PyDDRBG: A Python framework for benchmarking and evaluating static and dynamic multimodal optimization methods

Ali Ahrari, Saber Elsayed, Ruhul Sarker, Daryl Essam

School of Engineering and Information Technology, University of New South Wales, Canberra, ACT, Australia (E-mail: a.ahrari@unsw.edu.au, ailiahrari1983@gmail.com; 
{s.elsayed; r.sarker; d.essam}@adfa.edu.au)

Carlos A. Coello Coello

CINVESTAV-IPN, Departamento de Computaci´on, Mexico City, Mexico; Basque Center for Applied Mathematics (BCAM) & Ikerbasque, Spain (e-mail: ccoello@cs.cinvestav.mx)

Abstract

PyDDRBG is a Python framework for generating tunable test problems for static and dynamic multimodal optimization. It allows for quick and simple generation of a set of predefined problems for non-experienced users, as well as highly customized problems for more experienced users. It easily integrates with an arbitrary optimization method. It can calculate the optimization performance when measured according to the robust mean peak ratio. PyDDRBG is expected to advance the fields of static and dynamic multimodal optimization by providing a common platform to facilitate the numerical analysis, evaluation, and comparison in these fields.

Keywords: Test problems, Benchmarking, Niching, Performance indicator
### 1. Motivation and Significance

Many real-world problems are dynamic, which means that some aspects of the problem change over time. Finding the optimal solutions to such problems requires dynamic optimization. In the most recent decade, there has been a lot of research work on formulating dynamic problems \[1,2\] and developing evolutionary algorithms and swarm intelligence methods for dynamic optimization \[3,4\], some of which have been applied to real-world problems such as optimal control \[5\] and vehicle routing \[6\].

A successful dynamic optimization method should be able to track all good local minima since the change in the depth of the minima may result in a substantial change in the location of the global minimum \[7\]. This means that even though the problem asks for one global optimum at each time, optimization methods should employ a multimodal optimization \[8\] strategy to detect and track distinct global and local minima. Consequently, this field of research is known as dynamic multimodal optimization (DMMO) \[9\].

There are a few benchmark generator for DMMO. The most well-known and widely used one is the Moving Peak Benchmark (MPB) \[7,9\] and its variants and extensions \[10,11,12,13\]. The Real Rotation Dynamic Benchmark Generator (RRDBG) \[14\] applies rotation to change the location for the optima. Real Composition Dynamic Benchmark Generator (RCDBG) \[14\]
allows for employing more complex functions instead of a unimodal spherical function to form the fitness landscape around each optimum.

A more recent perspective to DMMO asks for multiple (near-) global optimum at the problem level at each time [15, 16, 17]. The aforementioned benchmark generators can be extended to this type of problems with some minor modification, e.g., the modified MPB in [16] which can simulate an arbitrary number of global optima.

A comprehensive benchmark generator for DMMO should reflect the diverse challenges associated with DMMO. In particular, these challenges can be divided into three groups [18]:

- *global optimization*, which determines how hard it is to find each global optimum. Intervening factors are the depth of local optima, correlation among variables, and badly scaled problems [19].

- *multimodal optimization*, which determines how difficult it is to detect distinct global optima. The factors involved are the irregularity in the distribution of global optima, their shapes and sizes

- *dynamic optimization*, which is affected by the change frequency, change severity, and irregularities in the pattern of the change in the fitness landscape.

Such classification measures the difficulty of DMMO from three distinct perspectives, each associated with a distinguishable type of challenges. This allows for analyzing the pros and cons of each DMMO method more reliably. The recently proposed Dynamic Distortion and Rotation Benchmark (DDRB) generator [18] allows for simulation and control of these challenges. When compared with exiting benchmark generators for DMMO, it can simulate more diverse features or at least has the advantages of easy integration with existing static multimodal optimization problems and deterministic and thus platform independent nature (see [18] for in-depth analysis and comparison with other benchmark generators for DMMO).

It is possible to analyze the characteristics of landscape using landscape metrics such as those defined and used in [20, 21] for deceptiveness of local optima and in [22] for irregularity of distribution and sizes of global optima; however, user-defined parameters that control the difficulty of each aspect of the problem eliminates the need for performing such analysis for each generated problems. This is the case with DDRB, in which:

- the parameter \( h_{GO} \) controls the difficulty of reaching each global optimum (global optimization difficulty) either by increasing the depth of local optima or by increasing the condition number of the landscape.
the parameter $e_c$ controls the difficulty of multimodal optimization by
determining the distortion of the landscape during dynamic changes
(see [18] for details), which is the extent of the changes in the shapes,
sizes, and irregularity in the distribution of global minima.

- three other parameters control dynamic aspects of the problem, which
are the severity of the change (one parameter), change frequency (one
parameter), and randomness in the pattern of the change (one parameter).
The former two are shared features of most dynamic test problems
(e.g., see [23]).

This study provides a user-friendly Python framework for the DDRB gen-
erator [18]. This framework, called PyDDRBG, also provides a method that
calculates the performance of a static or dynamic multimodal optimization
method, which will be explained in Subsection 3.5. This framework has been
structured such that:

- It is easy to use and understand by non-expert users, allowing them to
  simulate the exact problems employed in [18].
- It allows for detailed control and customization for more experienced
  users who wish to create different benchmark problems.
- It can be easily and simply integrated with any arbitrary optimization
  method.

2. Software Description

The problems generated by PyDDRBG can be static (no change in the
problem landscape) or dynamic (the problem landscape changes at predefined
intervals). For static problems, a promising optimization method should be
able to correctly identify as many global minima as possible, while in dynamic
problems, it should also be able to track these minima over time. In PyD-
DRBG, a DMMO problem is formed by first generating a static multimodal
optimization (SMMO) problem and then simulating dynamic behavior by
distortion and rigid rotation of the fitness landscape at predefined intervals.

PyDDRBG has five predefined static parametric composite multimodal
optimization functions ($G_i, i = 11, 12, \ldots, 15$), each formulated by a com-
bination of three basic functions. Each composite function has a tunable
parameter denoted by $D_I$, which controls the number of global minima
(numGlobMin) in the problem. For each composite function, two values
of $D_I$ have been considered by default, resulting in 10 standard problems
(PID = 1, 2, \ldots, 10). Table 2 presents the properties of these problems.
The user can further control the properties of the static problem by changing the corresponding attributes of `statAttr`, including:

- `+dim: int`, which defines the search space dimensionality. See Table 2 for allowable values of `dim` for each PID,

- `+h_GO: float (0 ≤ h_GO ≤ 1)`, which controls the difficulty of global optimization in the problem. This parameter does not change the location of the global minima but makes finding them harder by increasing the depths of undesirable local minima or increasing the condition number of the basic functions that form the parametric composite function (see [18] for more details.)

- `+maxEvalCoeff: float (maxEvalCoeff > 0)`, which controls the evaluation budget of the static problem or the zeroth time step if the problem is dynamic. This budget is `maxEvalCoeff × dim × numGlobMin`.

- `+rotAngle: float (rotAngle ∈ ℝ)`, which defines the rotation angle for the rigid rotation of the search space (static problem).

Dynamic problems are formed by simulating a dynamic behavior to the defined static problems, which results in some change in the problem landscape after a certain time. This time is measured in terms of the used function evaluations, i.e., the number of calls to the objective function. PyDDRBG allows for a lower level control of dynamic properties of the benchmark problem, including the following attributes of `dynaAttr`:

- `+chSevReg: float (chSevReg > 0)`, which defines the severity of the regular (patterned) change. A greater value means a less severe change.

- `+chSevIrreg: float (chSevIrreg > 0)`, which defines the severity of irregular (patternless) change. A greater value means a less severe change.

- `+chFrCoeff: float (chFrCoeff > 0)`, which controls the change frequency for the first time step onward (the change frequency is `chFrCoeff × dim × numGlobMin`). The change frequency is the duration of the interval in which the problem remains unchanged. Each interval is called a time step (`timeStep = 0, 1, …, numTimeStep − 1`).

- `+numTimeStep: int (numTimeStep ≥ 0)`, which is the number of time steps in the dynamic problem (one if the problem is static).

- `+e_c: float (e_c > 0)`, which is the eccentricity for the scaling function to control the intensity of dynamic distortion in the landscape. A greater value means a less severe distortion in the problem landscape.
Table 2: Specifications of the predefined static problems in PyDDRBG. For all functions, the search range is \([-5, 5]^D\), and \(k \in \mathbb{Z}_{\geq 0}\)

<table>
<thead>
<tr>
<th>PID</th>
<th>Function</th>
<th>statAttr.D</th>
<th>numGlobMin</th>
<th>Valid dimensionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(G_{11})</td>
<td>2</td>
<td>4</td>
<td>(2k + 2)</td>
</tr>
<tr>
<td>2</td>
<td>(G_{12})</td>
<td>2</td>
<td>2</td>
<td>(2k + 2)</td>
</tr>
<tr>
<td>3</td>
<td>(G_{13})</td>
<td>2</td>
<td>3</td>
<td>(2k + 2)</td>
</tr>
<tr>
<td>4</td>
<td>(G_{14})</td>
<td>1</td>
<td>3</td>
<td>(k + 1)</td>
</tr>
<tr>
<td>5</td>
<td>(G_{15})</td>
<td>2</td>
<td>4</td>
<td>(2k + 2)</td>
</tr>
<tr>
<td>6</td>
<td>(G_{11})</td>
<td>4</td>
<td>16</td>
<td>(2k + 4)</td>
</tr>
<tr>
<td>7</td>
<td>(G_{12})</td>
<td>6</td>
<td>8</td>
<td>(2k + 6)</td>
</tr>
<tr>
<td>8</td>
<td>(G_{13})</td>
<td>4</td>
<td>9</td>
<td>(2k + 4)</td>
</tr>
<tr>
<td>9</td>
<td>(G_{14})</td>
<td>2</td>
<td>18</td>
<td>(k + 2)</td>
</tr>
<tr>
<td>10</td>
<td>(G_{15})</td>
<td>4</td>
<td>16</td>
<td>(2k + 4)</td>
</tr>
</tbody>
</table>

- `+performDynaRot: bool` (False/True), which determines if dynamic rotation should be performed.

For multimodal optimization, the peak ratio (PR) \([24]\) is the most commonly adopted indicator for performance evaluation. It simply calculates the fraction of global minima that has been detected given a predefined target tolerance for the solution value and a niche radius. The robust peak ratio (RPR), as introduced in \([18]\), provides a more robust indicator which eliminates the need to specify the niche radius for performance evaluation. For each detected global minimum, a partial score in \([0, 1]\) is calculated with respect to the predefined loosest and tightest tolerances. RPR is then calculated as the average of these partial scores.

2.1. Software Functionalities

The use cases of the PyDDRBG framework are as follows:

- A simple way to set all problem properties to predefined values is by choosing `PID` (see Table 2) and `dynaScn` \([3]\).

- Customization of problem properties to create diverse test problems. This customization is optional and might be preferable for more experienced users. The problem properties that can be customized are attributes of `statAttr` and `dynaAttr`.

- Calculation of the problem data required for a solution evaluation and performance evaluation.
Table 3: Dynamic scenarios predefined for the PyDDRBG. In all scenarios, statAttr.dim=10, statAttr.maxEvalCoeff=4000, statAttr.rotAngle=π/6, dynaAttr.numTimeStep=40, and dynaAttr.performDynaRot=True. The user may customize these properties to generate new problems.

<table>
<thead>
<tr>
<th>dynaScn</th>
<th>isDynamic</th>
<th>statAttr.h.GO</th>
<th>dynaAttr.chSevReg</th>
<th>statAttr.chSevIrreg</th>
<th>statAttr.chFrCoeff</th>
<th>dynaAttr.chFrCoeff</th>
<th>dynaAttr.numTimeStep</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>False</td>
<td>0.3</td>
<td>0.5</td>
<td>30</td>
<td>∞</td>
<td>2000</td>
<td>1</td>
<td>Base Scenario (Static)</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>0.3</td>
<td>0.5</td>
<td>30</td>
<td>∞</td>
<td>2000</td>
<td>40</td>
<td>Base Scenario (Dynamic)</td>
</tr>
<tr>
<td>2</td>
<td>True</td>
<td>0.6</td>
<td>0.5</td>
<td>30</td>
<td>∞</td>
<td>2000</td>
<td>40</td>
<td>Hard global optimization</td>
</tr>
<tr>
<td>3</td>
<td>True</td>
<td>0.3</td>
<td>0.1</td>
<td>30</td>
<td>∞</td>
<td>2000</td>
<td>40</td>
<td>Hard niching problem</td>
</tr>
<tr>
<td>4</td>
<td>True</td>
<td>0.3</td>
<td>0.5</td>
<td>10</td>
<td>∞</td>
<td>2000</td>
<td>40</td>
<td>Severe changes</td>
</tr>
<tr>
<td>5</td>
<td>True</td>
<td>0.3</td>
<td>0.5</td>
<td>30</td>
<td>5</td>
<td>2000</td>
<td>40</td>
<td>Irregular changes</td>
</tr>
<tr>
<td>6</td>
<td>True</td>
<td>0.3</td>
<td>0.5</td>
<td>30</td>
<td>∞</td>
<td>500</td>
<td>40</td>
<td>Fast-changing problem</td>
</tr>
</tbody>
</table>

- Integration with an arbitrary optimization method. problem has all the required data and methods for optimization.
- Static and dynamic multimodal optimization with an exemplary method, which has been provided to demonstrate how to integrate an arbitrary optimization method with a test problem generated by PyDDRBG.
- A static method to calculate robust mean peak ratio for static and dynamic problems given the tightest and loosest tolerances on the objective function.

2.2. Software Architecture

Figure 1 depicts the class diagram of PyDDRBG in the Unified Modeling Language (UML). It also illustrates how it interacts with an external static or dynamic multimodal optimization method. DDRB.py is the main file of PyDDRBG which enables the user to generate a static or dynamic multimodal optimization problem and evaluate a solution. Methods and attributes of this class are presented in Table 4. Notably, statAttr and dynaAttr include all control parameters of the problem that can be customized by the user, and the boolean attribute isDynamic determines if the problem is dynamic.
3. Implementation Steps

The steps to generate and employ a test problem from PyDDRBG are explained in this section.

3.1. Create the problem object

First, the user should create the problem object by creating an object from the DDRB class:

\[
\text{problem}=\text{DDRB}(\text{PID},\text{dynaScn}).
\]  

(1)

This sets the control parameters of the problem according to the static problem ID (see Table 2) and the selected dynamic scenario (see Table 3).

3.2. Customize the problem properties

This step is optional and can be useful for an experienced user, who can change properties of the problem by changing the corresponding attributes in \textit{statAttr} and \textit{dynaAttr}. For example, the hardness of the problem from the global optimization perspective can be intensified by increasing \( h_{\text{GO}} \):

\[
\text{problem.statAttr.h}_{\text{GO}} = 1.0.
\]  

(2)
<table>
<thead>
<tr>
<th>DDRB</th>
<th>Role of the attribute/method</th>
</tr>
</thead>
<tbody>
<tr>
<td>+statAttr: StaticAttribute</td>
<td>Determines static properties of the problem</td>
</tr>
<tr>
<td>+dynaAttr: DynamicAttribute</td>
<td>Determines dynamic properties of the problem</td>
</tr>
<tr>
<td>+statData: StaticData</td>
<td>Calculates and stores the data related to static aspects of the problem</td>
</tr>
<tr>
<td>+dynaData: DynamicData</td>
<td>Calculates and stores the data related to dynamic aspects of the problem</td>
</tr>
<tr>
<td>+maxEvalTotal: int</td>
<td>Defines the evaluation budget of the problem</td>
</tr>
<tr>
<td>+isDynamic: bool</td>
<td>Determines if the problem is dynamic</td>
</tr>
<tr>
<td>+numCallObj: int</td>
<td>Tracks the number of calls to the objective function</td>
</tr>
<tr>
<td>+DDRB(int PID, int dynaScn)</td>
<td>Constructs the problem object and sets statAttr and dynaAttr</td>
</tr>
<tr>
<td>+calc_problem_data()</td>
<td>Calculates problem data and stores them in statData and dynaData</td>
</tr>
<tr>
<td>+func_eval(float[] x): float</td>
<td>Calculates and returns the value of solution x</td>
</tr>
<tr>
<td>+func_eval_static(float[] x, StaticAttribute statAttr, DynamicAttribute dynaAttr): float</td>
<td>Calculates and returns the value of solution x excluding the effect of dynamic distortion and rotation</td>
</tr>
</tbody>
</table>

Table 4: Methods and attributes of the DDRB class
3.3. Generate the problem data

After customization of the problem parameters, problem data should be calculated and stored in the problem object as follows:

\[
\text{problem.calc\_problem\_data().}
\]  

(3)

\text{problem} has all the data required for a solution evaluation, which are stored in\text{statData} (data related to static aspects of the problem) and\text{dynaData} (data related to dynamic aspects of the problem). No further change to\text{problem} is allowed.

3.4. Solution evaluation

Given\text{problem}, an arbitrary solution\text{x} can be easily calculated for both static and dynamic problems using the following method:

\[
y=\text{problem.func\_eval(x)}.
\]  

(4)

in which\text{y} is the value of solution\text{x}. It is worth indicating that\text{problem} keeps track of the number of calls to the objective function in \text{problem.numCallObj}. Besides,\text{problem} has all the information and methods required for optimization. Therefore, for integration with an optimization method, it is sufficient to provide\text{problem} for that method. Integration with a sample optimization method will be demonstrated in Section 4.

3.5. Performance evaluation

Although there are many performance indicators for multimodal optimization [25], Peak Ratio (PR) is the most widely accepted one which has also been employed in competitions on niching methods for multimodal optimization [24]. PyDDRBG can calculate Robust Peak Ratio (RPR) [18]. When compared with PR, RPR is less sensitive to predefined function tolerance as it can assign partial credits for a solution if its value is between the predefined loosest and tightest tolerances. It also eliminates the sensitivity of PR and its need for the preset niche radius. Given the results of static/dynamic multimodal optimization, RPR is calculated using the following static method:

\[
\text{RPR, valDiff=PerformIndic.calc\_RPR(X, foundEval, tolFunScore, problem)},
\]  

(5)

in which:

-\text{X} (2D float array) is the set of near-optimal solutions reported by the optimization method.
• foundeEval (1D int array) stores the function evaluations at which these solutions were found.

• tolFunScore (1D float array) is the loosest and tightest tolerance for calculation of RPR (defined by the experimental setup).

• $0 \leq \text{RPR} \leq 1$ is the calculated RPR. If the problem is static, RPR is a scalar. If the problem is dynamic, RPR is a 1D array showing RPR at each time step.

• valDiff $\geq 0$: For a static problem, it is a 1D array showing the difference between the global minimum value and the value of the reported approximate solution for that global minimum. If there is no approximate solution for a global minimum, the corresponding element of valDiff will be $\infty$. For dynamic problems, valDiff is a 2D array showing these differences for each time step.

For DMMO problems, this method returns the calculated RPR at the end of each time step in the form of 1D array. The average of these values is the mean RPR, which is regarded as the performance measure.

4. Illustrative Example

The file example_optim.py provides an illustrative example for generation of a test problem, optimizing it and evaluating the optimization results. The main steps performed for this purpose are as follows:

• Create the problem object with predefined properties by specifying PID and DynaScn:

  $$\text{PID}=1$$
  $$\text{dynaScn}=6$$
  $$\text{problem}=\text{DDRB(PID,dynaScn)}$$

• Change the values of attributes of problem.statAttr and/or problem.dynaAttr if desired. This step is optional and can be useful for experienced users. As an example, we change the problem’s dimensionality and the number of time steps:

  $$\text{problem.statAttr.dim}=8$$
  $$\text{problem.dynaAttr.numTimeStep}=10$$

• Calculate the problem data and store them in problem:

  $$\text{problem.calc_problem_data()}$$
• Optimize the problem using an external optimization method and get the reported optimal solutions and the time (number of evaluations) at which these solutions have been found. A simple optimization method is provided in the file `example_optim.py` for demonstration, which is called as follows:

\[
\text{foundEval, solution} = \text{optimize_full}(\text{problem})
\]

• Calculate the performance after defining the loosest and tightest tolerance for the objective value:

\[
\text{tolFunScore} = \text{np.array}([0.1, 1e-5])
\]

\[
\text{RPR, valDiff} = \text{PerformIndicator.calc_RPR} (\text{solution}, \text{foundEval}, \text{tolFunScore}, \text{problem})
\]

5. Impact

SMMO is already a well-developed field of research. The importance of SMMO in real-world problems is already well-understood. The remarkable number of studies on SMMO [26, 8] and competitions on niching methods for SMMO, which have regularly been held at the Genetic and Evolutionary Computation Conference (GECCO) and at the IEEE Congress on Evolutionary Computation (CEC), is evidence for this claim. At the same time, a number of test suites have been proposed for performance evaluation and comparison of SMMO methods (see [22] for an example). In particular, the CEC’2013 test suite for static multimodal optimization [24] has served as a widely accepted tool for comparing SMMO methods since 2013, which has provided a substantial contribution to advancing the knowledge in this field.

DMMO, when multiple global minima should be tracked over time, is a relatively new field of research with application to some real-world problems. One familiar example is the problem of finding the fastest route to a destination by GPS. This problem demands multimodal optimization since the driver might be interested in multiple routes with similar estimated time of arrival (ETA) or even routes which might be slightly longer but may be preferable because of the familiarity for the driver with the road, safety, average speed, and so on. At the same time, this problem is dynamic since the optimal routes may change because of changes in traffic conditions, accidents, or even a missed turn by the driver. In these situations, it is desirable that the route finding algorithm updates the optimal routes as fast as possible. Other real-world exemplary applications are finding solutions to a system of nonlinear time-dependent equations [27] and tracking multiple moving targets [17].
Existing studies on DMMO (e.g. [16]) generally employ simple benchmark generators that may not be able to simulate all the challenges associated with DMMO. PyDDRBG provides a comprehensive test suite for both static and dynamic multimodal optimization. It is expected to become a widely adopted test suite for both static and dynamic multimodal optimization in the future. Ease of implementation, possibility for customization, deterministic nature of the problems, and lower-level control over the properties of the generated problems are good reasons to support this expectation.

6. Conclusions

Dynamic multimodal optimization (DMMO) is an emerging field of research with some practical applications. The developed python framework in this work provides an easy tool for benchmarking, analyzing and comparing arbitrary methods for both static multimodal optimization (SMMO) and DMMO. The ease of integration with optimization methods and the deterministic nature of the generated test problems should encourage researchers in the field of multimodal optimization (both dynamic and static) to employ this benchmark generator in their research. The parametric nature of these test problems allows the user to control the difficulty of different features of each problem to facilitate identification of the pros and cons of each method, which will illuminate the path to advancing knowledge in this field.

7. Conflict of Interest

No conflict of interest exists: We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgment

This work was supported by Australian Research Council (Discovery Project DP190102637). The last author acknowledges support from CONACyT project no. 2016-01-1920 (Investigación en Fronteras de la Ciencia 2016) and from a 2018 SEP-Cinvestav grant (application no. 4). He was also partially supported by the Basque Government through the BERC 2018-2021 program by the Spanish Ministry of Science.
References


