LEA3D

Ligand by Evolutionary Algorithm

Version 3.0

References:

Douguet D., Munier-Lehmann H., Labesse G. and Pochet S., LEA3D: A Computer-Aided Ligand Design for Structure-Based Drug Design, *J. Med. Chem.*, **2005**, *48*, 2457-2468. Doi: 10.1021/jm0492296

Douguet D., e-LEA3D: a computational-aided drug design web server, Nucleic Acids Res., **2010**, 38, Suppl:W615-21. Doi:10.1093/nar/gkq322

Douguet D., Thoreau E. and Grassy G., LEA (Ligand by Evolutionary Algorithm): A Genetic Algorithm for the Automated Generation of Small Organic Molecules, *J. Comput.-Aided Mol. Design*, **2000**, *14*, 449-466. Doi: 10.1023/A:1008108423895

Introduction:

Computer-aided drug design methods contribute to the early stage of the drug discovery process. Computational methods include *in silico* high throughput screenings of existing or virtual chemical databases in order to identify new bioactive molecules.

LEA3D is a suite of automated *de novo* molecular design programs. It is designed to create new molecules by using a library of molecular fragments and by determining best combinations of molecular fragments that fit user-defined physicochemical properties (also called constraint function or fitness function). LEA3D (Ligand by Evolutionary Algorithm) is based on a genetic algorithm that evolves the molecular structures generation after generation until the emergence of fitted molecules. Each molecule of each generation is evaluated thanks to a fitness function (constraints) which can be either molecular properties, an affinity prediction by a docking program… (Figure 1).

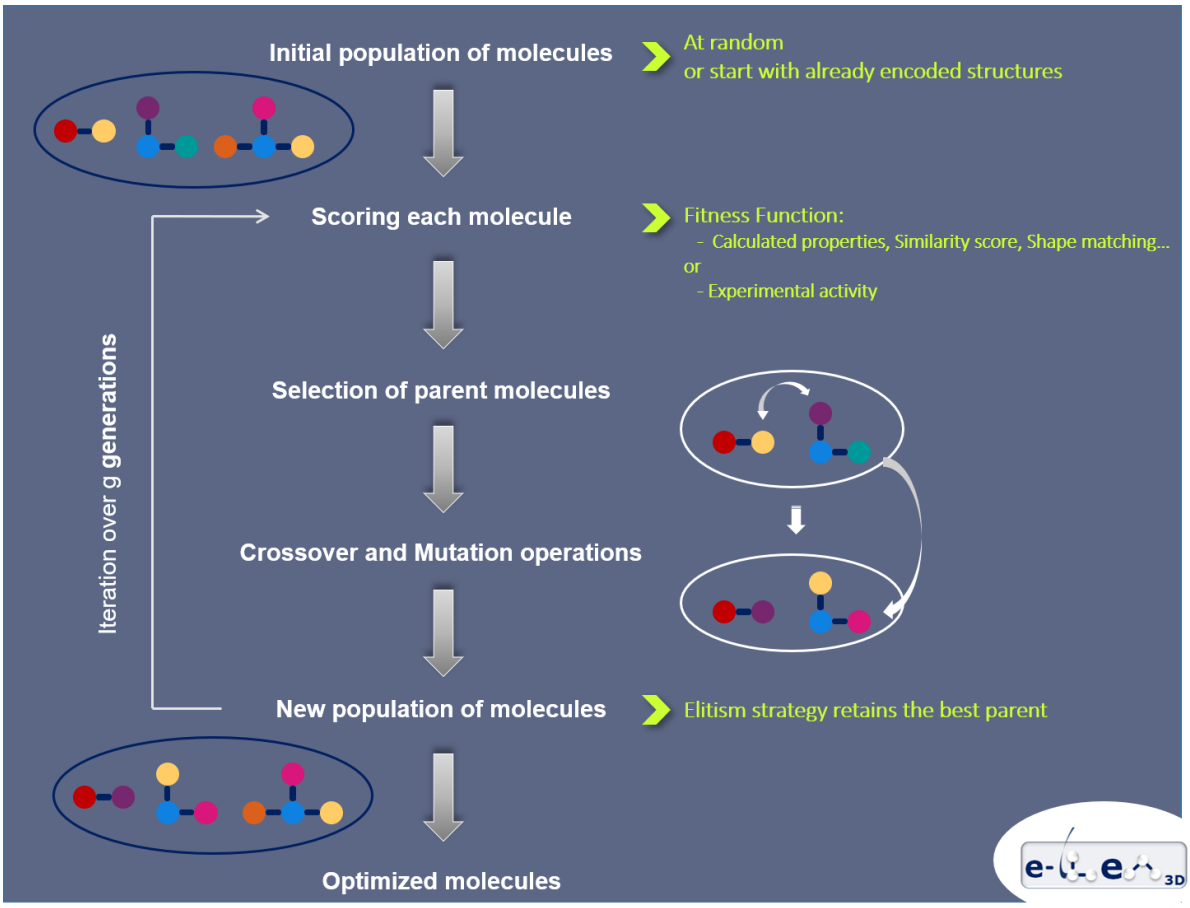


Figure 1. General flowchart for LEA genetic algorithm. An initial population of candidate solutions is generated, usually, by random process but an option allows to start with a pool of molecules. The fitness of each candidate is evaluated via a fitness function (or score), which takes as input a candidate solution and returns a numeric score. Selection criteria are applied to choose candidates based on their fitness score for breeding. Breeding functions, **crossover** and **mutations (suppress, add, replace or permutate a fragment)**, are applied to produce new solutions that replace the parent solutions. The cycle (or generation g) continues until convergence criteria is met (usually, solutions are no more improved).

Each molecule results from the association of various 3D fragments also called “legos” which are, in our case, extracted from FDA approved drugs (see the e-Drug3D database; https://chemoinfo.ipmc.cnrs.fr/MOLDB/index.html). The database of “legos” contains thousands of fragments associated with their frequency in drugs (files all.sdf and frequencies.txt in LEGO folder). The greater the frequency, the more likely the fragment selection will be. Each fragment possesses one or more ‘X’ dummy atoms that retain the substitution pattern of the original drug (Figure 2). During the recombination step with other fragments, the substitution site of a fragment is selected randomly among the original substitutions if there are several.

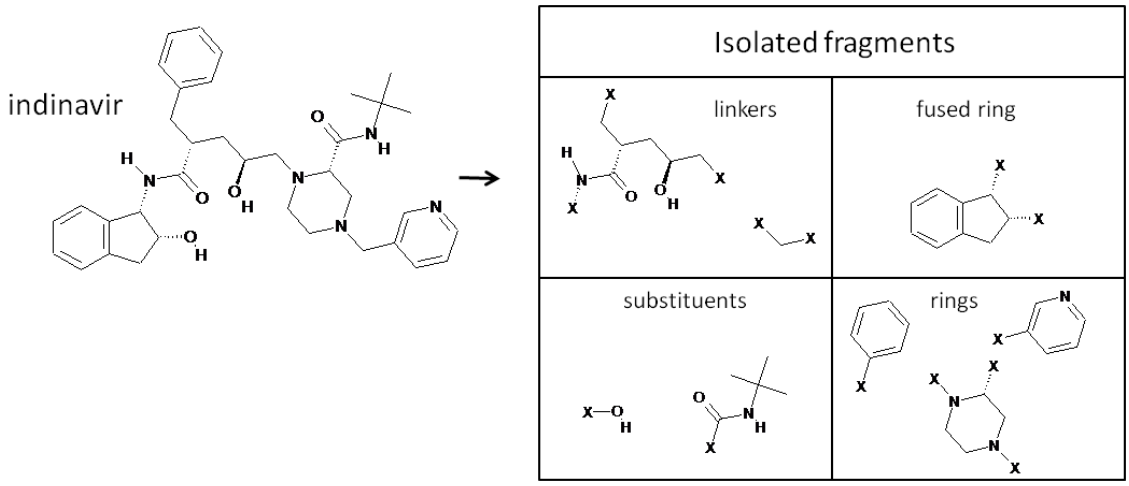


Figure 2. Fragmentation of drug Indinavir. Isolated fragments are linkers, fused rings, substituents and rings.

Alternatively, LEA3D may be used to screen a database of molecules, in this case, only the first step of scoring 'initial population of molecules' is performed (*ie* number of generation = 0).

A - Installation of the LEA3D core

Virtual environment for python with conda (for Windows for example)

Install conda or Miniconda.

Launch Anaconda Prompt, then complete the installation:

```
conda update conda
conda create -n lea3d
conda activate lea3d

(lea3d) > conda install python=3.7 numpy
(lea3d) > conda install perl
(lea3d) > conda install -c conda-forge rdkit
(lea3d) > conda install -c schrodinger pymol
      If an error regarding PyQT occurs then:
      (lea3d) > install -c anaconda pyqt
(lea3d) > conda install matplotlib
```

Retrieve and unzip LEA3D repository in your desired folder. The directory containing executables is called LEA3D-main. See below for running the program MAIN.

B - Tutorials

Create a folder called “Project”, copy the content of the folder “examples” in the “Project” directory and launch conda.
In the folder “Project”, create a folder called “VISU”.

(base) > activate lea3d
(lea3d) > cd Project

a) Use lea3d to design molecules using some aspirin molecular properties

In this example, the file ligand-aspirin.in defines parameters of the genetic algorithm (a population of 10 molecules that will evolve over 30 generations) and indicates the fitness function file to read (ligand-aspirin.func). The fitness function includes 4 properties to evaluate (molecular weight, number of atoms, fsp3 value and the presence of 2 chemical functions (ester and acid)).

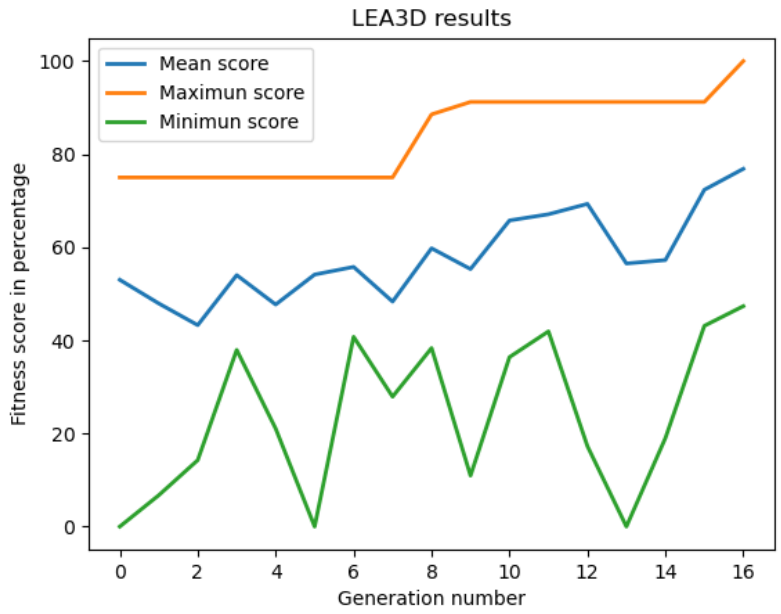
Execute:

(lea3d) > perl ../LEA3D-main/MAIN ligand-aspirin.in

The result file summary.out indicates the score of candidate molecules of the last generation ranked in descending order. The file edesign.sdf contains all generated molecules over the run. The file VISU/list.sdf contains the best candidate molecule of each generation. The file popopop.txt contains the encoded molecules of the last generation. The last can be used to start a new run when using the option START in the parameter file ligand.in. The file fitmoy.dat allows to plot the maximum, minimum and average scores in function of the generation number. In addition, the file operator.out records the crossover and mutation operations, indicates the difference in score value and which lego is involved (if any). It allows to analyze the efficiency of each operator. Of note, at the end of the run, the list of the privileged legos that improve candidate molecules is indicated.

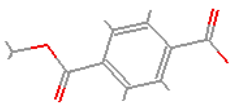
To plot the maximum, minimum and average scores in function of the generation number, the script in directory ./utils is executed:

(lea3d) > python ../LEA3D-main/utils/plot-scores.py fitmoy.dat



Visualize the best candidate of each generation:

(lea3d) > pymol VISU/list.sdf



The best candidate of the run is shown in the Figure above: it is the last molecule of the file list.sdf of the run.

b) Use lea3d to generate 3 molecules using a pre-defined combination of legos:

In this example, the objective is to build molecules that are already encoded without evaluation of any properties. The file list_mol_sulfapyridine-aspirin_venetoclax contains the encoding for three molecules (one per line) and the file ligand-aspirin.in indicates which library of fragment to use (here, the default SDF file called all.sdf from the folder LEGO).

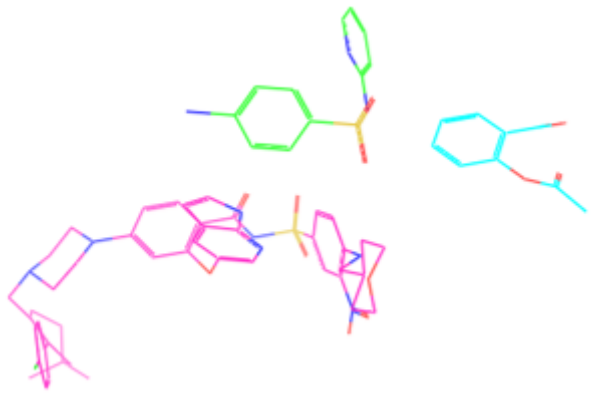
Execute:

```
(lea3d) > perl ../LEA3D-main/MAIN -v ligand-aspirin.in list_mol_sulfapyridine-aspirin_venetoclax
```

Generated molecules are saved under the name mol_1.sdf, mol_2.sdf...

Visualize the three generated structures:

```
(lea3d) > pymol mol_1.sdf mol_2.sdf mol_3.sdf
```



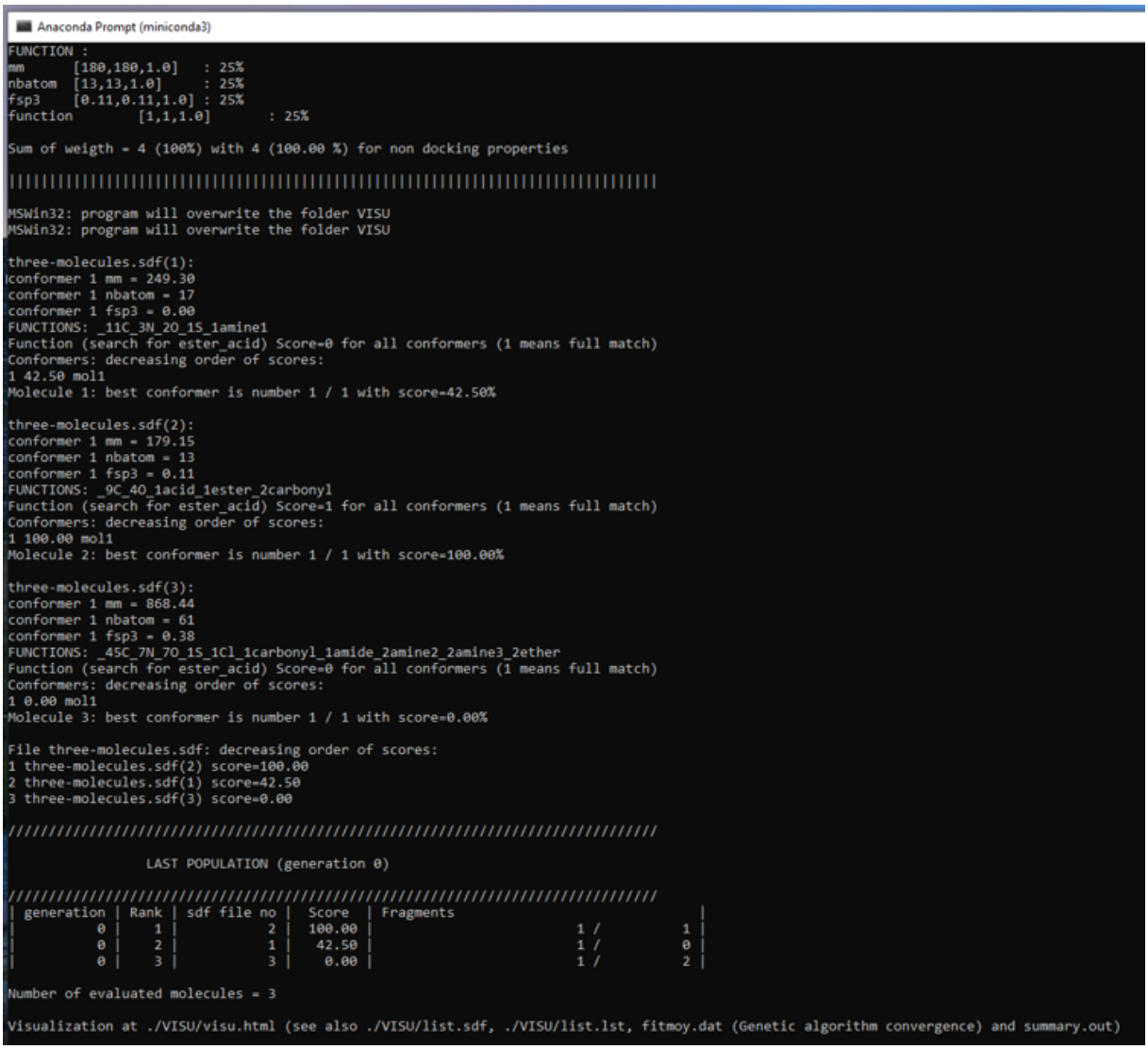
c) Use lea3d to evaluate molecules using a fitness function

In this example, the objective is to use the program to evaluate the fitness function of a set of molecules. The SDF file of molecules is given as input and the file ligand-aspirin.in indicates the fitness function to use for the evaluation (here, ligand-aspirin.func).

Execute:

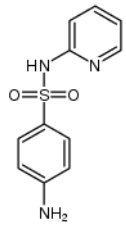
```
(lea3d) > perl ../LEA3D-main/MAIN  -e  ligand-aspirin.in  three-molecules.sdf
```

The output is written on the screen:



As indicated, molecule number 2 in the sdf file has a score of 100%. This was expected as the aspirin itself is the second molecule of the file three-molecules.sdf. The file summary.out indicates the score of screened molecules ranked in descending order of score.

C - how molecular structures are represented in LEA3D



e.g:
Sulfapyridine

Here:
legos = fragments
X = dummy atom that indicates where the fragment was originally substituted

X-pyridine	ring (lego 1 (sdf number 1 during the run, see below))
X-phenyl-X	ring (lego 2 (sdf number 2 during the run, see below))
NH2-X	substituent (lego 4 (sdf number 1206 during the run, see below))
X-S(=0)(=0)NH-X	linker (lego 3 (sdf number 913 during the run, see below))

In LEA3D, the syntax of the molecule looks like:

1*5-3*4_3*1-2*3_2*6-4*1 / 1 2 913 1206

Where the first part before '/' indicates how to combine the list of fragments indicated after '/'.
Thus, here:

lego 1 at the heavy atom 5 will be linked to lego 3 at the heavy atom 4
then lego 3 at the heavy atom 1 will be linked to lego 2 at the heavy atom 3
then lego 2 at the heavy atom 6 will be linked to lego 4 at the heavy atom 1

lego 1 is the fragment number 1 in the uploaded file of fragments
lego 2 is the fragment number 2 in the uploaded file of fragments
lego 3 is the fragment number 913 in the uploaded file of fragments
lego 4 is the fragment number 1206 in the uploaded file of fragments

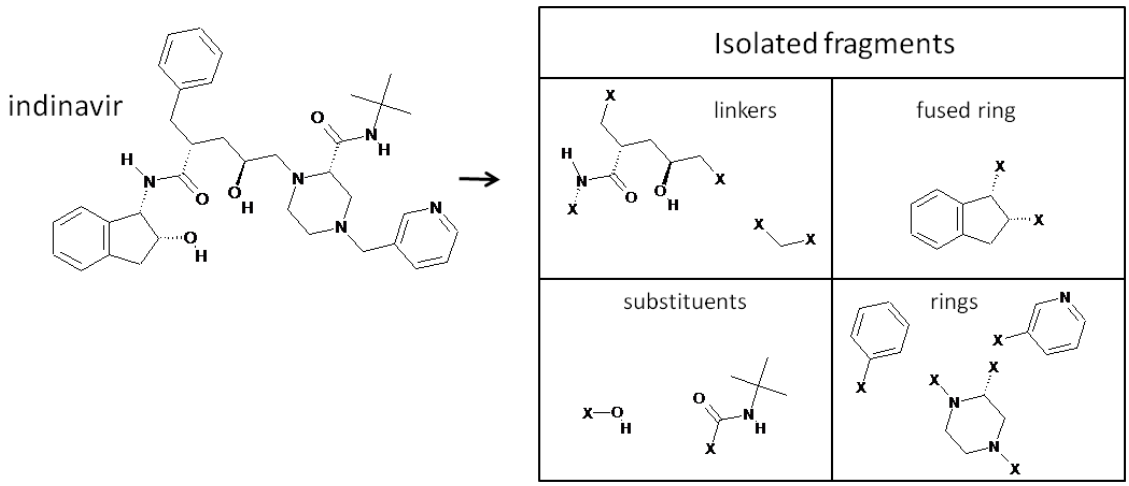
a) The order of the uploaded fragments from file(s) **is important**. The reading is sequential as it follows the order of the uploaded sdf files in ligand.in after the tag 'BASES'. Thus, if you change libraries (or their ordering) then the molecular building of a same syntax will produce a different structure.

b) The order in the first part of the encoding is important because a chaining must exists:
For example this writing will not work: 1*5-3*4_2*6-4*1_3*1-2*3 / 1 2 913 1206
Because after building the first assembly (linking fragment 1 and 3), then, it is impossible to add the fragments number 2 (2*6-4*1) because fragment number 2 and 4 are not present in the first assembly. The next fragment must be fragment 1 or fragment 3.
The following one will not work either because fragment number 2 is not yet in the first assembly (even if it will be linked to fragment number 3):
1*5-3*4_2*3-3*1_2*6-4*1 / 1 2 913 1206

D – how to create fragment files

In this minimal version of LEA3D, the folder lead3d-mklego contains tools to prepare fragments using an input sdf file of molecules to split.

The fragmentation of the drug Indinavir is given as an example:



'X' dummy atoms retain the substitution pattern of the original molecule. In the resulting sdf file, the <POINTS> data block contains the number of the heavy atom that is connected to the 'X' dummy atom. The 'X' dummy atom tags specifically the atom to be replaced that is useful in case of an asymmetric carbon to retain the configuration.

a) Prepare the input SDF file

Molecules to fragment must be in 3D. You can use RDKit to convert the input file as follows:

(lea3d) > python ../LEA3D-main/rdkit-confs.py input.sdf nb-conformers output3D.sdf

Where nb-conformers sets the requested number of conformers.

eg:

(lea3d) > python ../LEA3D-main/rdkit-confs.py three-molecules.sdf 1 output3D.sdf

b) Fragment the input SDF file

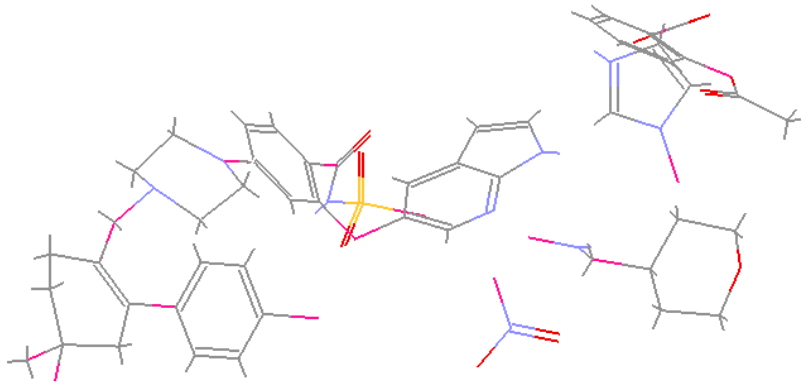
Using five molecules as input (see the file five-molecules.sdf), the script make-lego.py extracts 17 fragments which can be used in LEA3D. This script removes 2D structures (eg: molecule number 5 in five-molecules.sdf) then, dissociates salts (eg: molecule number 3 in five-molecules.sdf) then, fragments molecules, removes fragments without X dummy atoms (those cannot be substituted; eg: molecule number 4 in five-molecules.sdf) and, finally, removes duplicates.

Execute:

(lea3d) > python ../LEA3D-main/lea3d-mklego/make-lego.py five-molecules.sdf

Visualize generated fragments:

(lea3d) > pymol lea3d-legos.sdf



You can copy the file lea3d-legos.sdf in the directory LEGO and/or write the pathway to upload them in ligand.in under the tag “BASES”.

Fragment files contains data blocks for each fragment for classification purpose (if necessary):

> <ncycles>
Indicates the number of rings

> <natomescycle>
Indicates the number of ring atoms

><cyclearom>
Indicates the number of sp2 atoms in rings

><nbringar>
Indicates the number of aromatic rings

><natomes>
Indicates the number of atoms

><nrotatable>
Indicates the number of acyclic single bonds

><nhetat>
Indicates the number of heteroatoms

> < ARcenter>
Indicates the coordinates of the center of mass of the fragment (here aromatic ring system)

> <LIPcenter>
Indicates the coordinates of the center of mass of the fragment

Of note, <natomes> is a data block that is attributed to acyclic fragments, <natomescycle> is a data block that is attributed to cyclic fragments, <LIPcenter> is a data block that is attributed to non aromatic fragments and < ARcenter> is a data block that is attributed to aromatic fragments.

c) Miscellaneous tools

* Replacing 'X' dummy atoms in a SDF file

(lea3d) > python ../LEA3D-main/lea3d-mklego/replacex.pl file.sdf

It replaces 'X' dummy atoms by a hydrogen 'H'. Indeed, most computational programs require organic atoms and 'X' dummy atoms are not recognized. The old file file.sdf is replaced by the new one.

For example, copy lea3d-legos.sdf into test.sdf and then execute:

(lea3d) > perl ../LEA3D-main/lea3d-mklego/replacex.pl test.sdf

lea3d-legos.sdf contains 'X' dummy atoms but test.sdf does not.

* Reset or update the datablocks <POINTS> of fragments

(lea3d) > python ../LEA3D-main/lea3d-mklego/getx.pl file_fragments.sdf

It reads and rewrites the datablock <POINTS> in a new sdf file called new_file_fragments.sdf by using the number of the heavy atoms that are connected to 'X' dummy atoms.

E – Parameter Setting

Usage can be obtained using:

(lea3d) > perl ../LEA3D-main/MAIN -i

Whatever the option (no option (= *de novo* design), -e (evaluation only) or -v (visualization only)), LEA3D requires 2 input files: an input file .in (eg: ligand.in), and a function file .func (eg: ligand.func). One example of each is present in the installation directory (here, LEA3D-main).

File ligand.in

The .in input file contains the genetic algorithm parameters (number of individuals, number of generations...) and more general parameters (conformer generation …). It must contain the name of the function file .func (ligand.func for example). Default parameters are set. '#' means the line is commented. **A more detailed description is given for each parameter in the file ligand.in.**

In this LEA3D core version, two options are not available : calculation of partial charges (set CHARGE 0) and the druglike filtering (set FILTER 0).

File ligand.func

The function file contains the evaluation function composition to calculate the user-defined composite score. Functions may be physicochemical properties, an affinity prediction… '#' means that the line is commented. Whatever the property, the first fourth fields are set as following:

<function_name> <lower limit or '-'> <upper limit or '–'> <weight> .....

Depending on the program used to calculate the score, additional parameters may be required. Those can be added after the fourth field <weight>.

The following table explains how to fill the second and third field when one wants to set a lower limit, an upper limit, an exact value or a range for the selected property. Of note, **the weight of a constraint must be > 0.0 in order to be included** (fourth field). Each property contributes to the final score proportionally to the assigned weight. The last is transformed in percentage ($weight_i$ / ($\sum_i weight_i$)) with i a property.

Example 1: set the minimal value only				
Molecular weight	100	-	1.0	Means MW must be ≥ 100
Example 2: set the maximal value only				
Molecular weight	-	469	1.0	Means MW must be ≤ 469
Example 3: set the minimal and the maximal value with the same value				
Molecular weight	100	100	1.0	Means MW must be exactly 100
Example 4: set the minimal and the maximal value with different values				
Molecular weight	50	469	1.0	Means MW must be ≥ 50 and must be ≤ 469

In this LEA3D core version, a minimal set of properties is offered: molecular weight, fsp3, logp, radius of gyration, moment of inertia ix and iy (iz=1), length, nbhd (number of hydrogen bond donor), nbha (number of hydrogen bond acceptor), nbatom (number of heavy atoms) and functional groups searches. **A more detailed description is given for each property in the file ligand.func.**

* Syntax of the chemical function property:

function 1 1 0.0 <function1>_<function2>

The lower and upper limits are always '1'. The list of the request chemical functions is indicated at the fifth position. The separator between 2 chemical functions is '_'. Each chemical function has a proportional weight (the total weight is divided by the number of chemical functions). The searchable chemical functions are:

acid ester carbamate amide amide-ter aldhehyde keto amine amine1 amine2 amine3 alcohol alcohol1 alcohol2 alcohol3 ether thiol carbonyl C O N S P F Cl Br I

amine1 refers to a primary amine, amine2 refers to secondary amine, amine3 refers to tertiary amine and amine refers to any type of amine.

alcohol1 refers to primary alcohol, alcohol2 refers to secondary alcohol, alcohol3 refers to tertiary alcohol and alcohol refers to any type of alcohol.

F – How to customize the fitness function

To add a new function to evaluate, three files must be modified: ligand.func, MAIN and SCORE.pl.

* File .func

In the file .func, the new function must follow the format described above:

> <function_name> <lower limit or '-'> <upper limit or '–'> <weight> <(additional parameters if required)>

with the weight put as the fourth field.

* File MAIN

Modify the main program MAIN in order to add the new function in the list. Follow the example of "shape" or function for example.

- Modify the table %prop=() to add the new property
- Create a binary flag $evaluate_xxx (as "evaluate_shape" for example). Sometimes additional variables must be set in order to be passed to the program that will calculate the property (as variable $shape_ref and $shape_score). In such case, they can be read from .func file (<additional parameters>)

* File SCORE.pl

Modify the file SCORE.pl to add the function in the scoring. It is activated only if evaluate_xxx is set to 1.

Here, 2 tables must be filled with results of the calculation: @score and @properties. Then, the main program MAIN uses both tables to rank molecules and print outputs.