User Manual for EDOLAB: Running, Configuring, and Extending the Platform

MAI PENG[*], School of Automation, China University of Geosciences, Wuhan, Hubei Key Laboratory of Advanced Control and Intelligent Automation for Complex Systems, and Engineering Research Center of Intelligent Technology for Geo-Exploration, Ministry of Education, China

DELARAM YAZDANI, Liverpool Logistics, Offshore and Marine (LOOM) Research Institute, Faculty of Engineering and Technology, Liverpool John Moores University, United Kingdom

ZENENG SHE, School of Computer Science and Technology, Harbin Institute of Technology, China

DANIAL YAZDANI[*], Faculty of Engineering & Information Technology, University of Technology Sydney, Australia

WENJIAN LUO, Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, School of Computer Science and Technology, Harbin Institute of Technology and Peng Cheng Laboratory, China

CHANGHE LI, School of Artificial Intelligence, Anhui University of Sciences & Technology, China

JUERGEN BRANKE, Information Systems Management and Analytics in Warwick Business School, University of Warwick, United Kingdom

TRUNG THANH NGUYEN, The Liverpool Logistics, Offshore and Marine (LOOM) Research Institute, Faculty of Engineering and Technology, Liverpool John Moores University, United Kingdom

AMIR H. GANDOMI, Faculty of Engineering & Information Technology, University of Technology Sydney, Australia and University Research and Innovation Center (EKIK), Obuda University, Hungary

SHENGXIANG YANG, Institute of Artificial Intelligence (IAI), School of Computer Science and Informatics, De Montfort University, United Kingdom

YAOCHU JIN, Department of Artificial Intelligence, School of Engineering, Westlake University, China

XIN YAO, School of Data Science, Lingnan University, Hong Kong SAR and The Center of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, United Kingdom

[*]For more assistance, please feel free to reach out to Danial Yazdani at danial.yazdani@gmail.com or Mai Peng at pengmai@cug.edu.cn.

Authors' addresses: Mai Peng, pengmai@cug.edu.cn, School of Automation, China University of Geosciences, Wuhan, Hubei Key Laboratory of Advanced Control and Intelligent Automation for Complex Systems, and Engineering Research Center of Intelligent Technology for Geo-Exploration, Ministry of Education, China, 430074; Delaram Yazdani, delaram.yazdani@yahoo.com, Liverpool Logistics, Offshore and Marine (LOOM) Research Institute, Faculty of Engineering and Technology, Liverpool John Moores University, Liverpool, United Kingdom, L3 3AF; Zeneng She, 20s151103@stu.hit.edu.cn, School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China, 518055; Danial Yazdani, danial.yazdani@gmail.com, Faculty of Engineering & Information Technology, University of Technology Sydney, Ultimo, Australia, 2007; Wenjian Luo, luowenjian@hit.edu.cn, Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, School of Computer Science and Technology, Harbin Institute of Technology and Peng Cheng Laboratory, Shenzhen, China, 518055; Changhe Li, changhe.lw@gmail.com, School of Artificial Intelligence, Anhui University of Sciences & Technology, Hefei, China, 230026; Juergen Branke, Juergen.Branke@wbs.ac.uk, Information Systems Management and Analytics in Warwick Business School, University of Warwick, Coventry, United Kingdom, CV4 7AL; Trung Thanh Nguyen, T.T.Nguyen@ljmu.ac.uk, The Liverpool Logistics, Offshore and Marine (LOOM) Research Institute, Faculty of Engineering and Technology, Liverpool John Moores University, Liverpool, United Kingdom, L3 3AF; Amir H. Gandomi, Gandomi@uts.edu.au, Faculty of Engineering & Information Technology, University of Technology Sydney, Ultimo, Australia, 2007 and University Research and Innovation Center (EKIK), Obuda University, Budapest, Hungary, 1034; Shengxiang Yang, syang@dmu.ac.uk, Institute of Artificial Intelligence (IAI), School of Computer Science and Informatics, De Montfort University, Leicester, LE1 9BH, United Kingdom; Yaochu Jin, jinyaochu@westlake.edu.cn, Department of Artificial Intelligence, School of Engineering, Westlake University, Hangzhou, China, 301130;

This user manual provides a comprehensive guide for running, configuring, and extending EDOLAB, an open-source MATLAB platform for evolutionary dynamic optimization algorithms (EDOAs). EDOLAB includes two key modules: the *Education* module, which visualizes algorithm behavior over time, and the *Experimentation* module, designed for conducting experiments and comparing algorithms. The platform supports both GUI and non-GUI modes, offering flexibility for users. Additionally, instructions are provided for running EDOLAB in Octave, an open-source alternative to MATLAB. To begin, clone the EDOLAB project from the GitHub repository: [https://github.com/Danial-Yazdani/EDOLAB-MATLAB].

M-I ARCHITECTURE

EDOLAB is a function-based software implemented in MATLAB. The *MATLAB App Designer* was used to develop the GUI for EDOLAB. The software can be operated either with or without the GUI.

The root directory of EDOLAB includes the following:

- A .MLAPP file, which is the GUI developed using MATLAB App Designer.
- Two .m files: (1) `RunWithGUI.m`–the exported GUI .m file, and (2) `RunWithoutGUI.m`–a function for using EDOLAB without the GUI.
- Five folders:
  - **Algorithm**: This folder contains several sub-folders, each corresponding to an EDOA listed in Table 1. Each EDOA sub-folder generally includes several .m files: (1) `main_EDOA.m`–the main file that invokes and controls other EDOA functions, (2) `SubPopulationGenerator_EDOA.m`–a sub-population generator function that generates the sub-populations for the optimization component, (3) `IterativeComponents_EDOA.m`–a function containing the EDOA components that are executed every iteration, or when certain conditions are met, and (4) `ChangeReaction_EDOA.m`–a function that includes the change reaction components of the EDOA.
  - **Benchmark**: This folder contains a sub-folder for each benchmark generator included in EDOLAB. Each benchmark sub-folder includes two .m files: (1) `BenchmarkGenerator_Benchmark.m`–responsible for setting up the benchmark and generating environments, and (2) `fitness_Benchmark.m`–includes the baseline function of the benchmark for calculating function values. These functions are called by three .m files located in the Benchmark folder: (1) `BenchmarkGenerator.m`–invokes the initializer and generator of the benchmark problem selected by the experimenter, (2) `fitness.m`–calls the related benchmark's baseline function for calculating fitness values, manages benchmark parameters, counters, and flags, and gathers the information needed to calculate performance indicators, and (3) `EnvironmentVisualization.m`–an environment visualization function responsible for depicting the problem landscape in the educational module.
  - **Results**: For each experiment, EDOLAB generates an Excel file (if selected by the user) that contains the results, statistics, and experiment settings. These output Excel files are stored in this folder.
  - **Utility**: This folder includes various utility functions, such as those for generating output files and figures, which are located in the `Output` sub-folder.

Xin Yao, xiny@sustech.edu.cn, School of Data Science, Lingnan University, Hong Kong SAR and The Center of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, University of Birmingham, Birmingham, United Kingdom, B15 2TT.

- **Octave_compatibility**: This folder contains updated versions of key files modified for compatibility with Octave, including RunWithoutGUI.m and several others. Users wishing to run EDOLAB in Octave should replace the corresponding files in the main EDOLAB directory with those provided in this folder.

Figure M-1 illustrates a general sequence diagram for running an EDOA in EDOLAB, which demonstrates how the platform operates. First, the user sets up an experiment using either the GUI or the RunWithoutGUI.m and initiates the run. The interface then invokes the main function of the selected algorithm (for example, main_AmQSO.m). At the start of the main function, the benchmark generator function (BenchmarkGenerator.m) is called. This function is responsible for initializing the benchmark and generating a sequence of environments based on the parameters defined by the user.

In EDOLAB's experimentation module, identical random streams are used when initializing problem instances and generating environmental changes in BenchmarkGenerator.m. As a result, with the same parameter settings, the same problem instance sequence (from the first environment to the last) is generated for all comparison algorithms. Using different random seeds in experiments can produce problem instances with varying characteristics and difficulty levels [Yazdani et al. 2021b], potentially leading to biased comparisons. In EDOLAB, we have addressed this issue by controlling the random streams. After generating the sequence of environments, the initial sub-population(s) or individuals are generated by the sub-population/individual generation function (for example, SubPopulationGenerator_AmQSO.m).

Afterward, the main loop of the EDOA is executed. In each iteration, the iterative components of the EDOA, such as the optimizer (e.g., PSO or DE), diversity control, and population management [Yazdani et al. 2020], are executed by calling the iterative components function (for example, IterativeComponents_AmQSO.m). In many EDOAs with adaptive sub-population numbers and/or population sizes, new individuals or sub-populations are generated when certain conditions are met [Yazdani et al. 2021a]. Additionally, some diversity and population management components may require the reinitialization of certain sub-populations or individuals. Therefore, if any sub-populations or individuals need to be (re)initialized during an iteration, the sub-population/individual generation function is called. The updated population is then returned to the main EDOA function.

At the end of each iteration, if the environment has changed, the change reaction components are called (for example, ChangeReaction_AmQSO.m). The main loop of the EDOA continues until the number of function evaluations (FE) reaches its predefined maximum value ($FE_{\max}$). This procedure is repeated for the specified number of runs (RunNumber). Afterward, the results are processed, including the calculation of performance indicators. The results, along with any collected data, are then sent to output generator functions responsible for creating output plots, tables, and files. Finally, the output tables and figures are returned to the interface.

M-II RUNNING

As previously mentioned, EDOLAB can be operated either with or without a GUI. In the following, we describe both methods of use.

M-II.1 Using EDOLAB via GUI

The GUI for EDOLAB is developed using MATLAB App Designer and can be accessed by executing either GUI.MLAPP or RunWithGUI.m from the root directory of EDOLAB. Note that the GUI is designed for MATLAB R2020b and is not backward compatible. Therefore, to use EDOLAB with the GUI, the user must have MATLAB R2020b or a newer version. Users with older MATLAB versions can still use EDOLAB by running RunWithoutGUI.m (see Section M-II.2). EDOLAB's GUI contains two modules—*Experimentation* and *Education*—which are explained below.

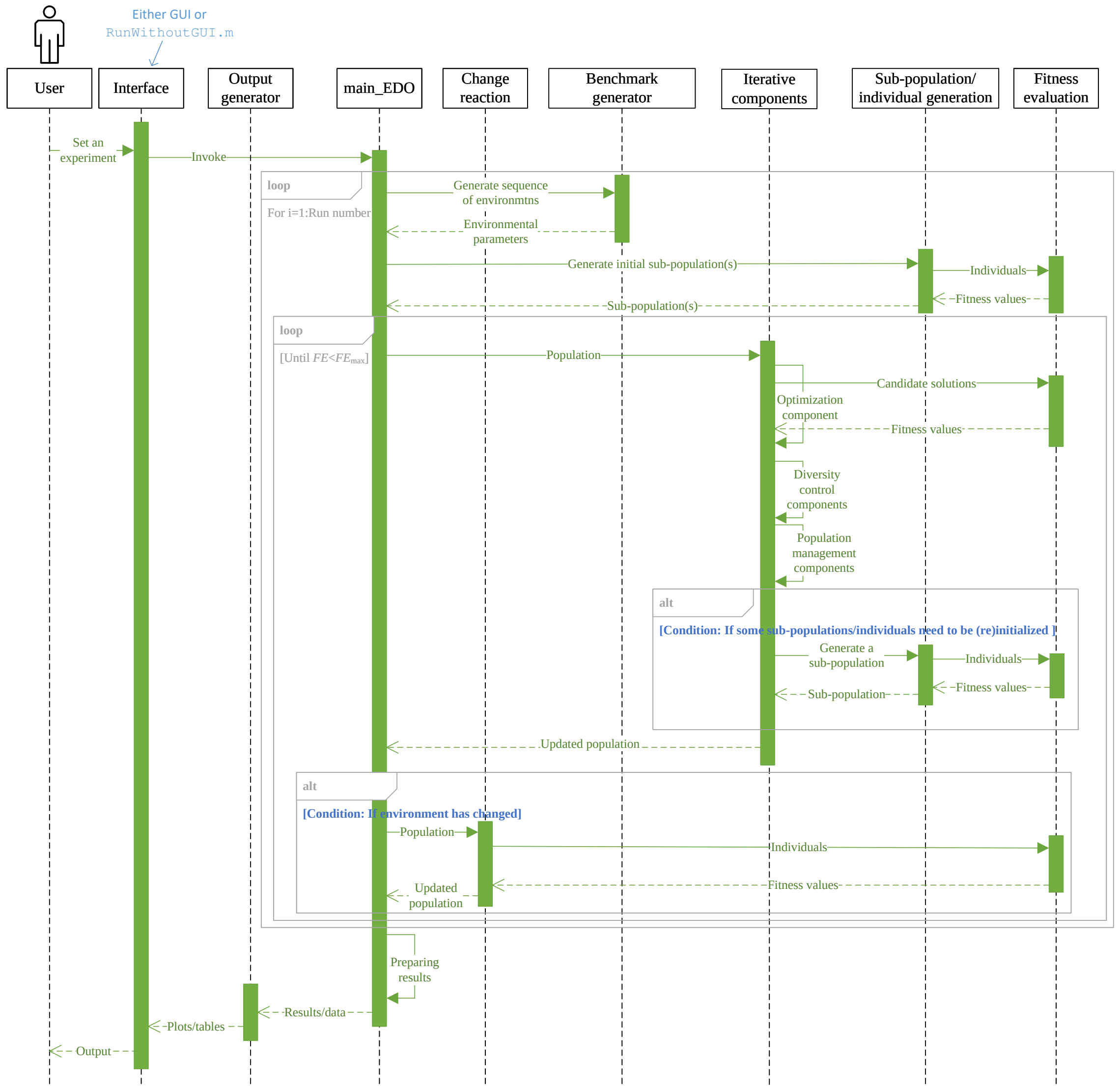
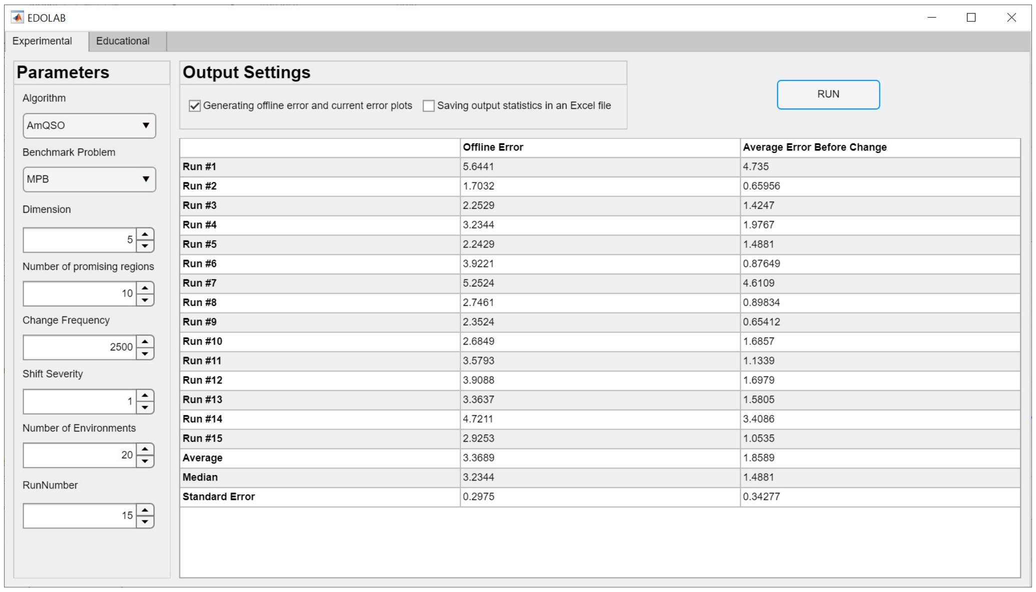


Fig. M-1. A general sequence diagram of running an EDOA in EDOLAB.

M-II.1.1 Experimentation module. The experimentation module is designed for conducting experiments. Figure M-2 shows the interface of this module, where users can select the algorithm (EDOA) and the benchmark generator. Additionally, users can configure the parameters of the benchmark generator to generate the desired problem instance. Note that EDOLAB's GUI does not provide an option to adjust the parameter settings of the EDOAs. This is because EDOAs typically have numerous parameters, which vary across algorithms depending on their structural components. Adding a feature to modify these parameters in the GUI would significantly increase complexity and make the interface harder to use and more confusing. Therefore, in EDOLAB, the parameters for each EDOA are preset based on the recommended values from their original references. Our evaluations show that these settings yield the best performance



Parameters

Algorithm: AmQSO

Benchmark Problem: MPB

Dimension: 5

Number of promising regions: 10

Change Frequency: 2500

Shift Severity: 1

Number of Environments: 20

RunNumber: 15

Output Settings

[✓] Generating offline error and current error plots ☐ Saving output statistics in an Excel file

	Offline Error	Average Error Before Change
Run #1	5.6441	4.735
Run #2	1.7032	0.65956
Run #3	2.2529	1.4247
Run #4	3.2344	1.9767
Run #5	2.2429	1.4881
Run #6	3.9221	0.87649
Run #7	5.2524	4.6109
Run #8	2.7461	0.89834
Run #9	2.3524	0.65412
Run #10	2.6849	1.6857
Run #11	3.5793	1.1339
Run #12	3.9088	1.6979
Run #13	3.3637	1.5805
Run #14	4.7211	3.4086
Run #15	2.9253	1.0535
Average	3.3689	1.8589
Median	3.2344	1.4881
Standard Error	0.2975	0.34277

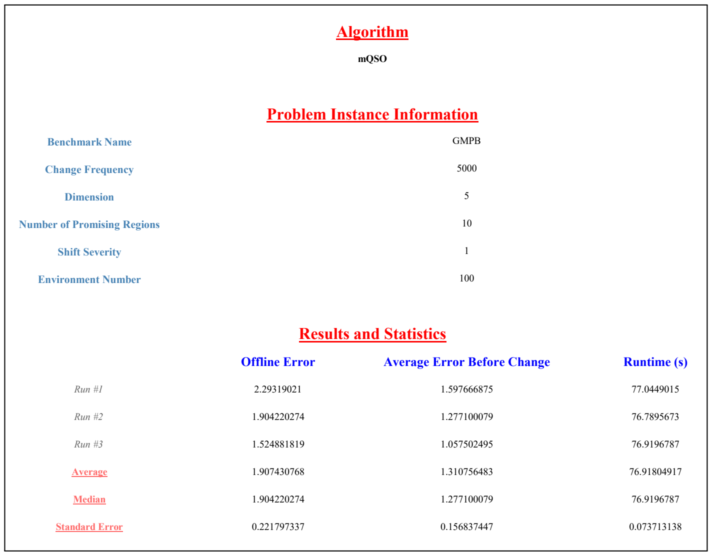
Fig. M-2. The experimentation module of EDOLAB.

for the EDOAs. For users interested in performing sensitivity analysis on EDOA parameters, adjustments can be made directly in the source code.

As illustrated in Figure M-2, users can configure the number of runs and several key benchmark parameters, such as dimension, number of promising regions, change frequency, shift severity, and the number of environments—these parameters are common between MPB, GDBG, GMPB, and FPs. The type and recommended values for these parameters are provided in Table M-I. In most studies, only the dimension, number of promising regions, change frequency, and shift severity are modified to generate different problem instances. Finally, users need to configure the "output settings." Using two checkboxes, they can choose whether to generate a figure with offline error and current error plots, and/or an Excel file containing the experiment results and statistics.

Once the experiment configuration is complete, the experiment can be started by pressing the RUN button in the top-right corner of the interface. The duration of the experiment depends on the complexity of the chosen EDOA and the configured problem instance, and it may take a significant amount of time to finish. It is worth noting that due to the complexity of EDOAs and dynamic benchmark generators, runs in this field generally take longer than those in other sub-fields of evolutionary computation, such as evolutionary static optimization or evolutionary multi-objective optimization with similar problem dimensionalities.

To track progress, EDOLAB displays the current run number and environment in the MATLAB Command Window. After the experiment is complete, the average, median, standard error values of the performance indicators, and runtime statistics are displayed in the Command Window. The detailed results of individual runs, along with their averages, medians, standard error values, runtime data, and the main benchmark parameters, are saved in an Excel file located in the `Results` folder, if the corresponding checkbox was selected. The Excel file name includes the EDOA, benchmark name, and the date and time of the experiment (e.g., `EDOA_Benchmark_DateTime.xlsx`). These results and statistics can be used for further statistical analysis using MATLAB or other software. An example of an Excel file generated by



Algorithm

mQSO

Problem Instance Information

Benchmark Name	GMPB
Change Frequency	5000
Dimension	5
Number of Promising Regions	10
Shift Severity	1
Environment Number	100

Results and Statistics

	Offline Error	Average Error Before Change	Runtime (s)
Run #1	2.29319021	1.597666875	77.0449015
Run #2	1.904220274	1.277100079	76.7895673
Run #3	1.524881819	1.057502495	76.9196787
Average	1.907430768	1.310756483	76.91804917
Median	1.904220274	1.277100079	76.9196787
Standard Error	0.221797337	0.156837447	0.073713138

Fig. M-3. An Excel output table generated by EDOLAB's `OutputExcel.m` function.

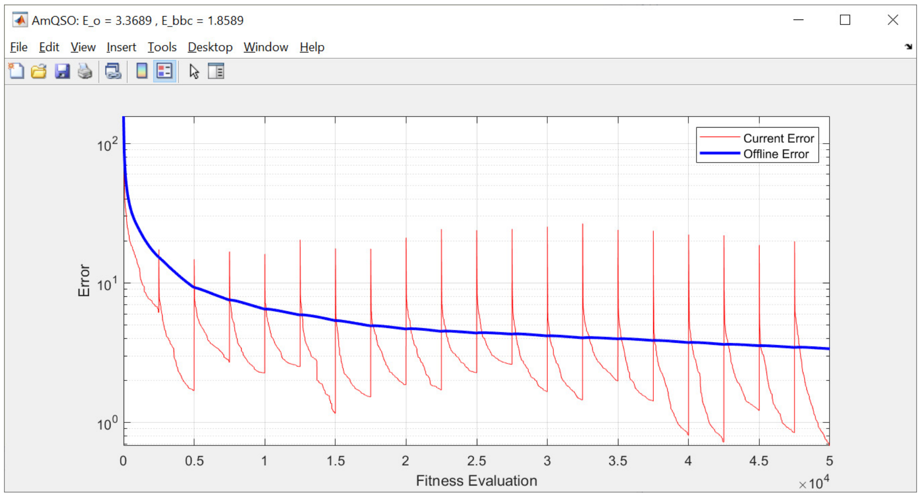


Fig. M-4. An output figure of an experimentation in EDOLAB. This figure depicts the plots of offline and current errors over time. The plots are the average of all runs.

EDOLAB is shown in Figure M-3. Additionally, if the user selected the relevant checkbox, a figure with plots of the offline and current errors over time is generated. An example of the output plots is provided in Figure M-4.

A Note on Parameter Settings of Benchmarks. The four benchmark generators included in EDOLAB share several common parameters, which have been widely manipulated in the literature to generate problem instances with varying levels of difficulty and characteristics. The GUI in EDOLAB facilitates the adjustment of these parameters, allowing users to easily configure benchmark scenarios. In Table M-I, we provide suggested values for these parameters, which

Table M-I. Types and suggested values for the main parameters of the benchmark generators in EDOLAB. The highlighted values represent the default settings for each parameter. When testing algorithms on specific parameters (e.g., different dimensions), the other parameters should be set to their default values to generate consistent problem instances.

Parameter	Name in the source code	Type	Suggested values
Dimension	`Problem.Dimension`	Positive integer	$\in \{2, 5, 10, 20\}^{\star}$
Number of promising regions	`Problem.PeakNumber`	Positive integer	$\in \{10, 25, 50, 100\}$
Change frequency	`Problem.ChangeFrequency`	Positive integer	$\in \{500, 1000, 2500, 5000\}$
Shift severity	`Problem.ShiftSeverity`	Non-negative real valued	$\in \{1, 2, 5\}$
Number of environments	`Problem.EnvironmentNumber`	Positive integer	$100^{\dagger}$

* These are suggested values for the experimentation module. In the education module, the dimension can only be set to two.

† For the sake of understandability, the number of environments is suggested to set between 10 and 20 in the education module.

can be used to generate standardized problem instances for comparing the performance of different algorithms. The key parameters that can be adjusted include:

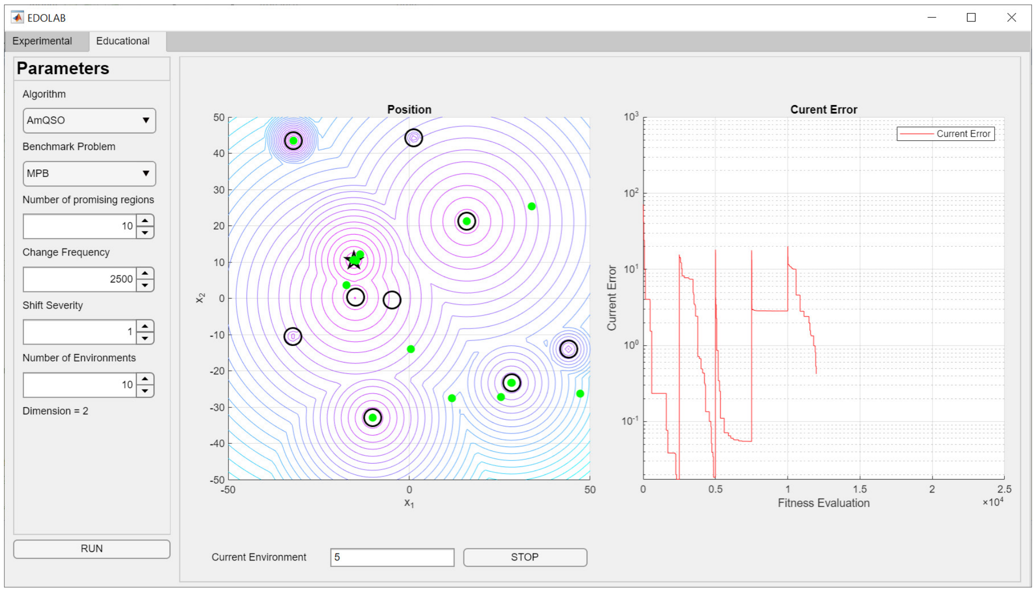
- Number of Promising Regions: Defines the number of promising regions in the search space.
- Shift Severity: Controls how significantly the search space changes between environments.
- Dimension: Sets the number of variables in the optimization problem.
- Change Frequency: Specifies how often the environment changes during the optimization process.

To create a well-rounded experimental setup, we recommend selecting one benchmark generator from FPs or MPB and one from GDBG or GMPB. This approach ensures a balance between simpler and more complex problem instances, allowing for a more comprehensive evaluation of algorithm performance. FPs and MPB represent benchmarks with fewer challenges, making them suitable for baseline comparisons, while GDBG and GMPB introduce more difficult problem instances with complex characteristics. This diversity helps to test the EDOAs' ability to adapt to varying levels of difficulty and complexity. For each chosen benchmark generator, apply the parameter settings provided in Table M-I. By using the different parameter settings provided in Table M-I, we can generate 12 distinct problem instances for each chosen benchmark generator.

Using the suggested parameter settings in Table M-I, researchers can generate diverse problem instances from the included benchmark generators, helping to establish a standardized experimental setup for algorithm comparison. These settings provide a common foundation for most studies in the field of evolutionary dynamic optimization. However, it is important to consider that specific studies may require different parameter settings, depending on the scope and focus of the research. For example, higher dimension values are used in research focused on large-scale dynamic optimization [Bai et al. 2022] Ultimately, while these suggestions aim to provide a consistent framework for comparison, they can be adapted to suit the requirements of targeted studies.

M-II.1.2 Education module. The education module allows users to visually observe the current environment, environmental changes, and the positions and behaviors of individuals over time. Figure M-5 displays the interface of EDOLAB's education module. On the left side of the interface, users can configure an experiment in a manner similar to the experimentation module. However, only 2-dimensional problem instances are supported in the education module, as the goal is to visualize the problem space and individuals.

Once the experiment is configured and the RUN button is pressed, the experiment begins, and the environmental parameters and positions of individuals over time are archived. The time required for the run will depend on the CPU,



Algorithm AmQSO

Benchmark Problem MPB

Number of promising regions 10

Change Frequency 2500

Shift Severity 1

Number of Environments 10

Dimension = 2

Current Environment 5

Fig. M-5. The education module of EDOLAB.

the selected EDOA, and the benchmark settings. After the run is complete, the archived information is displayed within the education module interface.

Using the archived data, the education module generates a video showing the environments and the positions of individuals over time. As depicted in Figure M-5, a 2-dimensional contour plot is used for this visualization. In the contour plot, the center of each visible promising region—those not covered by larger regions—is marked with a black circle, the global optimum position is indicated by a black pentagram, and the individuals are represented by green filled circles. The positions of individuals are updated every iteration, and the contour plot is refreshed after each environmental change. Additionally, the current error plot and the current environment number are shown to provide further insights, which enhance the understanding of the problem and the behavior of the EDOA.

By monitoring the positions of individuals, the search space/environment, the current environment number, and the current error plot over time, users can observe the EDOA's performance in exploration, exploitation, and tracking within each environment. Furthermore, the effectiveness of various EDOA components—such as mutual exclusion in promising regions [Blackwell and Branke 2006], the generation of new sub-populations [Blackwell et al. 2008], promising region coverage, mechanisms for increasing global diversity, and change reaction—can also be analyzed using the education module.

Unlike the experimentation module, where identical random streams are used across all experiments, the education module employs different random streams for each run. Consequently, users can observe the behavior and performance of the EDOA in different problem instances during each run of the education module.

M-II.2 Using EDOLAB without GUI

EDOLAB can also be operated without the GUI, which offers more advanced and flexible options for users. In this mode, users interact directly with the source codes of EDOLAB, which enables them to: (1) modify the parameter settings of the EDOAs, (2) alter or disable certain components of the EDOAs, and (3) adjust all parameters of the benchmarks. To enhance readability, understanding, and ease of navigation within EDOLAB's source code, we have:

- divided the code into *sections* using the `%%` command, with each section having a descriptive header. These sections group related lines of code, such as those implementing components (for example, exclusion [Blackwell and Branke 2006]), initializing EDOA parameters, or preparing output values,
- assigned meaningful and descriptive names to all structures, parameters, and functions in EDOLAB, and
- added informative comments throughout the code to assist users.

To run an EDOA without the GUI, users interact with the `RunWithoutGUI.m` file in the root directory of EDOLAB. Within this file, users can select the EDOA, choose the benchmark, and configure the main benchmark parameters (as shown in Table M-I). To specify an EDOA and benchmark, the user sets `AlgorithmName` to the desired EDOA (for example, `AlgorithmName = 'mQSO'`) and `BenchmarkName` to the desired benchmark (for example, `BenchmarkName = 'GMPB'`).

Users can also choose between the experimentation and education modules within `RunWithoutGUI.m`. Similar to the GUI's education module (see Figure M-5), selecting the education module in `RunWithoutGUI.m` will display contour plots of the environments, the positions of individuals, and the current error over time. The education module is activated when the user sets `VisualizationOverOptimization = 1`.

If `VisualizationOverOptimization = 0` is set, the experimentation module is activated. When using the experimentation module, users can configure the outputs. By setting `OutputFigure` to 1, users can generate visual plots of offline and current errors (see Figure M-4). Additionally, setting `GeneratingExcelFile = 1` will save an Excel file containing output statistics and results in the `Results` folder. These archived results in the Excel file can later be used for statistical analysis.

The parameters of the selected EDOA can also be modified in its main function (for example, $\texttt{main}_{\texttt{m}}\texttt{QSO.m}$), which is located in the EDOA's sub-folder. By default, these parameters are set to the values recommended in their original references. The lines of code for initializing EDOA parameters are found in the `%% Initializing Optimizer` section of the EDOA's main function. A structure named `Optimizer` contains all the parameters of the EDOA.

In addition to the main parameters of the benchmark generators listed in Table M-I, each benchmark has additional parameters. Typically, researchers modify only the main parameters to generate different problem instances. However, users wishing to evaluate EDOA performance on instances with specific characteristics can adjust other parameter values in the corresponding `BenchmarkGenerator_Benchmark.m` file. For example, Table M-II shows the parameters of GMPB that can be altered by the user in `BenchmarkGenerator_GMPB.m`, located in `EDOLAB\Benchmark\GMPB`.

Once the configurations are complete, the user can run `RunWithoutGUI.m` to initiate the experiment. During the run, progress information—including the current run number and environment number—is displayed in the MATLAB Command Window. Upon completion of the experiment, the results are also presented in the MATLAB Command Window.

M-III EXTENSION

Users can extend EDOLAB, as it is an open-source platform. Below, we describe how to add new benchmark generators, performance indicators, and EDOAs to EDOLAB.

Table M-II. Parameters of GMPB that can be changed by the user to generate problem instances with different morphological and dynamical characteristics.

Parameter	Name in the source code	Suggested value(s)
Dimension[†]	`Problem.Dimension`	$\in \{1, 2, 5, 10\}$
Numbers of promising regions[†]	`Problem.PeakNumber`	$\in \{10, 25, 50, 100\}$
Change frequency[†]	`Problem.ChangeFrequency`	$\in \{500, 1000, 2500, 5000\}$
Shift severity[†]	`Problem.ShiftSeverity`	$\in \{1, 2, 5\}$
Number of environments[†]	`Problem.EnvironmentNumber`	100
Height severity	`Problem.HeightSeverity`	7
Width severity	`Problem.WidthSeverity`	1
Irregularity parameter τ severity	`Problem.TauSeverity`	0.2
Irregularity parameter η severity	`Problem.EtaSeverity`	10
Angle severity	`Problem.AngleSeverity`	$\pi/9$
Search range upper bound	`Problem.MaxCoordinate`	50
Search range lower bound	`Problem.MinCoordinate`	-50
Maximum height	`Problem.MaxHeight`	70
Minimum height	`Problem.MinHeight`	30
Maximum width	`Problem.MaxWidth`	12
Minimum width	`Problem.MinWidth`	1
Maximum angle	`Problem.MaxAngle`	π
Minimum angle	`Problem.MinAngle`	$-\pi$
Maximum irregularity parameter τ	`Problem.MaxTau`	1
Minimum irregularity parameter τ	`Problem.MinTau`	0.1
Maximum irregularity parameter η	`Problem.MaxEta`	50
Minimum irregularity parameter η	`Problem.MinEta`	0

[†] These are commonly used parameters to generate different problem instances with various characteristics. As stated before, these parameters are common among the benchmark generators of EDOLAB and can be either set in the GUI or `RunWithoutGUI.m`.

M-III.1 Adding a benchmark generator

Suppose a user wants to add a new benchmark called ABC. First, the user must create a new sub-folder named ABC within the `Benchmark` folder. Then, two functions, `fitness_ABC.m` and `BenchmarkGenerator_ABC.m`, need to be added to this folder.

In `BenchmarkGenerator_ABC.m`, the user defines and initializes all the parameters of the new benchmark within a structure named `Problem`, similar to how the parameters of existing benchmark generators in EDOLAB are defined. Subsequently, the environmental parameters for all environments must be generated in this function, and all the environmental and control parameters of ABC must be stored in the `Problem` structure.

The second function, `fitness_ABC.m`, contains the code for the baseline function of ABC. Both `BenchmarkGenerator_ABC.m` and `fitness_ABC.m` must have inputs and outputs consistent with those of EDOLAB's current benchmarks. No changes are required in other functions, and ABC will automatically be added to the list of benchmarks in the GUI and can also be accessed via `RunWithoutGUI.m`.

M-III.2 Adding a performance indicator

Typically, the information required for calculating performance indicators in dynamic optimization problem (DOP) literature is gathered over time—either at the end of each environment [Trojanowski and Michalewicz 1999], after every function evaluation [Branke and Schmeck 2003], or when solutions are deployed in each environment [Yazdani 2018]. In EDOLAB, this data is collected in `fitness.m` and stored in the `Problem` structure.

To add a new performance indicator, the user first needs to modify `fitness.m` to collect the necessary data and store it in the `Problem` structure. The code for calculating the performance indicator should then be added to the `%% Performance indicator calculation` section in the main function of the EDOA (e.g., `main_mQSO.m`). Additionally, the results of the newly added performance indicator must be included in the outputs, which can be done in the `%% Output preparation` section at the bottom of the EDOA's main function.

M-III.3 Adding an EDOA

Adding a new EDOA to EDOLAB requires minimal modifications to the source code to ensure compatibility. Users should follow these steps:

- First, create a sub-folder inside the `Algorithm` folder, named according to the new EDOA. Then, add the EDOA's functions to this sub-folder.
- The new EDOA must be invoked by `RunWithoutGUI.m`. The user should ensure that the inputs and outputs of the EDOA's main function are compatible with `RunWithoutGUI.m`.
- In the main function of the new EDOA, call `BenchmarkGenerator.m` to generate the problem instance.
- To enable the education module, include the code that generates and collects information related to the education module in the main loop of the EDOA. This code can be found in the `%% Visualization for education module` section of other EDOAs.
- Use `fitness.m` for evaluating the fitness of solutions.
- Before initializing the optimizer in the main function of the EDOA, define parameters and data structures for gathering runtime, performance indicators, and other output information. After each run, ensure that the necessary information is stored in these parameters and arrays.
- Before initializing the optimizer in the main function of the EDOA, define parameters and data structures for gathering runtime, performance indicators, and other output information. After each run, ensure that the necessary information is stored in these parameters and arrays.
- At the end of the main function of the EDOA, include the code for output preparation similar to the structure in the existing algorithms.
- The main function of the newly added EDOA should be named `main_EDOA.m` to make it accessible through EDOLAB.

For example, if the new EDOA is called XYZ, the sub-folder should be named XYZ, and the main function file should be named `main_XYZ.m`. Once this is done, the new EDOA will automatically be added to the list of available algorithms in both the GUI modules. Additionally, by setting `AlgorithmName = 'XYZ`'`, the algorithm can be run using `RunWithoutGUI.m`.

M-IV USING EDOLAB IN OCTAVE

Since EDOLAB was originally developed in MATLAB, some users may prefer to use an open-source alternative to run the platform. Octave is a widely-used open-source software that is largely compatible with MATLAB, providing researchers with a free alternative to access EDOLAB's features without requiring a MATLAB license. With minor modifications, many of EDOLAB's functionalities can be used in Octave, though there are certain limitations. One major limitation is that the GUI functionality is not supported in Octave, as it relies on MATLAB's App Designer. Consequently, all experiments and tasks in Octave must be carried out through `RunWithoutGUI.m`. Below are guidelines on how to use EDOLAB with Octave and the necessary modifications to ensure compatibility.

M-IV.1 Notes on Using `RunWithoutGUI` in Octave

While Octave is largely compatible with MATLAB, there are a few important differences to keep in mind when running `RunWithoutGUI.m` in Octave.

- **Necessary packages**: The `statistics` and `io` packages must be loaded to run EDOLAB in Octave. These packages provide functions essential for statistical computations, distance calculation (`pdist2` function), and reading/writing files.
- **Generating Excel Files**:
  - The `xlswrite` function requires the `io` package, which is automatically loaded.
  - ActiveX is not supported in Octave, meaning Excel files cannot be opened automatically after they are generated.
  - The `actxserver` function, which MATLAB uses to control Excel, is unavailable in Octave.
  - Font color, font style, and other formatting options are unsupported, so Excel files generated in Octave will have a basic, unformatted style.
- **Generating Plots**:
  - Functions like append, `parfor`, and `blkdiag` (used within the append function) are not supported in Octave. These are needed for advanced plotting, so alternative methods may be required to replicate this functionality.
- **Calling and Writing Benchmark and Algorithm Names**:
  - Octave does not support the `""` syntax for strings. Instead, users must use cell arrays for strings within the `RunWithoutGUI.m` script.
- **Free Peaks (FPs) Benchmark**:
  - Octave cannot read `.mexw64` files generated by MATLAB from `.cpp` files. To resolve this, delete the existing `.mexw64` file in the KDTree folder and generate a new `.mex` file using the following command:

    ```
    mkoctfile -v --mex ConstructKDTree.cpp
    ```
  - Ensure that Octave is in the same directory as the `.cpp` file when running this command.
  - If errors occur, you may need to install the MinGW-w64 compiler.

M-IV.2 Octave Compatibility Folder

To simplify the process of using EDOLAB in Octave, we have created a folder named `Octave_compatibility`, which contains all the files that have been updated for compatibility with Octave. These modifications ensure that the core functionalities of EDOLAB work without issues in Octave. The key changes include:

- `RunWithoutGUI`: Adapted to work with Octave's syntax and configured to automatically load the necessary packages (`statistics` and `io`).
- `OutputExcel`: Simplified to function without ActiveX or advanced Excel formatting features.
- `OutputPlot`: Adjusted to account for limitations in Octave's plotting capabilities.

- `KDTree`: The generation of `.mex` files from `.cpp` files must be performed manually, as described earlier.

Users wishing to run EDOLAB in Octave can do so by replacing the corresponding files in the main EDOLAB directory with the modified files provided in the `Octave_compatibility` folder. Once the files are replaced, `RunWithoutGUI.m` can be executed in Octave to run experiments and tasks without further adjustments.

REFERENCES

Hui Bai, Ran Cheng, Danial Yazdani, Kay Chen Tan, and Yaochu Jin. 2022. Evolutionary large-scale dynamic optimization using bilevel variable grouping. *IEEE Transactions on Cybernetics* 53, 11 (2022), 6937–6950.

Tim Blackwell and Juergen Branke. 2006. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation* 10, 4 (2006), 459–472.

Tim Blackwell, Juergen Branke, and Xiaodong Li. 2008. Particle swarms for dynamic optimization problems. In *Swarm Intelligence: Introduction and Applications*, Christian Blum and Daniel Merkle (Eds.). Springer Lecture Notes in Computer Science, 193–217.

Juergen Branke and Hartmut Schmeck. 2003. Designing Evolutionary Algorithms for Dynamic Optimization Problems. In *Advances in Evolutionary Computing*, A. Ghosh and S. Tsutsui (Eds.). Springer Natural Computing Series, 239–262.

Krzysztof Trojanowski and Zbigniew Michalewicz. 1999. Searching for optima in non-stationary environments. In *Congress on Evolutionary Computation*, Vol. 3. 1843–1850.

Danial Yazdani. 2018. *Particle swarm optimization for dynamically changing environments with particular focus on scalability and switching cost*. Ph. D. Dissertation. Liverpool John Moores University, Liverpool, UK.

Danial Yazdani, Ran Cheng, Cheng He, and Juergen Branke. 2020. Adaptive control of subpopulations in evolutionary dynamic optimization. *IEEE Transactions on Cybernetics* 52, 7 (2020), 6476–6489.

Danial Yazdani, Ran Cheng, Donya Yazdani, Jürgen Branke, Yaochu Jin, and Xin Yao. 2021a. A Survey of Evolutionary Continuous Dynamic Optimization Over Two Decades – Part A. *IEEE Transactions on Evolutionary Computation* 25, 4 (2021), 609–629.

Danial Yazdani, Ran Cheng, Donya Yazdani, Jürgen Branke, Yaochu Jin, and Xin Yao. 2021b. A Survey of Evolutionary Continuous Dynamic Optimization Over Two Decades – Part B. *IEEE Transactions on Evolutionary Computation* 25, 4 (2021), 630–650.