Global Dataset Discovery in pyVO

Markus Demleitner

Herbsttagung der Astronomischen Gesellschaft, Köln, 2024-09-12

The Main API

The purpose is to enable a global search for datasets (for now: images) constrained only by space, time and spectrum:

```
images, log = discover.images_globally(
        space=(132, 14, 0.1),
        time=time.Time(58794.9, format="mjd"),
        spectrum=600*u.eV,
        inclusive=False)
```

images is a list of obscore-like metadata, and *log* has information on which services yielded how much (or how they failed or were skipped).

If True, *inclusive* makes the function ask services without coverage information and return SIAP1 images that have no usable information on time and spectrum.

A Two-Step Process

In the VO, global dataset discovery has two steps:

1. Locate services that *could* have relevant datasets
2. Send appropriate queries to each service discovered.

Extra trouble: In the VO, images can be published through any combination of:

- SIAPv1 – the old "simple" (i.e., atomic HTTP parameters) protocol for searching images
- SIAPv2 – a newer "simple" protocol for searching images
- Obscore – standard metadata tables queried via TAP

Challenge: Too Many Services

How many candidate services are there for images? Try:

```
import pyvo
print("#sia", len(pyvo.registry.search(servicetype="sia")))
print("#sia2", len(pyvo.registry.search(servicetype="sia2")))
print("#obscore", len(pyvo.registry.search(datamodel="obscore")))
```

This yields:

sia	263
sia2	106
obscore	38

Even if each service returned within a second, a sequential all-VO scan thus is more than five minutes. And that *if* is a strong if.

The Registry to the Rescue

An increasing number of VO resources define their coverage in space, time, and spectrum in the Registry. Using this, we can skip services that do not cover the user's region of interest.

However – and this is an ADQL query against RegTAP tables, executable, for instance, on the TAP service at http://dc.g-vo.org/tap):

```
SELECT COUNT(*) FROM
rr.stc_spatial
NATURAL JOIN rr.capability
WHERE standard_id LIKE 'ivo://ivoa.net/std/sia%'
```

At the moment, only 84 SIAP1/2 services declare their spatial coverage (29 for spectral, 43 for temporal; SSAP has 35 spatial coverages).

Part of the reason for that relatively meagre takeup is that the standard defining how to declare coverage is relatively recent (VODataService 1.2 from 2021-11-02). Another part is that basically, only Aladin has done something interesting with coverage so far, and then only with spatial coverage. I do hope that as people do more global discovery, service operators have a stronger incentive to define coverage.

The Obscore Problem

Obscore services are currently discovered as TAP services with the obscore data model.

Hence, the coverage – if given at all – is generally not useful: there can be a lot else in the TAP service.

This is un-recoverable. We need to fix Obscore registration.

Beyond "Obscore tables need to get resource records of their own", this is VO nerd stuff. If you are a VO nerd, see the TableReg note and comment: https://ivoa.net/documents/TableReg.

Dupes, Dupes, Dupes

Another big issue: Blindly querying everything will return many datasets multiple times.

First reason for that: Services having both SIAP1 and SIAP2 interfaces. How many are there?

```
SELECT COUNT(*) FROM
rr.capability AS a
JOIN rr.capability AS b
USING (ivoid)
WHERE
  a.standard_id='ivo://ivoa.net/std/sia'
  AND b.standard_id='ivo://ivoa.net/std/sia#query-2.0'
```

That's 24 at the moment; in the service selection, one can filter these out by preferring SIAP2. But…

Dupes from Obscore

At the GAVO data centre, all of its 20 SIA services are also reflected in its Obscore table (and a sitewide SIAP2 service, too). It would be a bad waste of resources to fire off the 20 extra requests.

Not to mention we would have lots of dupes.

Proposal (re-using auxiliary capabilities): SIAP(2) and SSAP records should include isServedBy relationships to Obscore, TAP, and sitewide SIAP2 services. Yes, it *is* a bit silly if a service record has an isServedBy relationship. But until convinced otherwise, I'll claim it's not grossly inadequate and does the job with minimal extra effort.

Running the Queries

SIAP2 and Obscore are easy: Just translate the constraints to queries and collect the rows you get back (slightly normalised).

SIAP1 is more difficult: no (generally usable) constraints on time and spectrum, so you need to filter locally if possible. Also: result rows need to be mapped to the Obscore DM.

Main problem, though: Dealing with hanging or dead services, timeouts, hanging reverse proxies...

Inclusive: Services

By default, discover will only consider resources that give coverage. A welcome side effect of that is that well-maintained resources will receive preferential treatment.

To consider services without declared coverage, pass `inclusive=True`.

At this point, this will *dramatically* increase runtime and the amount of brokenness you will see.

Inclusive: Datasets

In particular for SIA1, inclusive has a second effect: It will also return *datasets* that do not declare temporal or spectral coverage.

For SIA2, the decision whether to return datasets with unknown coverage is left to the service.

For Obscore, we *could* add clauses like `OR time_min IS NULL` for inclusive searches – but we don't at this point. Should we?

How to log

Reminder:

```
images, log = images_globally(...)
```

How machine-readable should *log* be?

It's an artefact of provenance at the very least: you need to know which services happened to be down when your discovery ran.

It is also potentially a debugging aid.

But if you log everything, it will be extremely painful to see anything in the log. Feedback on what I am doing right now is most welcome.

More API: Custom Service Lists

Perhaps you want to query a custom subset of services?

Well: Pass a `services` parameter, perhaps like this:

```
datasets, log = discover.images_globally(
  space=(274.6880, -13.7920, 1),
  services=registry.search(registry.Datamodel("obscore")))
```

discover will still do service elision on the service list you pass in, i.e., evaluate *isServedBy* relationships.

More API: Watching live

When you have something like a UI, or you want to break out of discovery, you want to get notifications of what is going on.

Define a `callback(discoverer, msg)` that is called each time something happens; you can muck around in the Discoverer instance in the first argument:

```
def say(discoverer, s):
        print(s)

datasets, log = discover.images_globally(
  time=(time.Time(’1995-01-01’), time.Time(’1995-12-31’)),
  watcher=say)
```

OMG Testing

Writing unit tests for code of this kind is a nightmare: We basically need to mock a major part of the VO.

What did not work: Record requests and responses on the http level to build the mock environment. PyVO and python produce different requests in different versions.

What I do: Some small tests with remote data. Yuck.

Demo Time: API

```
def say(disco, msg):
    print(process_time(), msg)

images, logs = discover.images_globally(
    space=(134, 11, 0.1),
    spectrum=600*u.eV,
    time=(time.Time(’1990-01-01’), time.Time(’1999-12-31’)),
    watcher=say)

print("============ Result ===========")
print(logs)
for res in images:
print(res.access_estsize, res.access_url)
```

Demo Time: A Simple GUI

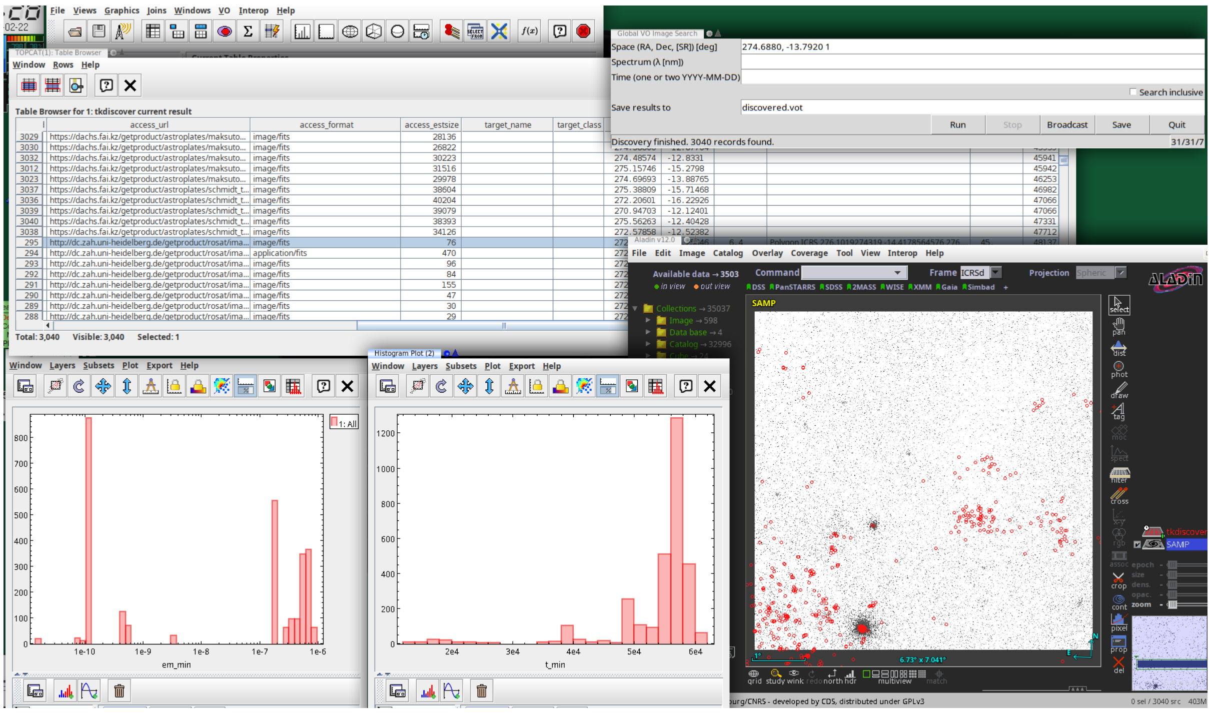
Table Browser for 1: tkdiscover current result

	access_url	access_format	access_estsize	target_name	target_class
3029	https://dachs.fai.kz/getproduct/astroplates/maksuto...	image/fits	28136		
3030	https://dachs.fai.kz/getproduct/astroplates/maksuto...	image/fits	26822		
3032	https://dachs.fai.kz/getproduct/astroplates/maksuto...	image/fits	30223		
3012	https://dachs.fai.kz/getproduct/astroplates/maksuto...	image/fits	31516		
3023	https://dachs.fai.kz/getproduct/astroplates/maksuto...	image/fits	29978		
3037	https://dachs.fai.kz/getproduct/astroplates/schmidt_t...	image/fits	38604		
3036	https://dachs.fai.kz/getproduct/astroplates/schmidt_t...	image/fits	40204		
3039	https://dachs.fai.kz/getproduct/astroplates/schmidt_t...	image/fits	39079		
3040	https://dachs.fai.kz/getproduct/astroplates/schmidt_t...	image/fits	38393		
3038	https://dachs.fai.kz/getproduct/astroplates/schmidt_t...	image/fits	34126		
295	http://dc.zah.uni-heidelberg.de/getproduct/rosat/ima...	image/fits	76		
294	http://dc.zah.uni-heidelberg.de/getproduct/rosat/ima...	application/fits	470		
293	http://dc.zah.uni-heidelberg.de/getproduct/rosat/ima...	image/fits	96		
292	http://dc.zah.uni-heidelberg.de/getproduct/rosat/ima...	image/fits	84		
291	http://dc.zah.uni-heidelberg.de/getproduct/rosat/ima...	image/fits	155		
290	http://dc.zah.uni-heidelberg.de/getproduct/rosat/ima...	image/fits	47		
289	http://dc.zah.uni-heidelberg.de/getproduct/rosat/ima...	image/fits	30		
288	http://dc.zah.uni-heidelberg.de/getproduct/rosat/ima...	image/fits	29		

Total: 3,040 Visible: 3,040 Selected: 1

Global VO Image Search

Space (RA, Dec, [SR]) [deg]: 274.6880, -13.7920 1
Spectrum (λ [nm]):
Time (one or two YYYY-MM-DD):
Search inclusive
Save results to: discovered.vot

Discovery finished. 3040 records found. 31/31/7

Find the source code of the Tkinter-based program on github: https://github.com/ivoa/tkdiscover

For the demo, I first ran with the default parameters, yielding two ROSAT images; then, I changed λ to 400 nm, which does not change much (because too few services give their spectral coverage). Checking "inclusive" changes the picture; show people the progress log in the lower right corner, let it run for a while and then broadcast the result to TOPCAT. Show the discovery log in the global table metadata (discovery_log).

Future Directions

Once the basics are there, here's some extensions I could see:

- Registry work with data providers to make this faster and more reliable
- Allow RoI geometries (polygons, perhaps even MOCs), intervals for scalars
- Enable object lists for upload (but: that will only work for Obscore)
- Optionally, automatic cutouts to the RoI using SODA?
- Non-remote testing by mocking and recording on the XService level?

Conclusion

Everything is hard (at least at first) in a global, distributed system.

Please do join me in this effort!

See also my blog post on this: https://blog.g-vo.org/global-dataset-discovery-in-pyvo.html.