School of Sciences
Computational Physics and Biophysics Group

Running the Monte Carlo Pathway Search for Setting TASS Simulation Windows in GROMACS

Abhishek Acharya
November 2024

Abstract

This document provides a description of the steps for setting up the inputs for the TASS simulations. The core MCPS procedure used in the setup process is adapted from an earlier study by Haloi et al. 2021. The present protocol can be used to obtain initial configurations for running enhanced sampling simulations using GROMACS. In this document, we describe briefly the full procedure for generating inputs for simulations of antibiotic permeation through a membrane channel.

0.1. Generation of possible solute positions and orientations within the channel.

This step uses the script `prepINP.py`. The script generates all possible configurations of a solute within a channel(s). The configurations are generated by first generating rotations of the solute that are points on a Fibonacci sphere, followed by translation of the sphere over grid points specified by the user. Finally, the configurations that lead to clashes and ring piercings are removed. The code takes as input a param file that specifies the number of channels, solute ID, the number of Fibonacci points to use, head and tail atoms of the solute, and XYZ grid dimensions (see details below on the parameter file). For info on the command line flags run: `python prepINP.py -h`. An input PDB file containing the fully equilibrated system is also required as input. A full equilibration is important before initiating any enhanced sampling calculations. For GROMACS, you can generate the PDB by running *gmx editconf* on the equilibrated configuration. And optionally, the `--remove_vs` flag can be set to `True`, if the input has virtual site atoms and one needs to remove those before the enhanced sampling run. By default, this is set to `False`.

0.1.1. Input PDB file for the full system

As in case of normal TASS simulation (or other methods such as umbrella sampling, metad, etc.), we need a fully equilibrated system containing the channel embedded within a lipid membrane and the antibiotic. Typically, the inputs for different umbrella windows are generated using pulling simulations. Here we employ an alternative scheme for the same. The scripts however expect an equilibrated system in the PDB format. In gromacs, the default `.gro` format can be converted to a `.pdb` format using *gmx editconf*.

0.1.2. Parameter file

A parameter file that specifies the required additional inputs for running the setup. The list of valid parameters is as follows. Note that some of the parameters are not used by the `prepINP.py` script, but are needed in the subsequent steps. Thus, it makes sense to prepare the full parameter file for convenience. An example file `params_mcps.dat` is provided.

- `SOLUTE` This is used to specify the residue name for the solute of interest.
- `NFIB` Number of Fibonacci points used to generate all the possible orientations of the antibiotic within the sphere.
- `HEAD_ATM` The atom name for the atom to be used as the head atom for calculation of the antibiotic axis vector
- `TAIL_ATM` The atom name for the atom to be used as the tail atom to calculate the antibiotic axis vector.
- `ELF_HEAD_RESID` The residue number is to be used to specify the head atom of the internal vector along the transverse electric field along the channel. The vector is used as a reference for the calculation of the azimuthal angle for each antibiotic orientation.
- `ELF_TAIL_RESID` The residue number to be used to specify the tail atom of the internal vector that is along the transverse electric field along the channel.
- `X_DIM` Specify the grid box dimension along the x direction specified as [spanx-, spanx+, binwidth].

- `Y_DIM` Specify the grid box dimension along the y direction specified as [spany-, spany+, binwidth].
- `Z_DIM` Specify the grid box dimension along the z direction specified as [spanz-, spanz+, binwidth].
- `N_CHANNELS` The number of channels in the input PDB file.

0.1.3. Additional Flags

The `--remove_vs` flag can be optionally set to `True` if the input PDB has virtual site atoms and one wishes to remove them before the setup. The default is `False`.

0.1.4. Output

The script outputs coordinate files in `.pdb` format that contains all the antibiotic-channel configurations generated by the code and the corresponding files containing the total number of antibiotic configurations (`NFRAMES.dat`). The number of `.pdb` files is equal to the number of channels. Note: The choice of `NFIB` value in the param file affects the volume of the output file generated. In our experience, a value of 10-12 is sufficient and higher values are unnecessary. We have used a value of 12 for our application, which provides a good balance between the throughput and computational cost of generating and minimizing all configurations.

0.1.5. Usage

```
python prepINP.py -c system.pdb -p params_mcps.dat
```

0.2. Energy Minimization

In this step, we will perform energy minimization of all configurations in the coordinate file(s) generated in the previous step. However, depending on the specific application the number of configurations may be between 100K to 500K. The `runEM.py` script processes the trajectory file to perform energy minimization on the configuration. Additionally, for the final minimized configuration it collects the information on the protein-drug interaction energy (ε), the z position of the antibiotic along the channel and the drug orientation in terms of the azimuthal (ϕ), and inclination (θ) angles. Before running this script, the user needs to prepare all the required inputs to the script carefully. The code assumes that the minimization is run using the GROMACS package. The code takes as input several arguments that are detailed below. For details of the respective flags, one can also run `python runEM.py -h`.

Practically, running this step on even 100-200K configurations using the present implementation takes a long time. It is therefore useful to divide the calculations into chunks that can be run over different nodes. For this, we provide an additional helper script `chunks_inp.py` described at the end of this section.

0.2.1. Parameter file

The input parameter file specifies the required additional inputs for running the setup. The list of valid parameters is the same as those described in a previous section (Sec. 0.1.2). See also the file `params_mcps.dat`.

0.2.2. Input trajectory

The trajectory file is generated by the `prepINP.py` script, or alternatively, the trajectory chunks produced by the `chunk_inp.py` are processed for the EM step. Note that `prepINP.py` may produce more than one trajectory file depending on the number of channels. However, `runEM.py` can be run only on a single trajectory file at a time.

0.2.3. System Topology and the channel position restraint file

Before running the script, the user should prepare the GROMACS topology file for the system containing the channel and the drug molecule. It is important to note that the minimization is performed in the gas phase with strong restraints on all the heavy atoms of the channel protein except for the side chains of residues that are within 3.5 angstroms of the drug molecule. For these atoms, a considerably weaker restraint is applied.

For setting up these minimization runs, we suggest that the user creates a `topology` directory within the working directory. This directory should contain all the topology files, including a separate position restraint file for the protein channel atoms. This position restraint file is modified by the script on the fly depending on the antibiotic-channel configuration that is being minimized. The full path to the main topology file (`.top`) file and the protein position restraint file (`.itp`) is to be supplied to the script using the flags `--input_top` and `--input_posre`, respectively.

0.2.4. Minimization MDP file

The gas phase energy minimization `.mdp` file needs to be supplied. Make a note of the `define` directive in the file and make sure that the protein position restraint filename supplied with the `--input_posre` flag correctly "included". For details, look up the GROMACS manual for the position restraints and the `include` mechanism employed for topology constriction. An example `em.mdp` file is supplied with the code. This file (appropriately modified to your system) should be in the working directory and supplied to the script with the flag `--minim_mdp`.

0.2.5. Protein Interaction Energy MDP file

After the minimization step, the protein-drug interaction energy is calculated for the minimized configuration using the `gmx energy` command. For this step, the script first uses `mdrun` with the `rerun` flag to obtain an `.edr` file contains the interaction energy terms for the protein and drug. An example `pie.mdp` file is supplied with the code. Be careful to check the contents of this file and ensure that the `energygrps` flag is correct and only contains the two groups corresponding to the protein channel and the antibiotic molecule.

0.2.6. Additional Flags

The script takes in additional flags such as the `--gmx_command` that can be used to specify the GROMACS prefix. By default, the code tries `gmx` or `gmx_mpi`. The flag `--cont` can be set to `True` if a continuation is requested. This is helpful in cases where the calculations have stopped due to memory issues on some hardware. If the calculations prematurely stop, the code writes out temporary files with the suffixes `_mintraj_temp.pdb` and `_PIE_temp.dat`. If `--cont` is set to `True`, additional flags `--trajcnt` and `--pie` must be used to supply to the script the two temporary files.

0.2.7. Output

The script writes out a trajectory file in `.pdb` format containing minimized solute-channel configurations, and the corresponding output `.dat` files with the ε, z, θ, and ϕ values for each frame.

0.2.8. Usage

For the initial run:

```
python runEM.py -p params_mcps.dat -c traj.pdb -t topology/topol.top
-pr topology/posre.itp -m em.mdp
```

For continuation run:

```
python runEM.py -p params_mcps.dat -c traj.pdb -t topology/topol.top
-pr topology/posre.itp -m em.mdp -ct True -trj trajfile_mintraj_temp.pdb
-pie trajfile_PIE_temp.dat
```

0.2.9. Helper script: `chunk.py`

The code divides an input trajectory file into several smaller chunks. Apart from the input trajectory PDB file, the code takes the number of frames in the trajectory as input. This was written out in the NFRAMES.dat file in the first step (see Section 0.1.4).

0.3. MCPS calculations

The `runMCPS.py` is a Python script implementing the Monte Carlo Pathway Search (MCPS) algorithm for identifying potential energy transition pathways that traverse the energy landscapes based on user-defined spatial and angular parameters. The script supports parallel processing for efficient computation.

0.3.1. Inputs

The script reads the *.dat files containing the ε, z, θ, and ϕ values to perform the MCPS calculation. The data points and frames with unusually high energy values (¿ 1000 kJ/mol) are removed. Briefly, the MCPS algorithm involves initializing from a random frame around one end of the channel and iteratively selecting neighboring frames based on the selected inclination, azimuthal, and z-coordinate step sizes, and performing Metropolis Monte Carlo acceptance testing for energy transitions. The script also takes as input the number of iterations of MCPS searches to attempt through the `--niter` command line flag.

0.3.2. Outputs

The script writes out the pathways specified by the corresponding frame numbers into the `transition_search.dat` file. Note that this numbering is according to the serial number in the final data accumulated from all the trajectory chunks after removing the high-energy data points. Therefore, also written out are the frame numbers that were removed from each trajectory chunk on account of high energies. This information is used in the next step to account for the discarded data and obtain the correct frame numbers as specified in the `transition_search.dat` file.

0.3.3. Usage

```
python runMCPS.py --niter 100000
```

0.4. Filter Transition Paths

The `filterTransitions.py` script analyzes and filters transition paths based on their energy and inclination values. It processes the data specified in the `transition_search.dat` file and extracts the ε, z, θ, and ϕ values of the associated frames for each trajectory. The average of the protein interaction energy ε at the constriction region (CR) is used to sort the loaded paths. Thereafter, the average inclination θ angle value in the CR region is used to classify the paths into two categories (path I and path II).. The script writes out the sorted and classified trajectories and the z positions for the top paths.

Note: This script is useful for only a very specific use case where we are interested only in obtaining trajectories corresponding to path I and path II type transitions.

0.4.1. Inputs

The script reads the `transition_search.dat` file and the full dataset for the ε, z, θ, and ϕ values from the disk. Also, the data on the frames that were discarded in the previous step are read and the final dataset with the valid ε, z, θ, and ϕ is compiled. In addition, the script accepts command line flags that specify the number of paths to write out for the Path I and Path II clusters (using `--npaths`) and the number of paths to load from the input `transition_search.dat` file (using `--nload`).

`Note:` The cutoff values for θ used for the classification step are hard-coded and can be modified by changing the `MEAN_INC_HIGH` and `MEAN_INC_LOW` variables. The region of the CR used for calculating the average ε and θ is specified by the `CR_CUT_HIGH` and `CR_CUT_LOW` variables in the script. For a new system, the user may need to choose the CR cutoff values to account for the CR region inside the channel correctly.

0.4.2. Outputs

The outputs of the script include the filtered transition paths written out as a trajectory file in PDB format; `pathI_transition_<index>_<serial>.pdb` and `pathII_transition_<index>_<serial>.`
for path I and path II clusters. Here, `<index>` specifies the position in the list of transition paths that is sorted based on average energies. It is the ranking for the paths in each of the clusters. The `<serial>` is the transition path ID that is specified for each path in the `transition_search.dat` file.

0.4.3. Usage

```
python filterTransitions.py --npaths 10 --nload 50000
```

0.5. Setup Simulation Windows

The `setupInputs.py` script can be used to set up input files for the umbrella sampling method (US) and its various flavors including the temperature-accelerated sliced sampling (TASS) scheme. The script can process transition trajectory files written out by the `FilterTransitions.p`
and integrates them with reference molecular configuration to generate simulation-ready systems.

0.5.1. Inputs

The code takes as input a single or a list of trajectory files using the `--trajlist` flag. By default, the code expects a single trajectory file which is equal to the default value for the `--nchannels` flag. A reference equilibrated structure is supplied using the `--refpdb` flag. This should be a fully equilibrated system that one intends to simulate. The flag `--paramfile` is used to supply the `params_tass.dat` file that specifies additional parameters needed for setup. The `-nchannels` flag specifies the number of channels for which the simulation windows need to be prepared. In the default case, when only a single input trajectory file is provided, the code automatically detects the channel and proceeds with the system preparation. If the reference input system (`--refpdb`) has more than one pore, the system by default selects the first one that appears in the reference PDB. If a user needs to set up simulations with a specific chain in a multi-pore system, this can be specified using the optional `--poreid` flag. This defaults to 1 which corresponds to the first chain in order of appearance.

0.5.2. Parameter file

The parameter file `params_tass.dat` specifies additional run parameters needed for the setting up the windows.

- `NUMB` Specify the number of windows to construct.
- `UPOS` The window mean positions along the z CV to be used for preparing windows.
- `BOXDIM` The box dimension in angstrom, as mentioned in the input PDB file supplied by the `--refpdb` flag.
- `RESNAME` The residue name of the antibiotic same as specified in the PDB file.

0.5.3. Other setup options

The default behavior of the script is to set up windows only for a single channel. The code also supports setting up simultaneous biased runs on multiple pores. If the system contains X number of channels, then X number of trajectory files can be supplied using the `--trajlist` flag. In this case, the `--nchannels` flag must be set to X. In such a multipore setup, the `--poreid` flag does not affect the output. If a multiple pore setup is requested, by default the script prepares the system such that the antibiotic molecule has the same window mean position in all three channels. Depending on the channel, this could lead to strong mutual interactions between the antibiotics. To reduce this effect, as staggered positioning may be requested using the flags `--staggered_placement` and `--staggered_shift`.

0.5.4. Usage

For a single pore setup with configurations from a pathI transition path data:

```
    python setupInputs.py --trajlist pathI_transition_1_298.pdb --refpdb system.pdb
--paramfile params_tass.dat
```

The above command would also work if the user has a system with multiple pores and is interested in setting up windows for a single pore. By default, the first pore is selected by the code as per the residue serial number. If a system contains multiple pores and the user needs to setup simulation windows for a specific pore, the following command can be used for selecting the third pore as per residue numbering.

```
    python setupInputs.py --trajlist pathI_transition_1_298.pdb --refpdb system.pdb
--paramfile params_tass.dat --poreid 3
```

For setting up simultaneous runs in all pores, the following command can be used for a 3 pore system.

```
python setupInputs.py --trajlist pathI_transition_1_2898.pdb
pathI_transition_1_1688.pdb pathI_transition_1_4728.pdb --refpdb system.pdb
--paramfile params_tass.dat --nchannels 3
```

For setting up simultaneous runs in all pores using a staggered placement of antibiotics, the following command can be used for a 3-pore system.

```
python setupInputs.py --trajlist pathI_transition_1_2898.pdb
pathI_transition_1_1688.pdb pathI_transition_1_4728.pdb --refpdb system.pdb
--paramfile params_tass.dat --nchannels 3 -SP True -SH 15
```