Agents No=150; Iteration Number=150; Decrease from=4,

For WOA: Dimension=150; Search Agents Number =150; Iteration Number=150.

All three given algorithms were run in the range of lower bound=-10, upper bound=10 using the Dixon-Price Function.

All three algorithms were run at 150 iterations, 150 dimension value and 150 population size value. There are two algorithms that arrive at the lowest and best value. These were Gray Wolf and Whale Optimization Algorithms. The Whale Optimization Algorithm decreased significantly in the 0-20 iteration range and the fitness value was fixed at 0. Gray Wolf Algorithm could not improve in the 0-60 iteration range, but the fitness value decreased to 0 in the 60-120 iteration range. While the Genetic Algorithm showed rapid improvement in the 0-60 iteration range, it improved slowly in the 60-150 iteration range.

4.1. Results and Discussion

4.1.1. Project Results and Achievements

19 different optimization algorithms and 18 different functions are used within the scope of the project. The input parameters of each algorithm are designed to be received from the user. These parameters consist of dimension, iteration number, population size and other algorithm-specific parameters. 3 different algorithms can be run at the same time. These algorithms are run by choosing one of 18 functions. When three algorithms run at the same time are given equal dimension, population size and iteration number, these algorithms can be compared. The fitness values of the 3 algorithms run in each iteration are shown in a chart and the improvement amounts in the algorithms are compared.

4.1.2. Challenges and Solutions

Various difficulties were encountered during the project construction stages. These difficulties are: finding the various algorithms and functions that need to be integrated, receiving the parameters of the integrated algorithms from the user, transferring the fitness values of the algorithms to the graph during operation, and allowing the user to provide special function input. The necessary algorithms and functions were integrated by doing sufficient research on the internet and by writing the code of the algorithms that could not be found by ourselves. The parameters of each algorithm were reviewed one by one and the parameters were received from the user through the interface created using PyQt5 Designer. Graph drawing was carried out using Matplotlib. We are continuing to work on enabling the user to provide special function input.

4.2. Future Studies

4.2.1. Development Potential of the Project

The project is designed in a way that the algorithms and functions developed as new research and development studies are carried out can be integrated. It can be easily decided which parameters will be taken from the user. Development can be easily done by someone who knows the Python programming language.

4.2.2. Additional Features and Improvements

New additional features can be added to the project. Apart from the 18 different functions offered to the user, a special function feature is being worked on where the user can write his own code via the interface. It is designed so that various features can be added according to the needs of the user and the needs of the project being worked on.

References

https://seyedalimirjalili.com/projects
https://github.com/7ossam81/EvoloPy/tree/master/optimizers
https://github.com/thieu1995/mealpy/tree/master/mealpy