Quick guide to MC-SPQR code(v8.6)

Simón Poblete

The code is stored in the directory `MPI_mc_X.X`, where `X.X` is the version. It must be compiled with the command `./make_here` for the single processor runs. Its most important binaries are `MC` and `XYZENERG`, used for a simple run and calculating energies. Additionally, `./make_here -MPI` for the parallel version, which includes the simulated annealing executable `T_ANN` and a simple Monte Carlo run over several initial conditions, `MPIMC`. This compilation feature requires the modules `pgi` and `mpich` in the sbp workstations, and `openmpi` in the falisca cluster (modules can be loaded by typing `module load mymodule`).

The compilation must be done with some care. There are three makefile files, `Make_mpi` , `Make_no_mpi` and `Make_xyz` . In each of them, one can define certain flags in the line of `CFLAGS` . In some cases, the use of features like the frozen particles or secondary structure restraint require the presence of certain keywords in there before compiling. These words will be specified in the next sections.

The input files (which must be all in the same directory) are

- `input.mc`
- `energies.pms`
- `bonds.dat`
- `tab_energs` (Directory)
- `configs` (Directory)
- `xyz_inits`(Directory)
- `gly_pck.dat`(Optional)
- `mobile.lst`(Optional)
- `sec_strc.lst`(Optional)
- `sim_ann.pms`(Required for simulated annealing).

1 Output files

The configurations will be saved in the directory `configs` in binary format in the file `confs.mc` while some checkpoints are created in `xyz` format. The trajectory files are always appended!!! This means that keeping the directory `configs` empty before starting a simulation is extremely important otherwise the output files can easily be corrupted. The function of all these files will be explained below.

2 Binary files

- `MC` (single processor) Performs the integration of a simple simulation at constant temperature by Metropolis Monte Carlo. It has to be run as `./MC input.mc` . It generates the configurations in the file `configs/confs.mc` and, if it finishes, `lastmc.xyz` .

  It also creates the checkpoints `cmc.XXXXXXXX.xyz` , on which `XXXXXXXX` is the number of iteration written with 8 digits.

- `XYZENERG` (single processor) Calculates the energy of an `xyz` configuration stored in `xyz_inits/mc.p20.xyz` . Yes, it has to be called like that and stored there. It must be run as `XYZENERG X` , where `X` is an index. When `X` is zero, it calculates the total energy. `-1` gives the annotations. More options can be displayed by running `./XYZENERG help me` . It does not create any output file.

- `MPIMC` (multiple processor) It runs an MC simulation just like `MC` , but with a different initial condition per processor. These configurations must be stored in `xyz_inits` , which is described in the next section. Also, each independent initial condition has a different random seed. It must be run as `mpirun -np X ./MPIMC input.mc` , where `X` is the number of processors. The output files now contain the number of processor with two digits, as `confs.pXX.mc` and `lastmc.pXX.xyz` and `cmc.YYYYYYYY.pXX.xyz` .

- `T_ANN` Performs the simulated annealing simulation. It must be run as `T_ANN input.mc` . It requires the file `sim_ann.pms` . This file can be a single one, from which all processors take their parameters, or it can be different for each processor as `sim_ann.pXX.pms` , where `XX` is the processor number written with two digits.

  It generates the files `new_sim_ann.pXX.pms` , which contains the final parameters of the simulation. These files are saved with certain frequency, and are fundamental for restarting a long annealing simulation.

3 Input files

The simulation length will be `MC_STEPS` times `MC_BETWEEN_CHECKPOINTS` . That is, the code runs `MC_BETWEEN_CHECKPOINTS` Monte Carlo steps, saves a configu-

ration in the `configs.mc` file, and then runs again. This is repeated `MC_STEPS` times. However, `MC_STEPS` and `MC_BETWEEN_CHECKPOINTS` are defined in different files.

- `energies.pms` `MC_RCUT` (float float float float) contains the four cutoff radii for the non-bonded interactions: total, base-pairing, base-phosphate and excluded volume. No need to touch this. `MC_N_BOND_TYPES` (float float) is an obsolete variable. `VL_SKIN` (float) is the skin for the Verlet lists. Since they are deactivated , they are irrelevant.

  Finally, `MC_BETWEEN_CHECKPOINTS` (int) must be specified.

- `input.mc` Specifies some global features of the simulation. `DIMENSIONS` (float float float) requires three float numbers for the simulation box size, `TEMPERATURE` stands for the temperature (float), `MC_PH_XYZ` (float) is the maximum displacement of a phosphate particle in a MC trial, `MC_NT_XYZ` (float) is the maximum displacement of a nucleoside in a MC trial and `MC_STEPS` (int) is specified. `RANDOM_SEED` (int)is the random seed, which must be a positive integer, and finally, `MC_NT_ANGLE` (float float) is the maximum rotational angle, in radians, for the MC trial of a nucleoside rotation. It requires two arguments: the first is for the rotations around a random axis centered on the base or geometrical center, and the second, for rotations around the **z** axis of the base centered on the sugar.

- `bonds.dat` Specifies which nucleotides are connected through the backbone. The first column is the nucleotide index, for human readability. The second is the number of nucleotides which are bonded with it. The third is obsolete and it must be zero. Then, in the same line, a list of numbers must be introduced in case some bonds exist. They must be two numbers per bonded nucleotide. They specify the index of the neighbor and then the type of bond, which in this case is zero due to historical reasons. The order must be : index - type of bond - index - type of bond - and so on. The bonds must be specified in both directions: that is, if a is bonded with a, so is b with a.

- `configs` It is the directory where the trajectories and checkpoints are saved. It must be checked before running any simulation, since the trajectories append on top of the existent ones by default.

- `xyz_inits` It contains the initial condition. This has to be called `mc.xyz` , and it has to be written in `xyz` format. That is: in the first line, the number of atoms; in the second line, a comment, and then, the list of each atom starting with its type and the three coordinates. In this representation, the atoms must be written sequentially as: base center (type 0, 1, 2 or 3 for A, U, C, G.), **x** and **y** atoms for defining the base orientation (type -1 and -2, respectively), sugar (type -3) and phosphate (type -4). In total, there are 5 atoms per nucleotide, so the number of lines of the file must always be $N_{nt} \times 5 + 2$, where N_{nt} is the number of nucleotides which has

to be consistent with the number of atoms of the `xyz` file. In some cases, for visualization, the nucleotides are labeled with types A, U, C, G, X, Y, S and P. This is only for visualization and the code will complain if such initial condition is given.

If there is more than one processor, one can specify an initial condition for each of them, which added to their particular random seed, increases the efficiency of the sampling. In this case, the initial configurations are called `mc.pXX.xyz` , where `XX` is the processor number written with two digits (that is, 1 as 01 and 10 as 10, and so). In case these files are not present, the program will look for `mc.xyz` and use it for all processors. Since the random seeds are different, the trajectories will differ for each of them. If no initial condition is given, the program will use random positions and it will be a mess, but it won't necessarily crash.0

- `tab_energs` Here are contained the tables of the energies. For the moment, there is no need to touch them.

- `mobile.lst` (Optional). A list that contains the information about which nucleotides are mobile. It is composed of two columns of integers. The first is the nucleotide index, for human readability, while the second, and index that characterizes its motion. `0` means frozen, both nucleoside and phosphate, `1` means only the nucleoside is mobile, `2` means that only the phosphate is mobile and `3` stands for a completely mobile nucleotide. In the absence of this file, everything is mobile as expected. It requires the compilation flag `FROZEN` .

- `gly_pck.dat` (Optional). A list of information about glycosidic angle conformations and puckers. It consists of three columns: the first contains the nucleotide index, the second, the glycosidic angle conformation (`A` ,`H` , and `S` for anti, high-anti and syn, respectively) and the third, the sugar pucker (`3` for C3' endo and `2` for C2'endo). In the absence of this file, all nucleotides are set to anti - C3'endo.

- `sec_strc.lst` (Optional). It contains the information about the secondary structure restraints. In a similar fashion to the bond file, it consists of several columns. The first is the nucleotide index for readability. The second, the number of nucleotides that are restrained to interact with each of them. If this number is greater than zero, one must specify two consecutive numbers per restraint: the index of the second nucleotide and the type of restraint. `0` means that they are only allowed to interact through base-pairing interactions, 2allows only base-phosphate interactions between the base of the first and the phosphate of the second, 3 allows only base-phosphate interactions between the phosphate of the first and the base of the second, while `1` allows any kind of these three possible combinations. Each time a restraint is specified, it must be done also in the opposite sense, i.e., if `a` is restricted to interact with `b` , so it is `b` with `a` , and it must also be explicitly specified. If the number of particles restricted

to interact with a given nucleotide is `-1` , it means that this nucleotide won't be allowed to pair with anyone. This feature is activated with the compilation flag `SECONDSTC`.

- `sim_ann.pms`(Optional for `MC` / Obligatory for `T_ANN`). It contains the parameters for the annealing protocol. It's a single line with the following parameters : Initial temperature, minimum temperature, temperature scaling factor, current step, total number of steps, final energy of the previous step, scaling factor of MC moves and and index of how many times the temperature has been rescaled. The annealing performs a simulation on which it measures its minimum energy. After comparing the energy of two subsequent simulations, it will decrease the temperature by a multiplicative prefactor only if the minimum energy has not decreased with respect to the previous run. Once the temperature reaches a value equal or lower than the minimum value, it will be set to zero and the MC trial moves will become smaller by another prefactor. Most of these parameters are kept for monitoring what is going on during the annealing. The scaling factors for temperature and MC moves, as the initial temperature and the number of steps of the annealing are worth to adjust if one wants to try different annealing approaches.

4 Tools

Several tools are needed both for the analysis and the visualization of the results. The most basic ones are presented here.

- `analysis_template.c` (int) Since the binary configuration files are written in a particular manner, this file contains the basic functions for opening and reading the positions and identities of all the particles, storing them in arrays. It contains the structure to read them in a loop over the number of configurations that the binary file contain, so on each step one can perform an analysis over the coordinates of the particles. It has to be compiled with `lm` . The argument must be the number of configurations to be read from the input file, which is called `confs.mc` by default.

- `extract_trajs_NNT5.c` (char *, int, int, int). It takes a binary configuration file and writes it in `.xyz` format. The first argument must be the name of the configuration file. The second, third and fourth arguments stand for the number of configurations, the initial configuration to be saved (to skip equilibration steps, for example) and the number of steps to skip between each snapshot (since, in many cases, it is convenient to observe or analyze the trajectory every certain number of configurations).

- `add_topology.tcl` A tcl script to be used with VMD. Once a configuration or trajectory (in `xyz` format) is visualized, this script takes care of searching for the connected pairs and bond the particles properly. This

must be done provided that a native, or reference structure is contained in the same folder. This native structure must, of course, have the same sequence and number of nucleotides of the visualized structure, and in many cases, it can be the same. However, it must exist as an independent file called `native.xyz` . The script must be loaded from VMD using the command `sourceadd_topology.tcl` from the VMD console or from the terminal once VMD is open.

- `get_rmsd.bash` and `rmsd.py` A bash and python script to calculate the RMSD between two `.xyz` files. It is not very convenient since the coarse-grained RMSD is not rigorously defined, and this can also be done in VMD. Still, it is a relatively good indicator of how similar two structures are (aim for less than 4 Å!). While the python script must be untouched, the bash script has to be run as `bash get_rmsd.bash my_structure.xyz` . It is also required that a reference structure called `native.xyz` is present in the same directory.