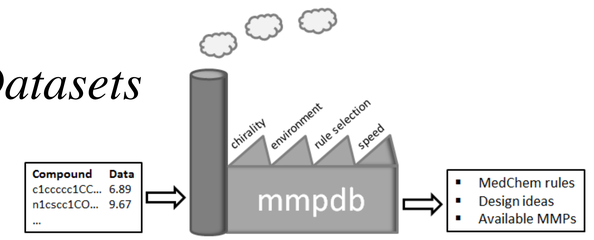
JCIM DOI: 10.1021/acs.jcim.8b00173
ChemRxiv DOI: 10.26434/chemrxiv.5999375.v1
GitHub: https://github.com/rdkit/mmpdb
"pip install mmpdb"

mmpdb: A Matched Molecular Pair Platform for Large Multi-Property Datasets

Andrew Dalke & Christian Kramer
dalke@dalkescientific.com Andrew Dalke Scientific, Sweden
Christian.Kramer@roche.com F. Hoffmann-La Roche Ltd, Switzerland



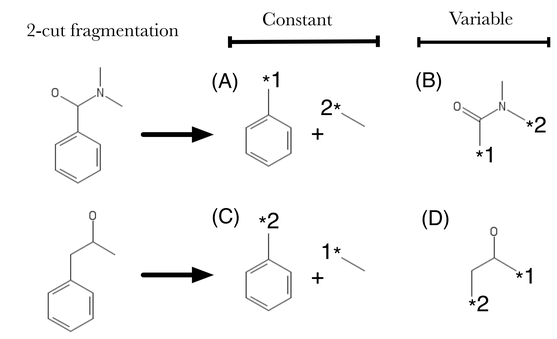
General Features

- MMP algorithm derived from Hussain&Rea[1], up to 3 cuts
  - Faster fragmentation and indexing, including parallelized fragmentation
  - Can re-use fragmentations from a previous run
  - Fully canonical transforms
  - Better support for chirality
- Capture local circular environment around transformations as a fingerprint
- Store transformations, environments, and pair statistics in a SQLite database
- Implements two analysis algorithms:
  - transform: Suggest novel compounds based on starting structure.
  - predict: Given two compounds, find transformations from one to the other and show previously measured pairs with the same transform.
- Fast analysis. Once loaded, hERG prediction from 6,000 compounds takes < 2s
- Built on RDKit
- Freely available under the 3-clause BSD license

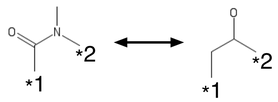
[1] Hussain, J.; Rea, C. Computationally Efficient Algorithm to Identify Matched Molecular Pairs (MMPs) in Large Data Sets. *J. Chem. Inf. Model.* **2010**, *50* (3), 339–348.

Nomenclature

Fragmentation: Identify bonds to fragment and generate all 1-, 2-, and 3-cut fragmentations. (Special support for 1-cut hydrogens not shown.)

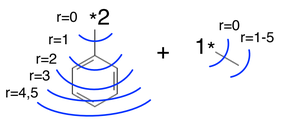


Indexing: Find matching constants and match up the attachment points in the variables to give the transform.
Constants (A) and (C) match giving a transform between variables (B) and (D)



[*:1]C(=O)N([*:2])C>>[*:1]CC([*:2])O

Environment fingerprint: A characteristic value describing the circular fingerprint centered at the attachment points of the constants.



r	environment fingerprint
0	59SlQURkWt98BOD1VlKTGRkiqFDbG6JVkeTJ3ex3bOA
1	W3GvF79axS2OEOrHQSzbDU9DdNmHb/BpV0/lF/uPcT0
2	8RcKhgVxyFGIHOg3PvZ5MD1YdRc7BcXP2EnyPQagHKw
3	WJ6FrdGEjla/ILVaN2YFWQvL3Hz1r8anFYTjFMCUTOE
4	WJ+Dva7GX6SRw/sLLWUgRO6bbfhW5hHGOA5s6miYBSQ
5	WJ+Dva7GX6SRw/sLLWUgRO6bbfhW5hHGOA5s6miYBSQ

The environment fingerprint is the SHA256 of the concatenated circular fingerprints around each of *n* attachment points in the constant, ordered from left to right in the constant's SMILES. The variable's constant environment depends on the attachment order, so a search for a matching environment may result in a search for any of n! (e.g., 6 for 3-cuts) possible fingerprints.

Statistics: Find all pairs for each combination of transform and environment radius. For each physical property record the number of pairs found, the average transformation effect, its standard deviation, median, skew, etc.

Overall data flow

Stage 0 - Identify and prepare suitable datasets for MMPA

Assemble at least a few hundred compounds. Clean up and standardize molecules as you would for QSAR/QSPR. Aggregate multiple measurements into one (mmpdb expects at most one value per compound per property). Save compounds into a SMILES file and property values into a tab-delimited file.

Stage 1 - Generate the database

a) Fragment the SMILES
```
% mmpdb fragment dataset.smi -o dataset.fragments
```

b) Index the fragments to make the MMP database
```
% mmpdb index dataset.fragments -o dataset.mmpdb
```

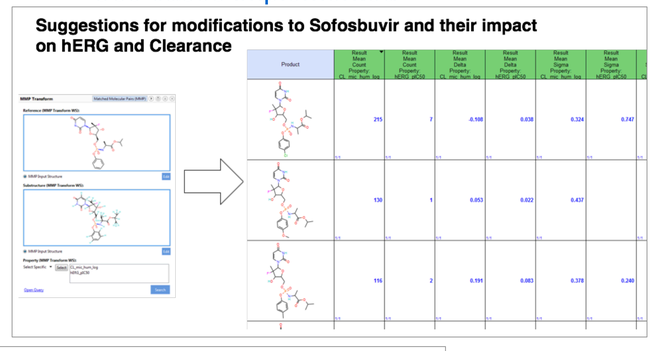
c) Add/update properties and calculate statistics
```
% mmpdb loadprops -p properties.csv dataset.mmpdb
```

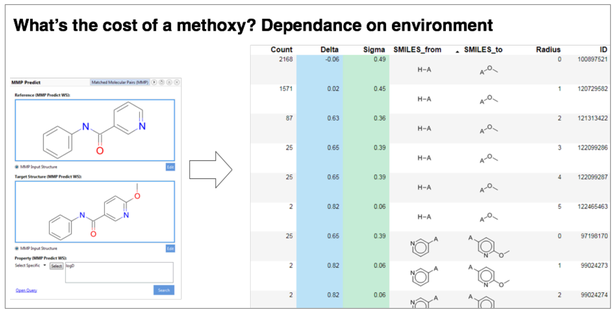
Stage 2 - Use the database for MMP analysis

```
% mmpdb transform --smiles 'c1cccnc1O' input_data.mmpdb --property MW
 ID      SMILES MW_from_smiles MW_to_smiles  MW_radius  \
  1  Clc1ccccn1         [*:1]O       [*:1]Cl          1
  2   Nc1ccccn1         [*:1]O        [*:1]N          1
  3    c1ccncc1         [*:1]O      [*:1][H]          1

                          MW_fingerprint  MW_rule_environment_id  \
tLP3hvftAkp3EUY+MHSruGd0iZ/pu5nwnEwNA+NiAh8                     298
tLP3hvftAkp3EUY+MHSruGd0iZ/pu5nwnEwNA+NiAh8                     275
tLP3hvftAkp3EUY+MHSruGd0iZ/pu5nwnEwNA+NiAh8                     267

  MW_count  MW_avg  MW_std  MW_kurtosis  MW_skewness  MW_min  MW_q1  \
         1    18.5     NaN          NaN          NaN    18.5   18.5
         3    -1.0     0.0          NaN          0.0    -1.0   -1.0
         4   -16.0     0.0          NaN          0.0   -16.0  -16.0
... additional output columns omitted
```

Use cases implemented

Transform
Suggest novel compounds and their predicted effect based on starting structure.

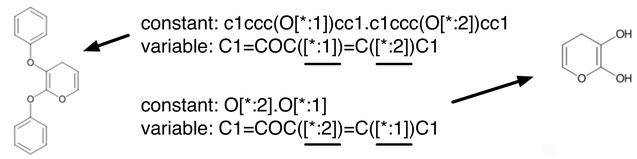




Predict
Show previously measured pairs, associated data, and summary statistics for all transformations linking two compounds

New Feature - Fully canonical transform SMIRKS

The Hussein and Rea algorithm uses a canonical *labeled* constant SMILES to store the connection map. If the constant is symmetric then the labels break symmetry, causing the variable SMILES to depend on the input SMILES order. This may give several different SMIRKS for the same transform, which makes statistics and analysis more difficult.

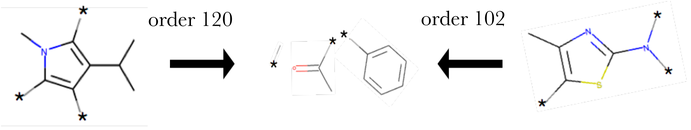


Our algorithm uses canonical *unlabeled* SMILES for the constant and variable. The connection map is stored as an "order" value. Multiple orders may be valid. We select the smallest to give a fully canonical fragmentation description. We also track the symmetry class description of the attachment points of the constant and variable.

symmetry class	variable smiles	order	symmetry class	constant smiles
12	*C1=C(*)OC=CC1	01	11	*O.*O
12	*C1=C(*)OC=CC1	01	11	*Oc1ccccc1. *Oc1ccccc1

The unlabeled attachment points may add symmetry to the variable part, causing its canonical SMILES to lose chirality information. We identify the *n* atoms in the variable SMILES which lost chirality and canonically enumerate the 2^n assignments of @ or @@ until it can be re-assembled with the constant to match the input structure.

Indexing uses a lookup table based on the LHS symmetry and order, constant symmetry, and RHS symmetry and order to give a canonical SMIRKS.



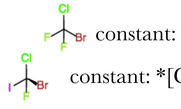
symmetry class 123 symmetry class 123 symmetry class 122

[*:1]c1c(C(C)C)c([*:2])n(C)c1[*:3] >> [*:1]c1sc(N([*:2])[*:3])nc1C

LHS attachments are ordered. RHS uses the numerically smallest of all possible assignments.

New Feature - Chiral enumeration

Transformations may remove/introduce chirality.

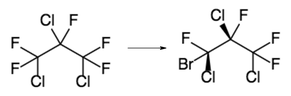


constant: *C(Br)(Cl)F variable: *F

constant: *[C@](Br)(Cl)F variable: *I

} We won't find this transform because the constants don't match.

May have several prochiral atoms:



During fragmentation, identify the *n* stereocenters in the constant which were not stereocenters in the original structure. Enumerate all 3^n-1 assignments of "@", "@@", and unassigned chirality and label these fragmentations "C". Do a similar "up-enumeration" of the variable term and label these "V".

The up-enumerated constants for FC(Br)(Cl)F are *[C@](Br)(Cl)F and *[C@@](Br)(Cl)F.

.

During indexing, find all variables with the same constant SMILES such that least one of the pair is not up-enumerated.