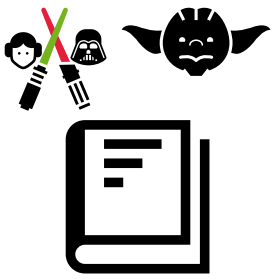
`galactic` user guide

The Galactic Organization <contact@thegalactic.org>



0.0.8

Contents

List of Figures

1 Introduction

The `py-galactic-bin` project includes several applications using the `py-galactic-core` library .

- `galactic-laser` is able to draw the Hasse diagram of a concept lattice using a data file and an optional explorer file.

2 Applications

2.1 The `galactic-laser-bin` tool

`galactic-laser-bin` is the console application able to draw the Hasse diagram of a concept lattice.

You can check the available options using:

```
$ galactic-laser-bin -h
```

or

```
$ galactic-laser-bin --help
```

```
$ galactic-laser-bin -h
usage: galactic-laser-bin [-h] [--explorer EXPLORER] [--mode {compact,full}]
```

[1]

```
                      [--join-irreducible-color JOIN-IRREDUCIBLE-COLOR]
                      [--meet-irreducible-color MEET-IRREDUCIBLE-COLOR]
                      [--log]
                      DATA [OUTPUT]

Create Hasse diagram image file from a data file.

positional arguments:
  DATA                  data file using format: .yaml, .yml, .toml, .txt,
                        .slf, .json, .ini, .dat, .csv, .cxt
  OUTPUT                Hasse diagram image file in .pdf, .svg, .png, .jpg,
                        .jpeg or .dot format

optional arguments:
  -h, --help            show this help message and exit
  --explorer EXPLORER   explorer in .yaml format
  --mode {compact,full}
                        display individuals in compact or full form
  --join-irreducible-color JOIN-IRREDUCIBLE-COLOR
                        graphviz color for join irreducible concept (see
                        http://graphviz.org/doc/info/colors.html) or use
                        '#colorcode' from '#000000' 'to '#FFFFFF'
  --meet-irreducible-color MEET-IRREDUCIBLE-COLOR
                        graphviz color for meet irreducible concept (see
                        http://graphviz.org/doc/info/colors.html) or use
                        '#colorcode' from '#000000' 'to '#FFFFFF'
  --log                 log operations

Do. Or do not. There is no try. - Yoda
```

You need to indicate a path for the image that will be generated and its extension.

Example:

```
$ galactic-laser-bin \
    --join-irreducible-color '#123456' \
    /data/animals.yaml \
    /data/animals.svg
```

2.2 The galactic-laser-ui tool

To launch the application, type in your terminal

```
$ galactic-laser-ui
```

The application window will open. Then you have to click on the data button.

Figure 1: galactic-laser data window

If nothing happens, check your installation of the *galactic* eco-system: Installation guide.

This action will open a dialog box, then you will have to look for a supported data file.

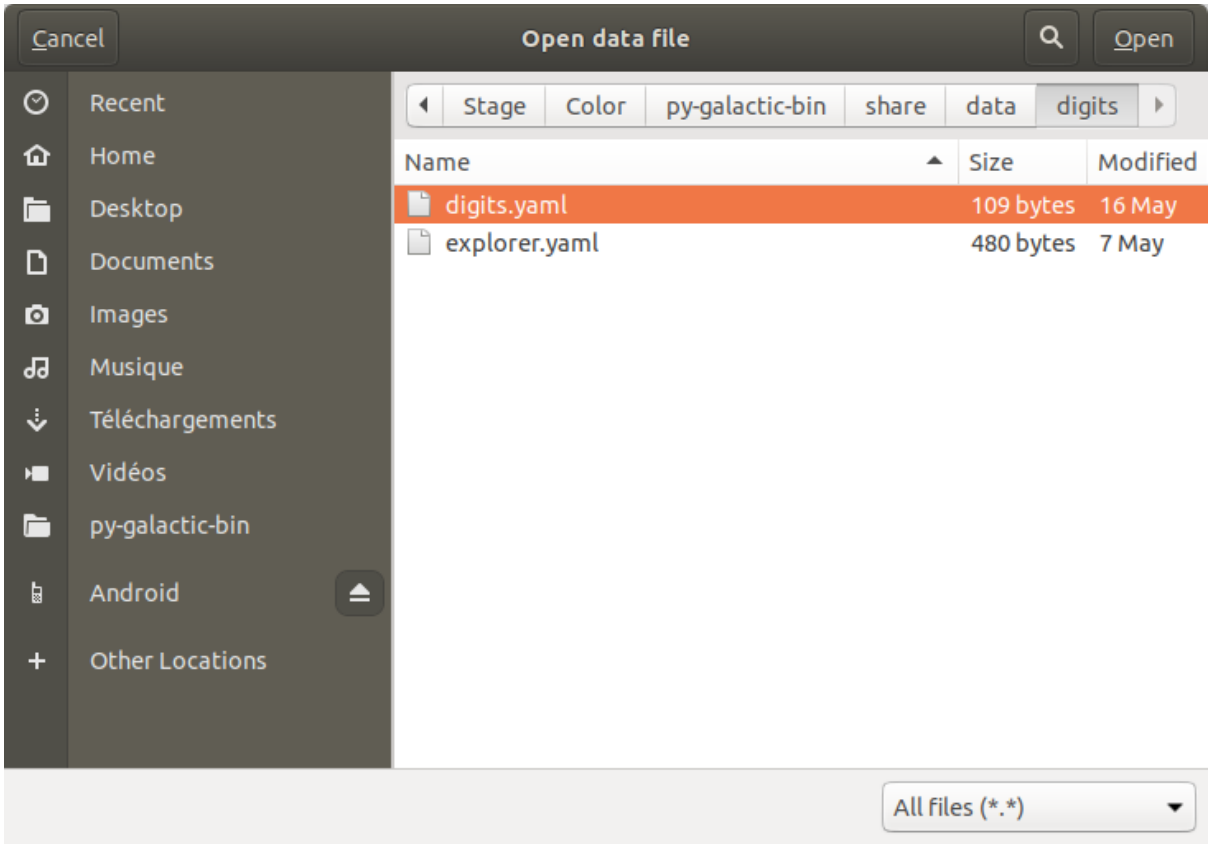


Figure 2: galactic-laser data dialog

Once the data file is chosen, you have several possible interactions.

It is possible to add an explorer file or the application will try to use a default behaviour.

You can change:

- the colors of the nodes for the current concept, the join-irreducible and the meet-irreducible;
- the node mode to the compact or complete nodes;
- the trace mode
- the draw mode.

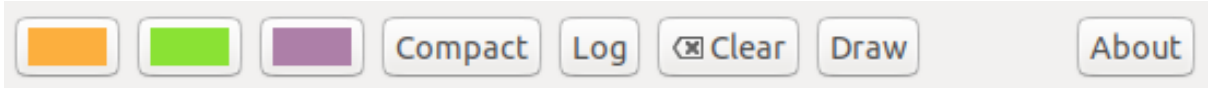


Figure 3: galactic-laser option chooser

To create the graph, just click on the 'Start' button.

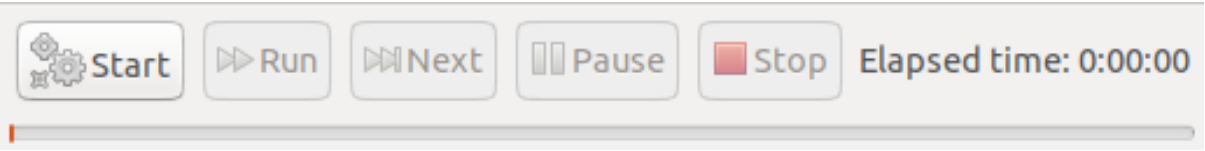


Figure 4: galactic-laser action buttons

Then you have two choices:

- click on the 'Next' button if you want to process step by step;
- click on the 'Run' button with the possibility to pause or stop.

If the drawing is just paused, there are possibilities to either continue or switch to the step-by-step mode.

If there is a press on the 'Stop' button the drawing will be stopped and you will have to start again from the beginning.

3 Data readers

Data readers are used by the different applications to read contexts. They are organized in plugins, so the *py-galactic-core* engine can be easily extended. For now on, here is the list of readers we have developed:

- *YAML* data reader
- *JSON* data reader
- *CSV* data reader
- *TOML* data reader
- *INI* data reader
- *TEXT* data reader
- *FIMI* data reader
- *Burmeister* data reader
- *SLF* data reader

Data reader	Boolean attributes	Heterogenous attributes	Complex attributes
YAML	yes	yes	yes
JSON	yes	yes	yes

Data reader	Boolean attributes	Heterogenous attributes	Complex attributes
CSV	yes	yes	no
TOML	yes	yes	no
INI	yes	yes	no
FIMI	yes	no	no
Text	yes	no	no
Burmeister	yes	no	no
SLF	yes	no	no

3.1 *YAML* data reader

py-galactic-data-reader-yaml is a YAML data reader plugin for *py-galactic*.

It uses the `yaml.load` function of the PyYAML library.

The file extension is `.yaml` or `.yml`. The individuals are represented either by:

- a list: individuals are named by an integer starting from 0;
- a dictionary: individuals are named by their keys.

For example:

```
0: [c, e, s]
1: [o, s]
2: [e, p]
3: [o, p]
4: [c, e, s]
5: [o, p]
6: [c, e]
7: [o, p]
8: [c, e]
9: [c, o, s]
```

3.2 *JSON* data reader

py-galactic-data-reader-json is a JSON data reader plugin for *py-galactic*.

The file extension is `.json`. The individuals are either represented by a list or by a dictionary. For example:

```
{
  "#1": {
    "name": "Galois",
    "firstname": "Évariste"
  },
  "#2": {
    "name": "Wille",
    "firstname": "Rudolf"
  }
}
```

This reader uses the `json.load` function of the python core library.

3.3 *CSV* data reader

py-galactic-data-reader-csv is a CSV data reader plugin for *py-galactic*.

The file extension is `.csv`. The first line of each column contains the names of the attributes. Each new line contains a new individual with its values. For example, an extraction of Fisher's IRIS data:

```
"sepal length","sepal width","petal length","petal width","class"
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica
6.5,3.0,5.8,2.2,Iris-virginica
```

This reader uses the `csv.DictReader` class of the python core library.

3.4 *TOML* data reader

py-galactic-data-reader-toml is a TOML data reader plugin for *py-galactic*.

The file extension is `.toml`. The individuals are represented in named sections by key = "value" pairs. For example:

```
[individual1]
    name = "Galois"
    firstname = "Évariste"
[individual2]
    name = "Wille"
    firstname = "Rudolf"
```

This reader uses the `toml.load` function of the toml library.

3.5 *INI* data reader

py-galactic-data-reader-ini is an INI data reader plugin for *py-galactic*.

The file extension is `.ini`. The individuals are represented in named sections by key = "value" pairs. For example:

```
[#1]
name = "Galois"
firstname = "Évariste"
[#2]
name = "Wille"
firstname = "Rudolf"
```

This reader uses the `configparser.ConfigParser` class of the python core library.

3.6 *TEXT* data reader

py-galactic-data-reader-text is a *TEXT* data reader plugin for *py-galactic*.

The file extension is `.txt`. There is a first section describing the observations and the attributes using the keywords `Observations` and `Attributes`, then each individual is described by a space separated list of attributes. For example:

```
Observations: 1 2 3 4
Attributes: a b c d e
1: a c
2: a b
```

```
3: b d e
4: c e
```

3.7 *FIMI* data reader

py-galactic-data-reader-fimi is a data reader plugin for *py-galactic*.

It uses the FIMI format.

The file extension is `.dat` and the file format consists in describing each individual in one line using a space separated list of attribute names. For example:

```
c e s
o s
e p
o p
c e s
o p
c e
o p
c e
c o s
```

3.8 *Burmeister* data reader

py-galactic-data-reader-burmeister is a data reader plugin for *py-galactic*.

The *Burmeister* context format uses the extension `.cxt`.

Each file is structured as follows:

- the first line consists of a single B;
- the second line is an empty line (its content is ignored);
- the third and fourth lines contain the object and attribute count;
- the fifth line is an empty line (its content is ignored);
- after that, all objects and all attributes are listed, each on a line
- finally, the context is given as a combination of `.` and X, each row on its own line.

```
B

10
5
```

```
0
1
2
3
4
5
6
7
8
9
c
e
o
p
s
XX..X
..X.X
.X.X.
..XX.
XX..X
..XX.
XX...
..XX.
XX...
X.X.X
```

3.9 *SLF* data reader

py-galactic-data-reader-slf is a data reader plugin for *py-galactic*.

The file extension is `.slf`. The internal format is composed of 4 sections:

- the first section starts with a header containing the keyword `Lattice` surrounded by square brackets. It is followed by two lines containing the number of objects and the number of attributes;
- the second section starts with the keyword `Objects` surrounded by square brackets. It is followed by lines containing the names of the objects;
- the third section starts with the keyword `Attributes` surrounded by square brackets. It is followed by lines containing the names of the attributes;

- the fourth section starts with the keyword `Relation` surrounded by square brackets. It is followed by lines containing the binary relation expressed as a matrix composed of 0 and 1.

For example:

```
[Lattice]
10
5
[Objects]
0
1
2
3
4
5
6
7
8
9
[Attributes]
c
e
o
p
s
[Relation]
1 1 0 0 1
0 0 1 0 1
0 1 0 1 0
0 0 1 1 0
1 1 0 0 1
0 0 1 1 0
1 1 0 0 0
0 0 1 1 0
1 1 0 0 0
1 0 1 0 1
```

4 Explorers

Explorer files describe how a data file will be processed using a set of strategies. It's a file using the `yaml` format.

```
# ...
- !strategy.categorized.basic.Category
  # parameters of the strategy
- !strategy.numerical.basic.Normal
  # parameters of the strategy
# ...
```

Each tagged element is described using a dictionary containing:

- `arguments`: a list of arguments;
- `params`: a list of named parameters.

If the parameters of the element are only a list of arguments or a list of named parameters, you can use directly that list without the use of the ('`arguments`', '`params`') notation.

The components supported by the *py-galactic-core* engine comes either from the core library or from plugins.

4.1 Attributes

4.1.1 *core* attributes

The *core* library exposes 3 attribute classes known to the console utilities by the following names:

- `attribute.core.Key` is used to describe an attribute for a python object using array-like access;
- `attribute.core.Property` is used to describe an attribute for a python object using property-like access;
- `attribute.core.Member` is used to describe an attribute for a python object using membership-like access.

In the following table, the prefix `attribute.core.` must be added to the names.

Name	Keyword parameters
Key	• name: to specify the attribute name • attribute: to specify an internal attribute
Property	• name: to specify the attribute name • attribute: to specify an internal attribute
Member	• name: to specify the member name • attribute: to specify an internal attribute

For example:

```
!attribute.core.Member
  params:
    name: c
```

describes a membership-like attribute named c.

For example:

```
!attribute.core.Member
  attribute:
    !attribute.core.key
      name: v
  params:
    name: c
```

describe a membership-like attribute named c accessible from the key-like parameter v. Thus, for the yaml data file:

```
1:
  v: [a,c]
  w: 2.3
2:
  v: [a,b]
  w: 1.2
3:
```

```
  v: [b,c]
  w: -5.6
```

The values of this attribute for the individuals 1, 2 and 3 will be `True`, `False` and `True`.

4.1.2 *logical* attributes

py-galactic-attribute-logical is a plugin package for *py-galactic*.

It exposes 6 attribute classes known to the console utilities by the following names:

- `attribute.logical.Not`: to get the *negation* of another boolean attribute;
- `attribute.logical.And`: to get a logical *and* between other boolean attributes;
- `attribute.logical.Or`: to get a logical *or* between other boolean attributes;
- `attribute.logical.Equivalence`: to get a logical *equivalence* between other boolean attributes;
- `attribute.logical.Nor`: to get a logical *nor* between other boolean attributes;
- `attribute.logical.Nand`: to get a logical *nand* between other boolean attributes.

In the following table, the prefix `attribute.logical.` must be added to the names.

Name	Arguments
`Not`	an attribute to negate
`And`	a list of boolean attributes
`Or`	a list of boolean attributes
`Equivalence`	a list of boolean attributes
`Nor`	a list of boolean attributes
`Nand`	a list of boolean attributes

For example, to use the *negation* of a boolean attribute stored in a key membership attribute named "`prime`", the following notation must be used in the *yaml* explorer file:

```
!attribute.logical.Not
- !attribute.core.Member
  params:
    name: "prime"
```

For example, to use the logical *and* of two boolean attributes stored in key membership attribute named "`prime`" and "odd", the following notation must be used in the *yaml* explorer file:

```
!attribute.logical.And
- attribute: !attribute.core.Member
    params:
      name: "prime"
- attribute: !attribute.core.Member
    params:
      name: "odd"
```

This plugin defines 2 description spaces for boolean attributes:

- the classical one defined on one boolean attribute producing either the attribute itself or none attribute;
- the *logical* one defined on multiple boolean attributes producing clauses using attributes and their negations.

4.1.3 *numerical* attributes

py-galactic-attribute-numerical is a plugin package for *py-galactic*.

It exposes 5 attribute classes known to the console utilities by the following names:

- `attribute.numerical.Number`: to convert an attribute to a number;
- `attribute.numerical.Linear`: to apply a linear transformation to a number;
- `attribute.numerical.Positive`: to test if an attribute is positive;
- `attribute.numerical.LowerLimit`: to test if an attribute is greater than a lower limit;
- `attribute.numerical.UpperLimit`: to test if an attribute is smaller than an upper limit.

In the following table, the prefix `attribute.numerical.` must be added to the names.

Name	Keyword parameters
`Number`	• `attribute`: to specify the attribute to convert
`Linear`	• `attribute`: to specify the attribute to linearize • `coefficient`: the coefficient of linearization
`Positive`	• `attribute`: to specify the attribute to test

Name	Keyword parameters
LowerLimit	• attribute: to specify the attribute to test • limit: to specify the limit • strict: to specify the test type
UpperLimit	• attribute: to specify the attribute to test • limit: to specify the limit • strict: to specify the test type

For example, to use a numerical attribute stored in a key attribute named "sepal length", the following notation must be used in the *yaml* explorer file:

```
!attribute.numerical.Number
params:
  attribute: !attribute.core.Key
    params:
      name: "sepal length"
```

This plugin defines 1 description space for numerical attributes producing the equations of the convex hull defined by the individuals.

4.1.4 *categorized* attributes

py-galactic-attribute-categorized is a plugin package for *py-galactic*.

It exposes 3 attribute classes known to the console utilities by the following names:

- attribute.categorized.Category: to specify a simple category attribute;
- attribute.categorized.SubSet: to specify a subset attribute;
- attribute.categorized.SuperSet: to specify a superset attribute.

In the following table, the prefix attribute.categorized. must be added to the names.

Name	Keyword parameters
Category	• attribute: to specify the attribute containing the category • values: to specify the values admitted by this category
SubSet	• attribute: to specify the attribute containing the subset • values: to specify the values admitted by this subset
SuperSet	• attribute: to specify the attribute containing the superset • values: to specify the values admitted by this superset

For example, to use the category attribute stored in a key attribute named "class" and to compare it with the admitted values "Iris-setosa", "Iris-versicolor", the following notation must be used in the *yaml* explorer file:

```
!attribute.categorized.Category
params:
  attribute: !attribute.core.Key
    params:
      name: "class"
  values: ["Iris-setosa", "Iris-versicolor"]
```

4.1.5 *chain* attributes

py-galactic-attribute-chain is a plugin package for *py-galactic*.

This plugin defines 3 description space for chain attributes producing the regular expressions defined by the individuals.

It exposes 2 attribute classes known to the console utilities by the following names:

- attribute.chain.SimpleMatch: to specify a simple chain attribute;
- attribute.chain.DistanceMatch: to specify a chain attribute with match cardinality and distances.

In the following table, the prefix attribute.chain. must be added to the names.

Name	Keyword parameters
`SimpleMatch`	• `attribute`: to specify the attribute that hold the chain • `chain`: to specify the chain of this attribute
`DistanceMatch`	* `attribute`: to specify the attribute that hold the chain * `chain`: to specify the chain of this attribute * `cardinality_min` to specify the minimal number of match with this chain * `cardinality_max` to specify the maximal number of match with this chain * `list_distances` contains a set of minimal and maximal distances between the elements of the chain

For example, to use the chain attribute stored in a key attribute named "`sequence`" and to match it with the chain "`chaintest`", the following notation must be used in the *yaml* explorer file:

```
!attribute.chain.SimpleMatch
params:
  attribute: !attribute.core.Key
    params:
      name: "sequence"
  chain: "chaintest"
```

4.2 Strategies

Strategies are used by the *galactic* core engine to produce new boolean attributes based on the individuals to partition. These attributes are in fact predicates defined on a description space.

4.2.1 *core* strategies

There are 2 strategies defined in the *core* library:

- `strategy.core.LimitFilter` which limits the successors to those whose measure does not exceed a limit;
- `strategy.core.SelectionFilter` which select the best or the worst successors.

They are a kind of filter strategy. They have to use a measure.

In the following tables, the prefix `strategy.core.` must be added to the names.

Name	Arguments
`LimitFilter`	a list of strategies
`SelectionFilter`	a list of strategies

Name	Keyword parameter
`LimitFilter`	• `measure`: the measure used to select the strategies • `limit`: the limit not to be exceeded • `strict`: is the limit strict? • `lower`: is the limit a lower limit or an upper limit?
`SelectionFilter`	• `measure`: the measure used to select the strategies • `keep`: the number of best strategies (1 allows to keep only the very best ones, 2 allows to keep only the best and those whose measure is the second value, …) • `maximize`: keep the best strategies or the worst • `decrease`: force the measure to increase (or decrease) • `strict`: decrease or increase strictly • `rate`: rate applied to the predecessor measure to force the decreasing (or increasing)

For example, to apply the '`SelectionFilter`' over the quantile numerical strategy for a numerical attribute named '`sepal width`', the following notation must be used in the *yaml* explorer file:

```
!strategy.core.SelectionFilter
arguments:
  - !strategy.numerical.quantile.Quantile
    arguments:
      - !attribute.numerical.Number
        attribute: !attribute.core.Key
```

```
          name: "sepal width"
    params:
      quantile: null
      lower: true
      count: null
params:
  measure: !measure.entropy.Entropy
    attribute: !attribute.core.Key
      name: "class"
  maximize: false
```

4.2.2 *basic logical* strategies

py-galactic-strategy-logical-basic is a plugin package for *py-galactic*.

It exposes 2 strategy classes known to the console utilities by the following names:

- `strategy.logical.basic.Boolean`: to specify a simple boolean strategy;
- `strategy.logical.basic.Dual`: to specify a dual strategy dealing with attributes and their negations.

In the following table, the prefix `strategy.logical.basic.` must be added to the names.

Name	Arguments
`Boolean`	an attribute to be used as a possible selector
`Dual`	a list of attributes to be used either as positive or negative

For example, to use the basic logical boolean strategy for a boolean attribute name "`prime`", the following notation must be used in the *yaml* explorer file:

```
!strategy.logical.basic.Boolean
arguments:
  - !attribute.core.Member
    params:
      name: prime
```

4.2.3 *basic numerical* strategies

py-galactic-strategy-numerical-basic is a plugin package for *py-galactic*.

It exposes 1 strategy class known to the console utilities by the following name:

- `strategy.numerical.basic.Normal`: to specify a simple numerical strategy.

In the following tables, the prefix `strategy.numerical.basic.` must be added to the names.

Name	Arguments
`Normal`	a list of numerical attributes

Name	Keyword parameter
`Normal`	• `coefficient`: the multiplier coefficient of the standard deviation

For example, to use the basic numerical strategy for a couple of numerical attributes named "`sepal length`" and "`sepal width`", the following notation must be used in the *yaml* explorer file:

```
!strategy.numerical.basic.Normal
arguments:
  - !attribute.numerical.Number
    params:
      attribute: !attribute.core.Key
        params:
          name: "sepal length"
  - !attribute.numerical.Number
    params:
      attribute: !attribute.core.Key
        params:
          name: "sepal width"
params:
  coefficient: 1
```

This strategy projects the 2-D points on the main axis of inertia and try to cut the individuals using 2 limits equal to the average plus or minus the standard deviation multiplied by the coefficient.

> Note
> This strategy can currently deal with only 2 numerical attributes.

4.2.4 *quantile* numerical strategies

py-galactic-strategy-numerical-quantile is a plugin package for *py-galactic*.

It exposes 1 strategy class known to the console utilities by the following name:

- `strategy.numerical.quantile.Quantile`: to specify a numerical strategy using quantiles to cut the individuals.

In the following tables, the prefix `strategy.numerical.quantile.` must be added to the names.

Name	Arguments
`Quantile`	a list of 1 numerical attribute

Name	Keyword parameter
`Quantile`	• `quantile`: the number of divisions of the individuals • `cut`: the location of the cut in the divisions

For example, to use the quantile strategy with 4 divisions for one numerical attribute named "`sepal width`" with a cut set to the first and the last division, the following notation must be used in the *yaml* explorer file:

```
!strategy.numerical.quantile.Quantile
arguments:
  - !attribute.numerical.Number
    params:
      attribute: !attribute.core.Key
      params:
        name: "sepal width"
params:
  quantile: 4
```

> Note
> This strategy can currently deal with only 1 numerical attribute.

4.2.5 *basic categorized* strategies

py-galactic-strategy-categorized-basic is a plugin package for *py-galactic*.

It exposes 1 strategy class known to the console utilities by the following name:

- `strategy.categorized.basic.Category`: to specify a simple categorized strategy.

In the following tables, the prefix `strategy.categorized.basic.` must be added to the names.

Name	Arguments
`Category`	a list of 1 modal attribute

For example, to use the basic categorized strategy for a modal parameter named "`class`", the following notation must be used in the *yaml* explorer file:

```
!strategy.categorized.basic.Category
arguments:
  - !attribute.core.Key
    params:
      name: "class"
```

This strategy removes possible values for the defined modal attribute reducing the set of individuals.

> Note
> This strategy can currently deal with only 1 modal attribute.

4.2.6 *basic chain* strategies

py-galactic-strategy-chain-basic is a plugin package for *py-galactic*.

It exposes three strategies classes known to the console utilities by the following name:

- `strategy.chain.basic.SimpleMatch`: to specify a simple chain strategy.
- `strategy.chain.basic.CompleteMatch`: to specify a chain strategy using maximal subsequences.

- `strategy.chain.basic.DistanceMatch`: to specify a chain strategy that uses cardinality and distances between elements of the match.

In the following tables, the prefix `strategy.chain.basic.` must be added to the names.

Name	Arguments
`DistanceMatch`	• `attribute`: to specify the attribute that hold the chain • `length`: to specify the length of the match.

For example, to use the distance chain strategy for a modal parameter named "`class`", and for length of match 3, the following notation must be used in the *yaml* explorer file:

```
!strategy.chain.basic.DistanceMatch
arguments:
  - !attribute.core.Key
      name: "class"
params:
  length: 3
```

Note
This strategy can currently deal with only 1 modal attribute.

4.3 Measures

4.3.1 *core* measures

There are 3 measures in the *core* library:

- `measure.core.Cardinality` which computes the cardinality of the successor;
- `measure.core.PredecessorCardinality` which computes the cardinality of the predecessor;
- `measure.core.CardinalityRatio` which computes the cardinality ratio between the successor and the predecessor.

4.3.2 *entropy* measure

py-galactic-measure-entropy is a plugin package for *py-galactic*.

It exposes 1 measure class known to the console utilities by the following name:

- `measure.entropy.Entropy`: to compute the entropy of a successor.

In the following table, the prefix `measure.entropy.` must be added to the names.

Name	Keyword parameters
`Entropy`	• `attribute`: to specify the attribute containing the category • `alpha`: the float coefficient applied to individuals kept in the successor

For example, to use the entropy measure of a key attribute named "`class`" for an attribute named "sepal width" using the quantile strategy, the following notation must be used in the *yaml* explorer file:

```
!strategy.core.SelectionFilter
arguments:
  - !strategy.numerical.quantile.Quantile
    arguments:
      - !attribute.numerical.Number
        attribute: !attribute.core.Key
          name: "sepal width"
    params:
      quantile: null
      lower: true
      count: null
params:
  measure: !measure.entropy.Entropy
    attribute: !attribute.core.Key
      name: "class"
  best: false
```

Extension list

- `attribute.core.Key`
- `attribute.core.Property`

- attribute.core.Member
- measure.core.Support
- measure.core.PredecessorSupport
- measure.core.SupportRatio
- strategy.core.LimitFilter
- strategy.core.SelectionFilter
- attribute.logical.Not
- attribute.logical.And
- attribute.logical.Or
- attribute.logical.Equivalence
- attribute.logical.Nor
- attribute.logical.Nand
- attribute.numerical.Number
- attribute.numerical.Linear
- attribute.numerical.Positive
- attribute.numerical.LowerLimit
- attribute.numerical.UpperLimit
- attribute.categorized.Category
- attribute.categorized.SubSet
- attribute.categorized.SuperSet
- strategy.logical.basic.Boolean
- strategy.logical.basic.Dual
- strategy.numerical.basic.Normal
- strategy.numerical.quantile.Quantile
- strategy.categorized.basic.Category
- strategy.chain.basic.SimpleMatch
- strategy.chain.basic.CompleteMatch
- strategy.chain.basic.DistanceMatch
- measure.entropy.Entropy