pyM2SA Documentation

Release

Antonio Benítez-Hidalgo

Oct 04, 2018

Contents:

pyM2SA is an open source software tool aimed at for solving Multiple Sequence Alignment problems with multi-objective metaheuristics.

Warning: Documentation is WIP!! Some information may be missing.

CHAPTER 1

About

pyM2SA is being developed by Antonio Benítez-Hidalgo (email) and Antonio J. Nebro (email), associate professor at the University of Málaga.

CHAPTER 2

API documentation

2.1 Algorithms

2.1.1 Multiobjective algorithms

dNSGA-II

pym2sa.algorithm.multiobjective.dnsgaii.**R**

pym2sa.algorithm.multiobjective.dnsgaii.**create_new_solution**(*problem*)

class pym2sa.algorithm.multiobjective.dnsgaii.**dNSGAII**(*population_size: int, problem: jmetal.core.problem.Problem[S], max_evaluations: int, mutation: jmetal.core.operator.Mutation[S], crossover: jmetal.core.operator.Crossover[S, S], selection: jmetal.core.operator.Selection[typing.List[S], S], number_of_cores: int, client: <Mock name='mock.Client' id='140082754666000'>*)

Bases: jmetal.core.algorithm.Algorithm

create_initial_population() → typing.List[S]

get_name() → str

get_result()

run()

update_progress(*population*)

pym2sa.algorithm.multiobjective.dnsgaii.**reproduction**(*population: typing.List[S], problem: jmetal.core.problem.Problem[S], crossover_operator: jmetal.core.operator.Crossover[S, S], mutation_operator: jmetal.core.operator.Mutation[S]*) → S

Cross and mutate a list of solutions and return an individual (whichever scores better attending to one objective).

2.2 Components

2.2.1 Evaluator

class pym2sa.component.evaluator.**DelayedEvaluator**

Bases: jmetal.component.evaluator.Evaluator

evaluate(*solution_list: typing.List[S], problem: jmetal.core.problem.Problem*) → typing.List[S]

class pym2sa.component.evaluator.**MapEvaluator**(*n_workers: int = 4*)

Bases: jmetal.component.evaluator.Evaluator

evaluate(*solution_list: typing.List[S], problem: jmetal.core.problem.Problem*) → typing.List[S]

class pym2sa.component.evaluator.**MultithreadedEvaluator**(*n_workers: int = 1*)

Bases: jmetal.component.evaluator.Evaluator

evaluate(*solution_list: typing.List[S], problem: jmetal.core.problem.Problem*) → typing.List[S]

class pym2sa.component.evaluator.**ProcessPoolEvaluator**(*processes: int = 4*)

Bases: *pym2sa.component.evaluator.SubmitEvaluator*

class pym2sa.component.evaluator.**SubmitEvaluator**(*submit_func*)

Bases: jmetal.component.evaluator.Evaluator

evaluate(*solution_list: typing.List[S], problem: jmetal.core.problem.Problem*) → typing.List[S]

class pym2sa.component.evaluator.**ThreadPoolEvaluator**(*workers: int = 4*)

Bases: jmetal.component.evaluator.Evaluator

evaluate(*solution_list: typing.List[S], problem: jmetal.core.problem.Problem*) → typing.List[S]

2.2.2 Observer

class pym2sa.component.observer.**WriteSequencesToFileObserver**(*output_directory*) → None

Bases: object

update(**args*, ***kwargs*)

2.3 Core

2.3.1 Problem

class pym2sa.core.problem.**MSAProblem**
Bases: jmetal.core.problem.Problem

Class representing MSA problems

evaluate(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

get_name() → str

2.3.2 Solution

class pym2sa.core.solution.**MSASolution**(*problem*, *msa: list*) → None
Bases: jmetal.core.solution.Solution

Class representing MSA solutions.

add_gap_to_sequence_at_index(*seq_index: int*, *gap_position: int*)
Add one gap to an specific sequence.

Parameters

- **seq_index** – Index of the sequence on the alignment.
- **gap_position** – Index of the gap.

decode_alignment_as_list_of_pairs() → list

decode_alignment_as_list_of_sequences() → list

decode_sequence_at_index(*seq_index: int*)

get_char_position_in_original_sequence(*seq_index: int*, *position: int*)

get_gap_columns_from_alignment() → list
Get index of gap columns in the alignment.

get_length_of_alignment() → int
Get length of the alignment (i.e., length of the first sequence).

get_length_of_gaps(*seq_index: int*) → int

get_length_of_sequence(*seq_index: int*) → int
Get length of an specific sequence.

Parameters **seq_index** – Index of the sequence in the alignment.

get_next_char_position_after_gap(*seq_index: int*, *gap_position: int*)

get_number_of_gaps_groups_of_sequence(*seq_index: int*) → float
Get number of gaps groups of an specific sequence.

get_number_of_gaps_of_sequence_at_index(*seq_index: int*)
Get number of gaps of an specific sequence.

Parameters **seq_index** – Index of the sequence in the alignment.

get_original_char_position_in_aligned_sequence(*seq_index: int*, *position: int*)

get_total_number_of_gaps() → int
Get total number of gaps in the alignment.

is_gap_char_at_sequence(*seq_index: int*, *index: int*) → bool

is_gap_column(*column: int*) → bool
Check if an specific column in the alignment is in all gaps groups (i.e., column consist only of gaps).

Parameters **column** – Index of the column in the alignment.

is_valid_msa() → bool
Check if all sequences of the alignment have the same length.

merge_gaps_groups() → None
Merge consecutive gaps groups in the alignment.

remove_full_of_gaps_columns() → None
Remove columns that consist only of gaps.

remove_gap_column(*column: int*) → None

remove_gap_from_sequence(*seq_index: int*, *position: int*)

remove_gap_group_from_sequence_at_column(*seq_index: int*, *column_index: int*) → None

split_gap_column(*column: int*) → None

2.4 Operators

2.4.1 Crossover

class pym2sa.operator.crossover.**HMSA**(*probability: float*) → None
Bases: jmetal.core.operator.Crossover

Implements an horizontal recombination for MSA.

do_crossover(*parents: typing.List[pym2sa.core.solution.MSASolution]*) → typing.List[pym2sa.core.solution.MSASolution]

execute(*parents: typing.List[pym2sa.core.solution.MSASolution]*) → typing.List[pym2sa.core.solution.MSASolution]

get_name() → str

get_number_of_parents() → int

class pym2sa.operator.crossover.**SPXMSA**(*probability: float*, *remove_gap_columns: bool = True*) → None
Bases: jmetal.core.operator.Crossover

Implements a single point crossover for MSA.

cross_parents(*cx_point: int, parents: typing.List[pym2sa.core.solution.MSASolution], cutting_points_in_first_parent: list, column_positions_in_second_parent: list*) → typing.List[pym2sa.core.solution.MSASolution]

do_crossover(*parents: typing.List[pym2sa.core.solution.MSASolution]*) → typing.List[pym2sa.core.solution.MSASolution]

execute(*parents: typing.List[pym2sa.core.solution.MSASolution]*) → typing.List[pym2sa.core.solution.MSASolution]

fill_sequences_with_gaps_to_reach_the_max_sequence_length(*solution: pym2sa.core.solution.MSASolution*, *max_length: int*, *cutting_points: list*)

find_cutting_points_in_first_parent(*solution: pym2sa.core.solution.MSASolution*, *position: int*) → list

Find the real cutting points in a solution. If the column is a gap then the next non-gap symbol must be found

find_length_of_the_largest_sequence(*solution: pym2sa.core.solution.MSASolution*)

find_original_positions_in_original_sequences(*solution: pym2sa.core.solution.MSASolution*, *column: int*) → list

Given a solution, find for each sequence the original positions of the symbol in the column in the original unaligned sequences

get_name() → str

get_number_of_parents() → int

2.4.2 Mutation

class pym2sa.operator.mutation.**MultipleMSAMutation**(*operator: typing.List[jmetal.core.operator.Mutation[S]]*, *probability: float*) → None

Bases: jmetal.core.operator.Mutation

do_mutation(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

execute(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

get_name() → str

class pym2sa.operator.mutation.**OneRandomGapInsertion**(*probability: float*, *remove_gap_columns: bool = False*) → None

Bases: jmetal.core.operator.Mutation

do_mutation(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

execute(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

get_name() → str

class pym2sa.operator.mutation.**ShiftClosedGapGroups**(*probability: float*, *remove_gap_columns: bool = True*) → None

Bases: jmetal.core.operator.Mutation

For every sequence, selects a random group and shift it with the closest gap group.

do_mutation(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

execute(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

get_name() → str

class pym2sa.operator.mutation.**ShiftGapGroup**(*probability: float*, *remove_gap_columns: bool = True*) → None

Bases: jmetal.core.operator.Mutation

Selects a gap group randomly in all the sequences of a solution and shifts it one position to the left or to the right.

do_mutation(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

execute(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

get_name() → str

class pym2sa.operator.mutation.**TwoRandomAdjacentGapGroup**(*probability: float*, *remove_gap_columns: bool = True*) → None

Bases: jmetal.core.operator.Mutation

Selects a random gap group and merges it with the adjacent gaps group.

do_mutation(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

execute(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution

get_name() → str

2.5 Problems

2.5.1 BAliBASE

class pym2sa.problem.BAliBASE.**BAliBASE**(*balibase_instance: str*, *balibase_path: str*, *score_list: typing.List[pymsa.core.score.Score]*) → None

Bases: *pym2sa.problem.MSA.MSA*

Creates a new problem based on an instance of BAliBASE.

Parameters

- **balibase_instance** – Instance name (e.g., BB12010).
- **balibase_path** – Path containing two directories: *bb_aligned*, with the pre-computed alignments and *bb_release*, with the original sequences.
- **score_list** – List of scores.

DATA_FILES = ['tfa_clu', 'tfa_muscle', 'tfa_kalign', 'tfa_retalign', 'fasta_aln', 'tfa

create_solution() → pym2sa.core.solution.MSASolution
Read and import an instance of BAliBASE.

get_name() → str

2.5.2 Generic MSA

class pym2sa.problem.MSA.**MSA**(*score_list: typing.List[pymsa.core.score.Score]*, *sequences_without_gaps: typing.List[str]*, *sequences_names: typing.List[str]*)

Bases: *pym2sa.core.problem.MSAProblem*

Creates a new generic MSA problem.

Parameters

- **score_list** – List of scores to evaluate MSAs.

- **sequences_without_gaps** – List of original sequences (without gaps).
- **sequences_names** – List of sequences names.

create_solution() → pym2sa.core.solution.MSASolution

evaluate(*solution: pym2sa.core.solution.MSASolution*) → pym2sa.core.solution.MSASolution
Evaluate a multiple sequence alignment solution.

Parameters **solution** – MSA to evaluate.

get_name() → str

2.6 Utils

2.6.1 Graphic

class pym2sa.util.graphic.**MSAPlot**(*plot_title: str*, *axis_labels: list = None*)
Bases: jmetal.util.graphic.FrontPlot

Creates a new *MSAPlot* instance. Suitable for problems with 2 or more objectives.

Parameters

- **plot_title** – Title of the graph.
- **axis_labels** – List of axis labels.

to_html(*filename: str = 'front'*) → None
Export the graph to an interactive HTML (solutions can be selected to show some metadata).

Parameters **filename** – Output file name.

CHAPTER 3

Installation steps

Via pip:

```
$ pip install pym2sa
```

Via Github:

```
$ git clone https://github.com/benhid/pyM2SA.git
$ cd pyM2SA
$ python setup.py install
```

CHAPTER 4

Features

- **The scores that are currently available are those from pyMSA (v0.5.1):**
  - Sum of pairs,
  - Star,
  - Minimum entropy,
  - Percentage of non-gaps,
  - Percentage of totally conserved columns,
  - STRIKE.
- **The algorithm that is currently available is:**
  - NSGA-II
- **Crossover operator:**
  - Single-point crossover (`GapSequenceSolutionSinglePoint`).
- **Mutation operators:**
  - Shift closest gap group (`ShiftClosedGapGroups`),
  - Shift gap group (`ShiftGapGroup`),
  - Random gap insertion (`OneRandomGapInsertion`),
  - Merge two random adjacent gaps group (`TwoRandomAdjacentGapGroup`),
  - Multiple mutation (`MultipleMSAMutation`).

Python Module Index

d

p

Index

P

R

S

T

U

W