

Implementation of a Replica Exchange Monte Carlo algorithm using Python

Document réalisé via L^AT_EX

REPORTER: M. GELLY J.-C. and GALOCHKINA T.

ROUAUD Lucas — Master 2 BI: Université Paris Cité

Abstract

Background THACHUK C. and *al.* describe the use of a Replica Exchange Monte Carlo algorithm. It shall permit, through Dill's H/P model, the prediction of protein folding on a 2D matrix. But only for an extremely simplified case of hydrophobic amino acid or not. In this paper, we discuss how we implement this program using Python language and why doing it in this such of manner.

Results When trying on small dataset, fraction of the one the authors used in their article, energy results are very different. This show that the effectuation of an unfavourable movement might be badly implemented, or need a huge amount of MC/REMC step.

Availability and implementation This paper and the corresponding algorithm with its documentation are available on a GitHub repository: https://github.com/FilouPlains/PROJECT_PY_MONTE_CARLO/

ABBREVIATIONS: *MC* = Monte Carlo; *REMC* = Replica Exchange Monte Carlo.

1. Introduction

In an article, THACHUK C., SHMYGELSKA A. and HOOS H. H. talk about a REMC algorithms in a context of protein folding, following Dill's H/P model [1]. Here, we implement this algorithm using Python language.

2. Materials and methods

To implement this program, Python 3.10.4 is used. The module numpy 1.23 is used for vector computing and the module tqdm 4.64.0 to show the estimate time to perform a Monte Carlo algorithm.

The probability of doing an unfavourable movement is given by:

$$\begin{cases} 1 & \text{if } \Delta E < 0 \\ e^{\frac{-\Delta E}{T \times K_B}} & \text{else} \end{cases}$$

with K_B = Boltzmann's constant; ΔE = energy variation between two conformations.

While the probability of doing a replica exchange is given by:

$$\begin{cases} 1 & \text{if } \Delta \leq 0 \\ e^{-\Delta} & \text{else} \end{cases}$$

With Delta equal to:

$$\Delta = \left(\frac{1}{T_j} - \frac{1}{T_i} \right) \times (E(c_i) - E(c_j))$$

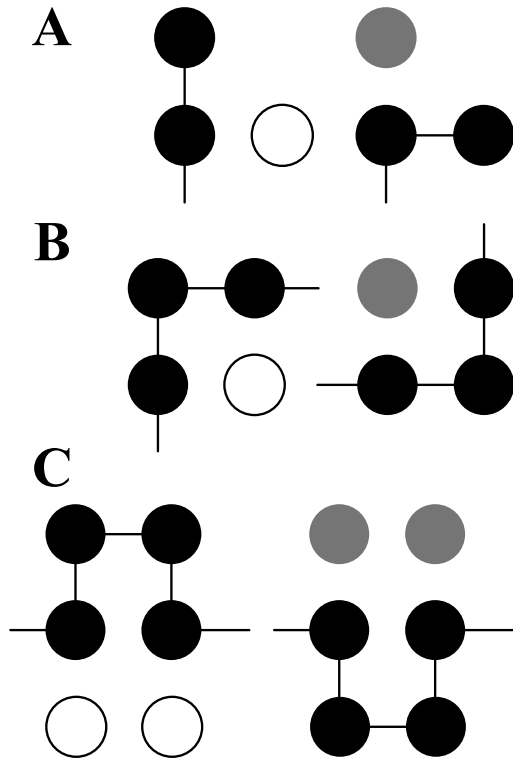


Fig. 3: All possible VSHD move set. Black = Actual position; grey = old position; white = position to move next step; A = end move; B = corner move; C = crankshaft move.

The [fig. 3] describe all implemented movement. To do so, three object where create. One that stock coordinates, an other amino acids and a last the sequence, which enable communication with those two previous one. Like that, we do not manipulate matrix, which has a complexity of $O(N^2)$.

To finish, we produce a .csv file to manipulate the data (especially energies) and optionally a .mol2 file, which contains all "frame" during a movement. Like that, with a molecule visualization software like PyMol or VMD, it is possible to observe the trajectory.

3. Results

First, the [fig. 1] show that conformation are sometime lost or blocked while doing a MC algorithm. For instance, the [fig. 1 : A] show that the conformation is some-

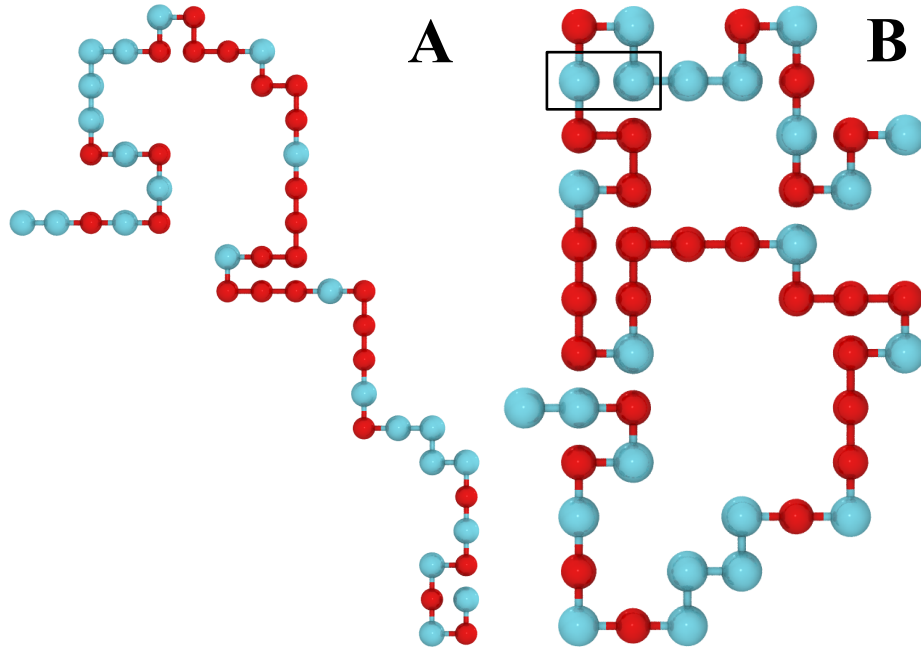


Fig. 1: Conformation example found by MC algorithm, for a same starting sequence. Red = polar residues; blue = hydrophobic residues; A = the sequence is unfolded and lose energy; B = the sequence have one hydrophobic interaction (black square), but it appears that polar residues are "attract" inside the protein instead of outside.

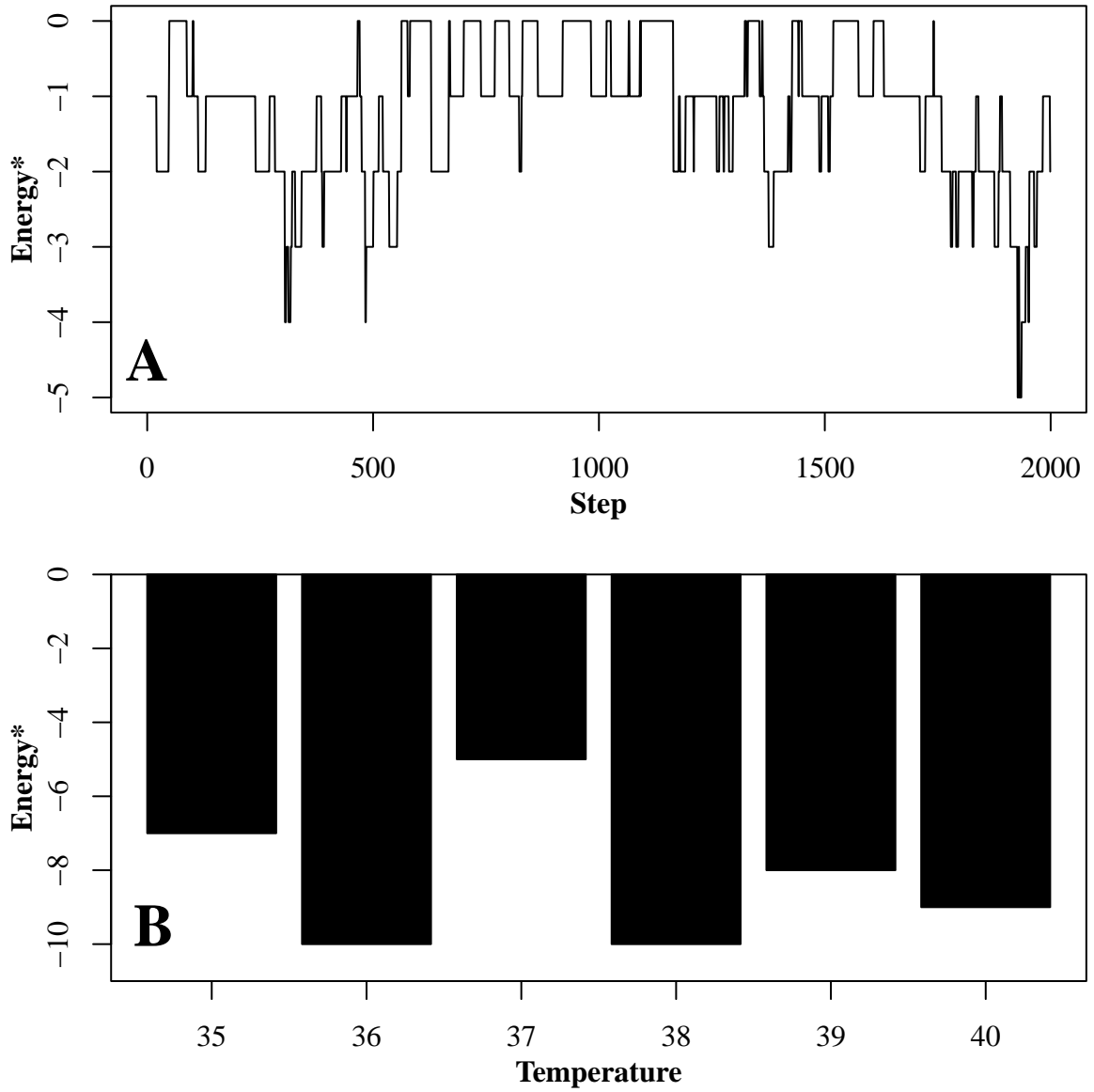


Fig. 2: Graphics obtain after MC algorithm. *A* = energy* in function of the step; *B* = bar plot of min energy obtain in function of the temperature.

how lost (like a denaturation of a protein) by a linearization; while the [fig. 1 : B] show that the hydrophobic residues are going outside, which provoke a diminution of the energy. Note that those conformation are obtained at the end of a 2,000 step MC algorithm.

To continue, when we observe the [fig. 2 : A] we see that the protein do not stay in "big" local minimum, but tend to do the

contrary. Also, when tacking all minimum energies values on the [fig. 2 : B], we observe that they are not that low compare to the original article of THACHUK C. and *al.*. Those observations tend to show that the algorithm search well minimum, but not efficiently and do not "stay in them". In other words, we found a local minimum, but not the best one.

4. Discussion

Firstly, the pull move was not implemented, as far as it was too complex to do so. The general method to do it to shift diagonally two residues and move their left or right followers (using recursion) until conditions are met: the first one is that all amino acids have been moved until end of chain; the second one is that one amino acid after movement is still at a distance of 1 of his neighbour. The thing was there's too much conditions to take in consideration, making quite hard to find a good, general and functional solution.

Secondly, parallelization is never used in the algorithm. Even though the algorithm is fast, there are huge amount of time to win by doing it, especially when doing a REMC. Indeed, multiple replica with different temperature are independent for a given number of step (until exchange if there is). For example, let's say that we have a PC with 8 cores. When executing the program, it takes less than one core. With parallelization achieved, it's possibly a division of the time's computation by eight!

Lastly, project organization and object

communication isn't quite perfect. Not all native Python's function are well-used (like magic method to work on chain list). Thus, even if in this case the implementation has flaws, it's still better and faster than working with a matrix system.

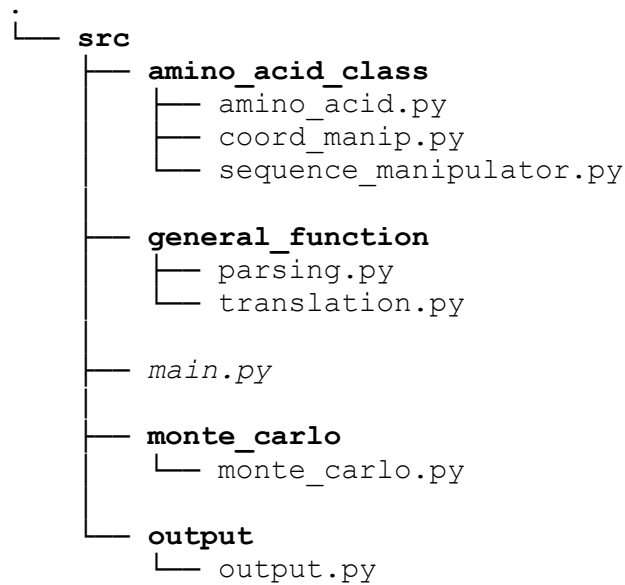
5. Conclusion

The algorithm is indeed implement, but not perfectly at all. There's some hints showing that the program is not good at finding global minimum and keeping the optimal conformation. You may also notice that only MC algorithm was analysed, but not the REMC. This is done like so to facilitate and accelerate data production.

6. References

- [1] Thachuk, C., Shmygelska, A. & Hoos, H. H. A replica exchange Monte Carlo algorithm for protein folding in the HP model **8**, 342. URL <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-342>.

Appendix



Appendix I: Dependencies tree. *Bold* = directory; *italic* = main file, the one used to launch the program.

The [app. \[I\]](#) show all the Python programs create during this project. To have the help, simply use:

```
python3 src/main.py -h
```

Or use:

```
python3 src/main.py -help
```

After that, you can launch the program. There are a lot of options details into the `README.md` of the project. The minimal functional command is the next one:

```
python3 src/main.py -i AA -s 1 -o file.csv
```

We give a sequence of two AA amino acids, then we ask for 1 step of MC. Finally, results are output into a `file.csv`. This format has been chosen as far as he is very easy to parse and have pre-built function to manipulate it in some cases (like in Python or R for example).