structure_comp Documentation

Release 0.1

Kevin M. Jablonka

Jun 30, 2020

CONTENTS

The purpose of the structure_comp package is to provide computational materials' scientist with easy-to-use tools to:

- remove structural duplicates from structural databases (useful for high-throughput studies)
- compare structures and distributions across feature and structure space (useful for machine learning)
- sample from structural database
- clean and check structures

CHAPTER

ONE

INSTALLATION

The latest version can always be installed from github using

```
pip install git+https://github.com/kjappelbaum/structure_comp.git
```

All requirements are automatically installed by pip. If you want to install the development extras, use

```
pip install git+https://github.com/kjappelbaum/structure_comp.git#egg=project[testing,
↪ docs, pre-commit]
```

1.1 Known issues

For the addition of missing hydrogens and the geometry relaxation (both in the `Cleaner` class) we rely on openbabel. We made the openbabel python package (`pybel`) an optional requirement in the `setup.py` which you can install with

```
pip install git+https://github.com/kjappelbaum/structure_comp.git
↪#egg=project[openbabel]
```

Note, that it still requires that you already have openbabel installed on you machine. We found it most convenient to use the anaconda package manager to do so. For this reason, we also provide an `environment.yml` file that easily allows to create a conda environment with all dependencies using

```
conda env create --name <envname> -f=environment.yml
```

CHAPTER

TWO

BACKGROUND

2.1 Introduction

In high-throughput studies one often uses databases such as the Core-COF [TLYZ17] or Core-MOF [NCC+17] database. Over the course of our work, we noticed that these databases contain a non-negligible number of duplicates, hence we wrote this tool to easily (and more or less efficiently) find and eliminate then.

Starting doing machine learning with these databases, we also noticed that we need tools for comparing them: We wanted to quantify 'diversity' and 'distance' of and between databases. Especially the `DistExampleComparison` class was written due to the fact that ML models are often good at interpolation but bad at extrapolation [MAC+18] – hence we needed tools to detect this.

For expensive simulations – or more efficient ML training – one also wants to have tools for clever sampling. This is what the `sampling` module tries to do.

2.2 Duplicate removal

The approach in the main duplicate removal routines is the following:

1. get the the Niggli reduced cell to have the smallest possible, well defined structure
2. get some cheap scalar features that describe the composition and use them to filter out structures that probably have the same composition.
3. after we identified the structures with identical composition (which are usually not too many) we can run a more expensive structural comparisons using structure graphs or the Kabsch algorithm As the Kabsch algorithm is relatively inexpensive (but it needs a threshold) we also have an option, where one can use the Kabsch algorithm and then do a comparison based on the structure graphs.

2.2.1 Kabsch algorithm

The Kabsch algorithm [Kab76][Kab78] attempts to solve the problem of calculating the RMSD between to structures, which is in general not well defined as it depends on the relative orientations of the structures w.r.t each other. The algorithm calculates the optimal rotation matrix that mininmizes the RMSD between the structures and it is often used in visualizations to e.g. align structures. The basic idea behind the algorithm is the following:

1. First we center the structures at the origin
2. Then, we calculate the covariance matrix

$$\mathbf{H}_{ij} = \sum_{k}^{N} \mathbf{P}_{ki} \mathbf{Q}_{kj}$$

where P_{ki} and Q_{kj} are the point-clouds (position coordinates) of the two structures.

3. The optimal rotation matrix is then given by

$$\mathbf{R} = \left(\mathbf{H}^{\mathsf{T}}\mathbf{H}\right)^{\frac{1}{2}} \mathbf{H}^{-1}$$

which can be implemented using a SVD decomposition.

The implementation in this package is based on the rmsd package from Jimmy Charnley Kromann [Kro19], we just added routines for periodic cases.

2.2.2 Graph based

2.2.3 Hashing

2.3 Statistics

The statistics module (`comparators.py`) implements three different classes that allow to

- measure the structural diversity of one database (`DistStatistic`)
- compare two databases (`DistComparison`)
- compare one sample with a database (`DistExample`)

The `DistStatistic` class implements parallelized versions of random (with resampling) RMSD and structure graph comparisons within a database whereas the `DistComparison` class also implements those but also several statistical tests like:

- maximum mean discrepancy
- Anderson-Darling
- Kolmogorov-Smirnov
- Mutual information

that work on a list of list or dataframe of features.

The main `Statistics` class also implements further statistical metrics, such as measures of central tendency like the trimean which are not that commonly used (unfortunately).

The `DistExample` class clusters the database (e.g. based on same property space) and then compares the sample to the k samples closest to the centroids.

2.3.1 maximum mean discrepancy (MMD)

MMD basically uses the kernel trick.

Warning: There are better implementations for MMD and especially the choice of the kernel width. In a future release, we might introduce shogon as optional dependency and use it if installed.

2.4 Sampling

For all sampling, we standardize the features by default to avoid overly large effects by e.g. different units [TFT17]. In case you want to use different weights one different features you can multiply manually the columns of the dataframe with weight factors and then turn the standardization off.

2.4.1 Farthest point sampling

The greedy farthest point sampling (FPS) [PeyrePechaudKC10] tries to find a good sampling of the point set S by selecting points according to

$$x_{k+1} = \text{argmax}_{x \in S} \min_{0 \leq i \leq k} d(x_i, x)$$

where $d(x_i, x)$ is an appropriate distance metric, which in our case is by default Euclidean. We initialize x_0 by choosing a random point from S.

2.4.2 KNN based

The k-nearest neighbor based sample selection clusters the S into k cluster and then selects the examples closest to the centroids. This is based on the rational that k (hence we sample from different clusters as we want a diverse set).

2.5 Cleaning

A problem when attempting high-throughput studies with experimental structures, e.g from the Cambridge Structural Database, is that structures [SHK+19]

- contain unbound water
- are disordered (e.g. methyl groups in two positions, aromatic carbon exist in several configurations in the `.cif` file
- contain a lot of information that is not necessarily useful for the simulation and can cause problems when using the structure as an input file for simulation packages. Also, dropping unnecessary information can significantly reduce the filesize.

There has already been work done on this topic: The authors of the Core-MOF database described their approach in the accompanying paper [CCH+14] and the group around David Fairen-Jimenez published small scripts that use Mercury and a pre-defined list of solvents to remove unbound solvents [MLW+17].

Unfortunately, to our knowledge, there exist no open-source tools at try to address all of the three issues mentioned above.

Warning: We are well aware of the problems of automatic structure sanitation tools [ZPM19]. and also advise to use them with care and to report issues such that we can improve the tools.

2.5.1 Rewriting the `cif` files

For the first stage of rewriting the `.cif` files, we use the PyCifRW package [Hes06] which is the most robust `.cif` parser we are aware of. We keep only the lattice constants and the most important loops (fractional coordinates, type and labels as well as the symmetry operations) whilst also using the atomic types as label as this is imperative for some simulation packages.

Furthermore, we remove all uncertainty indications and sanitize the filename (e.g. remove non-unicode and unsafe characters such as parentheses).

Optionally, we also remove all disorder groups other than `.` and `1`. This assumes that the disorder groups were properly labelled by the crystallographer.

2.5.2 Removing unbound solvent

For removal of unbound solvent, we construct the structure graph and query for the molecular subgraphs (pymatgen internally constructs a supercell to distinguish moleules from e.g. 2D layers). If the composition of one of the molecular subgraphs is in our list of solvent molecules we delete the molecule from the structure.

2.5.3 Removing disorder

> **Warning:** Please note that this module is experimental and does not work in all cases.

2.6 Checking

The `Checker` class allows you to run with the `run_flagging` function to run several test on a set of structures:

- It checks if there are clashing atoms (which might be due to disorder)
- It checks if there a hydrogens in the structure, in three different strictness levels: * check if there are any hydrogens at all * check if there are carbons and any hydrogens at all * check if there are carbons with less or equal two non-hydrogen neighbors if they contain any hydrogens (i.e.

  something like aromatic carbons that contain no H)
- Check if there are unbound (solvent) molecules/atoms in the structure, in two versions: * threshold-based: is there any atom that has no neighbor closer than the threshold? * graph based: are there unbound molecules in the structure graph (basically a more fancy nearest-neighbor thing)
- Check if pymatgen can read the structure

CHAPTER

THREE

QUICKSTART GUIDE

3.1 Removing Duplicates

To get the duplicates in a directory with structures, you can run something like

```
from structure_comp.remove_duplicates import RemoveDuplicates

rd_rmsd_graph = RemoveDuplicates.from_folder(
    '/home/kevin/structure_source/csd_mofs_rewritten/', method='rmsd_graph')

rd_rmsd_graph.run_filtering()
```

The filenames and the number of duplicates are saved as attributes of the `RemoveDuplicates` object.

Warning: Large databases (e.g. CCSD MOF subset) can require a large a amount of temporary hard drive ("swap") space which we use to store the Niggli reduced structures. As the main routine runs in a contextmanager, the temporary files will be deleted even if the program runs into an error. If you use `cached=True` we will not write temporary files but keep everything in memory. This is of course not feasible for large database.

We already use KDTrees and spare matrices where possible to reduce the memory requirements.

3.2 Getting Statistics

3.2.1 Measuring the diversity of a dataset

If you have properties – great, use those! With you don't have any, calculate some using some package like zeo++ or matminer. If you really want to compare structures, you can use the `DistStatistic` class. Using the randomized RMSD is decently quick, constructing structure graphs can take some time and probably does not lead to more insight:

```
from structure_comp.comparators import DistStatistic
core_cof_path = '/home/kevin/structure_source/Core_Cof/'

# core cof statistics
core_cof_statistics = DistStatistic.from_folder(core_cof_path, extension='cif')
randomized_rmsd_cc = core_cof_statistics.randomized_rmsd()
randomized_jaccard_cc = core_cof_statistics.randomized_graphs(iterations=100)
```

Then, it might be interesting to plot the resulting list as e.g. a violinplot to see whether the distribution is uniform (which would be surprising) or which RMSDs are most common as well as (what is probably most interesting) what is the width of the distribution. A example is shown in the Figure below.

3.2.2 Comparing two property distributions

If you have two dataframes of properties and you want to find out if they come from the same distribution the `DistComparison` class is the one you might want to use.

Under the hood, it runs different statistical tests feature by feature and some also over the complete dataset and then returns a dictionary with the test statistics.

QQ-plot

Something really useful is to do a QQ-test. By default, we will plot the result but also give you some metrics like the deviation of the slope of Huber regression trough the qq-plot from the diagonal. If the distributions are identical, you should see something like

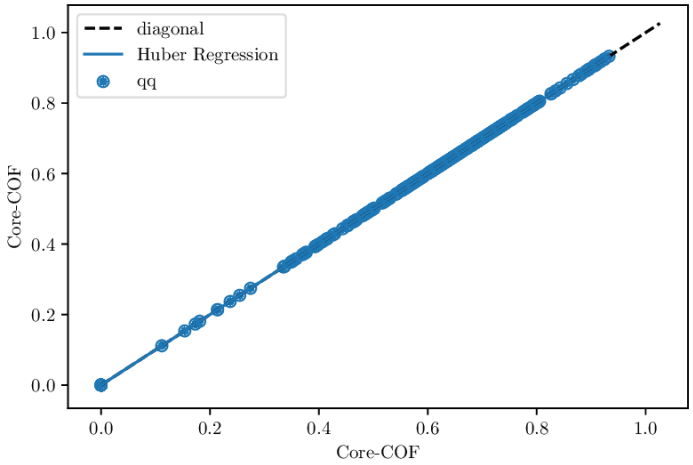

Fig. 1: QQ-plot for the void fractions of the structures in the Core-COF datset. [TLYZ17].

Whereas, if the value of the property is consistently lower for one dataset, we would expect something like

To run the QQ-test, you only need something like the following lines

```
void_fraction_martin_cc = DistComparison(property_list_1=df_martin['voidfraction'].
↪values,
                      property_list_2=df_cc['voidfraction'].values)

void_fraction_martin_cc.qq_test()
```

In our results dictionary, we would find `'deviation_from_ideal_diagonal': -3.6` which indicates that the Huber regression is much steeper than the diagonal.

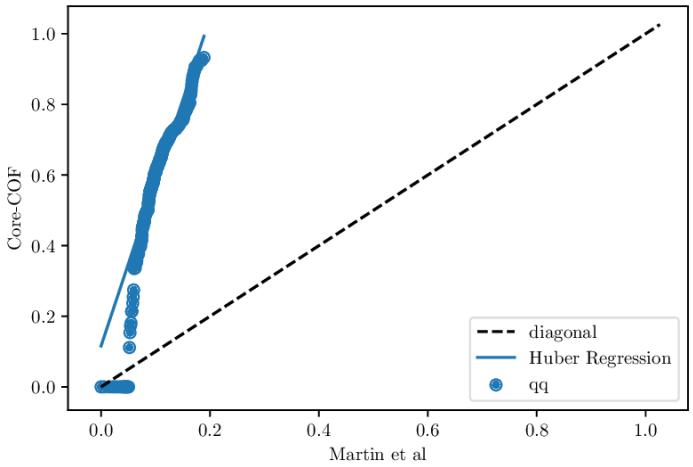

Fig. 2: QQ-plot for the void fractions of the structures in the Core-COF datset and the hypothetical COF dataset of Martin et al.

Statistical tests

Besides the QQ-test/plot you can also choose to run a variety of statistical tests on the dataset. If you provide multiple feature columns, we will run the tests column-wise and some of them globally.

A nice overview about some statistical tests can be found some Jupyter notebooks by Eric Schles.

To run the statistical tests, you can use some code snippet like the following

```
comparator = DistComparison(property_list_1=df0, property_list_2=df1)
result_dict = comparator.properties_test()
```

where `df0` and `df1` are the two property dataframes (they need to have the same number and order of columns, if you provide a dataframe, we will use the column names as keys in the output dictionary).

3.2.3 Finding out if a structure is different from a distribution

In this case you have the following possibilities:

- you can do a property-based comparison
- you can do a structure based comparison, guided by properties
- you can do a random structure based comparison

3.3 Sampling

The sampler object works on dataframes, since this interfaces smoothly with featurization packages like matminer. So far, a greedy and a clustering-based farthest point sampling have been implemented.

To start sampling you have to initialize a sampler object with dataframe, columns, the name of the identifier column and the number of samples you want to have:

```
from structure_comp.sampling import Sampler
import pandas as pd
zeolite_df = pd.read_csv('zeolite_pore_properties.csv')
columns = ['ASA_m^2/g', 'Density', 'Largest_free_sphere',
     'Number_of_channels', 'Number_of_pockets', 'Pocket_surface_area_A^2']
zeolite_sampler = Sampler(zeolite_df, columns=columns, k=9)

# use the knn-based sampler
zeolite_samples = zeolite_sampler.get_farthest_point_samples()

# or use the greedy sampler
zeolite_sampler.greedy_farthest_point_samples()
```

If you want to visualize the samples, you can call the `inspect_sample` function on the sampler object:

```
zeolite_sampler = inspect_sample()
```

If you work in a Jupyter Notebook, don't forget to call

```
%matplotlib inline
```

3.4 Cleaning Structures

3.4.1 Rewriting a `.cif` file

Most commonly we use the following function call to "clean" a `.cif` file

```
from structure_comp.cleaner import Cleaner

cleaner_object = Cleaner.from_folder('/home/kevin/structure_source/csd_mofs/', '/home/
↪kevin/structure_source/csd_mofs_rewritten')
cleaner_object.rewrite_all_cifs()
```

You will find a new directory with structures that:

- have "safe" filenames
- have no experimental details in the `cif` files
- are set to P1 symmetry
- have a `_atom_site_label` column that is equal to `_atom_site_type_symbol` which we found to work well with RASPA
- by default, we will also remove all disorder groups except `.` and `*`
- optionally you can also remove duplicates (atoms closer 0.1 sAngstom) using a ASE routine.

If you input files have a `_atom_site_charge` column, you wil also find it in the output file.

Note: You also have the option to symmetrization routines by setting `clean_symmetry` to a float which is the tolerance for the symmetrization step.

3.4.2 Removing unbound solvent

Warning: Note that this routine is slow for large structures as it has to construct the structure graph.

3.4.3 Remove disorder

For removal of disorder, we implemented two algorithms of different complexities. One performs hierachical clustering on the atomic coordinates and then merges the clashing sites of same elements.

To more naive version, simply build a distance matrix (a KDTree for efficiency reasons) and then merges the the clashing sites with following priorities: * if the elements are the same, the first site remains * if the elements are different, the heavier element remains

A code that is conservative (first uses the checker and then fixes the issues separately) could look as follows

:: from structure_comp.utils import read_robust_pymatgen from pathlib import Path

for structure in clashing_structures: s = read_robust_pymatgen(structure) name = Path(structure).name s_cleaned = Cleaner.remove_disorder(s, 0.9) s_cleaned.to(filename=os.path.join('unclashed', name))

3.5 Checking structures

To run a large variety of checks on a structural database you can use something like

```
from structure_comp.checker import Checker
from glob import glob
import pandas as pd

checker_structures = glob('*/*.cif')
checker_object = Checker(checker_structures)

problems_df = checker_object.run_flagging()
problems_df.head()
```

CHAPTER

FOUR

HOW TO CONTRIBUTE?

In general, you might find this page about Contributing to Open Source Projects useful.

4.1 Feature request or bug found

If you want to see a feature implemented in the package, open a issue. If you already have a implementation, open a pull request.

If you want me to implement a test/sampler or whatever, it would be great if you could point me at a reference describing the thing you want to have implemented.

Note: Note that we only support python 3.x. We will make no efforts to support python 2.x.

Note: At this stage I would be particulary happy about suggestions for better API design. During testing I noticed that some function names are probably not the best ones one could chose. I would be happy over all suggestions!

4.2 Code contributions

If you want to contribute code, please follow a few little guidelines:

- Try to keep PEP-8 in mind. Optimally, use a a tool like yapf and a linter and/or a good IDE like PyCharm or simply install the pre-commit hooks with

```
pre-commit install
```

- Use Google-Style docstrings
- Please write unittests.
- I think that type annotations are quite useful (it is e.g. a lot easier to keep the type annotations updated than the complete docstrings or documentation), please try to use them as well.

CHAPTER

FIVE

BIBLIOGRAPHY

CHAPTER

SIX

INDICES AND TABLES

- modindex

BIBLIOGRAPHY

[CCH+14] Yongchul G. Chung, Jeffrey Camp, Maciej Haranczyk, Benjamin J. Sikora, Wojciech Bury, Vaiva Krungleviciute, Taner Yildirim, Omar K. Farha, David S. Sholl, and Randall Q. Snurr. Computation-Ready, Experimental Metal–Organic Frameworks: A Tool To Enable High-Throughput Screening of Nanoporous Crystals. *Chemistry of Materials*, 26(21):6185–6192, November 2014. URL: http://pubs.acs.org/doi/10.1021/cm502594j, doi:10.1021/cm502594j.

[Hes06] J. R. Hester. A validating CIF parser: \textit PyCIFRW. *Journal of Applied Crystallography*, 39(4):621–625, August 2006. URL: http://scripts.iucr.org/cgi-bin/paper?S0021889806015627, doi:10.1107/S0021889806015627.

[Kab76] W Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, A32:922–923, 1976. URL: http://journals.iucr.org/a/issues/1976/05/00/a12999/a12999.pdf, doi:10.1107/S0567739476001873.

[Kab78] W Kabsch. A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, A34:827–828, 1978. URL: http://journals.iucr.org/a/issues/1978/05/00/a15629/a15629.pdf, doi:10.1107/S0567739478001680.

[Kro19] Jimmy Charnley Kromann. Calculate Root-mean-square deviation (RMSD) of two molecules, using rotation, in xyz or pdb format: charnley/rmsd. March 2019. original-date: 2013-04-24T22:46:14Z. URL: https://github.com/charnley/rmsd.

[MAC+18] Bryce Meredig, Erin Antono, Carena Church, Maxwell Hutchinson, Julia Ling, Sean Paradiso, Ben Blaiszik, Ian Foster, Brenna Gibbons, Jason Hattrick-Simpers, Apurva Mehta, and Logan Ward. Can machine learning identify the next high-temperature superconductor? Examining extrapolation performance for materials discovery. *Molecular Systems Design & Engineering*, 3(5):819–825, 2018. URL: http://xlink.rsc.org/?DOI=C8ME00012C, doi:10.1039/C8ME00012C.

[MLW+17] Peyman Z. Moghadam, Aurelia Li, Seth B. Wiggin, Andi Tao, Andrew G. P. Maloney, Peter A. Wood, Suzanna C. Ward, and David Fairen-Jimenez. Development of a Cambridge Structural Database Subset: A Collection of Metal–Organic Frameworks for Past, Present, and Future. *Chemistry of Materials*, 29(7):2618–2625, April 2017. URL: https://doi.org/10.1021/acs.chemmater.7b00441, doi:10.1021/acs.chemmater.7b00441.

[NCC+17] Dalar Nazarian, Jeffrey S. Camp, Yongchul G. Chung, Randall Q. Snurr, and David S. Sholl. Large-Scale Refinement of Metal-Organic Framework Structures Using Density Functional Theory. *Chemistry of Materials*, 29(6):2521–2528, March 2017. URL: https://doi.org/10.1021/acs.chemmater.6b04226, doi:10.1021/acs.chemmater.6b04226.

[PeyrePechaudKC10] Gabriel Peyré, Mickael Péchaud, Renaud Keriven, and Laurent D. Cohen. *Geodesic Methods in Computer Vision and Graphics*. Now Publishers Inc, December 2010. ISBN 978-1-60198-396-1. Google-Books-ID: XuuVsQ7XOl8C.

[SHK+19] Arni Sturluson, Melanie T Huynh, Alec R Kaija, Caleb Laird, Feier Hou, Zhenxing Feng, Christopher E Wilmer, Yamil J Colon, Yongchul G Chung, Daniel W Siderius, and Cory M Simon. The role of molecular modeling & simulation in the discovery and deployment of metal-organic frameworks for gas storage and separation. *ChemRxiv*, pages 56, 2019. doi:10.26434/chemrxiv.7854980.v1.

[TFT17] Trevor Tibshirani, Jerome Friedman, and Robert Tibshirani. *The Elements of Statistical Learning - Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2 edition, January 2017. URL: https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf.

[TLYZ17] Minman Tong, Youshi Lan, Qingyuan Yang, and Chongli Zhong. Exploring the structure-property relationships of covalent organic frameworks for noble gas separations. *Chemical Engineering Science*, 168:456–464, August 2017. URL: https://linkinghub.elsevier.com/retrieve/pii/S000925091730310X, doi:10.1016/j.ces.2017.05.004.

[ZPM19] Pezhman Zarabadi-Poor and Radek Marek. Comment on \textquotedblleft Database for CO $_\textrm 2$ Separation Performances of MOFs Based on Computational Materials Screening\textquotedblright . *ACS Applied Materials & Interfaces*, pages acsami.8b15684, March 2019. URL: http://pubs.acs.org/doi/10.1021/acsami.8b15684, doi:10.1021/acsami.8b15684.