

---

# **pyM2SA Documentation**

***Release***

**Antonio Benítez-Hidalgo**

**Oct 04, 2018**



---

## Contents:

---

<b>1</b>	<b>About</b>	<b>3</b>
<b>2</b>	<b>API documentation</b>	<b>5</b>
<b>3</b>	<b>Installation steps</b>	<b>13</b>
<b>4</b>	<b>Features</b>	<b>15</b>
	<b>Python Module Index</b>	<b>17</b>



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.





## 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[pym2sa.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\_realign', '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[pym2sa.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
```



- **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`).



### d

`dnsgaii` (*Unix, Windows*), [5](#)

### p

`pym2sa.algorithm.multiobjective.dnsgaii`,  
[5](#)

`pym2sa.component.evaluator`, [6](#)

`pym2sa.component.observer`, [6](#)

`pym2sa.core.problem`, [7](#)

`pym2sa.core.solution`, [7](#)

`pym2sa.operator.crossover`, [8](#)

`pym2sa.operator.mutation`, [9](#)

`pym2sa.problem.BAlibASE`, [10](#)

`pym2sa.problem.MSA`, [10](#)

`pym2sa.util.graphic`, [11](#)



## A

add\_gap\_to\_sequence\_at\_index()  
(pym2sa.core.solution.MSASolution method),  
7

## B

BALiBASE (class in pym2sa.problem.BALiBASE), 10

## C

create\_initial\_population()  
(pym2sa.algorithm.multiobjective.dnsgaii.dNSGAIIMethod), 5

create\_new\_solution() (in module  
pym2sa.algorithm.multiobjective.dnsgaii),  
5

create\_solution() (pym2sa.problem.BALiBASE.BALiBASE  
method), 10

create\_solution() (pym2sa.problem.MSA.MSA method),  
11

cross\_parents() (pym2sa.operator.crossover.SPXMSA  
method), 8

## D

DATA\_FILES (pym2sa.problem.BALiBASE.BALiBASE  
attribute), 10

decode\_alignment\_as\_list\_of\_pairs()  
(pym2sa.core.solution.MSASolution method),  
7

decode\_alignment\_as\_list\_of\_sequences()  
(pym2sa.core.solution.MSASolution method),  
7

decode\_sequence\_at\_index()  
(pym2sa.core.solution.MSASolution method),  
7

DelayedEvaluator (class in  
pym2sa.component.evaluator), 6

dNSGAIIMethod (class in pym2sa.algorithm.multiobjective.dnsgaii),  
5

dNSGAIIMethod (module), 5

do\_crossover() (pym2sa.operator.crossover.HMSA  
method), 8

do\_crossover() (pym2sa.operator.crossover.SPXMSA  
method), 8

do\_mutation() (pym2sa.operator.mutation.MultipleMSAMutation  
method), 9

do\_mutation() (pym2sa.operator.mutation.OneRandomGapInsertion  
method), 9

do\_mutation() (pym2sa.operator.mutation.ShiftClosedGapGroups  
method), 9

do\_mutation() (pym2sa.operator.mutation.ShiftGapGroup  
method), 10

do\_mutation() (pym2sa.operator.mutation.TwoRandomAdjacentGapGroup  
method), 10

## E

evaluate() (pym2sa.component.evaluator.DelayedEvaluator  
method), 6

evaluate() (pym2sa.component.evaluator.MapEvaluator  
method), 6

evaluate() (pym2sa.component.evaluator.MultithreadedEvaluator  
method), 6

evaluate() (pym2sa.component.evaluator.SubmitEvaluator  
method), 6

evaluate() (pym2sa.component.evaluator.ThreadPoolEvaluator  
method), 6

evaluate() (pym2sa.core.problem.MSASolution method),  
7

evaluate() (pym2sa.problem.MSA.MSA method), 11

execute() (pym2sa.operator.crossover.HMSA method), 8

execute() (pym2sa.operator.crossover.SPXMSA method),  
8

execute() (pym2sa.operator.mutation.MultipleMSAMutation  
method), 9

execute() (pym2sa.operator.mutation.OneRandomGapInsertion  
method), 9

execute() (pym2sa.operator.mutation.ShiftClosedGapGroups  
method), 9

execute() (pym2sa.operator.mutation.ShiftGapGroup  
method), 10

execute() (pym2sa.operator.mutation.TwoRandomAdjacentGapGroup method), 10

**F**

fill\_sequences\_with\_gaps\_to\_reach\_the\_max\_sequence\_length() (pym2sa.operator.crossover.SPXMSA method), 8

find\_cutting\_points\_in\_first\_parent() (pym2sa.operator.crossover.SPXMSA method), 9

find\_length\_of\_the\_largest\_sequence() (pym2sa.operator.crossover.SPXMSA method), 9

find\_original\_positions\_in\_original\_sequences() (pym2sa.operator.crossover.SPXMSA method), 9

**G**

get\_char\_position\_in\_original\_sequence() (pym2sa.core.solution.MSASolution method), 7

get\_gap\_columns\_from\_alignment() (pym2sa.core.solution.MSASolution method), 7

get\_length\_of\_alignment() (pym2sa.core.solution.MSASolution method), 7

get\_length\_of\_gaps() (pym2sa.core.solution.MSASolution method), 7

get\_length\_of\_sequence() (pym2sa.core.solution.MSASolution method), 7

get\_name() (pym2sa.algorithm.multiobjective.dnsgaii.dNSGAI method), 5

get\_name() (pym2sa.core.problem.MSAProblem method), 7

get\_name() (pym2sa.operator.crossover.HMSA method), 8

get\_name() (pym2sa.operator.crossover.SPXMSA method), 9

get\_name() (pym2sa.operator.mutation.MultipleMSAMutation method), 9

get\_name() (pym2sa.operator.mutation.OneRandomGapInsertion method), 9

get\_name() (pym2sa.operator.mutation.ShiftClosedGapGroups method), 9

get\_name() (pym2sa.operator.mutation.ShiftGapGroup method), 10

get\_name() (pym2sa.operator.mutation.TwoRandomAdjacentGapGroup method), 10

get\_name() (pym2sa.problem.BaliBASE.BaliBASE method), 10

get\_name() (pym2sa.problem.MSA.MSA method), 11

get\_char\_position\_after\_gap() (pym2sa.core.solution.MSASolution method), 7

get\_number\_of\_gaps\_groups\_of\_sequence() (pym2sa.core.solution.MSASolution method), 7

get\_number\_of\_gaps\_of\_sequence\_at\_index() (pym2sa.core.solution.MSASolution method), 7

get\_number\_of\_parents() (pym2sa.operator.crossover.HMSA method), 8

get\_number\_of\_parents() (pym2sa.operator.crossover.SPXMSA method), 9

get\_original\_char\_position\_in\_aligned\_sequence() (pym2sa.core.solution.MSASolution method), 7

get\_result() (pym2sa.algorithm.multiobjective.dnsgaii.dNSGAI method), 5

get\_total\_number\_of\_gaps() (pym2sa.core.solution.MSASolution method), 7

**H**

HMSA (class in pym2sa.operator.crossover), 8

**I**

is\_gap\_char\_at\_sequence() (pym2sa.core.solution.MSASolution method), 8

is\_gap\_column() (pym2sa.core.solution.MSASolution method), 8

is\_valid\_msa() (pym2sa.core.solution.MSASolution method), 8

**M**

MapEvaluator (class in pym2sa.component.evaluator), 6

merge\_gaps\_groups() (pym2sa.core.solution.MSASolution method), 8

MSA (class in pym2sa.problem.MSA), 10

MSAPlot (class in pym2sa.util.graphic), 11

MSAProblem (class in pym2sa.core.problem), 7

MSASolution (class in pym2sa.core.solution), 7

MultipleMSAMutation (class in pym2sa.operator.mutation), 9

MultithreadedEvaluator (class in pym2sa.component.evaluator), 6

**O**

OneRandomGapInsertion (class in pym2sa.operator.mutation), 9



## P

ProcessPoolEvaluator (class in pym2sa.component.evaluator), 6  
 pym2sa.algorithm.multiobjective.dnsgaii (module), 5  
 pym2sa.component.evaluator (module), 6  
 pym2sa.component.observer (module), 6  
 pym2sa.core.problem (module), 7  
 pym2sa.core.solution (module), 7  
 pym2sa.operator.crossover (module), 8  
 pym2sa.operator.mutation (module), 9  
 pym2sa.problem.BAlIbASE (module), 10  
 pym2sa.problem.MSA (module), 10  
 pym2sa.util.graphic (module), 11

## R

R (in module pym2sa.algorithm.multiobjective.dnsgaii), 5  
 remove\_full\_of\_gaps\_columns() (pym2sa.core.solution.MSASolution method), 8  
 remove\_gap\_column() (pym2sa.core.solution.MSASolution method), 8  
 remove\_gap\_from\_sequence() (pym2sa.core.solution.MSASolution method), 8  
 remove\_gap\_group\_from\_sequence\_at\_column() (pym2sa.core.solution.MSASolution method), 8  
 reproduction() (in module pym2sa.algorithm.multiobjective.dnsgaii), 6  
 run() (pym2sa.algorithm.multiobjective.dnsgaii.dNSGAI method), 5

## S

ShiftClosedGapGroups (class in pym2sa.operator.mutation), 9  
 ShiftGapGroup (class in pym2sa.operator.mutation), 9  
 split\_gap\_column() (pym2sa.core.solution.MSASolution method), 8  
 SPXMSA (class in pym2sa.operator.crossover), 8  
 SubmitEvaluator (class in pym2sa.component.evaluator), 6

## T

ThreadPoolEvaluator (class in pym2sa.component.evaluator), 6  
 to\_html() (pym2sa.util.graphic.MSAPlot method), 11  
 TwoRandomAdjacentGapGroup (class in pym2sa.operator.mutation), 10

## U

update() (pym2sa.component.observer.WriteSequencesToFileObserver method), 6

update\_progress() (pym2sa.algorithm.multiobjective.dnsgaii.dNSGAI method), 5

## W

WriteSequencesToFileObserver (class in pym2sa.component.observer), 6