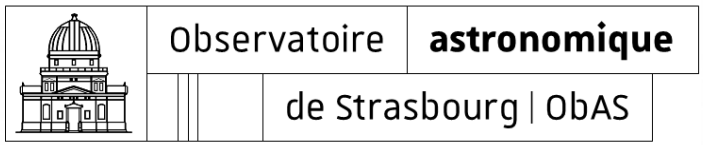
Recent and Future Developments in MOCpy

Matthieu Baumann (CDS), Thomas Boch[1]

[1]Centre de Données astronomiques de Strasbourg

14 May 2019



Summary

New developments tools

New Features

Future of MOCPy

General presentation

- MOCPy is a Python library allowing easy creation, parsing and manipulation of MOCs (Multi-Order Coverage maps)
  - On GitHub
  - Multi-platforms and works for Python 2 and 3
  - Has a few dependencies:
    1. astropy_healpix (BSD-3 clause HEALPix library)
    2. numpy
    3. matplotlib
    4. spherical-geometry
  - BSD-3 licensed
- Available through pip

```
pip install --upgrade mocpy
```

- Latest version: **v0.5.6**

New developments tools

Documentation

- reStructuredText files compiled to html static files using Sphinx.

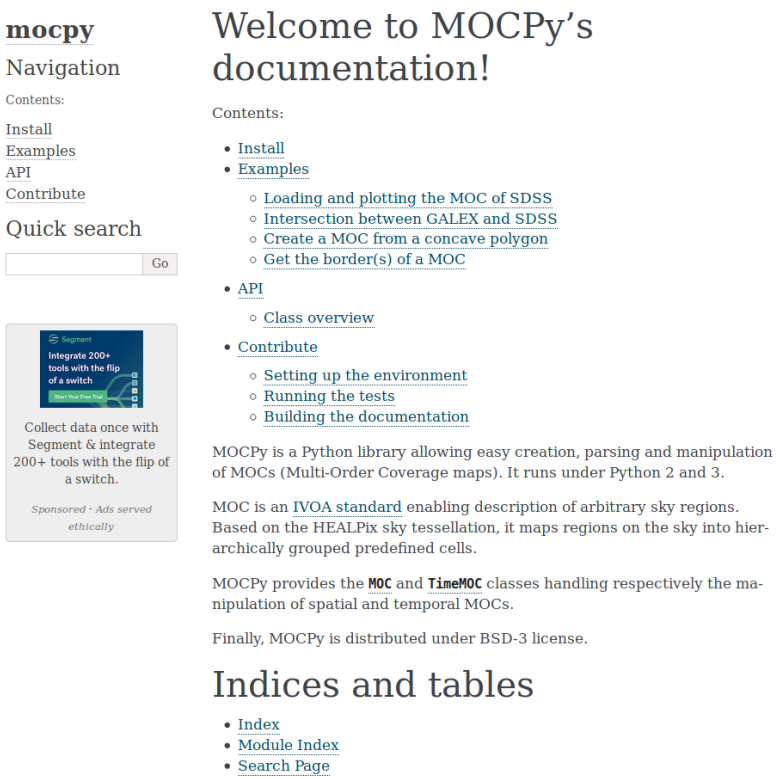


Figure 1: https://mocpy.readthedocs.io

Documentation...

- Sphinx extensions are convenient
  1. **autodoc**: Sphinx looks for API commentaries in the .py files, compiles them to html and bind the API doc to the html files coming from the .rst files

```
@classmethod                                                    [docs]
def from_lonlat(cls, lon, lat, max_norder):
    """
    Creates a MOC from astropy lon, lat `astropy.units.Quantity`.

    Parameters
    ----------
    lon : `astropy.units.Quantity`
        The longitudes of the sky coordinates belonging to the MOC.
    lat : `astropy.units.Quantity`
        The latitudes of the sky coordinates belonging to the MOC.
    max_norder : int
        The depth of the smallest HEALPix cells contained in the MOC.

    Returns
    -------
    result : `~mocpy.moc.MOC`
        The resulting MOC
    """
```

classmethod **from_lonlat**(*lon*, *lat*, *max_norder*) [source]

Creates a MOC from astropy lon, lat **astropy.units.Quantity**.

Parameters:

lon : ***astropy.units.Quantity***
The longitudes of the sky coordinates belonging to the MOC.

lat : ***astropy.units.Quantity***
The latitudes of the sky coordinates belonging to the MOC.

max_norder : *int*
The depth of the smallest HEALPix cells contained in the MOC.

Returns:

result : ***MOC***
The resulting MOC

Documentation...

2. **doctest**: Example code snippets can be written in the API doc commentaries and can be run with

```
make doctest
```

3. **matplotlib.sphinxext**: matplotlib has a Sphinx extension for executing portions of code and showing the image file next to the source code in the html generated files!

Testing

- Unit tests added making mocpy more robust to API and core changes
- **pytest**:
  1. Tests files are put in a **mocpy/tests** directory
  2. In the root run the tests with
     `python -m pytest mocpy`
  3. Unit tests are methods beginning with the name `test_*`

```
def test_union(moc1, moc2):
    assert moc1.union(moc2) == MOC.from_json({
        '0': [0, 1, 2, 3, 4, 5, 7]
    })
```

Testing...

4. Specific tests can be run with:

```
python -m pytest mocpy -k test_union
```

5. Several extensions:

   5.1 For benchmarking `pytest_benchmark`
   5.2 For running code coverage statistics `pytest-cov` (**91% code coverage** in mocpy)
   5.3 For profiling purposes `pytest-profiling`

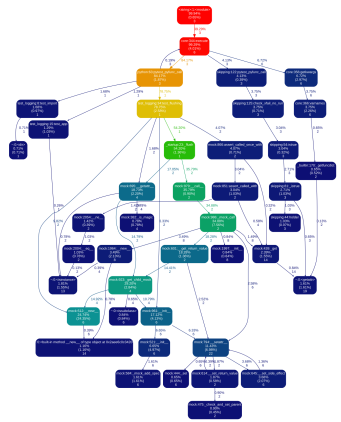


Figure 2: Result profiling SVG graph example

Continuous Integration

- At each new commit pushed, Travis-CI runs automatically a script:
  1. That clones the repo
  2. Makes a conda environnement that contains all the deps (e.g. for running the tests...) and activates it
  3. Runs the tests with pytest and prints the coverage stats
  4. Runs the notebook examples
  5. Builds the docs with Sphinx
  6. Runs the code examples in the doc API
  7. If the **previous steps passed** and the commit is **tagged** then a new version of MOCPy is deployed on the pip servers

New Features

New developments tools

New Features

- Plot MOC enhancement
- String (de)serialization
- Creating a MOC from a polygon

Future of MOCPy

Plotting MOC enhancement

- Two methods:
  1. `MOC.fill` draws the HEALPix cells of a MOC one by one
  2. `MOC.border` draws only the external border(s) of a MOC
- They accept a `matplotlib.axes.Axes`, an `astropy.wcs.WCS` and several matplotlib styling kwargs (linewidth, color, fill, ...)
- MOC.WCS is a new class that essentially wraps an `astropy.wcs.WCS`. It creates a WCS from:
  1. A center `astropy.coordinates.SkyCoord`
  2. A fov `astropy.coordinates.Quantity`
  3. A coordsys ('icrs' or 'gal')
  4. A rotation `astropy.coordinates.Angle`
  5. A projection type (all astropy supported projections)

Plot examples

```
from mocpy import MOC, WCS
from astropy.coordinates import Angle, SkyCoord
import astropy.units as u
# Plot the MOC using matplotlib
import matplotlib.pyplot as plt
fig = plt.figure(111, figsize=(10, 10))
# Define a astropy WCS easily
with WCS(fig,
 fov=150 * u.deg,
 center=SkyCoord(0, 0, unit='deg', frame='icrs'),
 coordsys="icrs",
 rotation=Angle(0, u.degree),
 projection="AIT") as wcs:
    ax = fig.add_subplot(1, 1, 1,
                         projection=wcs)
    galex.fill(ax=ax, wcs=wcs,
               alpha=0.5, fill=True,
               color="red", linewidth=0,
               label="GALEX")
    sdss.fill(ax=ax, wcs=wcs,
              alpha=0.5, fill=True,
              color="green", linewidth=0,
              label="SDSS9")

...
plt.show()
```

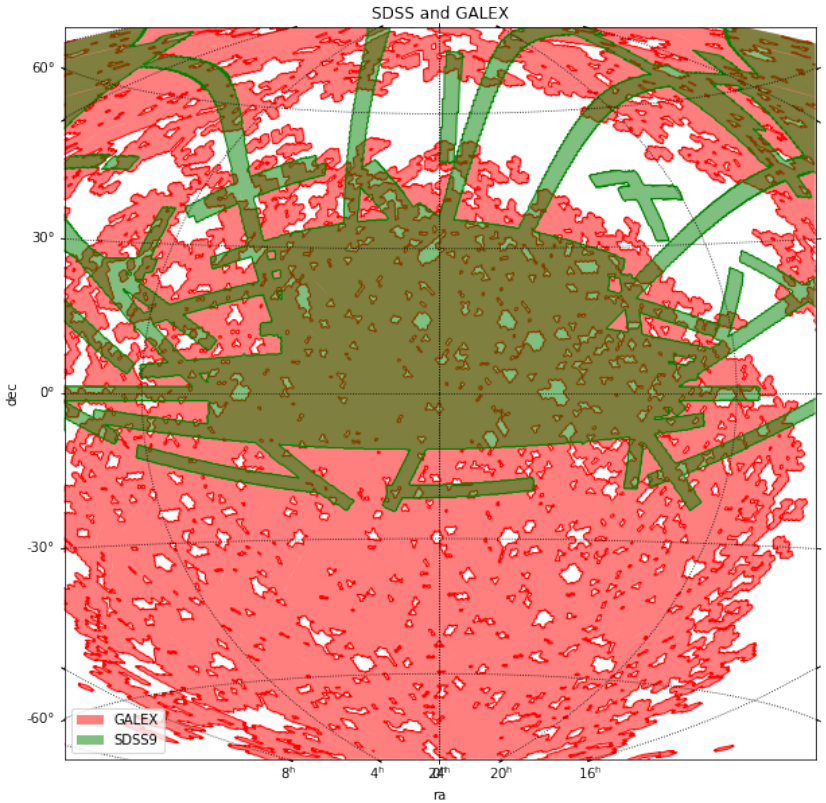


Figure 3: Rendered with MOCpy

String (de)serialization

Deserialization

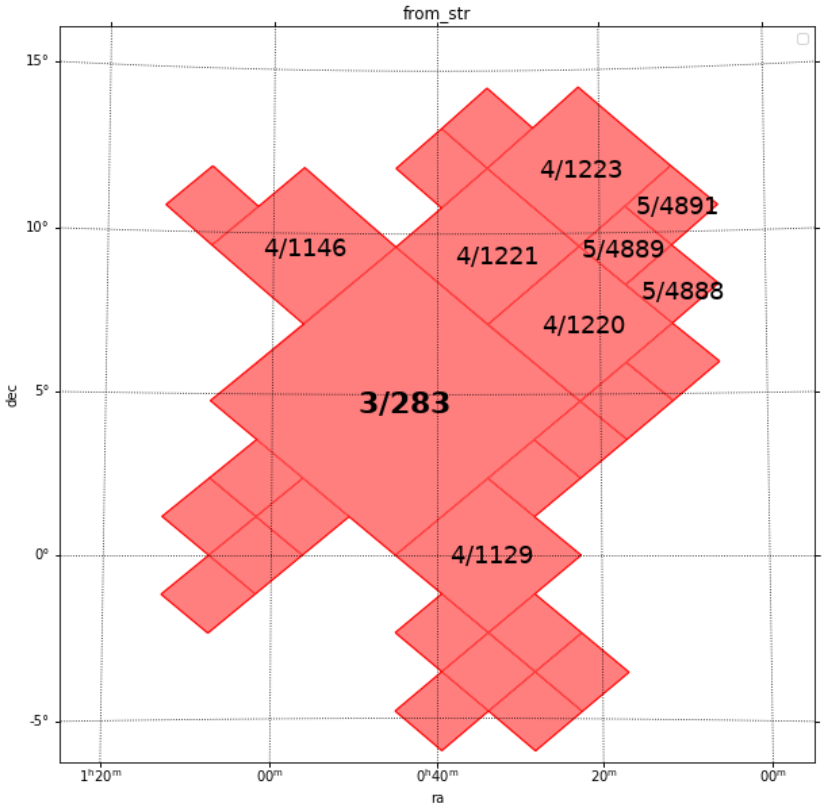
1. `MOC.from_str` takes a string following this EBNF grammar

```
moc      ::= ordpix (sep+ ordpix)*
ordpix   ::= int '/' sep* pixs
pixs     ::= pix (sep+ pix)*
pix      ::= int? | (int '-' int)
sep      ::= [ ,\n\r]
int      ::= [0-9]+
```

2. Use of lark-parser, a python library generating a parser from a grammar. The parser is generated the first time `MOC.from_str` is called
3. Submitting a string either:
   - raises an exception if the string does not match the grammar
   - or returns an AST that is then converted to a json format `{'depth': int[]}`
4. The json is passed to `MOC.from_json` and the resulting MOC is returned

String (de)serialization

- Examples




```
MOC.from_str(
'3/283 \
 4/1129,1146,1220-1221,1223 \
 5/4489-4491,4494,4499,4505, \
  4507-4508,4510,4512-4513, \
  4525,4527,4588,4869,4871, \
  4888-4889,4891,4930,4936'
)
```

String (de)serialization

Serialization

- Serialization: to string

```
moc_str = moc.serialize(format='str')
```

New MOC from a polygon

- `MOC.from_polygon` takes lon, lat `astropy.coordinates.Quantity` and a depth defining the maximum depth of the MOC
- Relies on spherical-geometry, a C-python library handling polygon intersections on the unit sphere.
- (lon, lat) must not define a self-intersecting polygon.

3. Algorithm:
   - 3.1 Begin with the 12 base cells in a queue
   - 3.2 We take one cell from the queue and remove it
   - 3.3 If the cell is not intersecting the polygon
     - 3.3.1 If it is outside, it is discarded
     - 3.3.2 If it is inside, it is added to the MOC
   - 3.4 If the cell intersects the polygon
     - 3.4.1 If the cell is at the max depth then it is added to the MOC
     - 3.4.2 If not, then it is divided in its 4 children. They are added to the queue and wait to be tested
   - 3.5 Loop over **3.2** to **3.4** until there is no more cells in the queue

Examples of from_polygon

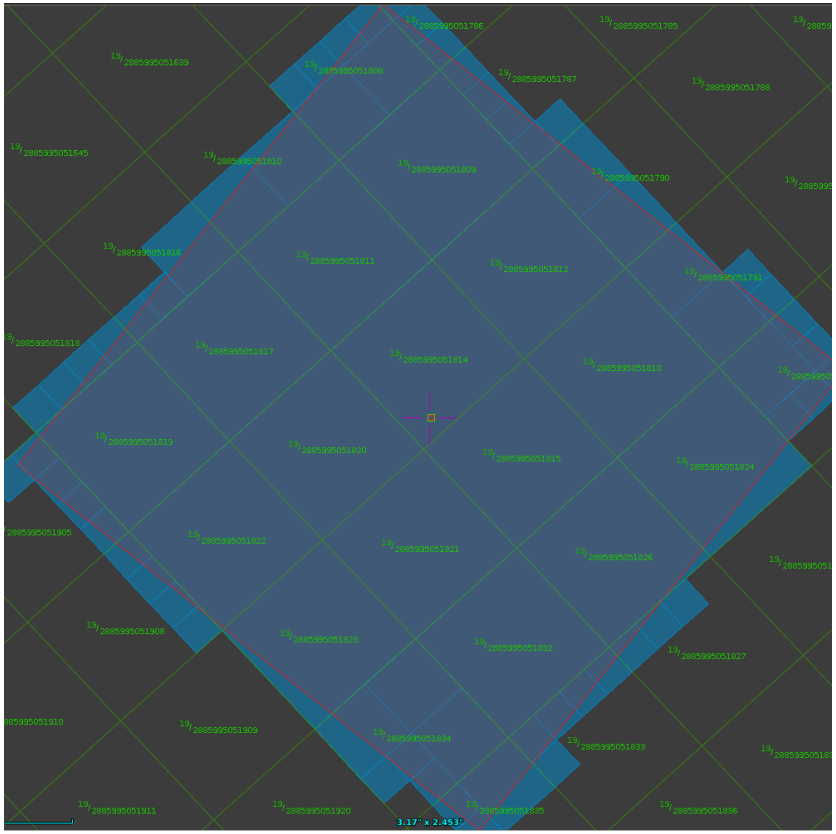


Figure 4: MOC from an HST window defined at the depth 21

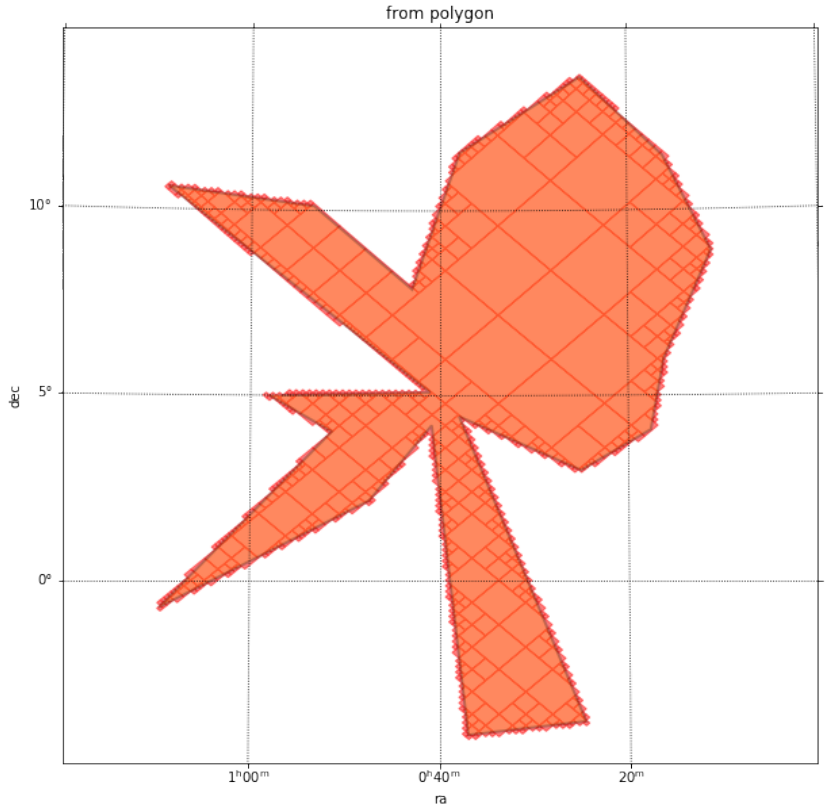


Figure 5: A MOC from a concave polygon on the unit sphere

Future of MOCPy

Future developments

- Replace `astropy_healpix` dependency with `cdshealpix`
- `cdshealpix`: **pip install cdshealpix**
  1. New python wrapper developped by the CDS (github & doc)
  2. Is a wrapper around the new Rust HEALPix library developped by F.-X. Pineau.
  3. Provides new features: **polygon**/cone and **elliptical** search.
  4. Has very good performance
     - 4.1 *lonlat_to_healpix* 10x faster than astropy_healpix
     - 4.2 *healpix_to_lonlat* 7x faster than astropy_healpix
     - 4.3 *vertices* (returns the position of the 4 vertices on the sky of a HEALPix cell) 13x faster
     - 4.4 *cone_search* 4x faster
- Make MOCPy an astropy affiliated package

Future developments

- Develop Rust extensions that will enhance the overall performance of the library
- Rust is a new system programming language released in 2015
  1. performant, safe and concurrent
  2. compiled, no garbage collector, strong static rules (e.g. borrow checker), generics, interfaces (i.e. Traits), no inheritance, type inference...
  3. open source, maintained/developped by Mozilla
- `from_lonlat`, `from_json`, `from_fits`, `degrade_to_depth`, `union`, `difference`, `intersection` **already ported** in Rust (See rust_ext branch)
- Some performance statistics:
  1. Creating a MOC from **4.8M positions** (`from_lonlat`) takes ~**200-300ms** (compared to ~**5-10sec** with the pure python `from_lonlat`).
  2. Loading the SDSS9 MOC (i.e. max depth: 11) now takes ~**15ms** compared to **450ms** from the pure python `from_fits`.

Questions ?

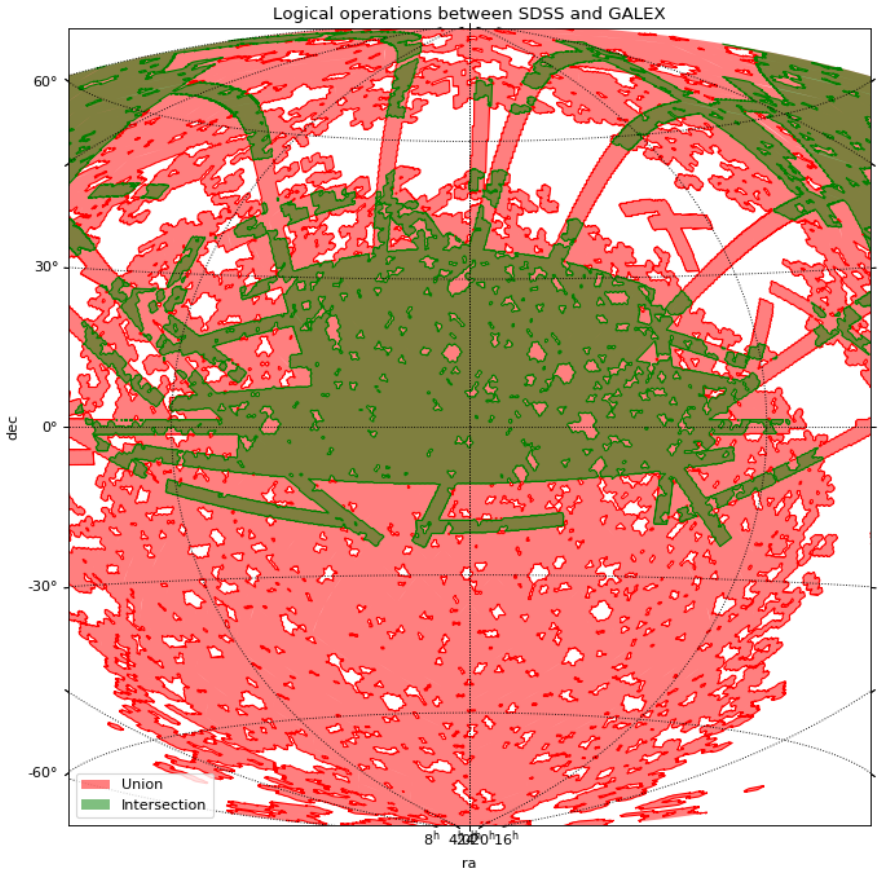


Figure 6: Rendered with MOCPy