Caspailleur

Lightweight Python package for Formal Concept Analysis

https://github.com/smartFCA/caspailleur

Egor Dudyrev, ConSoft workshop @ CONCEPTS'25, Cluj-Napoca, Romania

Outline

- What Caspailleur can do
- Approaches for faster computation
- Conclusion

Before Caspailleur

During 2020-2022 I developed FCA.

I embraced OOP and made so many classes that the system became hard to work with.

README | Contributing | GPL-3.0 license

FCApy

pypi v0.1.4.5 | build repo or workflow not found | docs failing | coverage 95% | license GPL-3.0

A python package to work with Formal Concept Analysis (FCA).

> **Note**
>
> The development of `FCApy` is paused since 2023. Check out `caspailleur` package for mining formal concepts and implications. And check out `paspailleur` package for mining pattern concepts and implications. Tutorials for both packages are presented in `expailleur` repository.
>
> The current `FCApy` package can be used for visualising concept lattices and ordered sets.

Install

FCApy can be installed from PyPI:

```
pip install fcapy
```

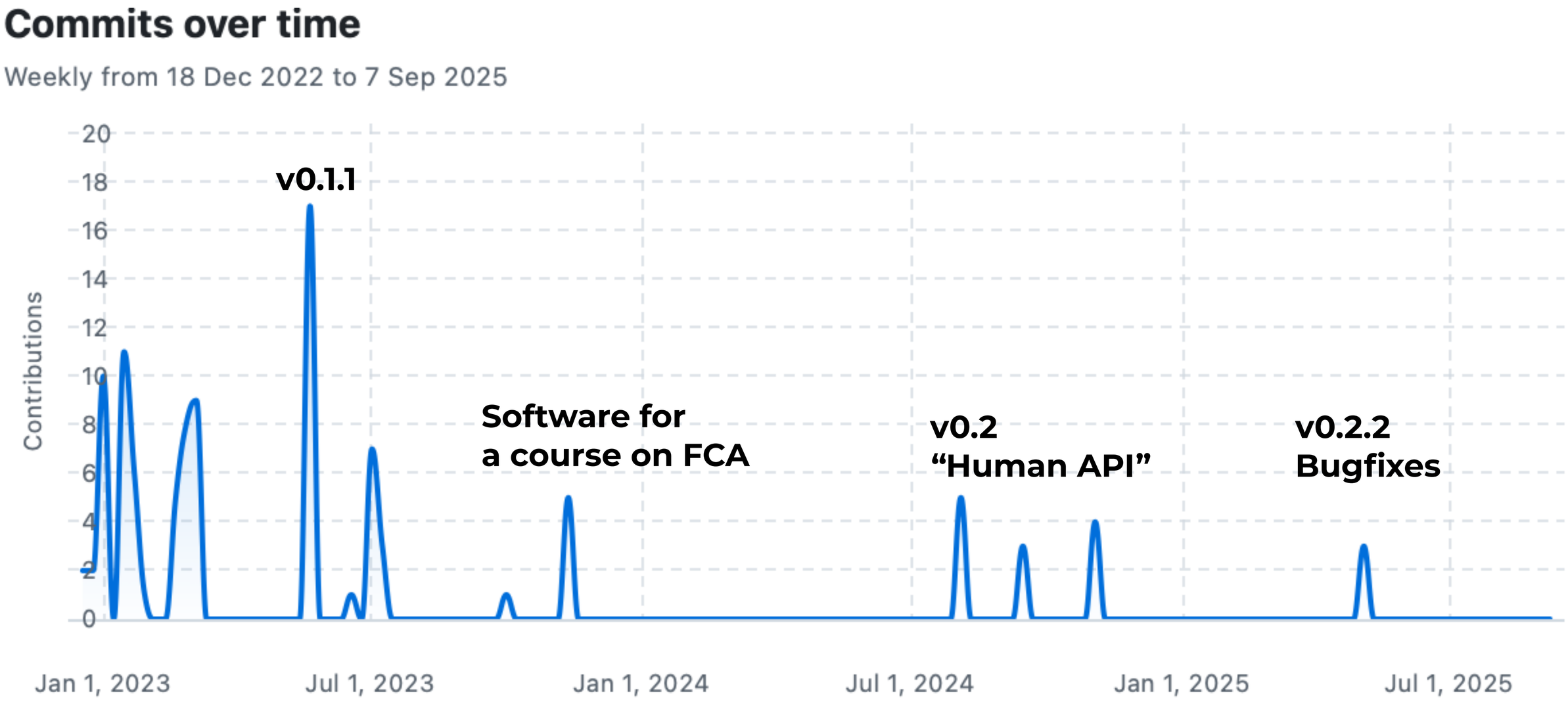
Gentle Intro to Formal Concept Analysis

Formal Concept Analysis (FCA) is a mathematical framework aimed at simplifying the data analysis.

To achieve so, FCA introduces a **concept lattice**: a hierarchical representation of the dataset. A concept lattice can be visualized in an appealing tree-like manner, while keeping all the dependencies of the corresponding binary dataset.

The following Figure compares the tabular, Formal Context-based data representation (on the left), with the hierarchical, Concept Lattice-based data representation on the right. Both representations describe the same "Live in water" dataset. But the right subfigure also unravels the dichotomy between the ones who "can move" (i.e. animals) and the ones who "needs chlorophyll" (i.e. plants).

Project timeline



Current version

https://github.com/smartFCA/caspailleur

README GPL-3.0 license

pypi v0.2.2 | build passing | license GPL-3.0 | Made in LORIA | Funded by SmartFCA

Caspailleur is a python package for mining concepts and implications in binary data with FCA framework. Part of SmartFCA ANR project.

Get started

The stable version of the package can be installed from PyPI with:

```
pip install caspailleur
```

and the latest version of the package can be installed from GitHub repository:

```
pip install caspailleur@git+https://github.com/smartFCA/caspailleur
```

Analysis example

Glossary

The field of Formal Concept Analysis has many mathematical terms and some conflicting notation traditions. Here is the glossary used throughout the `caspailleur` package: Glossary.md.

What can do

Data Mining

- Mine Concepts
- Mine Implications
- Mine Descriptions (all of them)
- Order Concepts

Visualisations? No

Data Preprocessing

- Input values:
  - Itemsets
  - Formal context
  - Boolean matrix
  - Dictionary
  - Binary dataframe
- Import from FCA repo
- Save/load to/from .cxt

Data Mining

Concepts and their generators

- Closed descriptions
  - LCM algo (via scikit-mine)
  - gSofia algo (simplified one)
- Keys (minimal generators)
  - Custom “Carpathia-G”-like algo for a lattice of intents

Implications

- Canonical Direct basis
  - (quite efficient)
- Canonical bases
  - (not really efficient)

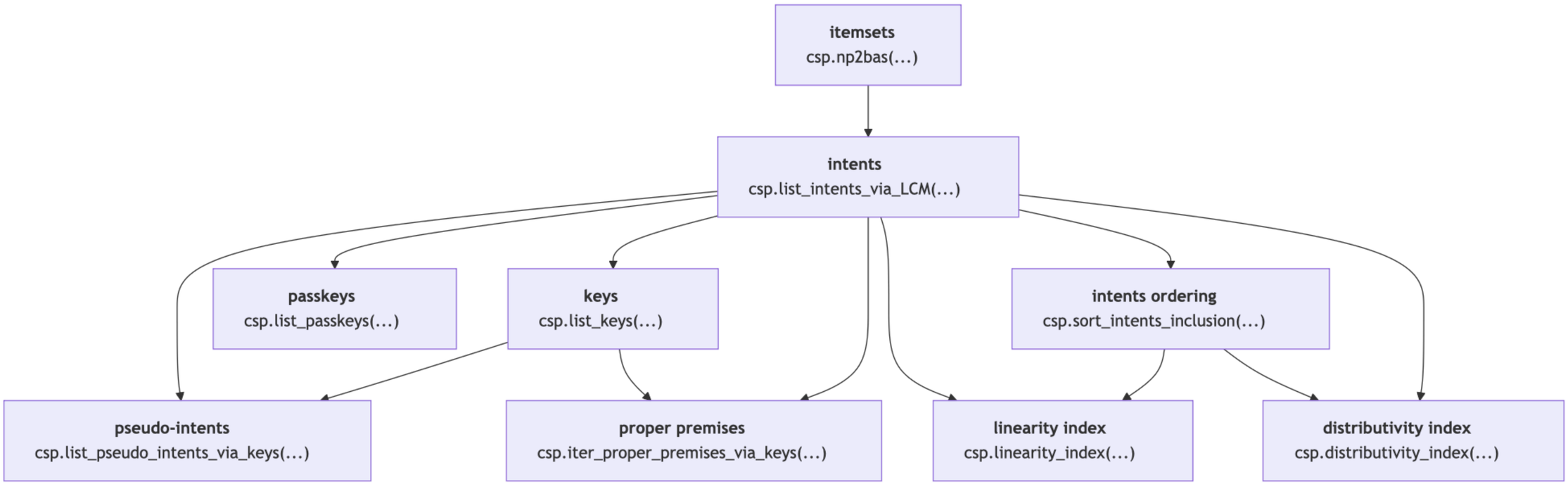
“I’ve done some tests.
Caspailleur is among the fastest software.”

Alexandre Bazin

Approaches for faster computations

1. Exploit the structure on descriptions,
2. Rely on LCM algo from scikit-mine,
3. Vertical mining with bitarrays.

Structure on descriptions



Based on a diagram from *Alexey Buzmakov, et al.*
Data complexity: An FCA-based approach, IJAR, 2024

Structure on descriptions

Pros

- Bootstraps the work
- Functions become smaller
- Ended up being fast for Proper premises
- Similar ideas published in L. Szathmary et al, 2014

Cons

- No guarantee of optimality
- The functions become interdependent: one brakes -> everything brakes
- Ended up being slooooow for Pseudo-intents

LCM algorithm

https://scikit-mine.github.io/scikireference/itemsets.html#lcm

class **skmine.itemsets.LCM**(**, min_supp=0.2, n_jobs=1, verbose=False*) [source]

Linear time Closed item set Miner.

LCM can be used as a **generic purpose** miner, yielding some patterns that will be later submitted to a custom acceptance criterion.

It can also be used to simply discover the set of **closed itemsets** from a transactional dataset.

Parameters: **min_supp** (*int or float, default=0.2*) – The minimum support for itemsets to be rendered in the output either an int representing the absolute support, or a float for relative support. By Default to 0.2 (20%)

n_jobs (*int, default=1 The number of jobs to use for the computation. Each single item is attributed a job to*) – discover potential itemsets, considering this item as a root in the search space. **Processes are preferred** over threads. **Carefully adjust the number of jobs** otherwise the results may be corrupted especially if you have the following warning: UserWarning: A worker stopped while some jobs were given to the executor.

References

[1] : Takeaki Uno, Masashi Kiyomi, Hiroki Arimura "LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets", 2004

[2] : Alexandre Termier "Pattern mining rock: more, faster, better"

Examples

```
>>> from skmine.itemsets import LCM
>>> from skmine.datasets.fimi import fetch_chess
>>> chess = fetch_chess()
>>> lcm = LCM(min_supp=2000)
>>> patterns = lcm.fit_transform(chess)
>>> patterns.head()
    itemset  support
0      [58]     3195
1      [52]     3185
2  [52, 58]     3184
3      [29]     3181
4  [29, 58]     3180
>>> patterns[patterns.itemset.map(len) > 3]
```

LCM algorithm

Pros

- A fast algorithm with a fast implementation
- Simple fit_transform API

Cons

- Not the fastest algorithm in existence
- Imports sci-kit learn that imports a bazillion of other packages
- (And gives some strange warnings lately)

Bitarrays

https://pypi.org/project/bitarray/

A formal context is represented not as objects-attributes-connections but as a list of attribute extents. And every attribute extent is represented with its characteristic vector.

Type '/' to search projects

Help Docs Sponsors Log in Register

bitarray 3.7.1

```
pip install bitarray
```

Latest version

Released: Aug 29, 2025

efficient arrays of booleans -- C extension

Navigation

- Project description
- Release history
- Download files

Verified details

These details have been verified by PyPI

Maintainers

ilanschnell

Unverified details

*These details have **not** been verified by PyPI*

Project links

Project description

bitarray: efficient arrays of booleans

This library provides an object type which efficiently represents an array of booleans. Bitarrays are sequence types and behave very much like usual lists. Eight bits are represented by one byte in a contiguous block of memory. The user can select between two representations: little-endian and big-endian. All functionality is implemented in C. Methods for accessing the machine representation are provided, including the ability to import and export buffers. This allows creating bitarrays that are mapped to other objects, including memory-mapped files.

Key features

- The bit-endianness can be specified for each bitarray object, see below.
- Sequence methods: slicing (including slice assignment and deletion), operations `+`, `*`, `+=`, `*=`, the `in` operator, `len()`
- Bitwise operations: `~`, `&`, `|`, `^`, `<<`, `>>` (as well as their in-place versions `&=`, `|=`, `^=`, `<<=`, `>>=`).
- Fast methods for encoding and decoding variable bit length prefix codes.

Bitarrays

Pros and Cons

Pros

- Really fast
- Low memory usage
- Easy to use

- For references, see papers on Vertical mining by L. Szathmary

Cons

- Hard to read by humans

Visualisations

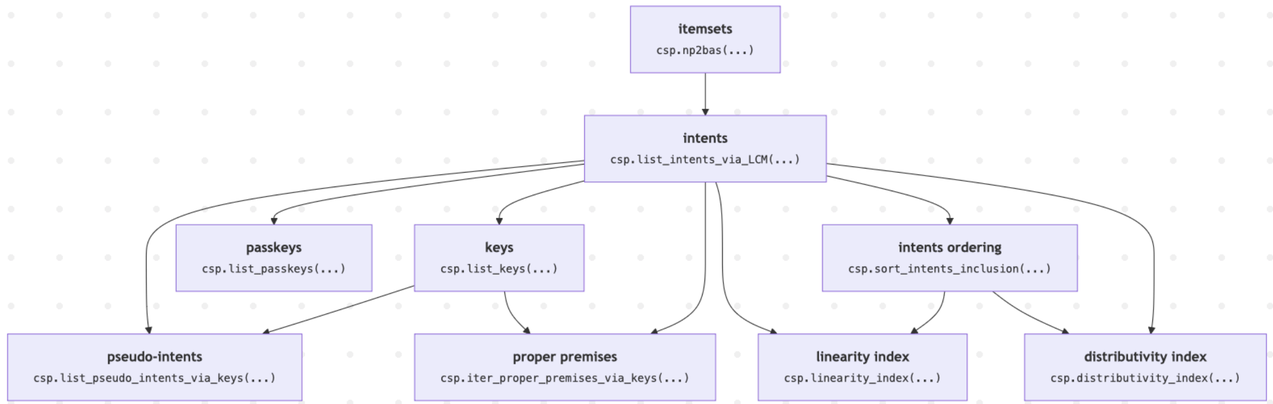
Mermaid diagrams

Mermaid Live Editor | Playground - more features, no account required | Share | Save diagram

Code | Config | Docs

```
graph TD;
    S["<b>itemsets</b><br><small><tt>csp.np2bas(..
    A["<b>intents</b><br><small><tt>csp.list_inter
    B["<b>keys</b><br><small><tt>csp.list_keys(...
    C["<b>passkeys</b><br><small><tt>csp.list_pass
    D["<b>intents ordering</b><br><small><tt>csp.s
    E["<b>pseudo-intents</b><br><small><tt>csp.lis
    F["<b>proper premises</b><br><small><tt>csp.it
    G["<b>linearity index</b><br><small><tt>csp.li
    H["<b>distributivity index</b><br><small><tt>c

    S --> A
    A --> B;
    A --> C;
    A --> D;
    A --> E;
    B --> E;
    B --> F; A --> F;
    A --> G; D --> G;
    D --> H; A --> H;
```



https://mermaid.live/edit...

Conclusions

- Caspailleur is a stable package with 2.5 years history
- Can be used for rapid analysis of formal contexts
- Most of the algorithms work pretty fast but with no guarantee of SotA

Future work

- Implement many algorithms for mining intents, keys, implication bases
- Add association rules mining