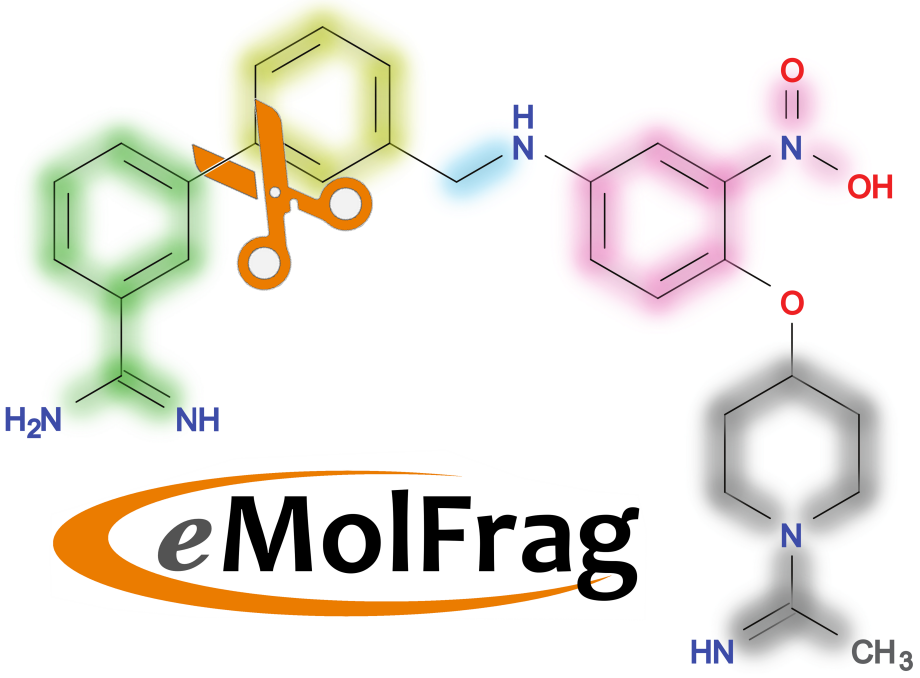

eMolFrag User's Guide

Tairan Liu

Jan 30, 2017

Disclaimer and Acknowledgements

The *eMolFrag* program was developed by Tairan Liu in *iCORE* Lab at the Department of Mechanical Engineering, Louisiana State University.
Email address:
Tairan Liu: tliu7@lsu.edu

Contents

1 Introduction

The eMolFrag is a new open source software to decompose organic compounds into non-redundant fragments retaining molecular connectivity information.
It is implemented in Python using RDKit [4] as an easy to use and platform-independent tool. And it uses the BRICS algorithm [2] which implemented in RDKit as the basis of fragmentation. It gives users access to break down small molecules into bio-active fragments which can be used to generate new molecules to expand the molecule library.
The output of eMolFrag is exactly suitable for eSynth to do further molecular synthesis.
The following sections will give an overview of different aspects of eMolFrag.

2 Installation and Setup eMolFrag

Prerequisites:
1. Python (either 2 or 3)
2. RDKit 2015.09.2 or newer (2016.03.3 has been tested). It is recommended to use Anaconda to install RDKit and use the following command: "conda install -c rdkit rdkit=2015.09.2". Creating conda environment may not work properly.
3. pkcombu [3]
4. eMolFrag scripts
5. Openbabel 2.3.1 [8] (Optional)

Installation:
Use ConfigurePath.py to configure paths. Paths only need to be set before the first run if no changes to the paths. After the script starts, instructions will be given for setting paths.

1. The first path is for eMolFrag scripts. The absolute path to the scripts folder is needed.
2. The second path is for pkcombu. The absolute path to pkcombu to be used is needed.

For example:

```
$ python /.../ConfigurePath.py # run Configure path,
   use absolute path
$ # step 1: assuming that path to eMolFrag.py is /.../
   eMolFrag_201x_xx_xx_xx/eMolFrag.py, type: /.../
   eMolFrag_201x_xx_xx_xx/
$ # step 2: assuming that path to pkcombu is /.../
   pkcombu, type: /.../pkcombu
```

3 Using eMolFrag

Run scripts to process data:

```
$ /Path_to_Python/python /Path_to_scripts/eMolFrag.py -
    i /Path_to_input_directory/ -o /
    Path_to_output_directory/ -p Number-Of-Cores -m
    Output-selection -c Output-format
```

Table 1: Input segments description.

Term	Description
/Path_to_Python/python	Can be simplified as python ignoring the absolute path.
/Path_to_scripts/eMolFrag.py	Main path to the scripts, relative path is also acceptable.
/Path_to_input_directory/	Path of the directory which contains input mol2 files, relative path is also acceptable.
/Path_to_output_directory/	Path of the directory for output, relative path is also acceptable.
Number-Of-Cores	Number of processes created in parallel step. It is better to set this parameter no larger than the number of cores of the system/node/cluster.
Output-selection	Different output select, remove redundancy or not.
Output-format	Keep or the files or put all the output bricks or linkers in 2 or 4 files.

Table 2: Input arguments description.

Parameter	Optional	Default argument	Example of argument	Description
-i	N	No default	/.../test-set100/	Input path
-o	N	No default	/.../output-100-1/	Output path
-p	Y	1	16	Parallel cores to be used
-m	Y	0	1	Output selection: 0: full process and output; 1: only chop (and reconnect); 2: chop and remove redundancy, but remove temp chop files, only output the bricks and linkers after remove redundancy
-c	Y	0	1	Output format: 1: all linkers in one file, all bricks in one file, all logs in one folder; 2: remove log folder; 0: traditional format.

Example:

```
$ mkdir TestEMolFrag/   # Any folder name you want
$ cp /.../test-set100.tar.gz /.../TestEMolFrag/
$ cd /.../TestEMolFrag/
$ tar -zxf test-set100.tar.gz   # Sample input data set
$ python ConfigurePath.py       # Configure path, use
   absolute path
# - Step 1: Assume path to eMolFrag.py is /.../
   eMolFrag_201x_xx_xx_xx/eMolFrag.py, then type: /.../
   eMolFrag_201x_xx_xx_xx/ at this step.
# - Step 2: Assume path to pkcombu is /.../pkcombu,
   then type: /.../pkcombu at this step.
$ python /.../eMolFrag_201x_xx_xx_xx/eMolFrag.py -i
   /.../TestEMolFrag/test-set100/ -o /.../TestEMolFrag/
   outputp-testset100-1/ -p 16 -m 0 -c 0  # Directory
   name can be changed to whatever you want.
$ # Check output.
```

4 Output

The output of eMolFrag will be fragments generally in two categories: brick and linker.
An example is shown below in Figure 1.

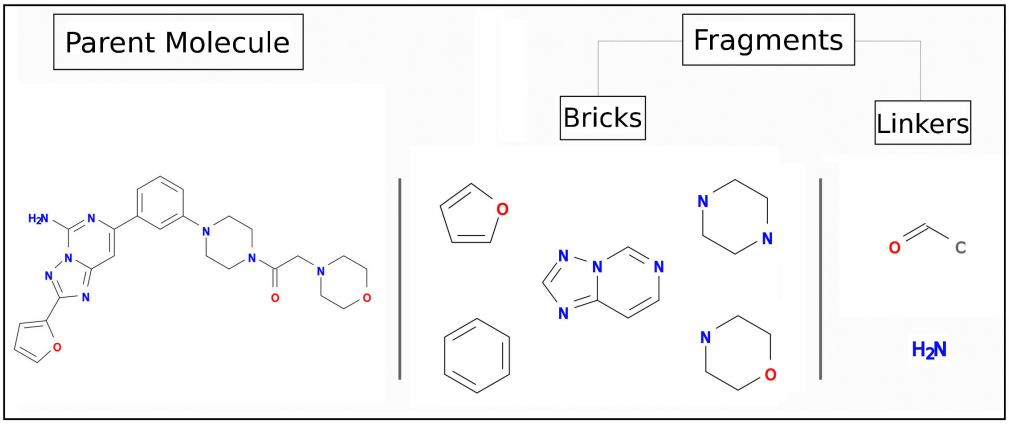

Figure 1: Example: parent molecule and fragments.

As in Figure 1, the first column is the parent molecule, the second and third columns are all the fragments generated from this parent molecule.

Now take a look of the detail of fragments.
The format of one generated brick fragment is shown in Figure 2 as an example.
Corresponding to the Structure Data Format [1] file (*.sdf), the first line is the name of the fragment. Then comes the information of the fragment structure. After that, "> <ATOMTYPES>" section shows the sybyl atom types [1] corresponding to each atom of this fragment. "> <BRANCH @atom-number eligible-atmtype-to-connect>" section shows all the possible sybyl atom type(s) [1] can be connected to the atoms of this fragment. For example, in this brick fragment, the 6th atom N.3 can connect to C.3 atom, so that is it. "> <fragments similar>" section gives the information of all the similar fragments including itself.

```
b-CHEMBL175476.mol2-000.sdf
     RDKit          3D

  6  6  0  0  0  0  0  0  0  0999 V2000
    1.2268  -10.0020   -0.0554 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.4759  -10.3733    1.5879 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.8909   -8.5086   -0.0838 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.8291   -8.8837    1.5761 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0595  -10.7412    0.3138 O   0  0  0  0  0  0  0  0  0  0  0  0
    0.3682   -8.1042    1.2294 N   0  0  0  0  0  0  0  0  0  0  0  0
  1  3  1  0
  1  5  1  0
  2  4  1  0
  2  5  1  0
  3  6  1  0
  4  6  1  0
M  END

> <ATOMTYPES>
C.3
C.3
C.3
C.3
O.3
N.3

> <BRANCH @atom-number eligible-atmtype-to-connect>
6 C.3


> <fragments similar>
synth-mol-output/b-CHEMBL175476.mol2-000.sdf
$$$$
```

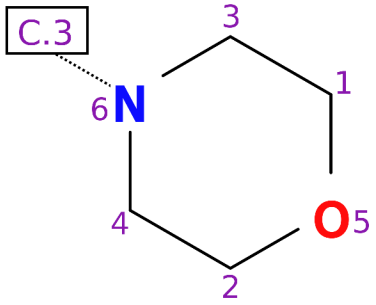

Figure 2: Example: brick fragment.

Also one generated linker fragment as an example in Figure 3.
In linker fragment file, "> <MAX-NUMBER-Of-CONTACT ATOMTYPES>" indicates the maximum number of connections and the atom type of each atom in this linker fragment. For example, the second line "1 C.3" means the second atom is C.3 and it can connect 1 other atom at most.

```
l-CHEMBL175476.mol2-001.sdf
     RDKit          3D

  3  2  0  0  0  0  0  0  0  0999 V2000
    2.4295   -6.4901    1.0459 O   0  0  0  0  0  0  0  0  0  0  0  0
    0.0858   -6.6629    1.2625 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.3774   -5.8952    1.1464 C   0  0  0  0  0  0  0  0  0  0  0  0
  1  3  2  0
  2  3  1  0
M  END

> <MAX-NUMBER-Of-CONTACTS ATOMTYPES>
0 O.2
1 C.3
1 C.2

$$$$
```

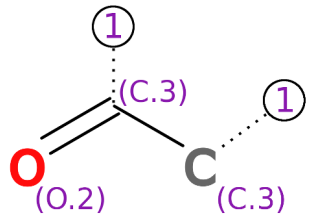

Figure 3: Example: linker fragment.

eMolFrag provides options to control the format of output. Here is a list of output folders/files and their corresponding descriptions for choosing different options.

Find correct output folder, assume /.../Output-xxxx/. Some files or folders may not appear if using different output selection choices.
A. Output format 0: Traditional format

Table 3: Output description.

Directory	File Name	Description
../output-chop-comb	*.sdf	All brick and linker fragments before removing redundancy.
../output-brick	b-*.sdf	All unique brick fragments after removing redundancy.
../output-linker	l-*.sdf	All unique linker fragments after removing redundancy.
../output-log	InputList	File contains all the input *.mol2 file names
	ListAll	File contains all the fragments before reconnecting small linkers, and total/carbon/nitrogen/oxygen atom numbers in each fragment
	BrickListAll.txt	File contains all the brick fragments and total/carbon/nitrogen/oxygen atoms in each fragment.
	LinkerListAll.txt	File contains all the linker fragments after reconnect and total/carbon/nitrogen/oxygen atoms in each fragment.
	BrickGroupList.txt	Brick fragments are grouped by total/carbon/nitrogen/oxygen atoms in each fragment as property, and a number of how many fragments have the same property.
	LinkerGroupList.txt	Linker fragments are grouped by total/carbon/nitrogen/oxygen atoms in each fragment as property, and a number of how many fragments have the same property.
	brick-log.txt	Log file for removing redundancy of brick fragments and similarity information.
	linker-log.txt	Log file for removing redundancy of linker fragments and similarity information.
	Process.log	Log file for the whole process.

B. Output format 1: Simple output format with log and statistics

Table 4: Output description.

Directory	File Name	Description
../	BrickFull.sdf	All brick fragments before removing redundancy in one file.
../	BrickUnique.sdf	All unique brick fragments after removing redundancy in one file.
../	LinkerFull.sdf	All linker fragments before removing redundancy in one file.
../	LinkerUnique.sdf	All unique linker fragments after removing redundancy in one file.
../output-log	InputList	File contains all the input *.mol2 file names
	ListAll	File contains all the fragments before reconnecting small linkers, and total/carbon/nitrogen/oxygen atom numbers in each fragment
	BrickListAll.txt	File contains all the brick fragments and total/carbon/nitrogen/oxygen atoms in each fragment.
	LinkerListAll.txt	File contains all the linker fragments after reconnect and total/carbon/nitrogen/oxygen atoms in each fragment.
	BrickGroupList.txt	Brick fragments are grouped by total/carbon/nitrogen/oxygen atoms in each fragment as property, and a number of how many fragments have the same property.
	LinkerGroupList.txt	Linker fragments are grouped by total/carbon/nitrogen/oxygen atoms in each fragment as property, and a number of how many fragments have the same property.
	brick-log.txt	Log file for removing redundancy of brick fragments and similarity information.
	linker-log.txt	Log file for removing redundancy of linker fragments and similarity information.
	Process.log	Log file for the whole process.

C. Output format 2: Simple output format.

Table 5: Output description.

Directory	File Name	Description
../	BrickFull.sdf	All brick fragments before removing redundancy in one file.
../	BrickUnique.sdf	All unique brick fragments after removing redundancy in one file.
../	LinkerFull.sdf	All linker fragments before removing redundancy in one file.
../	LinkerUnique.sdf	All unique linker fragments after removing redundancy in one file.

5 Algorithms

The algorithms used in this program are listed below as they appeared in the paper [5].

Algorithm 1 Molecular fragmentation

```
1: procedure FRAGMENT(SET<MOLECULE> M)
2:     List < Brick > B := φ
3:     List < Linker > L := φ
4:     for each m ∈ M do
5:         for each f ∈ FragmentOnBRICSBonds(m) do
6:             if f.isBrick() then
7:                 I := f.RemoveDummyAtoms()
8:                 f.RemoveHydrogen()
9:                 f.AddAppendix(I)
10:                B_m := B_m ∪ {f}
11:            end if
12:        end for
13:        B := B ∪ B_m
14:        L := l ∪ ComputeLinkers(m, B_m)
15:    end for
16:    return < B, L >
17: end procedure
```

Algorithm 2 Linker extraction

```
1: procedure ComputeLinkers(Molecule m, List<Brick> B_m)
2:     for each b ∈ B_m do
3:         m.removeBrick(b)
4:     end for
5:     List < Linker > ℓ := m.RemainingFragments()
6:     for each l ∈ ℓ do
7:         l.AdddAppendix(m)
8:     end for
9:     return ℓ
10: end procedure
```

Algorithm 3 Removal of redundant fragments

```
1: procedure RemoveRedundancy(List<Fragment> F)
2:     List < Fragment > U := φ // Unique fragment set
3:     List < Set < Fragment >> 𝒫/ := Partition(F)
4:     for each P ∈ 𝒫/ do
5:         while P ≠ φ do
6:             f_0 := P.removeFirst()
7:             U := U ∪ {f_0}
8:             for each f ∈ P do
9:                 if f_0 = f then
10:                    P := P{f}
11:                end if
12:            end for
13:        end while
14:    end for
15:    return U
16: end procedure
17: procedure RemoveRedundancy(List<Brick> ℬ,
    List<Linker> 𝒮)
18:    ℬ := RemoveRedundancy(ℬ)
19:    𝒮 := RemoveRedundancy(𝒮)
20: end procedure
```

6 Validation and Performance

A. Validation

eMolFrag was validated using a self-reconstruction test as described in [7]. Briefly, the process can be described as:
(1) Using the fragments generated from input molecule with eMolFrag to synthesis new molecules with eSynth.
(2) Check the TC with all the synthesis results of eSynth against input molecule, find the highest TC.
If the highest TC can be 1.0, then it is called fully reconstructed.
As a testing set, the 20408 active compounds for 102 protein targets from the Directory of Useful Decoys, Enhanced (DUD-E) database covering a diverse chemical space of pharmacologically relevant molecules [5, 6] were used. Also, a wall time of 1 hour was set to the test so that each set of self reconstruction was limited to 1 hour.
The result is listed below in Table 6.

Table 6: Self reconstruction result. [5]

TC	Percentage
$= 1.0$	82.82%
≥ 0.90	88.25%
≥ 0.80	92.16%
≥ 0.70	94.61%
≥ 0.60	96.08%
≥ 0.50	97.31%

B. Performance

When running eMolFrag with removing redundancy step, the time cost for different input size in both serial case (1 CPU) and parallel case (16 CPUs) are plotted in Figure 4. Also, the speedup factor of using different number of CPU cores is also plotted in Figure 4.
All tests were performed on compute nodes equipped with two 2.6 GHz 8-core Sandy Bridge Xeon 64-bit processors, 32GB 1666MHz RAM and 500GB HD, running Red Hat Enterprise Linux 6. [5]

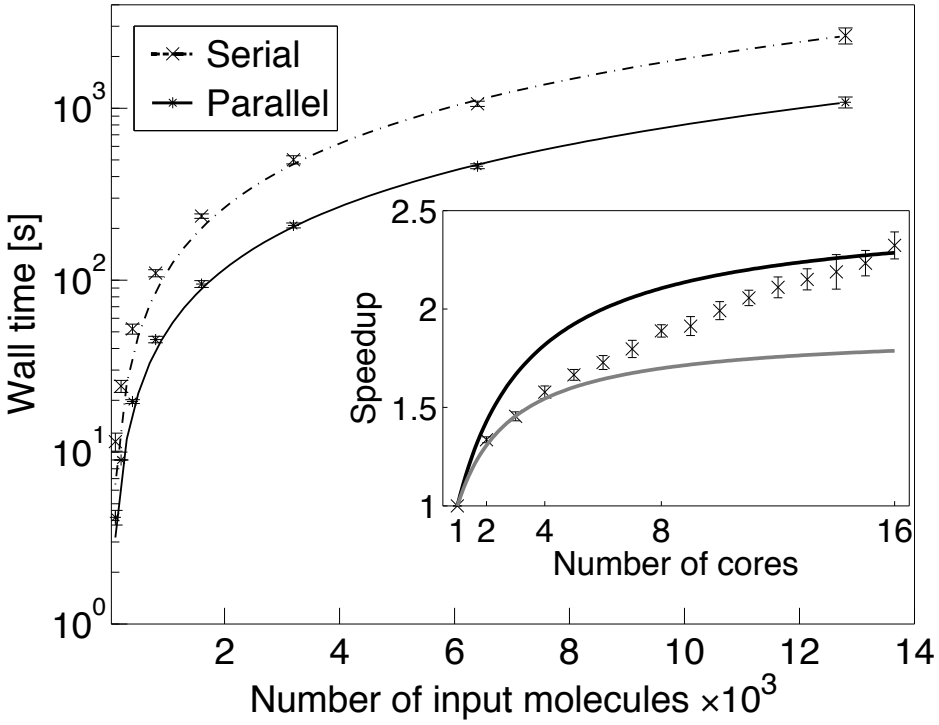

Figure 4: Time cost for different input size and speedup with different CPU cores. [5]

A comparison of time cost without removing redundancy step for eMolFrag serial case, parallel case and molBlocks are plotted in Figure 5.

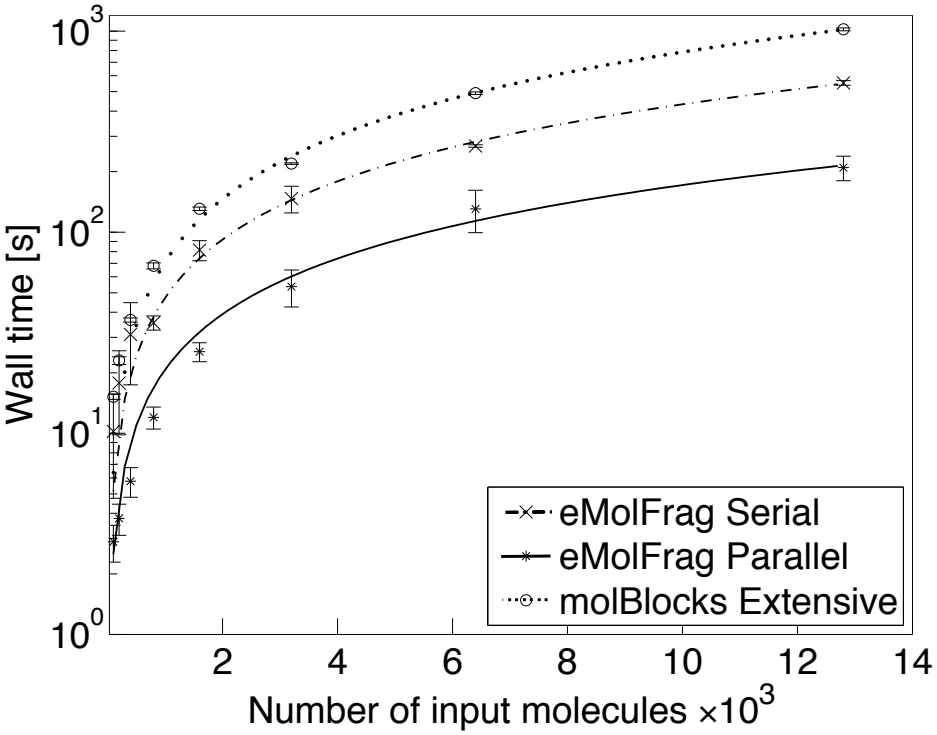

Figure 5: Timing comparison of eMolFrag and molBlocks. [5]

7 Fragment Statistics

The number of fragments generated from each input molecule and the number of atoms of each fragment are plotted as a 2D box plot in Figure 6. This figure also contains the distribution of numbers for molBLOCKS with and without extension cases.

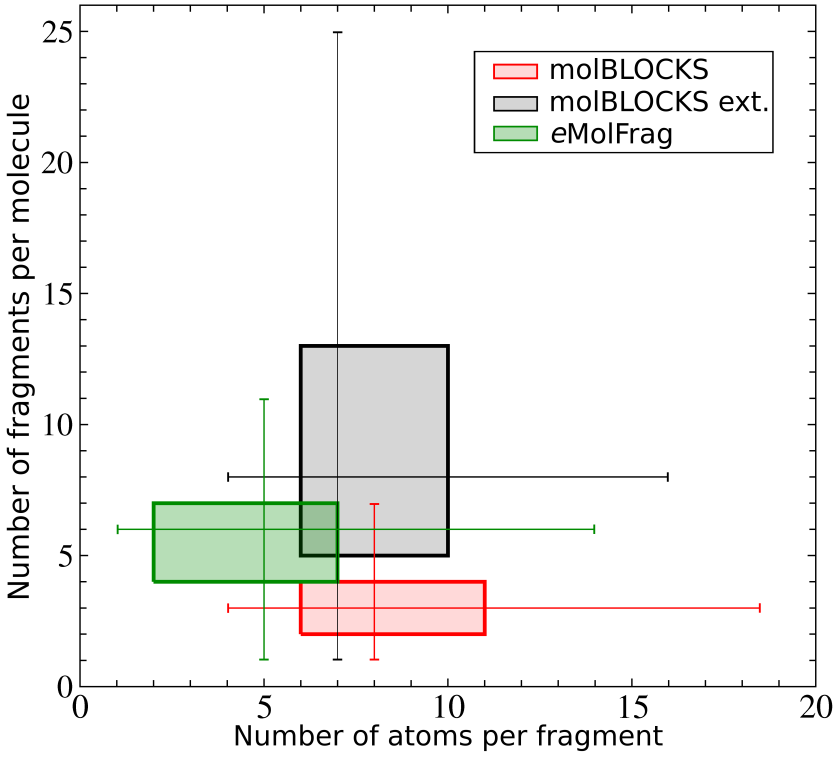

Figure 6: Number of fragments per molecule and number of atoms per fragment distribution. [5]

8 Molecular Synthesis with eSynth

A case study is attached as an example of how fragments are used to generate new molecules. Details of this case study can be found in [5].
As in Figure 7, the molecules in first row (A) are parent molecules for the molecular synthesis process. The second row (B) and third row (C) are brick and linker fragments generated from parent molecules. The fourth row are samples of synthesis results.
One of the synthesis result is "CHEMBL144979" of DUD-E library, which has very low similarity with all four parent molecules.
eSynth totally generated 4492609 molecules within 12 hours, and eSynth was actually stopped by the wall time bound. Which means there are more possibilities if sets a longer time bound.

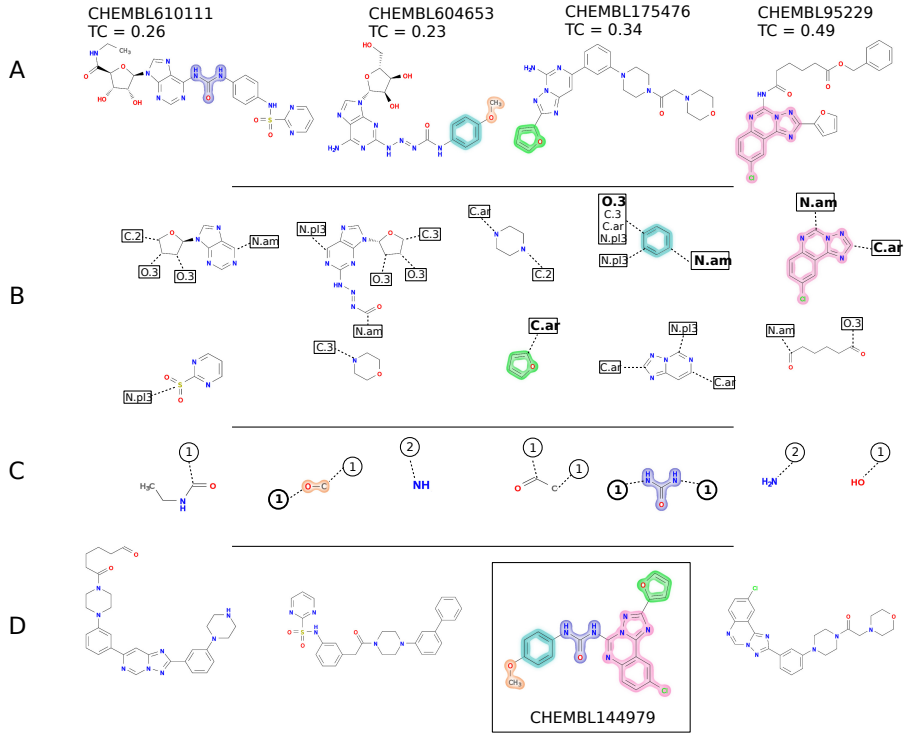

Figure 7: Case study for molecular synthesis [5]

9 Online Resources

New version of eMolFrag will be released on GitHub page:
`https://github.com/liutairan/eMolFrag`

If you meet any problems or find any bugs with eMolFrag, please raise an issue on GitHub issues page:
`https://github.com/liutairan/eMolFrag/issues`

Lastest stable version will also be released on:
`http://brylinski.cct.lsu.edu/content/emolfrag-standalone-package`

A web server which can provide fragmentation service:
`http://brylinski.cct.lsu.edu/content/emolfrag-webserver`

Molecular Synthesis tool: eSynth
`http://brylinski.cct.lsu.edu/content/molecular-synthesis`

References

[1] Matthew Clark, Richard D Cramer, and Nicole Van Opdenbosch. Validation of the general purpose tripos 5.2 force field. *Journal of Computational Chemistry*, 10(8):982–1012, 1989.

[2] Jörg Degen, Christof Wegscheid-Gerlach, Andrea Zaliani, and Matthias Rarey. On the art of compiling and using'drug-like'chemical fragment spaces. *ChemMedChem*, 3(10):1503–1507, 2008.

[3] Takeshi Kawabata. Build-up algorithm for atomic correspondence between chemical structures. *Journal of chemical information and modeling*, 51(8):1775–1787, 2011.

[4] Greg Landrum. Rdkit: Open-source cheminformatics. *Online). http://www. rdkit. org. Accessed*, 3(04):2012, 2006.

[5] T Liu, M Naderi, C Alvin, S Mukhopadhyay, and M Brylinski. Break down in order to build up: Decomposing small molecules for fragment-based drug design with emolfrag. Submitted.

[6] Michael M Mysinger, Michael Carchia, John J Irwin, and Brian K Shoichet. Directory of useful decoys, enhanced (dud-e): better ligands and decoys for better benchmarking. *Journal of medicinal chemistry*, 55(14):6582–6594, 2012.

[7] M Naderi, C Alvin, Y Ding, S Mukhopadhyay, and M Brylinski. A graph-based approach to construct target-focused libraries for virtual screening. *J Cheminform*, 8:14, 2016 2016.

[8] Noel M O'Boyle, Michael Banck, Craig A James, Chris Morley, Tim Vandermeersch, and Geoffrey R Hutchison. Open babel: An open chemical toolbox. *Journal of cheminformatics*, 3(1):33, 2011.