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, aliahrari1983@gmail.com; {s.elsayed; r.sarker; d.essam}@adfa.edu.au)

Carlos A. Coello Coello

CINVESTAV-IPN, Departamento de Computación, 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

Preprint submitted to SoftwareX *March 15, 2022*

Required Metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.0
C2	Permanent link to code/repository used for this code version	None
C3	Code Ocean compute capsule	None
C4	Legal Code License	MIT
C5	Code versioning system used	None
C6	Software code languages, tools, and services used	Python
C7	Compilation requirements, operating environments & dependencies	Python 3.7.9
C8	If available Link to developer documentation/manual	N/A
C9	Support email for questions	aliahrari1983@gmail.com

Table 1: Code metadata (mandatory)

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 generator [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 PyDDRBG, 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 combination 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 ($\mathsf{PID} = 1, 2, \ldots, 10$). 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 ($0 \leq$ 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 $\times$ dim $\times$ numGlobMin.
- +rotAngle: float (rotAngle $\in \mathbb{R}$), 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 (chFrCoef > 0), which controls the change frequency for the first time step onward (the change frequency is chFrCoeff $\times$dim$\times$ numGlobMin). The change frequency is the duration of the interval in which the problem remains unchanged. Each interval is called a time step (timeStep $= 0, 1, \ldots,$ 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.

PID	Function	statAttr.D_I	numGlobMin	Valid dimensionality
1	G_{11}	2	4	$2k+2$
2	G_{12}	2	2	$2k+2$
3	G_{13}	2	3	$2k+2$
4	G_{14}	1	3	$k+1$
5	G_{15}	2	4	$2k+2$
6	G_{11}	4	16	$2k+4$
7	G_{12}	6	8	$2k+6$
8	G_{13}	4	9	$2k+4$
9	G_{14}	2	18	$k+2$
10	G_{15}	4	16	$2k+4$

Table 2: Specifications of the predefined static problems in PyDDRBG. For all functions, the search range is $[-5, 5]^D$, and $k \in \mathbb{Z}_{\geq 0}$)

- +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.

dynaScn	isDynamic	statAttr.h_GO	dynaAttr.e_c	dynaAttr.chSevReg	dynaAttr.chSevIrreg	dynaAttr.chFrCoeff	dynaAttr.numTimeStep	Feature
0	False	0.3	0.5	30	∞	2000	1	Base Scenario (Static)
1	True	0.3	0.5	30	∞	2000	40	Base Scenario (Dynamic)
2	True	0.6	0.5	30	∞	2000	40	Hard global optimization
3	True	0.3	0.1	30	∞	2000	40	Hard niching problem
4	True	0.3	0.5	10	∞	2000	40	Severe changes
5	True	0.3	0.5	30	5	2000	40	Irregular changes
6	True	0.3	0.5	30	∞	500	40	Fast-changing problem

Table 3: Dynamic scenarios predefined for the PyDDRBG. In all scenarios, statAttr.dim=10, statAttr.maxEvalCoeff=4000, statAttr.rotAngle=$\pi/6$, dynaAttr.numTimeStep=40, and dynaAttr.performDynaRot=True. The user may customize these properties to generate new problems.

- 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.

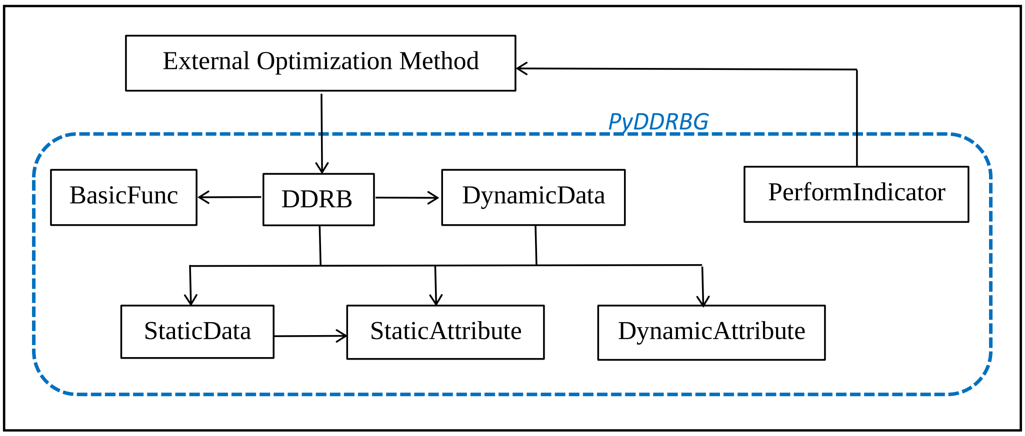

Figure 1: UML class diagram of PyDDRBG and its interaction with an external static or dynamic multimodal optimization method

A set of predefined static problems and dynamic scenarios have already been configured in PyDDRBG. These scenarios are presented in Table 3. The user can start from one of these predefined scenarios and then customize some parameters of the problem by changing the corresponding attribute in statAttr or dynaAttr.

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:

$$\mathsf{problem}{=}\mathsf{DDRB}(\mathsf{PID},\mathsf{dynaScn}). \tag{1}$$

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 statAttr and dynaAttr. For example, the hardness of the problem from the global optimization perspective can be intensified by increasing h_GO:

$$\mathsf{problem.statAttr.h\_GO} = 1.0. \tag{2}$$

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

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:

$$\mathsf{problem.calc\_problem\_data()}. \tag{3}$$

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

3.4. Solution evaluation

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

$$\mathsf{y{=}problem.func\_eval(x)}, \tag{4}$$

in which y is the value of solution x. It is worth indicating that problem keeps track of the number of calls to the objective function in problem.numCallObj. Besides, problem has all the information and methods required for optimization. Therefore, for integration with an optimization method, it is sufficient to provide 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:

$$\mathsf{RPR,\ valDiff{=}PerformIndic.calc\_RPR(X,foundEval,tolFunScore,problem)}, \tag{5}$$

in which:

- 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).
- $0 \leq$ RPR ≤ 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 ∞. 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:

```
PID=1
dynaScn=6
problem=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:

```
problem.statAttr.dim=8
problem.dynaAttr.numTimeStep=10
```

- Calculate the problem data and store them in problem:

```
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:

```
foundEval, solution=optimize_full(problem)
```

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

```
tolFunScore=np.array([0.1, 1e-5])
RPR,valDiff=PerformIndicator.calc_RPR(solution,foundEval,tolFunScore,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

[1] K. Deb, S. Karthik, et al., Dynamic multi-objective optimization and decision-making using modified nsga-ii: a case study on hydro-thermal power scheduling, in: International conference on evolutionary multi-criterion optimization, Springer, 2007, pp. 803–817.

[2] A. Ahrari, S. Elsayed, R. Sarker, D. Essam, C. A. C. Coello, Towards a more practically sound formulation of dynamic problems and performance evaluation of dynamic search methods, in: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2020, pp. 1387–1394.

[3] T. T. Nguyen, S. Yang, J. Branke, Evolutionary dynamic optimization: A survey of the state of the art, Swarm and Evolutionary Computation 6 (2012) 1–24.

[4] M. Mavrovouniotis, C. Li, S. Yang, A survey of swarm intelligence for dynamic optimization: Algorithms and applications, Swarm and Evolutionary Computation 33 (2017) 1–17.

[5] D. Rohde, B. R. Knudsen, T. Andresen, N. Nord, Dynamic optimization of control setpoints for an integrated heating and cooling system with thermal energy storages, Energy 193 (2020) 116771.

[6] B. Sun, Y. Yang, J. Shi, L. Zheng, Dynamic pick-up and delivery optimization with multiple dynamic events in real-world environment, IEEE Access 7 (2019) 146209–146220.

[7] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, in: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Vol. 3, IEEE, 1999, pp. 1875–1882.

[8] X. Li, M. G. Epitropakis, K. Deb, A. Engelbrecht, Seeking multiple solutions: an updated survey on niching methods and their applications, IEEE Transactions on Evolutionary Computation 21 (4) (2016) 518–538.

[9] I. Moser, R. Chiong, Dynamic function optimization: the moving peaks benchmark, in: Metaheuristics for Dynamic Optimization, Springer, 2013, pp. 35–59.

[10] D. Yazdani, M. N. Omidvar, J. Branke, T. T. Nguyen, X. Yao, Scaling up dynamic optimization problems: A divide-and-conquer approach, IEEE Transactions on Evolutionary Computation (2019).

[11] C. Li, S. Yang, A general framework of multipopulation methods with clustering in undetectable dynamic environments, IEEE Transactions on Evolutionary Computation 16 (4) (2012) 556–577.

[12] Y. Wang, J. Yu, S. Yang, S. Jiang, S. Zhao, Evolutionary dynamic constrained optimization: Test suite construction and algorithm comparisons, Swarm and Evolutionary Computation 50 (2019) 100559.

[13] S. Dennis, A. Engelbrecht, A review and empirical analysis of particle swarm optimization algorithms for dynamic multi-modal optimization, in: 2020 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2020, pp. 1–8.

[14] C. Li, S. Yang, A generalized approach to construct benchmark problems for dynamic optimization, in: Asia-Pacific Conference on Simulated Evolution and Learning, Springer, 2008, pp. 391–400.

[15] S. Kundu, S. Biswas, S. Das, P. N. Suganthan, Crowding-based local differential evolution with speciation-based memory archive for dynamic multimodal optimization, in: Proceedings of the 15th annual conference on Genetic and evolutionary computation, ACM, 2013, pp. 33–40.

[16] W. Luo, X. Lin, T. Zhu, P. Xu, A clonal selection algorithm for dynamic multimodal function optimization, Swarm and Evolutionary Computation 50 (2019) 100459.

[17] S. Cheng, H. Lu, Y.-n. Guo, X. Lei, J. Liang, J. Chen, Y. Shi, Dynamic multimodal optimization: A preliminary study, in: 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2019, pp. 279–285.

[18] A. Ahrari, S. Elsayed, R. Sarker, D. Essam, C. A. C. Coello, A novel parametric benchmark generator for dynamic multimodal optimization, Swarm and Evolutionary Computation (2021) 100924doi:10.1016/j.swevo.2021.100924.

[19] N. Hansen, S. Finck, R. Ros, A. Auger, et al., Real-parameter black-box optimization benchmarking 2009: noiseless functions definitions, Tech. Rep. RR-6829, INRIA (2009).

[20] K. M. Malan, et al., Characterising continuous optimisation problems for particle swarm optimisation performance prediction, Ph.D. thesis, University of Pretoria (2014).

[21] R. Bond, A. P. Engelbrecht, B. Ombuki-Berman, Evaluating landscape characteristics of dynamic benchmark functions, in: 2015 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2015, pp. 1343–1350.

[22] A. Ahrari, K. Deb, A novel class of test problems for performance evaluation of niching methods, IEEE Transactions on Evolutionary Computation 22 (6) (2017) 909–919.

[23] S. Jiang, S. Yang, X. Yao, K. C. Tan, M. Kaiser, N. Krasnogor, Benchmark problems for cec2018 competition on dynamic multiobjective optimisation, Tech. rep., Newcastle University (2017).

[24] X. Li, A. Engelbrecht, M. G. Epitropakis, Benchmark functions for cec'2013 special session and competition on niching methods for multimodal function optimization, Tech. rep., RMIT University (2013).

[25] J. Mwaura, A. P. Engelbrecht, F. V. Nepocumeno, Performance measures for niching algorithms, in: 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 4775–4784.

[26] S. Das, S. Maity, B.-Y. Qu, P. N. Suganthan, Real-parameter evolutionary multimodal optimization-a survey of the state-of-the-art, Swarm and Evolutionary Computation 1 (2) (2011) 71–88.

[27] S. Cheng, H. Lu, W. Song, J. Chen, Y. Shi, Dynamic multimodal optimization using brain storm optimization algorithms, in: International Conference on Bio-Inspired Computing: Theories and Applications, Springer, 2018, pp. 236–245.