$\mathcal{O}(1)$ Lennard-Jones sampling

Simon Vionnet
(Dated: June 30, 2023)

I. INTRODUCTION

System with many particles in interaction can be hard to treat in an analytic way, it is even harder when those particle interact through long range potential such as the Lennard-Jones one eq. (1). Thankfully, those systems can be studied numerically with the Monte Carlo method that use random exploration of a sample space to compute interesting observable.

$$U_{LJ} = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{1}$$

For the rest we will consider $\sigma = 1$ and $4\epsilon = 1$

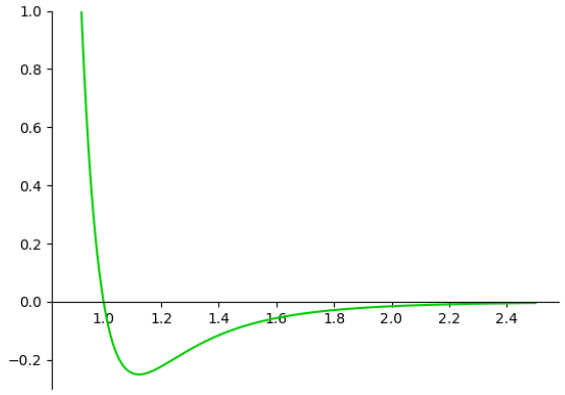

FIG. 1. Lennard-Jones potential

Even in numerical methods some are not applicable in reasonable time, the direct sampling for example can be done for few particles but it become unusable quickly. So to sample from the Boltzmann distribution one can consider using Markov chains (random chain of states where the state at step n depends only from the step at $n-1$, some talk about chain without memory). Some of the commonly used Markov Chain Monte Carlo will be discuss later.
To sample a correct distribution the algorithm has to respect the global balance condition eq. (3), which state that the probability flow going to a state is equal to the flow going outside of it. An important remark is that for each pair of state the total flow between them can be non zero, when this effective flow is zero we respect a condition that is more restrictive than the global balance, the local balance eq. (4). This condition are highly related to the method chose for the implementation. As shown in eq. (2), the flow is express thanks to the transfer matrix $\mathbb{P}(j,i)$ of the process, which give the probability to switch from state j to state i.

$$\mathcal{F}(j,i) = \pi(j)\,\mathbb{P}(j,i) \tag{2}$$

$$\sum_j \mathcal{F}(j,i) = \sum_j \mathcal{F}(i,j) \tag{3}$$

$$\mathcal{F}(j,i) = \mathcal{F}(i,j) \tag{4}$$

For rest of the discussion we aim to compare time complexity of different algorithms as functions of the number of particle in interaction denoted N_{part}.

II. THE COMMONLY USED ALGORITHMS

In this first part we will discuss some algorithms that can be used to sample Lennard-Jones systems starting from the most simple algorithm the Metropolis Monte Carlo Alg. 1 (`Metropolis`). In one one step, a move $(X \to X')$ is proposed with an a priori probability and is accepted with the probability $\mathbb{P}^{Met}(X,X')$ given in eq. (5), in which π is the Boltzmann weight related to the full potential.

$$\mathbb{P}^{Met}(X,X') = \min\left[1, \frac{\pi(X')}{\pi(X)}\right] \tag{5}$$

procedure `Metropolis`
input x
$x' \leftarrow x$
$\delta \leftarrow$ `ran`$(-1,1)$
$\Delta U \leftarrow U(x+\delta) - U(x)$
$\lambda \leftarrow \min\{1, \exp[-\beta\Delta U]\}$
$\Upsilon \leftarrow$ `ran`$(0,1)$
if $\Upsilon < \lambda$**:** $x' \leftarrow x + \delta$
output x'
——

Algorithm 1: `Metropolis`.

From the definition of Metropolis filter, we can show that this algorithm respect the local balance at one condition, the a priori probability must be symmetric.
From the pseudo code we can see that for each step, the time complexity is given by the time complexity to compute the potential change ΔU. Since we are moving one particle at a time, it can be compute in a time

$\mathcal{O}(N_{part})$ and so it is for one step.

The Metropolis algorithm as it is the easiest method will be the reference for our comparison in the following.

In fact Metropolis method is one of the elementary bricks for the Multiple Time Step Monte Carlo Alg. 2 (`MTSMC`) presented by Berne and Hetényi in [1]. MTSMC also work with the factorization of the potential U in two part U_I and U_{II}. In their paper they distinguish U_I and U_{II} as a rapidly changing potential and a slow changing one. The first term is used to compute the proposed moved by doing a Metropolis chain of n steps using the Boltzmann weights derived from U_I, the resulting position is then accepted with the Metropolis filter for the slow changing potential. In the case of the Lennard-Jones potential, U_I is the short range interaction, the potential truncated at the radius r_s and U_{II} is the far range contribution(the potential above r_s).

procedure `MTSMC`
input x
$x' \leftarrow x$
$y \leftarrow x$
for $i = 0, ..., n-1$:
  $\delta \leftarrow \texttt{ran}(-1, 1)$
  $\Delta U_I \leftarrow U_I(y+\delta) - U_I(y)$
  $\lambda = \min\{1, \exp[-\beta \Delta U_I]\}$
  $\Upsilon \leftarrow \texttt{ran}(0, 1)$
  if $\Upsilon < \lambda$: $y \leftarrow y + \delta$
$\Delta U_{II} \leftarrow U_{II}(y) - U_{II}(x)$
$\lambda = \min\{1, \exp[-\beta \Delta U_{II}]\}$
$\Upsilon \leftarrow \texttt{ran}(0, 1)$
if $\Upsilon < \lambda$: $x' \leftarrow y$
output x'

Algorithm 2: `MTSMC`, One single step between two sample point

As shown in [1], this algorithm respect local balance between the end points of the short range chain. Note that it is not possible to sample point during the short range step without inducing bias in the sampling and resulting on a non correct algorithm. One can also question the pertinence of doing the small chain as it is clear that we keep a time complexity $\mathcal{O}(N_{part})$.
In their paper they achieve to show a CPU time speedup, they also adjust their raw time with a diffusion coefficient using similar methods as in [2]. MTSMC is so almost 5 time faster than what we can expect to be a Metropolis in their paper.

Both previous algorithm rely on proposed move than is accepted or refused, with principle of consensus, we can factorize the problem and work with wisely choose and work with the Boolean variable (instead of multiplying the probabilities together, we will accepted the move if it is accepted for all the terms separately). It can be resume with the following truth table.

U_I	U_{II}	$U_I + U_{II}$
Refused	Refused	Refused
Refused	Accepted	Refused
Accepted	Refused	Refused
Accepted	Accepted	Accepted

Pushing the consensus idea with lifting the motion (the direction of the motion is fixed and is re-sampled once in a while) we can then work with piece wise deterministic simulation. This is the essence of Event Chain Monte Carlo (ECMC).
To explain it, let consider one particle moving along positive x direction interacting with one single other particle (going to multiple particles will be simple considering the consensus principle). Let note that the direction being fixed, the potential seen by the moving particle is not the one given in Fig. 1 but will be more like in Fig. 2.

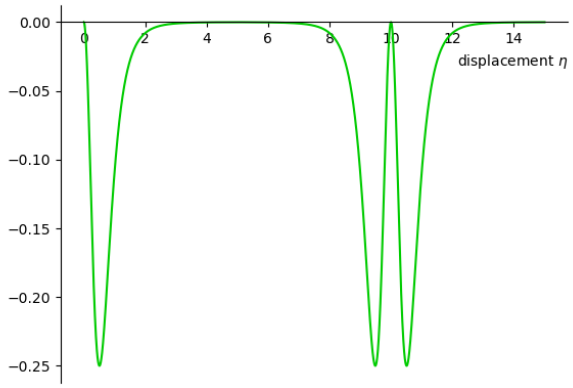

FIG. 2. Potential seen by a particle moving in a fixed direction along its displacement in a periodic boundary box. Here we have particle moving along x axis at $y = 0$ with a second particle (source of the interaction) at position $(0, 1)$

In the ECMC method, describe in [3], the moving particle will make infinitesimal displacement ds until the move is rejected. Let consider the particle doing small steps of length η, the probability to stop at the m-th move is given by $p(m) = \prod_{k=1}^{m-1} \exp(-\beta \Delta U^+(k))(1 - \exp(-\beta \Delta U^+(m)))$ with $\Delta U^+(k) = \max[0, \Delta U(k)]$ where $\Delta U(k)$ is the potential change induced by the k-th step. In the limits of $\eta \to 0$ and by changing from variable m to the total positive potential change U_M^+ express in eq. (**??**) (shown in Fig. 3), we have the following law $p(U_M^+) = \beta \exp(-\beta U_M^+)$ and so the potential change to

do before the rejection.

$$U_M^+(\eta_M) = \int_0^{\eta_M} \max\left[0, \partial_x U(\vec{r} + s\vec{e}_x)\right] ds = \int_0^{\eta_M} q_{at}(\vec{r} + s\vec{e}_x) ds \tag{6}$$

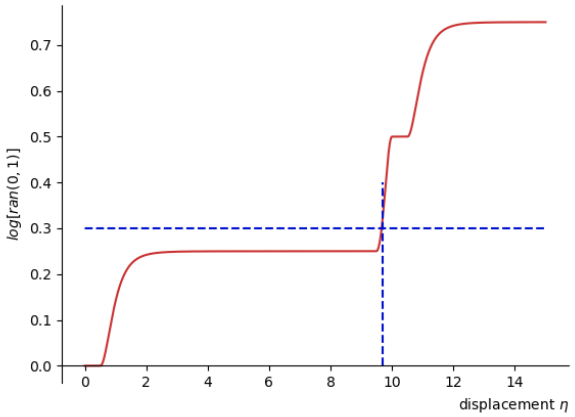

FIG. 3. Integration of the rejection which is the potential seen by the particle where we keep the increasing portion of it shifted to keep continuity.

To complete the ECMC process we then need to invert the cumulative rejection rate. This cannot be done by an analytic formula due to the periodicity of property of the system. Some methods can be implemented to compute the needed displacement.
The first one is the straight forward method, we will displace the particle by small steps Δs, will add up the positive potential change through the displacement and finally stop when we reach the wanted cumulative rejection rate. This methods has has two flaws, the total displacement is given with an accuracy of Δs and it is particularly slow for particles highly separated.
For our second method, we will inspire from the method used in JeLLyFysh (JF) [4] to inverted the real Lennard-Jones potential. We adjust the inverting function in the potential python class to take in account the periodicity. This methods are *findDisplacement* in Algorithm **??**
We then have our piece to complete the ECMC procedure for more than two particle. Note that we will make event chain of fixed length and sample only the ending of each chain (even if we can sample more points in the chains).

So to have the procedure from the two particles methods, we just have to do it for the possible pairs involving the chosen moving particle. To ensure the consensus along the chain, we then keep the lowest displacement and change the active particle for the one triggering the event. One can discuss what happen if two different particles trigger an event at the same time

procedure `ECMC`
input X
$a \leftarrow$ **choice**$\{particles\}$
$dir \leftarrow$ **choice**$\{\mathbf{e_x}, \mathbf{e_y}\}$
$Todo \leftarrow ChainLength$
while $ToDo > 0$**:**
  $Ds \leftarrow ToDo; nextPart \leftarrow a$
  $\Delta U^+ \leftarrow -\log\left[\mathtt{ran}(0,1)\right]$
  for $target = particles \neq a$**:**
    $\Delta s \leftarrow findDisplacement\left(a, target, \Delta U^+\right)$
    if $\Delta s < Ds$**:**
      $Ds \leftarrow \Delta s$
      $nextPart \leftarrow target$
  $X \leftarrow DoDisplacement(X, a, Ds) ToDo \leftarrow ToDo - Ds$
  $a \leftarrow nextPart$
output X

Algorithm 3: `ECMC`. Where X is ca configuration of particles positions that will be updated by the procedure

but those type of event are especially rare and since ce are working with numerical number, the space is discrete.

This algorithm still have a time complexity of $\mathcal{O}(N_{part})$ per step, with steps that can be particularly slow. So what is the point of presenting it? This algorithm will be use as a step to the last algorithm that will be discuss here, the Cell Veto Monte Carlo (CVMC) but first let make a small check point and see if all the three algorithms already presented have compatible results.

	mean Separation	standard deviation
Metropolis	3.89147	0.00429
MTSMC	3.88850	0.00284
ECMC	3.89148	0.00992

TABLE I. Mean separation between pair of particles obtain for 100 runs of 10^5 steps for each algorithm. We run the algorithms with 4 particles in a periodic box of size $L = 10$ with an inverse temperature $\beta = 1$. The MTSMC parameter was $n = 4$ and the ECMC chain length was 1.0. Finally for the MTSMC and the Metropolis, the step size was taken with a uniform distribution in $[0, 1]$

We can see that those three algorithm have compatible results. Note that it is note given with the best precision, more runs of each algorithm will be needed to have a more accurate value of the mean separation.

III. CELL-VETO

We have presented three algorithm that can sample a Lennard-Jones system, they all have a time complexity of $\mathcal{O}(N_{part})$ per step and so share the problem to not have the possibility to sample a system where many particle interact. We will here describe an algorithm based on the ECMC that can be implemented with a time complexity $\mathcal{O}(1)$.
Firstly introduce S.C.Kapfer and W.Krauth in [5], the cell veto algorithm Algorithm **??** is based on ECMC but the event will at first be trigger at cell level and then be verified as a true or a fake event.

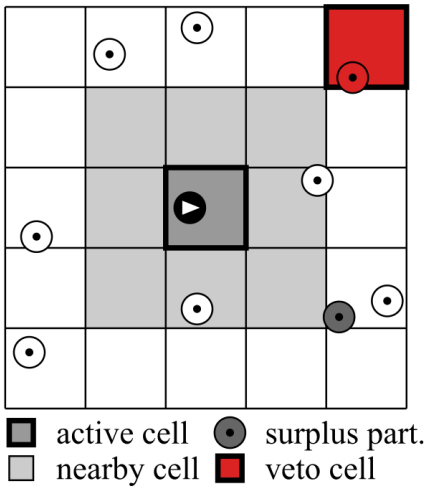

FIG. 4. Image extract from [5] to explain the Cell Veto algorithm. The events are trigger at cells level with a bounding event rate for each possible target cell with the active cell, the event is than confirmed at the particle level in the vetoing cell. Neighboring cells and surplus are treated separately.

The far field is then treated with a bounding potential (a potential for which the derivative along the chosen direction is always bigger than the derivative of the actual potential). In our case, we will have a piece wise linear bounding potential, ie. the derivative of the bounding potential will be constant on a cell.

To accept it as a true or a fake event one need to compare the event rate at a cell level Q_{at} with the one at particle level $q(\vec{r}_{at})$. If a particle exist in the triggering cell, the event is accepted with a probability $\frac{q(\vec{r}_{at})}{Q_{at}}$ Fig. 5.

The neighboring cells and the surplus particle will be treated individually. The contribution in time will be constant for a given cell size as the number of nearby cells will remain constant. The contribution of the surplus particle is also negligible as we can chose a clever cell size as a function of the particle density to ensure that surplus particles will be very rare.

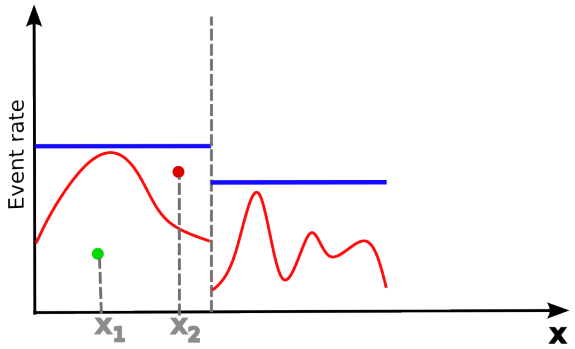

FIG. 5. In this schema, the blue line is the bounding potential event rate and the red line correspond to the one of the actual potential. In the derivative method, a displacement is sample thanks to the bounding potential (x1 or x2). Then the position below the blue line is sample randomly, it is consider a true event when it is below the red line (like the green point), if the point is above, the event is a fake one..

procedure `Cell-Veto`
input X
$a \leftarrow$ **choice**$\{particles\}$
$dir \leftarrow$ **choice**$\{\mathbf{e_x}, \mathbf{e_y}\}$
$Todo \leftarrow ChainLength$
while $Todo > 0$**:**
  $ds \leftarrow Todo$
  if $FrontierDistance < ds$**:** $ds \leftarrow FrontierDistance$
  $\Delta s \leftarrow -\log[\texttt{ran}(0,1)]/Q_{tot}$
  while $\Delta s < ds$**:**
    $targetCell \leftarrow WalkersSampling$
    if $\texttt{ran}(0,1) < \frac{q(\vec{r}_{at})}{Q_{at}}$**:**
      $nextPart \leftarrow CellOccupant[targetCell]$
      $ds \leftarrow \Delta s$
      break
    $\Delta s \leftarrow \Delta s + -\log[\texttt{ran}(0,1)]$
  $\Delta U^+ \leftarrow -\log[\texttt{ran}(0,1)]$
  for target in surplus or neighborhood**:**
    $\Delta s \leftarrow findDisplacement\,(a, target, \Delta U^+)$
    if $\Delta < ds$**:**
      $nextPart \leftarrow target$
      $ds \leftarrow \Delta s$
output X

Algorithm 4: `Cell-Veto`. As for ECMC, we will sample point at a fixed chain length.

This is enough to ensure a time complexity of $\mathcal{O}(N_{cell})$ but we will have $N_{cell} > N_{part}$ to ensure the rarity of surplus particle but we are not far from the promise land of a constant time complexity. We will need to have a clever way to select the triggering cell with a known displacement.

To fasten the process, we will treat the far field all together instead of each cell separately. We will trigger a possible event at the rate $Q_{tot} = \sum_{targetcells} Q_{at}$, this is possible because we consider infinitesimal steps.

With this event trigger at the level of the far field, we have to scale down to the cell level, a cell has a probability $\frac{Q_{at}}{Q_{tot}}$ to be the one triggering the event. We can chose the triggering cell with the tower sampling method Fig. 6, we consider a tower where each floor is a cell and the height of the floor is Q_{at}. With this tower, we sample in this tower with a random number between 0 and Q_{tot} and then we must know in which floor this number fall. This method is straightforward and simple to understand but its implementation lead to a complexity $\mathcal{O}(N_{cell})$ in the worst case. It is then needed to find a better method.

FIG. 6. Each color on the tower represent a cell, the height is proportional to the rate Q_{at}. The point on the axis are random number used to know which cell trigger the event (Red dot $\rightarrow$ green cell). Note that when red dot value need to be compare only twice, the blue one need to be compare four time

The tower sampling being quite slow, we will use the Walker's method Fig. 7 to find the triggering cell. Instead of working with a tall tower of height Q_{tot}, we will make many small towers of height $Q_{mean} = Q_{tot}/N_{cells}$, and so each tower have the same weight and can be chose uniformly. Each tower will be composite of a maximum of two cells (note that a cell can be represented in more than one tower) at a rate specific for each tower. All the information (rate, first cell, second one) will be stored in a table that is calculated only once.

With the table calculated, we have to take to random numbers, an integer to know which tower will be use and a float number between 0 and Q_{mean} to know which cell of the used tower will be used in the end. This way of sampling do not scale with the number of particle nor the number of cells but will use a bit more memory.

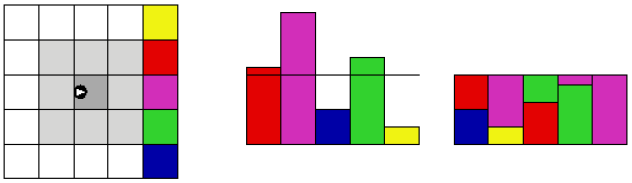

FIG. 7. **Left:** Cells in the far field must have finite rate; **Center:** Histogram of the five colored cell rate, the mean is indicated by the vertical bar; **Right:** The composites of height Q_{mean} composed of at most two cells

The far field treatment of Cell Veto Method Algorithm **??** can be implemented with a constant time complexity. We will work with a fixed chain length. It is also important to note that a pseudo-event (event only not trigger by the potential) will be use when the boundary of a cell is cross because we have to change the active cell.

	mean Separation	standard deviation
Metropolis	3.89147	0.00429
MTSMC	3.88850	0.00284
ECMC	3.89148	0.00992
CVMC	3.88933	0.01113

TABLE II. Mean separation between pair of particles obtain for 100 runs of 10^5 steps for each algorithm. We run the algorithms with 4 particles in a periodic box of size $L = 10$ with an inverse temperature $\beta = 1$. CVMC chain length used was 1.0 and the cell size 2.0

The table II show that CVMC have compatible result but with more variability.

IV. PERSPECTIVES AND CONTINUATION

We have describe and analyze the theoretical time complexity of four algorithms that can be used to sample Lennard Jones system. We still have to run the different algorithms with execution time computation for different values of N_{part}. In fact it has be done on buggy version of the implementation Fig. 8 and Fig. 9, even if only small change were made, it is important to redo the graphics with the actual implementation of the algorithm.

Since only small modifications were made, we can expect a similar result for the final implementation. From this early investigation, we see that the raw execution times give results compatible with the theoretical ones discuss previously, but raw time are not totally comparable as an algorithm can be slower per step but converge much faster.

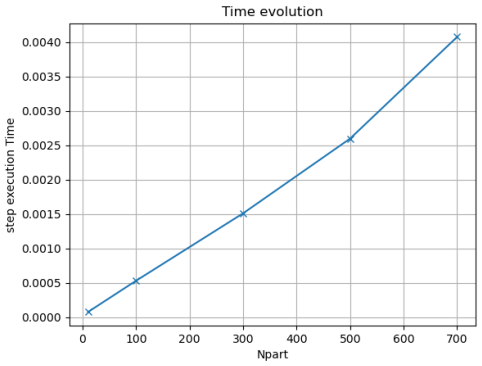

FIG. 8. Mean execution time per step against the number of particles in interaction. **Important:** This was obtain on an early version of the implementation and may differ a bit for the actual implementation

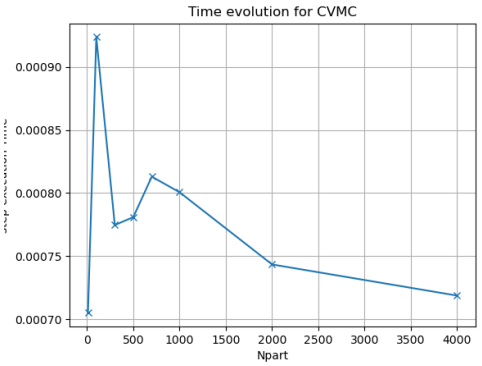

FIG. 9. Mean execution time per step against the number of particles in interaction. **Important:** This was obtain on an early version of the implementation and may differ a bit for the actual implementation

A way to compare and adjust the time execution must be explore. A possibility is to do as Hetènyi and Berne in [1], we can adjust by diffusion but event chain and cell veto has some ballistic component and so the adjustment will be bias in there favor. We must then find a way to compare the convergence of the algorithms.

This is the first continuation to the work already done. We can also discuss the optimal parameter for the cell veto, for a given particle density what is the best size for each cell. Here we worked with a grid of identical cells but in fact it is possible to work with cells of different size, for example we can increase the size of cells really far away. One can also optimize the length of the neighborhood since it is what give the number of particles that will be treated differently.

To conclude, the cell veto having a constant time complexity open the door to computation not done before due to unreasonable time need to run. One can think of system with to types of particles interacting with different intensities event different interactions can be implemented, this is possible in JeLLyFysh.

[1] B. Hetényi, K. Bernacki, and B. J. Berne, "Multiple "time step" Monte Carlo," *The Journal of Chemical Physics*, vol. 117, pp. 8203–8207, 10 2002.

[2] M. Rao, C. Pangali, and B. Berne, "On the force bias monte carlo simulation of water: methodology, optimization and comparison with molecular dynamics," *Molecular Physics*, vol. 37, no. 6, pp. 1773–1798, 1979.

[3] M. F. Faulkner, L. Qin, A. C. Maggs, and W. Krauth, "All-atom computations with irreversible Markov chains," *J. Chem. Phys.*, vol. 149, no. 6, p. 064113, 2018.

[4] P. Höllmer, L. Qin, M. F. Faulkner, A. C. Maggs, and W. Krauth, "JeLLyFysh-Version1.0 — a Python application for all-atom event-chain Monte Carlo," *Comput. Phys. Commun.*, vol. 253, p. 107168, 2020.

[5] S. C. Kapfer and W. Krauth, "Cell-veto Monte Carlo algorithm for long-range systems," *Phys. Rev. E*, vol. 94, p. 031302, 2016.