Fafoom -Flexible algorithm for optimization of molecules

October 6, 2015

Contents

1 An overview of Fafoom

Fafoom is a Python module for optimization of organic molecules primarily intended to work with FHI-aims (Fritz Haber Institute ab initio molecular simulations package). Fafoom can be utilized for, e.g., performing a genetic algorithm (GA) search for molecules. The genetic operations (crossover and mutation) explore the fitness function (energy) by changing the degrees of freedom of the molecule. Three sorts of degrees of freedom are currently implemented: torsions, *cis/trans* bonds and pyranose ring configurations. Further, user-defined degrees of freedom can be implemented. Fafoom:

- initializes the molecule from a SMILES code
- performs the global search on user-curated selection of degrees of freedom and performs local optimization based on full relaxation of Cartesian coordinates
- uses for the local optimization an external software that can be easily exchanged
- is distributed under GNU Lesser General Public License

Fafoom uses eV as a unit for energy. If the utilized software outputs energy in different units, the value is converted to eV after completed local optimization.

2 Requirements

For the python module:

- Python (used for testing: 2.7.6)
- Numpy (used for testing: 1.8.2)
- RDKit (used for testing: Release_2015_03_1)

For the first-principles methods:

- (recommended) FHI-aims (Fritz Haber Institute ab initio molecular simulations package)
- (alternative) NWChem (NWChem: Open Source High-Performance Computational Chemistry)
- (alternative) ORCA (An ab initio, DFT and semiempirical SCF-MO package).

3 Installation

1. Clone the fafoom repository

   ```
   git clone https://github.com/adrianasupady/fafoom
   ```

   or download and unzip the version from :

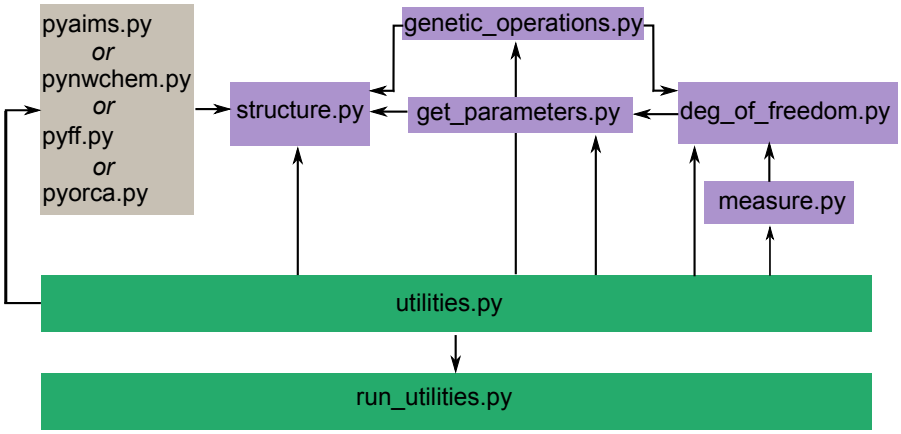
```
https://github.com/adrianasupady/fafoom/releases
```

2. Add the fafoom directory to your PYTHONPATH
3. Import the module in python

```
import fafoom
```

4 Module overview

Figure 1: Flexible algorithm for optimization of molecules Fafoom: Module overview



- *structure.py* - the core module of the package, defines two classes: (i) **MoleculeDescription** that takes care of initializing the molecule and the attributes, that are shared between different 3D structures and should never change their values, i.e. number of atoms or number and location of the user-defined degrees of freedom and (ii) **Structure** that allows for generating valid and unique 3D structures, calling the external optimizer or genetic operators; further, it takes care of the varying attributes, e.g. the sdf string or values of the degrees of freedom.

- *get_parameters.py* - this module is responsible for obtaining the permanent attributes. It also serves as a link between the *structure.py* and the *deg_of_freedom.py* module.

- *deg_of_freedom.py* - this module defines a separate class for each type of degree of freedoms. The methods of the classes allow for finding the degrees of freedom, assigning, mutating and updating their values as well as generating corresponding sdf strings.

- *measure.py* - collection of methods for measuring and setting the values of the degrees of freedom

- *genetic_operations.py* - collection of genetic operations needed to perform a genetic algorithm
- *pyaims.py* - wrapper for the FHI-aims simulation package; takes care of performing the local optimization and retrieving and storing the results
- *pynwchem.py* - wrapper for NWChem
- *pyorca.py* - wrapper for ORCA
- *pyff.py* - wrapper for force field calculations performed with RDKit
- *utilities.py* - collection of diverse help functions: vector operations, conversions between chemical formats, writing to files etc.
- *run_utilities.py* - collection of functions that can be used for performing a genetic algorithm run; i.e detecting the variant of the calculation or checking for convergence

5 Getting started

The provided example parameter file can be used to run a GA search for alanine dipeptide (Figure 2). Note, that the SMART pattern definitions (smart_torsion, smart_cistrans and filter_smart_torsion) are adjusted for this systems so that the peptide bond is treated in a *cis/trans* mode. Note that the division into sections is required for the parameter file.
For the meaning of the parameters and available options get familiar with section **Keywords**. Backup text files for the restart are created after each completed iteration. The details about the run are written to `output.txt`.

6 Example: Genetic algorithm search

The GA can be started with:

```
python ga.py parameter_file
```

The fafoom package will perform the genetic algorithm and call the selected software for local optimization of the 3D structures. The parameter file needs to define the provider of the energy function to be used ("FHI-aims" and "NWChem" for first principles, "RDKit" for force fields). Depending on the selected software, further keywords need to be specified (check the **Keywords** section).

6.1 GA with FHI-aims

Before running the algorithm, user needs to create a directory containing the `control.in` file that will be used for the FHI-aims calculations. One new directory is created for each FHI-aims calculation. Take a look at pyaims.py and adjust it for your needs.

```
[Molecule]

smile= "CC(=O)N[C@H](C(=O)NC)C"
optimize_torsion= True
optimize_cistrans= True
smart_torsion= "[C,N,O]~[!$(*#*)&!D1]-&!@[!$(*#*)&!D1]~[C,N,O]"
smart_cistrans= "C~[$(C=O)]-[$(NC)]~[C]"
filter_smart_torsion= "C~[$(C=O)]-[$(NC)]~[C]"
rmsd_type= "cartesian"
distance_cutoff_1= 1.2
distance_cutoff_2= 2.15
rmsd_cutoff_uniq= 0.25
chiral=True

[GA settings]

energy_var= 0.001
selection= "roulette_wheel"
fitness_sum_limit= 1.2
popsize= 10
prob_for_crossing= 0.95
prob_for_mut_cistrans= 0.6
prob_for_mut_torsion= 0.8
max_mutations_cistrans= 1
max_mutations_torsion= 2


[Run settings]

max_iter= 30
iter_limit_conv= 20
energy_diff_conv= 0.001

### FHI-aims
energy_function= "FHI-aims"
sourcedir= "adds"
aims_call= "mpirun -n 4  aims.071914_7.scalapack.mpi.x"

### for NWChem:
energy_function= "NWChem"
functional= "xpbe96"
basis_set= "STO-6G"
nwchem_call= "mpirun -n 4 nwchem"

### RDKit:
energy_function= "ff"
force_field= "mmff94"
steps= 1000
force_tol= 1e-04
energy_tol= 1e-06

### ORCA:
energy_function= "orca"
commandline= "opt pbe nopop miniprint"
chargemult= "0 1"
optsteps = 200
nprocs= 4
memory= 4000
orca_call= "/full/path/to/orca/"
```

Figure 2: **parameters.txt**

6.2 GA with NWChem

If you want to modify the geometry optimization settings, adjust the pynwchem.py file.

6.3 GA with RDKit

RDKit allows for performing local optimization with following force fields: UFF and MMFF94. The convergence settings for the geometry minimization with the force field can be set in the parameter file. The optimized structures will be written to a file: *optimized_structures.sdf*.

6.4 GA with ORCA

Details concerning the geometry optimization with ORCA can be found in following chapters of the ORCA manual: **Running Typical Calculations: Geometry Optimizations, Surface Scans, Transition States, MECPs** (5.2) and **Detailed Documentation: Geometry Optimization** (6.13).

6.5 Restart

The algorithm can be restarted if at least one generation has been successfully completed. The information transfer happens via the backup files that are generated after a completed generation. During the restart, the population and the blacklist are rebuild. If all necessary backup files are present, the restart can be started directly with:

```
python ga.py parameter_file
```

7 Keywords

This section lists all recognized keywords. Some of them are depended on each other and on the selected system, e.g. 'optimize_cistrans' will cause no effect unused is no *cis/trans* bonds are found. The absolutely required keywords are underlined.

Molecule settings

- **smile**

  Simplified one-line notation (SMILES) of the compound you want to perform the search for.

- **optimize_torsion**, default = True

  Set True, if you want the torsions of be among the optimized degrees of freedom.

- **optimize_cistrans**, default = False

  Set to True, if you want the *cis/trans* bonds of be among the optimized degrees of freedom.

- **optimize_pyranosering**, default = False

  Set to True, if you want the pyranosering bonds of be among the optimized degrees of freedom.

- **smart_torsion**, default="[*]~[!$(*#*)&!D1]-&!@[!$(*#*)&!D1]~[*]"

  Pattern for matching torsions.

- **smart_cistrans**

  Pattern for matching *cis/trans* bonds.

- **filter_smart_torsion**

  The pattern defined here will be used to match torsions you want to ignore.

- **list_of_torsion**, **list_of_cistrans**, **list_of_pyranosering**

  If you know the positions of torsions/ *cis/trans* bonds/ pyranoserings to be optimized, you can pass them directly as a list of tuples with 4 atoms each. The numbering is consistent with the numbering of the atoms in the SMILES string.[1]

- **distance_cutoff_1**, default = 1.3 Å

  Parameter for the geometry check. If two non-bonded atoms are closer to each other than **distance_cutoff_1** (Å) the structure will be rejected.

- **distance_cutoff_2**, default = 2.15 Å

  Parameter for the geometry check. If two bonded atoms are further from each other than **distance_cutoff_2** (Å) the structure will be rejected.

- **rmsd_type**, default = "cartesian"

  You can decide between *cartesian* and *internal_coord* RMSD to be used for distinguishing between similar and different structures. If *cartesian* is chosen, the GetBestRMS RDKit routine will be used for calculating the RMSD between two structures. The *internal_coord* RMSD will compare directly the values of degrees of freedom between two structures. The *internal_coord* RMSD might be quicker than the *cartesian* RMSD, but is not symmetry corrected. However, you can adapt the get_vec function (in the utilities module) for your needs.

- **rmsd_cutoff_uniq**, default = 0.2 Å

  This parameter is used for blacklisting. A new structure is considered to be unique if it has an RMSD to all already existing structures higher than the **rmsd_cutoff_uniq**. If you set the threshold to 0.0 all structures will be treated as unique.

- **chiral**, default = True

  If set to False, not only the structure but also its mirror image will be used for comparisons.

[1] The first atom in the SMILES has the index 0. Hydrogen atoms don't get a number first, but only after adding all hydrogen atoms to the molecule.

- **weights_torsion**, **weights_cistrans**, **weights_pyranosering**

  Keywords for assigning weights for the options for the degrees of freedom. Example of use: For the *cis/trans* bonds, there are only two options: 0° and 180 °. Normally, the algorithm will try to generate both kinds of structures. However, if one wants to generate and optimize conformations only with *trans* bond (i.e. equal to 180°), it is possible with: **weights_cistrans**= [0., 1.]

GA settings

- **popsize**, default = 10

  Size of the initial pool of structures.

- **energy_var**, default = 0.001 (eV)

  If the difference between the highest and lowest energy in the population is lower than the **energy_var**, all the individuals will be assigned the same fitness of 1.0.

- **selection**, default= "roulette_wheel"

  Options for the selection mechanisms of the individuals. Another options are random and roulette_wheel_reverse.

- **fitness_sum_limit**, default = 1.2

  If the sum of the fitness values for all individuals is lower than this threshold the selection will be conducted independently from the chosen mechanism. The best and a random individual will be selected.

- **prob_for_crossing**, default = 1.0

  Probability for the crossing over.

- **prob_for_mut_torsion**, **prob_for_mut_cistrans**, **prob_for_mut_pyranosering**

  Probability for a mutation in torsions/ *cis/trans* bonds/ pyranose rings (active only if the corresponding optimize_torsion/ optimize_cistrans/ optimize_pyranosering = True).

- **max_mutations_torsions**, **max_mutations_cistrans**, **max_mutations_pyranosering**

  Maximal number of mutations for torsions/ *cis/trans* bonds/ pyranose rings. A random number between (1, **max_mutations_***) of mutations will be performed (active only if corresponding optimize_torsion/ optimize_cistrans/ optimize_pyranosering = True).

Run settings

- **energy_function**

  Name of the software/method to be used for the energy evaluations. Currently supported options are: FHI-aims, NWChem, RDKit and ORCA.

- **max_iter**, default = 30

  Number of iterations that will be performed after the initialization is finished.

- **iter_limit_conv**, default = 20

  Minimal number of iterations to be performed before any convergence criteria are checked.

- **energy_diff_conv**, default = 0.001 (eV)

  Parameter for checking the convergence. If the lowest energy hasn't change by more than **energy_diff_conv** (eV) after **iter_limit_conv** iterations, the GA-run is considered to be converged. Attention: convergence doesn't necessarily mean that the global minimum was found.

- **energy_wanted**

  If the energy of the global minimum is known it can also be used for checking if the convergence is achieved.

- ## FHI-aims related keywords

  1. **sourcedir**
     Name of your directory with control.in file.
  2. **aims_call**
     String for execution of FHI-aims.

- ## NWChem related keywords

  1. **functional**
     Functional to be used.
  2. **basis_set**
     Basis set to be used.
  3. **nwchem_call**
     String for execution of NWChem.

- ## RDKit related keywords

  1. **force_field**
     Name of the force field to be used.
  2. **steps**, default = 1000
     Number of steps for the minimization.
  3. **force_tol**, default = 1.0e-4
     Force tolerance.
  4. **energy_tol**, default = 1.0e-6
     Energy tolerance.

- ## ORCA related keywords

1. **commandline**
   A string for the ORCA input file defining the optimization type, e.g. "opt pbe".
2. **memory**
   Memory limit (in MB) per processing core.
3. **orca_call**
   Full path to the ORCA executable.
4. **chargemult**, default = "0 1"
   The first value denotes the total charge, the second the spin multiplicity.
5. **optsteps**, default = 500
   Max. number of iterations during one geometry optimization.
6. **nprocs**, default = 1
   Number of processors to use.

8 General advice

- (if FHI-aims is used) take your time to construct and test a reasonable control.in file
- be careful when adjusting the **distance_cutoff** parameters
- adjust the **smart_torsion** and **smart_cistrans** to your needs; check if the recognized torsions are fine for you

9 How to: use software of your choice for the local optimization

Currently, fafoom supports four software packages: FHI-aims, NWChem, RDKit and ORCA. Each of them has a dedicated wrapper: *pyaims.py*, *pynwchem.py*, *pyff.py* and *pyorca.py*. If you want to use another software, follow these steps:

1. Get familiar with the code of the existing wrappers. Note that the molecular structure that is passed to the wrapper is in the SD-Format (sdf string). The structure resulting from the local optimization, can be of any format first, but it needs to be converted to a SD-Format for the algorithm to proceed. Further, the wrapper needs to have a method for returning the energy of the structure after the local optimization. If needed, the energy value needs to be converted to eV.

2. Take a look at the e.g. **perform_aims** or **perform_orca** methods of the **Structure** class. They call the particular functions of the wrappers in a certain order to perform the local optimization and assign the new attributes to the structure. They also take care of the update of the values of the degrees of freedom.

3. Write your own wrapper and test its behaviour.

4. Add a "**perform_*yourmethod***" method to the structure class. Don't forget to add your new wrapper to the list of imported modules.

5. If your are planning on you using the provided *ga.py* script together with the parameters.txt file, few more adjustment are required. The *ga.py* reads from the parameter file the value of the keyword "**energy_function**" and performs the calls than the corresponding software. Take a look at the **detect_energy_function** in the *run_utilities.py* module. You can extend it via an elif clause. Finally you need to add another elif clause in the optimize method in the run_utilities.py with the direct call to the "**perform_*yourmethod***" from step 3.

10 How to: define your own degrees of freedom

During the initial parametrization of the molecule, fafoom identifies the kind and the 'position' of the degrees of freedom that the algorithm should take care of. All operations that involve the degrees of freedom rely on the iteration of the identified degrees of freedom. With this, the algorithm remains flexible, and all the logic concerning the particular degree of freedom can be collected in a dedicated class. If you want to add a new degree of freedom follow these steps:

1. Take a look at the *deg_of_freedom.py* module and how the particular classes are implemented.

2. The static method 'find' is required for each degree of freedom as the MoleculeDescription class requires it for the parametrization of the molecule (it is done only once per molecule).

3. Apart from ___init___ and the find method, the class for a new degree of freedom need to have following methods: get_random_values, apply_on_string, update_values, mutate_values and is_equal.

4. Once the new class is implemented, you need to adjust two methods in the *get_parameters.py* module: **get_positions** and **create_dof_object**. For each, add an elif clause, so that fafoom can recognize the new degree of freedom.

5. If there are new keywords needed, insert them into the 'Molecule' section of the parameter file. All of keywords declared there are automatically assigned as attributes to the molecule.

11 Ongoing

- symmetry correction in the torsional RMSD
- optimization of shared blacklisting

12 Acknowledgement

Mateusz Marianski (FHI Berlin) is kindly acknowledged for providing a function for sampling diverse six-membered ring conformations.

Philipp Traber (FSU Jena) is kindly acknowledged for providing the wrapper for ORCA (An ab initio, DFT and semiempirical SCF-MO package).