GPathFinder Documentation

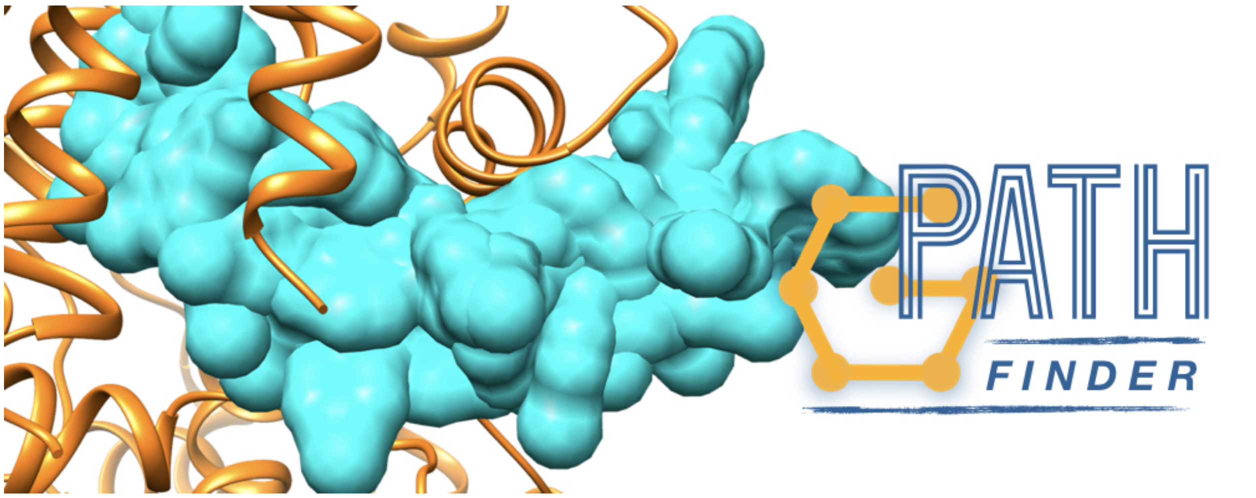
Release 1.3.0

José-Emilio Sánchez Aparicio, Giuseppe Sciortino, Daniel Villadri

Nov 10, 2020

User guide

GPathFinder is an extension built over GaudiMM core to allow the identification of ligand binding pathways at atomistic level.



CHAPTER 1

How to install

Recommended steps:

1 - Download the latest stable copy of UCSF Chimera and install it with the *.dmg* file (macOS) or with the following command (Linux):

```
chmod +x chimera-*.bin && sudo ./chimera-*.bin
```

Tip: When Chimera installer asks about *Install symbolic link to chimera executable?*, we recommend to choose option */usr/bin*, so GaudiMM will find Chimera installation without problem.

2 - Download Miniconda Python 2.7 Distribution for your platform and install it with:

```
bash Miniconda2*.sh
```

3 - Install `gpathfinder` with `conda` in a new environment called `gpathfinder` (or whatever name you prefer after the `-n` flag), using these custom channels (`-c` flags):

```
conda create -n gpathfinder -c omnia -c conda-forge -c bioconda -c josan_bcn␣
↪gpathfinder
```

4 - Activate the new environment as proposed:

```
conda activate gpathfinder
```

or

```
source activate gpathfinder
```

5 - Run it!

```
gpath
```

1.1 Check everything is OK

If everything went OK, you will get the usage screen:

```
Usage: gpath [OPTIONS] COMMAND [ARGS]...

  GPathFinder: Indentification of ligand pathways by a multi-objective
  genetic algorithm

  (C) 2019, InsiliChem
  https://github.com/insilichem/gpathfinder

Options:
  --version   Show the version and exit.
  -h, --help  Show this message and exit.

Commands:
  prepare  Create or edit a GPATH input file.
  run      Launch a GPATH input file.
  view     Analyze the results in a GPATH output file.
```

1.2 OS Compatibility

GPathFinder is compatible with Linux and macOS. This installation procedure has been checked with Chimera v.1.13.1 and the following distributions:

- macOS Mojave 10.14
- Mint 19.1
- Debian 9.9.0
- Ubuntu 16.04 and 18.04
- OpenSUSE Leap 15.1
- Manjaro 18.0.4

If you find some difficulties when installing it in a concrete distribution, please use the issues page to report them.

CHAPTER 2

Quick usage

If you created a conda environment to use GPathFinder (as proposed in installation), first you need to activate it with:

```
conda activate name_of_the_environment
```

or

```
source activate name_of_the_environment
```

Running GAUDI jobs is quite easy with `gaudi.cli.gaudi_run`. Put in your terminal:

```
gpath run /path/to/input_file.yaml
```

You will need at least three input files. A *.yaml* file with the configuration of the job and two *.mol2* files for the ligand and the receptor molecules. To learn how to create input files, go to *Input files*. You can also check the tutorials *Understanding the different sections of the input file* and *Preparing ligand and protein files*.

After the job is completed, you can use our home-made scripts to analyze them. A complete description of the output files is provided in *Output files*, and some tutorials on how to perform different analysis are available in *Analyzing GPathFinder results*.

To understand better the complete process of a GPathFinder calculation, you have also available the tutorial *Your first GPathFinder calculation*.

CHAPTER 3

Input files

GPathFinder needs the previous preparation of three mandatory input files:

- A *.yaml* file with the configuration of the calculation.
- A *.mol2* file containing the 3D structure of the ligand molecule.
- A *.mol2* file containing the 3D structure of the receptor molecule.

If you are considering to minimize the sample structures generated by Normal Mode Analysis for the global motions of the receptor, and your receptor has non-standard residues, you will also need a *.prmtop* file with the receptor topology and the parametrization of such non-standard residues.

3.1 *.yaml* configuration file

GPathFinder uses a YAML-formatted input file for setting up the calculation. YAML is a human-readable serialization format, already implemented in a broad range of languages. Input files must contain these five sections:

- **output**. Project options. Configure it to your liking
- **ga**. Genetic algorithm configuration. Normally, you don't have to touch this, except maybe the number of generations and population size.
- **similarity**. The similarity function to compare potentially redundant solutions.
- **genes**. List of descriptors used to define an individual
- **objectives**. The list of functions that will evaluate your individuals.

Normally, you can start from one of our standard input files, where Genetic Algorithm parameters have been set to appropiate values for a general case. You should choose the input file depending on the following:

- What kind of experiment you want to perform: discover (un)binding pathways or analyze a known pathway (initial and final points given in advance).
- If you want to minimize the receptor samples before starting the actual calculation or not.
- What method you will use to evaluate the solutions (clashes, vina, smina).

Link	Use	Minimization	Clashes	Vina	Smina
Input file 1	Discover unbinding pathways	No	Yes	No	No
Input file 2	Discover unbinding pathways	No	Yes	Yes	No
Input file 3	Discover unbinding pathways	Yes	Yes	Yes	No
Input file 4	Discover unbinding pathways	No	Yes	No	Yes
Input file 5	Discover unbinding pathways	Yes	Yes	No	Yes
Input file 6	Discover binding pathways	No	Yes	No	No
Input file 7	Discover binding pathways	No	Yes	Yes	No
Input file 8	Discover binding pathways	Yes	Yes	Yes	No
Input file 9	Discover binding pathways	No	Yes	No	Yes
Input file 10	Discover binding pathways	Yes	Yes	No	Yes
Input file 11	Analyze a known pathway	No	Yes	No	No
Input file 12	Analyze a known pathway	No	Yes	Yes	No
Input file 13	Analyze a known pathway	Yes	Yes	Yes	No
Input file 14	Analyze a known pathway	No	Yes	No	Yes
Input file 15	Analyze a known pathway	Yes	Yes	No	Yes

If you want to deepen the knowledge of the different parameters and fine-tune the input file, you can follow the tutorial *Understanding the different sections of the input file*

There is also a list of all the parameters and their default values in *List of parameters*

3.2 *.mol2* files for ligand and receptor

A typical workflow to prepare the two files for the ligand and receptor molecules starts from a *.pdb* structure of the Protein Data Bank. Of course, you can also use your own ligand and/or receptor files. For example, you can test a different ligand from the crystallographic one, or a conformation of the receptor obtained from a Molecular Dynamics simulation.

The requirements for the receptor file are:

- Small molecules that are not essential to consider in the calculation (like solvent molecules) should be removed.
- Alternative locations for residues (i.e. rotamers) should be removed. Only one conformation for each residue is allowed.
- In the case of clashes evaluation, it is not necessary to add Hydrogens if the user doesn't want to consider them.
- In the case of clashes+vina evaluation, adding Hydrogens is mandatory.
- In the case of minimizing the NMA samples, you have to take special care of terminal residues correctness and repair possible missing residues in your structure. Otherwise, OpenMM (in charge of the minimization) will complain and the calculation will be stopped.

The requirements for the ligand file are:

- In the case of unbinding pathway discovery, it is recommended that the 3D coordinates correspond to the binded position. Otherwise, the center of the binding site should be indicated explicitly in the *.yaml* configuration file (parameter `gaudi.genes.path.origin`).
- In the case of clashes evaluation, it is not necessary to add Hydrogens if the user doesn't want to consider them.
- In the case of clashes+vina evaluation, adding Hydrogens is mandatory.

A tutorial on how to generate *.mol2* files using UCSF Chimera starting from a crystallographic structure of the PDB is provided in *Preparing ligand and protein files*

CHAPTER 4

Output files

This section aims to provide the user with a complete description of all the files returned as output of a GPathFinder calculation. To learn how to perform different analysis over this data, you can check the tutorial *Analyzing GPathFinder results*.

The results of the calculation will be saved inside a folder on the path indicated in the `output.path` parameter of the *.yaml* input file. This folder will contain:

- **A '.gaudi-log' file**: contains the log information of the calculation, with data about the time employed in the execution, the evolution of the scores along the calculation and possible errors/warnings.
- **A '.yaml' file**: contains a replica of the *.yaml* input file employed in the calculation.
- **A '.gpath-output' file**: contains a summary of the obtained results, with data about their scores and name of the file that actually has the pathway. Can be used to get an overview of the quality of the solutions.
- **A 'summary.csv' file**: contains a summary of the scoring of all obtained results, detailed by frame. Also contains the coordinates of every solution (considering the center of mass of the ligand at every frame of the pathway). Can be used to see at a frame detail the quality of a concrete solution and differences between solutions.
- **A set of '.zip' files**: contain the different pathways obtained from the calculation. The number of solutions will be equal to the population size if `output.pareto` was set to *False* or equal to the size of the pareto frontier (i.e. dominant solutions) otherwise.

Optional files

- **A '.nmd' file**: contains the information about the prody modes calculated in Normal Mode Analysis of the receptor molecule. This file only will be present if `gaudi.genes.path_normalmodes.write_modes` was set to *True*.
- **A '.samples' file**: contains the information about all the samples generated during Normal Mode Analysis of the receptor molecule. This file only will be present if `gaudi.genes.path_normalmodes.write_samples` was set to *True*.

4.1 Contents of the *.zip* corresponding to each solution

Each *.zip* file contains all the necessary information to reproduce at atomic level the pathway proposed by GPathFinder as a solution for your problem. Inside the file there are the following contents:

- **Two '.mol2' files** with the original 3D structures of the ligand and the receptor.
- **A '.gaudi' file** with the summary of the files.
- **Another '.zip' file** which contains a set of *.pdb* files with all the frames of the pathway (each one has a conformation of the ligand and receptor molecules). You can open these *.pdb* files in any visualization tool like a MD-movie and work with them and analyze each frame. An example of a pathway that represents a GPathFinder solution can be found here.

Moreover, an *allele.txt* and a *scores.txt* files are present with all the necessary information to reconstruct the pathway from the original structures and the score information for every frame. These additional files can be used by your own scripts to analyze the results in further detail.

Finally, a *trajectory.pdb* file which contains the route that follows the ligand, as a set of points that represent the center of the ligand at each step of the trajectory. This can be used to easily compare the trajectory of different solutions obtained from GPathFinder calculations.

CHAPTER 5

Reproducibility

One of the key aspects of every research project is its reproducibility. Sometimes, specially when using softwares in their early lifecycle, provide enough information to reproduce calculations could be hard due to high amount of changes in the default values.

When reporting results obtained by a GPathFinder calculation, you should provide the following information:

- The full *.yaml* file, for example, in the supporting information.
- The version of the program you have used (for example, v.1.0.1). This will allow to deduce the values for the parameters that are not explicitly reported in the *.yaml* file.

The information of the default parameters for each version is provided in *List of parameters* section.

CHAPTER 6

List of parameters

6.1 Parameters for version 1.2.0

Genetic Algorithm parameters

- **population** (int): size of the starting population, in number of individuals. **Default: 12**
- **generations** (int): number of generations to simulate. **Default: 500**
- **mu** (float): the number of children to select at each generation, expressed as a multiplier of `ga.population`. **Default: 1**
- **lambda_** (float): the number of children to produce at each generation, expressed as a multiplier of `ga.population`. **Default: 1.0**
- **mut_eta** (float): crowding degree of the mutation. A high eta will produce a mutant resembling its parent, while a small eta will produce a solution much more different. **Default: 5**
- **mut_pb** (float): the probability that an offspring is produced by mutation. **Default: 0.8**
- **mut_indpb** (float): independent probability for each gene to be mutated. **Default: 1.0**
- **cx_eta** (float): crowding degree of the crossover. A high eta will produce children resembling to their parents, while a small eta will produce solutions much more different. **Default: 5**
- **cx_pb** (float): the probability that an offspring is produced by crossover. **Default: 0.2**

Path gene

- **radius_rotamers** (float): Maximum distance (in Angstroms) from any point of the ligand in every frame that is searched for possible rotamers of the protein side-chain. **Default: 3.0**
- **max_step_separation** (float): Maximum distance (in Angstroms) from one point of the ligand in the pathway to the next one. If not set by the user, GPathFinder calculates the value from the size of the ligand. **Default: None**
- **min_step_increment** (float): Minimum distance increment (in Angstroms) from the ligand's origin that has to be the ligand in one frame of the pathway with respect of the ligand's distance from the origin of the previous

frame of the pathway. If not set by the user, GPathFinder calculates the value as 2/5 by max_step_separation. **Default: None**

- **mut_pos_pb** (float): When a mutation occurs, this value is the probability of such mutation to be of the type *positions*, that is, the mutation changes the actual trajectory of the pathway. Warning: parameter for advanced users, usually the default value is correct for the vast majority of the systems. **Default: 0.10**

Path_torsion gene

- **flexibility** (int or float): Maximum number of degrees a bond can rotate. **Default: 360.0**
- **anchor** (str): Molecule/atom_serial_number of reference atom for torsions. If not set, the nearest atom to the molecule geometric center is selected. **Default: None**
- **rotatable_atom_types** (list of str): Which type of atom types (as in chimera.Atom.idatmType) should rotate. **Default: ('C3', 'N3', 'C2', 'N2', 'P')**
- **rotatable_atom_names** (list of str): Which type of atom names (as in chimera.Atom.name) should rotate. **Default: ()**
- **non_rotatable_bonds** (list of str): Which bonds (identified by [Molecule/SerialNumber1, Molecule/SerialNumber2]) are not allowed to rotate. **Default: ()**
- **non_rotatable_selection** (str): Which bonds (identified by a Chimera selection query) are not allowed to rotate. **Default: ()**

Path_rotamers gene

- **library** {'Dunbrack', 'Dynameomics'}: The rotamer library to use. **Default: Dunbrack**

Path_normalmodes gene

- **method** {'prody', 'gaussian', 'pca'}: *prody* to calculate normal modes using prody algorithms, *gaussian* to read normal modes from a gaussian output file and *pca* to perform a PCA analysis over a MD trajectory (.dcd file). **Default: prody**
- **modes** (list of int): Modes to be used to move the molecule. **Default: [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]**
- **pca_atoms** {'calpha', 'backbone', 'all'}: which atoms of the receptor are selected in the PCA analysis. **Default: calpha**
- **group_by** {'residues', 'mass', 'calpha', ''}: method to group atoms when using a coarse grain model of the receptor. **Default: residues**
- **group_lambda** (int): Either number of residues per group **(default=15)**, or total mass per group **(default=100)**
- **n_samples** (int): number of conformations to generate. **Default: 100**
- **rmsd** (float): average RMSD, in Angstroms, that conformations will have with respect to the initial conformation. **Default: 2.0**
- **minimize** (bool): whether to minimize the resulting samples or not. **Default: False**
- **forcefields** (list of str): Used when *minimize* is *True* to indicate which forcefields to use. It can be a .prmtop file containing the parametrization and topology of the Protein. **Default: ('amber99sbildn.xml',)**
- **minimization_tolerance** (float): used when *minimize* is *True*. Convergence criteria for energy minimization. In kJ/mol. **Default: 10.0**
- **minimization_iterations** (int): used when *minimize* is *True*. Max attempts to converge at minimization. **Default: 1000**

Path_scoring objective

- **radius** (float): maximum distance, in Angstroms, from any point of the ligand in every frame that is searched for possible interactions. **Default: 5.0**
- **method** {'sum', 'average', 'max'}: Method used to calculate the score (i.e. sum, average or maximum of the scores of all frames). **Default: average**
- **clash_threshold** (float): used when the scoring is *clashes*. Maximum overlap of van-der-Waals spheres. If the overlap is greater, it's considered a clash. **Default: 0.6**
- **bond_separation** (int): used when the scoring is *clashes*. Ignore clashes between atoms within n bonds. **Default: 4**
- **same_residue** (bool): used when the scoring is *clashes*. Include intra-molecular clashes. **Default: True**
- **smoothness_threshold** (float): used when method is *smoothness*. RMSD between ligands on two consecutive frames that is permitted considering a perfect score of smoothness. **Default: 0.0**
- **smina_scoring** (str): used when the scoring is *smina* to specify alternative builtin scoring function (e.g. vinardo). **Default: None**
- **smina_custom_scoring** (str): used when the scoring is *smina* to specify a custom scoring function file. **Default: None**
- **smina_custom_atoms** (str): used when the scoring is *smina* to specify a custom atom type parameters file. **Default: None**

6.2 Parameters for version 1.1.0

Genetic Algorithm parameters

- **population** (int): size of the starting population, in number of individuals. **Default: 12**
- **generations** (int): number of generations to simulate. **Default: 500**
- **mu** (float): the number of children to select at each generation, expressed as a multiplier of `ga.population`. **Default: 1**
- **lambda_** (float): the number of children to produce at each generation, expressed as a multiplier of `ga.population`. **Default: 1.0**
- **mut_eta** (float): crowding degree of the mutation. A high eta will produce a mutant resembling its parent, while a small eta will produce a solution much more different. **Default: 5**
- **mut_pb** (float): the probability that an offspring is produced by mutation. **Default: 0.8**
- **mut_indpb** (float): independent probability for each gene to be mutated. **Default: 1.0**
- **cx_eta** (float): crowding degree of the crossover. A high eta will produce children resembling to their parents, while a small eta will produce solutions much more different. **Default: 5**
- **cx_pb** (float): the probability that an offspring is produced by crossover. **Default: 0.2**

Path gene

- **radius_rotamers** (float): Maximum distance (in Angstroms) from any point of the ligand in every frame that is searched for possible rotamers of the protein side-chain. **Default: 3.0**
- **max_step_separation** (float): Maximum distance (in Angstroms) from one point of the ligand in the pathway to the next one. If not set by the user, GPathFinder calculates the value from the size of the ligand. **Default: None**

- **min_step_increment** (float): Minimum distance increment (in Angstroms) from the ligand's origin that has to be the ligand in one frame of the pathway with respect of the ligand's distance from the origin of the previous frame of the pathway. If not set by the user, GPathFinder calculates the value as 2/5 by max_step_separation. **Default: None**
- **mut_pos_pb** (float): When a mutation occurs, this value is the probability of such mutation to be of the type *positions*, that is, the mutation changes the actual trajectory of the pathway. Warning: parameter for advanced users, usually the default value is correct for the vast majority of the systems. **Default: 0.10**

Path_torsion gene

- **flexibility** (int or float): Maximum number of degrees a bond can rotate. **Default: 360.0**
- **anchor** (str): Molecule/atom_serial_number of reference atom for torsions. If not set, the nearest atom to the molecule geometric center is selected. **Default: None**
- **rotatable_atom_types** (list of str): Which type of atom types (as in chimera.Atom.idatmType) should rotate. **Default: ('C3', 'N3', 'C2', 'N2', 'P')**
- **rotatable_atom_names** (list of str): Which type of atom names (as in chimera.Atom.name) should rotate. **Default: ()**
- **non_rotatable_bonds** (list of str): Which bonds (identified by [Molecule/SerialNumber1, Molecule/SerialNumber2]) are not allowed to rotate. **Default: ()**
- **non_rotatable_selection** (str): Which bonds (identified by a Chimera selection query) are not allowed to rotate. **Default: ()**

Path_rotamers gene

- **library** {'Dunbrack', 'Dynameomics'}: The rotamer library to use. **Default: Dunbrack**

Path_normalmodes gene

- **method** {'prody', 'gaussian'}: *prody* to calculate normal modes using prody algorithms and *gaussian* to read normal modes from a gaussian output file. **Default: prody**
- **modes** (list of int): Modes to be used to move the molecule. **Default: [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]**
- **group_by** {'residues', 'mass', 'calpha', ''}: method to group atoms when using a coarse grain model of the receptor. **Default: residues**
- **group_lambda** (int): Either number of residues per group **(default=15)**, or total mass per group **(default=100)**
- **n_samples** (int): number of conformations to generate. **Default: 100**
- **rmsd** (float): average RMSD, in Angstroms, that conformations will have with respect to the initial conformation. **Default: 2.0**
- **minimize** (bool): whether to minimize the resulting samples or not. **Default: False**
- **forcefields** (list of str): Used when *minimize* is *True* to indicate which forcefields to use. It can be a .prmtop file containing the parametrization and topology of the Protein. **Default: ('amber99sbildn.xml',)**
- **minimization_tolerance** (float): used when *minimize* is *True*. Convergence criteria for energy minimization. In kJ/mol. **Default: 10.0**
- **minimization_iterations** (int): used when *minimize* is *True*. Max attempts to converge at minimization. **Default: 1000**

Path_scoring objective

- **radius** (float): maximum distance, in Angstroms, from any point of the ligand in every frame that is searched for possible interactions. **Default: 5.0**

- **method** {'sum', 'average', 'max'}: Method used to calculate the score (i.e. sum, average or maximum of the scores of all frames). **Default: average**
- **clash_threshold** (float): used when the scoring is *clashes*. Maximum overlap of van-der-Waals spheres. If the overlap is greater, it's considered a clash. **Default: 0.6**
- **bond_separation** (int): used when the scoring is *clashes*. Ignore clashes between atoms within n bonds. **Default: 4**
- **same_residue** (bool): used when the scoring is *clashes*. Include intra-molecular clashes. **Default: True**
- **smoothness_threshold** (float): used when method is *smoothness*. RMSD between ligands on two consecutive frames that is permitted considering a perfect score of smoothness. **Default: 0.0**
- **smina_scoring** (str): used when the scoring is *smina* to specify alternative builtin scoring function (e.g. vinardo). **Default: None**
- **smina_custom_scoring** (str): used when the scoring is *smina* to specify a custom scoring function file. **Default: None**
- **smina_custom_atoms** (str): used when the scoring is *smina* to specify a custom atom type parameters file. **Default: None**

6.3 Parameters for versions 1.0.x

Genetic Algorithm parameters

- **population** (int): size of the starting population, in number of individuals. **Default: 12**
- **generations** (int): number of generations to simulate. **Default: 500**
- **mu** (float): the number of children to select at each generation, expressed as a multiplier of `ga.population`. **Default: 1**
- **lambda_** (float): the number of children to produce at each generation, expressed as a multiplier of `ga.population`. **Default: 1.0**
- **mut_eta** (float): crowding degree of the mutation. A high eta will produce a mutant resembling its parent, while a small eta will produce a solution much more different. **Default: 5**
- **mut_pb** (float): the probability that an offspring is produced by mutation. **Default: 0.8**
- **mut_indpb** (float): independent probability for each gene to be mutated. **Default: 1.0**
- **cx_eta** (float): crowding degree of the crossover. A high eta will produce children resembling to their parents, while a small eta will produce solutions much more different. **Default: 5**
- **cx_pb** (float): the probability that an offspring is produced by crossover. **Default: 0.2**

Path gene

- **radius_rotamers** (float): Maximum distance (in Angstroms) from any point of the ligand in every frame that is searched for possible rotamers of the protein side-chain. **Default: 3.0**
- **max_step_separation** (float): Maximum distance (in Angstroms) from one point of the ligand in the pathway to the next one. If not set by the user, GPathFinder calculates the value from the size of the ligand. **Default: None**
- **min_step_increment** (float): Minimum distance increment (in Angstroms) from the ligand's origin that has to be the ligand in one frame of the pathway with respect of the ligand's distance from the origin of the previous frame of the pathway. If not set by the user, GPathFinder calculates the value as 2/5 by max_step_separation. **Default: None**

- **mut_pos_pb** (float): When a mutation occurs, this value is the probability of such mutation to be of the type *positions*, that is, the mutation changes the actual trajectory of the pathway. Warning: parameter for advanced users, usually the default value is correct for the vast majority of the systems. **Default: 0.10**

Path_torsion gene

- **flexibility** (int or float): Maximum number of degrees a bond can rotate. **Default: 360.0**
- **anchor** (str): Molecule/atom_serial_number of reference atom for torsions. If not set, the nearest atom to the molecule geometric center is selected. **Default: None**
- **rotatable_atom_types** (list of str): Which type of atom types (as in chimera.Atom.idatmType) should rotate. **Default: ('C3', 'N3', 'C2', 'N2', 'P')**
- **rotatable_atom_names** (list of str): Which type of atom names (as in chimera.Atom.name) should rotate. **Default: ()**
- **non_rotatable_bonds** (list of str): Which bonds (identified by [Molecule/SerialNumber1, Molecule/SerialNumber2]) are not allowed to rotate. **Default: ()**
- **non_rotatable_selection** (str): Which bonds (identified by a Chimera selection query) are not allowed to rotate. **Default: ()**

Path_rotamers gene

- **library** {'Dunbrack', 'Dynameomics'}: The rotamer library to use. **Default: Dunbrack**

Path_normalmodes gene

- **method** {'prody', 'gaussian'}: *prody* to calculate normal modes using prody algorithms and *gaussian* to read normal modes from a gaussian output file. **Default: prody**
- **modes** (list of int): Modes to be used to move the molecule. **Default: [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]**
- **group_by** {'residues', 'mass', 'calpha', ''}: method to group atoms when using a coarse grain model of the receptor. **Default: residues**
- **group_lambda** (int): Either number of residues per group **(default=15)**, or total mass per group **(default=100)**
- **n_samples** (int): number of conformations to generate. **Default: 100**
- **rmsd** (float): average RMSD, in Angstroms, that conformations will have with respect to the initial conformation. **Default: 2.0**
- **minimize** (bool): whether to minimize the resulting samples or not. **Default: False**
- **forcefields** (list of str): Used when *minimize* is *True* to indicate which forcefields to use. It can be a .prmtop file containing the parametrization and topology of the Protein. **Default: ('amber99sbildn.xml',)**
- **minimization_tolerance** (float): used when *minimize* is *True*. Convergence criteria for energy minimization. In kJ/mol. **Default: 10.0**
- **minimization_iterations** (int): used when *minimize* is *True*. Max attempts to converge at minimization. **Default: 1000**

Path_scoring objective

- **radius** (float): maximum distance, in Angstroms, from any point of the ligand in every frame that is searched for possible interactions. **Default: 5.0**
- **method** {'sum', 'average', 'max'}: Method used to calculate the score (i.e. sum, average or maximum of the scores of all frames). **Default: average**
- **clash_threshold** (float): used when the scoring is *clashes*. Maximum overlap of van-der-Waals spheres. If the overlap is greater, it's considered a clash. **Default: 0.6**

- **bond_separation** (int): used when the scoring is *clashes*. Ignore clashes between atoms within n bonds. **Default: 4**
- **same_residue** (bool): used when the scoring is *clashes*. Include intra-molecular clashes. **Default: True**
- **smoothness_threshold** (float): used when method is *smoothness*. RMSD between ligands on two consecutive frames that is permitted considering a perfect score of smoothness. **Default: 0.0**

CHAPTER 7

Refinement

The refinement funtion allows to generate a more continious path of configurations in between each frame generated by GPathFinder using an RRT-algorithm. It can be used as follows:

- Place the directory containing the gaudi GPathFinder result files into the refinement_input_files directory of the GPathFinder module
- In the terminal, run: python refinement.py <directory> <path_number (int)> <start_frame> <end_frame>

The refinement_input_files directory already contains example files suited to be run, for example:

python refinement.py 1ldi_a_clashes_01 0 10 20

runs the refinement process of the path in file 1ldi_a_clashes_02_000 through frames 10 to 20 and outputs a log file and the path .pdb file in the refinemnt_output_files directory.

CHAPTER 8

How to cite this software

GPathFinder is scientific software, funded by public research grants (Spanish MINECO's project `CTQ2017-87889-P`, and Generalitat de Catalunya's project `2017SGR1323`)

GaudiMM, root development over GPathFinder is built, is also scientific software, funded by public research grants (Spanish MINECO's project `CTQ2014-54071-P`, Generalitat de Catalunya's project `2014SGR989` and research grant `2015FI_B00768`, COST Action `CM1306`).

If you make use of GPathFinder in scientific publications, please cite our article in IJMS. GaudiMM has also its own article in JCC. It will help measure the impact of our research and future funding!

```
@article {IJMS:ijms20133155,
    author = {Sánchez-Aparicio, José-Emilio and Sciortino, Giuseppe and Viladrich␣
↪Herrmannsdoerfer, Daniel and Orenes Chueca, Pablo and Rodríguez-Guerra Pedregal,␣
↪Jaime and Maréchal, Jean-Didier},
    title = {GPathFinder: identification of ligand binding pathways by a multi-
↪objective genetic algorithm},
    journal = {International Journal of Molecular Sciences},
    volume = {20},
    number = {13},
    issn = {1422-0067},
    url = {https://www.mdpi.com/1422-0067/20/13/3155},
    doi = {https://doi.org/10.3390/ijms20133155 },
    pages = {3155},
    keywords = {multi-objective genetic algorithm, molecular modeling, ligand␣
↪diffusion, computational chemistry, molecular docking, drug design},
    year = {2019},
}
```

```
@article {JCC:JCC24847,
    author = {Rodríguez-Guerra Pedregal, Jaime and Sciortino, Giuseppe and Guasp,␣
↪Jordi and Municoy, Martí and Maréchal, Jean-Didier},
    title = {GaudiMM: A modular multi-objective platform for molecular modeling},
    journal = {Journal of Computational Chemistry},
    volume = {38},
```

(continues on next page)

(continued from previous page)

```
    number = {24},
    issn = {1096-987X},
    url = {http://dx.doi.org/10.1002/jcc.24847},
    doi = {10.1002/jcc.24847},
    pages = {2118--2126},
    keywords = {molecular modeling, protein-ligand docking, multi-objective␣
↪optimization, genetic algorithms, metallopeptides},
    year = {2017},
}
```

CHAPTER 9

Your first GPathFinder calculation

9.1 Objectives

This tutorial aims at providing an overview of a typical workflow when carrying out a GPathFinder calculation. It is divided in four sections. As all the input/output files for each section are available here, you can follow the entire tutorial sequentially or you can choose one of the sections individually.

We propose the example of Acetylcholine bound to Acetylcholinesterase to study the possible unbinding routes of the ligand. The structure that is going to be used corresponds to the PDB code **2ace**.

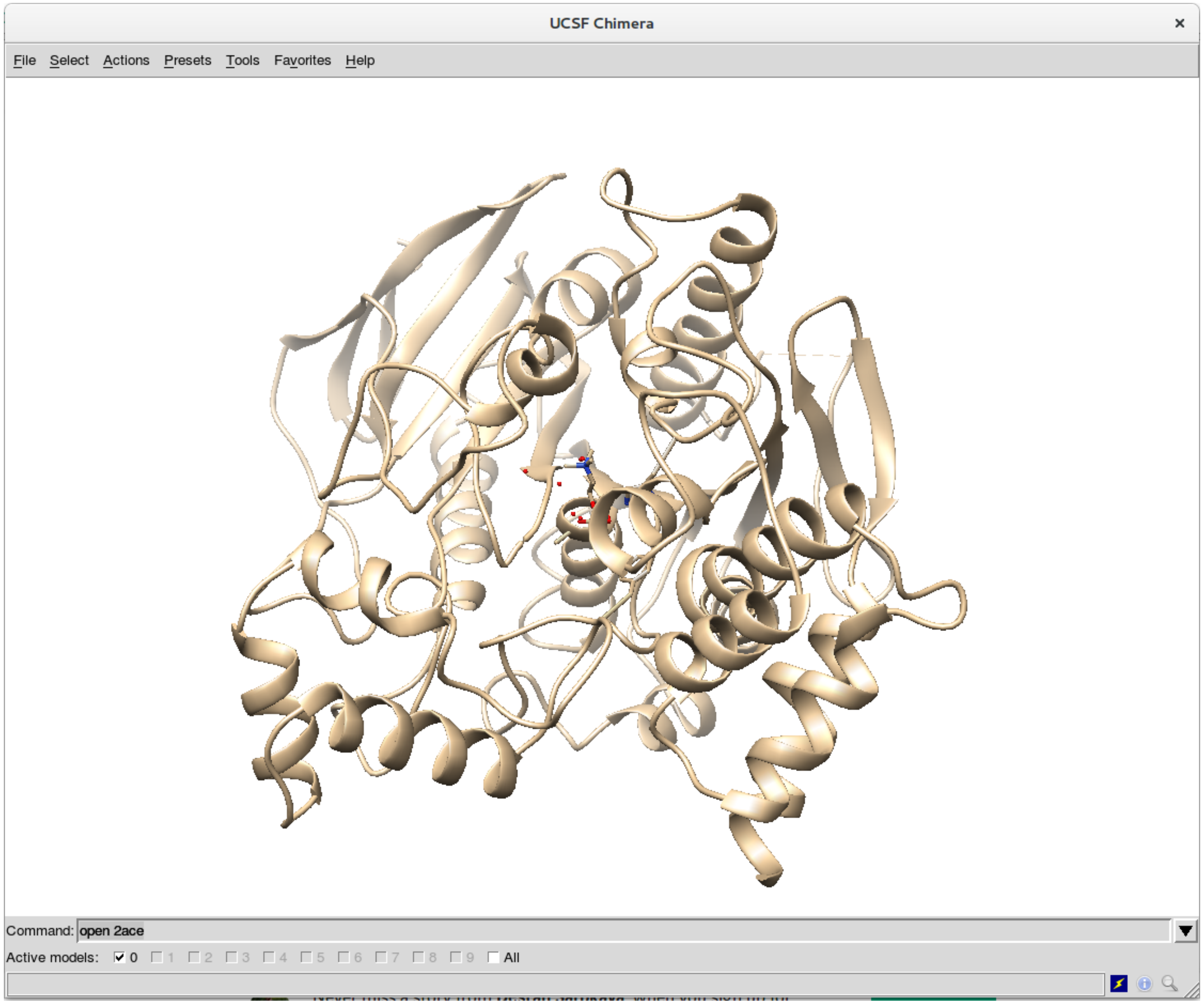
9.2 1. Preparing ligand and receptor *.mol2* files

- **Necessary files for this section**: None
- **Output files from this section**: ligand.mol2, ligand_with_H.mol2, protein.mol2, protein_with_H.mol2

The first step is to obtain the complete structure of the system. It can be done by several manners, but we propose the following two options:

- Download the *.pdb* file from the Protein Data Bank. Option `Download Files -> PDB Format`. Open the downloaded file (*2ace.pdb*) with UCSF Chimera (you can use other visualization tool, of course).
- Directly use UCSF Chimera and open the structure in the viewer with the command `open 2ace`. You can open the command terminal of Chimera with `Favorites -> Command Line`.

Regardless of the option you choose, you will end up with the structure of *2ace* in the viewer:


UCSF Chimera
File Select Actions Presets Tools Favorites Help
Command: open 2ace
Active models: 0 1 2 3 4 5 6 7 8 9 All

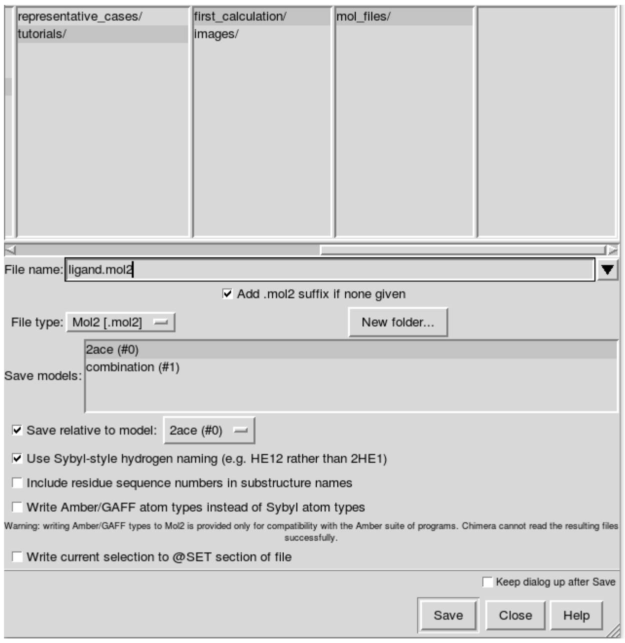
The next step is to clean up the system:

- Remove non essential small molecules. In this case, we remove all water molecules by the following Chimera command: `del :HOH`
- Remove alternative locations for residues (i.e. rotamers), if existing. In this case there are not such alternative locations, but it would be removed by the Chimera commands: `del @/altLoc=B`, `del @/altLoc=C`, . . .

Then, you have to split the system into two different models. In Chimera, it can be done by the following steps:

- In the model panel (`Favorites -> Model Panel`), `copy/combine` the whole system to have two identical replicas.
- Remove the receptor from the first model with command `del #0 & protein`
- Remove the ligand from the second model with command `del #1 & ligand`

Save the *ligand.mol2* and *protein.mol2* files with `File -> Save Mol2`. You have to select model (#0) when saving the **ligand** and model (#1) for the **protein**. Make sure that *Save relative to model: (#0)* is marked to preserve coherent relative coordinates in both files. Save the files in a folder called *mol_files* inside a *first_calculation* folder:


representative_cases/ tutorials/ | first_calculation/ images/ | mol_files/

File name: ligand.mol2

☑ Add .mol2 suffix if none given

File type: Mol2 [.mol2]

Save models: 2ace (#0) combination (#1)

☑ Save relative to model: 2ace (#0)

☑ Use Sybyl-style hydrogen naming (e.g. HE12 rather than 2HE1)

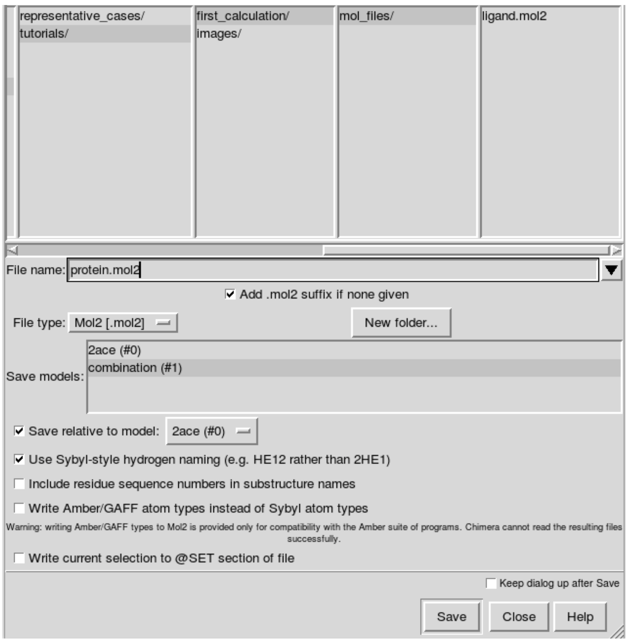
☐ Include residue sequence numbers in substructure names

☐ Write Amber/GAFF atom types instead of Sybyl atom types

Warning: writing Amber/GAFF types to Mol2 is provided only for compatibility with the Amber suite of programs. Chimera cannot read the resulting files successfully.

☐ Write current selection to @SET section of file

☐ Keep dialog up after Save


representative_cases/ tutorials/ | first_calculation/ images/ | mol_files/ | ligand.mol2

File name: protein.mol2

☑ Add .mol2 suffix if none given

File type: Mol2 [.mol2]

Save models: 2ace (#0) combination (#1)

☑ Save relative to model: 2ace (#0)

☑ Use Sybyl-style hydrogen naming (e.g. HE12 rather than 2HE1)

☐ Include residue sequence numbers in substructure names

☐ Write Amber/GAFF atom types instead of Sybyl atom types

Warning: writing Amber/GAFF types to Mol2 is provided only for compatibility with the Amber suite of programs. Chimera cannot read the resulting files successfully.

☐ Write current selection to @SET section of file

☐ Keep dialog up after Save

If you want to optimize the solutions using only clashes, that would be enough. But if you want to add a Vina scoring criterium, files with explicit hydrogens are needed. To do so, you can add hydrogens in Chimera with `Tools -> Structure Editing -> AddH`. Once you have both models (ligand and protein) with hydrogens, you can save them as before using different names, for example *ligand_with_H.mol2* and *protein_with_H.mol2*.

9.3 2. Preparing *input.yaml* file

- **Necessary files for this section**: ligand.mol2/ligand_with_H.mol2, protein.mol2/protein_with_H.mol2
- **Output files from this section**: input_clashes.yaml/input_clashes_vina.yaml

To prepare the configuration *.yaml* file, you have several templates available at *Input files*. Here, we are going to use either discover unbinding pathways with clashes evaluation or discover unbinding pathways with clashes+vina evaluation as base for our input configuration file.

Warning: Remember: if you are going to evaluate clashes+vina, the *.mol2* files prepared in *step 1* have to include explicit hydrogen atoms. For only clashes, without hydrogens would be enough.

You have to make the following changes/adjustements in your template file:

- `_path: input.yaml` (or the name you choose for your configuration file)

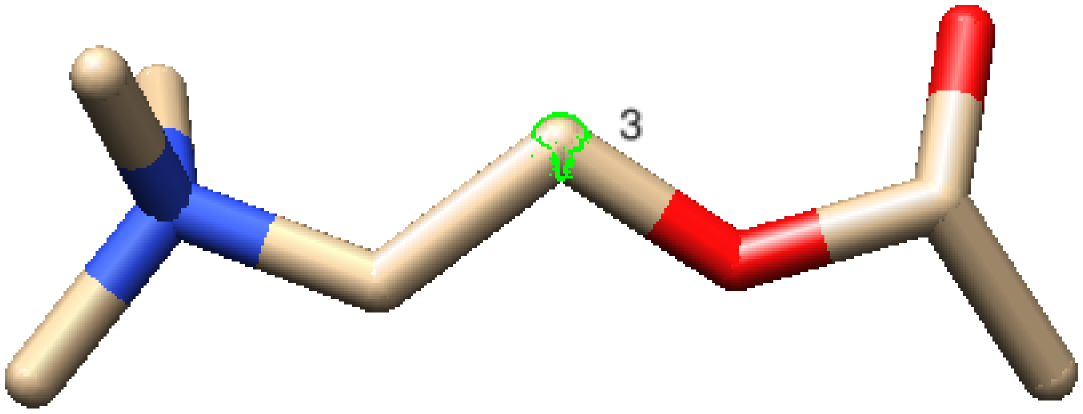
ga section

- `generations: XXX` (the values provided in the templates ensure good results for a general case, but you can adjust it to reduce the computation time)

genes section

- `path: ./mol_files/ligand.mol2` (path of your ligand *.mol2* file)
- `path: ./mol_files/protein.mol2` (path of your protein *.mol2* file)
- `anchor: Ligand/3`

Tip: To ensure that torsions of the dihedral angles in the ligand molecule are modified in a proper manner, we recommend to choose an **anchor atom** near the geometric center of the ligand. You have to indicate the name of the ligand gene and the Chimera serial number of the atom, in this case `Ligand/3`. To know the serial number, you can open in text mode the *.mol2* file, or, visualize the property inside Chimera: select the atom and, then, show its serial number with `Actions -> Label -> Other -> Label with attribute: serialNumber`.



output section

- `name: 2ace` (choose a name for your calculation)
- `path: ./results_2ace` (folder where the results of the calculation will be saved)

Finally, you have to save your *.yaml* inside your calculation folder, for example *first_calculation*.

9.4 3. Running the calculation

- **Necessary files for this section**: ligand.mol2/ligand_with_H.mol2, receptor.mol2/receptor_with_H.mol2, input_clashes.yaml/input_clashes_vina.yaml
- **Output files from this section**: folder with GPathFinder results

Running your calculation is as easy as open a **terminal**, activate your **conda environment** with:

```
conda activate name_of_the_environment
```

or

```
source activate name_of_the_environment
```

Go to your calculation folder, where *input_clashes.yaml* or *input_clashes_vina.yaml* is located, and run it with:

```
gpath run input_clashes.yaml
```

or

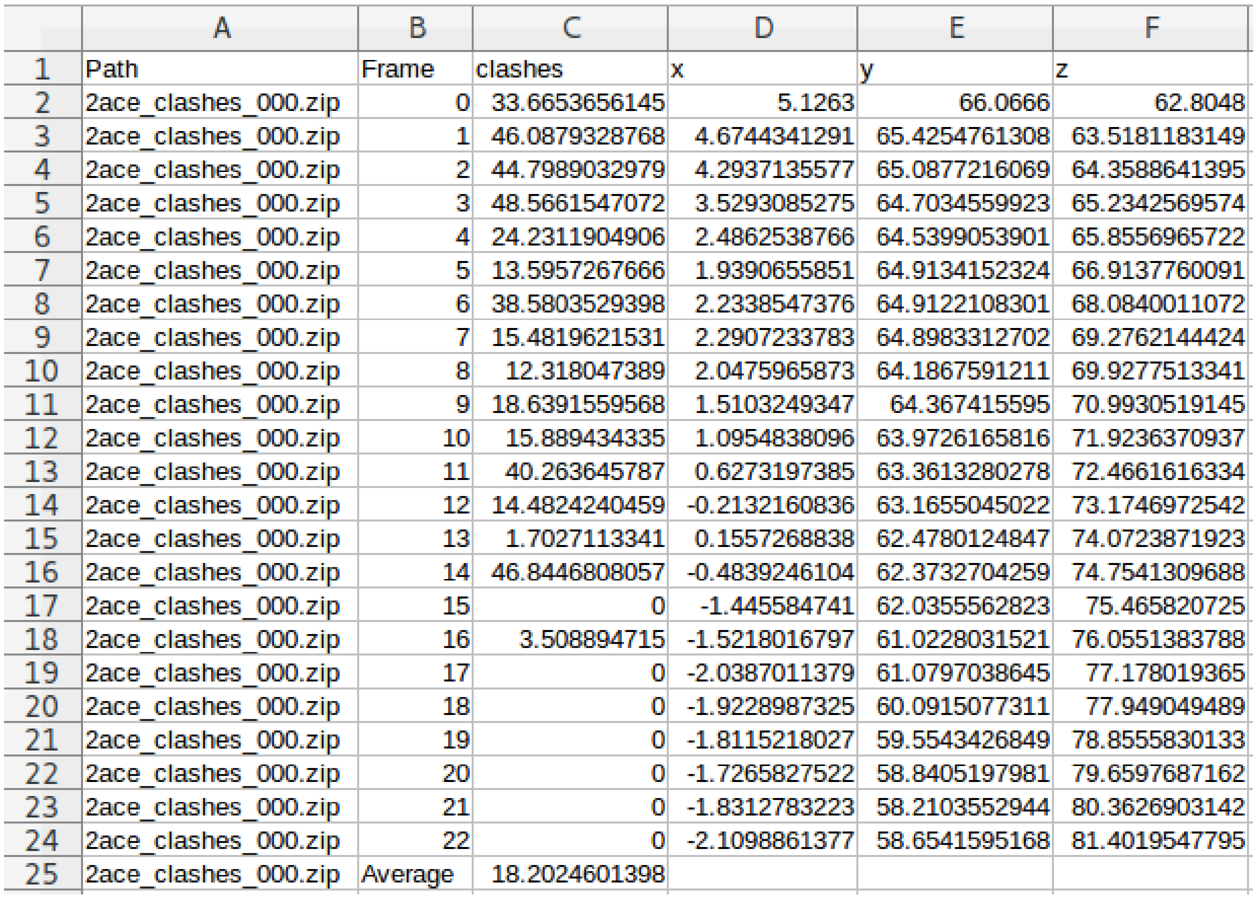
```
gpath run input_clashes_vina.yaml
```

9.5 4. Visualizing results

- **Necessary files for this section**: folder with GPathFinder results
- **Output files from this section**: None

Inside the *results* folder you can find, among other files, a *summary.csv* file with an overview of the solutions that GPathFinder has obtained from your calculation. As any standard *.csv* file, you can open it with *LibreOffice Calc*, *Microsoft Excel* or any other editor you want.

For each solution, you will find the scoring (clashes, vina, etc.) and the coordinates of the ligand at every frame forming the (un)binding pathway. Moreover, the last line of every solution will be the average score for all the frames of the pathway. For example, the solution called "2ace_clashes_000" has 23 frames with a clashes average of 18.2 Å^3:



	A	B	C	D	E	F
1	Path	Frame	clashes	x	y	z
2	2ace_clashes_000.zip	0	33.6653656145	5.1263	66.0666	62.8048
3	2ace_clashes_000.zip	1	46.0879328768	4.6744341291	65.4254761308	63.5181183149
4	2ace_clashes_000.zip	2	44.7989032979	4.2937135577	65.0877216069	64.3588641395
5	2ace_clashes_000.zip	3	48.5661547072	3.5293085275	64.7034559923	65.2342569574
6	2ace_clashes_000.zip	4	24.2311904906	2.4862538766	64.5399053901	65.8556965722
7	2ace_clashes_000.zip	5	13.5957267666	1.9390655851	64.9134152324	66.9137760091
8	2ace_clashes_000.zip	6	38.5803529398	2.2338547376	64.9122108301	68.0840011072
9	2ace_clashes_000.zip	7	15.4819621531	2.2907233783	64.8983312702	69.2762144424
10	2ace_clashes_000.zip	8	12.318047389	2.0475965873	64.1867591211	69.9277513341
11	2ace_clashes_000.zip	9	18.6391559568	1.5103249347	64.367415595	70.9930519145
12	2ace_clashes_000.zip	10	15.889434335	1.0954838096	63.9726165816	71.9236370937
13	2ace_clashes_000.zip	11	40.263645787	0.6273197385	63.3613280278	72.4661616334
14	2ace_clashes_000.zip	12	14.4824240459	-0.2132160836	63.1655045022	73.1746972542
15	2ace_clashes_000.zip	13	1.7027113341	0.1557268838	62.4780124847	74.0723871923
16	2ace_clashes_000.zip	14	46.8446808057	-0.4839246104	62.3732704259	74.7541309688
17	2ace_clashes_000.zip	15	0	-1.445584741	62.0355562823	75.465820725
18	2ace_clashes_000.zip	16	3.508894715	-1.5218016797	61.0228031521	76.0551383788
19	2ace_clashes_000.zip	17	0	-2.0387011379	61.0797038645	77.178019365
20	2ace_clashes_000.zip	18	0	-1.9228987325	60.0915077311	77.949049489
21	2ace_clashes_000.zip	19	0	-1.8115218027	59.5543426849	78.8555830133
22	2ace_clashes_000.zip	20	0	-1.7265827522	58.8405197981	79.6597687162
23	2ace_clashes_000.zip	21	0	-1.8312783223	58.2103552944	80.3626903142
24	2ace_clashes_000.zip	22	0	-2.1098861377	58.6541595168	81.4019547795
25	2ace_clashes_000.zip	Average	18.2024601398			

The corresponding *.zip* file (in this case, *2ace_clashes_000.zip*) contains the actual information about the solution. For a complete description of all the files, you can refer to the section *Output files*. Here we are centering on visualize the structures that form the (un)binding pathway. To do so, you have two possibilities:

- Open the `2ace_clashes_000.zip -> Pathway_000_Pathway.zip -> frame_XXX.pdb` files directly with your favourite visualization tool (e.g. UCSF Chimera). They contain the structure of the receptor + ligand complex at every position (frame) of the (un)binding process, so you can choose the frames that you want and examine them individually.
- Use the `Tools -> MD/Ensemble Analysis -> MD Movie` in UCSF Chimera (select the *PDB frames contained in multiple files* option). It will open the frames that you select as a movie, so you can analyze the (un)binding process:

Whatever option you choose, you will end with the desired structure/s in your visualization tool, so you can use it for further analysis. If you want to deepen the analysis of GPathFinder solutions, you can follow the tutorial *Analyzing GPathFinder results*.

CHAPTER 10

Understanding the different sections of the input file

Warning: This tutorial is work in progress.

CHAPTER 11

Preparing ligand and protein files

Warning: This tutorial is work in progress.

CHAPTER 12

Analyzing GPathFinder results

Warning: This tutorial is work in progress.

CHAPTER 13

How to use your custom pool of conformations in a GPathFinder calculation

13.1 Objectives

Since version 1.3.0 GPathFinder allows to provide a custom set of conformations for the ligand or the protein or both. It allows restraining the conformational space that the calculation will explore.

This feature would be of particular utility when the conformational space of a molecule is known beforehand. For example, suppose you have a sampling for your protein obtained from a Molecular Dynamics. In that case, it could be interesting to see if the ligand can access the binding site using only that set of conformations of the protein. The same applies for the ligand, you can have a bunch of conformers that are known to be the most stable, and therefore it could be interesting to test if it can access the binding site adopting only those conformations.

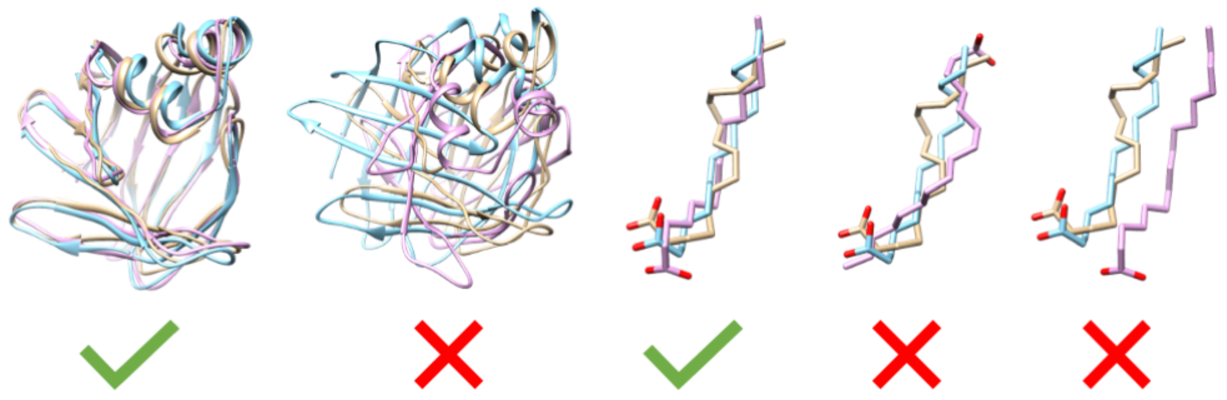
This tutorial aims at providing the information on how to configure a GPathFinder calculation to take profit of this feature. It is mainly divided into two sections: first, how to correctly generate the set of files for the conformations; and second, how to parametrize the *.yaml* file.

13.2 1. Preparing files for the pool of conformations

You need a *.mol2* file or a *.pdb* file for each conformation of your molecule/s, regardless of whether they are ligand or receptor. You can use either the *.mol2* or *.pdb* formats, but mixing different types of files for a molecule could produce strange results.

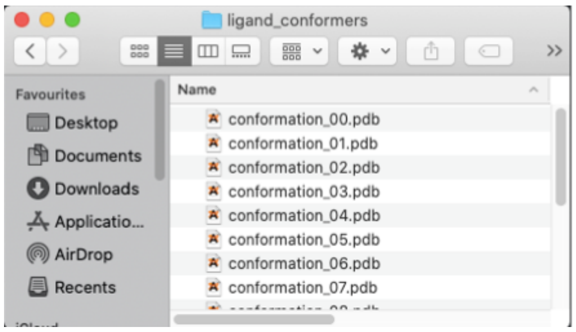
Warning: It is **essential** that all the files for a molecule share the same atomic structure and atom order. They are a pool of conformations, not a pool of different chemical structures. In particular, it is useful to check the files when they provide from various sources, for example, snapshots from an MD and a crystallographic structure from the PDB. For the protein files, it is also important that all files share the same residue numbering and order.

The conformations should be appropriately aligned. In the case of the protein, you can choose as the alignment reference either the whole structure or only the region of interest. Between ligand conformations, the centre of mass and orientation should be as similar as possible.

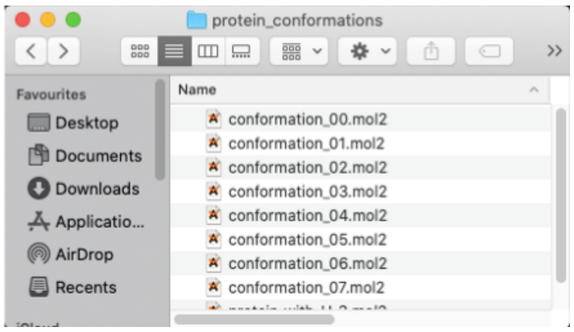


Tip: In *UCSF Chimera*, you can achieve a proper alignment of proteic molecules using the MatchMaker command. To align small molecules (i.e. ligand) you can use the match command.

Once you have generated the files of the conformations, you should group them in a folder containing all the files of the molecule. For example, suppose you are generating conformations for both the ligand and the receptor. In that case, you must end up with two folders, one containing **only** the files for the ligand and the other **only** the files for the receptor:



```
conformation_00.pdb
conformation_01.pdb
conformation_02.pdb
conformation_03.pdb
conformation_04.pdb
conformation_05.pdb
conformation_06.pdb
conformation_07.pdb
```

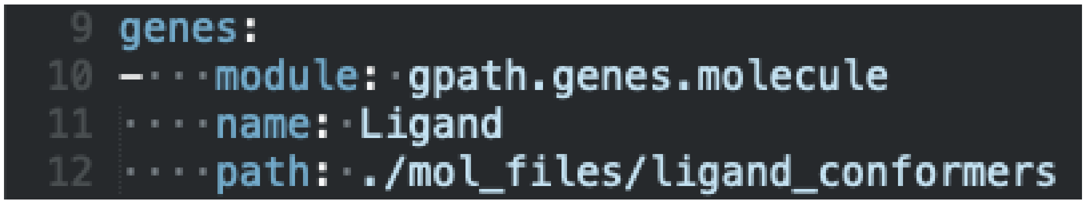


```
conformation_00.mol2
conformation_01.mol2
conformation_02.mol2
conformation_03.mol2
conformation_04.mol2
conformation_05.mol2
conformation_06.mol2
conformation_07.mol2
```

13.3 2. Preparing the GPathFinder .yaml input file

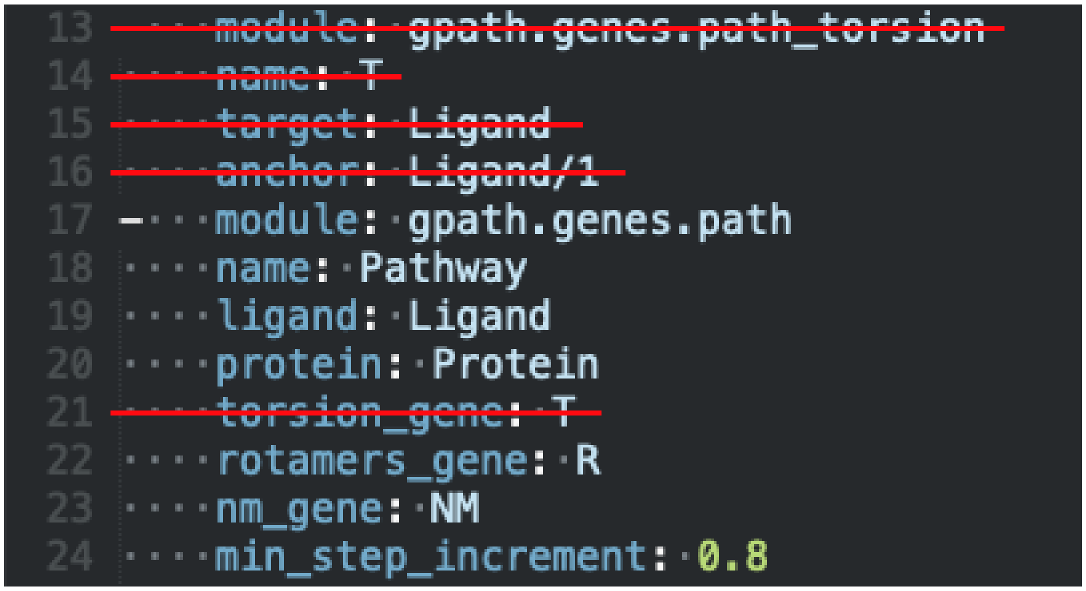
- If you are using several files for the ligand molecule:

1. You should indicate in the Ligand *molecule* gene the route to the folder containing the ligand conformations:



```
genes:
-   module: gpath.genes.molecule
    name: Ligand
    path: ./mol_files/ligand_conformers
```

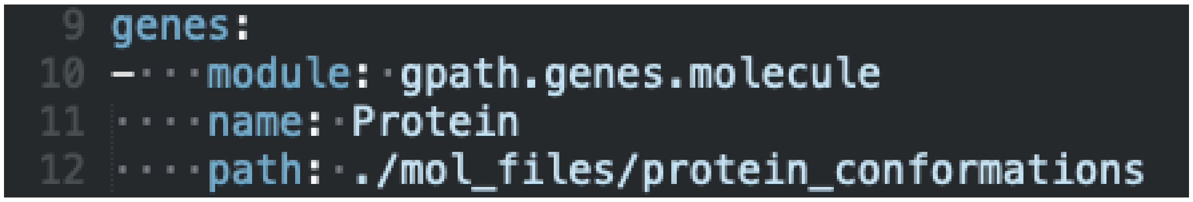
2. It is not allowed to use a *path_torsion* gene together with a custom pool of conformations for the ligand. Then, you should delete both the *path_torsion* gene and its reference in the *path* gene:



```
~~    module: gpath.genes.path_torsion~~
~~    name: T~~
~~    target: Ligand~~
~~    anchor: Ligand/1~~
-   module: gpath.genes.path
    name: Pathway
    ligand: Ligand
    protein: Protein
~~    torsion_gene: T~~
    rotamers_gene: R
    nm_gene: NM
    min_step_increment: 0.8
```

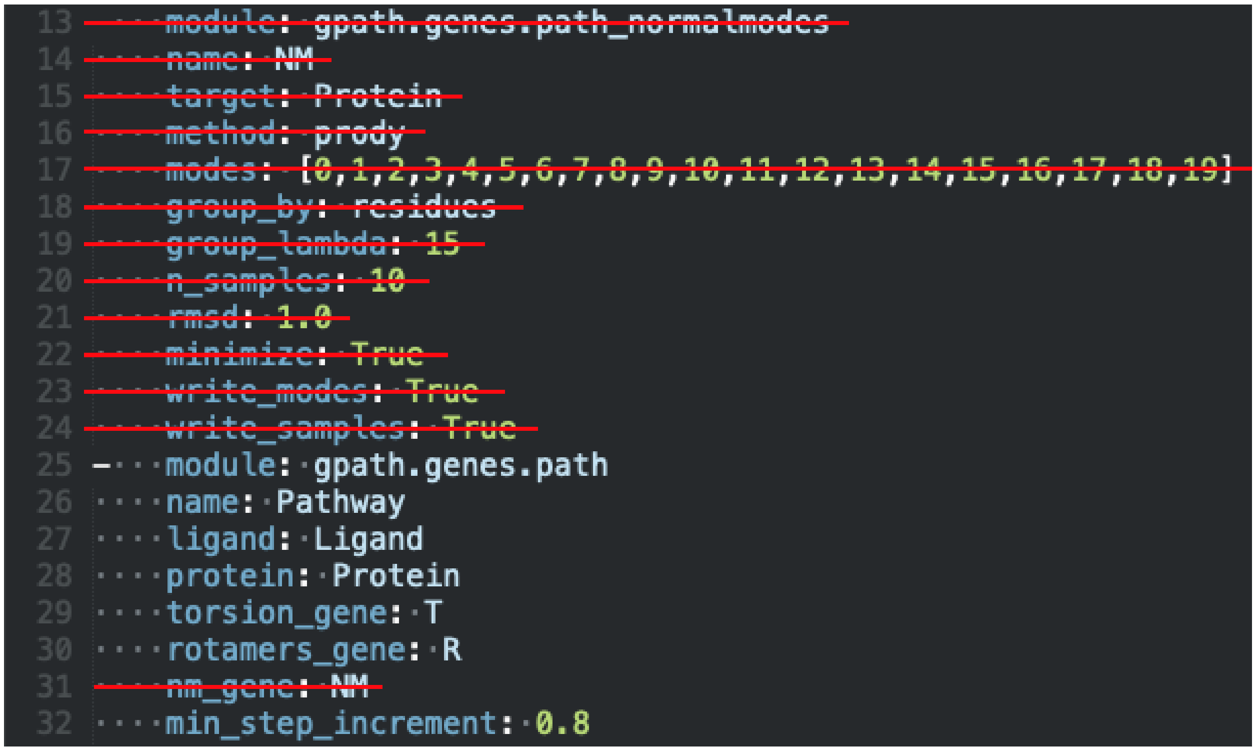
- If you are using several files for the protein molecule:

1. You should indicate in the Protein *molecule* gene the route to the folder containing the receptor conformations:



```
genes:
-   module: gpath.genes.molecule
    name: Protein
    path: ./mol_files/protein_conformations
```

2. It is not allowed to use a *path_normalmodes* gene together with a custom pool of conformations for the protein. Then, you should delete both the *path_normalmodes* gene and its reference in the *path* gene:

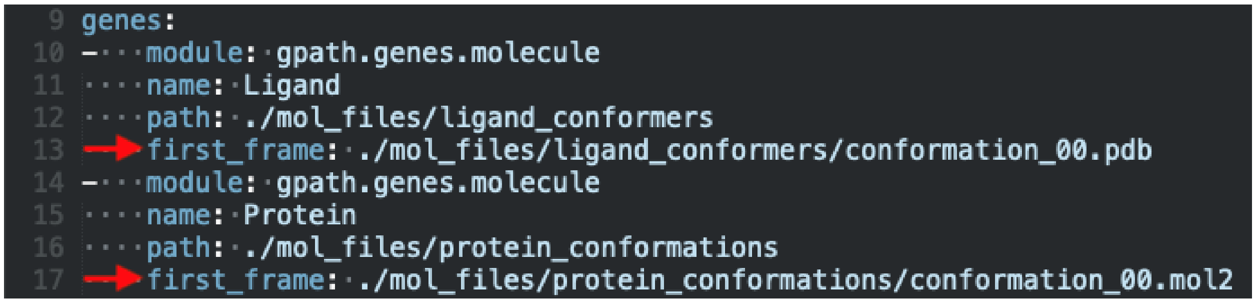


```
~~    module: gpath.genes.path_normalmodes~~
~~    name: NM~~
~~    target: Protein~~
~~    method: prody~~
~~    modes: [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]~~
~~    group_by: residues~~
~~    group_lambda: 15~~
~~    n_samples: 10~~
~~    rmsd: 1.0~~
~~    minimize: True~~
~~    write_modes: True~~
~~    write_samples: True~~
-   module: gpath.genes.path
    name: Pathway
    ligand: Ligand
    protein: Protein
    torsion_gene: T
    rotamers_gene: R
~~    nm_gene: NM~~
    min_step_increment: 0.8
```

Tip: As said, you can use a custom pool of conformations for one of the molecules (ligand or protein), or both. You would need to modify one or both *molecule* gene/s and the *path* gene accordingly.

13.4 3. Indicating the initial conformation of the pathway

Sometimes, for example, when you are calculating an unbinding pathway from an already known position, you may want to fix the initial conformation of the ligand/protein. This can be done by indicating the file-path of the initial conformation in the parameter *first_frame* of the corresponding *molecule* gene:



```
genes:
-   module: gpath.genes.molecule
    name: Ligand
    path: ./mol_files/ligand_conformers
    first_frame: ./mol_files/ligand_conformers/conformation_00.pdb
-   module: gpath.genes.molecule
    name: Protein
    path: ./mol_files/protein_conformations
    first_frame: ./mol_files/protein_conformations/conformation_00.mol2
```

CHAPTER 14

Developers guide

14.1 Introduction to Genetic Algorithms

The GA in GaudiMM stands for Genetic Algorithm, a search heuristic inspired by natural selection that is used for optimization processes.

Genetic Algorithms use a biologicist terminology. Each candidate solution to the problem is considered an **individual**, which is part of the so-called **population** (the set of all candidate solutions).

The initial population is generated from scratch, almost always randomly. These individuals also comprise the first generation of the evolution process. As in nature, only the **fittest** survive. The survival process is simulated with an evaluation function, that tests them against the optimization variable(s). This is called **selection**.

- How do we create an individual? With one or more **genes**, naturally. Genes describe how an individual should look like, of course!
- How do we evaluate that individual? With one or more **objectives**.

Also, as in nature, the fittest individuals are allowed to **mate** (exchange their defining values), and **mutate** (spontaneously modify their own defining values). This adds some more variability to the process, and that is key to survival.

After a number of generations repeating the process of selection, choosing the fittest over the weakest, we will obtain better and better solutions to our problem. Since it's all heuristics, it's up to us to stop at some point. We won't probably get *the* solution, but we can live with pretty good ones, right?

14.1.1 The One Max Problem

Ok, that was a lot of biology and we are trying to code, I get it. Let's explain a classical GA example, the trivial **One Max Problem**, adapted from the original deap documentation.

In this problem, we have a list of integers that can be either 0 or 1, and we want to obtain a list full of 1s. So, in this example, we have individuals defined by a single **gene** and evaluated with a single **objective**.

We build individuals with the gene **onemax**:

```python
import random.randint
def onemax(size=5):
    return [random.randint(0, 1) for i in range(size)]
```

```python
adam = onemax(5) # returns [0, 0, 0, 1, 0]
eve = onemax(5)  # returns [0, 1, 1, 0, 0]
```

The objective is also trivial. We have to maximize the sum of the numbers inside a given individual:

```python
def evaluate(individual):
    return sum(individual)
```

So… which is one is fittest, `adam` or `eve`? Obviously, `eve`:

```python
evaluate(adam)   # returns 1
evaluate(eve)    # returns 2
```

Of course, an initial population is usually larger! At least, a hundred individuals. With such a trivial case, given a big enough population, we may obtain the solution in the first generation by pure change. However, we must not rely on the initial population as the only diversity source.

Additional diversity is achieved with the mutation and mating operations, implemented as additional functions:

```python
def mate(a, b):
    """ Let a and b mate, in hope of fitter children """
    i = crossover_point = random.random() * min(len(a), len(b))
    c, d = a[:], b[:]
    c[:i], d[i:] = d[:i], c[i:]
    return c, d

def mutate(individual, probability):
    """ Spontaneous mutation at random places can result in a fitter individual """
    return [random.randint(0, 1) for i in individual if random.random() < probability]
```

Let's see how this is useful:

```python
cain, abel = mate(adam, eve)
# cain = [ 0, 1, 1, 1, 0 ]
# abel = [ 0, 0, 0, 0, 0 ]
evaluate(cain) # returns 3
evaluate(abel) # returns 0
```

See? `adam` and `eve` gave birth to `cain` and `abel`. `cain` had luck and inherited the good parts, while `abel`… Well, he was not that lucky. In the next selection process, `cain` will be selected over `abel`, and probably over its own father `adam`. Now, the population (`cain` and `eve`) as a whole is fitter, with an average fitness of 2.5. That's higher than the average in the previous generation (1.5). Evolution!

Mutation works similarly:

```python
enoch = mutate(cain)
# enoch = [ 1, 1, 1, 1, 0]
seth = mutate(eve)
# seth = [ 0, 0, 1, 0, 0]
```

Take into account that mutations can be beneficial, like in the case of `enoch`, but also detrimental, as in the case of `seth`. Some of them will contribute to evolution, and some of them not. Lucky ones will be selected, the others, discarded.

By the way, deap already defines some mutation and mating operators for you that will work in most cases. So, hopefully, this part will be trivial.

And that's it! Deap does the rest! So, to sum up, you only need to worry about:

- How to define your individuals.
- How to evaluate them.
- How to implement mutation and mating (normally, with deap built-in operators).

If you want to know more about Deap and Genetic Algorithms, go check their documentation. It's great!

14.2 Our implementation

GaudiMM is built as an extensible and highly modular Python platform. Although the main focus is Chemistry and molecular design, you can use your own genes and objectives. You can think of GaudiMM as a new API for deap that provides an object-oriented interface to easily create new individuals and objectives.

In `deap` an individual can be any Python object (check their overview and GA examples), which is a very versatile approach, but it tends to be very limited when your individual gets complex. For example, if an individual needs to be defined by several genes with different mutation strategies.

In GaudiMM, each **individual** is a `gaudi.base.Individual`, which is a very (bio)fancy name for a list of `genes`. To create a `gene`, you just subclass `gaudi.genes.GeneProvider` and define the needed methods: `express`, `unexpress`, `mutate`, and `mate`. The `gaudi.base.Individual` class then provides some wrapper methods that call the respective counterparts in each `gene`.

To evaluate the fitness of an individual, you must first define the set of evaluation functions. Each function is called `objective`, and you keep them inside a `gaudi.base.Environment`.

To create a new `objective`, you have to subclass `gaudi.objectives.ObjectiveProvider`, which provides a very simple interface: `evaluate`. Define your function there, and that's it!

Todo:

- Tutorial: How to create your own gene
- Tutorial: How to create your own objective

CHAPTER 15

API documentation

15.1 gaudi.cli

15.1.1 gaudi.cli.gaudi_cli

15.1.2 gaudi.cli.gaudi_run

15.2 gaudi.genes

These are the built-in genes in GPathFinder. You can also build your own, but these are ready to use.

15.2.1 GaudiMM standard Molecule gene

15.2.2 GPathFinder Path gene

15.2.3 GPathFinder Path_torsion gene

15.2.4 GPathFinder Path_rotamers gene

15.2.5 GPathFinder Path_normalmodes gene

15.2.6 Base class for all genes

15.3 gaudi.objectives

These are the built-in objectives in GPathFinder. You can also build your own, but these are ready to use.

15.3.1 Path scoring objective

15.3.2 Base class for all objectives

15.4 gaudi.algorithms

15.5 gaudi.base

15.6 gaudi.box

15.7 gaudi.exceptions

15.8 gaudi.parallel

15.9 gaudi.parse

15.10 gaudi.plugin

15.11 gaudi.similarity

15.12 gaudi.path_similarity

15.13 gaudi._cpdrift