January 2024

Int. to Heuristic Algorithms

Comparing Algorithms with GUI

Group 2

Burak CAFERLER - 152120201144

Aylin ÜNAL - 152120181117

Adem MAVANACI - 152120191040

İrem GÜNEŞ- 151320183048

INTRODUCTION

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.

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.

PYQT-5

- PyQt5 is a 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.

Project Description

Heuristic Algorithms

1.Genetic Algorithm (GA):
- Mimics biological evolutionary processes.
- Provides a general solution for various problems.
- Risk of getting trapped in local optima.

2.Grey Wolf Optimizer (GWO):
- Models the hunting behavior of wolf packs.
- Performs well in complex search spaces.

3.Harmony Search (HS):
- Inspired by musical composition processes.
- Requires little parameter adjustment.
- Noted for its simple structure.

4.Particle Swarm Optimization (PSO):
- Imitates movements of bird flocks or fish schools.
- Fast and efficient search structure.
- May struggle in complex search areas.

5.Simulated Annealing (SA):
- Mimics thermal processing of metals.
- Yields good results in broad search areas.
- High computational intensity.

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

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

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

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

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

11.Moth Flame Optimization (MFO):
- Mimics the flight pattern of moths towards a flame.

12.JAYA Algorithm:
- Simple, parameter-less, and effective optimization algorithm.

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

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

15.Differential Evolution (DE):
- Powerful optimization method based on evolutionary strategy.

16.Cuckoo Search (CS):
- Models the parasitic egg-laying behavior of cuckoo birds.

17.Bat Algorithm:
- Imitates the echolocation behavior of bats.

18.Grasshopper Optimisation Algorithm

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.

Optimization Functions

Zakharov

- Another function known for its complex structure.

Rosenbrock

- Characterized by a narrow valley, requiring algorithms to follow a specific path.

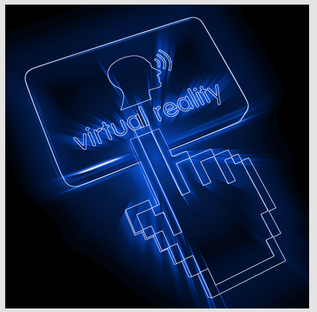
Dixon-Price

- Used to test the performance of evolutionary algorithms.

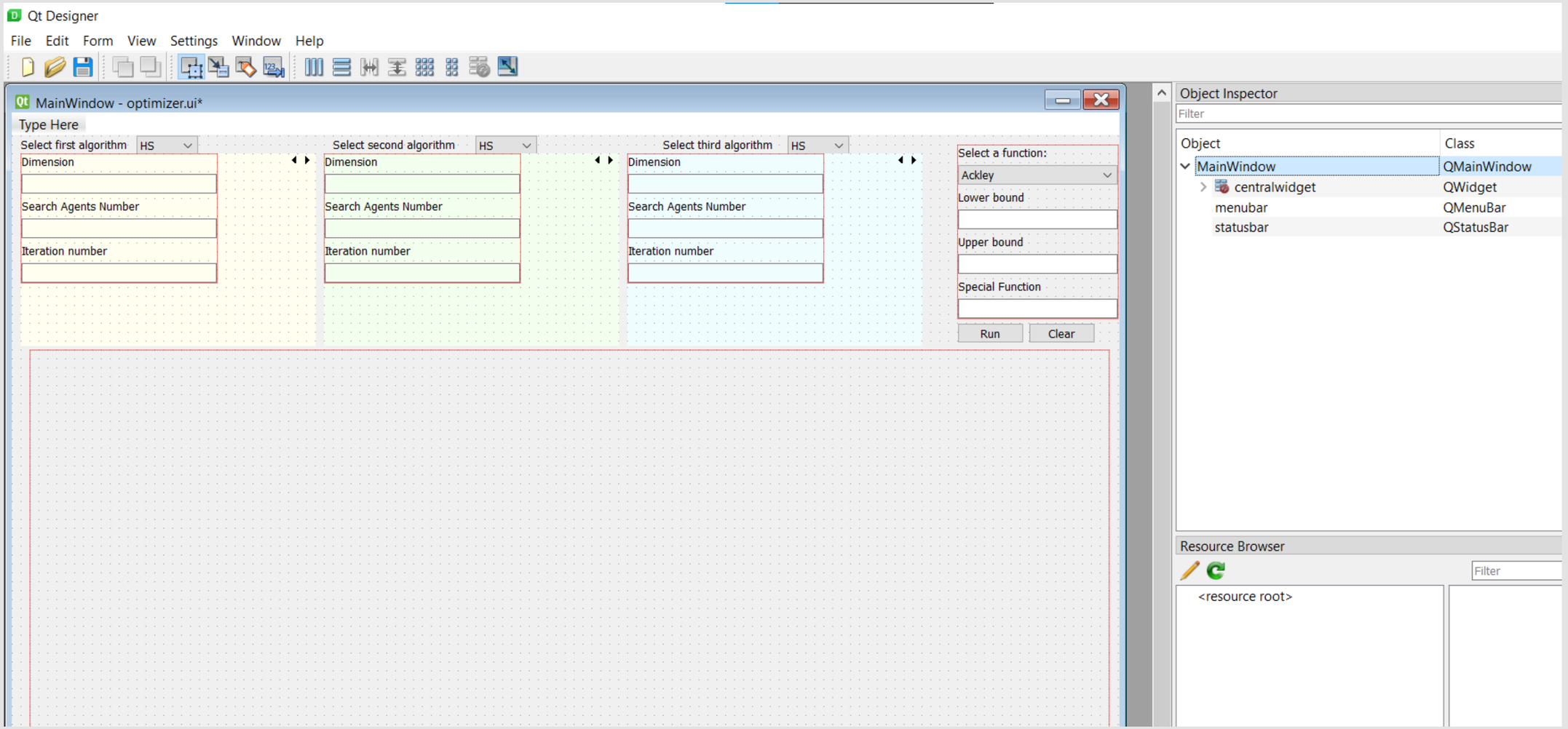
Levy

A function with a global minimum in a Gaussian-shaped basin, testing algorithms' ability to locate the optimal point in a smooth landscape.

More functions are covered in project report

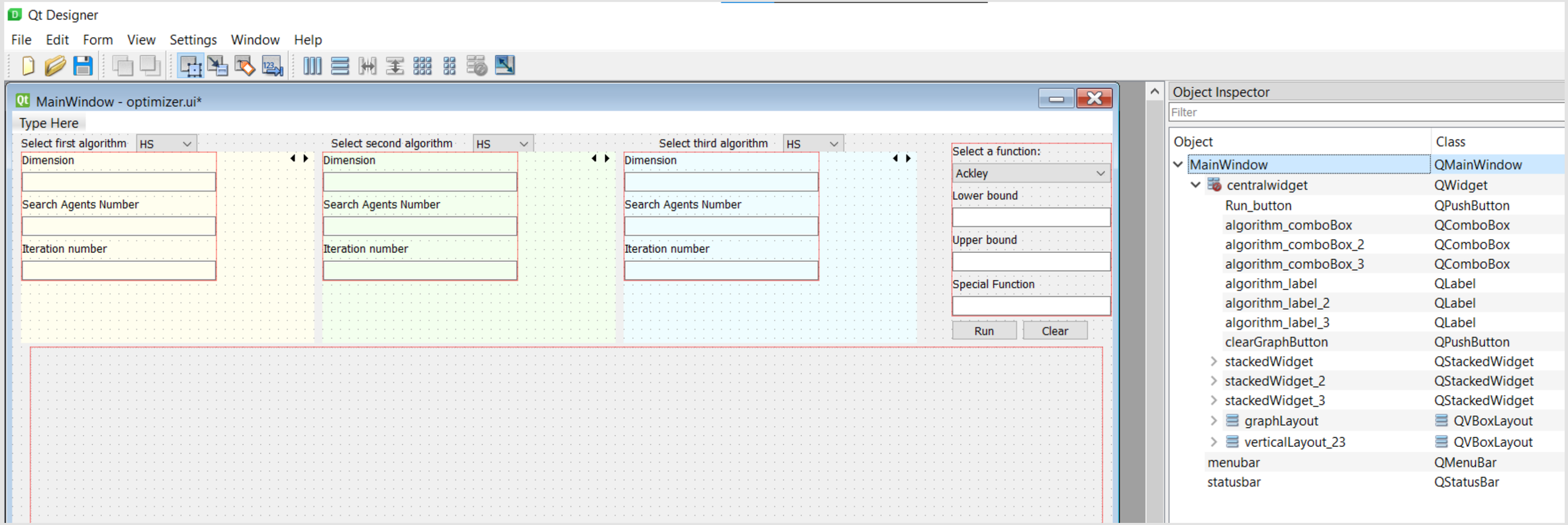


Structure of Interface



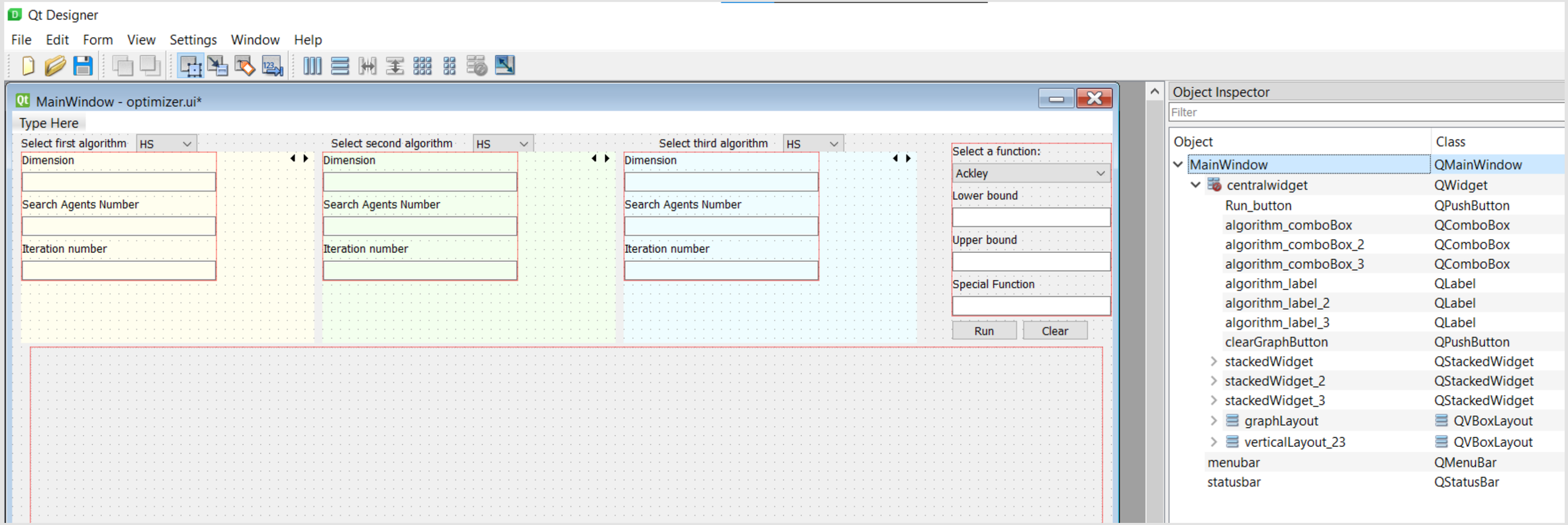
QT designer has been used for interface design. In MainWindow, there are three properties called centralWidget, menubar and statusbar.

Structure of Interface



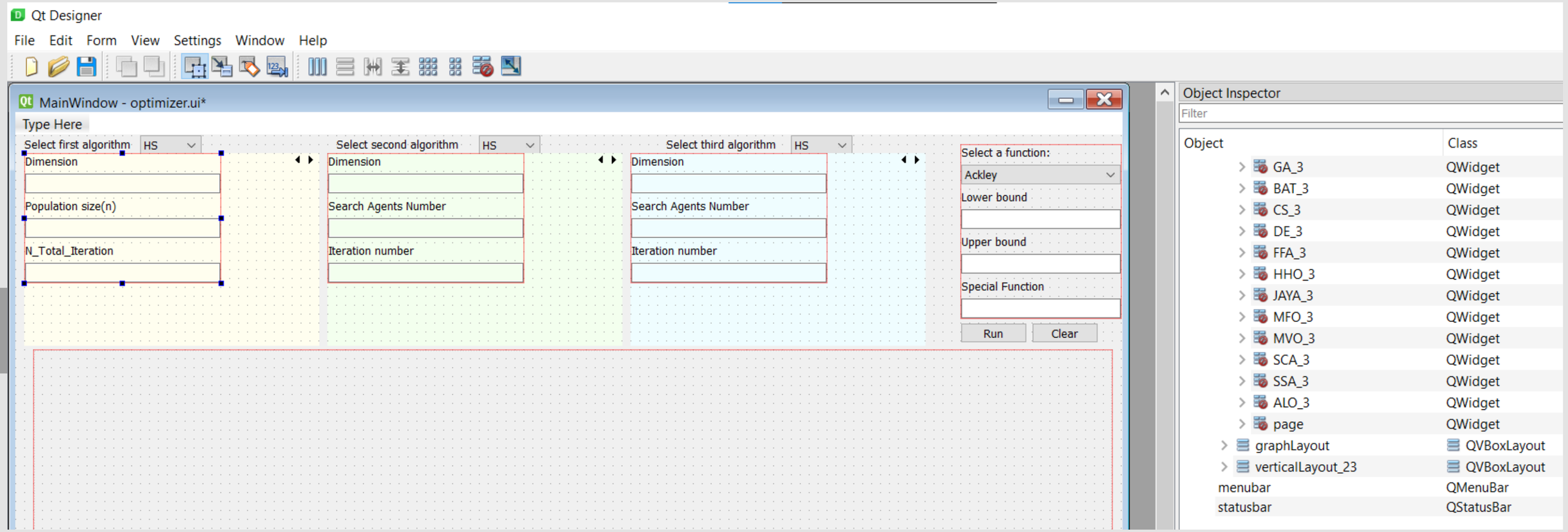
In centralWidget, there are two QPushButton which represents run_button to run algorithms and clearGraphButton to clear the graph that drawed before. To select three algorithms, there are three QCombobox called algorithm_comboBox, algorithm_comboBox_2, algorithm_comboBox_3.

Structure of Interface



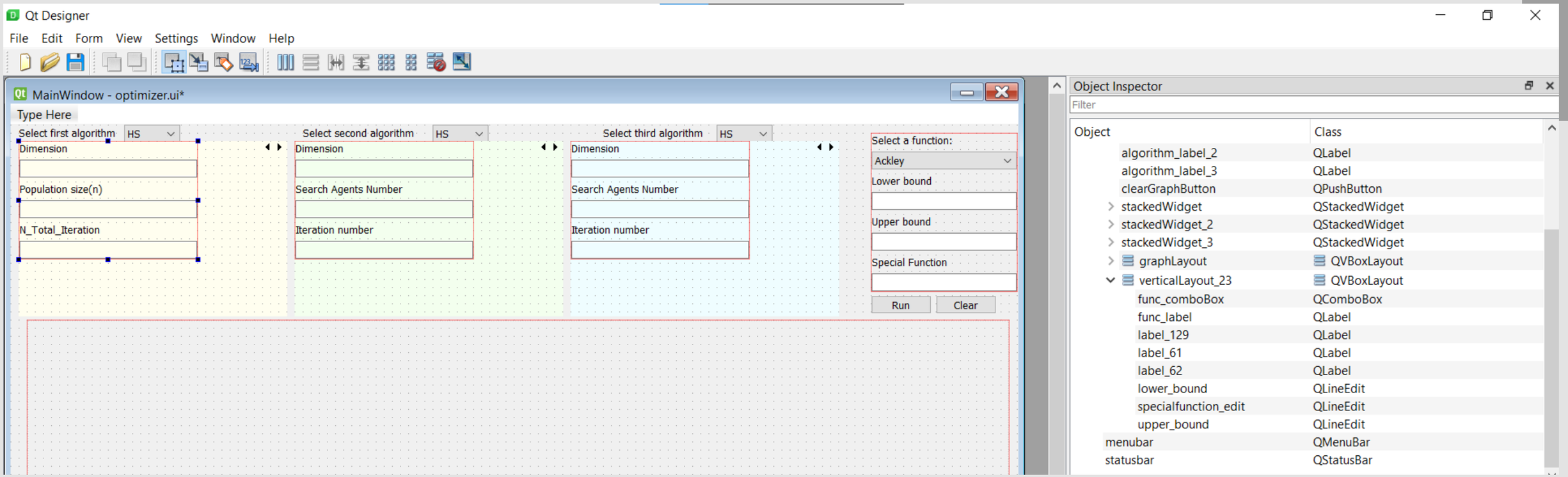
For each algorithms there are three label called algorithm_label, algorithm_label_2, algorithm_label_3. Each algorithms input values are stored in QStackedWidget called stackedWidget, stackedWidget_2, stackedWidget_3. Each QStackedWidget dynamically changes according to the selected algorithm in QComboBoxes.

Structure of Interface



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.

Structure of Interface



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.

Codes Explained

```
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)
```

```
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())
```

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.

Codes Explained

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

```
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)
```

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. When the user chooses a function, this method automatically updates the limits on the UI.

Codes Explained

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

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.

Codes Explained

```
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)
```

This piece of code contains the run_algorithm function, which is called with a specific algorithm type 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.

Codes Explained

```
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)
```

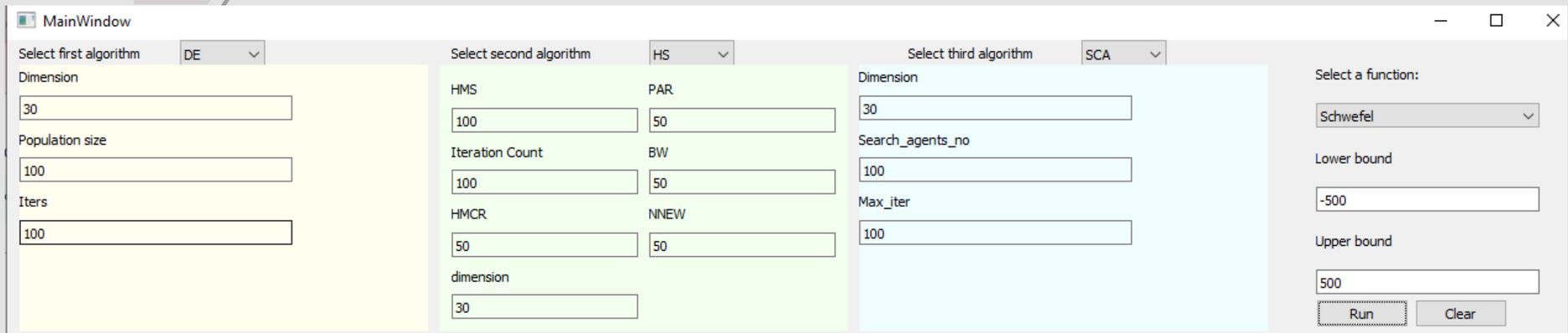
This feature enables users to choose an optimization target function based on a function they select in the interface. The "self.func_comboBox.currentIndex()" method retrieves the index of the selected function in the drop-down list and then selects a suitable target function based on that index.

Codes Explained

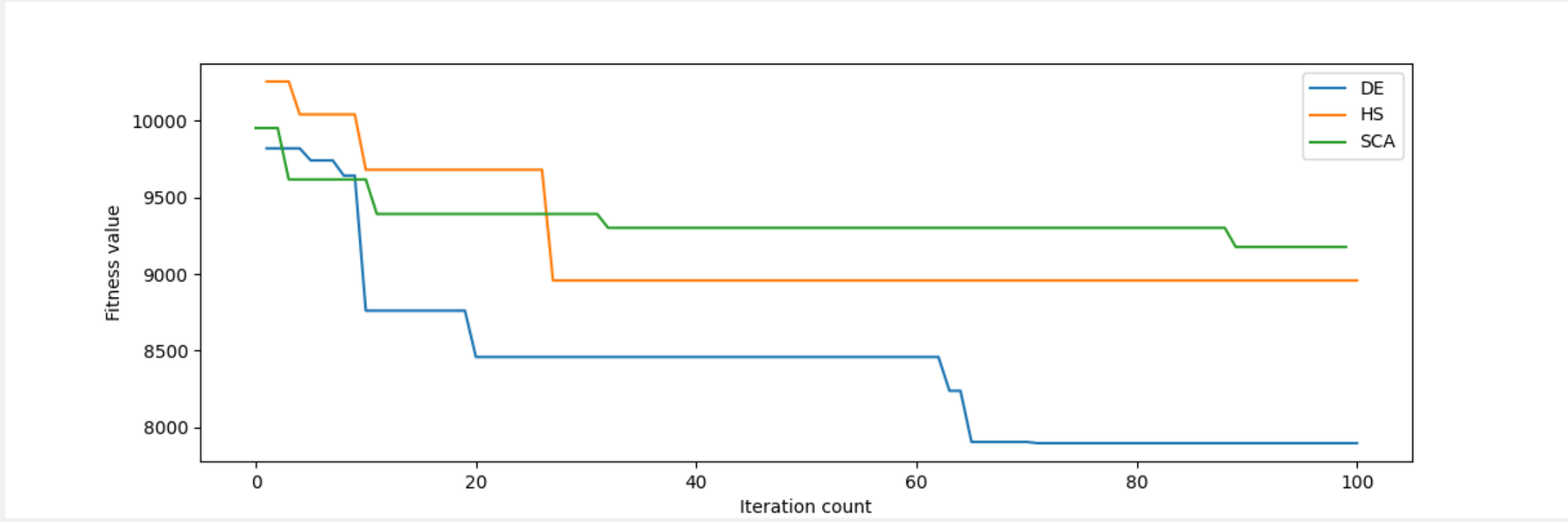
```
def update_graph(self,algorithm_type):
    sns.lineplot(x=self.sol.x, y=self.sol.y, ax=self.matplotlibWidget.axes, label=self.get_algorithm_label(algorithm_type))

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

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

    self.matplotlibWidget.draw()
```

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.

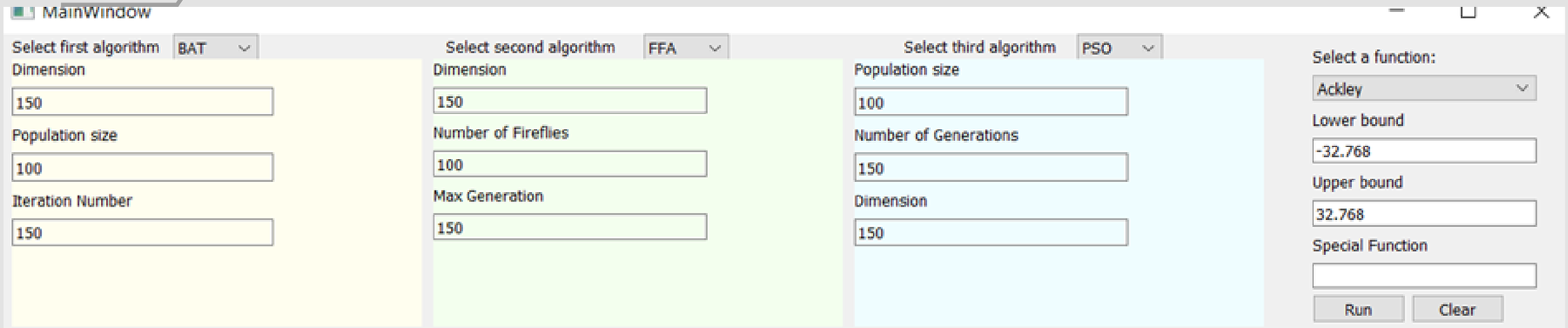
Tests Performed and Screen Photos



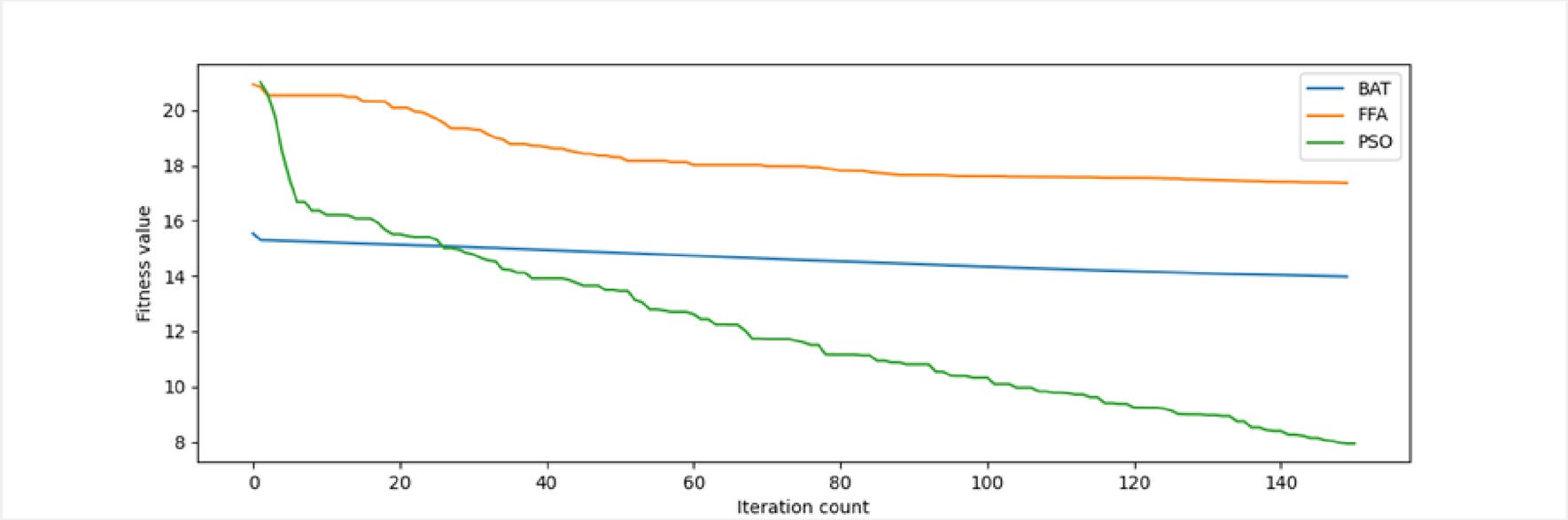
Differential Evolution, Harmony Search, and Sine Cousine Algorithms



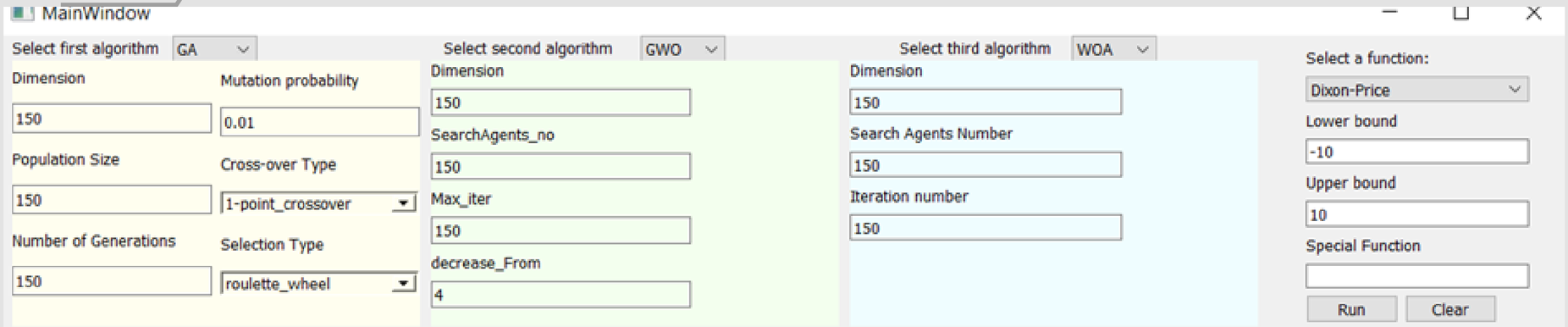
Tests Performed and Screen Photos



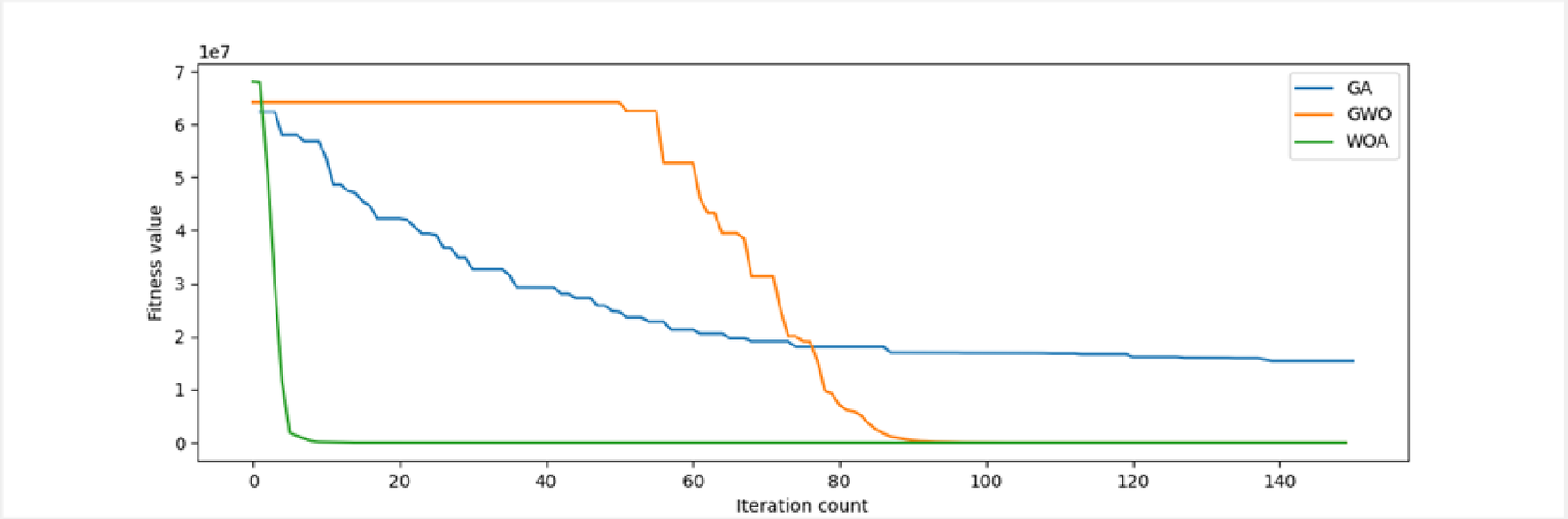
Bat, Firefly and Particle Swarm Optimization Algorithms



Tests Performed and Screen Photos



Genetic, Gray Wolf and Whale Optimization Algorithms



Project Results and Achievements

1

19 different optimization algorithms and 18 different functions are used within the scope of the project

2

The input parameters of each algorithm are designed to be received from the user.

3

3 different algorithms can be run at the same time.

4

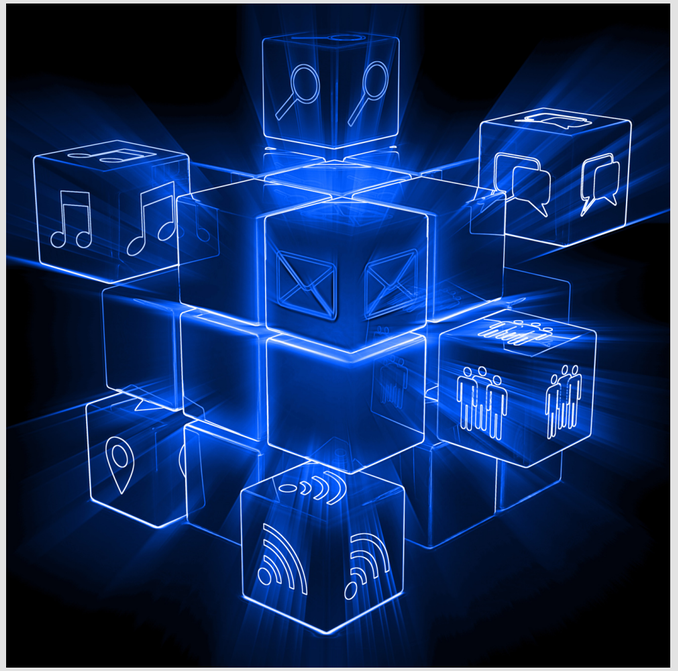
The fitness values of the 3 algorithms run in each iteration are shown in a chart and the improvement amounts in the algorithms are compared.

Challenges and Solutions

- 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.
- Allowing the user to provide special function input.

Future Studies

- 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.
- New additional features can be added to the project.
- A special function feature is being worked on where the user can write his own code via the interface.



January 2024

GROUP 2

THANKYOU