pyRVEA

Release 0.1

Bhupinder Saini

May 01, 2019

CONTENTS

CHAPTER

ONE

README

Binder

The python version reference vector guided evolutionary algorithm.

Currently supported: Multi-objective minimization with visualization and interaction support. Preference is accepted as a reference point.

To test the code, open the binder link and read example.ipynb.

Read the documentation here

1.1 Requirements:

- Python 3.6 or up
- Poetry dependency manager

1.2 Installation process:

- Download and extract the code
- Create a new virtual environment for the project
- Run `poetry install --no-dev` in the activated virtual environment to install the packages necessary to run the code.
- If you want to take part in the development process, run `poetry install` instead.

1.3 See the details of RVEA in the following paper

R. Cheng, Y. Jin, M. Olhofer and B. Sendhoff, A Reference Vector Guided Evolutionary Algorithm for Many-objective Optimization, IEEE Transactions on Evolutionary Computation, 2016

The source code of pyRVEA is implemented by Bhupinder Saini

If you have any questions about the code, please contact:

Bhupinder Saini: bhupinder.s.saini@jyu.fiProject researcher at University of Jyväskylä.

CHAPTER

TWO

PYRVEA PACKAGE

2.1 pyRVEA.EAs

2.1.1 pyRVEA.EAs.NSGAIII module

class pyRVEA.EAs.NSGAIII.**NSGAIII**(*population: Population*, *EA_parameters: dict = None*)

Bases: *pyRVEA.EAs.baseEA.BaseDecompositionEA*

Python Implementation of NSGA-III. Based on the pymoo package.

[description]

select(*population: Population*)

Describe a selection mechanism. Return indices of selected individuals.

Parameters **population** (Population) – Contains the current population and problem information.

Returns List of indices of individuals to be selected.

Return type list

set_params(*population: Population = None*, *population_size: int = None*, *lattice_resolution: int = None*, *interact: bool = True*, *a_priori_preference: bool = False*, *generations_per_iteration: int = 100*, *iterations: int = 10*, *plotting: bool = True*)

Set up the parameters. Save in self.params

2.1.2 pyRVEA.EAs.RVEA module

class pyRVEA.EAs.RVEA.**RVEA**(*population: Population*, *EA_parameters: dict = None*)

Bases: *pyRVEA.EAs.baseEA.BaseDecompositionEA*

The python version reference vector guided evolutionary algorithm.

See the details of RVEA in the following paper

R. Cheng, Y. Jin, M. Olhofer and B. Sendhoff, A Reference Vector Guided Evolutionary Algorithm for Many-objective Optimization, IEEE Transactions on Evolutionary Computation, 2016

The source code of pyRVEA is implemented by Bhupinder Saini

If you have any questions about the code, please contact:

Bhupinder Saini: bhupinder.s.saini@jyu.fi

Project researcher at University of Jyväskylä.

select (*population: Population*)

Describe a selection mechanism. Return indices of selected individuals.

APD Based selection. # This is different from the paper. # params.genetations != total number of generations. This is a compromise. Also this APD uses an archived ideal point, rather than current, potentially worse ideal point.

Parameters **population** (Population) – Population information

Returns list: Indices of selected individuals.

Return type list

set_params (*population: Population = None, population_size: int = None, lattice_resolution: int = None, interact: bool = False, a_priori_preference: bool = False, generations_per_iteration: int = 100, iterations: int = 10, Alpha: float = 2, plotting: bool = True*)

Set up the parameters. Save in RVEA.params. Note, this should be changed to align with the current structure.

Parameters

- **population** (Population) – Population object
- **population_size** (*int*) – Population Size
- **lattice_resolution** (*int*) – Lattice resolution
- **interact** (*bool*) – bool to enable or disable interaction. Enabled if True
- **a_priori_preference** (*bool*) – similar to interact
- **generations_per_iteration** (*int*) – Number of generations per iteration.
- **iterations** (*int*) – Total Number of iterations.
- **Alpha** (*float*) – The alpha parameter of APD selection.
- **plotting** (*bool*) – Useless really.

2.1.3 pyRVEA.EAs.baseEA module

class pyRVEA.EAs.baseEA.**BaseDecompositionEA** (*population: Population, EA_parameters: dict = None*)

Bases: *pyRVEA.EAs.baseEA.BaseEA*

This class provides the basic structure for decomposition based Evolutionary algorithms, such as RVEA or NSGA-III.

continue_evolution () → bool

Checks whether the current iteration should be continued or not.

continue_iteration ()

Checks whether the current iteration should be continued or not.

select (*population*) → list

Describe a selection mechanism. Return indices of selected individuals.

Parameters **population** (Population) – Contains the current population and problem information.

Returns List of indices of individuals to be selected.

Return type list

class pyRVEA.EAs.baseEA.**BaseEA**

Bases: object

This class provides the basic structure for Evolutionary algorithms.

set_params()

Set up the parameters. Save in self.params

2.2 pyRVEA.OtherTools

2.2.1 pyRVEA.OtherTools.IsNotebook module

pyRVEA.OtherTools.IsNotebook.**IsNotebook**() → bool

Checks if the current environment is a Jupyter Notebook or a console.

Returns True if notebook. False if console

Return type bool

2.2.2 pyRVEA.OtherTools.ReferenceVectors module

class pyRVEA.OtherTools.ReferenceVectors.**ReferenceVectors**(*lattice_resolution: int*, *number_of_objectives: int*, *creation_type: str = 'Uniform'*, *vector_type: str = 'Spherical'*, *ref_point: list = None*)

Bases: object

Class object for reference vectors.

adapt(*fitness: numpy.ndarray*)

Adapt reference vectors. Then normalize.

Parameters **fitness** (*np.ndarray*) –

add_edge_vectors()

Add edge vectors to the list of reference vectors.

Used to cover the entire orthant when preference information is provided.

iteractive_adapt_1(*ref_point*, *translation_param=0.2*)

Adapt reference vectors linearly towards a reference point. Then normalize.

The details can be found in the following paper: Hakanen, Jussi & Chugh, Tinkle & Sindhya, Karthik & Jin, Yaochu & Miettinen, Kaisa. (2016). Connections of Reference Vectors and Different Types of Preference Information in Interactive Multiobjective Evolutionary Algorithms.

Parameters

- **ref_point** –
- **translation_param** – (Default value = 0.2)

neighbouring_angles() → numpy.ndarray

Calculate neighbouring angles for normalization.

normalize()

Normalize the reference vectors to a unit hypersphere.

slow_interactive_adapt(*ref_point*)
Basically a wrapper around rotate_toward. Slowly rotate ref vectors toward ref_point. Return a boolean value to tell if the ref_point has been reached.

Parameters **ref_point** (*list or np.ndarray*) – The reference vectors will slowly move towards the ref_point.

Returns True if ref_point has been reached. False otherwise.

Return type boolean

pyRVEA.OtherTools.ReferenceVectors.**householder**(*vector*)
Return reflection matrix via householder transformation.

pyRVEA.OtherTools.ReferenceVectors.**normalize**(*vectors*)
Normalize a set of vectors.

The length of the returned vectors will be unity.

Parameters **vectors** (*np.ndarray*) – Set of vectors of any length, except zero.

pyRVEA.OtherTools.ReferenceVectors.**rotate**(*initial_vector*, *rotated_vector*, *other_vectors*)
Calculate the rotation matrix that rotates the initial_vector to the rotated_vector. Apply that rotation on other_vectors and return. Uses Householder reflections twice to achieve this.

pyRVEA.OtherTools.ReferenceVectors.**rotate_toward**(*initial_vector*, *final_vector*, *other_vectors*, *degrees: float = 5*)
Rotate other_vectors (with the centre at initial_vector) towards final_vector by an angle degrees.

Parameters

- **initial_vector** (*np.ndarray*) – Centre of the vectors to be rotated.
- **final_vector** (*np.ndarray*) – The final position of the center of other_vectors.
- **other_vectors** (*np.ndarray*) – The array of vectors to be rotated
- **degrees** (*float, optional*) – The amount of rotation (the default is 5)

Returns

- **rotated_vectors** (*np.ndarray*) – The rotated vectors
- **reached** (*bool*) – True if final_vector has been reached

pyRVEA.OtherTools.ReferenceVectors.**shear**(*vectors*, *degrees: float = 5*)
Shear a set of vectors lying on the plane z=0 towards the z-axis, such that the resulting vectors 'degrees' angle away from the z axis.

z is the last element of the vector, and has to be equal to zero.

Parameters

- **vectors** (*numpy.ndarray*) – The final element of each vector should be zero.
- **degrees** (*float, optional*) – The angle that the resultant vectors make with the z axis. Unit is radians. (the default is 5)

2.2.3 pyRVEA.OtherTools.newRV module

2.2.4 pyRVEA.OtherTools.plotlyanimate module

`pyRVEA.OtherTools.plotlyanimate.`**`animate_2d_init_`**(*data: Union[numpy.ndarray, pandas.core.frame.DataFrame, list], filename: str*) → dict

Initiate a 2D scatter animation.

Only for 2D data.

Parameters

- **data** (*Union[np.ndarray, pd.DataFrame, list]*) – Objective values
- **filename** (*str*) – Name of the file to which plot is saved

Returns Plotly Figure Object

Return type dict

`pyRVEA.OtherTools.plotlyanimate.`**`animate_2d_next_`**(*data: Union[numpy.ndarray, pandas.core.frame.DataFrame, list], figure: dict, filename: str, generation: int*) → dict

Plot the next set of individuals in a 2D scatter animation.

Parameters

- **data** (*Union[np.ndarray, pd.DataFrame, list]*) – The objective values to be plotted
- **figure** (*dict*) – Plotly figure object compatible dict
- **filename** (*str*) – Name of the file to which the plot is saved
- **generation** (*int*) – Iteration Number

Returns Plotly Figure Object

Return type dict

`pyRVEA.OtherTools.plotlyanimate.`**`animate_3d_init_`**(*data: Union[numpy.ndarray, pandas.core.frame.DataFrame, list], filename: str*) → dict

Plot the first (or zeroth) iteration of a population.

Intended as a frames object. Plots Scatter 3D data.

Parameters

- **data** (*Union[np.ndarray, pd.DataFrame, list]*) – Contains the data to be plotted. Each row is an individual's objective values.
- **filename** (*str*) – Contains the name of the file to which the plot is saved.

Returns Plotly figure object

Return type dict

`pyRVEA.OtherTools.plotlyanimate.`**`animate_3d_next_`**(*data: Union[numpy.ndarray, pandas.core.frame.DataFrame, list], figure: dict, filename: str, generation: int*) → dict

Plot the next set of individuals in an animation.

Plots scatter for 3D data.

Parameters

- **data** (*Union[np.ndarray, pd.DataFrame, list]*) – The objective values to be plotted
- **figure** (*dict*) – Plotly figure object compatible dict
- **filename** (*str*) – Name of the file to which the plot is saved
- **generation** (*int*) – Iteration Number

Returns Plotly Figure Object

Return type dict

pyRVEA.OtherTools.plotlyanimate.**animate_init_**(*data: Union[numpy.ndarray, pandas.core.frame.DataFrame, list], filename: str*) → dict

Plot the first (or zeroth) iteration of a population.

Intended as a frames object. Plots Scatter for 2D and 3D data. Plots parallel coordinate plot for higher dimensional data.

Parameters

- **data** (*Union[np.ndarray, pd.DataFrame, list]*) – Contains the data to be plotted. Each row is an individual's objective values.
- **filename** (*str*) – Contains the name of the file to which the plot is saved.

Returns Plotly figure object

Return type dict

pyRVEA.OtherTools.plotlyanimate.**animate_next_**(*data: Union[numpy.ndarray, pandas.core.frame.DataFrame, list], figure: dict, filename: str, generation: int*) → dict

Plot the next set of individuals in an animation.

Plots scatter for 2D and 3D data, parallel coordinate plot for 4D and up.

Parameters

- **data** (*Union[np.ndarray, pd.DataFrame, list]*) – The objective values to be plotted
- **figure** (*dict*) – Plotly figure object compatible dict
- **filename** (*str*) – Name of the file to which the plot is saved
- **generation** (*int*) – Iteration Number

Returns Plotly Figure Object

Return type dict

pyRVEA.OtherTools.plotlyanimate.**animate_parallel_coords_init_**(*data: Union[numpy.ndarray, pandas.core.frame.DataFrame, list], filename: str*) → dict

Plot the first (or zeroth) iteration of a population.

Intended as a frames object. Plots parallel coordinate plot for >3D data.

Parameters

- **data** (*Union[np.ndarray, pd.DataFrame, list]*) – Contains the data to be plotted. Each row is an individual's objective values.
- **filename** (*str*) – Contains the name of the file to which the plot is saved.

Returns Plotly figure object

Return type dict

pyRVEA.OtherTools.plotlyanimate.**animate_parallel_coords_next_**(*data: Union[numpy.ndarray, pandas.core.frame.DataFrame, list], figure: dict, filename: str, generation: int*) → dict

Plot the next set of individuals in an animation.

Plots parallel coordinate plot for 4D and up.

Parameters

- **data** (*Union[np.ndarray, pd.DataFrame, list]*) – The objective values to be plotted
- **figure** (*dict*) – Plotly figure object compatible dict
- **filename** (*str*) – Name of the file to which the plot is saved
- **generation** (*int*) – Iteration Number

Returns Plotly Figure Object

Return type dict

pyRVEA.OtherTools.plotlyanimate.**test**()

pyRVEA.OtherTools.plotlyanimate.**test2**()

2.2.5 pyRVEA.OtherTools.symmetric_vectors module

2.3 pyRVEA.Population

2.3.1 pyRVEA.Population.Population module

class pyRVEA.Population.Population.**Population**(*problem: baseProblem, assign_type: str = 'RandomAssign', plotting: bool = True, *args*)

Bases: object

Define the population.

add(*new_pop: numpy.ndarray*)

Evaluate and add individuals to the population. Update ideal and nadir point.

Parameters **new_pop** (*np.ndarray*) – Decision variable values for new population.

append_individual(*ind: numpy.ndarray*)
Evaluate and add individual to the population.

Parameters **ind** (*np.ndarray*) –

create_new_individuals(*design: str = 'LHSDesign', pop_size: int = None, decision_variables=None*)
Create, evaluate and add new individuals to the population. Initiate Plots.

The individuals can be created randomly, by LHS design, or can be passed by the user.

Parameters

- **design** (*str, optional*) – Describe the method of creation of new individuals. "RandomDesign" creates individuals randomly. "LHSDesign" creates individuals using Latin hypercube sampling.
- **pop_size** (*int, optional*) – Number of individuals in the population. If none, some default population size based on number of objectives is chosen.
- **decision_variables** (*numpy array or list, optional*) – Pass decision variables to be added to the population.

eval_fitness()
Calculate fitness based on objective values. Fitness = obj if minimized.

evaluate_individual(*ind: numpy.ndarray*)
Evaluate individual.

Returns objective values, constraint violation, and fitness.

Parameters **ind** (*np.ndarray*) –

evolve(*EA: BaseEA = None, EA_parameters: dict = None*) → Population
Evolve the population with interruptions.

Evolves the population based on the EA sent by the user.

Parameters

- **EA** (*"BaseEA"*) – Should be a derivative of BaseEA (Default value = None)
- **EA_parameters** (*dict*) – Contains the parameters needed by EA (Default value = None)

hypervolume(*ref_point*)
Calculate hypervolume. Uses package pygmo. Add checks to prevent errors.

Parameters **ref_point** –

keep(*indices: list*)
Remove individuals from population which are not in "indices".

Parameters **indices** (*list*) – Indices of individuals to keep

mate()
Conduct crossover and mutation over the population.

Conduct simulated binary crossover and bounded polunomial mutation.

non_dominated()
Fix this. check if nd2 and nds mean the same thing

plot_init_()
Initialize animation objects. Return figure

plot_objectives (*iteration: int*)
Plot the objective values of individuals in notebook. This is a hack.

Parameters iteration (*int*) – Iteration count.

update_ideal_and_nadir (*new_objective_vals: list = None*)
Updates self.ideal and self.nadir in the fitness space.

Uses the entire population if new_objective_vals is none.

Parameters new_objective_vals (*list, optional*) – Objective values for a newly added individual (the default is None, which calculated the ideal and nadir for the entire population.)

2.4 pyRVEA.Problem package

2.4.1 Submodules

2.4.2 pyRVEA.Problem.baseProblem module

class pyRVEA.Problem.baseProblem.**baseProblem** (*name=None, num_of_variables=None, num_of_objectives=None, num_of_constraints=0, upper_limits=1, lower_limits=0*)

Bases: `object`

Base class for the problems.

constraints (*decision_variables*)
Accept a sample and/or corresponding objective values.

Parameters decision_variables –

objectives (*decision_variables*)
Accept a sample. Return Objective values.

Parameters decision_variables –

update ()
Update the problem based on new information.

2.4.3 pyRVEA.Problem.testProblem module

class pyRVEA.Problem.testProblem.**testProblem** (*name=None, num_of_variables=None, num_of_objectives=None, num_of_constraints=0, upper_limits=1, lower_limits=0*)

Bases: *pyRVEA.Problem.baseProblem.baseProblem*

Defines the problem.

constraints (*decision_variables, objective_variables*)
Calculate constraint violation.

Parameters

- **decision_variables** –
- **objective_variables** –

objectives (*decision_variables*) → list

Use this method to calculate objective functions.

Parameters **decision_variables** –

2.5 pyRVEA.Recombination package

2.5.1 Module contents

2.6 pyRVEA.Selection package

2.6.1 Submodules

2.6.2 pyRVEA.Selection.APD_select module

pyRVEA.Selection.APD_select.**APD_select** (*fitness: list, vectors: ReferenceVectors, penalty_factor: float, ideal: list = None*)

Select individuals for mating on basis of Angle penalized distance.

Parameters

- **fitness** (`list`) – Fitness of the current population.
- **vectors** (`ReferenceVectors`) – Class containing reference vectors.
- **penalty_factor** (`float`) – Multiplier of angular deviation from Reference vectors. See RVEA paper for details.
- **ideal** (`list`) – ideal point for the population. Uses the min fitness value if None.

Returns A list of indices of the selected individuals.

Return type [type]

2.6.3 pyRVEA.Selection.NSGAIII_select module

pyRVEA.Selection.NSGAIII_select.**NSGAIII_select** (*fitness: list, ref_dirs: ReferenceVectors, ideal_point: list = None, worst_point: list = None, extreme_points: list = None, n_survive: int = None*)

pyRVEA.Selection.NSGAIII_select.**associate_to_niches** (*F, ref_dirs, ideal_point, nadir_point, utopian_epsilon=0.0*)

pyRVEA.Selection.NSGAIII_select.**calc_niche_count** (*n_niches, niche_of_individuals*)

pyRVEA.Selection.NSGAIII_select.**calc_perpendicular_distance** (*N, ref_dirs*)

pyRVEA.Selection.NSGAIII_select.**get_extreme_points_c** (*F, ideal_point, extreme_points=None*)

Taken from pymoo

pyRVEA.Selection.NSGAIII_select.**get_nadir_point** (*extreme_points, ideal_point, worst_point, worst_of_front, worst_of_population*) `pyRVEA.Selection.NSGAIII_select.`**`niching`**(*F*, *n_remaining*, *niche_count*, *niche_of_individuals*, *dist_to_niche*)

PYTHON MODULE INDEX

p

INDEX