Molecule folder

Georgios Samaras
Department of Informatics and Telecommunications
University of Athens

September 4, 2014

Abstract

The purpose of this project is to generate new configurations of a given molecule, based on a number of rotations, by a certain angle interval (select randomly an angle in $[-\theta, \theta]$ for each rotation). This project is a follow up of the comparison of Nearest Neighbors Libraries for the final project of the course of Bioinformatics.

1 Introduction

The need of more data was what initiated this project. We had some data, to test our implementation of the paper "Fast and Reliable Analysis of Molecular Motion Using Proximity Relations and Dimensionality Reduction ", which Vangelis Anagnostopoulos and me, Georgios Samaras studied for the needs of the Bionformatics course of our university, but they were not enough challenging for the algorithm, in terms of size.

The idea is to use random rotations, between the atoms, so that the configuration of the molecule will change, thus a new configuration is generated. However, there are certain limitations to these rotations. Moreover, the new configuration has to be a configuration that is sane. In other words, it should not only something that we generated, but something we could really find in nature, a configuration that the molecule could reach naturally.

2 Rotation about an arbitary axis

Our first approach was to locate a certain group of atoms (their first occurence) and focus on them about the rotation. The rotation would then be done about an arbitary axis, that would be chosen randomly (following a uniform distribution), by an angle that would be an input parameter.

The group of atoms would consist of the following: C,C_a, O, N, C_b and C_c. These atoms form two groups. We keep the one group static and we rotate the other one, by an angle x_1, as the following picture shows:

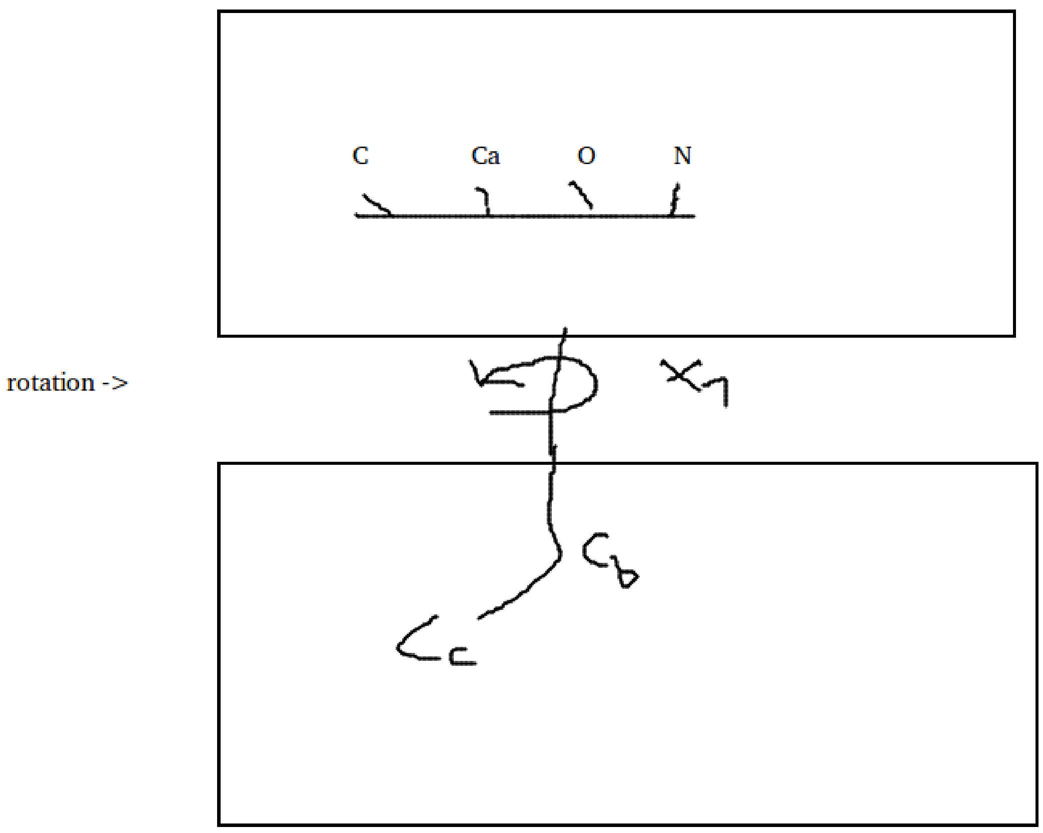

Fig. 1 - Rotate one of two groups, by an angle of x_1.

This approach was easy to implement and the code was fast. We generated about 1080 new configurations, based on the "abeta_trajectory" file, which was kindly provided by Mrs. Chrysina. However, when we eyeballed some of the configurations with Coot or PyMol, the differences were so small, that we had to zoom again and again until we spotted them. Sometimes, the configurations seemed identical, even though the rotation was applied.

Our thought was that we rotate once (affecting only some atoms), based on two groups that we find without any significant effort and that the rotation was perfomed about an arbitary axis. That lead us to a new approach, presented in next section.

3 Rotations about bonds

Our second approach was to perform a number of rotations, about the bonds, by an angle which would be randomly chosen in an integral given by the user. To be specific the bonds would allow us to decide which atoms can be used as rotation centers.

The code written, is divided in these sections:

1. The configuration file
2. Determining bonds
3. Finding the atoms suitable to be used as rotation centers
4. Transformation matrix

This approach produces better results than the approach discribed above, but it's far more slow. However, we believe that the trade-off is certaintly fair, since the datasets are going to be created once.

Notice that for this approach, we used as a starting point, the "2LFM.pdb" file, since this file is downloaded from PDB and is for sure, correct.

Below, we used a configuration of the "2LFM.pdb" file as a starting point and performed 1000 rotations, with an angle interval of $[-15, 15]$. That means that the 1st rotation will pick, at random, a rotation center and a random value in $[-15, 15]$, which will be the rotation angle and perform one rotation. The rest of the rotations follow the same logic.

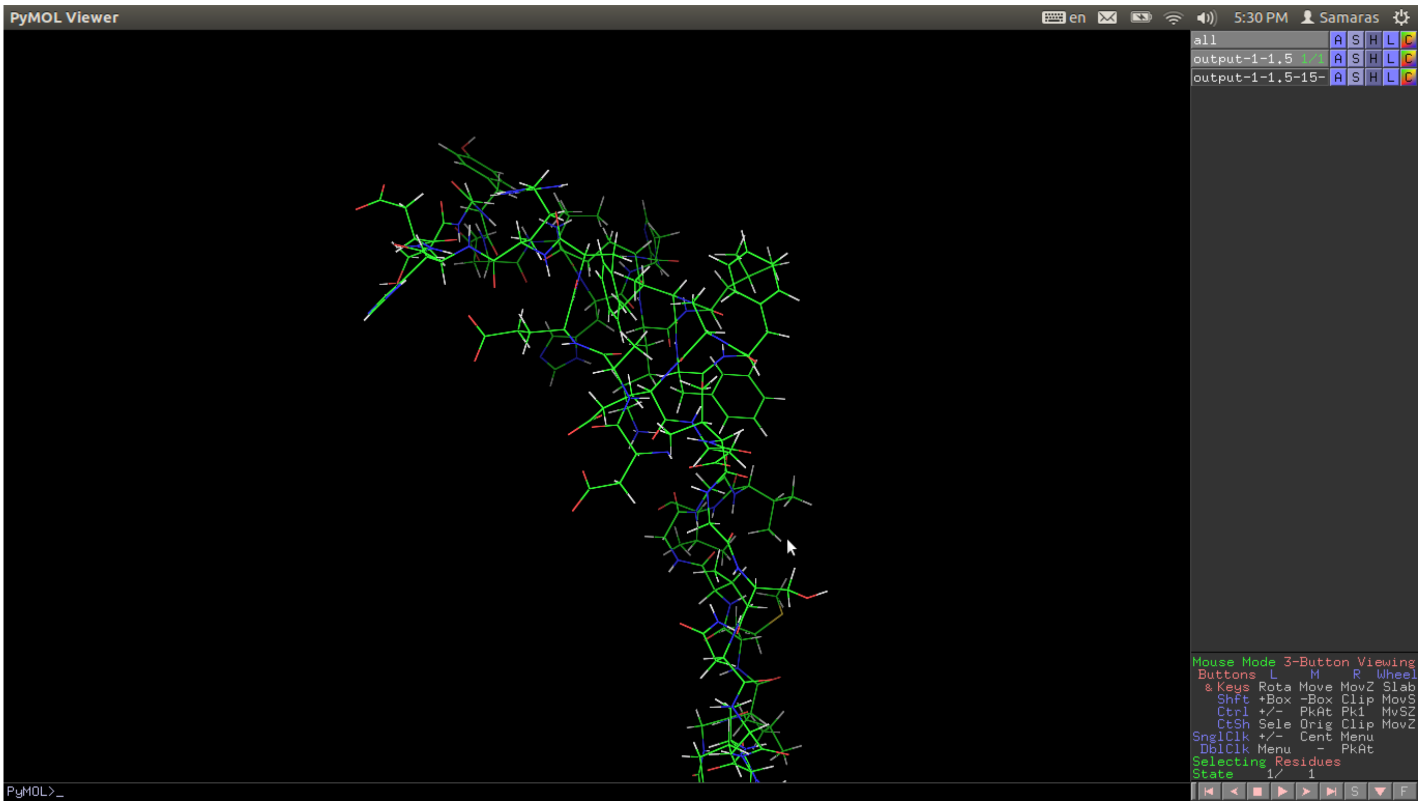

Fig. 2 - Starting configuration.

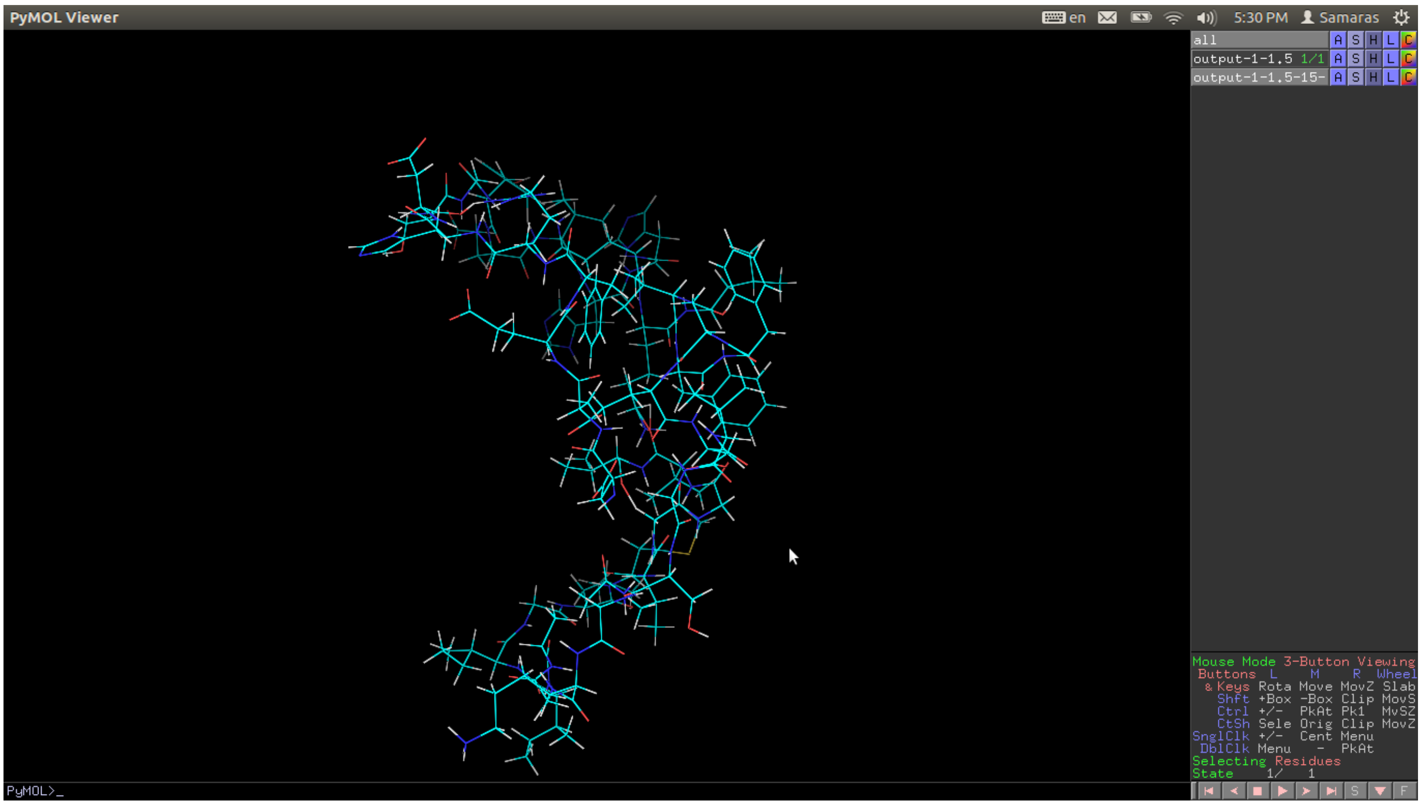

Fig. 3 - Generated configuration.

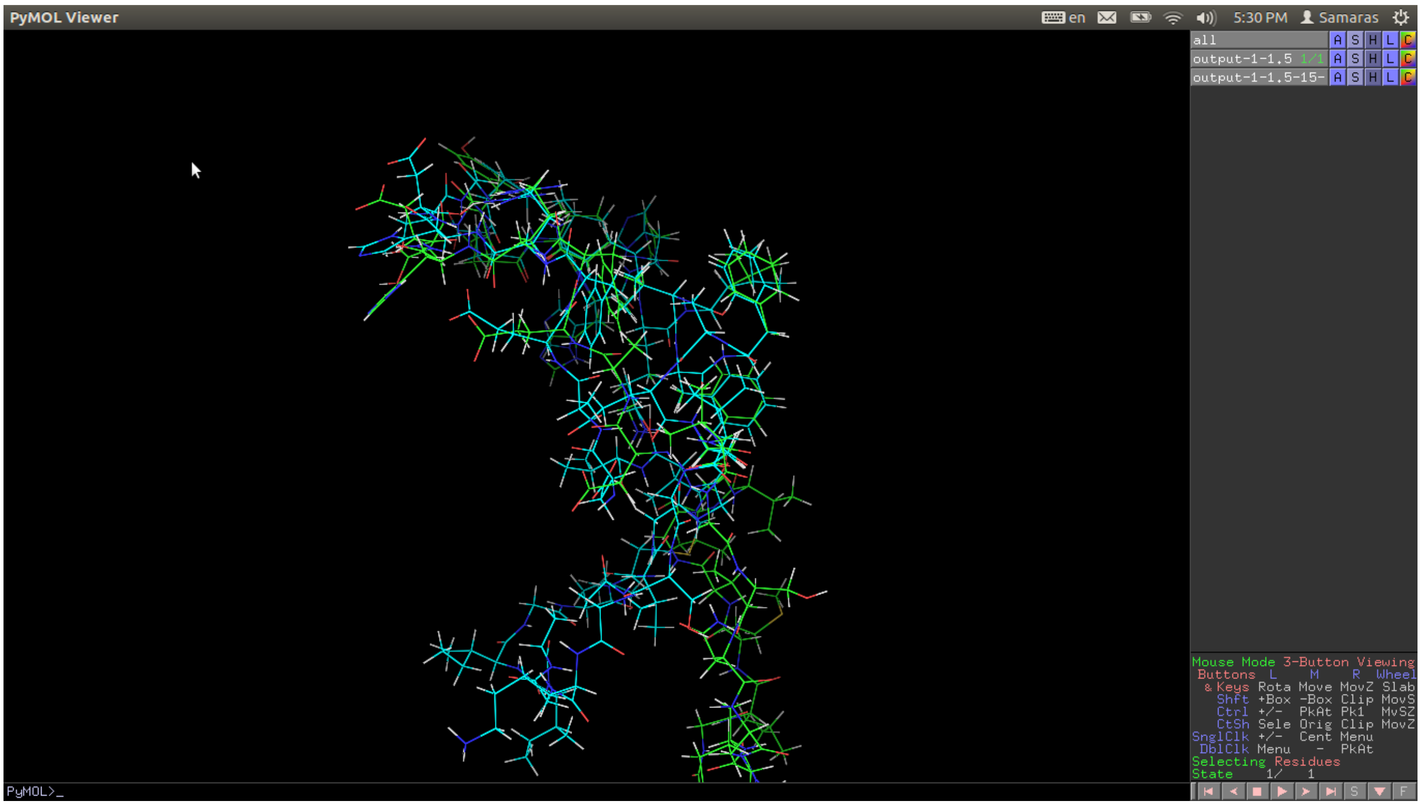

Fig. 4 - Starting configuration(green), superimposed with the generated configuration(aqua).

We generated 10000 new configurations, in 42 minutes. However, some of them may be not so different from the others. Below, we will analyze how this project works.

3.1 The configuration file

The configuration file is restricted to the 'configure()' function. It does not use getline() (or fgets()), because we wanted it to be able to read files with any newline convention – this is important, if the files are edited in e.g. Notepad in Windows (or some other program in Mac OS X).

The configuration file defines two arrays explicitly: the bond lengths array, and the atom configuration array. It also defines a third array implicitly: the elements. Since elements are referred to using names (between one and three characters long, from H for hydrogen to UUo for Ununoctium), and we do want to use the element names (atomic symbols) instead of some obscure numbers, we need a mapping.

The function 'use_element()' takes the symbol name as a string (up to 7 characters, here), and searches for it in the current elements array. If found, it returns the corresponding element number. Otherwise, the new element symbol is added to the array, and the new element number is returned. If the element array is already full, the function returns -1, which also causes the 'configure()' function to fail with an error message.

The line read(coming from the file) is expected to have the format:

```
ATOM symbol xcoord ycoord zcoord ignored...
```

with "symbol" having at most 7 characters (the array needs one more, 8, to accommodate the end-of-string NUL byte). Symbol lookup is done using 'use_element()'.

Atom array is dynamically resized as needed. Our resizing algorithm starts with the pointer being NULL, and allocates atoms to next multiple of 512. The important point is to reallocate the array before we try to use the entries, and that reallocating for every single atom would be slow; we're better off allocating a chunk at a time. (How large a chunk, depends! If too large, we waste memory; if too small, the reallocations slow down the code. It is a matter of balance.). Notice that the array could be replaced by a vector.

As a matter of fact, we first created a program which would determine the bonds and write them into a file. As a result, we took care of this case too. The BOND line syntax is very similar to ATOM lines:

BOND symbol1 symbol2 length

and handled in the same way, with one exception: Each BOND line actually defines TWO entries in the array:

bond_squared [number1][number2] = length;
bond_squared [number2][number1] = length;

so that we can just refer to the array. Notice that we could use the half size of the array $((N*(N-1))/2$ entries = $N*N/2-N/2$ entries, currently the array is $N*N$ entries), but that would complicate indexing too much, in our opinion.

3.2 Determining bonds

As we mentioned, we provide a software to determine the bonds of a given molecule. The logic is similar to this step of this software, thus we do not describe it in another section.

Function 'compute_bonds()' does basically a double loop ($O(N^2)$) over the atoms, checking the distance from each atom to every other atom. If the distance is at most the bond length (for those two elements), then the atoms are considered bonded.

The way the double loop is done, it covers each pair just once (since$j \leq i$). If we draw a diagram, with i increasing right and j increasing up, then this loop covers only the lower triangular part, including the diagonal and positive i axis.

3.3 Finding the atoms suitable to be used as rotation centers

The function 'find_rotation_center_atoms()' implements the logic determining which atoms are suitable as rotation centers.

The logic is simple: If removing the atom would split the molecule into two (or more) parts, the atom is suitable to be used as a center.

We do this using the .group field of each atom. We use the field to paint the center atom as GROUP_CENTER, one of the atoms bonded to it as GROUP_ROTATE, and the rest of the atoms GROUP_STATIC. Then, we use the 'groupify()' function to "spread" the paint to each GROUP_STATIC atom bonded to a GROUP_ROTATE atom.

This causes the part of the molecule connected via bonds to the original GROUP_ROTATE atom we chose, to be painted GROUP_ROTATE. The center atom – or atoms, if we in the future choose to select more than one center atom, just paint them all with GROUP_CENTER – acts as a paint stopper, so the GROUP_ROTATE paint will not reach the center atom, or any atoms bonded to the center atom (unless the paint spreads via other bonds).

The result is the same as if we'd taken the molecule bonds, constructed a graph out of it, and then removed the atom we chose as a center. We get all atoms connected to the bonded atom we also picked, marked as GROUP_ROTATE. All other atoms remain GROUP_STATIC.

It would be more efficient to implement this using a real Disjoint Set datatype, If there were a lot of atoms. It turns out that with this few atoms, the data structures tend to fit in the cache. Moreover, notice that Disjoint Set datatype would avoid recalculations. Just doing the disjoint set once per atom would "paint" each separate part of the molecule (with the center removed separating them), and we could just find the combination of the paint counts that yields the closest to half the atoms. BOOST provides an API for Disjoint sets, that might come in handy for some future modifications.

Because the initial bonded atom choice affects which "limb" of the molecule we paint GROUP_ROTATE, we do this for each atom bonded to the original atom, and pick the one where the smaller region is largest. As a result, we pick the one which produces most nearly HALF the molecule.

In this approach we also used some limitations to our procedures. For example, each bonded atom should at least ROTATE_MINIMUM atoms. Currently ROTATE_MINIMUM is set to 4. Another significant limitation, is that when we check if a rotation is acceptable, we use several attributes listed above to determine if the rotated atoms come too close to the static ones. If so, the rotation is discarded. We decide whether these atoms are too close, based on their distance, compared with the length of the bond. We used squared values, for efficiency.

3.4 Transformation matrix

The function 'transform_center()' computes a transformation matrix, when given the center of rotation (index to center[] array), and the angle of rotation.

We calculate the sum of the bond vectors from the center atom. The bond vector is the vector from the center atom to the bonded atom, i.e. the vector difference of the atom coordinates.

If the vectors are symmetrically arranged in a plane, or symmetrically in a lattice on both sides of the atom, the sum of the bond vectors may be zero. (In practice, something much less than the shortest bond vector.). In that case, this approach to find a suitable rotation axis does not work.

Notice that we normalize the sum vector to length 1, suitable as the rotation axis. Then we compute the rotation matrix t, where:

$$\text{t} = \begin{bmatrix} X.x & X.y & X.z & T.x \\ Y.x & Y.y & Y.z & T.y \\ Z.x & Z.y & Z.z & T.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If we use homogenous coordinates for any point p,

$$\text{p} = [\ \text{x y z 1}\ ]$$

then we can rotate and translate p by computing

$$\text{p' = t p}$$

where T is the translation, (X.x Y.x Z.x) is the new X axis unit vector, (X.y Y.y Z.y) is the new Y axis unit vector, and (X.z Y.z Z.z) is the new Z axis unit vector.

Note that this is calculated based on the axis of rotation,

$$\text{axis, } |axis| = 1$$

and the amount of rotation in radians, 'radians'. The translation is such that the center atom is along the axis of rotation, and therefore does not move. The center atom is actually at the origin. As the axis of rotation must pass through origin, origin is the one point that always lies on the axis. Or, to put it in another way, origin never moves in a rotation.

3.5 Overall program

The program uses the configured, element-pair-specific bond lengths to determine whether each atom pair in the molecule is bonded or not. This is a simple $O(N^2)$ search. Notice that the time complexity can probably be improved.

Each atom in the molecule is tested, to see if it is a possible center of rotation. If removing the atom splits the molecule into two or more parts – meaning the separated parts only connected via bonds through that atom –, then the atom is a possible center of rotation.

This implementation tries to pick the set nearest to half the atoms in the molecule, by checking each bond starting from the original atom. The implementation is relatively inefficient – it is not significant since this is only done once at the beginning, and it takes at most a fraction of a second using the 2LFM.pdb data set even on a very slow computer – ,and a proper implementation using a Disjoint Set data type would avoid the unnecessary calculations. But, since this way is already sufficiently fast, it should do for this.

If the atom is suitable to be a center of rotation, the best set (resulting from checking each bond in turn) is saved into center[]: the index to the center atom, the number of atoms to rotate, the indexes of the atoms to rotate, and the indexes of the atoms to rotate bonded to the center atom.

Note that the rotated atoms bonded to the center atom are saved in center[].bond, so that they can be used to pick the rotation axis. It does not contain ALL atoms bonded to the center atom; use atom[center[].center].bond for that.

Obviously, since we end up modifying the atom coordinates, we cannot just precalculate a rotation axis, and store a fixed axis in the center[] data structure. We need to calculate the axis based on the nearby other atoms, and to do that, we need to save the indexes to those atoms.

The core loop in the program is an iterative loop. For each iteration, a random center atom, and a random rotation angle, is picked. The atoms specific to that center are rotated, and checked for collisions with the unrotated atoms. If the atoms do not come too close, the rotation is accepted,and counted as "one rotation". Otherwise it is discarded.

4 Future work

As stated in the Introduction section, the generated configurations should be sane and not just a result of a human's action. They should be configurations that we can find in the nature too, because only this configurations are real and will be observed in real life problems.

Our first approach (Rotation about an arbitary axis) didn't really show any caution on this aspect of our problem. The second approach (Rotations about bonds), took into account some information to try not to break this rule, but we can not be competely sure that we did achieve this in every generated configuration.

As a result, we focused on tools, which would help us simulate the molecule and produce new data. LAMMPS and Gromacs were two tools that we think are suitable for our task. We focused only in Gromacs.

There is a chance, that the "abeta_trajectory" file was an output of this kind of tools, like Gromacs. However, from the received package of the dataset, there were some significant files that were missing, assuming that the simulation was really performed by Gromacs.

Such files are conf.gro and topol.top. As a result we tried to set up our own simulation. Gromacs has many tutorials and also a mailing list. Installation was also easy. However, when the time for the simulation came, we were completely lost, because of the huge amount of the information in the tutorials. Maybe a Biologist would do better. The result was that we couldn't simulate the molecule.

5 Summary

To restate, our first approach was rotating a few hardcoded atoms, by a given angle, about a random axis. This however, didn't result in configurations that were different enough for the human eye to detect via Coot/PyMol. We produced 1080 new configurations.

Our second approach was to determine the bonds of the molecule, which would lead as to some atoms that could be used as rotation centers. The rotation center, pretty much cuts the molecule in two groups. The one group is going to be rotated.We performed a number of rotations. For every rotation, a random rotation center was chosen, as well as a random rotation angle, from some given interval.

This approach produced configurations that were considerably different from the initial ones. However he had to run much slower code. This approach also checks if the rotation is acceptable, by determining whether the atoms are located too close after the rotation. We produced 10000 new configurations in 42 minutes with 4 CPUs.

The idea of rotating some atoms of the molecule in order to reach a new configuration seems to give results and for sure can generate big datasets. The open question is whether these generated configurations are sane. Even our second approach that takes into account cases of the atoms to come too close, can not guarantee us that the results are approved by nature. For this reason, tools like Gromacs and LAMMPS should be studied more, maybe as a new project for next year's class.