Application of irreversible Monte Carlo in long-range realistic systems

Liang Qin

PSL UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée au Laboratoire de Physique à l'École Normale Supérieure de Paris

Application of Irreversible Monte Carlo in Realistic Long-range Systems

Soutenue par
Liang QIN
Le 21/10/2020

Ecole doctorale n° 564
Physique en Île de France

Spécialité
Physique

Composition du jury :

Silke BIERMANN Professor at École Polytechnique	*Président du jury*
Ben LEIMKUHLER Professor at the University of Edinburgh	*Rapporteur*
Ralf EVERAERS Professor at École Normale Supérieure de Lyon	*Rapporteur*
Kurt KREMER Professor at the Max Plank Institute for Polymer Research	*Examinateur*
Werner KRAUTH Professor at École Normale Supérieure de Paris	*Directeur de thèse*

ENS ÉCOLE NORMALE SUPÉRIEURE | PSL RESEARCH UNIVERSITY PARIS

Département de Physique
École normale supérieure

UNIVERSITÉ PSL (PARIS SCIENCES & LETTRES)

ÉCOLE DOCTORALE PHYSIQUE EN ILE-DE-FRANCE — EDPIF 564

LABORATOIRE DE PHYSIQUE
DE L'ÉCOLE NORMALE SUPÉRIEURE

THÈSE DE DOCTORAT

Application of Irreversible Monte Carlo in Realistic Long-range Systems

présentée par Liang QIN

pour obtenir le titre de docteur de l'École normale supérieure

Soutenue le 21 octobre 2020 devant le jury composé de:

Silke BIERMANN	Président du jury
Ben LEIMKUHLER	Rapporteur
Ralf EVERAERS	Rapporteur
Kurt KREMER	Examinateur
Werner KRAUTH	Directeur de thèse

Acknowledgement

I feel fortunate to pursue the highest academic degree, lighting up my way to explore the world of infinite knowledge throughout life. I want to express my acknowledgment to the diverse and inclusive school, École normale supérieure, that offered me a scholarship covering my master's and doctoral studies. I cherish this opportunity helping me step into the world.

I can never imagine today's accomplishment without the direction of my advisor, Werner Krauth, who patiently guided me in every project, from scientific details to academic conduct. You brought me a friendly working environment and the most intriguing problems that I could not help thinking about all the time.

I also express my appreciation to my collaborator, Anthony Maggs, for your rich knowledge and willingness to share. Nothing can be more encouraging than that direct questions received your immediate replies.

The doctoral school makes me feel at home. I thank the former director, Jean-François Allemand, for caring for us students not only on administrative procedures but on our growth. I thank Frédéric Chevy as the new director and former head of master education. My years in Paris would be harder without your supports. Also, I want to thank every supporting member of the department.

I appreciate Alberto Rosso for his course on computational physics that opened to me the door of Monte Carlo, Synge Todo and Hidemaro Suwa for expanding my research horizon, and Koji Hukushima for the day of exciting discussion over the extended ensemble Monte Carlo. Also, I would like to thank Marc Mézard, who I have no direct contact with, for the most impressive course as if the spin glass is the most valuable treasure in the world, and for directing the whole ENS.

I want to thank Pierre Pansu as my tuteur and Nicolas Regnault as my parrain for carefully listening to my progress twice at the end of academic years and giving me advice as professors outside my project.

The completion of my thesis is also the fruit of my colleagues. I want to thank the former members in the group, Etienne Bernard, Sebastian Kapfer, Manon Michel, Tommaso Comparin, and Yoshihiko Nishikawa, for orienting me at the difficult beginning and for your pioneering works prior to my thesis. I want to thank Ze Lei and Juliane Klamser for telling me everything I need to lead a doctoral life. A particular appreciation is given to two years of collaboration with Philipp Höllmer, for your expertise and rigorous working manner. I also thank Michael Faulkner for coworking in the first paper, and thank Botao Li for your continued and invaluable discussions.

I also appreciate all the young friends who work around me. Besides the infinite help, your daily greetings are the warmest things in a day.

Finally, I express my gratitude to the thesis jury for the careful examination and recognition of my works. They are Silke Biermann, Ben Leimkuhler, Ralf Everaers, and Kurt Kremer. I thank Werner Krauth, Philipp Höllmer and Botao Li for reviewing this manuscript and for the helpful feedback during the preparation of my defense.

General Introduction

People have developed classical molecular models to understand the behavior of systems ranging from hard disks to proteins. Equilibrium statistical physics connects atomic interactions and macroscopic physical quantities such as pressure, specific heat, and, more profoundly, order parameters that decide criticality. Although chemical reactions and biochemical conformational changes are often out-of-equilibrium, their statistics like reaction paths and rates can also be understood by simulating atomic models. The Boltzmann distribution is fundamental in statistical physics but requires high-dimensional integral beyond any analytical solution. Molecular mechanics and Monte Carlo stand out as the two most successful numerical methods, laying the base of modern molecular simulations.

Much effort was devoted in pursuit of increasingly large-scale simulations in various fields of study. Phase transitions of special physical interests occur in the thermodynamic limit, and studies of finite-size effect also desire simulations of larger space and time for the sake of accuracy. In biochemical settings, numerous interesting reactions occur on a millisecond time scale, far beyond the nanosecond time scale reached by personal computers and state-of-the-art molecular dynamics.

Among all problems, long-range interactions are notoriously onerous from a computational perspective, in spite of their pervasive role in nature. Electrostatic interactions are indispensable in reproducing realistic properties in a system with a polarized solvent. Practical large-scale algorithms must avoid pairwise iteration while calculating the Coulomb force. And so far, the most commonly used method heavily relies on the spatial discretization, which brings systematic bias.

Event-chain Monte Carlo (ECMC) is a novel irreversible Monte Carlo method applicable in any continuous system, and it has demonstrated excellent efficiency from simple hard disks to spin systems. Taking advantage of its factorized nature, its first long-range variant was proposed with $\mathrm{O}(N^{4/3})$-per-sweep time complexity for three-dimensional Coulomb particles.

During the past three years, we focused on long-range algorithms under the framework ECMC, and applied them to realistic water systems. We established its theoretical foundation and also initiated an open-source application for all researchers. This thesis is a summary of my work under the supervision of Werner Krauth and in collaboration with Anthony Maggs, Philipp Höllmer, and Michael Faulkner.

The thesis will expand as follows. In chapter 1, we will review molecular simulation methods as a general background. Methods other than ECMC—molecular dynamics, reversible Monte Carlo, and existing trials over irreversibility—will be briefly

introduced.

In chapter 2 we introduce ECMC in detail. Apart from other ECMC formulations, we describe its theoretical foundation with general many-body factors, through a novel graphic lifting scheme, and establish a connection between factorization and redundant flow. We then provide the event-driven implementation and its specific thinning process, which brings ECMC with faster speed and applicability to all energy forms. In chapter 3, we talk about existing work and our progress of long-range ECMC adaptation. We find that the tin-foil electrostatics is the most suitable because its pairwise form facilitates factorization. Apart from the traditional Ewald summation, various approaches exist for ECMC's Coulomb calculation, and they all converge to the tin-foil electrostatics. Furthermore, factorization plays a significant role in reducing the event rate for dipoles, thus lowers the overall complexity. Three lifting schemes corresponding to dipole factors are proposed and result in different dynamics. Combining all these discoveries plus the cell-veto method leads to a three-dimensional Coulomb ECMC method of $\mathrm{O}(N \log N)$-per-sweep complexity. This work [1] has been published on *the Journal of Chemical Physics*.

We also attach emphasis on application development and release JELLYFYSH-V1.0 as a universal python application for all-atom simulations. Summarizing our work [2] published on *Computer Physics Communication*, chapter 4 describes its underlying architecture, in which we reformulate ECMC in favor of event processing, with all potential operations as an event stream. The mediator design pattern featuring a central hub and peripherical functioning modules is believed to accommodate future extensions best. Then in chapter 5, we continue our way towards the real simulation by providing event handlers for diverse environments with concrete instances for users to follow.

We put JELLYFYSH into practice in chapter 6 with large systems of water molecules. Besides the programming adaptation to the real Coulomb system, we also make great effort to optimize the long-range modules, in particular the cell-veto method. Detailed profiling of Coulomb events provides insights over the rate, distribution, and types of Coulomb events. Furthermore, we test JELLYFYSH with a water system up to 216 molecules. The polarization signal not only reflects the speed of relaxation but shows unusual excitation for few-molecule states. For the first time, we incorporate molecular translation into single-active-particle ECMC and measure its effect in the relaxation of overall polarization.

Given the double time scales in the equilibration of water polarization, we proceed with an additional study over sequential choices of direction in Monte Carlo in chapter 7. Moving directions that change progressively exhibit exotic dynamics of the dipole angle, and we find that proper sequential schemes outperform the conventional way that always resamples along orthogonal axes. It also reveals that the set of resampling directions has a great impact on the performance, whose comprehensive mechanism is left as our future study. We submitted the results [3] to journals and now it is a preprint.

In the appendices, we attach the three aforementioned papers published or submitted during my doctoral study.

Contents

Chapter 1

Simulation method review

1.1 The need for large-scale simulation

The impression when talking about simulation is that we are going to integrate an equation of motion such as Newton's law $m_i \partial^2 \mathbf{r}_i / \partial t^2 = -\nabla_i U$, or equations for continuum like Maxwell equations or Navier-Stokes equations. We can know everything if the evolution of a system's elements is evident.

Problems arise concerning non-continuous space such as spin systems in which each variable takes discrete values. Such physical configurations serve as abstract models of reality but can reflect the intrinsic properties of complex systems. No deterministic equation of motion is present for discrete systems. On the other hand, due to the Lyapunov instability, the behavior of the simulated system is very sensitive to the initial condition. In other words, even if a sufficiently sophisticated program can predict the exact state $\mathbf{r}_i(t)$ for some time t, the computer-representation inaccuracy of the initial state $\mathbf{r}_i(0)$ results in an exponential difference between $\mathbf{r}_i(t)$ and the real position, let alone the cumulative error brought by numerical integrators, and deviation from rough model and environment noises. In fact, progress has deviated from the exact solution of $\mathbf{r}_i(t)$ since the invention of simulation algorithms like molecular dynamics and Monte Carlo.

Usually, we care more about statistics under thermodynamic conditions than orbits, and equilibrium statistical mechanics is capable of capturing the nature of static states. It can calculate the freezing point of water, predict the stable phase of an alloy, and obtain the average time for two molecules to react as well as the reaction path. The ultimate physical equation we need to solve throughout our work is Boltzmann's law describing the probability of a state in equilibrium

$$\pi(c) = \frac{1}{Z} e^{-\beta U(c)}, \tag{1.1.1}$$

where β, U and Z are the inverse temperature, energy and normalizing factor respectively.

Physical quantities can be acquired by sampling the configuration space, and flexible approaches exist to this end. Molecular dynamics simulates a canonical ensemble with the help of a thermostat by integrating Newton's equation of motion (or more

generally, Hamilton's equations), which can reach the target Boltzmann distribution in small step limit. Monte Carlo methods perform moves that are not connected by the equation of motion and reach the desired distribution by respecting the global balance condition. It is a convenience of Monte Carlo methods that only the relative probability between two states is required. Now Monte Carlo becomes a universal technology to reveal properties of a variety of systems, from simple particles, biochemical molecules, to engineering and finance.

1.1.1 Examples

Here we give two examples of how simulations help us understand nature.

Hard disks

Even though modern physics has developed to be so sophisticated and high-energy that daily experiments can never justify, it is surprising to see that the hard-disk melting problem has been puzzling physicists for half a century.

Phases in condensed matter are often characterized by correlation length, typically classified into long-range, quasi-long-range, and short-range orders. Hard disk serves as the two-dimensional prototype to explain the classical phase transition on a plane. Theorists have early established through an elastic model that long-range order cannot appear in systems of less than three dimensions. Thus, two-dimensional systems become the focus of physical interests, in which solid states feature quasi-long order and liquid states feature short-range order. Kosterlitz and Thouless [4] proposed in 1973 their famous theory that topological defects—that are unbinding pairs of vortex and anti-vortex in XY model—are suddenly favorable when the temperature passes a threshold. First-order scenarios can also exist, such as in the narrow-well XY model that resembles $q > 4$ Potts model. For hard-disks, things become more complicated in the presence of positional and orientational orders. The well-known Kosterlitz-Thouless-Halperin-Nelson-Young theory [5, 6, 7] proposed in the late 1970s predicts that the unbinding of dislocation and disclination will consecutively break positional and orientational orders, making hard disks transition from the solid to the hexatic state and finally to the liquid through second-order transition, in competition with other first-order theories.

A large number of realistic and numerical experiments were conducted just for the hard-disk phase transition. Since the first simulation [8], people simulated via molecular dynamics and Monte Carlo with increasing system size and precision. Not until later works [9, 10, 11] cleaned up the puzzle using large-scale ECMC simulation, people had been debating over the nature of hard-disk phase transition for thirty years.

In 2011, the ECMC algorithm was applied [9] to equilibrate up to a million disks around criticality, which was confirmed by later work [11] which also compared several most advanced methods at that year and drew the long-anticipated equation of state of hard disks. They concluded that: with decreasing density, hard disks undergo a second-order transition to a hexatic phase, then to a liquid phase with the first-order transition, which contradicts both the KTHNY theory and other first-order

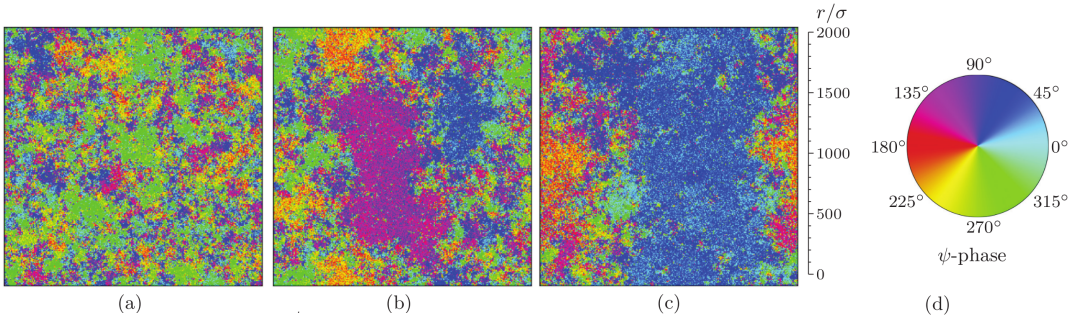


Figure 1.1: Orientational order parameter field of configurations obtained with the massively parallel Monte Carlo algorithm for $N = 1024^2$ disks. With increasing density, (a) pure liquid (low density), (b) a bubble of hexatic phase (middle density), and (c) a stripe regime of hexatic phase (high density) are visible. (d) A scale bar illustrates the size of the fluctuations. (From Ref [11].)

scenarios. Shown in Fig. 1.1 is the orientational order of three configurations around critical density produced with massively parallel Monte Carlo. With the information of equilibrated states, they were able to differentiate phases and measure the transition orders.

SARS-CoV-2

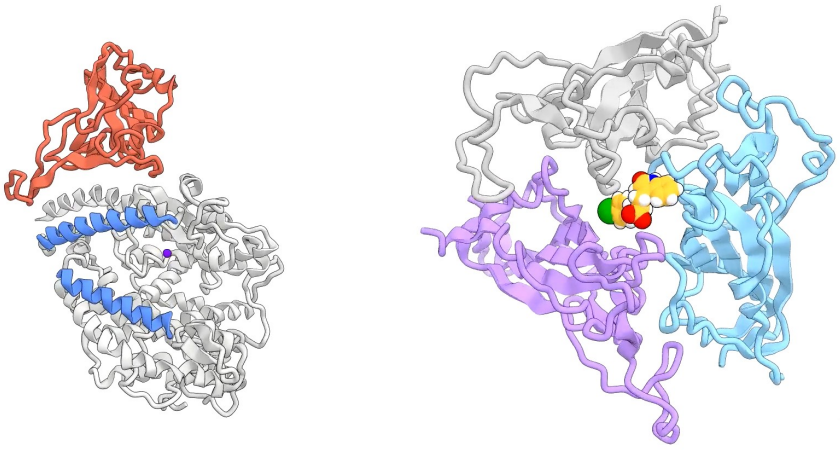
Large-scale biochemical simulations are instrumental in revealing the dynamics and conformation of proteins, in which D. E. Shaw is a pioneer who founded his research center called D. E. Shaw Research, and devised a special-purpose supercomputer to perform all-atom simulations.

Less than one month after the outbreak in the U.S. of Covid-19, the disease caused by SARS-CoV-2, D.E.Shaw Research released their simulation results [12] relevant to the virus' proteins. Shown in Fig. 1.2 are two critical biochemical configurations of the trimetric spike protein of the virus, human ACE2 (receptor of the virus protein), and tested drug molecules. The millisecond large-scale atomic simulations are rich in dynamical information of the virus. They suggest that specific molecules from the human body are susceptible to the virus' characteristic protein, making them easy targets of the deadly virus. And medical solutions at the molecular level come out at the same time as simulations show that part of drug molecules prohibits the required binding of the virus and human protein.

They reported that among 5152 molecules in the drug library, 78 can be bound to ACE2, and 50 can be bound to the spike protein as candidates for the next pharmaceutic process. The valuable information provided by D. E. Shaw Research is believed to be helpful in drug development.

1.1.2 State of the art

Despite the progress in hard disks and the success in simulating SARS-CoV-2 proteins, months-long intensive computation played a crucial role, and larger-scale sim-



(a) (b)

Figure 1.2: Two simulated states of SARS-CoV-2 protein dynamics. (a) The virus spike protein (orange) is binding human ACE2 (white and blue). (b) A drug molecule (center, multiple colors) is binding the virus' trimetric spike protein. This drug is said to have been approved or investigated by the Food and Drug Agency of the U.S. (From Ref [12].)

ulations are still heavily in demand. In order to prepare 64 independent 1024^2-disk states, Bernard and Krauth spent nine months running ECMC on a single CPU core [9, 10]. The only algorithm that can outperform ECMC and validate their finding was massively parallel Monte Carlo in 2013, while such high performance was only achieved with the help of 1536 cores of an NVIDIA GeForce GTX 680. Still, such intensive simulation appeared slow in relaxing the system's orientational order, and a larger system of hundreds of millions of disks is still desirable for benchmark and confirmation.

The data by D. E. Shaw Research was produced by a special supercomputer called Anton [13] that was designed exclusively for protein molecular dynamics simulation and outperformed all the general supercomputers at that time. Anton, supported with 512 nodes, each containing highly optimized functioning components, can advance a system of 20,000 atoms by ten microseconds in a day, compared with less than 1ns by GROMACS on a personal computer. Furthermore, it can potentially handle larger systems in a scalable manner (almost constant complexity per sweep) thanks to its maximally parallel architecture.

There is a variety of scientific software for molecular simulations, and it is all designed for special contexts. From quantum-effect particles to macro-scale proteins, from serial ones to GPU-parallelization, using Monte Carlo or molecular dynamics, each software works in its specific field to balance versatility and performance. Many widespread tools are open-source, so they attract a worldwide community to maintain and contribute to them. For example, LAMMPS [14] stands for Large-scale Atomic/Molecular Massively Parallel Simulator, and is now maintained and distributed by researchers in Sandia National Labs and Temple University. In this work, we aim to develop ECMC for large-scale all-atom simulations and regard LAMMPS as the fu-

ture model.

As for methodology, molecular dynamics tells important time-dependent information such as the accessibility of a given molecular surface, the transition time between two protein states, the appearance and disappearance of a particular chemical channel or cavity. And due to the simultaneous motion of all elements, fast algorithms exist to compute the long-range force on each particle, and parallel algorithms by domain decomposition apply as efficiently as short-range systems. That is why molecular dynamics becomes popular in most biochemical simulation software. On the other hand, just in terms of operation, Monte Carlo abandons the system's natural motion, and it is allowed to take imaginary moves such as cluster flipping in spin models, chain regeneration for polymers, and atomic tunneling to realize substantial changes (see section 1.3.3.4 for more examples). Actually, Monte Carlo restricted only by Eq. 1.1.1 bears much more flexibility in diverse contexts. In the problem of ensemble consideration, parallelization, energy minimization, and many others, Monte Carlo has successfully exploited the freedom of a given system.

In the rest of this chapter, we will introduce molecular dynamics in section 1.2, talking about how the state-of-the-art molecular dynamics works and its major practical concerns. Then in sections 1.3 and 1.4 we will introduce Monte Carlo methods from the most naive reversible one to our irreversible ECMC. In section 1.5, we will give some other necessary knowledge, including the molecular models and measurables in preparation for our work.

1.2 Molecular dynamics

According to the ergodic theorem, the time average of a natural orbit equals the ensemble average over the configuration space, so integrating the equation of motion and taking the average along the path converge to statistics in equilibrium. An ideal integrator perfectly reproduces the solution of Newton's equation of motion, or more systematically, of Hamilton's equations

$$\begin{aligned} \frac{\partial q_i}{\partial t} &= \frac{\partial H}{\partial p_i}, \\ \frac{\partial p_i}{\partial t} &= -\frac{\partial H}{\partial q_i}, \end{aligned} \tag{1.2.1}$$

where q_i's are generalized coordinates, and p_i's are generalized momentums.

A molecular dynamics algorithm consists of a numerical integrator and a thermostat used to correct the deviation caused by the integrator. Nevertheless, statistical bias is inevitable during the numerical integration. A second source of error comes from the long-range force calculation since modern Coulomb computation heavily relies on the spatial discretization. What is more annoying is that a source of error implies a set of parameters that a user must handle, and in the case of molecular dynamics, they are integrators, mesh sizes, interpolation methods.... By comparison, Monte Carlo methods do not perform temporal discretization, and ECMC can even avoid long-range force error through its distinct way of the Coulomb force calculation.

1.2.1 Integrator

For time integral, people have developed many algorithms (see [15, 16] for reference) that can approximate Newton's equation of motion. Numerical integrators in general all bear some errors without exception, while the error remains at a relatively low level if the integrator respects the structural properties of Hamilton's equations—the time reversibility, volume preservation, conservation of energy and momentum, and more importantly, simplecticity. In practice, energy drift is observed in many kinds of integrators. Zhong and Marsden [17] in 1988 proved that if the energy and the momentum map include all the integrals of motion, then one cannot create integrators that are symplectic, energy preserving, and momentum preserving. As symplecticity implies phase space volume preservation, one cannot, in general, have preservations of phase space volume, energy, and momentum all in one method. In molecular dynamics practice, people use two classes of mechanical integrators: symplectic-momentum and energy-momentum integrators.

1.2.2 Long-range algorithms

Biochemistry requires models with the Coulomb interaction in which truncations in its tail lead to serious artifacts [18]. The solvent consisting of water and possible ions can impose a polarizing effect over charged organic molecules. Challenges arise when long-range interaction is considered, and even modern algorithms heavily depend on mesh methods on which charge spreading and interpolation are commonly based. Imprecision does not vanish, but we control it at a tolerable level. When using LAMMPS with tunable precision levels, one can observe the slowdown with higher precision.

We need fast computation of the Coulomb interaction upon each atom without iterating all atomic pairs. Among all trials, the so-called particle-mesh Ewald [19, 20], particle-particle-particle-mesh [21] and an Anton adaptation called k-space Gaussian split Ewald [22] stand out as today's widespread methods.

All predominant long-range methods are developed upon Ewald's summation that splits the potential into a short-range core and a long-range smooth tail. The short-range part is calculated via visiting each particle's neighbors, and the long-range part is dealt with by sophisticated mesh-based methods. Usually, charges are spread into nearby grid points, then it performs convolution by fast Fourier transformation to gain potential values on grids, and does interpolation to acquire forces on individual charges. The second step for the long-range term takes $\mathrm{O}(N \log N)$ time and often predominates the total time consumption in molecular simulations (see Table 1 of [13]).

1.2.3 Event-driven molecular dynamics

It is worth mentioning event-driven molecular dynamics here for two reasons: it does improve molecular dynamics performance by getting rid of small timesteps, and our ECMC resembles event-driven molecular dynamics in terms of event processing.

Event-driven molecular dynamics deals with short-range interactions exclusively. In order to get rid of small timesteps, it sacrifices much accuracy of the potential by replacing it with a step-wise potential [23]. Velocities change only when two particles touch the potential step. Such improvement makes molecular dynamics event-driven that the program needs only to predict the next "step-touching" event between which all velocities remain constant, and calculates outgoing velocities. It is an accurate algorithm to simulate inaccurate models.

Event-driven molecular dynamics is particularly suitable for hard disks, and one has to update the time stamps only for colliding disks even though all disks are moving. All upcoming collisions are stored in a central heap to select the soonest event, which is very similar to event processing in ECMC. Consequently, the central heap becomes the algorithmic bottleneck and usually takes $\mathrm{O}(\log N)$ to have an event, although there are efforts [24, 25] to ease the burden of the heap.

The unpredictable correlation between any parts of the simulated system renders event-driven molecular dynamics difficult to parallelize. An event happening in one corner may trigger immediate events in the distance and destroy all speculations by another thread. People have devised many algorithms [26, 27, 28] for parallel hard-disk simulators and successful algorithms rely on good avoidance of conflicts and fast reversion of already-done moves, and more effort will have to be made for further interaction range.

1.3 Traditional Monte Carlo

The term Monte Carlo is used in most cases in company with Markov chain, which indicates a stochastic process that the next state relies only on the current state. The program walks along a chain of target-space states $s_1 \to s_2 \to s_3 \cdots$, and the probability distribution of s_{i+1} is solely determined by s_i. This is the behavior of all simulations[1], so we will use the name Monte Carlo but omit Markov chain[2] except in section 1.3.1 when talking about its mathematics.

A Monte Carlo algorithm samples a prior probability distribution $\pi(s)$ of a state space that one only has to know relative probabilities between any two states, i.e., $\pi(s_1)/\pi(s_2)$. It includes a strategy to jump to the next state, given the current state. The probability in an n-state space can be described with a vector $\boldsymbol{\pi} = (\pi_1, \pi_2, ..., \pi_n)$, and let T be the transition matrix whose element T_{ij} indicates the probability to go from s_i to s_j. It satisfies

$$\sum_j T_{ij} = 1, \;\; T_{ij} \geq 0, \;\; \forall 1 \leq i, j \leq n. \tag{1.3.1}$$

Matrix satisfying Eq. 1.3.1 is also known as stochastic matrix. The distribution of the next state can be conveniently written as $\boldsymbol{\pi}T$ and it becomes $\boldsymbol{\pi}T^n$ after n moves.

[1] Direct sampling can be an example of non-Markovian Monte Carlo, but it is not a simulation method.

[2] In *Understanding Molecular Simulation* by Frenkel and Smit [15], they use Monte Carlo throughout as well, though the Markovian property is obvious.

1.3.1 Properties of transition matrix

1.3.1.1 Balance condition

The goal of Monte Carlo algorithms is to acquire statistics for the target distribution $\boldsymbol{\pi}$, so we hope that $\boldsymbol{\pi}$ is the stationary distribution of the transition matrix T. Expanding $\boldsymbol{\pi}T = \boldsymbol{\pi}$ we get

$$\sum_j \pi(s_j)T_{ji} = \pi(s_i), \tag{1.3.2}$$

which is called global balance condition. This is a necessary condition for target distribution, and it means the conservation of probability flow

$$\sum_j f_{ji} = \sum_k f_{ik} = \pi(s_i), \tag{1.3.3}$$

with $f_{ji} = \pi(s_j)T_{ji}$ the probability flow from j to i.

There are degrees of freedom to construct a transition matrix T respecting the global balance condition. However, universal constructs applicable to all spaces and energy forms fall under a smaller category. It obeys detailed balance condition that enforces canceling flow between any two states

$$\pi(s_i)T_{ij} = \pi(s_j)T_{ji}, \quad \forall i, j. \tag{1.3.4}$$

It is easy to verify that detailed balance implies global balance by summing over j of Eq. 1.3.4. In section 1.3.3 we will present several detailed-balance Monte Carlo methods, but decades of development tell us that a Monte Carlo method as general as the Metropolis-Hasting algorithm and without detailed balance does not exist.

Also, we have two properties for the transition matrix which will be used later

Proposition.

- *If $T^{(1)}$ and $T^{(2)}$ both obey the global balance condition, their product $T^{(1)}T^{(2)}$ satisfies global balance too.*
- *If $T^{(1)}$ and $T^{(2)}$ commute and obey detailed balance, $T^{(1)}T^{(2)}$ satisfies the detailed balance condition.*

1.3.1.2 Convergence to stationary distribution

A stochastic matrix with the balance condition is not sufficient for the Markov chain to sample the entire space as desired. Intuitively, one must be able to go to any state with an arbitrary starting point, so the space must be connected by the transition matrix. Another less common situation is that if the chain exhibits periodicity, the chain itself does not converge, but its long-term average does. The two properties are defined as

- Irreducibility. For any i and j, there exists a positive integer r so that $T^r_{ij} > 0$.
- Aperiodicity. Let GCD be the greatest common divisor of a set of integers, then $\mathcal{T} = \mathrm{GCD}\{s : T^s_{ii} > 0\}$ can prove independent of i, and T is aperiodic if $\mathcal{T} = 1$.

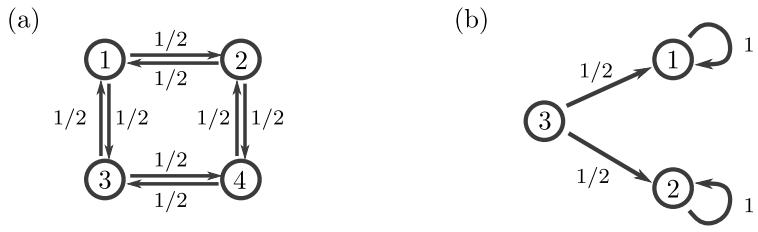


Figure 1.3: Examples of Markov chain that motivate aperiodicity and the ergodic theorem. (a) A 4-state space with homogeneous probability, in which the Markov chain switches between $\{1,4\}$ and $\{2,3\}$. This chain has period 2. (b) A state violating irreducibility. If many chains start with state 3, they can still reach the distribution $\pi_1 = \pi_2 = 1/2$, $\pi_3 = 0$. But for a single chain, it either stays at state 1 or state 2.

As for the criteria of convergence, total variation distance [29] is the most common one defined as

$$\|\boldsymbol{\pi}' - \boldsymbol{\pi}''\|_{\mathrm{TVD}} = \frac{1}{2}\sum_i |\pi_i' - \pi_i''|. \tag{1.3.5}$$

In the following, we call $\boldsymbol{\pi} = (\pi_1, \pi_2, ..., \pi_n)$ with $0 \le \pi_i \le 1$ and $\sum_i \pi_i = 1$ a valid initial state.

Fig. 1.3 (a) shows an example with period $\mathcal{T} = 2$ thus the chain jumps between the sets $\{1,4\}$ and $\{2,3\}$. In the case $\mathcal{T} > 1$, $\boldsymbol{\pi}^{(0)}T^p$ does not converge as $k \to \infty$, while the average $\frac{1}{p}\sum_{k=0}^{p} \boldsymbol{\pi}^{(0)}T^k$ converges as $p \to \infty$.

Theorem 1. *Given an $n \times n$ irreducible stochastic matrix T, and a valid initial state $\boldsymbol{\pi}^{(0)}$, then the average after p iterations*

$$\boldsymbol{\pi}^p = \frac{1}{p}\sum_{k=0}^{p-1} \boldsymbol{\pi}^{(0)}T^k, \tag{1.3.6}$$

converges to a vector $\boldsymbol{\pi}^\infty$ as $p \to \infty$. This $\boldsymbol{\pi}^\infty$ satisfies $\boldsymbol{\pi}^\infty T = \boldsymbol{\pi}^\infty$, and is unique regardless of the initial state.

Its proof can follow that of Theorem. 2, which is given by Levin et al. [29]. In existence of aperiodicity the convergence holds without taking the average.

Theorem 2. *Given an $n \times n$ irreducible aperiodic stochastic matrix T, and a valid initial state $\boldsymbol{\pi}^{(0)}$, then $\boldsymbol{\pi}^{(0)}T^k \to \boldsymbol{\pi}^\infty$ as $k \to \infty$. This $\boldsymbol{\pi}^\infty$ is the unique stationary distribution. Furthermore, there exists $\alpha \in (0,1)$ and $C > 0$ that*

$$\|\boldsymbol{\pi}^{(0)}T^k - \boldsymbol{\pi}^\infty\|_{\mathrm{TVD}} < C\alpha^k. \tag{1.3.7}$$

A proof can be found in [29], theorem 4.9. This theorem rules out chains jumping between non-intersecting sets, as shown in Fig. 1.3 (a) .

In fact, irreducibility and aperiodicity are satisfied in most physics simulations. The global balance condition is equivalent to that the target distribution is one eigenvector with eigenvalue 1. Theorem. 2 tells us that the Markov chain will surely reach the desired distribution. Therefore, the global balance condition is usually the key requirement to construct Monte Carlo methods.

1.3.1.3 Ergodic theorem

The ergodic theorem has a similar role as the strong law of large numbers. It states that the stationary distribution is reached, not only statistically when we start from numerous initial states, but also in a single chain when the time tends to infinity. Fig. 1.3 (b) provides an example in which $\pi_1 = \pi_2 = 1/2$ is reached with infinite trials starting from state 3, but no single chain can visit both state 1 and state 2. Irreducibility is violated in this example.

With irreducibility, the target distribution can be ensured when taking statistics of a single chain.

Theorem 3. *A Markov chain $(X_1, X_2, ...)$ is drawn with an arbitrary starting state and transition rule T that is an irreducible stochastic matrix. The mean vectors $\boldsymbol{\mu}^{(p)} = (\mu_1^{(p)}, \mu_2^{(p)}, ..., \mu_n^{(p)})$ are defined as*

$$\mu_i^{(p)} = \frac{1}{p} \left(\text{number of } i\text{'s in } (X_1, X_2, ..., X_p) \right). \tag{1.3.8}$$

Then the probability

$$P\left(\lim_{p\to\infty} \boldsymbol{\mu}^{(p)} = \boldsymbol{\pi}^{\infty} \right) = 1, \tag{1.3.9}$$

with $\boldsymbol{\pi}^{\infty}$ the unique stationary distribution.

If one starts with state 3 of Fig. 1.3 (b) , with $P = 1/2$ it gets the chain $(3, 1, 1, 1, ...)$ and otherwise gets the chain $(3, 2, 2, 2, ...)$, which is excluded by this theorem due to the violation of irreducibility. See appendix C of [29] for a proof.

We also have the conjecture without proof that aperiodicity can lead to a stronger conclusion

Conjecture 1. *A Markov chain $(X_1, X_2, ...)$ is drawn with a starting state X_1 and the transition rule T that is an irreducible aperiodic stochastic matrix. We pick an infinite sequence of positive integers $a_1 < a_2 < a_3 < \cdots$ The mean vectors $\boldsymbol{\nu}^{(p)} = (\nu_1^{(p)}, \nu_2^{(p)}, ..., \nu_n^{(p)})$ are defined as*

$$\nu_i^{(p)} = \frac{1}{p} \left(\text{number of } i\text{'s in } (X_{a_1}, X_{a_2}, ..., X_{a_p}) \right). \tag{1.3.10}$$

Then the probability

$$P\left(\lim_{p\to\infty} \boldsymbol{\nu}^{(p)} = \boldsymbol{\pi}^{\infty} \right) = 1, \tag{1.3.11}$$

with $\boldsymbol{\pi}^{\infty}$ the unique stationary distribution.

If this proposition holds, mixing is guaranteed for any pre-selected elements of a single Markov chain.

1.3.1.4 Eigenvalue analyses

Eigenvalues and eigenvectors are informative in understanding the structure of an operation encoded as a matrix. For a transition matrix representing detailed-balance Markov chain, it has n real eigenvalues.

Theorem 4. *Let T be an $n \times n$ stochastic matrix that is irreducible and aperiodic, and its stationary vector $\boldsymbol{\pi} = (\pi_1, ..., \pi_n)$ satisfies $\pi_i T_{ij} = \pi_j T_{ji}\ \forall i, j$. Then it is diagonalizable and has eigenvalues $1 = \lambda_1 > \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n > -1$.*

Intuitively, starting with any single state, we project the initial state onto the eigenvectors of T, the component corresponding to the stationary distribution will stay invariant, while other components will be multiplied with λ_i after each move. Hence the distance to the stationary distribution decays exponentially, at the rate of

$$\lambda^\star = \max\{|\lambda_i| : \lambda_i \neq 1, i = 1, ..., n\}, \tag{1.3.12}$$

if the component corresponding to $\lambda^\star$ does not vanish. More specifically, we have the following theorem

Theorem 5. *Let T be an irreducible aperiodic transition matrix for a Markov chain with detailed balance condition, and $d(p)$ be the worst case total variation distance at time p*

$$d(p) = \max\left\{\|\boldsymbol{\pi}^{(0)} T^p - \boldsymbol{\pi}^\infty\|_{\mathrm{TVD}} : \boldsymbol{\pi}^{(0)} \text{ is a valid initial state}\right\}, \tag{1.3.13}$$

where $\boldsymbol{\pi}^\infty$ is the stationary distribution of T. Then

$$\lim_{p\to\infty} d(p)^{1/p} = \lambda^\star. \tag{1.3.14}$$

It means that the second largest eigenvalue really indicates the speed of convergence. We recommand the chapter 12 of [29] as systematic proofs of the two theorems above.

1.3.2 Tricks by modifying target distribution

One of the excitement while thinking about the Monte Carlo method is that we can never expect what will happen if we combine two distinct ideas. Here, we are going to cover several important Monte Carlo methods throughout its history. What is intriguing is that they are not mutually exclusive but extend the power of Monte Carlo in different dimensions. For example, skew detailed balance described in section 1.4.2.2 has an obvious application in the one-dimensional walk in a non-uniform space, so can apply to temperature jumps in extended Monte Carlo. Stochastic potential switching is able to derive the factorization for the Metropolis-Hastings algorithm and ECMC, and can also lead to the cluster algorithm for soft-potential particles.

Many Monte Carlo methods improve by proposing unrealistic moves specific to each physical system, and others construct probability flows beyond the detailed balance condition. Nevertheless, some ideas appear even by reconsidering the target distribution. Here we introduce three methods that reach the desired probability via a different energy function.

Importance sampling

Sometimes the target distribution $\boldsymbol{\pi}$ fails to cover the phenomenon of interests, e.g., singularities with little weight but significant contribution to physics, rare events that

occur infrequently but are crucial to the studied system. If we have some prior knowledge under what circumstance the crucial event happens more, then we can bias the sampler by giving it a new distribution $\boldsymbol{\pi}^\star$. $\boldsymbol{\pi}^\star$ concentrates more on the interesting points, and the Monte Carlo algorithm with importance sampling takes $\boldsymbol{\pi}^\star$ as the target distribution and performs any Monte Carlo method. However, the drawn samples are weight with $\pi(s_i)/\pi^\star(s_i)$ to make the estimator unbiased

$$\langle O\rangle = \sum_{i=1}^{n} O_i\pi_i = \sum_{i=1}^{n}\left[\frac{O_i\pi_i}{\pi_i^\star}\right]\pi_i^\star \approx \frac{1}{N_s}\sum_{k=1}^{N_s} O(s_k)\frac{\pi(s_k)}{\pi^\star(s_k)}, \tag{1.3.15}$$

where on the rightmost side, $\{s_k\}$ are N_s states generated by a standard Monte Carlo algorithm with biased $\pi^\star$.

A straightforward example of importance sampling is to integrate $\int_0^1 x^{-p}\mathrm{d}x$ (see [30], section 1.4.2), where naive Monte Carlo by drawing randomly $x_i \in (0,1]$ is slow to capture the property around $x = 0$. Applications of importance sampling include rare event simulations [31]. However, it has two drawbacks: First, $\boldsymbol{\pi}^\star$ is not always obvious [3]; Second, $\boldsymbol{\pi}/\boldsymbol{\pi}^\star$ is hard to calculate, especially when the partition function is unknown.

Stochastic potential switching

Stochastic potential switching was first formulated by Mak [32]. Before, the initial idea appeared in Swendsen and Wang's cluster algorithm [33] for the Ising model that probabilistically turns off part of interactions. Here we present it by splitting the desired probability distribution.

One can split the target distribution into two positive parts $\boldsymbol{\pi} = \boldsymbol{\pi}' + \boldsymbol{\pi}''$. For each term we design a tailored Monte Carlo algorithm, i.e., using one Monte Carlo algorithm to explore $\boldsymbol{\pi}'$ and the other to explore $\boldsymbol{\pi}''$. This is feasible because Monte Carlo only needs to know $\pi(s_i)/\pi(s_j)$ instead of absolute values. Before each Monte Carlo step or several steps, the program must determine which term of $\boldsymbol{\pi}$ it is going to explore by the corresponding ratio, namely $\pi'(s_i)/\pi(s_i)$ and $\pi''(s_i)/\pi(s_i)$.

Fig. 1.4 illustrates the stochastic potential switching process. Changing $\boldsymbol{\pi}$ to $\boldsymbol{\pi}'$ or $\boldsymbol{\pi}''$ amounts to using new potentials instead, and they satisfy $\exp(-\beta U) = \exp(-\beta U') + \exp(-\beta U'')$ and hence the name. Splitting $\boldsymbol{\pi}$ into many does not always speed the mixing of states. However, if we have a fast Monte Carlo algorithm for $\boldsymbol{\pi}'$ then the overall sampling is fast[4]. Mak found another example [32] besides the cluster Ising model. In the Lennard-Jones liquid, he turned off by certain probability the attractive tail of the Lennard-Jones interaction (each interaction pair is switched separately, so strictly speaking, $\boldsymbol{\pi}$ is divided into 2^P terms where P is the number of pairs interacting with Lennard-Jones tail). With the majority of interactions off, faster algorithms become effective such as cluster flipping, which promotes the overall efficiency of the original simulation.

[3] Imagine that we are simulating protein folding events, then we must have prior knowledge, that is, what configurations are favorable of the folding event.

[4] Typically, if we alternate our Monte Carlo algorithm among MC1, MC2,..., MCn, each with $\mathrm{O}(1)$ switching probability, the resulting performance is determined on the fastest one and has minimal dynamic exponent as the fastest one too.

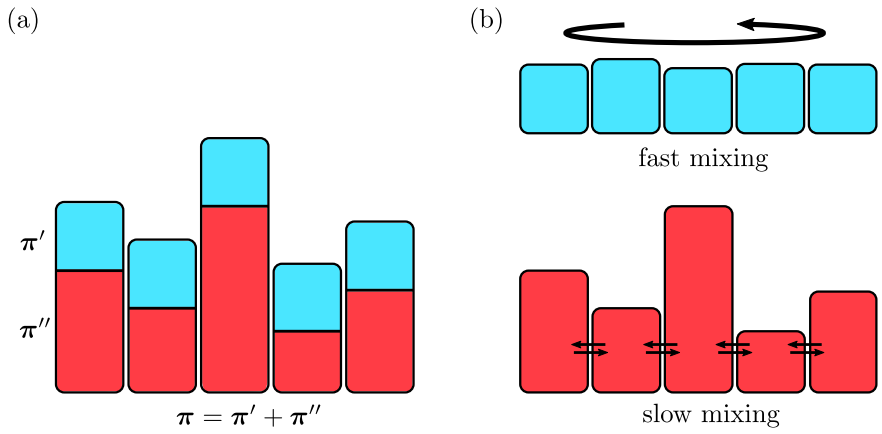


Figure 1.4: Schema of stochastic potential switching. The target distribution π is divided into π' and π'', each simulated with a specific Monte Carlo algorithm. The overall Monte Carlo is fast if one of the subsystems is fast.

Incidentally, stochastic potential switching can also derive factorization, for both Metropolis-Hastings algorithm and ECMC. Taking the Metropolis-Hastings algorithm for a two-state system as example, let $\pi(s_1) = 1$ and $\pi(s_2) = e^{-\beta(U_1+U_2)}$, then a Metropolis-Hastings acceptance ratio is $\min\{1, e^{-\beta(U_1+U_2)}\}$. We assume $U_1 \leq U_2$ and split their probabilities as

$$\begin{aligned}
\pi'(s_1) =& \min\left\{1, e^{-\beta U_1}\right\}, \\
\pi''(s_1) =& 1 - \min\left\{1, e^{-\beta U_1}\right\}, \\
\pi'(s_2) =& \min\left\{1, e^{-\beta U_1}\right\} \min\left\{1, e^{-\beta U_2}\right\}, \\
\pi''(s_2) =& e^{-\beta(U_1+U_2)} - \min\left\{1, e^{-\beta U_1}\right\} \min\left\{1, e^{-\beta U_2}\right\}.
\end{aligned} \tag{1.3.16}$$

Transition rates become switching rates multiplied by acceptance rates for both channels. One can check that the transition rate $s_1 \to s_2$ is $\min\left\{1, e^{-\beta U_2}\right\} \min\left\{1, e^{-\beta U_1}\right\}$ through π', and is 0 through π'' under the condition $U_1 \leq U_2$ (otherwise we switch U_1 and U_2). The result is the same as the factorized Metropolis-Hastings method (see section 1.3.3.1).

Replica exchange Monte Carlo

Replica exchange Monte Carlo is also known as extended ensemble Monte Carlo. The equilibrium state is physically a function of temperature $\pi = \pi(\cdot, \beta)$, and it is intuitive that, for one dimension, the high temperature boosts the transition across the energy barrier between local minimums. One way to exploit high-temperature's fast transition while leaving the estimation unbiased is by extending the ensemble to replicas of multiple temperatures.

The state space $\{s_i\}$ is then given an additional temperature variable $\{s_i, \beta_k\}$ with $\beta_k \in \{\beta_1 > \beta_2 > \cdots > \beta_B > 0\}$ and β_1 is the desired system temperature. $\pi(s_i, \beta_k) \sim$

$e^{-\beta_k U(s_i)}/Z_k$ agrees with the state probability at β_k, and the program just samples in this extended space. If we hope that program to stay in all temperature replica with the same probability then we have to set $Z_k = \sum_{s_i} e^{-\beta_k U(s_i)}/B$, the free energy at β_k divided by B.

Simulated tempering proposed by Marinari and Parisi [34] keeps one replica at each moment as normal Monte Carlo. It travels at fixed β_k, but jumps to β_{k-1} or β_{k+1} occasionally and leaves the physical state invariant. To reach homogeneous probability in β, the spirit of Monte Carlo is employed again, and the simplest Metropolis-Hastings method (see section 1.3.3.1) suggests $r = \min\{1, e^{-(\beta_{k\pm1}-\beta_k)U(s)} Z(\beta_k)/Z(\beta_{k\pm1})\}$ where $U(s)$ is the energy for the current state independent of β. Numerous problems revolve around simulated tempering. How to acquire $Z(\beta)$? Is the homogeneous distribution always good? Are there better schemes than local jumps of the Metropolis-Hastings style?

Hukushima and Nemoto [35] proposed parallel tempering where B states each with $\beta_1, \cdots, \beta_B$ are kept and simulated simultaneously. At certain moments neighboring states exchange their configurations, i.e., $\{s_1, \beta_k\}, \{s_2, \beta_{k+1}\} \to \{s_2, \beta_k\}, \{s_1, \beta_{k+1}\}$ and the Metropolis-Hastings style acceptance ratio is $\min\{1, e^{-(\beta_k-\beta_{k+1})(U(s_2)-U(s_1)}\}$. Parallel tempering serves as a powerful tool for the acceleration of all Monte Carlo simulations and attracts a great number of studies over its implementation and adaptation. The additional temperature axis brings another degree of freedom concerning the temperature switching scheme. Finally, it is important to note that the number of temperatures scales as the total energy variation in order to keep a reasonable acceptance in switching, while the total energy is usually proportional to the system size.

1.3.3 Several reversible algorithms

If detailed balance holds, the chance for a chain $(s_1, s_2, \cdots, s_n)$ equals its reverse because

$$\begin{aligned} &P(\text{chain} = (s_1, \cdots, s_n)) = \pi(s_1) T_{s_1 s_2} \cdots T s_{n-1} s_n \\ =&T_{s_2 s_1} \cdots T(s_n s_{n-1}) \pi(s_n) = P(\text{chain} = (s_n, \cdots, s_1)). \end{aligned} \tag{1.3.17}$$

In the second step the detailed balance condition is applied. The converse statement is also true by expanding $P(\text{chain} = (s_i, s_j)) = P(\text{chain} = (s_j, s_i))$. Under the detailed balance condition people have developed lots of Monte Carlo variants for it allows a wide range of probability redistribution schemes.

1.3.3.1 Metropolis-Hastings algorithm

Metropolis et al. [36] first applied Monte Carlo in physical simulations, and their idea was generalized by Hastings [37]. They decomposed the choice of the next state T_{ij} into proposing followed by accepting, with probabilities f_p and f_a respectively. If the proposal rate is symmetric, i.e., $f_p(s_i \to s_j) = f_p(s_j \to s_i)$, then detailed balance requires

$$e^{-\beta U(s_i)} f_a(s_i \to s_j) = e^{-\beta U(s_j)} f_a(s_j \to s_i). \tag{1.3.18}$$

Metropolis et al. found a simple solution

$$f_a(s_i \to s_j) = \min\left\{1, e^{-\beta(U(s_j)-U(s_i))}\right\} \tag{1.3.19}$$

satisfying the detailed balance. In fact, for a two-state space, the famous Metropolis-Hastings algorithm maximizes the flow between the states.

The acceptor obviously relies on the total energy change of the system. Interestingly, if we split the total energy into terms $U = \sum_r U_r$ and let each term lead an individual acceptor, detailed balance is respected in which the final acceptance implies that all terms accept, which we called factorized acceptance ratio

$$f_a^{\text{Fact}}(s_i \to s_j) = \prod_r \min\left\{1, e^{-\beta(U_r(s_j)-U_r(s_i))}\right\}. \tag{1.3.20}$$

We call each U_r a factor, and the way to split U is called factorization. In appearance, a factorization does not speed up mixing because it reduces the transition rate while increasing the rejection rate (i.e., the probability of staying in the same state). In some special cases, factorization has inspired fast algorithms without calculating total energy (see section 3.3). If the system involves long-range interactions, any move of a particle leads to changes of $\mathrm{O}(N)$ terms, and the factorized form allows efficient sampling instead of $\mathrm{O}(N)$ enumeration, which is the key idea for ECMC to simulate long-range systems.

The Metropolis-Hastings algorithm is so simple in concept and easy to implement that it becomes the baseline of all later advances. However, such a simple idea suffers a significant drawback: a naive implementation enforces local proposals, and the detailed balance condition imposes diffusive dynamics over the probability flow, which renders the method slow to reach the stationary distribution.

1.3.3.2 Gibbs sampler

The Gibbs sampler [38] is also known as heat-bath Monte Carlo. In a multivariate system if one component of the state $s_i = (x_1, x_2, ..., x_p)$ takes among n values $v_1, ..., v_n$, Gibbs sampler relaxes one variable each time to its conditional target probability. Putting explicitly, let $s_i = (x_1, x_2, ..., x_k = v_i, ..., x_p)$ and $s_j = (x_1, x_2, ..., x_k = v_j, ..., x_p)$ then

$$T_{s_i s_j} = \frac{\pi((x_1, x_2, ..., x_k = v_j, ..., x_p))}{\sum_{v=v_1}^{v_n} \pi((x_1, x_2, ..., x_k = v, ..., x_p))} \tag{1.3.21}$$

which does not rely on the current value of x_k. The Gibbs sampler that chooses random i and updates $\{x_i\}$ satisfies detailed balance, whereas the one updating x_i in a sequential manner violates detailed balance but preserves only global balance (see section 1.4.1.1).

Taking a two-state example, the Gibbs sampler always means that the next state is s_1 with probability $\pi(s_1)$ and s_2 with $\pi(s_2)$. This is in fact slower than the Metropolis-Hastings algorithm because the latter will never allow two consecutive s_2 if $\pi(s_1) \geq \pi(s_2)$.

1.3.3.3 Overrelaxation

For simplicity, assume that we are sampling according to a univariate normal distribution $\pi(x) \sim e^{-ax^2}$, and by Gibbs sampling the transition from x to x' is identical to probability of $\pi(x')$. Adler [39] modified the proposal distribution to have the form

$$T_{x,x'} = \frac{1}{Z} \exp\left\{ -\frac{a}{\omega(2-\omega)} \left[(1-\omega)x - x' \right]^2 \right\}, \tag{1.3.22}$$

with an adjusting parameter ω and the normalizing parameter Z. When $\omega = 0$, $x' = x$; when $\omega = 1$, the same Gaussian distribution is taken where x' centers around 0; when $\omega = 2$, $x' = -x$.

This method, called overrelaxation, directly applies to the energy of the quadratic form. Whitmer [40] showed that taking ω close to 2 leads to much faster sampling in multi-quadratic actions of the lattice model. Moreover, as the number of variables increases, ω close to 2 is more favorable, meaning that the algorithm always tends to negate the previous x_i.

Efforts have been made to generalize to arbitrary energy functions since the overrelaxation of quadratic form was proposed. Parameterization into a conditional Gaussian distribution [41] is possible, yet more systematic solutions [42] rely on sampling and filtering. Neal [43] proposed a smart algorithm by sampling a number of the variable to be updated, conditioned on other variables, and taking the same index as in the old value but in reverse order.

1.3.3.4 Cluster Monte Carlo

Cluster Monte Carlo very well reflects the advantage of the Monte Carlo method. It performs substantial changes in the system, connecting two faraway points in the phase space that molecular dynamics cannot achieve. Some systems are really hard to sample using local methods. For example, the two-dimensional Ising spin at the critical point has infinite spatial correlation, giving rise to huge clusters (i.e., connected same spins)[5]. Here we present briefly cluster Monte Carlo for spins, hard disks, and polymers (see [44] and section 2.4 of [45] for reference).

Spin Swendsen and Wang [33] first proposed a fast-relaxation algorithm for the critical-point 2D Ising model by constructing clusters based on configurations and randomly assigning the same spin within clusters. This idea was reformulated by Wolff [46] later to become the most popular cluster spin algorithm.

Each cluster move of the Wolff algorithm starts with one random spin, adds a same neighboring spin into the cluster with probability $e^{-2\beta J}$ where coupling constant appears in $U = -\sum_{i,j} J s_i s_j$, and flips the spin of the entire cluster as one move. They argued that this sophisticated rejection-free scheme satisfies detailed balance, and numerical tests showed that it brought the dynamical critical exponent down to $z = 0.13$ from $z \approx 2$ of local Monte Carlo.

[5] At high temperature, the Ising model has small clusters, so it is easy; at low temperature, it is easy too if the entire-system flipping is introduced.

The success in the Ising model inspired the generalization in continuous spin models [46], anti-ferromagnetic models and spin glasses [47, 48]. A key issue to the efficiency of cluster algorithms is the cluster size. We hope that both the cluster size and its complement are of the magnitude of system size, which occurs at the percolation threshold.

Hard disk Large-scale moves of hard disks are more easily realized in relatively low density. The best-known algorithms are the geometric cluster algorithm [49] and the Jaster algorithm [50], both of which favor low density to achieve fast mixing.

The geometric cluster algorithm chooses a pivot to perform reflections on the original configuration. The overlap of new and old configurations induces bonds between disk pairs, and clusters are formed with the inter-disk bonds. Reflecting one entire cluster does not have conflicts with any other cluster, so clusters can be either reflected or remain unchanged, which is rejection-free. On the other hand, the Jaster algorithm starts with translating a random disk by distance ℓ in a random direction, and does the same for the one causing collision until no collision occurs, and it accepts the translated configuration. If one translation results in more than one conflict, then it is rejected. The same concern of moved cluster size exists in both algorithms. Incidentally, the Jaster algorithm resembles ECMC in the sense that they both induce a chain of hard disks displaced in the same direction.

Polymer A polymer can be described as chains of interacting particles $p_1, p_2, ..., p_N$. The ties between two adjacent particles within a chain are much stronger, sometimes even rigid, than the rest of particle pairs, so polymer transformation is usually considered as an unconventional move to fast explore the polymer's conformational space.

Proposed by Siepmann and Frenkel [51] is a typical Monte Carlo algorithm by resampling part of the polymer's chain. It would be easier if we set all particles on lattice sites. First, it relaxes a random segment $p_k, p_{k+1}, ..., p_N$ of the chain from one end, then settles p_k and records non-occupied sites W_k' and all possible sites W_k for it. And do the same for p_{k+1} until p_N. The acceptance rate is the product $\prod_i^N W_i'/W_i$ which is lowered with increasing density.

1.4 Irreversible Monte Carlo

Irreversible Monte Carlo evolves through a Markov chain with a definite time direction, generating a Markov chain with a different probability from its reversed chain. As reversibility is synonymous with the detailed balance condition, the reversible probability flows f_{ij} and f_{ji} cancel between i and j under detailed balance, so the program explores the configuration space in a random-walk manner. Only with detailed balance, universal algorithms exist, such as the Metropolis-Hastings algorithm and the Gibbs sampler applicable to all energy functions.

We call a series of operations that change $\mathrm{O}(N)$ elements a sweep, such as N single-flip trials for a spin system. To understand the dynamical benefit brought by irreversible Monte Carlo, we express the time consumption to draw an effective sample

as

$$t = C * t_{\text{swp}} * t_{\text{cor}}. \tag{1.4.1}$$

Time for a sweep t_{swp} is $\mathrm{O}(N)$ for short-range system, and is complicated for long-range system depending on methods to compute $\mathrm{O}(N)$ interactions. It is also noteworthy that a larger system usually implies a greater rejection rate if pairwise factorization is employed. The correlation time t_{cor} indicates how many sweeps it costs to have an independent sample. It is not evident but is usually acquired by realistic measurements, taking the form of $t_{\text{cor}} = \mathrm{O}(L^z)$ where L is the system side length, since it relates to physical modes propagating across the system. z is called dynamical exponent and has special physical interests.

Since the invention of reversible Monte Carlo, people have been trying to improve it by introducing directions to guide the program in the state space. Random walks in a system of size N usually take $\mathrm{O}(N^2)$ steps, and from the perspective of a continuous system, the Metropolis-Hastings algorithm introduces diffusive dynamics, so any density disturbance takes $\mathrm{O}(L^2)$ to propagate, that is why it often results in $z \approx 2$ in many common systems.

Turitsyn et al. [52] draw an analogy to mixing sugar in a cup of coffee, in which detailed-balance Monte Carlo with diffusive behavior equilibrates like the sugar naturally dissolving into the coffee. Stirring the liquid will conceivably accelerate the mixing, and in this case, there is a preferred direction for any point in the configuration space. Illustrated in Fig. 1.5 are different degrees of irreversibility. Sometimes it is not difficult to construct small loops in a micro-system, i.e., directional tour in a few local states, which proves helpful but cannot essentially change Monte Carlo's behavior. Only with a global direction can an algorithm get rid of $\mathrm{O}(L^2)$ scaling and achieve fast exploration.

However, there is no easy construct of irreversible Monte Carlo, for it must satisfy the global balance condition. Methods presented in the following achieve irreversibility either with limited improvement, or under additional information. Our goal is to have an algorithm that eliminates the random-walk behavior so that observable autocorrelations will not scale up as $\mathrm{O}(L^2)$. The challenge is that even people painstakingly devise an irreversible Monte Carlo, it proves locally irreversible (case of Fig. 1.5 (b)), and the exponent z does not improve. By contrast, ECMC works in any continuous space[6], and achieves a reduction in the dynamical exponent in many systems though its move is local.

We classify irreversible Monte Carlo methods into two classes: one with lifting variables and the other with no lifting variable. Information kept in lifting variables somehow resembles a general direction, so it presents more hope to achieve global directionality, whereas non-lifting Monte Carlo has no more information than traditional ones, so it is expected to only accelerate by a constant.

[6]Converting a discrete space to a continuous one is possible, with artificial constraints and potentials, but there is no study on whether ECMC keeps its advantages under such environment.

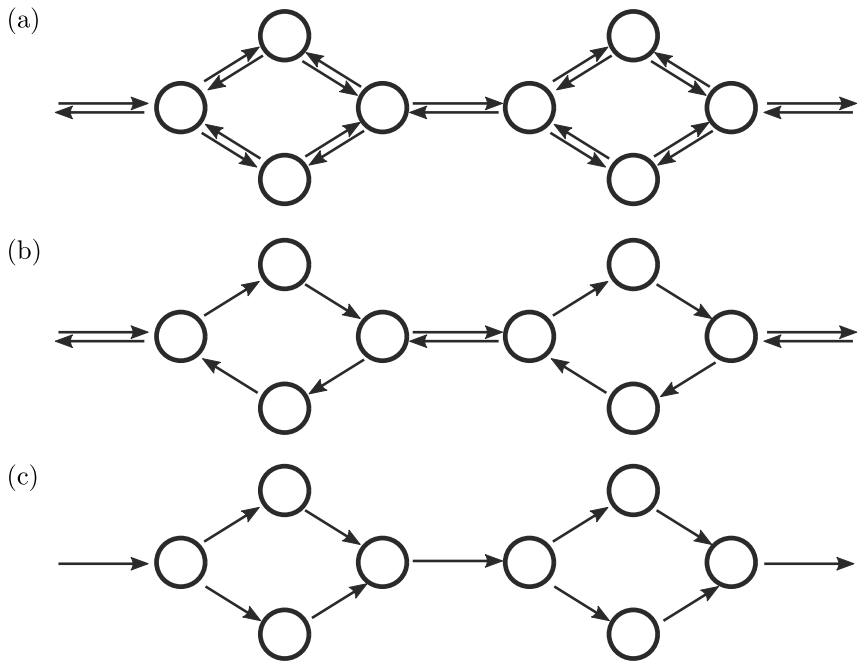


Figure 1.5: Different degrees of irreversibility. (a) Reversible Monte Carlo that satisfies $f_{ij} = f_{ji}$. (b) Locally irreversible Monte Carlo which has the same complexity in traveling the whole space. (c) Globally irreversible Monte Carlo traveling at lower complexity.

1.4.1 Non-lifting irreversible Monte Carlo

While developing ECMC, we always wonder what role the lifting variable that indicates active particle plays for the efficiency of ECMC. Such a lifting variable makes the active particle motion persistent, and by imposing very short chains, the performance of ECMC returns to the case of traditional Monte Carlo. However, advancing active particles in the same direction is not equivalent to traveling in phase space without turning back. On the other hand, keeping explicit directions in lifting variable does not always help, as suggested by the example of skew detailed balance (see section 1.4.2.2).

By presenting three examples of non-lifting Monte Carlo followed by two with lifting, we review people's effort in achieving irreversibility. Actually, two of three non-lifting irreversible Monte Carlo keep additional implicit information: The next variable to relax in a sequential scheme, and the momentum in hybrid Monte Carlo. Checking how different degrees of irreversibility improve mixing provides us instructive (though feeble) clues over the role of lifting in ECMC's excellent performance.

1.4.1.1 Sequential updates

The most straightforward irreversible scheme comes from updating variables in a sequential manner. While performing Metropolis-Hastings or Gibbs style updates on spin systems, instead of randomly choosing the next spin to flip for the Ising model, we can try to change spin 1, spin 2,..., until spin N and then go back to spin 1. This method is obvious without detailed balance since one cannot change back spin k immediately after making trial on it.

Denote $T^{(k)}$ as the transition matrix for relaxing the variable k using any reversible scheme. The whole procedure is then imposing $T^{(i)}$ consecutively for i from 1 to N. Since each $T^{(k)}$ satisfies detailed balance, the product $T^{(1)}T^{(2)}\cdots T^{(N)}$ preserves global balance.

Early in the inital paper [36] of Monte Carlo methods, Metropolis et al. has already applied sequential updates of particles. Ren and Orkoulas [53] investigated the performance of a sequentially updating scheme in comparison with a random one. Intuitively, updating one spin will result in the tendency of changing its neighbors. They argued that the transition matrix $T^S = \prod_{i=1}^{N} T^{(i)}$ has smaller diagonal terms T^S_{ii} than those for random updates. According to Peskun's theorem [54], a transition matrix with small diagonal terms implies less rejection rate and faster equilibration.

The advantage of sequential updates is not only a prefactor faster in simulation, but underpins a rigorous parallelization paradigm. Ren and Orkoulas [55] showed that sequential updates can be split into several non-interacting branches based on domain decomposition, which can be performed concurrently. Multiple processors will be able to execute each branch in a parallel manner without losing accuracy.

In fact, the concept of sequential updates has broader usage than just in Metropolis-Hastings or Gibbs updates. It favors a manual selection of the next variable to be relaxed. Lei and Krauth [56] proved that, if we treat each chain in ECMC as an operation, for N hard disks on a ring, a sequential choice of the initial particle of each chain can reduce the mixing time scaling from $\mathrm{O}(N^2 \log N)$ to $\mathrm{O}(N^2)$. Furthermore,

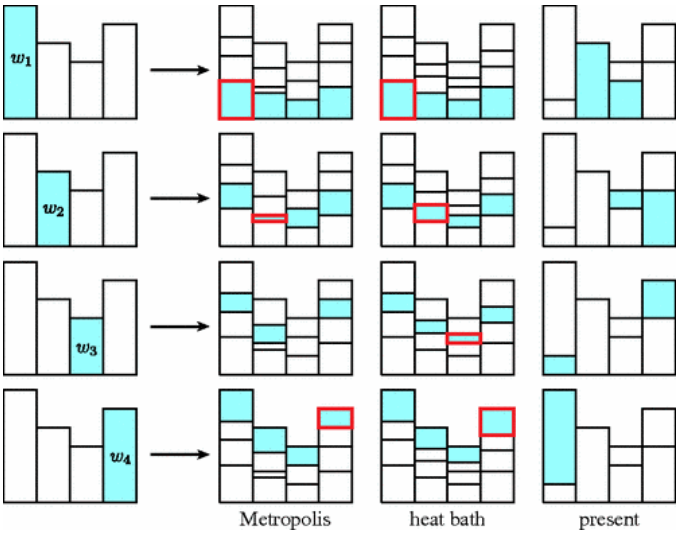


Figure 1.6: Schema of probability allocations under global balance condition. Three algorithms are compared: Metropolis-Hastings, Gibbs samples and Suwa-Todo. Each column represents a state with a solid rectangle and a frame. A valid Markov chain allocates rectangles into frames and the rejection is indicated with self-allocation. (From Ref [57, 58].)

a manual sequence of directions is valid, such as $xyzxyz...$. Fig. 3.19 in section 3.6.2 confirms the faster relaxation by sequential directions.

1.4.1.2 Geometric Allocation

When the number of states is limited, Suwa and Todo [57, 58] graphically interpreted the global balance condition as reallocating probabilities. They found an efficient scheme with zero flow into the same configuration.

Demonstrated in Fig. 1.6 are probability allocations for three common algorithms. Each state s_i has contribution $\pi(s_i)$ (i.e., outgoing flow) and capacity $\pi(s_i)$ (i.e., incoming flow), illustrated as color bars and frames respectively in Fig. 1.6. The Metropolis-Hastings algorithm is the random proposal followed by a ratio filter, and Gibbs sampler relies on proportionate redistribution, both of which cannot avoid self-flow f_{ii}. Instead, the Suwa-Todo algorithm starts from the maximum-probability state denoted as s_1, inserts contributions $\pi(s_1), \pi(s_2)...$ into a first-in-first-out (FIFO) queue. Then, it begins fulfilling capacities from s_2 with elements outputting from the FIFO queue. In this way, if $\pi(s_1) \leq 1/2$, the rate of rejection is always zero.

For systems whose configurations are countable, it is usually possible to perform geometric allocation with respect to individual components. For a state consisting of p variables, $s = (x_1, x_2, ..., x_p)$ with x_k taking value among $v_1, ..., v_n$, a list of n elements $\pi((x..., x_{k-1}, \cdot, x_{k+1}, ...))$ will be created with n elements to update variable x_k.

Direct applications include the Potts model [57, 58] in which the Suwa-Todo algorithm exhibits considerable acceleration depending on the system specification. An interesting application is the temperature change in extended-ensemble Monte

Carlo (see section 1.3.2). People have tested the Suwa-Todo allocation in temperature updates for both simulated tempering [59] and parallel tempering [60], and it is reported that the irreversible Suwa-Todo scheme facilitates temperature exchange and hence the overall performance. Despite the zero probability returning to any state, the Suwa-Todo method remains locally irreversible and the speedup is up to a constant, because it is usually unfeasible to put the entire phase space to the FIFO queue.

1.4.1.3 Hybrid Monte Carlo

Hybrid Monte Carlo, also known as Hamiltonian Monte Carlo, was introduced by Duane et al. [61]. It applies only to continuous configuration space because it uses a molecular dynamics integrator to generate proposal moves. The Metropolis-Hastings filter is used to accept the move (see [62], chapter 5).

Hybrid Monte Carlo process can be described as:

1. Extend the state space $s = (q_1, ..., q_n)$ to a phase space with velocity so that $s = (\mathbf{q}, \mathbf{v})$, $\mathbf{q} = (q_1, ..., q_n)$, and $\mathbf{v} = (p_1, ..., p_n)$. Define the Hamiltonian $H(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + \sum_{i=1}^{n} p_i^2/2$.

2. (a) For the current $(\mathbf{q}, \mathbf{p})$, perform Hamiltonian mechanics of Eq. 1.2.1 for time t, which reduces to $\dot{q}_i = p_i$ and $\dot{p}_i = -\partial U/\partial q_i$. Then let $\mathbf{p}' = -\mathbf{p}(t)$, $\mathbf{q}' = \mathbf{q}(t)$.
(b) Accept $(\mathbf{q}', \mathbf{p}')$ with the Metropolis-Hastings filter, and the energy is replaced with H.

3. Flip $\mathbf{p}' \leftarrow -\mathbf{p}'$.

4. Disturb $\mathbf{p}$ by $\mathbf{p}'' = \mathbf{p}'\sqrt{1-\alpha} + \alpha\mathbf{N}$, where $\mathbf{N}$ is a normal distribution with same dimension of $\mathbf{p}$ centering around zero and has variance corresponding to the temperature, α is a tuning parameter.

5. Take $(\mathbf{q}', \mathbf{p}'')$ as a new state and go to step 2 until halted.

Let $T^{(1)}, T^{(2)}, T^{(3)}$ be operators representing steps 2, 3, and 4 respectively. We can argue that hybrid Monte Carlo ensures the desired distribution $\boldsymbol{\pi}$. First, $\mathbf{p}$ and $\mathbf{q}$ are decoupled in H so $\int_{\mathbf{p}} \boldsymbol{\pi}(\mathbf{q}, \mathbf{p}) \propto e^{-\beta U(\mathbf{q})}$. Second, which is crucial, Hamiltonian mechanics is reversible and volume-preserving, thus $T^{(1)}$ respects detailed balance, and detailed balance for $T^{(2)}$ and $T^{(3)}$ is obvious, so their product $T^{(1)}T^{(2)}T^{(3)}$ respects global balance. Third, ergodicity is ensured by $\alpha > 0$.

Flipping twice $\mathbf{p}$ is for arguing the balance condition, which needs not explicitly performing. Step 2 must be done with a numerical integrator, and it follows that the integrator must be time-reversible and volume-preserving, such as the leap-frog method. Nevertheless, the entire scenario is clear: New state is deterministically generated by the Hamiltonian integrator for certain time steps. If $\alpha = 1$, it accepts the new state through the filter, then erases the previous momentum and resamples anew. If $\alpha = 0$, it keeps the momentum for an accepted move and reverses the momentum for a rejected move. Keeping the momentum contributes to directional exploration, while momentum reversal makes it double back. Thus α serves as a crucial parameter over

reversibility[7], and good balance [63] must be kept for optimal efficiency. Also, hybrid Monte Carlo introduces a set of tuning parameters such as the small step ϵ, t, and possibly masses m_i assigned to each momentum, which all affect the performance.

1.4.2 Lifting irreversible Monte Carlo

Reversible Monte Carlo methods employ symmetric proposal probabilities $f_p(i \to j) = f_p(j \to i)$ so that tuning the acceptance rate allows us to realize detailed balance. Many systems achieve fast exploration with explicit use of direction. For example, particle momentum counteracts diffusive dynamics to make components in a fluid mix faster. Angular momentum drives a molecule under constant rotation. Persistence in flipping up spins or flipping down will also lead to rapid decorrelation between spin configurations.

Lifting variables allow people to add information in Monte Carlo moves so that the program has the memory of previous moves. A state s_i is extended with a lifting variable $s_i = (\mathbf{x}_i, a_i)$, and the proposal depends heavily upon the lifting variable a_i. Here we present several lifting irreversible Monte Carlo ideas, including one-dimensional random walk, skew detailed balance, and our ECMC.

1.4.2.1 One-dimensional random walk

A chain of N sites with uniform desired probability is one of the few models with lifting variables that rigorous mathematics can apply. Diaconis et al. [64] established the first rigorous scaling of convergence for lifting irreversible Monte Carlo, proving that its mixing time converges within $\mathrm{O}(N)$ steps compared to $\mathrm{O}(N^2)$ with an undirected random walk.

Previously, an unbiased random walk may jump from site i to $i-1$ or $i+1$ each with probability $1/2$. The end of the chain needs special treatment if periodic boundary conditions are not applied. Diaconis et al. added the direction to the state represented by (i, d) with $d = \pm 1$, and the transition rule is

$$\begin{aligned} T_{(i,d)\to(i+d,d)} &= 1 - \frac{1}{N}, \\ T_{(i,d)\to(i+d,-d)} &= \frac{1}{N}. \end{aligned} \tag{1.4.2}$$

A special transition rule is needed for the site 0 and $N-1$ since they did not employ periodicity. It means that Monte Carlo will probably move in the same direction as the preceding move with probability $1-1/N$, but may also change the lifting variable with probability $1/N$. They proved that starting with any state, the total variation distance with respect to the stationary state after l updates drops to $\sim C^{l/N}$ where $0 < C < 1$. This implies that the convergence rate in terms of sweeps, does not rely on the system size, which outperforms unbiased random walk which takes N^2 steps (namely N sweeps) to reach a distance of N. The generalization to two-dimensional cases is mentioned in Chen et al. [65]

[7] Most of the literature classifies hybrid Monte Carlo as a reversible method, perhaps due to the key step 2, which alone is reversible. However, for $\alpha > 0$ case, $(\mathbf{q}, \mathbf{p})$ travels with clear direction.

For non-uniform distribution, Diaconis et al. [64] also gave an algorithm with transition

$$\begin{aligned} T_{(i,d)\to(i+d,d)} &= p_{\mathrm{MH}}(1-\theta), \\ T_{(i,d)\to(i+d,-d)} &= p_{\mathrm{MH}}\theta, \\ T_{(i,d)\to(i,d)} &= (1-p_{\mathrm{MH}})\theta, \\ T_{(i,d)\to(i,-d)} &= (1-p_{\mathrm{MH}})(1-\theta), \end{aligned} \tag{1.4.3}$$

where $p_{\mathrm{MH}} = \min\{1, e^{-\beta(U(s_{i+d})-U(s_i))}\}$ is the Metropolis-Hastings filter, and θ is a small adjusting parameter. Special conditions are needed for sites on the boundary. Improvement depends on specific problems and individual distributions.

In fact, this one-dimensional non-uniform sampler is not unique. The Suwa-Todo algorithm in section 1.4.1.2 gives another construct with zero flow back to the same site, but needs to know $\pi(s_i)$ for all i before allocation. The ladder construct of non-detailed-balance Hybrid Monte Carlo by Sohl-Dickstein et al. [66] provides one more possibility, which reduces rates of $d \to -d$ by looking ahead (If current site is s_i, it calculates relative values of $\pi(s_i), ..., \pi(s_{i+K})$). Skew detailed balance introduced in section 1.4.2.2 has a direct application in the one-dimensional chain, and achieves minimal turnaround rate of $d \to -d$ using $\pi(s_{i\pm1})$ and $\pi(s_i)$ only.

1.4.2.2 Skew detailed balance

Skew detailed balance was first proposed by Turitsyn et al. [52, 67], a framework allowing people to manipulate proposed directions. It is constructed upon existing reversible flow.

To apply skew detailed balance, as other lifting Monte Carlo methods, we first extend any state with an additional lifting variable $s = (\mathbf{x}, a)$. For simplicity, we let $a \in \{-1, +1\}$. We have to make clear what the proposed states are for any value of $\mathbf{x}$ and a. For example, in the two-dimensional Ising model, $a = 1$ means that the next move is to flip up one spin, and $a = -1$ means to flip down one spin. Thus the proposed set of configurations for s_i are grouped into two small sets $\{p_i^-\}$ and $\{p_i^+\}$.

Then one has to choose an underlying reversible flow that is presumably Metropolis-Hastings style while other schemes have been tested [68]. We duplicate states $\{s_i\}$ into two copies $\{s_i^+\}$ and $\{s_i^-\}$. For $\{s_i^+\}$ the flow $\{p_i^-\} \to s_i^+ \to \{p_i^+\}$ are kept, and for $\{s_i^-\}$ the flow $\{p_i^+\} \to s_i^- \to \{p_i^-\}$ are kept. Also, we have to set up flows between s_i^+ and s_i^- to make incoming and outgoing flows meet, together with self-loops to reach global balance. Specifically we have

$$\begin{aligned} T_{(j,d),(i,d)} =& T_{ji}, \quad \forall j \neq i,\ j \in \{p_i^{-d}\}, \\ T_{(i,d),(j,d)} =& T_{ij}, \quad \forall j \neq i,\ j \in \{p_i^{d}\}, \\ \pi(s_i) T_{(i,d),(i,-d)} =& \left[\sum_{j\in\{p_i^{-d}\}} \pi(s_j) T_{(j,d),(i,d)} - \sum_{j\in\{p_i^{d}\}} \pi(s_i) T_{(i,d),(j,d)} \right]^+ + C, \\ T_{(i,d),(i,d)} =& 1 - \sum_{j\in\{p_i^{d}\}} T_{(i,d),(j,d)} - T_{(i,d),(i,-d)}, \end{aligned} \tag{1.4.4}$$

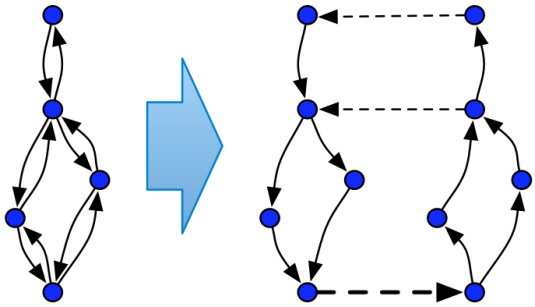


Figure 1.7: Schematic representation of the state space replicas. Dashed lines represent replica switching transitions, which compensate for compressibility of the probability flows associated with solid lines. (From Ref [52].)

where $[x]^+ = \max\{0, x\}$, $d \in \{-, +\}$ and C is a non-negative constant indicating the mutual flow between (i, d) and $(i, -d)$, restricted only by $T_{(i,d),(i,d)} \geq 0$. Fig. 1.7 illustrates the schema of skew detailed balance.

Note that skew detailed balance requires flows of both incoming and outgoing states to determine replica switching flow, which is not always feasible and limits the applicability of skew detailed balance algorithms. Turitsyn et al. [52] originally applied the skew detailed balance to the mean-field Ising model, i.e., Ising model in which any spin interacts with all others with the same intensity[8]. They found that critical slowdown appearing in the Metropolis-Hastings algorithm does not exist with irreversible skew detailed balance, which was also reported by Fernandes and Weigel [69]. Sakai and Hukushima [70] studied the transition matrix under skew detailed balance condition on a N-state circle with a uniform distribution and concluded that proper choice of parameters does improve the relaxation rate.

Further investigations were done in both one-dimensional and two-dimensional Ising models. The Ising model is one of the few models that allow us to calculate flow sums like $\sum_{s \in p_i^-} s \to s_i^+$ in $\mathrm{O}(1)$ time, with the bookkeeping of the number of spins for each kind of neighbor compositions. Sakai and Hukushima [71] discussed in detail the effect in the dynamics of different constructs of skew detailed balance. They found that one of the parameter choices was able to reduce the dynamical critical exponent. For the two-dimensional Ising model, however, no evidence shows a significant reduction in dynamical exponent [69, 72]. Nevertheless, skew detailed balance provides an ideal sampling algorithm in one-dimensional states, with the only requirement of neighboring state probabilities. Sakai and Hukushima [73] also applied it to simulated tempering and gained acceleration in the relaxation of the two-dimensional Ising model.

[8] In the mean-field Ising model, all spins are identical because interactions exist identically for all pairs. Thus for a model of N spins, a state can be represented by a number N^+ indicating the number of up-spins. This model is essentially a one-dimensional $(N+1)$-state model.

1.5 Models and measurements

We are not going to study various molecular models in this thesis but will take the all-atom SPC/Fw water model [74] as an ideal testbed.

1.5.1 Molecular models

Classical molecular models are classified into three levels—all-atom, united-atom, and coarse-grain models, depending on the represented atomic groups for one simulated unit. The all-atom model is also known as explicit-atom model, meaning that each atom is represented by one object, typically a point charge, regardless of the atom type and size. The united-atom model, in contrast, groups several atoms into one unit, usually called a radical group like -CH_2-, -CH_3 and -O-. Each united atom group is treated as one object. Moreover, a coarse-grain model combines a couple of united atoms into a bigger object. The interactions in all three models consist of bounded interactions between two adjacent units and non-bounded interactions, possibly between any pair. Parameters for the interaction are obtained from simulations or *ab initio* methods. From all-atom model, united-atom model, to coarse-grain model, they become more efficient in computation but are less accurate [75].

We focus on the all-atom water model in this thesis for the sake of simplicity even though ECMC can, in principle, handle all kinds of models and interactions.

1.5.2 Measurements

There are various approaches to measure the convergence of a certain Monte Carlo method such as correlation time, mixing time, χ^2 rate of convergence, etc (see [29] for reference). They do not agree in value and even in scaling with respect to the system size. Here we are not going to discuss their difference but present the correlation time, which is the simplest and most visible in our numerical experiments.

An observable is a function of states: $X : \mathcal{C} \to \mathbb{R}$ possibly representing any physical quantity[9], where $\mathcal{C}$ is the configuration space. Through a Markov chain we get a series of observables $\{X_i\}$ for $i = 1, ..., M$ (see section 2.3.3 for sampling in ECMC). Then we obtain its estimate by

$$\langle X \rangle = \sum_i \pi(s_i) X(s_i) \approx \frac{1}{M} \sum_i X_i, \tag{1.5.1}$$

The deviation of data scales as $1/\sqrt{M}$ and we wonder how the quality of the estimate increases with larger sample size. Explicitly, a good measure for the convergence speed is

$$\mathrm{var}(X) = \lim_{M \to \infty} M \mathrm{var} \left[\frac{\sum_{i=1}^{M} X_i}{M} \right]. \tag{1.5.2}$$

This limit converges to a constant and it is determined by the number of independent samples in a chain. Thus it is straightforward to introduce the sample autocorrelation

$$\mathrm{cor}_X(t) = \frac{\langle (X(s) - \bar{X})(X(s+t) - \bar{X}) \rangle}{\bar{X^2} - \bar{X}^2} \tag{1.5.3}$$

where $\bar{X}$ is the same as $\langle X \rangle$. By expanding $\mathrm{var}(X)$ we have

$$\begin{aligned} \mathrm{var}(X) &= \lim_{M\to\infty} \frac{1}{M} \left(\left\langle \left(\sum X_i \right)^2 \right\rangle - M^2 \bar{X}^2 \right) \\ &= \left(\sum_{t=-\infty}^{+\infty} \mathrm{cor}_X(t) \right) \left(\bar{X^2} - \bar{X}^2 \right), \end{aligned} \tag{1.5.4}$$

with the assumption that $\mathrm{cor}_X(t)$ decays exponentially for large t. Actually $\mathrm{cor}_X(t) \sim e^{-t/\tau_{\mathrm{cor}}}$, and τ_{cor} is the autocorrelation time of X. The curve $\mathrm{cor}_X(t)$ is our measure of convergence in later chapters.

[9] In some cases, an observable is also expressed as a complex number, then Eq. 1.5.3 becomes

$$\mathrm{cor}_X(t) = \frac{\langle (X(s) - \bar{X})^* (X(s+t) - \bar{X}) \rangle}{|\bar{X}|^2 - |\bar{X}|^2} \tag{1.5.5}$$

Chapter 2

Event-chain Monte Carlo

2.1 Introduction

ECMC was proposed [76] by Bernard and Krauth in 2009. It was initially designed to equilibrate hard-disk systems. In that work, hard disks undergo sequential moves called chains, during which one hard disk moves in a fixed direction until it hits another disk. Then the target disk, which previous moving disk collides with, takes over the velocity until it hits a third disk, or the sum of displacements of all disks reaches the constant chain length ℓ. This simple process preserves global balance, and different options exist in deciding the target disk's direction of motion and how to sample the initial direction for each chain. Bernard and Krauth found that the scheme moving all disks in the same direction within one chain, which violates detailed balance, achieves the fastest mixing.

Later, they applied ECMC in an unprecedented large-scale hard-disk simulation [9] and revealed the phase transition of the hard-disk system with the most convincing evidence ever. The continuous transition followed by a first-order phase transition in solid-hexatic-liquid melting scenario contradicts both the long-established one-step melting and the two-step KTHNY process [4, 5, 6, 7]. In the following work, different state-of-the-art simulation methods were implemented. ECMC, event-driven molecular dynamics and massively parallel local Monte Carlo were compared in the sense of their performance. For 512^2 hard disks, ECMC is roughly 70 times faster than local Monte Carlo and is seven times faster than event-driven molecular dynamics, and is only defeated by parallel local Monte Carlo when more than one thousand GPU cores run simultaneously.

The generalization of ECMC to arbitrary continuous potentials did not appear until the work by Peters and With [77] which prove that the global balance condition is respected if, for a factor, the energy change before the next event is drawn from an exponential distribution with mean $1/\beta$. Afterward, the extension for many-body interactions was proposed [78] by Harland et al., with the same event rate $\beta[\partial U/\partial x_{\mathrm{A}}]^+$ as a two-body factor, but one must specify an additional scheme to determine the next active particle. We will give a graphic explanation for it in section 2.2.2. In addition, Michel et al. found a succinct expression [79] to obtain NVT system pressure via ECMC statistics.

In the meanwhile, ECMC is tested in other systems including hard spheres [80], soft-potential particles [81], XY model [82, 83], Heisenberg model [84], one-dimensional short-range system [85, 56, 86], dense polymers [87], and even quantum chromodynamics [88]. It is noteworthy that in XY and Heisenberg spin models, ECMC can fast explore systems driven by spin waves, while it suffers a critical slowdown that the dynamical exponent is higher around the system's critical point. And the reason [83] is found related to the topological defect of the model. So far, all the complexities come from numerical experiments except that the mixing time of one-dimensional hard disks [85, 56] can be obtained rigorously.

The discovery of ECMC has spawned numerous studies of the irreversible Markov chains. The original work [76] tested reversible and irreversible straight-chain ECMC, and a reflected-chain alternative. Infinite choices that respect the global balance exist in theory for direction change at the collision moment and at the end of chains, yet only a few studies [89, 90] have contributed to this freedom. On the other hand, the idea of factor field [86] has lowered the dynamical exponent to the theoretical lower bound $1/2$ in one-dimensional short-range systems, a novel concept countering all existing experience gained in conventional Monte Carlo.

It is not difficult to generalize ECMC to arbitrary energy landscapes, giving rise to mathematical variants such as the bouncy particle sampler [91] and the zigzag process [92]. In the following, we will introduce ECMC using the classical physics model. The proof of global balance will not be rigorous but is brief and sufficiently convincing. In section 2.2 we explain the basic process of ECMC via infinitesimal displacement, then in section 2.3 we talk about approaches that can significantly speed up the computation.

2.2 Basics

We start with lifted configurations and derive the event rate in the presence of a single factor based on the global balance condition. Unlike other ECMC tutorials, we assume since the beginning that factors are many-body, of which pairwise interactions are special cases. Lifting schemes will also be given in section 2.2.2. Then we extend to the case of multiple factors and provide a more general view of the probability flow.

2.2.1 Lifted configurations

ECMC is built upon lifted configurations. The basics of equilibrium statistical mechanics tell us that the distribution of a physical configuration satisfies $\pi(c) = e^{-\beta U(c)}/Z$ where c is the general position of a physical system relying on $\tilde{N}$ coordinates $c = c(x_1, x_2, ..., x_{\tilde{N}})$ which we call physical variables. An extended configuration attaches a lifting variable a to the physical variable, and we denote the extended configuration as a tuple (c, a). This lifting variable often represents the moving index at the currrent moment, and in many cases a is an integer ranging from 1 to $\tilde{N}$, meaning that the coordinate x_a is changing. For a physical configuration, the probability $\pi(c)$ is equally shared by each lifted configuration, so $\pi(c, a) = e^{-\beta U(c)}/ZN_a$, where N_a

is the number of values that a can take (we sometimes omit this normalizing factor in the following arguments). The lifted configuration is also viewed as replicas of original configuration in the literature.

For example, for N particles moving along a one-dimensional chain we can set up the lifted configuration as $c = ((x_1, ..., x_N), a)$ where x_i's are coordinates and $a \in \{1, .., N\}$ meaning that x_a is advancing. For two dimensions, physical variables become $(x_1, ..., x_N, y_1, ..., y_N)$ and $a \in \{1_x, ..., N_x, 1_y, ..., N_y\}$. Generally speaking, a is allowed to take more values than the set of physical variables. For instance, if we allow one-dimensional N particles to move forward and backward, then a can take values from the set $\{1, .., N\} \times \{+, -\}$ of $2N$ elements, the sign indicating the direction of motion. Likewise, a can also indicate the mode where multiple variables change together, which we will encounter in section 5.1.4 and section 6.3.2 for molecular translation.

2.2.2 Factor event rate

In ECMC events are triggered by factors. Factors originate from interactions. They can be one interaction, one or more terms of an interaction, or even a set of interactions. Different arrangements of factors considerably contribute to ECMC's variety and flexibility. Later in chapter 4, we even regard auxiliary operations as pseudo-factor such as sampling and direction switching. Here we denote a factor as a tuple of its index set and the type (I_M, T_M) where I_M is a subset of $\{1, .., \tilde{N}\}$. We start with a single many-body factor.

Between two consecutive events, active particles move at a constant velocity. An event in ECMC changes the lifted variable, i.e., the velocity, while keeping the physical variable invariant. The universal event rate per unit length in ECMC is given by

$$q = \beta \left[\frac{\partial U}{\partial x_a} \right]^+ \tag{2.2.1}$$

where $[x]^+ = \max(x, 0)$ and $U = U(\{x_i\})$ is the potential for the factor. In the language of infinitesimal steps, the active particle has to hand over its activity with the probability $q\Delta s$ in the forthcoming small step Δs. It means that events occur only when the active particle move increases the potential, and a scheme must be specified to choose the next active particle, which we call target particle. Then the previous active particle stops, whereas the target particle starts moving until a new event happens, and the activity passes to a third particle(it can be the same as the first active particle). Fig. 2.1 gives an illustration of ECMC process. For a factor involving two particles, the other particle is the target. However, for many-body interactions, the scheme is not straightforward and is in general not unique.

Consider a lifted configuration $((x_1, ..., x_N), a)$. The incoming probability flow consists in two parts, as shown in Fig. 2.2, the mass flow $(c', a) \to (c, a)$ and the lifting flow $(c, b \neq a) \to (c, a)$ where $c = (x_1, ..., x_N)$, $c' = (x_1, ..., x_a - \epsilon, ..., x_N)$ and ϵ is a small

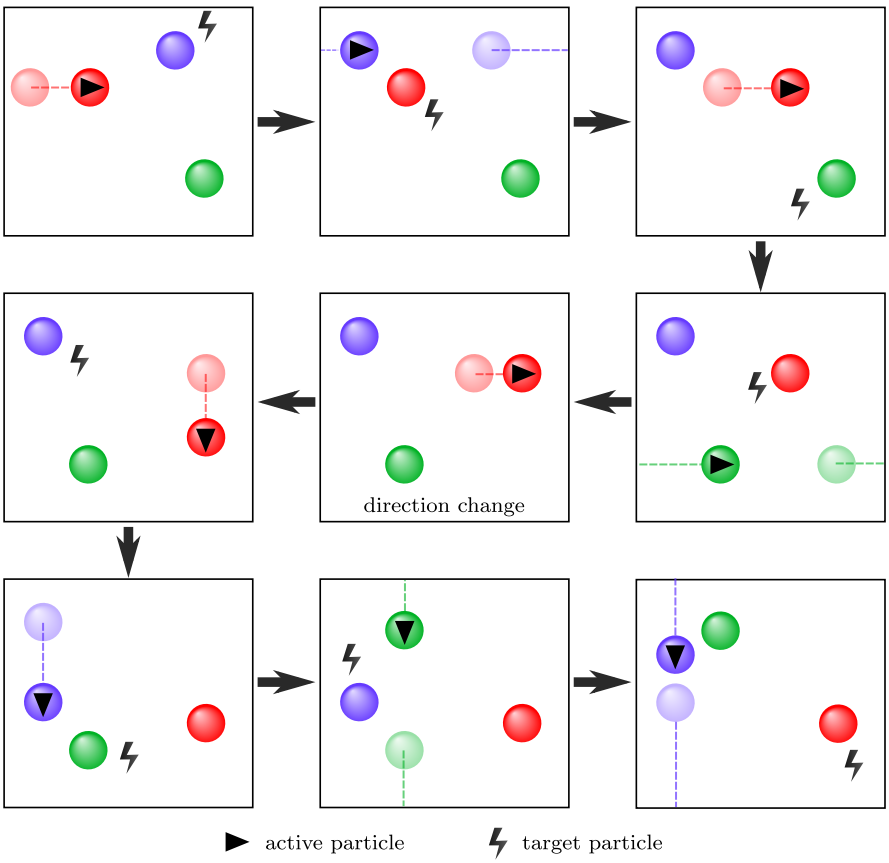


Figure 2.1: Process of ECMC evolution with three particles on a plane. We demonstrate here the system evolution under ECMC with three particles in a periodic box. The nine subfigures are the event moments, including eight physical events and one direction switching.

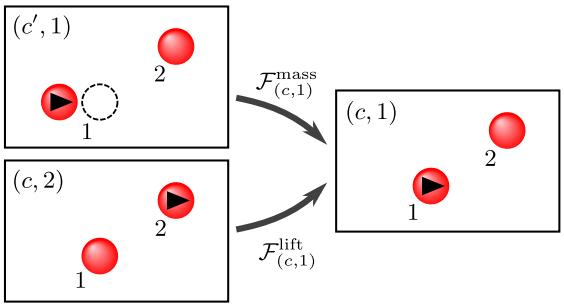


Figure 2.2: Mass flow and lifting flow into the lifted configuration $(c,1)$, where c is a two-particle state in a box. The mass flow comes from $(c',1)$ where c' is ϵ distance behind c. The lifting flow is from the state with particle 2 active. (From Ref [1].)

displacement. Ignoring the normalizing factor $1/NZ$, its value is

$$\begin{aligned}
\mathcal{F}^{\text{mass}} &= e^{-\beta U(c')}\left(1-\beta\left[\frac{\partial U}{\partial x_a}\right]^+\epsilon\right)+o(\epsilon)\\
&= \left(e^{-\beta U(c)}-\frac{\partial e^{-\beta U(c)}}{\partial x_a}\epsilon\right)\left(1-\beta\left[\frac{\partial U}{\partial x_a}\right]^+\epsilon\right)+o(\epsilon)\\
&= e^{-\beta U(c)}\left(1+\beta\frac{\partial U}{\partial x_a}\epsilon\right)\left(1-\beta\left[\frac{\partial U}{\partial x_a}\right]^+\epsilon\right)+o(\epsilon)\\
&= e^{-\beta U(c)}\left(1+\beta\left[\frac{\partial U}{\partial x_a}\right]^-\epsilon\right)+o(\epsilon),
\end{aligned} \tag{2.2.2}$$

where $[x]^- = \min(x,0)$.

On the other hand, the lifting probability flow coming from (c,b) with $b\neq a$ is

$$\mathcal{F}^{\text{lift}} = \sum_{b\neq a} e^{-\beta U(c)}\beta\left[\frac{\partial U}{\partial x_b}\right]^+\epsilon\lambda_{ba}+o(\epsilon). \tag{2.2.3}$$

Here we refer to λ_{ij} as the probability to choose j as the proceeding active particle when i is active at the event time. It has $\lambda_{ij}\in[0,1]$ and $\sum_j\lambda_{ij}=1$.

By applying global-balance condition $\mathcal{F}^{\text{mass}}+\mathcal{F}^{\text{lift}}=\pi(c)=e^{-\beta U(c)}$, we get

$$\left[\frac{\partial U}{\partial x_a}\right]^- + \sum_{b\neq a}\left[\frac{\partial U}{\partial x_b}\right]^+\lambda_{ba}=0,\quad \forall a. \tag{2.2.4}$$

If $\partial U/\partial x_a>0$, then $\lambda_{ba}=0$ for all b, meaning that events happen while the energy is increasing. If $\partial U/\partial x_a\leq 0$, no event happens as particle a moves, but moves of other particles within the factor may activate a with the event rate multiplied by λ_{ba} that obeys

$$\sum_{b\neq a}\left[\frac{\partial U}{\partial x_b}\right]^+\lambda_{ba} = -\frac{\partial U}{\partial x_a},\quad \forall\frac{\partial U}{\partial x_a}<0. \tag{2.2.5}$$

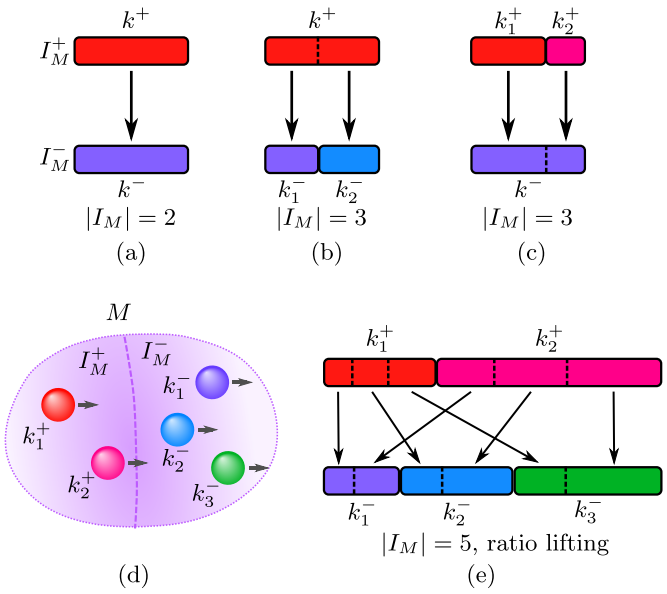


Figure 2.3: Graphic interpretation of lifting schemes. (a) Two-body factor. (b) (c) Three-body factors. Choices for $|I_M| \leq 3$ is unique. (d) (e) Five-body factor, for which there are infinite ways to allocate positive bars to negative bars. (From Ref [1].)

To better understand this process, at the instant when the event happens, we divide all particles within this factor into two sets I^+ and I^-, so that

$$\partial_{x_i} U < 0, \quad \forall i \in I^-, \tag{2.2.6}$$
$$\partial_{x_i} U \geq 0, \quad \forall i \in I^+. \tag{2.2.7}$$

Then we create bars for each of them whose lengths equal $|\partial_{x_i} U|$. Put the bars with positive derivatives above, and place those with negative derivatives below. Total length of positive bars should equal total length of negative bars due to the translational invariance, namely $\sum_i \partial U / \partial x_i = 0$. We refer to a specific choice of λ_{ba} as a lifting scheme, which can be interpreted as the allocation of positive bars into negative bars, as shown in Fig. 2.3.

For a factor (I, T) with $|I| = 2$, the target particle is unique, and $\lambda_{ij} = 1$ for $\partial_{x_i} U > 0$ and $\partial_{x_j} U < 0$ otherwise $\lambda_{ij} = 0$. For $|I| = 3$ the choice is unique, too, while a random number is needed if two particles have negative derivatives. For $|I| > 3$ there can be infinite choices, a simple ratio lifting scheme is

$$\lambda_{ij} = \frac{|\partial_{x_j} U|}{\sum_{j' \in I^-} |\partial_{x_{j'}} U|}, \tag{2.2.8}$$

which is independent of i. Now we obtain a general lifting scheme, and in section 3.5 we will order the atoms according to their molecular indices and devise lifting schemes that prioritize the flow going inside or outside the active molecule.

2.2.3 Factorization

2.2.3.1 Multiple Factors

If the potential can be written as a sum of many terms $U = \sum_{M\in\mathcal{M}} U_M$, each U_M can be regarded as a factor and it can trigger events independently.

Each factor M that involves active particle a has an effect at every small step of the ECMC process, with the same expression of the event rate $\beta[\partial U_M/\partial x_a]^+$, and with individual lifting schemes λ_{ij}^M. Throughout the derivation in section 2.2.2, ϵ is small enough that two events are improbable to coincide in the same interval ϵ, so that the probability of encountering more than one events is at most $\mathrm{O}(\epsilon^2)$.

Therefore the probability flows in existence of many factors, take similar forms as Eqs. 2.2.2 and 2.2.3

$$\begin{aligned}\mathcal{F}^{\text{mass}} &= e^{-\beta U(c')}\left(1-\sum_M \beta\left[\frac{\partial U_M}{\partial x_a}\right]^+\epsilon\right)+o(\epsilon)\\ &= e^{-\beta U(c)}\left(1+\sum_M \beta\left[\frac{\partial U_M}{\partial x_a}\right]^-\epsilon\right)+o(\epsilon),\end{aligned} \tag{2.2.9}$$

$$\mathcal{F}^{\text{lift}} = \sum_M\sum_{b\neq a} e^{-\beta U(c)}\beta\left[\frac{\partial U_M}{\partial x_b}\right]^+\epsilon\lambda_{ba}^M + o(\epsilon), \tag{2.2.10}$$

with λ_{ij}^M that

$$\left[\frac{\partial U_M}{\partial x_a}\right]^- + \sum_{b\neq a}\left[\frac{\partial U_M}{\partial x_b}\right]^+\lambda_{ba}^M = 0, \quad \forall a. \tag{2.2.11}$$

It is easy to verify $\mathcal{F}^{\text{mass}} + \mathcal{F}^{\text{lift}} = e^{-\beta U(c)} + o(\epsilon)$ and global balance holds in presence of more than one factors.

Factorization in traditional Monte Carlo does not usually have advantages. Splitting the potential into small factors often leads to a higher rejection rate, thus slows the system's relaxation. However, ECMC favors factorization for the reason that factors involving many particles have to invoke complicated lifting schemes that require all relevant particles' derivatives at the event time. On the contrary, splitting factors usually facilitates the calculation of the event time and the target particle. In the simulation of soft-potential particles [81], the two-body factor is always employed. Another example of factorization is the particle system interacting via the Lennard-Jones potential $U = A/|\mathbf{r}|^{12} - B/|\mathbf{r}|^6$, where it is favorable to split into $A/|\mathbf{r}|^{12}$ and $-B/|\mathbf{r}|^6$ so that both terms can be easily inverted.

2.2.3.2 Redundant flow

We can interpret the difference between factorizations as introducing loops of excess probability flow. Interaction consisting of many terms $U = \sum_M U_M$ may be treated as one or more small factors. If only one factor exists, particles with positive derivatives are sources of the probability transfer, i.e., have outgoing flows; particles with negative derivatives are receivers of the probability flow. When we split one factor into

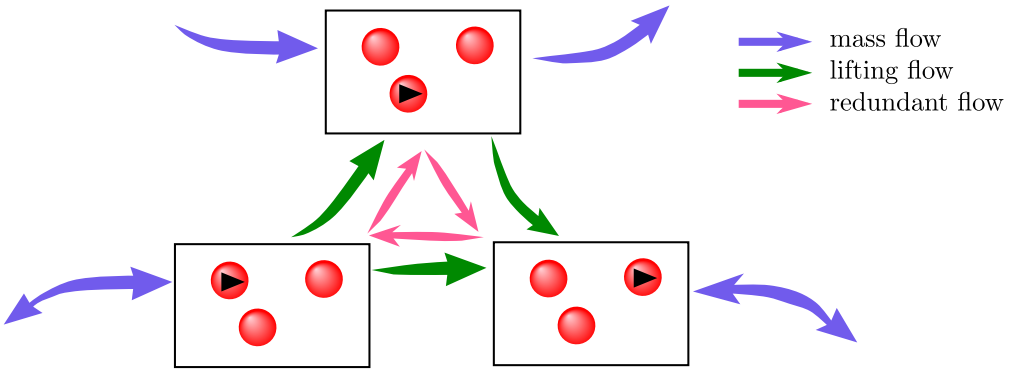


Figure 2.4: Redundant flow loop caused by factorization. Three boxes represent the lifted configurations with a common physical state. Each of them has incoming and outgoing mass flows to other physical states, and their potential induces lifting flow in between. By splitting the potential into more than one factor, we effectively impose extra loop to the lifting flow.

many, additional loops are added to the original graph, resulting in loops of redundant flow. Fig. 2.4 gives an example of this operation.

We can describe a redundant flow loop by a set of additional sources $q_i^R > 0$ for $i = 1, ..., N$, and a distribution ratio λ_{ij}^R. The Eqs. 2.2.2 and 2.2.3 now become

$$\begin{aligned} \mathcal{F}^{\text{mass}} &= e^{-\beta U(c')} \left(1 - \beta \left[\frac{\partial U}{\partial x_a} \right]^+ \epsilon - q_a^R \epsilon \right) + o(\epsilon) \\ &= e^{-\beta U(c)} \left(1 + \beta \left[\frac{\partial U}{\partial x_a} \right]^- \epsilon - q_a^R \epsilon \right) + o(\epsilon) \end{aligned} \tag{2.2.12}$$

$$\mathcal{F}^{\text{lift}} = \sum_{b \neq a} e^{-\beta U(c)} \left(\beta \left[\frac{\partial U}{\partial x_b} \right]^+ \lambda_{ba} + q_b^R \lambda_{ba}^R \right) \epsilon + o(\epsilon), \tag{2.2.13}$$

under the following constraint

$$\sum_{b \neq a} q_b^R \lambda_{ba}^R = q_a^R, \quad \forall a. \tag{2.2.14}$$

Again, $\mathcal{F}^{\text{mass}} + \mathcal{F}^{\text{lift}} = e^{-\beta U(c)} + o(\epsilon)$ holds in the presence of the term q^R.

The similarity between Eqs. 2.2.9 $\sim$ 2.2.11 and Eqs. 2.2.12 $\sim$ 2.2.14 suggests their same mathematical essence. Factorization stems from splitting physical interactions, which is more apparent and intuitive, while the construct of redundant flow can be arbitrary and flexible.

It is the flexible choice of probability flows that brings ECMC with infinite variety and enormous posibilities. In the first place, all interactions are one factor U; thus, the incoming flow and outgoing flow do not coexist upon one coordinate. Then we split U into smaller factors, usually for computational convenience. Sometimes one may even add flows from nothing, i.e., by adding $0 = U - U$, which is our current manner for direction change that can be considered as a mutual flow $x \leftrightarrow y$ that

guarantees ergodicity. It appears that redundant flows have no gain in simulation dynamics, making it more likely to stay with the current configuration. However, the construct of the factor field [86] enables us to lower the dynamical exponent in a one-dimensional system, countering people's intuition.

To sum up, the ECMC algorithm for single active particles via a time-driven approach can be described as follows, with the example of N particles moving in one direction:

1. Initialize the lifted configuration (c, a).

2. Iterate over the factors U_M with $x_a \in U_M$.

3. For each factor, calculate the corresponding derivative with respect to x_a. Draw a random number $r \in [0, 1]$, if r happens to be in $[0, \beta\,[\partial_{x_a} U_M]^+\,\epsilon]$, an event occurs. If more than one events coincides within ϵ, then pick a random one. ϵ should be tuned to ensure that coincidences are very rare.

4. If an event occurs, calculate the next active particle using the derivatives of all particles of the factor. For two-body factors, the target particle is simply the non-active one. The current active particle stops, and the target particle starts moving.

5. If no event occurs, advances the active particle by ϵ.

6. Go back to step 2 and repeat.

2.3 Fast implementation

If ECMC stays in the time-driven approach, it will hardly be capable of competing with other well-established methods. It consumes a great deal of time to integrate over small timesteps and introduces imprecision that is hard to analyze. Molecular dynamics has accumulated progress to overcome this inefficiency over the years, by parallel computing forces throughout the entire system, or altering potentials into piecewise-constant ones (see section 1.2.3) to which event-driven approaches can apply.

ECMC is a stochastic process with a natural event-driven formulation with whatever potential the system has. In section 2.3.1 we describe how to convert small step integration into inverting potential functions. Then in section 2.3.2, we introduce the bounding potential that simplifies the event calculation so that only the derivative of the original potential is called.

2.3.1 Event-driven approach

Assume that the system has only one factor. At a moment the active particle is at position x_a. Its probability to travel a distance of η without any event rejecting it,

$p(\eta)$, satistifies the equation

$$\frac{\mathrm{d}p}{\mathrm{d}\eta} = -p\beta \left[\frac{\partial U}{\partial x}\right]^+ \Bigg|_{x=x_a+\eta}, \tag{2.3.1}$$

with the initial condition $p(0) = 1$. By integrating over η, we get

$$p(\eta) = e^{-\beta \int_{x_a}^{x_a+\eta} [\partial_x U]^+ dx} = e^{-\beta U^+(\eta)} \tag{2.3.2}$$

where $U^+(\eta) = \int_{x_a}^{x_a+\eta} [\partial_x U]^+ \,\mathrm{d}x$ keeps the positive-derivative part of U and is flat otherwise, as exemplified in Fig. 2.5.

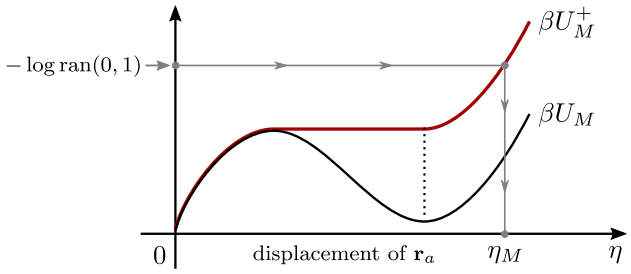


Figure 2.5: Sampling of the displacement in an event-driven manner for a potential U_M. U_M^+ represents the part of U_M with positive derivative. Then an exponential random number is drawn as the energy change. The displacement is calculated and matches the energy change of U_M^+. (From Ref [1].)

The Eq. 2.3.2 tells us that the probability for the active particle to go beyond η is $e^{-\beta U^+(\eta)}$. The change of U^+ to next event can be sampled via

$$\Delta U^+ = -\frac{1}{\beta} \log r \tag{2.3.3}$$

in which r is a random number uniformly distributed in $(0, 1]$. Then we invert $\Delta U_M^+(\eta)$ to get η.

For the case with more than one factor, we can obtain the displacement η_M from each relevant factor M. The displacement to the next event is the minimum of displacements among all factors, and the triggering factor is the one with smallest displacement,

$$M_{\mathrm{trigger}} = \mathrm{argmin}_{M,a\in I_M} \{\eta_M\}, \tag{2.3.4}$$

$$\eta = \min_{M,a\in I_M} \{\eta_M\}. \tag{2.3.5}$$

The succeeding active particle is calculated using a lifting scheme within the triggering factor.

2.3.2 Bounding potential

The method described in section 2.3.1 requires explicitly calculating the inversion of U^+ that is not usually possible in most realistic cases. In our SPC/Fw water model that employs the Amber force field in section 5.3.3, the intra-molecular harmonic potential, and oxygen-oxygen Lennard-Jones potential are invertible, while the tension part and the Coulomb interaction with periodic boundary conditions are not. Bounding potentials allow us to sample the next event time of non-invertible potential factors.

For a potential U, the bounding potential U_B is a function of x_a whose derivative not less than $\partial_x U$ everywhere if $\partial_x U > 0$, i.e.,

$$[\partial_x U_B]^+ \geq [\partial_x U]^+ . \tag{2.3.6}$$

An example is shown in Fig. 2.6. In the time-driven approach, we can use U_B in place of U to check if an event occurs within displacement ϵ. Since U_B overestimates the event rate, a certain portion of events may be false, with ratio given by

$$r_C = \begin{cases} \left[\frac{\partial U}{\partial x}\right]^+ / \frac{\partial U_B}{\partial x}, & \text{if } \frac{\partial U_B}{\partial x} > 0; \\ 0, \quad \text{otherwise.} \end{cases} \tag{2.3.7}$$

It means that we have to perform a confirmation by drawing a random number between 0 and 1 to check if it falls within $[0, r_C]$ so the event has truly happened.

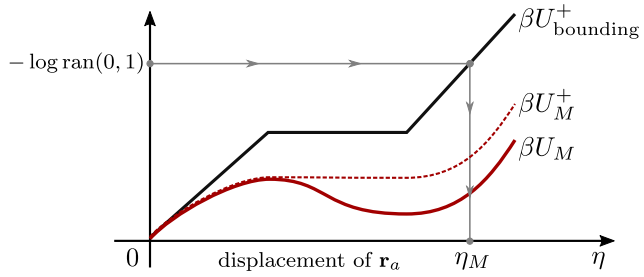


Figure 2.6: Sampling the displacement with the help of bounding potential.

The method discussed in section 2.3.1 replaces small timesteps with a direct sampling of event times. Likewise, we can sample the displacement given by an bounding potential in an event-driven manner, then confirm it by Eq. 2.3.7. Since no more restrictions exist on the specific form of bounding potential except Eq. 2.3.6, we have a variety of choices to make it easily invertible.

In practice, we employ various forms of bounding potentials. In section 3.4 we will take $c/|\mathbf{r}_{\text{at}}|$ as bounding potential for the Coulomb interaction under periodic boundary conditions. The cell algorithm that we will introduce in section 3.3 is essentially a potential with static piecewise-constant derivatives, estimated before Markov chains. Moreover, a potential with dynamic piecewise-constant derivative by speculation is also possible and is our event-driven solution for the tension force of the SPC/Fw H_2O model.

2.3.3 Sampling

Sampling is an auxiliary but indispensable component of ECMC. It does not affect ECMC's dynamics but is the way for ECMC to output observables of physical interests.

In principle, any observable X can be extracted using the configuration average along the path of ECMC

$$\langle X \rangle \sim \frac{1}{T} \int_0^T X(c(t)) dt \tag{2.3.8}$$

where $c(t)$ is the configuration at time t. This formula contains integral, so it is hard to compute. Empirically we can instead draw samples in constant interval

$$\langle X \rangle \sim \frac{1}{N_s} \sum_{n=1}^{N_s} X(c(nt_s)) \tag{2.3.9}$$

where t_s is the sampling interval.

One may also consider non-constant sampling intervals such as exponential distribution, but no studies have contributed to it. A pitfall for sampling is that one draws samples exactly at the event time to avoid explicit "sampling events" in the program. This approach leads to wrong estimates of physical observables. For example, in hard-disk simulations, two disks touch each other at event times, while the probability of touching disks tends to zero among all possible states.

2.3.4 Direction switching and restarts

Direction switching is necessary to ensure ergodicity, e.g., to mix hard disks in both directions. Its validity can be understood as a constant flow $x \leftrightarrow y$ that will not break the global balance. In higher dimensions, switching schemes have more options, either choosing the next direction randomly or cyclically. We will give a measurement in section 3.6. In principle, the constant probability flow $x \leftrightarrow y$ means that the switching interval should be an exponential random number. In practice, however, a constant time interval is usually employed without bias in the results.

Sometimes we can also set up restart events that drop current active particles and randomly select a new one. This operation can be regarded as a constant flow between each pair of particles. Restarts have been shown to affect dynamics [85, 56, 86, 93] of ECMC though more studies are on the way.

Chapter 3

Event-chain Monte Carlo for long-range interaction

3.1 Long-range system

The Coulomb potential is a fundamental interaction whose nature is strongly influenced by its long tail [18]. The Lennard-Jones potential and inverse-power-law potentials, if no cutoff is introduced, lead to correlations between the system's far-away components. Modern biochemical simulations commonly use the Amber force field involving inter-molecular long-range terms with empirical parameters, without which the resulting statistics fail to reflect the true nature of simulated bodies [18]. Although long-range interactions are crucial in revealing realistic physical phenomena, they remain a bottleneck of large-scale biochemical simulation algorithms.

In this chapter, we focus on long-range systems simulated by ECMC to see if the novel irreversible Monte Carlo can bring breakthroughs. As introduced in section 1.2.2, the state-of-the-art method that splits the Coulomb potential into a short-range core and a relatively smooth long-range tail leads to Ewald's formula [94], with the latter solved by fast Fourier transformation. New methods under the framework of ECMC without resort to Fourier transformation may be more efficient.

Long-range interactions are related to boundary conditions, at least in a computational sense, and the periodic boundary conditions are usually adopted in order to avoid boundary effects. Mathematically, let S^1 be the topological space of a circle, then a periodic plane becomes a torus denoted as $S^1 \times S^1$, illustrated in Fig. 3.1 (a) , and three-dimensional periodic space can be described as $S^1 \times S^1 \times S^1$. Taking Coulomb interaction as an example, the long-range interaction, at the first place, is ill-defined in light of periodicity because no solution exists for Poisson's equation $\nabla^2 U(\mathbf{r}) = \sum c_i \delta(\mathbf{r} - \mathbf{r}_i)$ that determines the potential of a point-charge system if the system is non-neutral (taking the zeroth Fourier mode leads to $0 = \text{const}$). We may take another view by unfolding the periodic system, which implies that every object has an infinite number of images, as shown in Fig. 3.1 (b) . Summing up contributions from all images for two charges $\sum_{\mathbf{n}} 1/|\mathbf{r}_{12} + \mathbf{n}L|$ obviously diverges.

We categorize long-range interactions into the following three classes:

1. Finite-range interactions that have cutoffs beyond which the forces vanish.

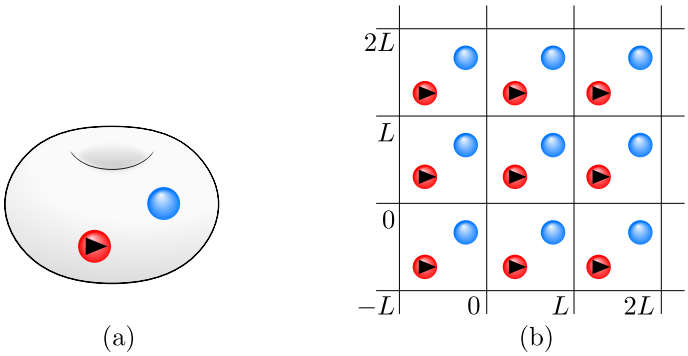


Figure 3.1: Representations of a two-dimensional periodic system. (a) A torus, to which the system is topologically equivalent. (b) Unfolded representation, with repeated images of particles. (Panel (a) is from Ref [1].)

2. Long-range interactions of fast decay, that is, the interaction whose tail scales as $1/r^p$ and $p > d$. The potential computation has no ambiguity since $\sum_{\mathbf{n}} 1/|\mathbf{r} + \mathbf{n}L|^p$ converges absolutely.

3. Long-range interactions of slow decay, that is, the interaction whose tail scales as $1/r^p$ and $p \leq d$.

Assuming that the system is big enough and that interactions exist between all pairs, we cannot enumerate all the interacting pairs to sample the event with reasonable time. In the first case, a cell system is sufficient for fast access of relevant pairs given the active particle. In the second case, a smart scheme is needed to avoid iteration over all the $N-1$ pairs. The cell algorithm [95], which we will introduce in section 3.3, can meet this requirement for homogeneous systems in which density $\rho \approx \text{const}$. In the third case, an ambiguity arises under periodic boundary conditions, and we will focus on one working convention called tin-foil electrostatics and develop its equivalent forms for ECMC.

We will address the formulation issue of slow-decay interaction at first in section 3.2, to derive a self-consistent definition of the two-body Coulomb event rate. We find that ECMC provides various approaches that circumvent Ewald summation for the calculation of pairwise Coulomb interaction. All these methods can be easily adapted to other kinds of long-range interactions too.

The second issue that is treated this chapter is to efficiently sample the next event among all $\mathrm{O}(N)$ factors that involve the active particle. The cell algorithm, together with Walker's method, serves as the mainstay to effectively reduce the complexity from $\mathrm{O}(N)$ to $\mathrm{O}(1)$ for the three-dimensional Coulomb plasma. This algorithm applies to short-range interaction as well and helps avoid iterating over all possible neighbors.

In addition to the Coulomb derivative given in section 3.2, one also needs a bounding potential since its periodic version is no longer invertible. We will give a simple Coulomb bounding potential in section 3.4, taking into account its $1/|\mathbf{r}|$ singularity at the origin.

Next, we will introduce our finding of the dipole factor that enable us to reduce the event rate from $\mathrm{O}(N^{\frac{1}{3}})$ to $\mathrm{O}(\log N)$. Furthermore, two lifting schemes in addition to the ratio lifting scheme for dipole systems will be proposed. The inside-first lifting scheme yields an inter-dipole lifing rate that scales as $1/r^4$. This fully exploits the $1/r^4$ dipole-dipole interaction within ECMC's capacity. Realistic systems mostly comprise or can be decomposed of dipoles, so the invention of the dipole factor will hopefully accelerate biochemical simulations in the future.

Good agreement between different methods such as the Metropolis-Hastings algorithm and ECMC, atomic and dipole factorizations, and various lifting schemes will be presented at the end of this chapter, and in chapter 5, we will provide more verifications of the cell setup.

3.2 Calculation of pairwise Coulomb interaction

3.2.1 Ewald summation

The Coulomb potential of N charges $c_1, c_2, ..., c_N$ in a cubic box under periodic boundary conditions is generally the solution of Poisson's equation $\nabla^2\phi = \sum c_i\delta(\mathbf{r}_i)/\epsilon_0$. However, this simple expression has no solution for non-neutral systems, and the fact that solutions are not unique raises physical unclarity. We will follow the formulation proposed by de Leeuw et al. [94] and will list the results below.

Intuitively, in the unfolded view of periodic boundary conditions, the total energy of the system can be expressed as

$$U_{\mathrm{C}} = \frac{1}{2}\sum_{\mathbf{n}\in\mathbb{Z}^3}\sum_{1\le i,j\le N}\frac{c_i c_j}{|\mathbf{r}_i - \mathbf{r}_j + \mathbf{n}L|}, \tag{3.2.1}$$

where L is the box side length, and the summation should exclude self-interaction of the same image. In this expression, different summing orders lead to distinct results, so it does not converge absolutely. de Leeuw et al. showed that summing for spherical shells results in

$$U_{\mathrm{C}} = \sum_{1\le i<j\le N}\frac{1}{L}c_i c_j\psi\left(\frac{\mathbf{r}_{ij}}{L}\right) + \frac{2\epsilon\pi}{3L^3}|\mathbf{P}|^2 + U_{\mathrm{self}}(\alpha), \tag{3.2.2}$$

in which

$$\psi(\mathbf{r}) = \sum_{\mathbf{n}}\frac{\mathrm{erfc}(\alpha|\mathbf{r}+\mathbf{n}L|)}{|\mathbf{r}+\mathbf{n}L|} + \frac{1}{\pi}\sum_{\mathbf{m}\in\mathbb{Z}^3,\mathbf{m}\neq\mathbf{0}}\frac{e^{2\pi i\mathbf{m}\cdot\mathbf{r}-\pi^2|\mathbf{m}|^2/\alpha^2}}{|\mathbf{m}|^2}, \tag{3.2.3}$$

with erfc the complementary error function

$$\mathrm{erfc}(x) = 1 - 2\pi^{-\frac{1}{2}}\int_0^x e^{-t^2}\mathrm{d}t, \tag{3.2.4}$$

U_{self} a term of self energy independent of $\mathbf{r}_i$, and $\mathbf{P}$ the system's polarization

$$\mathbf{P} = \sum_{1\le i\le N} c_i\mathbf{r}_i. \tag{3.2.5}$$

In Eq. 3.2.2, ϵ is the dielectric constant that originates from summing order. The value $\epsilon = 1$ corresponds to summing over spherical shells. To avoid many-body Coulomb interaction we adopt the so-called tin-foil convention where $\epsilon = 0$ so that the polarization term vanishes. Then the energy can be decomposed as a sum of pair energies

$$\begin{aligned} U_{\mathrm{C}} = \sum_{1\le i<j\le N} c_i c_j \Bigg[& \sum_{\mathbf{n}\in\mathbb{Z}^3} \frac{\operatorname{erfc}(\alpha|\mathbf{r}_{ij}+\mathbf{n}L|)}{|\mathbf{r}_{ij}+\mathbf{n}L|} \\ & + \frac{4\pi}{L^3} \sum_{\mathbf{q}\neq(0,0,0)} \frac{e^{-\mathbf{q}^2/(4\alpha^2)}}{\mathbf{q}^2} \cos(\mathbf{q}\cdot\mathbf{r}_{ij}) \Bigg] + U_{\mathrm{self}}(\alpha). \end{aligned} \tag{3.2.6}$$

where $\mathbf{q} = 2\pi\mathbf{m}/L$ with $\mathbf{m} \in \mathbb{Z}^3$. Here, α is a parameter that does not affect the result but influences the convergence speed. Considering that the complementary error function around infinity behaves as $\operatorname{erfc}(x) \sim e^{-x^2}/(\sqrt{\pi}x) > e^{-x^2}/\sqrt{\pi}$ we get bounds of $\mathbf{n}$ and $\mathbf{m}$

$$\begin{aligned} |\mathbf{n}| &\sim \sqrt{-\log(\sqrt{\pi}\delta)}/(\alpha L), \\ |\mathbf{m}| &\sim \sqrt{-\log(\pi L\delta)}\alpha L/\pi, \end{aligned} \tag{3.2.7}$$

that are sufficient to reach precision δ. In practice, α is chosen so that the real-space part, i.e., the term with the complementary error function, converges faster than the Fourier part, which can reduce the costly computation of the error function.

To get the event rate of a single moving particle in ECMC, we just take the derivative of Eq. 3.2.6, and the result comprises individual pairs, each consisting of a Fourier-space term and a real-space term

$$\tilde{q}_{\mathrm{Clb}}(\mathbf{r}_{12} = \mathbf{r}_2 - \mathbf{r}_1) = \tilde{q}_{\mathrm{real}}(\mathbf{r}_{12}) + \tilde{q}_{\mathrm{Four}}(\mathbf{r}_{12}), \tag{3.2.8}$$

with

$$\tilde{q}_{\mathrm{real}}(\mathbf{r}_{12}) = c_1 c_2 \sum_{\mathbf{n}\in\mathbb{Z}^3} \frac{x_{12}+n_x L}{|\mathbf{r}_{12}+\mathbf{n}L|^2} \left[\frac{\operatorname{erfc}(\alpha|\mathbf{r}_{12}+\mathbf{n}L|)}{|\mathbf{r}_{12}+\mathbf{n}L|} + \frac{2\alpha}{\sqrt{\pi}} e^{-\alpha^2|\mathbf{r}_{12}+\mathbf{n}L|^2} \right] \tag{3.2.9}$$

and

$$\tilde{q}_{\mathrm{Four}}(\mathbf{r}_{12}) = c_1 c_2 \frac{4\pi}{L^3} \sum_{\mathbf{q}\neq 0} q_x \frac{e^{-\mathbf{q}^2/(4\alpha^2)}}{\mathbf{q}^2} \sin(\mathbf{q}\cdot\mathbf{r}_{12}) \tag{3.2.10}$$

in which we assume that the particle 1 is active. Fig. 3.2 shows the values on four z profiles $q_{\mathrm{Clb}} = \beta\,[\tilde{q}_{\mathrm{Clb}}]^+$.

In the simulation of a number of charges, we find by profiling the program that direct calculation of the Coulomb derivative by Eqs. 3.2.9 and 3.2.10 takes the majority of total time (>80%). In fact, over a hundred terms must be taken into account to achieve the error level of double-precision float (2^{-53}). However, there are tricks to speed up the computation:

1. Using spherical cutoffs for both real-space and Fourier-space terms, instead of cubic cutoffs.

2. Tuning the parameter α to include more Fourier-space terms than real-space terms, to reduce calls of the error function.

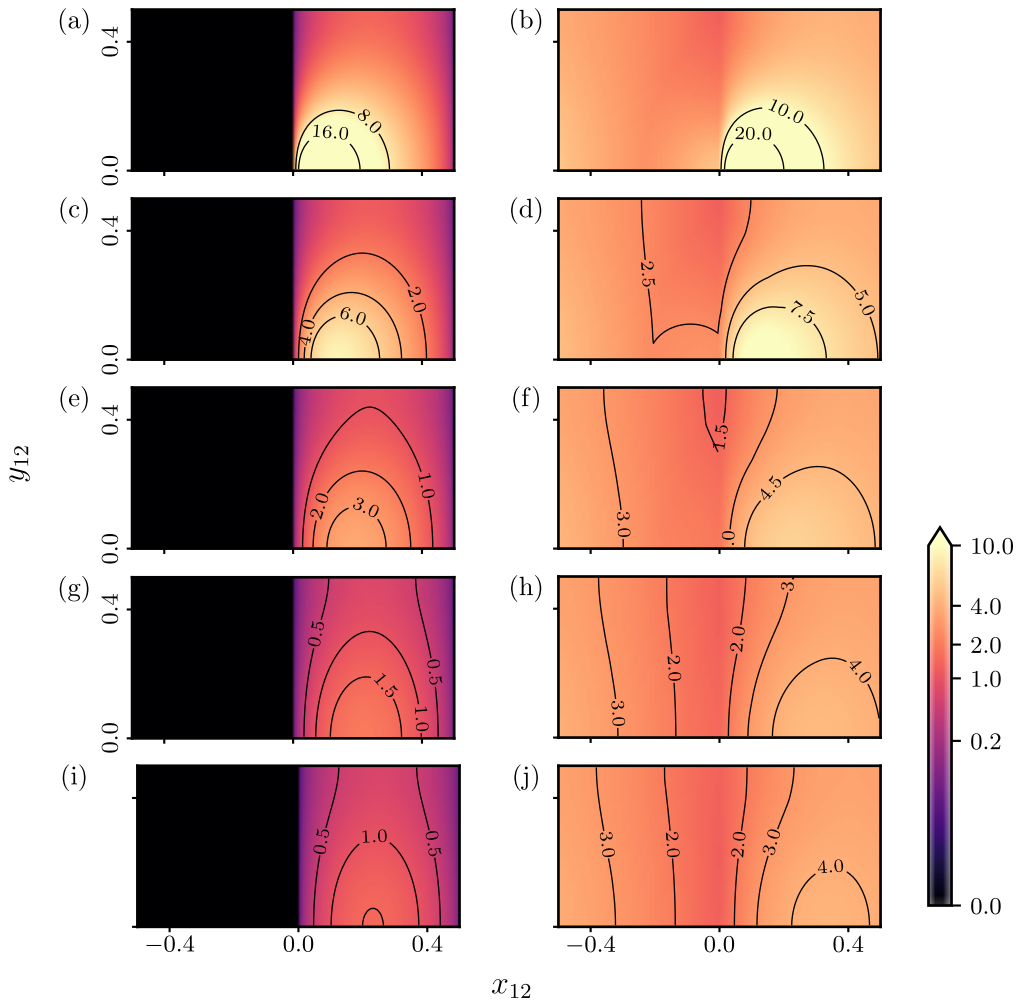


Figure 3.2: Pairwise Coulomb event rates $q_{\mathrm{Clb}}(\mathbf{r} = (x_{12}, y_{12}, z_{12}))$ at five different z sections for $\beta = L = c_1 = c_2 = 1$, with (a)(b)$z_{12} = 0.1$, (c)(d)$z_{12} = 0.2$, (e)(f)$z_{12} = 0.3$, (g)(h)$z_{12} = 0.4$, and (i)(j)$z_{12} = 0.5$. Subfigures on the left are event rates with the merged-image approach given by Eqs. 3.2.9 and 3.2.10, while those on the right are for separate-image event rates by Eq. 3.2.37. (From Ref [1].)

3. Exploiting the positive-negative symmetry of Fourier-space terms with respect to components of $\mathbf{q}$. In fact, only roughly $1/8$ of all terms have to be calculated.

4. Pre-calculating sin and cos of $q_x r_{12,x}$, $q_y r_{12,y}$, and $q_z r_{12,z}$, then $\sin(\mathbf{q}\cdot\mathbf{r}_{12})$ can be calculated in an iterative manner.

After taking into account the tricks above, the Coulomb derivative computation takes roughly 30% of the time (run by JELLYFYSH, see chapter 4). Actually, as a stochastic algorithm, ECMC has the potential to circumvent all detailed calculations while keeping the results unbiased, which is still under development.

3.2.2 Line-charge method

3.2.2.1 Construct

One may by intuition write down the two-body Coulomb derivative as the sum of image derivatives

$$\tilde{q}_{\text{Clb}} = c_1 c_2 \sum_{\mathbf{n}} \frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{12}|} = c_1 c_2 \sum_{\mathbf{n}} \frac{x_{12}}{|\mathbf{r}_{12}|^3}. \tag{3.2.11}$$

Again, this summation suffers from poor convergence even though it indeed converges to some true charge-charge derivatives with a certain dielectric constant depending on the summing order. For a faraway image $R = |\mathbf{n}L|$, the derivative decays as R^{-2}. The expression does not converge absolutely since $\int^{\infty} r^{-2} \mathrm{d}^3 r = \infty$.

The line-charge model overcomes this divergence by bundling a charged line of length pL image, laid along the direction of motion of the active particle with the target image in the center. Every line is charged evenly with a density $-c_2/pL$. The composite of the target image and the accompanying line charge, labeled as $\mathbf{n}$, will contribute to the derivative

$$\begin{aligned} &\tilde{q}_{\text{Clb,line}}\left(\mathbf{r} = \mathbf{r}_{12} + \mathbf{n}L = (x,y,z)\right) \\ =&c_1 c_2 \left[\frac{x}{|\mathbf{r}|^3} + \frac{1}{pL}\left(\frac{1}{|\mathbf{r}+pL\hat{\mathbf{e}}_x/2|} - \frac{1}{|\mathbf{r}-pL\hat{\mathbf{e}}_x/2|}\right)\right]. \end{aligned} \tag{3.2.12}$$

This formula results in identical statistics as Eqs. 3.2.9 and 3.2.10, as shown in Fig. 3.3 (a) . The agreement is intuitive whereas rigorous proof remains non-obvious. We will justify the equivalence by introducing a homogeneous volume-charge model.

Eq. 3.2.12 is computationally powerful in two respects. First, it has vanishing charge and dipole moments, and the leading term behaves as a charge-quadrupole interaction and thus scales as R^{-4}. Moreover, $\tilde{q}_{\text{Clb,line}}(\mathbf{r}_{12},\mathbf{n}) + \tilde{q}_{\text{Clb,line}}(\mathbf{r}_{12},-\mathbf{n})$ decays even faster at the speed of R^{-5}. When summing through cubic or spherical shells,

$$\int^{R} \tilde{q}_{\text{Clb,line}}(\mathbf{r}_{12},\mathbf{n})\mathrm{d}\mathbf{n} \sim \int^{R} r^{-3}\mathrm{d}r \sim U - R^{-2}, \tag{3.2.13}$$

argeeing with our numerical test shown in Fig. 3.3 (a) .

Second, Eq. 3.2.12 needs no explicit calculations of the line-charge contribution, that is, of the second and third terms in Eq. 3.2.12 for each image. Only those at the

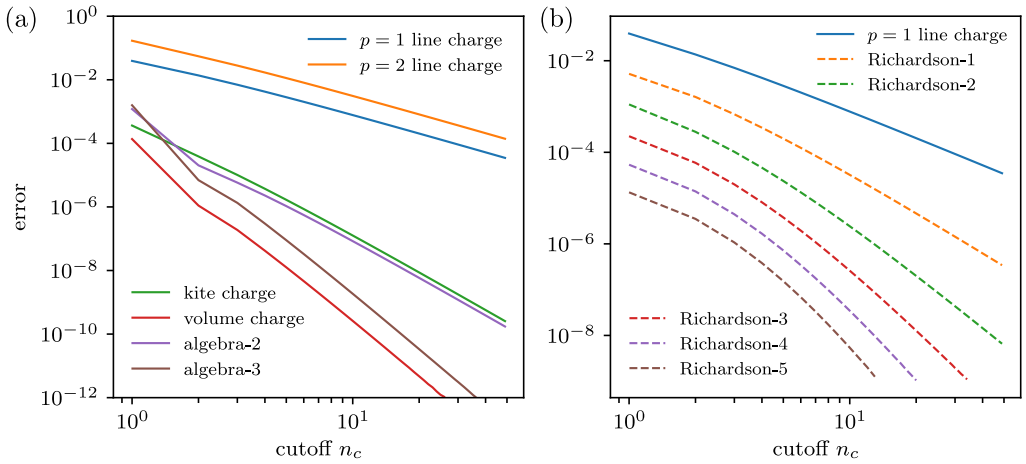


Figure 3.3: Convergence comparison in logarithmic scale between results of various methods for pairwise Coulomb derivatives at $\mathbf{r}_{12} = (1/3, 1/6, 1/7)$. (a) Line-charge model with $p = 1$ and $p = 2$, kite-charge model, volume-charge model, and algebraic compensation. The curve "algebra-2" corresponds to using T_1 of Eq. 3.2.33 as the compensating term, and "algebra-3" corresponds to T_3 of Eq. 3.2.33. (b) Richardson's method starting with $p = 1$ line-charge model. The original series is with respect to increasing cutoffs.

end of "concatenated" lines survive. This effect significantly reduces the computation work of the line-charge model, with only a few additional terms on the boundary.

For the choice of the line charge length p in Eq. 3.2.12, any positive integer is acceptable while $2p(2n+1)^2$ ends of lines are unable to cancel. In Fig. 3.4 (a) and (b) we show unfolded images of the target particle compensated by L and $2L$ line charges respectively. Usually, $p = 1$ is the ideal choice if we sum up Eq. 3.2.12. This way, we put all images as one factor, so it is called "merged-image" factorization. However, if we factorize images and treat each image accompanied by its screening charge as an individual factor, singularities appear at the image charge position and ends of the line charges. In this case, $p = 1$ suffers a non-physical singularity at $\mathbf{r}_{12} = (L/2, 0, 0)$ besides the physical one at $(0, 0, 0)$. Hence in the separate-image approach, we propose $p = 2$ so that these two singularities coincide.

3.2.2.2 Equivalence to tin-foil convention

Here we demonstrate the equivalence between Ewald's method with the tin-foil convention and the line-charge method. We will show that the line-charge method has the identical result as a homogeneous volume-charge method. The equivalence to the polarization-free Ewald's summation is because the net target charge vanishes in the volume-charge method, whereas we will not cover this part of the argument here.

For a static charge with structure factor $\rho_2(\mathbf{q})$, a test charge c_1 has a potential

$$U(\mathbf{r}_{12}) = 4\pi c_1 \int_{-\infty}^{\infty} \frac{\mathrm{d}^3\mathbf{q}}{(2\pi)^3} e^{i\mathbf{q}\cdot\mathbf{r}_{12}} \frac{\rho_2(\mathbf{q})}{|\mathbf{q}|^2}. \tag{3.2.14}$$

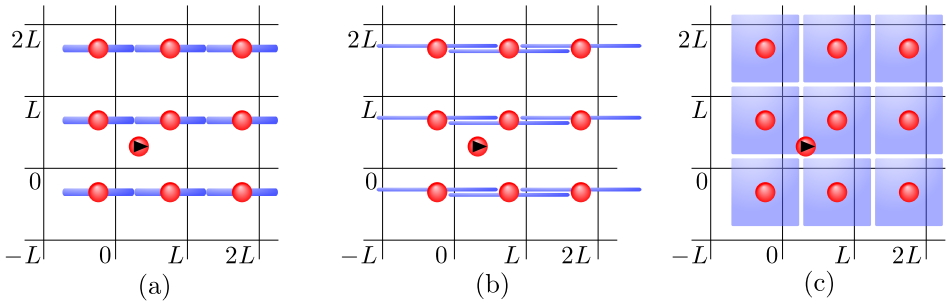


Figure 3.4: Screening charge models. (a) Line-charge model with $p = 1$. (b) Line-charge model with $p = 2$. (c) Sheet-charge model. (Panels (b) and (c) are from Ref [1].)

Summing up $(2K+1)^3$ images, we use the following formula

$$D_K(q_x) = \sum_{l=-K}^{K} e^{iq_x lL} = \frac{\sin\left[q_x L(K+1/2)\right]}{\sin(q_x L/2)}, \tag{3.2.15}$$

and we denote $\tilde{D}_K(\mathbf{q}) = D_K(q_x)D_K(q_y)D_K(q_z)$. Taking into account $(2K+1)^3$ images and in the limit of $K \to \infty$, we reach

$$D_K(q_x) \xrightarrow{K\to\infty} \frac{2\pi}{L} \sum_{m=-\infty}^{\infty} \delta\left(q_x - m\frac{2\pi}{L}\right). \tag{3.2.16}$$

For a point charge c with line-screening charge of length pL

$$\rho_{\text{line}} = c(1 - \operatorname{sinc}\frac{pq_x L}{2}), \tag{3.2.17}$$

and for one with volume-screening charge of side length L

$$\rho_{\text{volume}} = c(1 - \operatorname{sinc}\frac{q_x L}{2}\operatorname{sinc}\frac{q_y L}{2}\operatorname{sinc}\frac{q_z L}{2}) \tag{3.2.18}$$

where $\operatorname{sinc}(x) = \sin(x)/x$.

By taking their difference, we get

$$\begin{aligned}\Delta U &= U_{\text{line}} - U_{\text{volume}} \\ &= c_1 c_2 \int \frac{\mathrm{d}^3\mathbf{q}}{2\pi^2} D_K(\mathbf{q}) \frac{e^{i\mathbf{q}\cdot\mathbf{r}_{12}}}{|\mathbf{q}|^2} \left(\operatorname{sinc}\frac{q_x L}{2}\operatorname{sinc}\frac{q_y L}{2}\operatorname{sinc}\frac{q_z L}{2} - \operatorname{sinc}\frac{pq_x L}{2}\right).\end{aligned} \tag{3.2.19}$$

Eq. 3.2.16 tells us that $D_K(\mathbf{q})$ peaks around $q_i = 2\pi m/L, m \in \mathbb{Z}$, while $\operatorname{sinc}(q_i L/2)$ is zero at those values except for $q_i = 0$. A short analysis suggests that non-zero terms of Eq. 3.2.19 come from $q_x = 0$ (the $q_x = q_y = q_z = 0$ case needs special analysis). The line-charge and volume-charge models have different potentials. However, taking the derivative with respect to x_1 yields

$$\frac{\partial \Delta U}{\partial x_1} = c_1 c_2 \int \frac{\mathrm{d}^3\mathbf{q}}{2\pi^2} D_K(\mathbf{q}) \frac{q_x \sin(\mathbf{q}\cdot\mathbf{r}_{12})}{|\mathbf{q}|^2} \left(\operatorname{sinc}\frac{q_x L}{2}\operatorname{sinc}\frac{q_y L}{2}\operatorname{sinc}\frac{q_z L}{2} - \operatorname{sinc}\frac{pq_x L}{2}\right). \tag{3.2.20}$$

Due to an additional q_x, the integer points $q_x = 0$, $q_y q_z \neq 0$ vanish as well. At around $q_x = q_y = q_z = 0$ we expand the trigonometric functions and get

$$\frac{\partial \Delta U}{\partial x_1} \xrightarrow{\mathbf{q}\to 0} \frac{q_x^2 \left[(p^2-1)q_x^2 - q_y^2 - q_z^2\right]}{|\mathbf{q}|^2} D_K(\mathbf{q}) \sim 0. \tag{3.2.21}$$

Therefore, the contribution around $\mathbf{q} = 0$ is zero for large K too.

In summary, the above derivation demonstrates the equivalence between the line-charge method and the volume-charge method in the limit of infinite images.

3.2.3 Higher-order methods

The success of the line-charge method provides an alternative for calculation of the pairwise Coulomb derivative. As discussed in section 3.2.2.1, the remainder of counting $(2n+1)^3$ images decays as n^{-5}. To reach double-float precision (2^{-53}), we have to calculate $\sim 2^{30}$ terms, far beyond the allowed limit of any molecular simulator. Hence, to reach relatively high precision, the line-charge method is outperformed by Ewald's method. Nevertheless, algorithms exist that can converge faster than the compensating line charge besides Ewald's sum, and many of them inherit the idea of screening charge.

The following methods that we develop to compute Coulomb potential derivatives have actually exceeded the current need in the program. In JELLYFYSH introduced in chapter 4 and its applications, the Ewald summation is sufficient. It has the best precision with acceptable time consumption. We propose the following higher-order methods as complements that may be advantageous in other situations.

3.2.3.1 Volume-charge method

During the proof that the line-charge method Eq. 3.2.12 is equivalent to tin-foil electrostatics of Eq. 3.2.6, we introduce a volume-charge model illustrated in Fig. 3.5 (a) , which actually converges two-order faster than line-charge model due to its vanishing quadrupole and octupole moments.

Similar to Eq. 3.2.12, the volume-charge model for two charges c_1 and c_2 with c_2 surrounded by a homogeneous cubic compensating charge of side length L, has the following derivative

$$\begin{aligned}\tilde{q}_{\mathrm{Clb,volume}}(\mathbf{r}_{12}, \mathbf{n}) = c_1 c_2 \Bigg\{ & \frac{x_{12} + n_x L}{|\mathbf{r}_{12} + \mathbf{n}L|^3} \\ & + \frac{1}{L^3}\left[I_1(x_{12} + L/2, y_{12}, z_{12}, L) - I_1(x_{12} - L/2, y_{12}, z_{12}, L)\right] \Bigg\},\end{aligned} \tag{3.2.22}$$

where the integration is defined as

$$\begin{aligned} I_1(x_0, y_0, z_0, L) &= \int_{y_0 - L/2}^{y_0 + L/2} \mathrm{d}y \int_{z_0 - L/2}^{z_0 + L/2} \mathrm{d}z \frac{1}{\sqrt{x_0^2 + y^2 + z^2}} \\ &= \sum_{s_y, s_z \in \{+1, -1\}} s_y s_z I_2(x_0, y_0 + s_y L/2, z_0 + s_z L/2). \end{aligned} \tag{3.2.23}$$

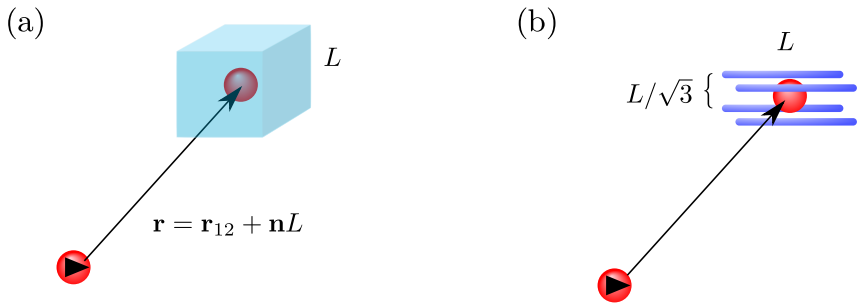


Figure 3.5: Two constructs of compensating charge with vanishing quadrupole moment. (a) Volume-charge model. (b) "Kite" model with each line of length L. The target image is in the center of the kite, and four legs of the kite make up a square of side length $L/\sqrt{3}$.

Explicitly integrating I_1, we have

$$\begin{aligned} I_2(x,y,z) =& y\log\left(\sqrt{x^2+y^2+z^2}+z\right) + z\log\left(\sqrt{x^2+y^2+z^2}+y\right) \\ & - x\arctan\left(\frac{yz}{x\sqrt{x^2+y^2+z^2}}\right) + x\arctan\left(\frac{y}{x}\right) - y. \end{aligned} \tag{3.2.24}$$

The entire expression consists of as many as 8×5 terms before simplification, and many of them contain costly logarithm and inverse trigonometric functions. It is thus difficult to evaluate, even in the case that compensating terms cancel for neighboring images. Thus we seek other methods to construct screening charges with zero quadrupole moments.

3.2.3.2 Kite-charge method

The volume-charge model is not the unique construct with a vanishing quadrupole moment. More than one compensating line charge can be associated with one image, giving more freedom of the screening charge construct. In fact, a tricky arrangement of four line charges can cancel the quadrupole moment.

According to the definition of the quadrupole moment of a single charge

$$Q_{ij} = q(3r_i r_j - |\mathbf{r}|^2\delta_{ij}), \tag{3.2.25}$$

a line of charge density $1/L$ extending from $(-L/2,y,z)$ to $(L/2,y,z)$ carries a quadrupole moment

$$Q = \begin{bmatrix} L^2/6-y^2-z^2 & 0 & 0 \\ 0 & -L^2/12+2y^2-z^2 & 3yz \\ 0 & 3yz & -L^2/12-y^2+2z^2 \end{bmatrix}. \tag{3.2.26}$$

It is easy to let the diagonal terms vanish by making $y^2 = z^2 = L^2/12$. To cancel the off-diagonal terms $3yz$ we should have more than one line. Considering symmetries required to cancel the dipole moment, we conclude that we need a minimum of four lines evenly distributed on the $y^2+z^2 = L/6$ circle. One possibility is

$(y,z)=(s_yL/\sqrt{12},s_zL/\sqrt{12})$ for $s_y,s_z\in\{+1,-1\}$, meaning that four lines of length L center around the target particle image pointing along $+x$ (hence the name "kite") as shown in Fig. 3.5 (b) . And furthermore, the composite remains invariant under the rotation along x axis, thus the simplest solution that we found is

$$\begin{aligned}(y_1,z_1)=(L/\sqrt{6},0),\quad (y_2,z_2)=(0,L/\sqrt{6}),\\ (y_3,z_3)=(-L/\sqrt{6},0),\quad (y_4,z_4)=(0,-L/\sqrt{6}),\end{aligned}\tag{3.2.27}$$

and the corresponding imagewise derivative becomes

$$\begin{aligned}&\tilde{q}_{\mathrm{Clb,kite}}\left(\mathbf{r}=\mathbf{r}_{12}+\mathbf{n}L=(x,y,z)\right)\\ =&c_1c_2\left[\frac{x}{|\mathbf{r}|^3}+\sum_{(y,z)\in\mathcal{K}}\frac{1}{4L}\left(\frac{1}{|\mathbf{r}+L\hat{\mathbf{e}}_x/2+y\hat{\mathbf{e}}_y/2+z\hat{\mathbf{e}}_z/2|}\right.\right.\\ &\left.\left.-\frac{1}{|\mathbf{r}-L\hat{\mathbf{e}}_x/2+y\hat{\mathbf{e}}_y/2+z\hat{\mathbf{e}}_z/2|}\right)\right],\end{aligned}\tag{3.2.28}$$

where $\mathcal{K}$ is the set whose elements are listed in Eq. 3.2.27. A kite of length L is given above while the generalization to $pL(p\in\mathbb{Z}^+)$ is straightforward. Fig. 3.3 (a) justifies its two-order faster decay than for the line-charge method with increasing n, in which $(2n+1)^3$ images are evaluated.

3.2.3.3 Richardson acceleration

Richardson's method [96] is a general algorithm to accelerate the convergence of a sequence with polynomial remainder. Assume that the sum S_n of the first n terms of a series has

$$S_n=S_\infty+\frac{C}{n^p}+o(n^{-p}),\quad p>0.\tag{3.2.29}$$

The constructed series,

$$S'_n=\frac{(n+1)^pS_{n+1}-n^pS_n}{(n+1)^p-n^p},\tag{3.2.30}$$

can eliminate the n^{-p} term, preserve the S_∞ limit, and thus converge at least as fast as n^{-p-1}. Iterating this process can create a series that converges at arbitrary order.

We let S_n be the sum of derivative contributions of $(2n+1)^3$ line-charge composites and then perform Richardson's acceleration, as shown in Fig. 3.3 (b) . The relative magnitudes of n and $S_n^{(m)}$ agree with the expectation that the m-th order Richardson's method applied to the line-charge model can converge at the speed of n^{-m-2}.

3.2.3.4 Algebraic compensation

All the methods mentioned above aim to accelerate the convergence of the summation of a sequence. All kinds of screening charges are derived from physical settings, but it may imply a more general series-acceleration algorithm from a mathematical point of view.

In the screening charge model of line shape or kite shape, the terms resulted from screening charges both take the form of the Coulomb potential $1/|\mathbf{r}|$, which can be viewed as an indefinite integral of the summed function (here the target function is $\partial_x(1/|\mathbf{r}|)$, and the integral is $1/|\mathbf{r}|$ which we called line charge). We find that taking derivatives of summed functions instead of integrals as compensating terms is capable of obtaining a series of faster convergence.

Assume that we hope to calculate $\sum_{p=1}^{\infty} f(p)$, and let $F(x) = \int f(x)$ and $f^{(n)}(x) = \mathrm{d}^n f(x)/\mathrm{d}x^n$ the nth order derivative. Their Taylor expansions are given by

$$\begin{aligned} f(x+a) &= \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(x) a^n, \\ F(x+a) &= \sum_{n=0}^{\infty} \frac{1}{(n+1)!} f^{(n)}(x) a^{n+1}, \\ f^{(m)}(x+a) &= \sum_{n=m}^{\infty} \frac{1}{(n-m)!} f^{(n)}(x) a^{n-m}. \end{aligned} \tag{3.2.31}$$

We expect that a combination of $F(x \pm 1/2)$ and $f^{(n)}(x \pm 1/2)$ is able to approximate $f(x)$ to arbitrary order. First, with inspiration of the line-charge model we get

$$F(x-1/2) + f(x) - F(x+1/2) = \mathrm{O}(f^{(2)}(x)), \tag{3.2.32}$$

with the assumption that $f^{(n)}$ decays faster for larger n, which is true for the Coulomb interaction. Next, we let

$$\begin{aligned} T_1(x) =& F(x) - \frac{1}{24} f^{(1)}(x), \\ T_3(x) =& F(x) - \frac{1}{24} f^{(1)}(x) + \frac{7}{5760} f^{(3)}(x), \\ T_5(x) =& F(x) - \frac{1}{24} f^{(1)}(x) + \frac{7}{5760} f^{(3)}(x) - \frac{31}{967680} f^{(5)}(x), \end{aligned} \tag{3.2.33}$$

then

$$\begin{aligned} T_1(x-1/2) + f(x) - T_1(x+1/2) &= \mathrm{O}(f^{(4)}(x)), \\ T_3(x-1/2) + f(x) - T_3(x+1/2) &= \mathrm{O}(f^{(6)}(x)), \\ T_5(x-1/2) + f(x) - T_5(x+1/2) &= \mathrm{O}(f^{(8)}(x)). \end{aligned} \tag{3.2.34}$$

Fig. 3.3 (a) verifies the faster convergence of algebraic constructs T_n in an application of three-dimensional periodic Coulomb system. Similar to the line-charge method, compensating terms T_n cancel for neighboring images. Only $T_n(\mathbf{r}_{12} + (\mathbf{n}_{+x} + \hat{\mathbf{e}}_x)L)$ and $T_n(\mathbf{r}_{12} - (\mathbf{n}_{-x} + \hat{\mathbf{e}}_x)L)$ survive. Although terms with higher-order derivatives of $1/|\mathbf{r}|$ decay faster, it introduces more complexity in the computation of their derivatives, which becomes the bottleneck of this algebraic approach. In fact, $F(x)$ and $f^{(n)}(x)$ at $x + p/2$ for $p \in \mathbb{Z}$ and $p > 1$ can also help eliminate slow decaying terms, and $\partial_y f$ and $\partial_z f$ are candidates for three-dimensional function too. A systematic investigation of this method is still missing.

3.2.4 Interpolation

Interpolation is a universal approximate method applicable to all sufficiently smooth functions. It is also feasible for evaluating the Coulomb derivative of Eq. 3.2.8. It is noteworthy that the Coulomb derivative with respect to x direction, contains an $x/|\mathbf{r}|^3$ singularity at the origin. Instead of interpolating $\tilde{q}_{\mathrm{Clb}}$ we approximate the following functions

$$\tilde{q}_{\mathrm{f},1}(\mathbf{r}_{12}) = \tilde{q}_{\mathrm{Clb}} \frac{|\mathbf{r}_{12}|^3}{x_{12}} \tag{3.2.35}$$

and

$$\tilde{q}_{\mathrm{f},2}(\mathbf{r}_{12}) = \tilde{q}_{\mathrm{Clb}} - \frac{x_{12}}{|\mathbf{r}_{12}|^3}, \tag{3.2.36}$$

where $\mathbf{r}_{12} \in [-L/2, L/2]^3$ is the separation of their nearest images. $\tilde{q}_{\mathrm{f},2}$ has branch cuts at the surfaces $x = \pm L/2$ since x_{12} changes its sign when passing from $\pm L/2 - \epsilon$ to $\pm L/2 + \epsilon$. Nevertheless, the singularity at the origin disappears, and $\tilde{q}_{\mathrm{f},1}$ and $\tilde{q}_{\mathrm{f},2}$ are both continuous for $\mathbf{r}_{12} \in [-L/2, L/2]^3$. Fig. 3.6 shows their values at six z-sections.

No evidence shows a substantial difference in the qualities of using $\tilde{q}_{\mathrm{f},1}$ or $\tilde{q}_{\mathrm{f},2}$, so $\tilde{q}_{\mathrm{f},1}$ is chosen in our event-driven implementation. We will not discuss various interpolation methods but take first-order and third-order Lagrange polynomial interpolations for testing purposes. They are also known as linear and cubic interpolations, respectively, and the generalization to three dimensions is straightforward. We compare their resulting errors in Fig. 3.7.

3.2.5 Separate-image method

As discussed in section 2.2.3, ECMC allows for different ways of grouping interacting terms, which presents no physical effects but leads to distinct algorithmic complexities and Markov-chain dynamics. All the methods mentioned above yield converging expressions for $\tilde{q}_{\mathrm{Clb}}$ of Eq. 3.2.8, but all regard the pair energy as one factor in ECMC. Terms of potential can be grouped into more than one factor. We will describe below the example of splitting Eq. 3.2.12.

In the separate-image approach, we do not sum up Eq. 3.2.12 to obtain $\tilde{q}_{\mathrm{Clb}}$ but regard each image represented by $\mathbf{n}$ as an individual factor that is able to trigger events with the rate

$$\begin{aligned} & q_{\mathrm{Clb,line}}\left(\mathbf{r} = \mathbf{r}_{12} + \mathbf{n}L = (x,y,z)\right) \\ =& \beta \left[c_1 c_2 \left\{ \frac{x}{|\mathbf{r}|^3} + \frac{1}{pL} \left[\frac{1}{|\mathbf{r} + pL\hat{\mathbf{e}}_x/2)|} - \frac{1}{|\mathbf{r} - pL\hat{\mathbf{e}}_x/2)|} \right] \right\} \right]^+ \\ =& \beta \left[\tilde{q}_{\mathrm{Clb,line}}\left(\mathbf{r}\right)\right]^+ . \end{aligned} \tag{3.2.37}$$

In the naive time-driven approach, each time the active particle proposes a tiny move ϵ, the program iterates over all images of the other particle (perhaps with a cutoff for images), calculates $q_{\mathrm{Clb,line}}$ for each image and checks whether an event happens within ϵ. Similarly to section 2.2.3, the displacement up to the next event in event-driven description is chosen from the minimum of all displacements given by each image factor. We verify imagewise factorization in Fig. 3.8 in a simple two-atom system.

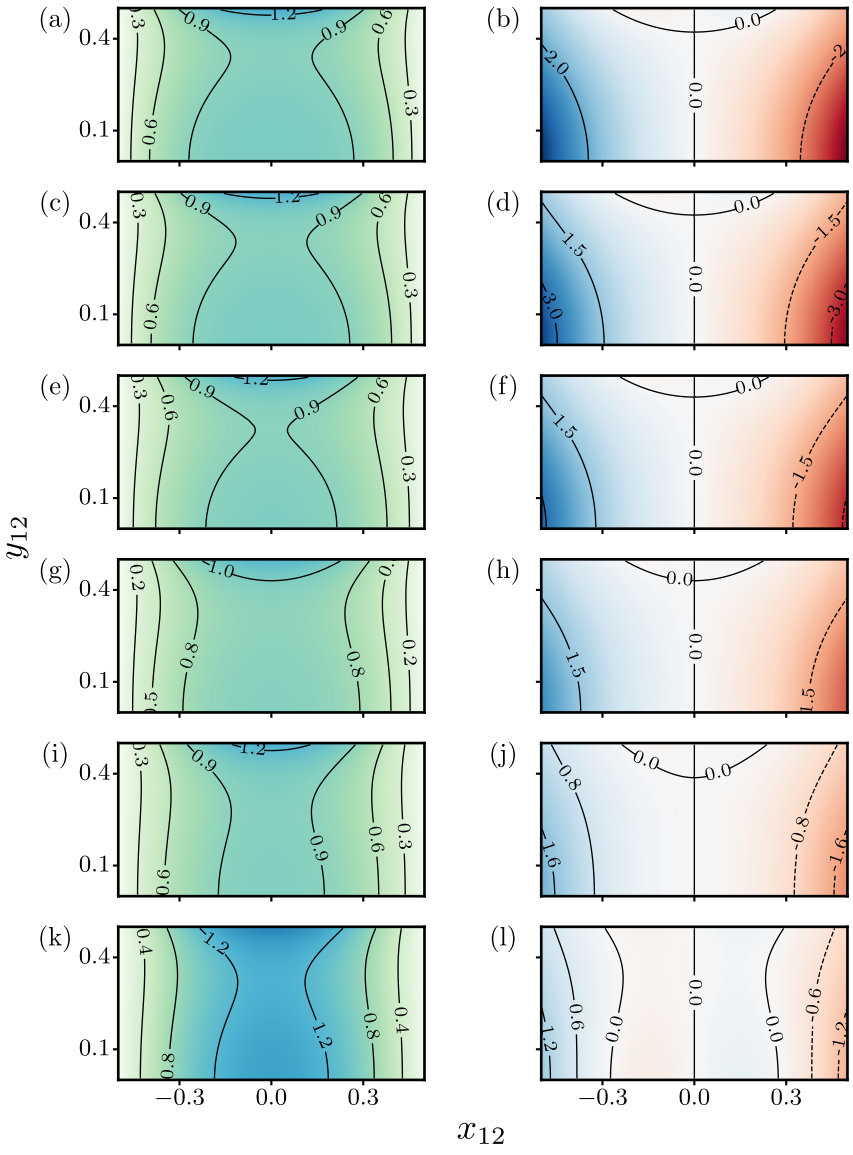


Figure 3.6: $\tilde{q}_{\mathrm{f},1}$ (left column) and $\tilde{q}_{\mathrm{f},2}$ (right column) at six z sections, with each row one z value. They are for $z = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5$ from top row to bottom row.

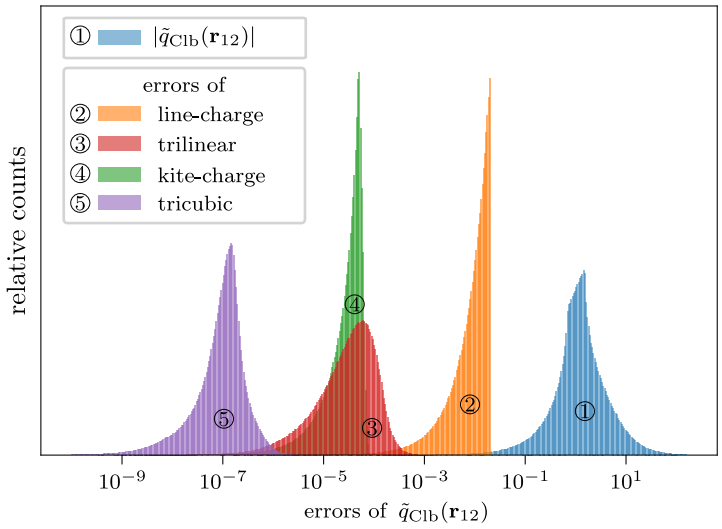


Figure 3.7: Distributions of $|\tilde{q}_{\mathrm{Clb}}|$ values and errors of several methods. *Blue*: reference values by Ewald's sum with $\alpha = 3.45$, real-space cutoff 2 and Fourier-space cutoff 6. *Orange*: Line-charge method with 5^3 images. *Green*: Kite-charge method with 5^3 images. *Red*: Trilinear interpolation using 100^3 grids. *Purple*: Tricubic interpolation using 100^3 grids. Results are gained through direct sampling, and separation $\mathbf{r}_{12}$ are drawn evenly from the simulation box.

In general, every formula calculating the pairwise Coulomb derivative has corresponding "term-wise" factorization, and how it facilitates the simulation depends on the difficulty of inverting each term, the efficiency of sampling remaining terms, and on whether loop flows can be avoided.

The analysis in section 2.2.3.2 tells us that factorization is equivalent to introducing extra probability flows which are particularly manifest in the separate-image method. For the line-charge model, we need to demonstrate the following

$$\begin{aligned}&\sum_{\mathbf{n}} q_{\mathrm{Clb,line}}\left(\mathbf{r}_{12}+\mathbf{n}L\right)-\sum_{\mathbf{n}} q_{\mathrm{Clb,line}}\left(\mathbf{r}_{21}+\mathbf{n}L\right)\\ =&\beta\tilde{q}_{\mathrm{Clb}}(\mathbf{r}_{12})=\beta\sum_{\mathbf{n}}\tilde{q}_{\mathrm{Clb,line}}(\mathbf{r}_{12}+\mathbf{n}L).\end{aligned}\tag{3.2.38}$$

In fact, the left-hand side of this equation can be written as

$$\begin{aligned}&\sum_{\mathbf{n}} q_{\mathrm{Clb,line}}\left(\mathbf{r}_{12}+\mathbf{n}L\right)-\sum_{-\mathbf{n}} q_{\mathrm{Clb,line}}\left(\mathbf{r}_{21}-\mathbf{n}L\right)\\ =&\sum_{\mathbf{n}} q_{\mathrm{Clb,line}}\left(\mathbf{r}_{12}+\mathbf{n}L\right)-\sum_{\mathbf{n}} q_{\mathrm{Clb,line}}\left(-\mathbf{r}_{12}-\mathbf{n}L\right)\\ =&\beta\sum_{\mathbf{n}}\left\{[\tilde{q}_{\mathrm{Clb,line}}(\mathbf{r}_{12}+\mathbf{n}L)]^{+}-[\tilde{q}_{\mathrm{Clb,line}}(-\mathbf{r}_{12}-\mathbf{n}L)]^{+}\right\}\\ =&\beta\sum_{\mathbf{n}}\left\{[\tilde{q}_{\mathrm{Clb,line}}(\mathbf{r}_{12}+\mathbf{n}L)]^{+}-[-\tilde{q}_{\mathrm{Clb,line}}(\mathbf{r}_{12}+\mathbf{n}L)]^{+}\right\}\\ =&\beta\sum_{\mathbf{n}}\tilde{q}_{\mathrm{Clb,line}}(\mathbf{r}_{12}+\mathbf{n}L)\end{aligned}\tag{3.2.39}$$

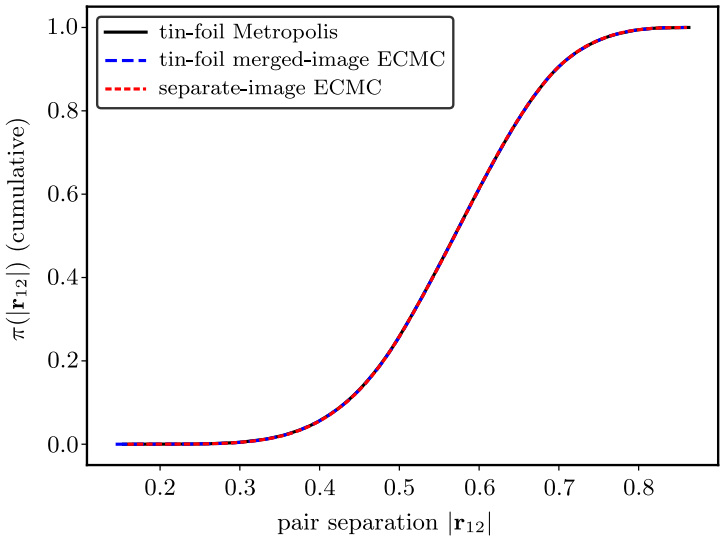


Figure 3.8: Cumulative distribution of $|\mathbf{r}_{12}|$ of two-charge system with $c_1 = c_2 = L = 1$ and $\beta = 2.0$. We tested three algorithms: Metropolis-Hastings MC, ECMC with merged images, ECMC with separate images. (From Ref [1].)

where we use that $\tilde{q}_{\text{Clb,line}}(-\mathbf{r}) = -\tilde{q}_{\text{Clb,line}}(\mathbf{r})$ which can be proved by Eq. 3.2.37. In summary, we show that the imagewise factorization in the line-charge model does not change the net flow, so it just amounts to an extra loop between the active charge and the target. Consequently, flows between them are greater than those in the merged-image approach. We show in Fig. 3.2 the comparison of magnitudes of flows of merged-image and separate-image approaches.

3.3 ECMC cell-veto algorithm

3.3.1 Description

The cell-veto algorithm [95] is an ideal algorithm, under the framework of ECMC, to sample the minimum displacement among dense interactions, i.e., any active particle interacting with other $\mathrm{O}(N)$ particles. This algorithm builds upon two ideas: pair-wise factorization and upper bounds for each of the factors. Consider for example a continuous spin system where the coupling constants are non-zero for all pairs but decay as $J(i,j) = |\mathbf{r}_{ij}|^{-p}$ where $\mathbf{r}_{ij}$ is the minimum separation vector of site i and j. Then we can bound from above the pair potential of each interaction by a constant, and it is easy to sample from N independent constant Poisson processes with different intensities.

In particle system the algorithm takes more complex forms. First, we have to divide the simulation box into cells, usually into square ones, denoted as $\mathcal{C}$. For the time being, each cell is allowed to have only one occupant (later we will consider the exception that more than one object coincides in one cell). Then for each pair of cells C and C', we must provide an upper bound for the event rate for *any* possible particle

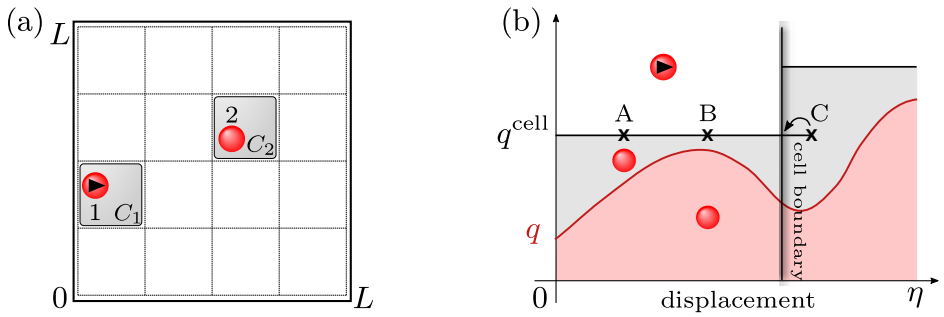


Figure 3.9: Cell-veto algorithm. (a) Division of simulation box into cells. (b) The cell setup provides a piecewise-constant bounding potential for the two-body interaction, and there are three kinds of events related to it—unconfirmed physical event, confirmed event, and cell-boundary event (labeled by A, B and C respectively). (From Ref [1].)

pair with one within C and the other within C'

$$q^{\text{cell}}(C,C') \geq \max_{\mathbf{r}_1\in C,\mathbf{r}_2\in C'} q(\mathbf{r}_{12}=\mathbf{r}_2-\mathbf{r}_1). \tag{3.3.1}$$

Instead of all other particles rejecting the active particle's move, now the triggering objects can be regarded as individual cells, each enforcing a constant event rate. We put all $q(C,C')$ except $C=C'$ into Walker's table, to later sample C' with probability proportional to $q(C,C')$ in $\mathrm{O}(1)$ time. The detail of Walker's method will be given in section 3.3.3. And we define the total cell-veto event rate as

$$q^{\text{tot}}(C) = \sum_{C'\in\mathcal{C},C'\neq C} q^{\text{cell}}(C,C'). \tag{3.3.2}$$

We show the decomposition of cells in Fig. 3.9 (a) .

During the Markov chain, the program samples the displacement η for the event rate q^{tot}, and thereafter samples from Walker's table which cell C' triggers the event. Then it advances the active particle by η and inspects the cell C'. If C' has no occupant, it is a fake event. If C' has an occupant, it calculates the instant separation $\mathbf{r}_{12}$ after advancing the active particle by η, the instant real event rate $q(\mathbf{r}_{12})$, and the confirmation rate $r_C=[q(\mathbf{r}_{12})/q^{\text{cell}}(C,C')]^+$. It confirms the event with probability r_C, and once confirmed, the activity is passed to the occupant of C'.

Usually the cell decomposition is sufficiently regular and symmetric, so that $q^{\text{cell}}(C,C')=q^{\text{cell}}(C'-C)$ relies on the relative separation of C and C'. Then only one Walker's table holding the relative cells is required. When we draw an element ΔC from Walker's table, we must retrieve the target cell through $C'=C+\Delta C$.

Cell-boundary events must be considered every time the active particle crosses the boundary. The computer then changes the internal registration of the cell system. Fig. 3.9 (b) shows three kinds of events related to cells. There are two kinds of exceptional events, one occurring if there is more than one particle in a cell, the other coming from infinite or too large $q^{\text{cell}}(\Delta C)$ for some ΔC's. First, we have to keep a list of surplus particles whose cells have already been occupied, and the number of

the surplus should keep $O(1)$; otherwise, iterating over the surplus list will considerably lower the performance. Second, we have to exclude the exceptional cells from Walker's table and iterate them separately for each move of the active particle. The full procedure can be described as follows

1. Divide the box into cells $\mathcal{C}$, register particles into cells, and keep the surplus list.
2. Estimate $q^{\text{cell}}(\Delta C)$.
3. Put all moderate-valued $q^{\text{cell}}(\Delta C)$ into Walker's table, and keep the list for exceptions.
4. During the Markov chain, for the current active particle, calculate displacements $\{\eta_M\}$ of other factors (those not related to cells).
5. Calculate displacements $\{\eta_e\}$ by occupants of exceptional cells, and by surplus particles.
6. Calculate the displacement η_b to the cell boundary.
7. Calculate the displacement η_c according to q^{tot}.
8. Determine the minimum displacement $\eta = \min\{\eta_c, \eta_b, \{\eta_e\}, \{\eta_M\}\}$ and advance the active particle by η.
9. If η comes from $\{\eta_M\}$ or $\{\eta_e\}$, use the normal procedure to determine the nature of the event (true or fake) and the target particle.
10. If η is from η_b, change the information registered in the cell occupancy system.
11. If η is from η_c, then draw from Walker's table an element ΔC, and calculate the triggering cell $C' = C + \Delta C$. Then inspect the occupant of C':
    - If C' is empty, the event is not confirmed.
    - Let $\mathbf{r}_{12}$ be the instant separation between active particle and the particle in C', calculate $r_C = q(\mathbf{r}_{12})/q^{\text{cell}}(\Delta C)$ and draw a random number $r \in [0, 1]$. If $r \leq r_C$ the event is confirmed, C''s occupant becomes active; otherwise, the event is not confirmed.
12. Go back to step 4 and repeat.

The cell-veto algorithm is essentially a constant bounding potential for arbitrary interactions within each cell. It is valid since $q^{\text{cell}}(C, C')$ is greater than any true event rate inside. Some remarks for cell algorithms are

- The cell size must be carefully tuned. Too small cells lead to too big q^{tot} and hence slow the simulation, while big cells will cause more surplus particles.
- We do not need the exact maximum $q(\mathbf{r}_{12})$ in a cell to obtain q^{cell}. A rough estimator suffices, and one can multiply it by a constant. We will provide several estimators in section 4.5.4.
- If an event is not confirmed or the event is boundary-crossing, we do not, in principle, have to recalculate all proposed displacements from step 4. In this case, only the unconfirmed event or the boundary event needs refreshing.

3.3.2 Cell-veto event rate

We devise the cell algorithm so that the surplus list and the exceptional cells stay constant with increasing system size. It is the cell-veto event rate that predominates all triggered events. The cell-veto events consist of empty-cell events, unconfirmed events due to overestimating q^{cell}, unconfirmed events due to variance of q, and real events.

$$q^{\text{tot}} = R_e\, R_{\text{over}}\, R_{\text{var}}\, q^{\text{tot}}_{\text{real}}. \tag{3.3.3}$$

R_e is the ratio of all cells over occupied cells. It is roughly ~ 10 for charged atoms and the water system described in section 3.6.2. R_{over} means the q^{cell} value returned by the estimator divided by the right-hand side of Eq. 3.3.1. It depends on the estimator and is usually slightly greater than 1. R_{var} is the estimate of the variance between the true $q(\mathbf{r}_{12})$ and $\max\{q(\mathbf{r}_{12})\}$. For faraway cells holding atoms it is close to one, while for dipole systems there is a factor of 4.0 (see Eq. 3.5.13).

The $q^{\text{tot}}_{\text{real}}$ represents the confirmed event rate. For the example of Coulomb particles with identical charges, we have

$$q^{\text{tot}}_{\text{real}} \sim \int_{r_0}^{L} \rho q(\mathbf{r}) \mathrm{d}^3\mathbf{r} \sim \int_{r_0}^{L} \frac{1}{r^2} r^2 \mathrm{d}r \sim L \sim N^{\frac{1}{3}}. \tag{3.3.4}$$

This means that the event rate for plasma simulation scales as $N^{\frac{1}{3}}$ at constant density. For $\mathrm{O}(N)$ particles to move constant distance each, it takes $\mathrm{O}(N^{\frac{4}{3}})$ time. This simplified analysis has actually captured the true complexity and we will give a more detailed derivation in section 3.5.2.

3.3.3 Walker's method

Given m items with probabilities $\pi_1, ..., \pi_m$ with $\sum_{i=1}^{m} \pi_i = 1$, we wonder how to efficiently draw items from the list so that their rates of occurrence correspond to $\{\pi_i\}$. One simple way is by binary search. First we draw a random number $r \in (0, 1]$, and then find the minimum number k so that $\sum_{i=1}^{k} \pi_i \geq r$, which can be done with $\mathrm{O}(\log N)$ time using binary search.

Walker's algorithm provides an optimally efficient approach that can sample with $\mathrm{O}(1)$ complexity, under the condition that the probabilities must be static and fixed before the simulation. It works through elaborately disassembling π_i, and rearranging them into m boxes with volume $1/m$. Each of these holds exactly two pieces of different origins (except for the end of the construct). The procedure can be described as follows

1. Put all π_i into two lists, one with $\pi_i > 1/m$ ("big" list) and the other with $\pi_i \leq 1/m$ ("small" list).

2. Take one item π_s from the small list and one π_b from the big list and register a Walker item $[(\pi_s, s), (1/m - \pi_s, b)]$. A Walker item is a tuple of the form (probability, original index).

3. The non-registered big item now is $\pi_b - (1/m - \pi_s)$. Put it back to the small or big list according to its remaining value. Go back to step 2 until one of the lists becomes empty.

4. Elements in the remaining list must be of probability $1/m$. Put them all into Walker's table.

After m iterations of the steps above, Walker's table has m elements, each with a tuple of two probability pieces. The two probabilities sum up to $1/m$ for each of the table's elements. Note that the number of remaining items in both lists equals the iterations to be done. Once one of the lists is exhausted, the remaining list items must be all with a value $1/m$. Fig. 3.10 illustrates the rearrangement of the raw probabilities. Once Walker's table is ready, one can randomly draw an item $[(\pi_i, i), (1/m - \pi_i, j)]$

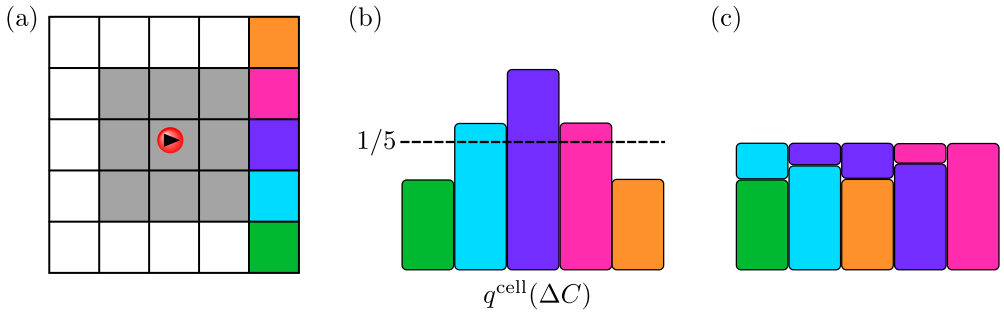


Figure 3.10: Schema of Walker's algorithm. (a) Cell system. Gray cells are treated as exceptions, while others are to be sampled through Walker's method, but we take five with color for simplicity. (b) Values of q^{cell} with the horizontal line indicating their average. (c) The resulting Walker's table. (Adapted from Ref [95].)

from its m elements, and return i with probability $m\pi_i$ and j with probability $1 - m\pi_i$.

3.4 Coulomb bounding potential

The Coulomb potential for periodic boundary conditions given by Eq. 3.2.6 is non-invertible. The cell algorithm has already provided an effective bounding potential based on cells. For excluded cells and surplus particles, however, the Coulomb derivative is not upper-bounded. Hence non-cell-based schemes are required to sample displacements.

We propose that $U^{(\mathrm{b})}_{\mathrm{Clb}}(\mathbf{r}_{12}) = k_b c_1 c_2 / |\mathbf{r}_{12}|$ serves as a good candidate for Coulomb bounding potential, because they both have a $1/|\mathbf{r}|$ at the origin. The constant k_b has to be big enough to bound the Coulomb derivative from above. According to section 3.2.4

$$k_b \geq \max_{\mathbf{r}} \tilde{q}_{\mathrm{f},1}(\mathbf{r}) \approx 1.5836. \tag{3.4.1}$$

Rather than being smooth everywhere except at the origin, $U^{(\mathrm{b})}_{\mathrm{Clb}}$ is discontinuous at the surface $x = \pm L/2$, but has the same period L as U_{Clb} of Eq. 3.2.6.

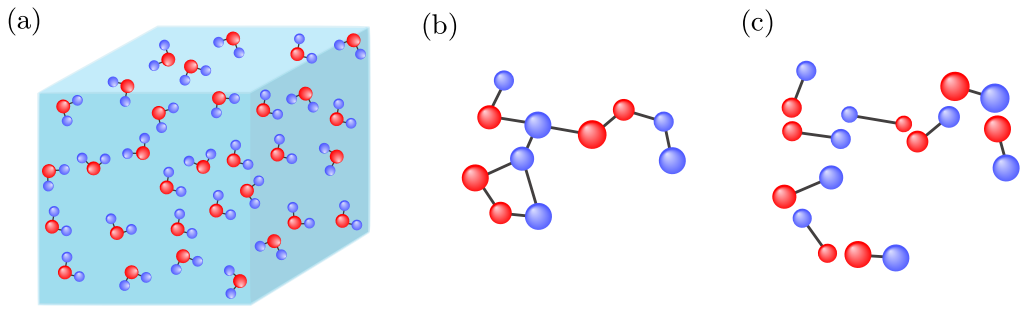


Figure 3.11: Dipole systems. (a) Water model with charged atoms. (b) A big neutral molecule, with each atom charged. (c) Decomposition of the big molecule into dipoles.

3.5 Dipole factors

Many realistic systems are neutral, mostly because they appear as dipoles or can be treated as a combination of dipoles. Models of small molecules consisting of charged atoms are often natural dipoles such as water; otherwise, for big molecules such as charged polymers, we have ways to decompose them into two-atom dipoles, as shown in Fig. 3.11.

As mentioned in section 3.3.2, the event rate of a three-dimensional Coulomb system increases as $N^{\frac{1}{3}}$. This scaling is a natural effect of the $1/r^2$ Coulomb interaction. In this framework, the event rate can however be lowered by exploiting the $1/r^4$ dipole-dipole interaction. The answer is positive in the framework of ECMC. By translating a dipole without rotating or changing its inner structure, the event rate triggered by a faraway dipole can scale as $1/|\mathbf{r}_{12}|^4$, which alone cannot, however, guarantee ergodicity. In this section, we will introduce the dipole factor for moving a single active atom. This way can take advantage of the $1/r^3$ charge-dipole interaction and finally reach an $\mathrm{O}(\log N)$ total event rate.

3.5.1 Two-dipole case

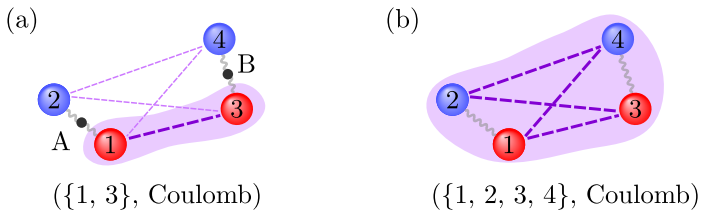
We consider the ECMC event rate for two dipoles in an infinite space. Particles in dipole one are 1 and 2, and those in the other are 3 and 4. Each particle carries the same charge unit, by $c_1 = c_3 = -c_2 = -c_4$. We denote their centers as A and B, with condition that $\mathbf{r}_{AB} \gg \mathbf{r}_{12}$ and $\mathbf{r}_{AB} \gg \mathbf{d}_{34}$, as shown in Fig. 3.12. Coulomb interactions exist between 1-3, 1-4, 2-3, and 2-4.

If we treat the four pairs of Coulomb potentials as four factors referred to as atom factors, $\mathcal{M} = \{(\{i,j\}, \mathrm{Coulomb})\, i = 1,2; j = 3,4\}$, the event rate for active particle 1 is

$$q_{1,\mathrm{atom}} = \beta c^2 \left(\left[\frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{13}|} \right]^+ + \left[-\frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{14}|} \right]^+ \right). \tag{3.5.1}$$

Using $\mathbf{r}_{13} = \mathbf{r}_{1B} + \mathbf{r}_{B3}$, we expansion

$$\frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{1B} + \mathbf{r}_{B3}|} \approx (1 + \mathbf{r}_{B3} \cdot \nabla_B) \frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{1B}|}. \tag{3.5.2}$$



({1, 3}, Coulomb) ({1, 2, 3, 4}, Coulomb)

Figure 3.12: Factorization of Coulomb for two dipoles. (a) Atom factorization. There are four factors, {1,3}, {1,4}, {2,3}, and {2,4}, but only {1,3} is shown. (b) Dipole factorization, and there is only one dipole factor. (From Ref [1].)

and the same for $1/|\mathbf{r}_{14}|$. By taking into account $\mathbf{r}_{B3}\cdot\nabla_B \sim r_{B3}/L \ll 1$, we find that the leading term survives in Eq. 3.5.1 and results in

$$\begin{aligned} q_{1,\text{atom}} &\approx \beta c^2 \left(\left[\frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{1B}|} \right]^+ + \left[-\frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{1B}|} \right]^+ \right) \\ &= \beta c^2 \left| \frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{1B}|} \right|, \end{aligned} \tag{3.5.3}$$

which decays as $1/|\mathbf{r}_{1B}|^2$ for large $\mathbf{r}_{1B}$.

We can also bundle four pairs of Coulomb interaction into one factor, $\mathcal{M} = \{(\{1,2,3,4\}, \text{Coulomb})\}$ which we call dipole factor. It sounds like an artificial four-body Coulomb interaction with no distinct physical effect from pairwise ones, but in ECMC when we calculate the event rate for active particle 1

$$\begin{aligned} q_{1,\text{dipole}} &= \beta c^2 \left[\frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{13}|} - \frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{14}|} \right]^+ \\ &\approx \beta c^2 \left[(\mathbf{r}_{43}\cdot\nabla_B) \frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{1B}|} \right]^+, \end{aligned} \tag{3.5.4}$$

in which the first term of the right-hand side of Eq. 3.5.2 cancels but the second term survives. Eq. 3.5.4 implies that event rate for the particle-dipole interaction scales as $1/|\mathbf{r}_{1B}|^3$ (one derivative lowers it by one order).

To further exploit the dipole-dipole interaction with its $1/r^4$ scaling, we have to expand to second order. Take $1/|\mathbf{r}_{13}|$ for example,

$$\frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{1A}+\mathbf{r}_{AB}+\mathbf{r}_{B3}|} \approx [1+\mathbf{r}_{B3}\cdot\nabla_B+\mathbf{r}_{A1}\cdot\nabla_A+\mathcal{T}_2] \frac{\partial}{\partial x_A} \frac{1}{|\mathbf{r}_{AB}|}, \tag{3.5.5}$$

where $\mathcal{T}_2$ is the second order expansion operator

$$\mathcal{T}_2 = \frac{1}{2}(\mathbf{r}_{B3}\cdot\nabla_B+\mathbf{r}_{1A}\cdot\nabla_A)^2. \tag{3.5.6}$$

The event rate caused by the move of particles 1 and 2 as a whole is

$$q_{12,\text{dipole}} = \beta c^2 \left[\frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{13}|} - \frac{\partial}{\partial x_1} \frac{1}{|\mathbf{r}_{14}|} - \frac{\partial}{\partial x_2} \frac{1}{|\mathbf{r}_{23}|} + \frac{\partial}{\partial x_2} \frac{1}{|\mathbf{r}_{24}|} \right]^+. \tag{3.5.7}$$

By inserting Eq. 3.5.5 into the above equation, we find that the leading term and the first order term of Eq. 3.5.5 both vanish, due to the relations $\mathbf{r}_{B3} = -\mathbf{r}_{B4}$ and $\mathbf{r}_{A1} = -\mathbf{r}_{A2}$. The first surviving term is brought by $\mathcal{T}_2$

$$\begin{aligned} q_{12,\text{dipole}} &\approx \beta c^2 \left[\frac{1}{2} \sum_{i=1,2;j=3,4} (\mathbf{r}_{Ai} \cdot \nabla_A + \mathbf{r}_{Bj} \cdot \nabla_B) \frac{\partial}{\partial x_A} \frac{1}{|\mathbf{r}_{AB}|} \right]^+ \\ &= \beta c^2 \left[\frac{1}{2} \left[(\mathbf{r}_{12} \cdot \nabla_A)^2 + (\mathbf{r}_{34} \cdot \nabla_B)^2 \right] \frac{\partial}{\partial x_A} \frac{1}{|\mathbf{r}_{AB}|} \right]^+ . \end{aligned} \tag{3.5.8}$$

It decays as $1/|\mathbf{r}_{AB}|^4$ for large $\mathbf{r}_{AB}$. As suggested before, when we translate the dipole as a whole, the derivative with respect to this move takes the form of Eq. 3.5.7. And we will show in section 3.5.3 that the inside-first lifting scheme that prioritizes the atom 2 as the next active particle, will have probability as $1/|\mathbf{r}|^4$ to pass its activity to atoms in dipole B.

3.5.2 Homogeneous systems

In this section, we will analyze the total event rate of the Coulomb factor with atom and dipole factorizations. We always assume that the system is homogeneous, and refer to the density of charges or dipoles as ρ.

First, the tin-foil Coulomb potential of Eq. 3.2.6 takes the form

$$U_{\text{C}}(\mathbf{r}_{12}) = \frac{c_1 c_2}{|\mathbf{r}_{12}|} f_1\left(\frac{\mathbf{r}_{12}}{L}\right), \tag{3.5.9}$$

where f_1 is a smooth function defined in $[-1/2, 1/2]^3$. This is manifest by noting that for $\mathbf{r}_{12} \to \gamma \mathbf{r}_{12}$ and for $L \to \gamma L$, we have $U_{\text{C}} \to U_{\text{C}}/\gamma$ (which can be obtained by $\alpha \to \alpha/\gamma$ in Eq. 3.2.6). Let $\mathbf{r}_s = \mathbf{r}/L$ and we have

$$U_{\text{C}}(\mathbf{r}_s) = \frac{c_1 c_2}{L} \frac{1}{|\mathbf{r}_s|} f_1(\mathbf{r}_s), \tag{3.5.10}$$

$$\tilde{q}_{\text{Clb}}(\mathbf{r}_s) = \frac{c_1 c_2}{L^2} \frac{x_s f_1(\mathbf{r}_s) - |\mathbf{r}_s|^2 \partial_{x_s} f_1(\mathbf{r}_s)}{|\mathbf{r}_s|^3} = \frac{c_1 c_2}{L^2} \frac{x_s}{|\mathbf{r}_s|^3} f_2(\mathbf{r}_s). \tag{3.5.11}$$

Since $\partial_{x_s} f_1(\mathbf{r}_s)|_{x_s=0} = 0$, $f_2(\mathbf{r}_s)$ is another smooth function that we denote as $\tilde{q}_{\text{f},1}$ in section 3.2.4.

For the case of atom factorization, assuming that all particles have charge c, the total event rate for an active charge is

$$\begin{aligned} \langle Q_{\text{atom}} \rangle &= \int_{[-L/2,L/2]^3 - R(\epsilon)} \rho \left[\tilde{q}_{\text{Clb}}(\mathbf{r}) \right]^+ \mathrm{d}^3\mathbf{r} \\ &= \int_{[-1/2,1/2]^3 - R(\epsilon/L)} \rho L^3 \left[\tilde{q}_{\text{Clb}}(\mathbf{r}_s) \right]^+ \mathrm{d}^3\mathbf{r}_s \\ &= \int_{[-1/2,1/2]^3 - R(\epsilon/L)} \rho c_1 c_2 L \left[\frac{x_s}{|\mathbf{r}_s|^3} f_2(\mathbf{r}_s) \right]^+ \mathrm{d}^3\mathbf{r}_s, \end{aligned} \tag{3.5.12}$$

where $R(\epsilon)$ is the sphere of radius ϵ centering around the origin. The integrand is proportional to $1/|\mathbf{r}_s|^2$ while $\mathrm{d}^3\mathbf{r}_s$ contributes $|\mathbf{r}_s|^2$ around the origin as well. Thus the integration is finite around $\mathbf{r}_s = 0$ and the final result is proportional to L. We conclude that the event rate for atom factorization scales as $N^{1/3}$ under periodic boundary conditions.

As for dipole factorization, we use $\mathbf{d}\cdot\nabla\tilde{q}_{\mathrm{Clb}}$ to approximate the charge-dipole event rate for a faraway dipole $\mathbf{d}$. However, we have to average over dipole orientations. To this end, let $\mathbf{v}$ be an arbitrary vector and $\mathbf{n}$ a vector uniformly distributed on $R(1)$. Then

$$\langle[\mathbf{n}\cdot\mathbf{v}]^+\rangle_{\mathbf{n}} = \frac{|\mathbf{v}|}{4\pi}\int_0^{\pi}\mathrm{d}\theta\int_0^{2\pi}\mathrm{d}\phi\sin\theta\,[\cos\theta]^+ = \frac{|\mathbf{v}|}{4}, \tag{3.5.13}$$

and so

$$\begin{aligned}\langle Q_{\mathrm{dipole}}\rangle &= \int_{[-L/2,L/2]^3-R(\epsilon)}\rho\langle[\mathbf{d}\cdot\nabla\tilde{q}_{\mathrm{Clb}}(\mathbf{r}_s)]^+\rangle_{\mathbf{d}}\mathrm{d}^3\mathbf{r}_s\\ &= \frac{|\mathbf{d}|}{4}\rho|c_1c_2|\int_{[-1/2,1/2]^3-R(\epsilon/L)}\left|\nabla_s\left(\frac{x_s}{|\mathbf{r}_s|^3}f_2(\mathbf{r}_s)\right)\right|\mathrm{d}^3\mathbf{r}_s,\end{aligned} \tag{3.5.14}$$

where ∇_s is the gradient operator with respect to $\mathbf{r}_s$ and $\nabla_s = L\nabla$. The integrand in Eq. 3.5.14 is proportional to $1/|\mathbf{r}_s|^3$. Thus we have

$$\langle Q_{\mathrm{dipole}}\rangle \approx \frac{|\mathbf{d}|}{4}|c_1c_2|\left(C-\log\frac{\epsilon}{L}\right) \approx \frac{|\mathbf{d}|}{4}|c_1c_2|\log L,\ \text{ as }\ L\to\infty. \tag{3.5.15}$$

Dipole factorization can reduce the event rate scaling to $\log L$.

Likewise, we may calculate the total event rate with respect to dipole translation.

$$\begin{aligned}\langle Q_{\mathrm{dipole-dipole}}\rangle &= \int_{[-L/2,L/2]^3-R(\epsilon)}\rho\langle[\mathbf{d}_2\cdot\nabla\left(\mathbf{d}_1\cdot\nabla\tilde{q}(\mathbf{r})\right)]^+\rangle_{\mathbf{d}_2}\mathrm{d}^3\mathbf{r}\\ &= \frac{|\mathbf{d}_1||\mathbf{d}_2|}{4}\rho\int_{[-L/2,L/2]^3-R(\epsilon)}|\nabla\cdot(\mathbf{n}_{d_1}\cdot\nabla\tilde{q}(\mathbf{r}))|\,\mathrm{d}^3\mathbf{r}\\ &= \frac{|\mathbf{d}_1||\mathbf{d}_2|c_1c_2\rho}{4L}\int_{[-1/2,1/2]^3-R(\epsilon/L)}\left|\nabla_s(\mathbf{n}_{d_1}\cdot\nabla_s)\left(\frac{x_s}{|\mathbf{r}_s|^3}f_2(\mathbf{r}_s)\right)\right|\mathrm{d}^3\mathbf{r}_s.\end{aligned} \tag{3.5.16}$$

As the integrand scales as $1/|\mathbf{r}_s|^4$, we obtain

$$\langle Q_{\mathrm{dipole-dipole}}\rangle \propto \frac{|\mathbf{d}_1||\mathbf{d}_2|c_1c_2\rho}{4}\left(\epsilon^{-1}-\frac{C}{L}\right), \tag{3.5.17}$$

which means that the total event rate is bounded from above and the difference to the limit decays as $1/L$.

3.5.3 Dipole lifting schemes

We introduce in section 2.2.2 that a lifting scheme is needed to select the next particle to move in the presence of many-body interactions. The ratio lifting scheme [78] is a universal way to distribute the probability flow in proportion to the magnitude of derivatives for those decreasing energy. For dipoles, the ratio lifting scheme works as

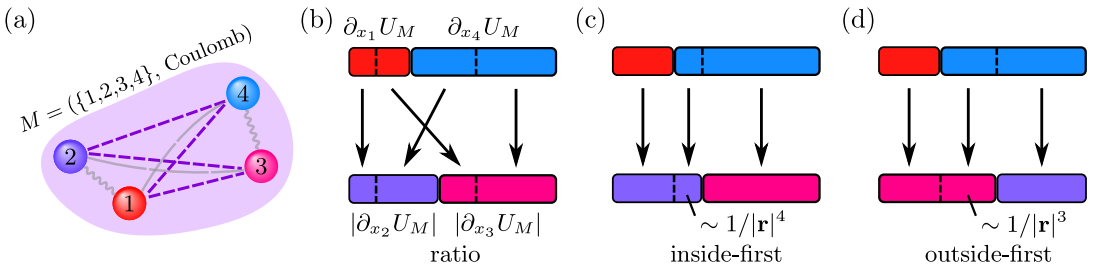


Figure 3.13: Lifting schemes for a dipole factor. (a) Two dipoles whose Coulomb interactions are included in a dipole factor. It is assumed that $\partial_{x_1} U_M > 0$, $\partial_{x_4} U_M > 0$, $\partial_{x_2} U_M < 0$ and $\partial_{x_3} U_M < 0$. (b) Ratio lifting scheme. (c) Inside-first lifting scheme. The outflow to activate atoms in the other dipoles is indicated by the mismatch of $\partial_{x_1} U_M$ and $|\partial_{x_4} U_M|$ that is proportional to $1/|\mathbf{r}|^4$. (d) Outside-first lifting scheme. (From Ref [1].)

indicated in Fig. 3.13 (b) and ensures the target distribution. Moreover, we propose in the following two other lifting schemes, namely an "inside-first" lifting scheme and an "outside-first" lifting one featuring dipole-dipole interaction.

We first classify and align the derivative bars, i.e., bars of length proportional to absolute value of the corresponding derivatives, into a positive table and a negative table. Those with positive derivatives are sorted in a positive table by their atom indices, and those with negative derivatives are inserted into a negative table by order of atom indices. Since the positive table's length equals the negative table's length, any position in the positive table corresponds to one in the negative one, meaning that we can establish a bijective map between the two tables. Then we draw a random position within the active bar, which must be in the positive table, and search the resulting point in the negative table. In this way, we find the target atom, as shown in Fig. 3.13 (c) . For the outside-first scheme, we reverse the negative table at first (or equivalently reverse the positive table), and do the same mapping to find the target atom, as shown in Fig. 3.13 (d) .

Since we index atoms in an dipole by dipole order[1], atoms of the same dipole will be sorted to the same side of both tables. Hence the inside-first scheme prioritizes other atoms in the same dipole to be the target atom. In contrast, the outside-first scheme favors atoms from the other dipole, and it maximizes the flow going out of the dipole. Furthermore, we find that outgoing flow, i.e., the inter-dipole lifting rate in the inside-first scheme is exactly indicated by the mismatch between positive bars and negative bars of one dipole (see Fig. 3.13 (c)), whose magnitude is given by Eq. 3.5.8 in proportion to $1/|\mathbf{r}_{AB}|^4$. Eq. 3.5.17 tells us that the outgoing probability is bounded from above no matter how large the system is.

The event rates and lifting rates are summarized in the Table. 3.5.3, for atom factorization and dipole factorization with three proposed lifting schemes.

[1] Atom indices must be fixed a priori. Orders in both tables for one physical configuration are invariable throughout the Markov chain. One easy pitfall is that one may tend to place the dipole containing the active atom always to the left, which proves to be wrong. One good practice is to index an atom by a tuple of molecule and atom number (i_M, i_A), and sort with relation $(i_1, j_1) \leq (i_2, j_2) \Leftrightarrow i_1 < i_2 \,\text{or}\, (i_1 = i_2 \,\text{and}\, j_1 \leq j_2)$.

scheme		q_{intra}	q_{inter}	$< Q_{\text{intra}} >$	$< Q_{\text{inter}} >$
atom factors		0	$\lvert\mathbf{r}\rvert^{-2}$	0	$N^{\frac{1}{3}}$
dipole factors	ratio lifting	$\lvert\mathbf{r}\rvert^{-3}$	$\lvert\mathbf{r}\rvert^{-3}$	$\log N$	$\log N$
	inside-first lifting	$\lvert\mathbf{r}\rvert^{-3}$	$\lvert\mathbf{r}\rvert^{-4}$	$\log N$	$B - C/N^{\frac{1}{3}}$
	outside-first lifting	$\lvert\mathbf{r}\rvert^{-3}$	$\lvert\mathbf{r}\rvert^{-3}$	$\log N$	$\log N$

Table 3.1: Pairwise Coulomb event rate scalings with respect to dipole separation, and total Coulomb event rate scalings with respect to system size, for different factorizations and lifting schemes.

3.6 Numerical tests

We give two more simulations apart from Fig. 3.8 to verify our methods. One contains two dipoles and the other simulates 32 water molecules.

3.6.1 Dipole system

In the dipole model, we introduce harmonic bonds connecting two atoms within a dipole

$$U_{\text{bond}}(\mathbf{r}_{12}) = \frac{1}{2} k_b \left(|\mathbf{r}_{12}| - r_b\right)^2 \tag{3.6.1}$$

and a short-range repulsive potential

$$U_{\text{rep}}(\mathbf{r}_{12}) = k_2 \left(\frac{r_0}{|\mathbf{r}_{12}|}\right)^6 \tag{3.6.2}$$

with k_b, k_2, r_0 and r_b all positive. We make it decay fast enough that no periodicity needs considering. With the indices of Fig. 3.13 (a) , harmonic potentials exist between atom 1 and 2, and between 3 and 4. Moreover, we impose repulsive potential between atom 1 and 4, 2 and 3. Thus the harmonic and repulsive factors are

$$\{(\{1,2\},\text{bond}),(\{3,4\},\text{bond}),(\{1,4\},\text{rep}),(\{2,3\},\text{rep})\}. \tag{3.6.3}$$

Note that we are using the notation introduced in section 2.2.2 that a factor is represented by (I_M, T_M) which are the index set and the type respectively. Two possible factorizations of the Coulomb interaction are

$$\begin{aligned}\text{Particle factorization} :\{&(\{1,3\},\text{Coulomb}),(\{1,4\},\text{Coulomb}),\\ &(\{2,3\},\text{Coulomb}),(\{2,4\},\text{Coulomb})\},\end{aligned} \tag{3.6.4}$$

and

$$\text{Dipole factorization} :\{(\{1,2,3,4\},\text{Coulomb})\}. \tag{3.6.5}$$

In Fig. 3.14, we simulate two dipoles using five methods: merged-image particle factorization, separate-image dipole factorization with inside-first lifting scheme, merged-image dipole factorization with ratio, inside-first, and outside-first schemes. All methods agree visibly in the distribution of $|\mathbf{r}_{13}|$ and $|\mathbf{r}_{14}|$, and we will give more accurate verifications in section 5.3.

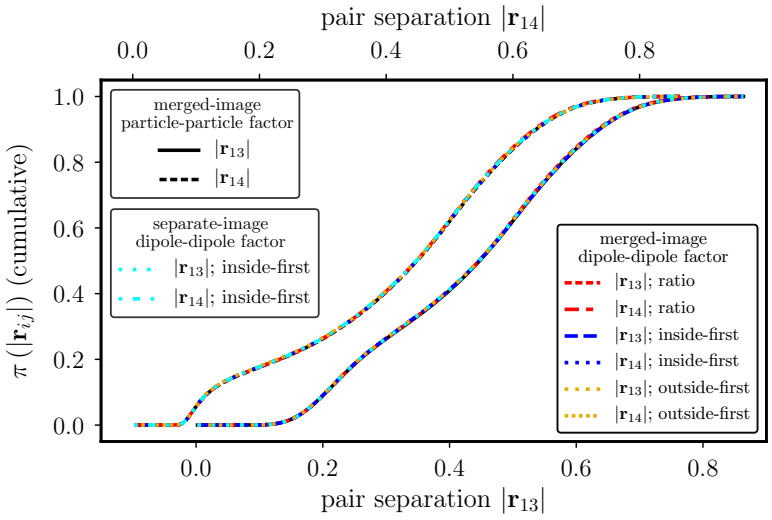


Figure 3.14: Cumulative distributions of $|\mathbf{r}_{13}|$ and $|\mathbf{r}_{14}|$ of two dipole simulation using five approaches, with factor sets indicated by Eq. 3.6.3, Eq. 3.6.4 and Eq. 3.6.5. Simulation box is periodic with $L = 1$, $c_1 = -c_2 = c_3 = -c_4 = 1.0$, $k_b = 400$, $k_2 = 1.0$, $r_0 = 0.1$ and $\beta = 1.0$. (From Ref [1].)

3.6.2 SPC/Fw water model

The popular SPC/Fw [74] (simple point-charge flexible) water model treats each atom as a point charge. Interactions include a harmonic force as in Eq. 3.6.1 with $r_0 = 1.012\,\text{Å}$ and $k_b = 1059.162\,\text{kcal}\,\text{mol}^{-1}\,\text{Å}^{-2}$, and a bending potential related to each H-O-H angle

$$U_{\text{bending}}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) = \frac{1}{2} k_a \left(\phi_{123} - \phi_0\right)^2, \tag{3.6.6}$$

where ϕ_{123} represents the angle formed by the vectors $\mathbf{r}_{21}$ and $\mathbf{r}_{23}$, with $\phi_0 = 113.24°$ and $k_a = 75.90\,\text{kcal}\,\text{mol}^{-1}\,\text{rad}^{-2}$. A Lennard-Jones potential acts between any pair of oxygens

$$U_{LJ}(\mathbf{r}_{12}) = k_{\text{LJ}} \left[\left(\frac{\sigma}{|\mathbf{r}_{12}|} \right)^{12} - \left(\frac{\sigma}{|\mathbf{r}_{12}|} \right)^{6} \right] \tag{3.6.7}$$

with $k_{LJ} = 0.62\,\text{kcal}\,\text{mol}^{-1}$ and $\sigma = 3.165\,\text{Å}$. The Lennard-Jones potential decays sufficiently fast that no periodic boundary conditions needs to be considered for it. Finally, Coulomb potentials exist between any intermolecular atom pair, with $-2c_{\text{H}} = c_{\text{O}} = -0.82e$ where e is elementary charge.

The factor set without Coulomb factor between two SPC/Fw water molecules labeled as in Fig. 3.15 (a) is

$$\begin{gathered}(\{1,2\},\text{bond}),\ (\{2,3\},\text{bond}),\ (\{4,5\},\text{bond}),\ (\{5,6\},\text{bond}) \\ (\{1,2,3\},\text{bending}),\ (\{4,5,6\},\text{bending}),\ (\{2,5\},\text{LJ}).\end{gathered} \tag{3.6.8}$$

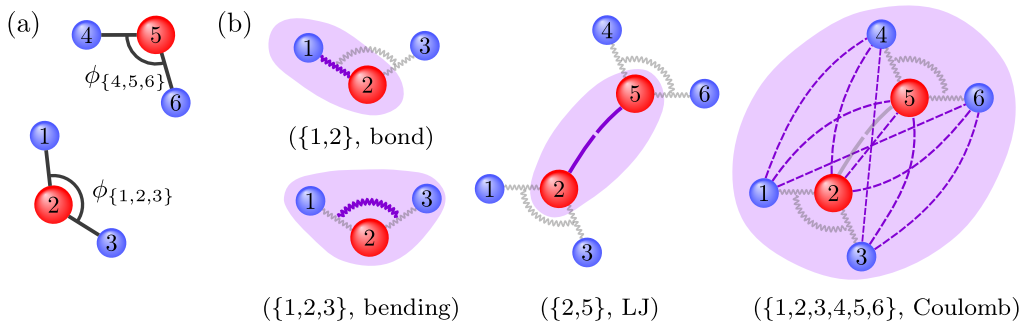


Figure 3.15: SPC/Fw water model and corresponding factors. (a) Two H_2O molecules consisting of six point charges indexed from 1 to 6. (b) Examples of four types of factors: bond, bending, Lennard-Jones, and Coulomb (via dipole factorization). (From Ref [1].)

Likewise, the particle and dipole Coulomb factor sets are[2]

$$\text{Particle factorization} : \{(\{i,j\}, \text{Coulomb}) \text{ for } i \in \{1,2,3\}, j \in \{4,5,6\}\} \tag{3.6.9}$$

$$\text{Dipole factorization} : \{(\{1,2,3,4,5,6\}, \text{Coulomb})\}. \tag{3.6.10}$$

Fig. 3.15b shows the dipole factorization for two H_2O molecules.

First, we verified our methods by comparing the distribution of the oxygen-oxygen separation $|\mathbf{r}_{OO}|$ as shown in Fig. 3.16. In order to investigate the event composition and justify the efficiency of dipole factors, we measure event rates for increasing system sizes for all four interactions, as shown in Fig. 3.17. We conclude that the Coulomb events are predominant, and that $\langle Q_{\text{Clb}} \rangle$ is proportional to $\log N_{H_2O}$ as $N_{H_2O} \to \infty$, while the event numbers of the other interactions remain roughly constant in varying system sizes since they are local. By extrapolation we expect that $\langle Q_{\text{Clb}} \rangle \approx 107 \text{Å}^{-1}$ for 2^{20} water molecules.

Furthermore, we select systems of $N_{H_2O} = 64,\ 128$ and 256 and focus on the distance between the active and the target particle at the moment when Coulomb events happen. Different lifting schemes for dipole factorization are compared in Fig. 3.18. All systems exhibit sharp peaks around 1Å with heights proportional to $\log N_{H_2O}$, complying with Table. 3.5.3. As expected, the inside-first scheme as shown in Fig. 3.18 (c) has a faster decay than others as $1/|\mathbf{r}|^2$ ($1/|\mathbf{r}|^4$ in Eq. 3.5.8 multiplied by the spherical measure $|\mathbf{r}|^2$), which numerically implies that the outflow of the inside-first scheme scales as $1/|\mathbf{r}|^4$ and the total outgoing probability is upper bounded.

Also, we test the influence of the chain length over the rotation speed of a single water molecule. Our ECMC implementation features atomic motion along axes in comparison with arbitrary directions, which greatly simplifies the code, while it remains a question whether such a scheme can achieve as fast molecular rotation as other methods.

We define the orientation of a molecule as its polarization

$$\mathbf{P} = \mathbf{r}_{OH1} + \mathbf{r}_{OH2}. \tag{3.6.11}$$

[2]It is possible to decompose a H_2O molecule into two dipoles each with moment $0.41e\mathbf{r}_{OH}$ but we believe that it does not make any significant difference from treating a water molecule as one dipole.

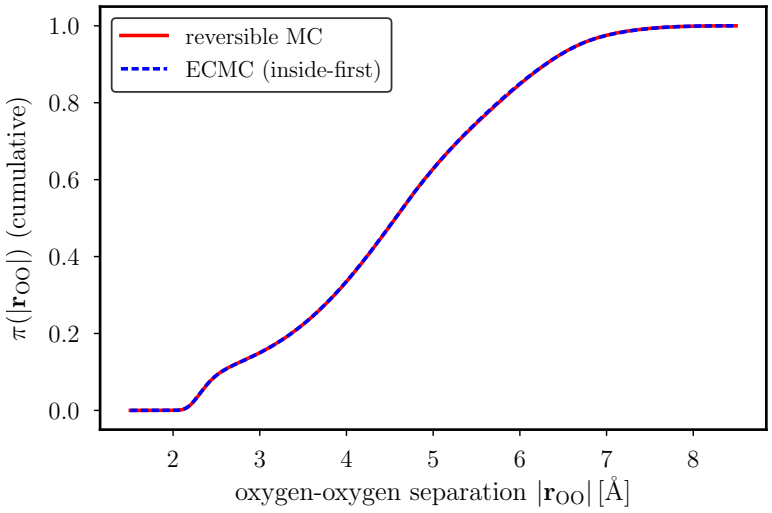


Figure 3.16: Distribution of the oxygen-oxygen separation through the traditional Monte Carlo and ECMC with a dipole factor and the inside-first lifting scheme. The simulation is done for 32 molecules in a time-driven manner, at temperature 300K and density 1g/cm^3. (From Ref [1].)

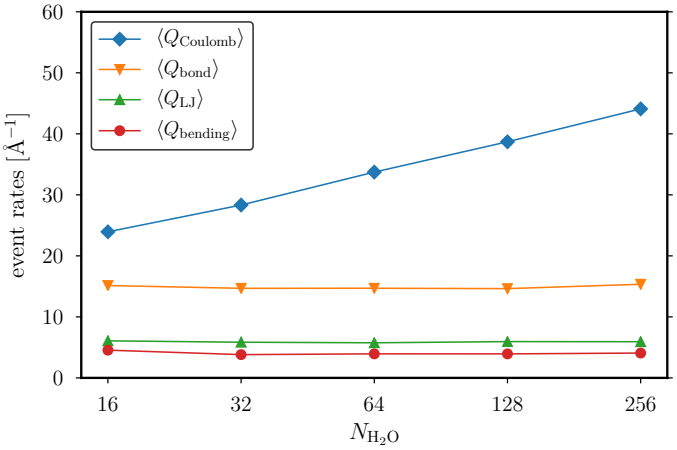


Figure 3.17: Event rates of different factor types with increasing system size, from 16 water molecules to 256. Measurements are done using dipole factorization and the same parameters as in Fig. 3.16. (From Ref [1].)

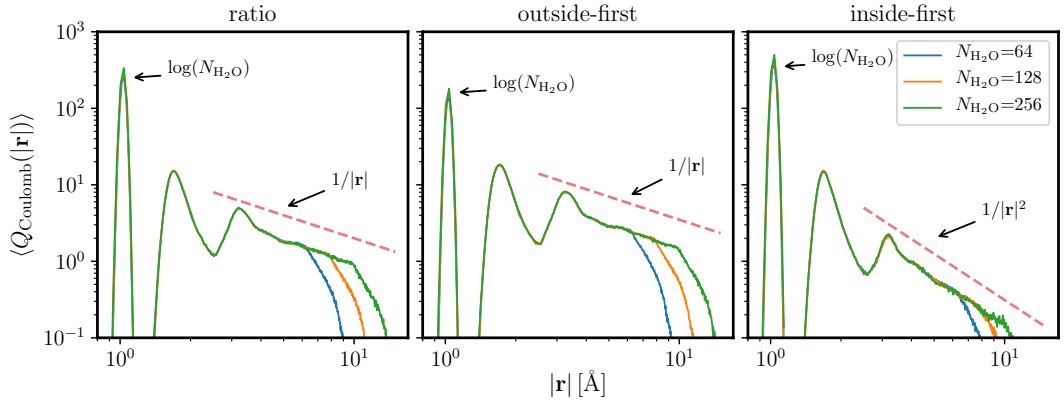


Figure 3.18: Distributions of $|\mathbf{r}_{at}|$ at moments when events occur, of Coulomb dipole factor through ratio, inside-first and outside-first lifting schemes. (From Ref [1].)

Taking a global one-angstrom displacement as a unit of time, we can define the auto-correlation of $\mathbf{P}$ through the inner product

$$C_{\mathbf{P}}(t) = \langle \mathbf{P}(s) \cdot \mathbf{P}(s+t) \rangle_s. \tag{3.6.12}$$

In equilibrium, $C_{\mathbf{P}}(t)$ decays exponentially as $C_{\mathbf{P}}(t) \sim e^{-t/\lambda}$ with the autocorrelation time λ. In Fig. 3.19, we test two direction switching schemes each with respect to a range of chain length ℓ. One scheme is the random choice of the next direction, and the other is cyclic as $xyzxyz\ldots$. It shows that minimal correlation is realized at around $\ell \approx 0.1$Å, and cyclic the scheme has smaller λ than the random scheme. We generalize the result to $\ell = 0.2N_{\mathrm{H_2O}}$ for N-molecule systems as used in section 6.3.2.

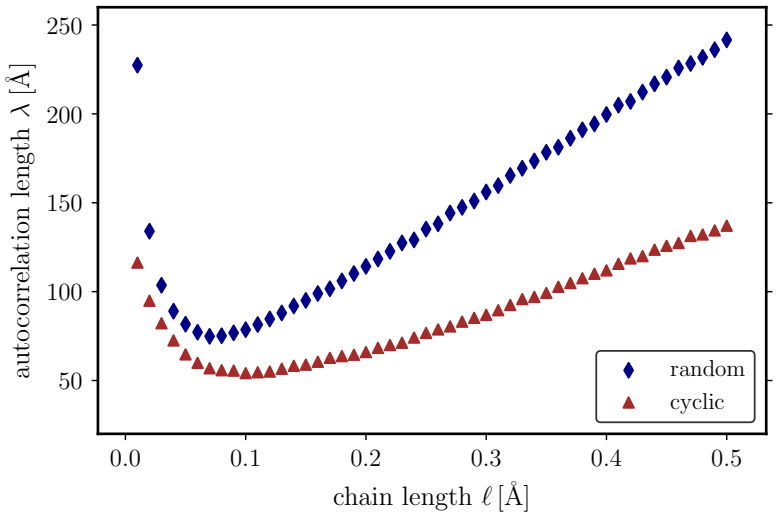


Figure 3.19: Autocorrelation length λ for the dipole moment in ECMC of a single H_2O molecule (fixed chain length ℓ) for the cyclic sequence of event-chain directions ($xyzxyz$...) and for random direction sequence. (From Ref [1].)

Chapter 4

JeLLyFysh architecture

All current molecular-simulation packages employ molecular dynamics, for the reason that Monte Carlo cannot produce time-dependent information and is slow in treating Coulomb interactions. With our development of long-range ECMC in theory, it becomes a promising candidate for long-range systems. Also, Monte Carlo performs non-realistic moves for a fast exploration of configuration space, which is helpful in various settings. For example, it may take long time for a molecular dynamics simulation to observe a certain chemical reaction, one can prepare multiple independent initial states via Monte Carlo methods and then simulate them with molecular dynamics. However, the slow diffusive bahavior of Monte Carlo methods has impeded this approach.

On the other hand, since ECMC has shown its powerful advantages in simple physics models, it is usually desirable to generalize to a more complicated system up to macro-scale biochemical molecules. In pursuit of methodological improvement and widespread usage, an efficient platform will considerably contribute. Reusable codes will ease the burden of reinventing programs for all researchers and reduce the risk of bugs that have long been undermining the reliability of simulation research.

Moreover, a robust application also serves as a scientific and methodological benchmark. New features of ECMC or comparison between methods will benefit from an ECMC application with excellent stability. Adaption to diverse chemical and biological models cannot be done by individuals but the entire society of relevant fields. Modern scientific software such as LAMMPS [14], GROMACS [97], and AMBER [98] has spawned their communities, to which thousands of worldwide developers make contributions. Taking them as our model, we developed JELLYFYSH (JF), an ECMC simulation application for all-atom systems, in hopes that JF can one day become useful in routine applications.

We are aware of many kinds of functionalities and desirable simulation environments in the long term. Apart from the advanced algorithms, many aspects are crucial to an application's success, such as parallelization, GPU support, data visualization, and extendability to non-ECMC methods(quantum, replica exchange, etc.). We leave these possibilities open for the future but lay the foundation of JF.

When JF-V1.0 was released [2] in August 2019, it was able to run particle systems consisting of atoms or dipoles, with short-range potentials and the long-range

Coulomb potential, and supported by cell-veto algorithms and other bounding potentials. We implemented its structure and simple applications through about 20,000 lines of python code, which was believed to be the best language for architecture and further extension. Modules are expected to speed up by replacing them with C/C++.

JF-V1.0 has included all the state-of-the-art methods in chapter 2 and chapter 3, so it is intrinsically event-driven. All physical collisions and operations like sampling, direction switching, and even the start and the end of the program, are treated as events. The program simulates in a manner of event flow processing, which we believe is the most compatible way to incorporate future developments.

JF is available on GitHub (repository `https://github.com/jellyfysh/JeLLyFysh`) with all source code open under the license GNU GPLv3.

4.1 Issues of application development

The JF application is more than a program to generate data shown in the previous chapters. Not only is it able to realize more functionalities in a single framework, but it also has to consider the possibility of user-customized modules in various contexts. Specifically, we list the considerations of scientific application in the following, though none of them is independent.

1. Structure. In order to incorporate multiple mandatory or optional functioning modules, a highly efficient structure [99] is required in analogy to a processing line. Different functioning parts must be written in separate modules, and we realized it for JF by different python packages. Modules to store the current configuration, distribute events, calculate event times, select the next event, and output something, all have to have their own relatively self-contained packages. In JF we adopt the "mediator" design pattern (see [99], chapter 5) to communicate data between modules.

2. Reusability. Any written code is supposed to function for the long term. To achieve this goal, one must make up a standard of code and has a mechanism to regulate extensions. In JF each package contains an abstract class that explicitly defines all mandatory interfaces for any implementation. Once a module is completed, a corresponding unittest must be provided so that errors are easily detected in future modifications.

3. Extendability. As mentioned above, we have only implemented a couple of realistic simulation examples and leave its vast potential for the future. Object-oriented programming languages are capable of meeting this requirement, and python is a modern object-oriented language. The majority of the modules in JF are coded as python classes. Future developers can extend any package by adding classes inheriting from an abstract class. Furthermore, users are allowed to create and customize the used set of packages by specifying configuration files. On the other hand, a simpler principle implies, in general, more compatibility. We treat all kinds of operations as events so that they can be realized by corresponding event handlers.

4. Readability. For future developers and contributors from outside, it will be beneficial to facilitate their involvement. Understanding several tens of thousands of lines is not easy. The aforementioned structure helps to simplify tedious specifications. We also enforce strict python coding conventions, including the naming of classes, modules, and variables. Also, docstrings are written for every function of every module of JF-V1.0 as an essential element of documentation.

5. Ecosystem. It is a big topic related to not only every aspect of the application itself but the way we manage it. One must maintain a good ecosystem for an application in order to attract more users and developers. All the aforementioned considerations are essential to a thriving ecosystem. Apart from them, we release JF on GitHub with source code open, and we will follow the working flow of GitHub to incorporate external contributions. We are aware that version control is crucial as well, though version 1.0 has only been released so far. License, code of conduct, and several quick manuals are provided in the main directory of the application.

4.2 Features of JELLYFYSH-V1.0

In JF-V1.0, we realized the modules that accomplish the working structure, provide basic settings, calculate basic events, and implement the long-range ECMC algorithms. The mediator pattern and the factory facility are expected to live for all versions. We designed a hierarchical representation of big molecular objects, and all current modules must respect such a protocol. As for the functioning parts, we have activators based on factor map and cells, event handlers that realize physical events and pseudo-events, potentials for the SPC/Fw model, and a heap scheduler. To support the state-of-the-art cell-veto method, the cell facility, the cell occupancy and corresponding event handlers are also provided. Inside the project, there are many other helping modules that we do not list here.

We hope to realize the following features in the future.

1. Parallelization. This will be done by allowing multiple active particles.

2. Multiprocessing support, to simultaneously calculate times for several candidate events.

3. Fast unconfirmed event processing. JF-V1.0 treats unconfirmed events as normal events that trash and recreate almost all other events. In fact, unconfirmed events can leave other events intact, which will significantly reduce the calculation burden.

4. Smart trashing and recreating schemes. For now, users must set up the lists for which events are trashed and which events are recreated.

5. Speedup via more efficient language like C/C++.

6. More packages of potentials and event handlers to adapt to various particle systems.

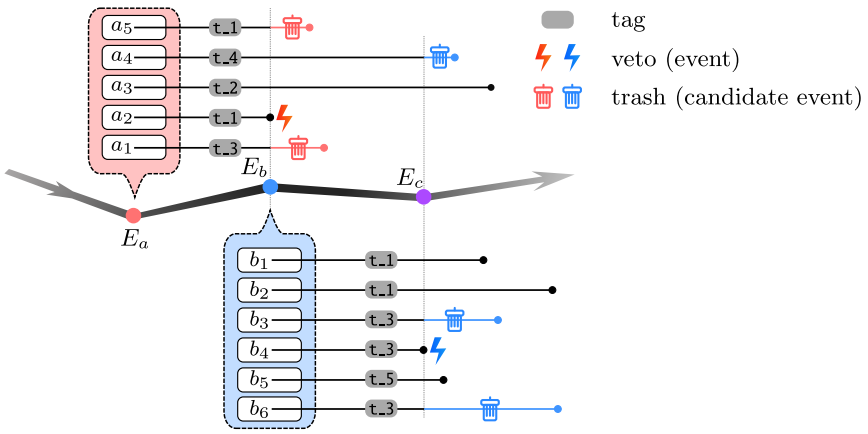


Figure 4.1: JF event flow. Events occurs at moments E_a, E_b and E_c selected as soonest of candidates. At each event time, some events are trashed while some are created. It refers to specific factors ($\{a_1, a_2, ..., a_5\}$, $\{b_1, b_2, ..., b_6\}$) in order to generate new candidate events with tags `t_1`,..., `t_5`. (From Ref [2].)

4.3 Basic concepts

4.3.1 Event flow

ECMC advances the configuration in a fixed direction and deflects it with lifting events. We devise JF as a processing line running with all kinds of events and treat auxiliary operations such as sampling in the same manner as treating physical collisions, except that no lifting move occurs at the sampling moment. We describe JF's event flow as Fig. 4.1.

Besides physical events, those operational events are called pseudo-events. They do not change the direction of configuration advances, as shown in Fig. 4.2. Moreover, we associate each pseudo-event with a triggering pseudo-factor. All events have event times, indicating when they occur. States restricted to its factor or pseudo-factor, before and after the event time, are called the event's in-state and out-state. Also, an event has a tag indicating its nature. The program must select from candidate events the soonest one and calculates its resulting operation. Then, it removes and recreates some events according to a specific scheme. We will describe this procedure in section 4.4.1.

4.3.2 Units

We denote particle objects simulated in JF as point masses. Each point mass has a position, a velocity, an identifier, charges, and a time stamp as properties. Thus we prepare `Unit` class with those five attributes. Later, when introducing the tree structure, the unit is also used to represent artificial center-of-mass objects (which are not point masses) in the tree's non-leaf node. Units are universal in JF acting as information carriers shipped among modules; thus, most functioning parts have to extract, process, and create units.

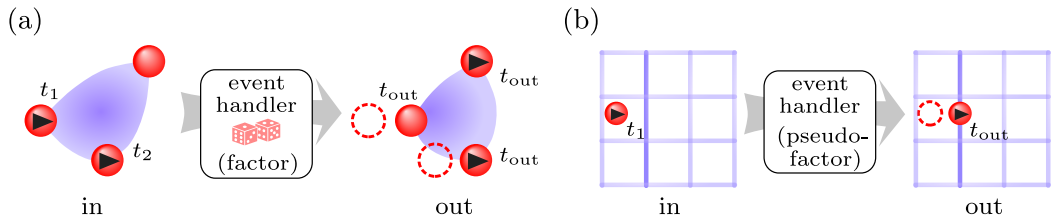


Figure 4.2: Example of JF events. (a) Event of a three-body factor. Lifting variables change when the event happens. (b) Cell-boundary crossing as a pseudo-event, in which only the internal state has to change. (From Ref [2].)

A similar term in JF especially in the tree state handler (see section 4.3.3) is called particle that has only position and charge. As the number of active point masses is much smaller than the total number of point masses, the tree state handler stores them in separate modules. Hence the position and charge of a unit independent of its lifting variable will be stored as a `Particle` instance.

Given the information of a unit with position $\mathbf{s}$, velocity $\mathbf{v}$ and time stamp t_0, its position can be inferred by

$$\mathbf{s}(t) = \begin{cases} \mathbf{s}, & \text{if } \mathbf{v} = \texttt{None}; \\ \mathbf{s} + \mathbf{v}(t - t_0), & \text{if } \mathbf{v} \neq \texttt{None}. \end{cases} \tag{4.3.1}$$

Eq. 4.3.1 is only sensible in an event-driven approach. The effect of an event usually includes time-slicing involved units, i.e., to update the position and time stamps up to the event time. Units that do not belong to the triggering factor are allowed to have time stamps lagging behind the most recent event. For example, sampling events and end-of-chain events have to time-slice all active units to the event time. The cell-boundary event, however, time-slices only the active unit. We have to note that whether to time-slice and which units to be time-sliced become more complicated in the presence of the tree structure in order to respect consistency.

4.3.3 Global state and internal state

The global state stands for the configuration of the point masses and related barycenters. JF-V1.0 treats configurations via a tree structure of connected nodes, with leaf nodes hosting a real point mass and upper nodes holding artificial objects that represent barycenters of their direct child nodes. A barycenter node together with all its child nodes, is treated as a composite object in JF-V1.0. Thus, a composite object has all information appearing in point masses, including position, velocity, identifier, charge, and time stamp. The position and velocity of a non-leaf node unit are calculated simply via the average of units of its direct children, and the program keeps in the state handler (see section 4.4.4) a list of top nodes in order to iterate over them. Fig. 4.3 (a) and (b) show an example of a tree representation of several units and their velocities.

The identifier adopted in JF-V1.0 comes from its order in the tree. The state handler numbers root nodes, and nodes number their child nodes. The identifier of a unit

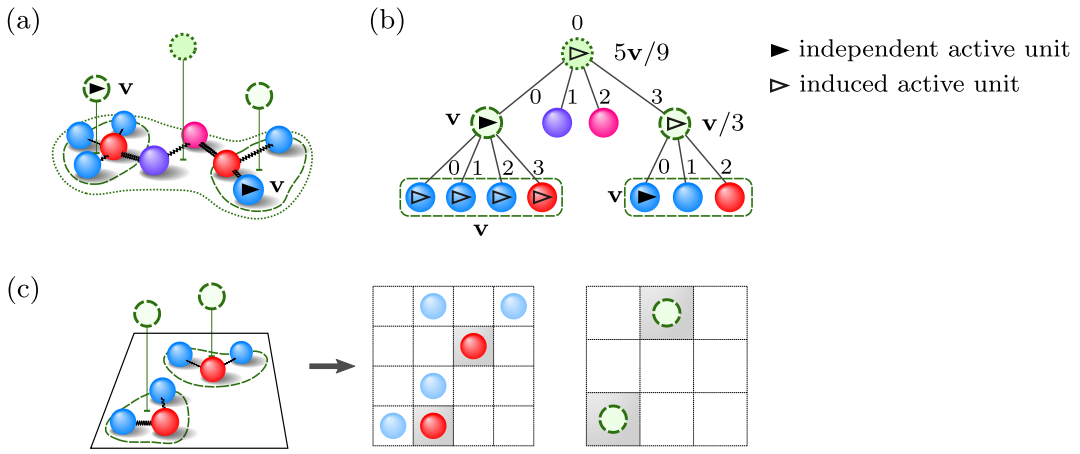


Figure 4.3: Global state and the internal state of JF. (a) Point masses with composite objects. Two composite objects are for the barycenter of molecular components, and a top composite object represents the whole molecule. (b) The tree structure for the molecule in (a). The top node and children of all nodes are numbered so that identifiers can be handily acquired by listing numbers along a path. Solid arrows reflect velocities in (a) while hollow arrows are the induced velocities. (c) Cell occupancies as examples of internal states. JF can simultaneously be equipped with multiple cell occupancies. Here, one is for point mass units, and the other is for composite object units. (From Ref [2].)

is the tuple with all the numbers along the path towards it.

An active point mass moves by itself, inducing its parent unit to move, and all its ancestors gain activities. On the other hand, a composite object unit can move at velocity $\mathbf{v}$ as well and makes all descendants within the subtree move at $\mathbf{v}$. The latter case is realistic, and can be molecule motion as a whole. Thus we introduce the term of an independent active unit and an induced active unit. The unit for an original moving point mass and composite object unit for molecule moving as a whole are regarded independent, while their ancestors and children acquiring velocity belong to induced active units. Fig. 4.3 (a) and (b) provide examples of independent active point masses and composite objects, as well as induced active units.

Apart from the global state, JF defines an internal state to maintain useful information concerning the global state. For example, cell occupancy makes use of a cell setup, and updates unit-cell correspondence according to the current global state, as shown in Fig. 4.3 (c) . Thus the internal state is partial information of the global state. JF keeps internal states in the activator (see section 4.4.3), and there can be multiple internal states created for one simulation.

An internal state has two mandatory methods. `update` is called during mediator step 2, and it uses the current active state to update the internal state. For those internal state changes without lifting moves, we must regard them as extra events. Internal states can be accessed with indices, so its `__getitem__` method is required.

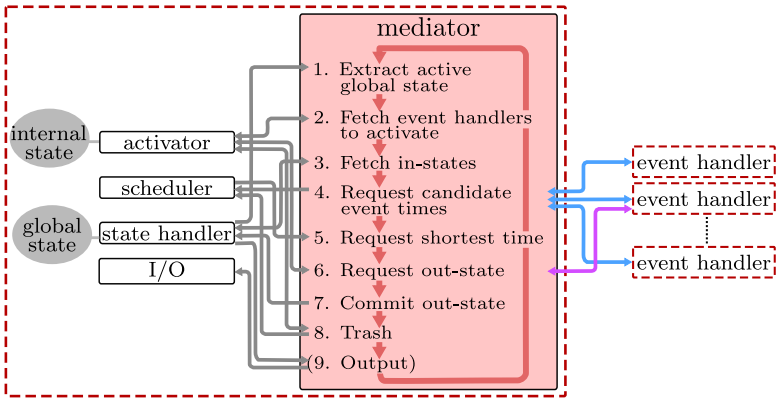


Figure 4.4: JF-V1.0 architecture and the main loop inside the mediator. Boxes indicate modules of JF, each contained in a python package. Arrows imply directional data communication (requests sent from the mediator are not deemed as data). Red dashed boxes represent processes of our multiprocessing trial. Steps inside the mediator describe the main loop's procedure, with each iteration being a leg in Fig. 4.1. (From Ref [2].)

4.4 Essential modules

Simulating particle systems demands the coordination of many functions. JF distributes them in various modules, and in JF-V1.0, it has a mediator, an activator, a state handler, event handlers, a scheduler, and an input-output (IO) handler. We depict their communication in Fig. 4.4 featuring a clear mediator design pattern whose topology takes a star shape. It means that the mediator stands in the center and connects all the rest while data exchanges between peripheral modules are forbidden.

4.4.1 Mediator

The mediator is the hub of all activities of JF. It provides a `Mediator` class mandating the `run` and `post_run` methods that will be called by the executing script. The main loop of event-driven ECMC is hosted in the `run` method. And also, the mediator is attached with event-handler-specific methods to get arguments for an event's out-state (methods with name `get_argument` + `event_handler_name`) and to do any operations at the event moment (methods with name `mediate` + `event_handler_name`).

We implemented the `SingleProcessMediator` and the `MultiProcessMediator`. The main loop of the `SingleProcessMediator` consists of nine steps:

1. The mediator requests the state handler for an active global state, whose format can vary according to different state handlers.

2. It sends the active global state to the activator. The activator returns a list of factors (indicated by event handlers in JF-V1.0) and involved unit identifiers.

3. It fetches the in-state from the state handler. Identifiers returned by the activator are replaced by corresponding units.

4. The mediator calls the method `send_event_time` of event handlers obtained in step 2 with the in-state arguments acquired in step 3. Then it pushes all returned candidate event times into the scheduler.

5. It asks the scheduler which event handler produced the nearest event.

6. The mediator calls the `send_out_state` method of the event handler that produced the smallest event time, optionally with arguments obtained by the mediator's event-handler-specific `get_argument` method that may inquire the activator.

7. The out-state is sent and recorded in the state handler.

8. The mediator asks the activator for event handlers to be stopped and removes them from the scheduler.

9. The mediator requests the entire global state from the state handler and sends it to the IO handler for measurement. This step is only executed for event handlers associated with the output handler.

This loop continues until an `EndOfRun` exception occurs. The `post_run` method is invoked afterwards.

The `MultiProcessMediator` makes use of python's `multiprocessing` package. It distributes the job of each event handler to one long-lived process and designates a two-way pipe (`multiprocessing.Pipe` object) to each event handler. The mediator and other modules are executed through another process, indicated by the red dashed line in Fig. 4.4. Only the mediator needs to be changed when switching from single processing to multiprocessing, and all event handlers are automatically modified in two ways. First, a `run_in_process` method is added to loop over the four states shown in Fig. 4.5; second, the `send_event_time` and `send_out_state` methods are decorated so that the only argument is the pipe connecting to the event handler, and that the in-state, the resulting out-state, and other arguments are transferred via pipes.

We split an event calculation into an event time and the following out-state. An event-handler process may stay in one of the four states during the Markov chain: `idle`, `event_time_started`, `suspended` and `out_state_started`. The event-handler process takes actions under the control of a multi-process mediator through shared `multiprocessing.Event` objects. A start event and a continuing event (this event is not the event in ECMC) are set up for an event-handler process to undergo stages of event computation in order. In addition, our multi-process mediator takes into consideration the available number of CPU cores. If N_c cores are designated for simulation, $N_c - 1$ are allowed to run at any moment in order to avoid task-switching cost, which we realize through a `multiprocessing.Semaphore` instance. During stage 4 that requests candidate event times, the mediator prioritizes the calculation of event times of all active event handlers while suspending out-state calculations. When a CPU core is idle, one of the event handlers' `out_state_started` events is released, and the CPU core can go on with its out-state calculation. This coordinating scheme can, in principle, optimize the usage of CPU resources while it suffers from a computational bottleneck in practice.

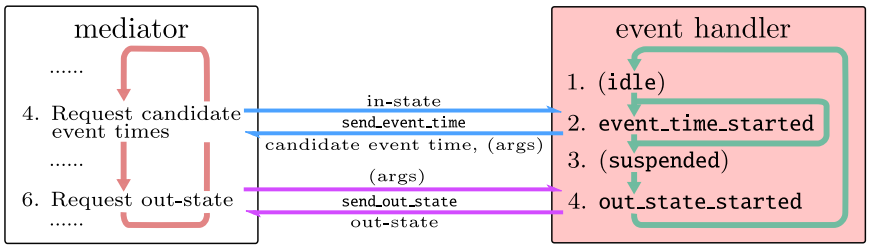


Figure 4.5: Stages of event handlers. Two mandatory methods `send_event_time` and `send_out_state` are called by the mediator during the stage of requesting candidate event times and requesting the out-state. `idle` and `suspended` stages are for multiprocessing, and actually they are labels in the multi-process mediator. (From Ref [2].)

4.4.2 Event handler

Since all physical lifting moves, artificial direction switching, and operational state recording are treated as events in JF, event handlers play a versatile role throughout JF's event flow.

The abstract class `EventHandler` mandates `send_event_time` and `send_out_state` methods. An in-state usually accompanies the former. As their names suggest, they return event times and out-states respectively, while out-states may not always be requested unless the event handler is chosen as the nearest event. Once `send_event_time` is called, the event handler calculates the event time, stores temporarily the state, and waits for the request of out-state. The mediator is responsible for making the proper invoking order, e.g., to avoid calling `send_out_state` consecutively, as shown in Fig. 4.5.

Some event handlers have no in-state such as the sampling event handler. However, some need more information to produce the out-state. For example, an end-of-chain event handler requires the current state at the event time, so the mediator sends it as an argument of out-state. A cell-veto event handler also requests the occupant information of the target cell by accompanying the event time with a cell index. It is important to note that all state updates are computed in event handlers. In particular, event handlers are responsible for maintaining the consistency of the hierarchical state representation.

Most event handlers do calculations in company with helping modules. Factor event handlers need to connect to potentials where physics is contained, and `potential` becomes a JF package that will be detailed in section 4.5.4. Customized event handlers and potentials reflect the variety of particle simulation reality.

4.4.3 Activator

Fig. 4.6 depicts the working of JF's activator. It does three jobs: 1. The method `get_event_handlers_to_run` searches relevant factors for given active units; 2. The method `get_trashable_events` provides the list of event handlers to be removed after an event; 3. It maintains internal states, and returns its information via the method

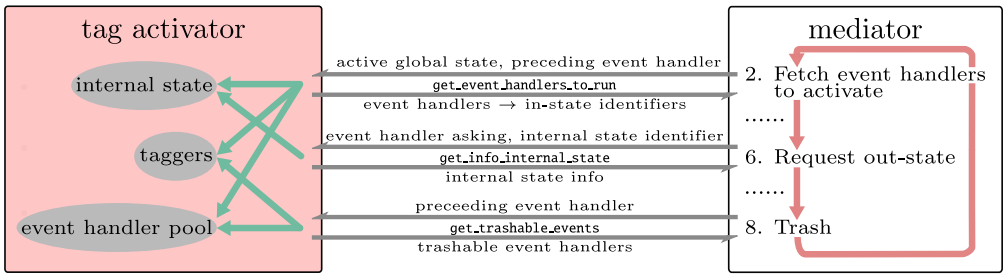


Figure 4.6: Inner state of tag activator and its interaction with the mediator. Tag activator maintains one or more internal states, a list of taggers and a event handler pool. It communicates with the mediator through three interface functions. Arguments and return values of the functions are shown above and below the method names respectively. (From Ref [2].)

`get_info_internal_state`. Here in JF-V1.0, a relevant factor is represented by a list of unit identifiers plus an event handler instance. To this end, the activator requires the proceeding event handler. Even though complete information on active units is sent to the activator, it keeps partial information of the global state in internal states.

In JF-V1.0 we implemented the class `TagActivator` which categorizes events with tags. Events with the same tag are deemed as the same nature, and they will be created or trashed indiscriminately. For example, while simulating Coulomb atoms via cell-based bounding potential, sampling, end-of-chain, cell-bounded Coulomb, excluded cell, and surplus particles appear as tags in order to process events in a simplified way. JF-V1.0 creates each type of event through `Tagger` instances, depending on how to find involved units for the event type. For instance, the `NoInStateTagger` class produces candidate events without seeking any in-state; a `FactorTypeMapInStateTagger` instance is equipped with a factor map specifying interactions between atoms according to their identifiers; cell-related taggers have the responsibility to invoke the update of the cell system, a typical internal state, and acquires the units of in-state by inquiring it.

Besides numerous ways to obtain in-state identifiers, taggers are specified with a multi-dispatching table detailed in the configuration file in JF-V1.0. For each tagger, a trashable list and a creating list of events are provided, then once an event occurs, the activator looks through the trashable list and returns all running event handlers with tags in it (that is why the proceeding event handler is necessary). It creates new candidate events in the same way. Another two lists are activating and deactivating lists which are exclusively used for root-mode motion in JF-V1.0. They allow events to deactivate and re-activate certain taggers, which provides a convenient realization of switching between atom motion and dipole translation. Also, the activator manages the event handler pool, and users must specify the number of event handler instances in JF-V1.0.

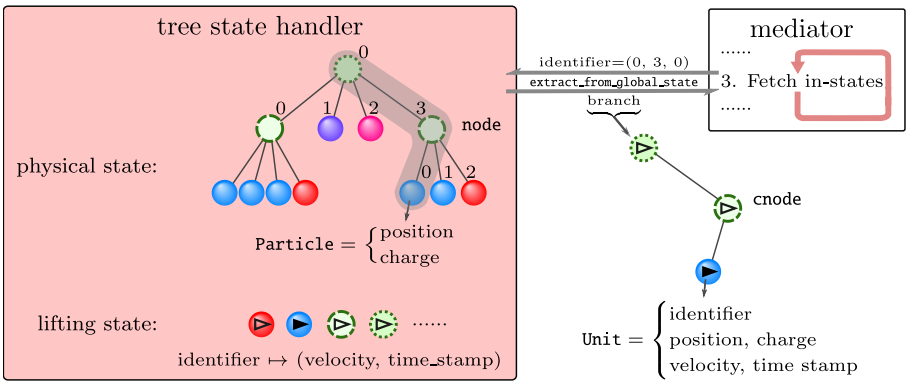


Figure 4.7: Fig:Inner mechanism of tree state handler, with an example of `extract_from_global_state`. The whole state is stored separately as physical state and lifting state. When the method is called with an argument `(0, 3, 0)`, a new branch is created with new `unit` instances attached on it and it is passed back to the mediator. (From Ref [2].)

4.4.4 State handler

A `StateHandler` instance not only stores the global state, namely the physical configuration plus lifting variables, but also rules the protocol of communication among the entire JF. The other modules must act in accordance with the output format of the specified state handler, and event handler return out-states that are acceptable by the state handler. In JF-V1.0, we provide `TreeStateHandler` class whose output values are tree branches, so event handlers, activators, and IO handlers of JF-V1.0 must work with the format of tree branches.

Positions and charges of units are taken as physical variables, and they are stored in an instance of the `PhysicalState`. The velocity and the time stamp are kept in a `LiftingState` instance. As the number of active units is limited, the storing strategies for them are usually different. For the `TreeStateHandler` that implements the tree structure described in section 4.3.3, it uses `Node` objects to construct trees in its `TreePhysicalState`, whereas it adopts a simple python dictionary to store non-zero elements of the lifting state. The instance of `TreePhysicalState` keeps the list of root nodes of composite objects, and each node has links to all direct descendants and parent node. Node information, instances of the `Particle` class, is kept in the `value` attribute. An example of the internal storage of the tree state handler is shown in Fig. 4.7. Both the lifting state and the physical state provide basic methods like `set` and `get` to update and output information.

The state handler has four abstract methods in total. `extract_from_global_state` returns the desired state for a given identifier. For the global tree state, we rule that the returned value is the "branch" of the corresponding node with all ancestors and descendants, and with units attached to each node. Fig. 4.7 illustrates the interface of this method. Another method `insert_into_global_state` is used to commit events' resulting out-state. The tree state handler accepts a sequence of root nodes of branches, then makes corresponding changes inside. During mediator step 1,

the method `extract_active_global_state` is called, in which information of active units (for tree state handler, it becomes branches of independent active units) are returned and passed to the activator. Finally, the `extract_global_state` method outputs the entire global state for the purpose of measurement that output handlers will perform.

It is important to note that the tree state handler reconstructs branches to send to the outside, using newly instantiated nodes and units. This convention means that outside modules have no access to the state handler's inner storage, and that is why the name `cnode` appears elsewhere in JF.

4.4.5 Scheduler

JF's scheduler has relatively simple functionality compared to other modules. The abstract class `Scheduler` has three methods. `push_event` is used to input an event handler together with its event time. `get_succeeding_event` returns the event handler instance with the smallest event time within the scheduler, and `trash_event` removes a certain event handler from it.

A heap structure is capable of treating all aforementioned requests with complexity $\mathrm{O}(\log N_E)$, where N_E is the current number of event handlers inside the scheduler. In JF-V1.0 we implemented the `HeapScheduler` class via the python package `heapq`. Actually, we found later that a naive list is faster since the number of candidate events is usually very small, while the `ListScheduler` class does not appear in JF-V1.0.

4.4.6 Input-output handler

The input-output handler is equipped with an input handler and an arbitrary number of output handlers. Each run of JF can specify one input handler and multiple output handlers. The class `InputOutputHandler` connects the mediator directly, and has `read` , `write` and `post_run` methods to read from the input handler, write to a specific output handler, and close output handlers.

The input handler is called only once at the beginning of the Markov chain, in order to prepare the initial configurations. Any implementation of an input handler needs to specify its `read` function. In JF-V1.0, we provide two input handlers. The `RandomInputHandler` class randomly generates an initial state, and the `PdbInputHandler` reads the state from a `.pdb` (protein data bank) file. This class depends on the `MDAnalysis` package.

Output handlers are diverse, including measurement, state recording, memory dumping, etc. They only have to implement the `write` method to generate output data and `post_run` method to close files. We provided in JF-V1.0 separation output handlers calculating atom separations for a configuration, the pdb output handler to write to `.pdb` file, one output handler to calculate $\mathrm{H_2O}$ molecule bond angle and bond length, and one to dump the entire program state via the `dill` package.

All input and output handlers work with tree representation. The input handler returns a list of trees with nodes attached with `Particle` objects. Output handlers get the global state organized as trees and have to resolve them.

4.5 Important tools

Several modules of JF-V1.0 are not crucial to its architecture, but are essential techniques of large-scale ECMC simulation or are necessary to provide modules with required information. The `setting` package and the `Initializer` class are indispensable to run any application. The `Cell`, `CellOccupancy`, `Estimator`, `Potential` and `LiftingScheme` classes are closely related to the ECMC algorithm, so we explain here briefly their implementation.

4.5.1 Globally used modules

The information of the simulated system, e.g., the temperature and the box side length, is kept in an independent package called `setting`, and actually, it leads another configuration tree (see section 5.2) other than the main architecture represented by the mediator.

The basic setting parameters include β, dimension, number of root nodes and so on (see file `setting/__init__.py`). They all live in the namespace of `setting`. A specific setting may have its own variables (e.g., the side length of a cubic setting), but it must give values of those in basic parameters. We implemented in JF-V1.0 the `HypercuboidSetting` and the `HypercubicSetting`, each with an additional `system_length` variable. Basic parameters have their copies under those specific settings, too. Also, by defining hypercuboid setting as a similar setting of the hypercubic one, the program can automatically determine values for the hypercuboid one. For example, once we set `beta` to be β and `system_length` to be L in hypercubic setting, β will also exist as `setting.beta` and `setting.hypercuboid_setting.beta`, and L will also be in `setting. hypercuboid_setting.system_lengths` as a tuple of D identical values.

We place functions related to position and separation calculations in implementation of the `PeriodicBoundaries` class, whose instance is an attribute of both basic setting module and specific setting, too. For example, method `separation_vector` that maps an arbitrary vector onto its nearest image, may be in `setting.hypercubic_setting.periodic_boundaries` and `setting.periodic_boundaries`.

Besides, some other commonly used functions, such as vector operations, are in `base` directory.

4.5.2 Cells and cell occupancy

The cell system is important for fast accessing the active particle's interacting pairs, and is also basic to Walk's method to sample the displacement for long-range systems (see section 3.3). JF-V1.0 regards cell occupancy as an internal state and takes cells as an attribute of cell occupancy. As exemplified by Fig. 4.3 (c) , a JF simulation is allowed to have more than one cell and cell occupancy instance.

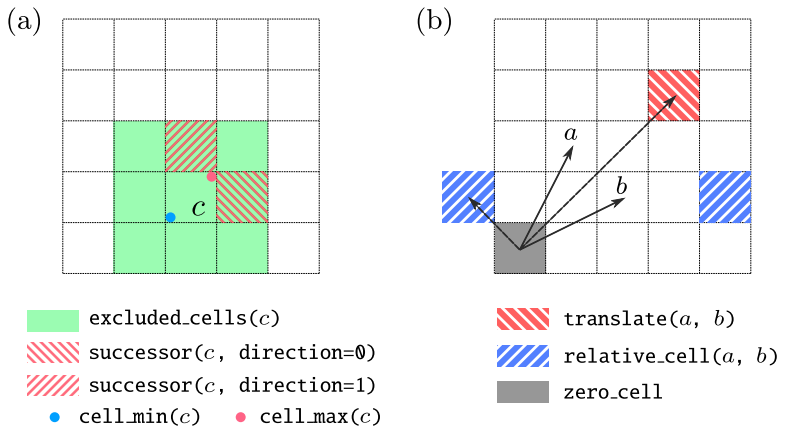


Figure 4.8: Methods of cell system. (a) `excluded_cells`, `successor`, `cell_min` and `cell_max` for the cell labeled with c. Horizontal and vertical directions are indexed as 0 and 1. These methods are required by `Cells` abstract class. (b) `translate`, `relative_cell` and `zero_cell` of `PeriodicCells` class. Two blue-striped cells are identical due to periodic boundary conditions. (From Ref [2].)

4.5.2.1 Cells

A cell instance defines the geometry of the cell division, including not only the position and the shape of each cell but their translational relation. There are six abstract methods within the `Cells` class. `yield_cells` is used to iterate all cells. Given one cell, `excluded_cells` returns the set of cells that have to be treated exceptionally[1]. The `position_to_cell` method, as the name suggests, returns the cell identifier for a given point. This method has been unambiguously implemented that any computer representation of float-number coordinates within the simulation box range is mapped to one cell[2]. `cell_min` and `cell_max` give points with minimum and maximum coordinates respectively within the cell, though it applies to only hypercuboid cells[3]. Finally, the `successor` method gives an adjacent cell in a certain direction. Fig. 4.8 (a) shows an example of the cell methods.

In addition to `Cells`, child class `PeriodicCell` mandates methods `translate`, `relative_cell` and `zero_cell`. `zero_cell` is the exact origin of the cell system used for other methods, though JF-V1.0 applications never call it. Method `translate` calculates the cell obtained by translating one cell by another cell's vector, which is especially useful in the cell-veto algorithm. Likewise, `relative_cell` returns the relative vector of cells a and b, which can also be expressed as a cell. The aforementioned methods make sense only when the system exhibits periodicity, and an example is shown in Fig. 4.8 (b) . A typical implementation of periodic cells is cuboid cells with homogeneous division, realized in `CuboidPeriodicCells`.

[1] Exceptional cells depend in principle on the potential, while in JF-V1.0 users have to set up in configuration files how many exceptional layers are around the current cell.

[2] It is not like a continuous real number. Two adjacent float numbers always have a tiny gap due to finite-bit representation.

[3] Strictly speaking, they return finite-bit floats. Otherwise, the maximum or minimum may not exist in real number set that is not closed.

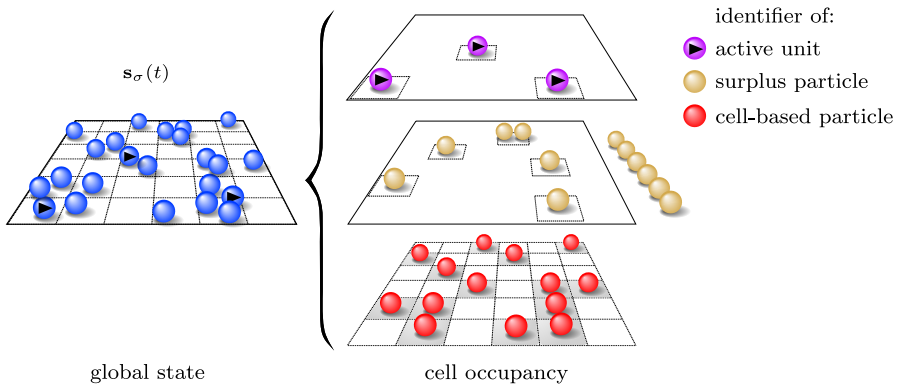


Figure 4.9: Cell occupancy classifies the global state into cell-based particles, surplus particles, and active units. (From Ref [2].)

4.5.2.2 Cell occupancy

JF cell occupancy comprises occupants, surplus units, and variables for active units and cells, as shown in Fig. 4.9. Information on the current active units is passed from the state handler via the mediator to the activator. The activator analyzes active unit information plus the proceeding event in order to update the internal states. The change of internal state leads to additional pseudo-events such as cell-boundary crossing.

The abstract class `CellOccupancy` keeps the cell occupancy information with the `yield_surplus` and `yield_active_cells` methods, and the `cells` and `cell_level` properties. Property `cells` returns the underlying `Cells` instance, and `cell_level` gives which level of units are taken as occupants. We implemented `SingleActiveCellOccupancy` in which one or zero active unit is present. It can optionally take into account a charge value specified during instantiation, so that only units of non-zero charge are registered as occupants. It keeps the occupants in a list mapping cell indices onto occupants or `None`, and surplus units are stored in a dictionary mapping cells onto a list of surplus units.

4.5.3 Initializer

The JF application starts with a factory (see section 5.2) to create all necessary instances according to the customized configuration file. The mediator's `run` method leads the main running body. Between instantiation and simulation, some modules need to be initialized because we find that several initializing operations must be done after all instances are well prepared. For example, an input handler has to read in the initial physical configuration, and install it within the state handler and the activator. Also, a cell-veto event handler requires charge information of particles in order to set up the cell-veto event rate.

We make JF perform those operations through the `initialize` method of required classes, and these classes must inherit from the `Initializer` class (thus multiple inheritances must apply). Instances of `Initializer` will freeze all their public methods

right after instantiation, but thaw them once the `initialize` method is called[4]. Although the purpose of initialization is straightforward, JF-V1.0 undergoes an obscure procedure to call all required `initialize` methods[5]. Hence it is not users' business to customize initializers, but it is for developers.

4.5.4 Potentials, estimator

We place physical potentials in a separate directory and have two abstract classes `Potential` and `InvertiblePotential`. All potentials must implement derivatives, and an invertible potential has to, in addition, provide a displacement for certain energy change. We designed them in a straightforward style. `derivative` takes one or more separation vectors and a direction of motion, and `displacement` requires potential change in addition. Users are encouraged to create any potential that is needed for simulation.

The estimator is used to obtain an upper bound and a lower bound within a hypercuboid area of given potential. This estimation can be achieved by taking evenly or randomly a large number of points inside the region. For point-point interaction, we implemented the `BoundaryPointEstimator` and the `InnerPointEstimator` which draw points evenly from the surfaces and from the entire hypercuboid respectively. For point-dipole interaction, two-point potentials are only provided, so the estimator must know the typical dipole moment and take all possible orientations into account. The `DipoleMonteCarloEstimator` samples a large number of dipole positions and orientations randomly, then takes their maximum; The `DipoleInnerPointEstimator` draws s^D points from a D-dimensional hypercuboid with s points per dimension, aligns an imaginary dipole along the direction of the potential gradient and takes the maximum from all positions, which proves more efficient than the Monte Carlo estimator. It is important to note that no method can ensure optimality in estimating an arbitrary function. Thus a prefactor can be specified, and the result will be multiplied by it. Moreover, an estimator allows the user to provide an empirical bound to rule out possible unphysical results. If the estimated result is greater than the empirical bound, the empirical bound is returned instead.

4.5.5 Lifting schemes

Lifting schemes described in section 2.2.2 serve as an independent package and are invoked by many-body event handlers. We provide in JF-V1.0 `RatioLifting`, `InsideFirstLifting`, and `OutsideFirstLifting` as three implementations of the `Lifting` abstract class. The base class has already realized methods for resetting and inserting new particles as well as their derivatives (and aligning them into negative and positive tables). Specific lifting schemes have to provide ways to select the next active particle among the negative table in the method `get_active_identifier`.

[4]Strictly speaking, one must call `initialize` of the base class, so `super().initialize` has to be called in child class.

[5]Initialization of the state handler is called in `SingleProcessMediator.__init__`. For the activator, it is called in `MediatorAbstractClass.__init__`. And for internal state, they are called in `Activator.initialize`. For event handlers, they are initialized by corresponding tag activators.

Chapter 5

JELLYFYSH simulation

In this chapter, we prepare realistic applications of JF with the help of concrete instances. We show how to write configuration files for methods introduced in chapter 3. By running real simulations, users will understand the functions and the mechanism of various event handlers, and how event handlers treat each event in a two-step manner. Taggers have many types, but fortunately, taggers that have already been developed satisfy the requirement of most particle simulations. The potential package will be supplemented with real potentials and bounding potentials.

5.1 Event handlers of JELLYFYSH-V1.0

5.1.1 Event handler for two-body invertible potentials

The `TwoLeafUnitEventHandler` class is the most typical event handler that inverts a two-body physical potential. Its instance is created with a invertible potential (an instance of the `InvertiblePotential` class) and the relevant charge name.

The design of this class complies with the tree structure of the internal state. When its `send_event_time` is called, it at first resolves the in-state in the form of a sequence of branches to obtain lists of leaf nodes and leaf units. Next, it identifies the active node and the active unit, as well as other information like the direction of motion and the speed. The random number corresponding to the energy change is generated within the `send_event_time` method, too. Finally, it calls the potential's `get_displacement` method with the separation between the active point mass and the non-active one to acquire the displacement s. A new event time t is returned by adding the corresponding traveling time to the active unit's time stamp.

For `send_out_state`, it just time-slices the active unit to t by advancing its position and time stamp, then exchanges the velocities and time stamps of two leaf units though they are `None` for the non-active unit. A specific series of methods is provided in the base classes to keep variables consistent along the trees in a register-and-commit manner. The entire procedure is typical for all event handlers.

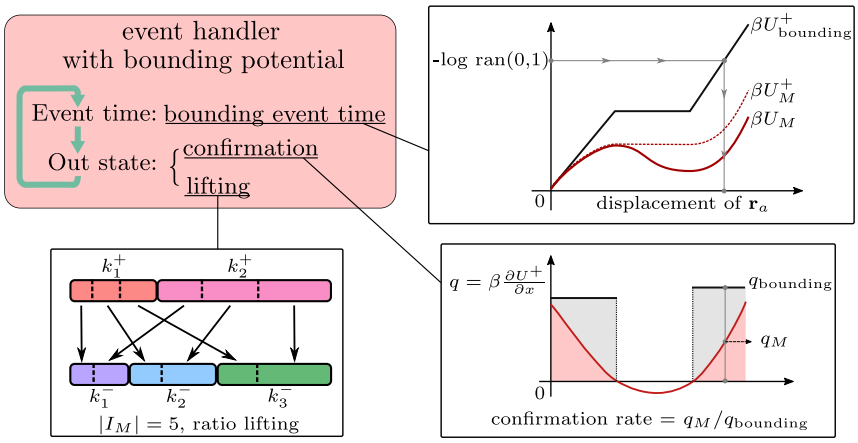


Figure 5.1: Schema of event handler with bounding potential. Displacements concerning only bounding potentials are drawn during `send_event_time`. Event confirmations and lifting changes are done within `send_out_state`.

5.1.2 Event handlers for non-invertible potentials

For non-invertible-potential event handler, the addition to an invertible-potential event handler is a bounding potential and a confirmation step (see section 2.3.2). In JF-V1.0, we provided three kinds of bounding potentials: by a direct invertible potential, by a cell-based potential with piecewise-constant derivatives, and by speculated potential with piecewise-constant derivatives.

5.1.2.1 Direct bounding potential

An `InvertiblePotential` instance as bounding potential must be specified for the `TwoLeafUnitBoundingPotentialEventHandler`, both for two-body interactions. In `send_event_time`, it samples a displacement under the bounding potential and returns the corresponding event time.

An event confirmation takes place only after the `send_out_state` method is called. The method advances point masses like in the invertible case, then confirms the event with the ratio given by Eq. 2.3.7. If the event is confirmed, an out-state will be returned with exchanged lifting variables; otherwise, velocities and time stamps remain the same while the active unit is advanced[1]. The procedure is shown in Fig. 5.1.

[1] Treating unconfirmed events in the same way as confirmed events leads to inefficiency especially in cell-veto events. Later improvements include letting out-state of unconfirmed events be `None`, and the mediator repeats requesting event times and out-states until concrete out-state is returned. See section 6.1.

5.1.2.2 Cell-based bounding potential

Even if an analytical bounding potential is hard to provide, JF is able to define one by setting derivatives to local maximums. Specifically, for any pairwise potential U

$$q_U^{\mathrm{cb}}(\mathbf{r}_1, \mathbf{r}_2) = q_U^{\mathrm{cell}}(C_1, C_2) \geq \max_{\mathbf{r}_1 \in C_1, \mathbf{r}_2 \in C_2} \left[\frac{\partial U(\mathbf{r}_1, \mathbf{r}_2)}{\partial x_1} \right]^+. \tag{5.1.1}$$

Obviously it is constant for any position pair with $\mathbf{r}_1 \in C_1$ and $\mathbf{r}_2 \in C_2$. $q_U^{\mathrm{cell}}(C_1, C_2)$ is usually simplified as $q_U^{\mathrm{cell}}(C = C_2 - C_1)$ for periodic cells. Users must provide with an `CellBoundingPotential` instance which is instantiated with an estimator, in order to calculate q_U^{cell} during initialization.

A process similar to section 5.1.2.1 with a cell bounding potential is performed with q_U^{cb} as its bounding event rate. We implemented in JF-V1.0 the `TwoLeafUnitCellBoundingPotentialEventHandler` class for atom factors and `TwoCompositeObjectCellBoundingPotentialEventHandler` class for dipole factors. Both their event handlers must be used in company with cell-boundary events (instance of the `CellBoundaryEventHandler` class) to incorporate internal state changes. The displacement sampled for the event time may drive the active unit out of the current cell.

5.1.2.3 Dynamic bounding potential

A bounding potential can also be constructed by speculating derivatives for a distance in front of the active unit. A simple speculation for two-body potentials U can be

$$q_U^{\mathrm{spec}}(\mathbf{r}_{12}) = \max\left\{q_U(\mathbf{r}_{12}), q_U(\mathbf{r}_{12} - \mathbf{v}_1 \Delta t)\right\} + c_s, \tag{5.1.2}$$

where Δt is an adjustable speculation time and c_s is a positive offset. This formula is able to bound the potential with relatively small fluctuation, and is implemented in the `TwoLeafUnitEventHandlerWithPiecewiseConstantBoundingPotential` class. Similarly, the `FixedSeparationsEventHandlerWithPiecewiseConstantBoundingPotential` class is given for many-body potentials. We find in practice that they provide efficient bounding potentials with very few exceptions for the Lennard-Jones potential in Eq. 3.6.7 and the bending potential in Eq. 3.6.6.

5.1.3 Cell-veto event handler

Cell-veto event handlers deal with a set of two-body potential factors with the cell-veto algorithm described in section 3.3. In JF-V1.0 we implemented `LeafUnitCellVetoEventHandler` for atom factors and `CompositeObjectCellVetoEventHandler` for dipole factors. They both feature an elaborate communication with the mediator, in the process of the event time and out-state computation.

The in-state for a cell-veto event handler is not the entire global state, but rather the branch for the active unit alone. A cell-veto event handler samples the event time through exponential random distribution with the rate q^{tot} defined by Eq. 3.3.2, and draws the relative cell ΔC with a probability proportional to $q_U^{\mathrm{cell}}(\Delta C)$, which can be

done in constant time through Walker's method. Then it has to identify the cell C_1 that hosts the active unit and calculate the triggering cell by $C_2 = C_1 + \Delta C$.

The triggering cell identifier C_2 is sent back accompanying the event time, then the `get_arguments_cell_veto_event_handler` method is called in the mediator to inquire the internal state for the occupant inside C_2. The branch corresponding to C_2 occupant will be sent via arguments of `send_out_state`. The cell-veto event handler uses the global state for the active unit and the triggering unit to compute the real event rate q^{real}, and confirms a cell-veto event by $q^{\text{real}}/q_U^{\text{cell}}$.

A cell-veto event usually does not appear alone to treat long-range potentials. Surplus units excluded cells often exist. Surplus and occupants of excluded cells are, however, handled by normal two-body event handlers with a bounding potential. Direct two-body events, and factor events for an active unit interacting with surplus units and excluded cell occupants, differ in taggers that create them. Like the event handler with cell-based bounding potential, the cell-veto event handler works with a cell-boundary event handler to update the internal state.

5.1.4 Event handlers for composite object motion

Motions in ECMC are not restricted to single particle moves. section 2.2.1 has formulated ECMC in such a way that any set of general coordinates $\{\mathbf{a}_i\}$ can be chosen and the lifting schemes require only $\sum_i \mathbf{a}_i \cdot \nabla U = 0$. In dipole or water molecule simulations, we can select a part of $\{\mathbf{a}_i\}$ to be translations of one molecule along x, y or z axes, accompanied with molecular rotation or normal particle motion to guarantee ergodicity.

There is much to be studied concerning the factorization, lifting schemes, bounding potentials, and cell scheme for composite object motion. We leave it for the future, but gave a simple JF implementation for the moment. We prepare in JF-V1.0 event handlers to move a composite object as a whole and event handlers to switch between point-mass motion and molecular motion. The `RootUnitActiveTwoLeafUnitEventHandler` class is for the event handler to deal with two-body inter-dipole factors. Similar to the `TwoLeafUnitEventHandler`, it gets an in-state with two leaf units and calculates event times as in particle mode, but requests the entire branch for the moving molecule and the object molecule by `send_out_state` arguments, time-slices them and exchanges their velocities. The `RootUnitActiveTwoCompositeObjectSummedBoundingPotentialEventHandler` class is used to treat interactions between all intermolecular pairs with dipole factorization, with bounding potential consisting of separate pairwise bounding potentials (note that no many-body lifting schemes are needed because there are only two general directions involved).

We also provide the `RootLeafUnitActiveSwitcher` class as an event handler to switch between the leaf mode and the root mode. It exploits the event handler's function of activation and deactivation, which allows to freeze and reactivate specific taggers at event times specified by the user. One must create an event handler as a leaf-to-root switch and the other as a root-to-leaf switch. They deactivate themselves and reactivate each other to achieve alternating motions. The event handler for leaf mode will pick randomly one leaf unit from the active branch as the next active point mass, while the event handler for root mode will impose the velocity onto the whole

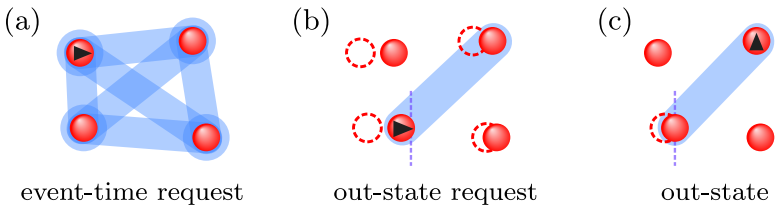


Figure 5.2: Workings of the end-of-chain event handlers. (a) Its operation can be regarded as being triggered by a set of pairwise pseudo-factors containing all particle pairs. The global state at the moment of calling `send_event_time` is not used to infer the out-state. (b) Global state when the mediator requests the end-of-chain out-state. One of the pseudo-factor triggers the event, and the relevant state is sent to the event handler. (c) The event-handler advances the currently active unit then passes the activity to a new unit, and changes to a new direction of motion. Then it sends back the out-state. (From Ref [2].)

molecule containing the active point mass.

5.1.5 Sampling event handler and end-of-chain event handler

The function of the sampling event handler is to draw samples of the global state at certain moments, and the end-of-chain event handler is used to impose artificial direction switches. Their logics both deserve sufficient consideration.

Taking `FixedIntervalSamplingEventHandler` as an example, it sets up an internal counter for sampling times by accumulating a constant interval and is instantiated with a specified output handler. The sampling event handler has no in-state, but at the event time, the active global state will be sent from the state handler to the sampling event handler via the mediator in order to push them all to the event time. After committing the out-state, the mediator dumps the entire global state to the corresponding output handler. All measurements and data recording will take place within the output handler.

The mechanism of the end-of-chain event handler is a bit more complicated. As shown in Fig. 5.2, the global state when `send_event_time` is called, is meaningless, but rather the state when `send_out_state` is called, will be processed. The new active unit identifier is returned alongside the event time, and branches for both the current active unit and the next active unit are communicated through `send_out_state` argument. With this information, the end-of-chain event handler can easily terminate the current motion and start the desired one. In JF-V1.0, end-of-chain event handlers are designed based on the `EndOfChainEventHandler` abstract class, which requires a specific implementation to only give a new chain length, active identifier, and direction of motion. The `SameActivePeriodicDirectionEndOfChainEventHandler`, as the simplest example, always takes the same chain length and does not change the active unit but alternates the direction of motion in a cyclic manner.

5.1.6 Other helping event handlers

JF simulation has no mandatory event handlers other than the `StartOfRunEvent-Handler` and the `EndOfRunEventHandler` class that indicate the beginning and the finishing of the run as their names suggest. It is inscribed inside the `TagActivator` class that, for the first time to generate candidate events, only the start-of-run event handler will be returned, which usually introduces an event at time zero. The first set of real candidate events can then be conveniently defined as in the creating list of the start-of-run event. And also, the `InitialChainStartOfRunEventHandler` class does the job of imposing the velocity and time stamp for the first active unit.

The total simulation time is set in the parameter of an `FinalTimeEndOfRunEvent-Handler` instance that is created at the beginning (by the start-of-run event handler) and stays inside the scheduler throughout the whole simulation. It does two things, to time-slice all active objects to the event time and to raise an `EndOfRun` exception.

We also created special facilities to dump the whole program for debugging purposes. This is realized through an `DumpingEventHandler` instance similar to a sampling event handler, and an `DumpingOutputHandler` instance. The magical thing is that, with the help of the `dill` package, all memory can be packaged and restored later by dumping the mediator object, because other objects are all direct or indirect attributes of it.

5.2 Factory

Figs. 5.3 and 5.4 best illustrate the forms of the class signature, configuration files and the underlying tree structure ruled by the factory design pattern [99]. By such a tree-like organization of program objects, JF preserves vast flexibility and extensibility.

To understand the factory pattern, we first have to introduce JF's packages. Physically, a JF package is a directory somewhere in the program, whose name is indicated by the directory's name. It contains several modules (i.e., formatted python scripts), each defining a usable class that we call "option" here. Options within a package follow the design of python class inheritance, and an abstract class usually exists in one script ruling the interface of its child classes. The name of an option is indicated by its script's name and class name, which are supposed to be the same except the difference between the Pascal case and snake case. For example, inside the package `scheduler` we can see abstract class `Scheduler`. Also, there is an option called `heap_scheduler` whose class `HeapScheduler` inherits from the `Scheduler` class.

Then we can explain the factory design pattern using the tree of Fig. 5.4.

1. Nodes of the tree except the root `Run` are packages of JF. An option must be specified for one node. For instance, the `HeapScheduler` option is for the node `Scheduler` required by a single-process mediator.

2. Signatures (i.e. the arguments of the `__init__` method) of each option must be specified with types as exemplified in Fig. 5.3 (a) . If an argument is a JF class, it points to a child node package, and an option must be given. Any JF class is capable of including other JF classes as its signature, thus the object tree can be constructed in an iterative way.

(a)

```
class SingleProcessMediator:
      def __init__(self, input_output_handler:  InputOutputHandler, state_handler:  StateHandler,
                         scheduler:  Scheduler, activator:  Activator):
```

(b)

```
[section] property value
[Run]
mediator = single_process_mediator
setting = hypercubic_setting

[HypercubicSetting]
system_length = 1
beta = 2
dimension = 3

[SingleProcessMediator]
state_handler = tree_state_handler
scheduler = heap_scheduler
activator = tag_activator
input_output_handler = input_output_handler

[TagActivator]
taggers =
         coulomb (factor_type_map_in_state_tagger),
         sampling (no_in_state_tagger),
         end_of_chain (no_in_state_tagger),
         start_of_run (no_in_state_tagger),
         end_of_run (no_in_state_tagger)

[Coulomb]
event_handler = two_leaf_unit_bounding_potential
                _event_handler

[TwoLeafUnitBoundingPotentialEventHandler]
potential = merged_image_coulomb_potential
bounding_potential = inverse_power_coulomb
                     _bounding_potential

[Sampling]
event_handler = fixed_interval_sampling_event_handler

[FixedIntervalSamplingEventHandler]
sampling_interval = 0.56789
output_handler = separation_output_handler

[EndOfChain]
event_handler = same_active_periodic_direction
                _end_of_chain_event_handler

[SameActivePeriodicDirectionEndOfChainEventHandler]
chain_length = 0.78965

[TreeStateHandler]
physical_state = tree_physical_state
lifting_state = tree_lifting_state

[InputOutputHandler]
output_handlers = separation_output_handler
input_handler = random_input_handler
```

Figure 5.3: Configuration file `coulomb_atoms/power_bounded.ini`. (a) Signature of the `single_process_mediator.__init__` method. (b) Excerpt of configuration file with some lines split for readability (which is not allowed in reality). A configuration file consists of sections, each having several properties and values. The section, properties and values correspond to the class name, `__init__` argument names and assigned values, as indicated by the color code of (b). (From Ref [2].)

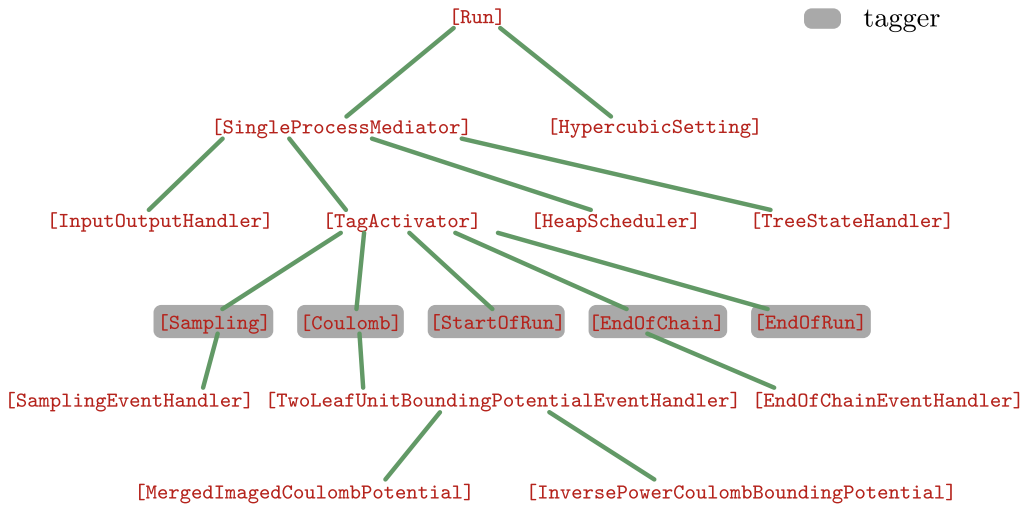


Figure 5.4: Tree structure corresponding to the configuration file of Fig. 5.3. Each node represents a JF class except the root node. The script `run.py` creates a `Mediator` instance and a `Setting` instance. Tags of events are highlighted in the figure. (From Ref [2].)

3. The `factory.py` script creates instances before the Markov chain, using the information given in the configuration file and the importing paths of modules. Usually, an object has access to its direct child objects, but not its indirect children or nodes of upper levels except that particular communication is performed initially.

The factory mechanism significantly facilitates customization and future extension of the application, which will be described in section 5.2.2. Next, we will describe aspects more related to user interfaces and then explain how the factory builds up such a tree-structured set of objects.

5.2.1 Running

The application starts with executing `run.py` in root directory using python3 or pypy3. In version 1.0, we provide a couple of running options that are shown with `python3 run.py -h` command, including the logging level, log file, and JF version. A configuration file must follow if we hope to start a simulation. A successful execution can be

```
python3 run.py -v -l logs/logs.txt\
config_files/2018_JCP_149_064113/\
coulomb_atoms/power_bounded.ini
```

in the directory `src`.

`run.py` executes the following jobs consecutively.

1. Set up the logging function.
2. Parse the configuration file through the `configparser` package.
3. Use the factory to build up a `Setting` instance to configure common parameters.
4. Use the factory to build up a `Mediator` instance.
5. Call `mediator.run()`, until an `EndOfRun` exception is thrown.
6. Call `mediator.post_run()`.

A similar `resume.py` is provided alongside `run.py` to resume the program with a dumped file. States of all objects under the mediator, as well as the setting and the random number object, are completely stored in the dumped file.

We have tried to minimize the usage of external packages. However, some are inevitable in order to perform special functions. The dumping facility has to import `dill`. The plotting scripts in `src/output/2018_JCP_149_064113` depends on `matplotlib` and `numpy`. `PdbInputHandler` and `PdbOutputHandler` use `mdanalysis` to read `.pdb` files.

```
[TagActivator]
taggers = coulomb (factor_type_map_in_state_tagger)

[Coulomb]
event_handler = two_leaf_unit_bounding
                _potential_event_handler

[TwoLeafUnitBoundingPotentialEventHandler]
potential = merged_image_coulomb_potential
bounding_potential = inverse_power_coulomb
                     _bounding_potential
```

```
[TagActivator]
taggers = harmonic
          (factor_type_map_in_state_tagger)

[Harmonic]
event_handler = two_leaf_unit_event_handler

[TwoLeafUnitEventHandler]
potential = harmonic_potential
            (displaced_even_power_potential)

[HarmonicPotential]
equilibrium_separation = 0.1
prefactor = 200
power = 2
```

Figure 5.5: Modification of the configuration file to replace the Coulomb potential with a harmonic potential. Left: Part of the configuration file for Coulomb potential. Right: Part for the harmonic potential.

5.2.2 Customization

In this section, we will briefly explain how to customize the simulation by adapting configuration files and how to extend the functionality of JF-V1.0 by writing our modules and packages. For more detail we refer readers to those introductory `.md` files in the root directory, especially `FACTORY.md`.

5.2.2.1 Configuration file

The format of configuration files implements a tree structure inside. For a section representing a JF class, properties correspond to arguments of its `__init__` method and users can specify values for the created instance. If an argument is a JF class (usually it is an abstract class), a value must be (or can be, if it is a keyword argument) specified with a child class of the required one in the snake case. Furthermore, in most cases, the given class must be instantiated with positional parameters, then we have to give a new section of the configuration file with the same name as the class. The configuration file allows aliases in addition to value-section correspondence. With the format "property = alias (value)", the section for it will be the "alias" instead of the "value" (conversion to Pascal case is still necessary).

For example, in Fig. 5.3 (a) , to create an instance of `SingleProcessMediator`, one has to give an `InputOutputHandler`, an `Activator`, a `Scheduler` and a `StateHandler`. Thus in the `SingleProcessMediator` section of Fig. 5.3 (b) , we designate four values for them which are child classes of the abstract ones. And also, three of them except `HeapScheduler` require arguments, so corresponding sections must be created detailing all necessary parameters which may also be JF classes. In this way, the tree-structured set of objects is expressed serially in the configuration files.

Fig. 5.5 shows how to change simulated Coulomb potential to a harmonic one. The bounding potential is no longer needed due to the invertibility of harmonic potential.

5.2.2.2 Development

It is easy to extend a package by adding one realization of the abstract class. For example, to write your own invertible potential, you just have to implement `derivative`, `potential`, and `displacement` methods. All necessary parameters for the potential are defined as arguments of the `__init__` method that must be accompanied with type hints. For more complicated modules like event handlers and taggers, the communication protocol with the mediator and the rest should be considered. If one of the arguments of `__init__` method is a JF class, a package must be created for it, as mentioned in the factory pattern. We choose `PEP8` style for python code except that the line length is set to 120 characters, and `PEP257` conventions for docstrings.

5.3 Verifications

Examples shown in this section serve purposes not only for JF integration test but also as examples of various taggers, event handlers, and potentials. Their physical systems are not different from those described in chapter 3—which are Coulomb atoms, dipoles, and the SPC/Fw water model—but much more precise measurements can bring confidence in both our algorithms and JF code.

For the verifications described below, they all feature a single-process mediator, heap scheduler, hypercubic setting, and tag activator. JF's integration of different systems and algorithms proves that we have adopted a vastly flexible code structure.

5.3.1 Coulomb atoms

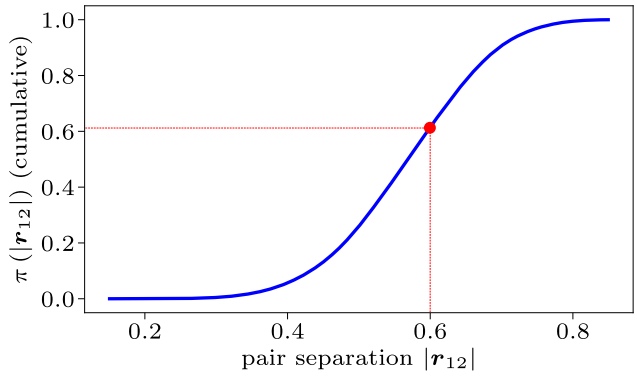
The two-Coulomb-atom system has a very simple factor, that is, merged-image Coulomb factor $\mathcal{M} = \{((1,2),\text{Coulomb})\}$, with $\beta = 2.0$, box side length $L = 1.0$, chain length $\ell = 0.78965$ and charges $c_1 = c_2 = 1.0$. We simulate this system via three methods with configuration files in the directory `src/config_files/2018_JCP_149_064113/coulomb_atoms`.

We performed multiple long runs and measured the probability of $|\mathbf{r}_{12}| < 0.6$ as well as its error with all three approaches in the following—the inverse-power bounding potential, cell-based bounding potential, and cell-veto algorithm—as shown in Fig. 5.6.

The initial physical state is generated with two random-positioned atoms inside the input handler, whereas a start-of-run event handler gives the lifting state. Hence one can change the number of atoms in the `RandomInputHandler` section and initial active atom in section `StartOfRunEventHandler`. Also, measurements take place within the `SeparationOutputHandler` instance, which is linked with the sampling event handler.

5.3.1.1 Inverse-power Coulomb bounding potential

The configuration file in `power_bounded.ini` is used for simulating atoms each with charge 1.0 through the inverse-power bounding potential described in section 3.4.



method	$\pi(\lvert\mathbf{r}_{12}\rvert < 0.6)$
①	0.61305 ± 0.00003
②	0.61306 ± 0.00006
③	0.61299 ± 0.00008
④	0.61307 ± 0.00012

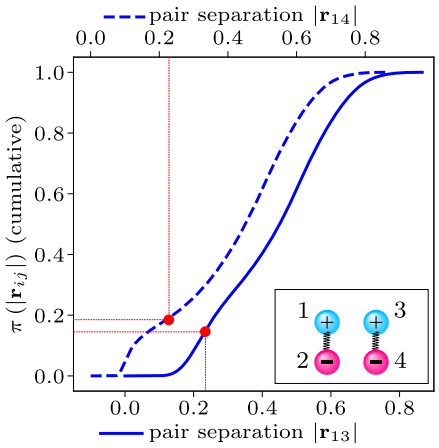
Figure 5.6: Validation with a two-atom system of section 5.3.1. The cumulative distribution of $|\mathbf{r}_{12}|$ is plotted on the left and the errors via four methods are given on the right. ①: Traditional Monte Carlo. ②: Method described in section 5.3.1.1. ③: Method described in section 5.3.1.2. ④: Method described in section 5.3.1.3. (From Ref [2].)

Corresponding part of the configuration file specifying the Coulomb factor is shown in the left panel of Fig. 5.5. To deal with the Coulomb interaction via inverse-power bounding potential, one must set up a `FactorTypeMapInStateTagger`, which reads in factors from a given file and designate two-body event handlers with bounding potential for it. Since only two atoms are simulated, one event handler will be sufficient. The Coulomb factor event trashes itself and recreates itself too. The Coulomb potential and the inverse-power bounding potential are set inside parameters of the event handler. Other events are sampling, end-of-chain, start-of-run, and end-of-run, all of which have no in-state; thus, the `NoInStateTagger` is used.

5.3.1.2 Cell-Based bounding potential

A cell-bounding-potential event must be configured to draw the event time and the displacement in place of a normal two-body event with bounding potential. Furthermore, as we introduce a cell-occupancy internal state, we have to create an additional cell-boundary event handler, sufficient excluded-cell event handlers, and cell-surplus event handlers. The configuration file for it is `cell_bounded.ini`

JF-V1.0 activates events of the cell-based bounding potential by instance of `CellBoundingPotentialTagger`, and specify `TwoLeafUnitCellBoundingPotential-EventHandler` for it since atoms are represented by separate nodes. Inside the event handler section, we specify its potential as `MergedImageCoulombPotential` and its bounding potential as `CellBoundingPotential` where the piecewise-constant-derivative potential is estimated and sampled. The estimator is specified as a parameter of cell-bounding potential. Excluded-cell events and cell-surplus events are activated by the `ExcludedCellsTagger` and the `SurplusCellsTagger` respectively while their event handlers are of the same `TwoLeafUnitBoundingPotentialEventHandler` class which uses an inverse-power Coulomb bounding potential, and works in the same way as that of section 5.3.1.1. We must give an internal state label, to all these cell-related taggers as well as the cell-boundary tagger.



method	$\pi(\lvert\mathbf{r}_{13}\rvert < 0.22)$	$\pi(\lvert\mathbf{r}_{14}\rvert < 0.22)$
①	0.12128 ± 0.00003	0.18453 ± 0.00003
②	0.12129 ± 0.00003	0.18458 ± 0.00004
③	0.12129 ± 0.00003	0.18456 ± 0.00005
④	0.12108 ± 0.00037	0.18412 ± 0.00053
⑤	0.12130 ± 0.00002	0.18457 ± 0.00003

Figure 5.7: Validation with a two-dipole system of section 5.3.2. The cumulative distributions of $|\mathbf{r}_{13}|$ and $|\mathbf{r}_{14}|$ are plotted on the left and the errors via five methods are given on the right. ①: Traditional Monte Carlo. ②: Method described in section 5.3.2.1. ③: Method described in section 5.3.2.2. ④: Method described in section 5.3.2.3. ⑤: Method described in section 5.3.2.4. (From Ref [2].)

5.3.1.3 Cell-veto method

A replacement of the cell-bounding potential tagger with a cell-veto tagger in configuration of section 5.3.1.2 leads to the cell-veto simulation of two charged atoms as configured in `cell_veto.ini`.

The cell-veto tagger is then given `LeafUnitCellVetoEventHandler` to deal with atom factors in which an estimator must be given. Likewise, cell-surplus, excluded-cell, and cell-boundary events are also necessary to run via the cell-veto method, and they are configured in the same way as in section 5.3.1.2.

5.3.2 Dipoles

The two-dipole system is defined by Eqs. 3.6.3 $\sim$ 3.6.5 consisting of harmonic, repulsive and Coulomb interactions. We prepare seven different simulation methods in the directory `src/config_files/2018_JCP_149_064113/dipoles`. All dipole simulations have parameters $\beta = 1.0$, $c_1 = -c_2 = c_3 = -c_4 = 1.0$, $L = 1.0$, $\ell = 0.78965$ with potential prefactors $k_b = 400$, $k_2 = 1.0$ and $r_0 = 0.1$.

Here we present five methods featuring atom or dipole factorizations, using inverse-power bounding potential or cell-based bounding potential, or cell-veto algorithm. The first four are with single active atom, while the last one is for testing the dipole mode of motion, as illustrated in Fig. 5.8 (a) and (b) respectively. Results and errors for $\pi(|\mathbf{r}_{13}|) < 0.22$ and $\pi(|\mathbf{r}_{14}|) < 0.22$ are compared in the right table of Fig. 5.7.

Like in the two-atom case, the initial dipole state is created randomly by an `RandomInputHandler` instance, while the node creator becomes the `DipoleRandomNodeCreator` class which distributes dipole centers evenly within the simulation box and samples dipole orientations on a unit sphere.

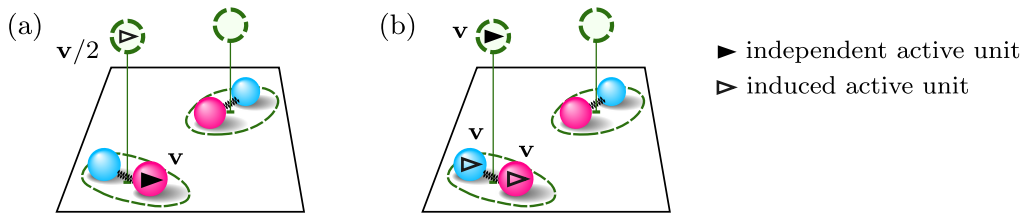


Figure 5.8: Two-dipole systems. (a) Leaf mode, in which one atom is independently active and its parent node acquires an induced velocity. (b) Root mode. In the root mode one dipole is translated as a whole, so the root has an independent active unit while its leaves have induced active units.

5.3.2.1 Atomic Coulomb factors

The configuration file `atom_factors.ini` adopts atomic Coulomb factorization given by Eq. 3.6.4. Coulomb, harmonic and repulsive factor events are all activated by `FactorTypeMapInStateTagger` instances. Like in section 5.3.1, the Coulomb interaction is treated with an inverse-power bounding potential, while the harmonic potential and the repulsive potential are invertible thus `TwoLeafUnitEventHandler` objects are created for them. For the active atom at any moment, there are four two-body factors relevant to it (2 Coulomb + 1 harmonic + 1 repulsive).

5.3.2.2 Dipole Coulomb factors, cell-based bounding potential

The configuration file `cell_bounded.ini` features a cell-based bounding potential to deal with the Coulomb dipole factor. To realize that, we must provide ways to identify the target dipole's cell, estimate the charge-dipole event rate, and give a bounding potential for exceptional cases.

The `SingleActiveCellOccupancy` class is coded so that it can take non-leaf units as occupants. Potential users can choose an arbitrary level of nodes as cell occupants by setting the property `cell_level` (value 1 for dipole, 2 for atom in the dipole case). The event handler to treat dipole factors via cell-based bounding potentials is an `TwoCompositeObjectCellBoundingPotentialEventHandler` instance, for which an underlying pairwise potential, a cell-based bounding potential and a lifting scheme must be specified. In the configuration section `CellBoundingPotential`, we have to choose a dipole estimator among those described in section 4.5.4.

As for the direct bounding potential, we choose the sum of individual pairwise bounding potential, i.e., the inverse-power one. The `TwoCompositeObjectSummedBoundingPotentialEventHandler` class is designed for this purpose. It needs to know only the two-body real potential and the two-body bounding potential. In this way, the bounding event rate is the sum of those of pairs involving the active atom so that the event time can be calculated as the minimum of each relevant pair. The real event rate takes into account the dipole factor, which can be guaranteed to be equal to or less than the bounding potential sum. Excluded-cell events and cell surplus events have to be treated in this way.

Since the cell occupancy registers dipoles instead of atoms, it is the dipole node position that determines the corresponding cell. Hence, the cell change of an atom

does not necessarily imply changes in the internal state. Luckily, `CellBoundaryEventHandler` can fully consider this matter and return the event time at which the relevant node crosses a boundary.

5.3.2.3 Dipole Coulomb factors, cell-veto

The configuration file `cell_veto.ini` is similar to `cell_bounded.ini` but uses the cell-veto algorithm instead. It treats dipole factors in normal cells by a `CompositeObjectCellVetoEventHandler` instance, which is activated by `CellVetoTagger` just like the one for atomic factors. The configuration for cell-occupancy, excluded-cell events and cell surplus events are the same as in section 5.3.2.2.

5.3.2.4 Dipole Coulomb factors, alternating leaf mode, and root mode

The configuration file `dipole_motion.ini` realizes dipole motion as a whole and switches between the dipole and the atom mode with dipole Coulomb factors. Fig. 5.8 illustrates the two modes. Factor events consist of two kinds—leaf mode events and root mode events. Taggers with a suffix "leaf" activate the leaf mode events. The corresponding tagger classes, event handlers, and other specifications are the same as the file `dipole_factor_inside_first.ini`.

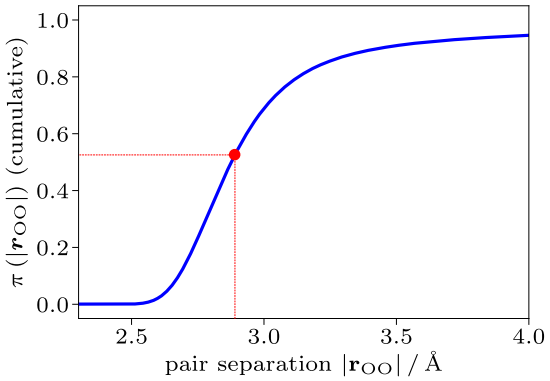
The root mode has Coulomb and repulsive factor events which are created also by the `FactorTypeMapInStateTagger`. Then, the root-mode Coulomb event is passed to the `RootUnitActiveTwoCompositeObjectSummedBoundingPotentialEventHandler` and the root-mode repulsive event is given to the `RootUnitActiveTwoLeafUnitEventHandler`. The former requires a pairwise bounding potential and takes the sum over all inter-dipole pairs as the dipole-dipole bounding potential for the dipole Coulomb factor. The latter treats invertible potential exclusively.

In addition, two root-leaf switchers are needed which are activated by the `ActiveRootUnitInStateTagger`. They feature special lists of activation and deactivation as described in section 5.1.4. The root-to-leaf event deactivates all root mode events and activates all leaf mode events while the leaf-to-root event does the contrary.

5.3.3 SPC/Fw water

In order to put JF into a more realistic application, we also test it for two sets of cells and a three-body potential. System parameters are identical to those in section 3.6.2 except that we restrict to two molecules and the system length is $L = 10.0$Å. The chain length here is set to be 2.12345Å.

We present four methods differing in how to treat the Coulomb and the Lennard-Jones factors. Confirmations and errors are shown in Fig. 5.9. The initial state is created randomly using a `WaterRandomNodeCreator` instance. We take the oxygen-oxygen separation as the observable that are calculated in the `OxygenOxygenSeparationOutputHandler`. Configuration files used in this section are in the JF directory `src/config_files/2018_JCP_149_064113/water`. For the moment, we concentrate on two water molecules for the sake of demonstration and verification. In chapter 6, we will simulate much larger water systems through JF for performance benchmark.



method	$\pi(\lvert\mathbf{r}_{OO}\rvert < 2.9\,\text{Å})$
①	0.5382 ± 0.0003
②	0.5385 ± 0.0003
③	0.5390 ± 0.0002
④	0.5400 ± 0.0010
⑤	0.5398 ± 0.0010

Figure 5.9: Validation with a two-water-molecule system of section 5.3.3. The cumulative distribution of $|\mathbf{r}_{OO}|$ is plotted on the left and the errors via four methods are given on the right. ①: Traditional Monte Carlo. ②: Method described in section 5.3.3.1. ③: Method described in section 5.3.3.2. ④: Method described in section 5.3.3.3. ⑤: Method described in section 5.3.3.4. (From Ref [2].)

5.3.3.1 Atomic Coulomb factors inverse-power bounded, Lennard-Jones inverted

The configuration file `coulomb_power_bounded_lj_inverted.ini` sets up the harmonic, bending, Coulomb and Lennard-Jones factors in a similar way as in section 5.3.1.1 and section 5.3.2.1. The factor map must be carefully set to reflect a total of 16 factors between two water molecules (9 Coulomb + 1 LJ + 2 bending + 4 harmonic) (see `factor_set_water_atomic.txt`).

The event handler different from previous examples is the `FixedSeparationsEventHandlerWithPiecewiseConstantBoundingPotential` which uses a speculated bounding potential (see section 5.1.2.3).

5.3.3.2 Dipole Coulomb factors inverse-power bounded, Lennard-Jones cell-bounded

The configuration file `coulomb_power_bounded_lj_cell_bounded.ini` takes dipole Coulomb factorization and uses cell-based bounding potential for Lennard-Jones interaction. For each pair of water molecule there exists only one Coulomb factor involving all six atoms. It treats Coulomb interaction using the `TwoCompositeObjectSummedBoundingPotentialEventHandler` class in the same way as we do for excluded-cell occupants and surplus units in section 5.3.2.2.

To have a cell-based bounding potential for the Lennard-Jones potential, we have to construct a cell occupancy exclusively for oxygen atoms. We exploit the "charge" property of the `SingleActiveCellOccupancy` class and set it as the oxygen indicator (an instance of `ChargeValues`). Oxygen atoms have 1.0 for this value, while hydrogen atoms have 0.0 so that oxygen atoms are registered in the cell occupancy.

5.3.3.3 Dipole Coulomb factors cell-veto, Lennard-Jones inverted

The configuration file `coulomb_cell_veto_lj_inverted.ini` uses the cell-veto algorithm for the dipole Coulomb factorization. To this end, we need to have a cell occupancy that reflects the position of the molecular center, which can be achieved by setting `cell_level` to be 1. Then the handling of the Coulomb event is the same as in section 5.3.2.3.

5.3.3.4 Dipole Coulomb factors cell-veto, Lennard-Jones cell-veto

The configuration file `coulomb_cell_veto_lj_cell_veto.ini` deals with the Coulomb and the Lennard-Jones interactions both via the cell-veto algorithm. Hence two sets of cell occupancy have to be created, one with cell level one, and the other with cell level two and the oxygen indicator as charge. For large systems, the cell-veto algorithm becomes indispensable to achieve acceptable time costs.

Chapter 6

Realistic water system

In this chapter, we put JF into real system simulation to have benchmarks for both the event-chain algorithm and JF application. In chapter 3, we propose an event-driven approach for the simulation of long-range systems though our proof of concept relies on simple time-driven codes. ECMC is supposed to face real challenges when entering complex biochemical systems. The configuration, observable, and physical order are no longer as simple as for the hard-disk testbed.

First, we will introduce important improvements to JF-V1.0 in order to eliminate obvious overhead in JF-V1.0. Then in section 6.2, we investigate the optimal setting for a large water system, notably for the cell-veto method that is rich in information where it costs most of the time. Finally, we test JF with water systems from 8 molecules to 216 against LAMMPS to better understand ECMC's efficiency.

6.1 Speeding up JELLYFYSH

The general architecture laid out in chapter 4 is a simplified event stream aimed to handle all kinds of operations. There, we focus on its extensibility but sacrifice the efficiency. For example, JF-V1.0 treats unconfirmed events in the same way as confirmed events, and the heap scheduler is developed for a large number of candidate events. We did not take into account features of simulated systems, including the event composition and time consumptions, which are crucial to the next improvement. Any popular scientific platform demands extreme optimizations, for the whole user group will repeatedly use common modules. In this sense, JF remains a prototype since the python codes are slow among programming languages, and functioning modules are to be rewritten.

6.1.1 Fast track for unconfirmed events

In JF-V1.0, the happening of each event requires recalculation of the other event times, except sampling, the start-of-run, and the end-of-run. This strategy becomes particularly costly since most of the events are unconfirmed events from bounding potentials and the cell-veto algorithm. In the cell-veto algorithm, event rates are highly overes-

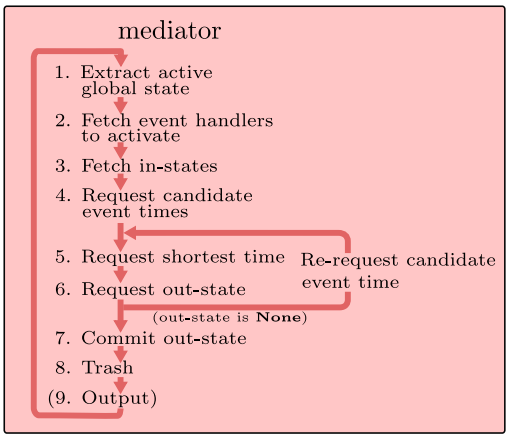


Figure 6.1: JF architecture adapted for events with bounding potentials. We add an extra loop to update by calling event handlers' `resend_event_time` without touching events already in the scheduler.

timated (see sections 3.3.2 and 6.2.2). In fact, it proves much cheaper to handle empty target cells and unconfirmed cell events than confirmed cell events.

It can save much computation power if unconfirmed events do not interfere with other events but update their event times. This goal is achieved using an extended communication scheme between event handlers and the mediator, and an additional loop inside the mediator. For the mediator, we create another loop inside the main loop between requesting the shortest event time and committing out-state, as shown in Fig. 6.1. It repeats until an event is confirmed by inspecting that the returned out-state is not `None`. The scheduler sorts event times as usual, and pops an event in every iteration of re-requesting event times. When an out-state is sent back, the inner loop breaks and the out-state is committed in the state handler for the subsequent operations.

In the event handler part, a new abstract class `EventHandlerWithUnconfirmed-Events` is created featuring the method `resend_event_time` that has no arguments. If the event is not confirmed, the value `None` will be sent back as the return value of `send_out_state`. The mediator then calls the event handler's `resend_event_time` instead of going back to call `send_event_time`. The whole event handler procedure is modified as shown in Fig. 6.2.

6.1.2 Other improvements

After the release of JF-V1.0, we developed other JF features to spread and speed up JF. First, for those time-consuming modules, it is advisable to implement in C and compile before execution. There are multiple ways to realize a mixture of C code in a python project, and we chose `cffi` to replace several calculation-intensive and self-contained modules[1]. Also, a scheduler using python **list** is provided. We found that

[1]For reference, see https://doc.pypy.org/en/latest/extending.html and https://cffi.readthedocs.io/en/latest/goals.html.

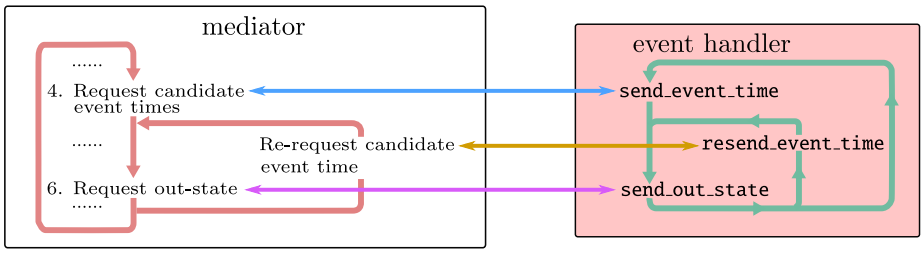


Figure 6.2: Stages of event handler with the method of resending the event time, for treating events with bounding potential. `resend_event_time` is called until `send_out_state` returns a value that is not `None`. The three event handler methods are called by three corresponding mediator stages on the left.

`HeapScheduler` costs a large portion of time probably because the external module `heapq` is not suited for event operation and because the number of candidate events does not warrant heap structure. The third thing we tested in order to speed up is a node-free version of JF, in which only `unit` is kept while `node` and the tree structure were abandoned. The measurements tell us that, for hard disks, the speed of the node-free version is 1.6 times the speed with nodes, so we will keep the node structure for all simulated systems to preserve consistency.

Another notable feature after JF-V1.0 is that we made JF into a python package. In fact, python provides tools to develop, encapsulate, distribute, and install packages (search python `steuptools` for reference). After the installation, one can run JF in any directory and the typical command becomes

```
jellyfysh -v -l logs.txt config_file.ini
```

where `logs.txt` and `config_file.ini` are the log file and the configuration file.

6.2 Coulomb event profiling

We study in this section how JF deals with long-range Coulomb events to find underlying statistics related to Coulomb events via the cell-veto algorithm, in hopes for more profound insights into time cost of the Coulomb event processing. We fix simulated system to be 20^3 water molecules in a box of side length 62.722Å (so with density $\sim 1\text{g}/\text{cm}^3$) at 300K. In such a system all algorithms dealing with the Coulomb interaction become prohibitively slow except the cell-veto algorithm. In the cell-veto algorithm, a key quantity $\tilde{Q}^{\text{cell}}_{\text{Clb}}$ is the reduced total Coulomb derivative. This is the sum kept in Walker's table, representing the sum of cell-based bounds of charge-water derivatives with unit active charge. When speaking of the Coulomb event rate $\langle Q^{\text{cell}}_{\text{Clb}}\rangle$, their relation is

$$\langle Q^{\text{cell}}_{\text{Clb}}\rangle = \beta(\frac{1}{3}|c_{\text{O}}| + \frac{2}{3}|c_{\text{H}}|)\tilde{Q}^{\text{cell}}_{\text{Clb}} = 0.918\tilde{Q}^{\text{cell}}_{\text{Clb}}. \tag{6.2.1}$$

6.2.1 Charge-water estimator

ECMC is an unbiased sampler under the condition that potentials are exact, and bounding potentials have event rates not less than the energy function derivatives everywhere, as indicated in Eq. 2.3.6. In the case of the Coulomb cell-veto algorithm, bounding potentials are constant within any cell and are given by an estimator in JF. $N = 20^3$ molecules imply around 40^3 cells, necessitating an efficient estimating method.

Early in JF-V1.0, we developed dipole estimators and applied them to two-molecule water simulations. However, there are signs showing that bounds by simplifying water as dipole fail to capture true properties of the charge-water interaction. A dipole model as simplified water molecule leads to significant underestimation. If one insists on the dipole estimator, a prefactor must be applied and results in worrying large event rates (see section 6.2.2 where bounds are given by charge-dipole estimator).

Thus, we implemented another two estimators for charge-water Coulomb interaction. They take water's inner structure into account but regard the water molecule as a three-point rigid structure. The first one is `ChargeWaterOptimizationEstimator` that uses `scipy.optimize.minimize` to obtain the maximum and minimum derivatives. It fixes at first the active atom, and calculates the effective position range of the target molecule, then explores the parameter space of the target molecule (a total of six degrees of freedom). This estimation is done by sampling the initial guess and calls `scipy` to find local extreme values. The second estimator randomly samples the position of the active atom as well as the target molecule's position and orientation. It can not only return extremes but the distribution of the charge-water derivative.

Assuming a homogeneous distribution of the active water molecule and the target molecule, for both their positions and orientations, we show in Fig. 6.3 the distributions of charge-water derivatives in two selected cells. In light of a large sample size of 1.28×10^9, we find that they rarely reach even 0.9 times of their maximums. Therefore, a precise estimation by searching the maximum is not necessary. Moreover, most of the extremes come from molecule positions on the cell boundary. Thus we expect that using a non-homogeneous distribution and favoring molecules on the boundary will lead to higher estimation efficiency.

6.2.2 Optimizing cell-veto events

We tested a wider range of cell-occupancy parameters including the number of cells per side and layers around the zeroth cell treated as exclusion. The result is shown in Table. 6.1.

Table. 6.1 contains much information. First, 32^3 cells have effectively eliminated surplus particles to an acceptable level, and with 34^3 cells, the surplus number can be ignored. Second, the highest speed comes with 40^3 cells and 5^3 exclusions, so we take $N_{\text{cell}} = 8N$ as the best choice of cell density from now on.

We also measured and got the confirmed Coulomb event rate for non-excluded cells as $\langle Q^{\text{cv}}_{\text{Clb}} \rangle = 43 \pm 1 \text{Å}^{-1}$. The reduced rate given by the estimator, however, is 13549Å^{-1} for 40^3 cells. It means that our cell-veto implementation has seriously over-

N_{cell}	N_{excluded}	$Q_{\text{Clb}}^{\text{tot}}$	$Q_{\text{LJ}}^{\text{tot}}$	$N_{\text{Clb}}^{\text{spl}}$	$N_{\text{LJ}}^{\text{spl}}$	speed(Å/s)
30^3	3^3	10434	2045.6	26	21	1.62
32^3	3^3	12074	2141.4	4	0	2.51
36^3	3^3	16377	2796.7	0	1	2.55
40^3	3^3	23034	3206.6	0	0	2.38
44^3	3^3	30600	3546.5	0	0	2.07
30^3	5^3	4632	5.7	26	21	1.07
32^3	5^3	5768	7.8	4	0	2.06
36^3	5^3	8626	12.8	0	1	2.51
40^3	5^3	13549	42.4	0	0	2.58
44^3	5^3	19533	177.3	0	0	2.42
48^3	5^3	25937	623.5	0	0	2.26
52^3	5^3	33097	1458.3	0	0	2.06
30^3	7^3	2734	0.8	26	21	0.36
32^3	7^3	3462	1.2	4	0	0.72
36^3	7^3	5314	2.6	0	1	1.16
40^3	7^3	7722	5.2	0	0	1.69
44^3	7^3	11192	9.4	0	0	1.98
48^3	7^3	15337	15.1	0	0	2.21
52^3	7^3	20179	21.2	0	0	2.19

Table 6.1: JF speed for 8000-molecule system with variable numbers of cells per side and exclusion layers. The first six columns from left to right are the number of cells, the number of excluded cells, the total cell-veto event rate for Coulomb and Lennard-Jones interactions, the number of surplus particles in molecular cell (for Coulomb), and that for oxygen cell (for Lennard-Jones). The speed in last column is measured in Å per second. Each case is run for 50Å on 2.10GHz Intel Xeon Gold 6230 CPU. The derivative bounds for Coulomb are gained through inner point charge-dipole estimator by treating water molecule as dipole of 0.82 elementary charge and 1.3Å separation, and results are multiplied by 1.2. Lennard-Jones derivative bounds are obtained through a particle-particle inner point estimator with a prefactor of 1.5. All data come from one 8000-molecule configuration in equilibrium produced by LAMMPS. And they are run with a JF version under development so only the relative speeds make sense.

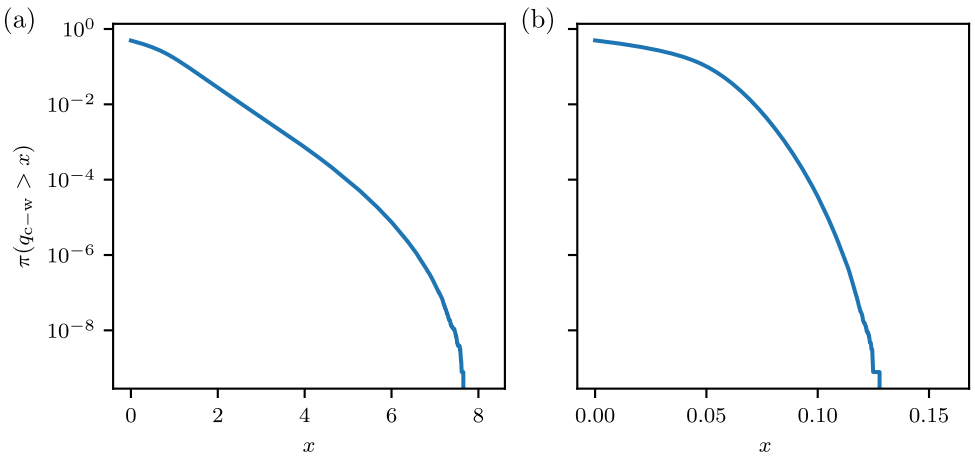


Figure 6.3: Differences between one and cumulative distribution of charge-water derivative. The simulated box is divided into 40^3 cells indexed from (0, 0, 0) to (39, 39, 39), and here we focus on two cells, each with 1.28×10^9 samples randomly drawn from zero cell and the target cell. (a) Target cell is (4, 1, 0), whose maximum returned by `scipy` optimization is 8.12. (b) Target cell is (5, 6, 7), whose maximum is 0.157. Note that q_{c-w} has roughly symmetric distribution around 0, so the curves here starts from (0, 0.5).

estimated the cell-based charge-water event rate (at the same time, violations are still reported that the true event rate exceeds the bounding event rate for some cells). Eq. 6.2.1 accounts for 0.918. A factor of eight comes from the occupancy rate, and another factor of four comes from dipole orientations as derived in Eq. 3.5.13. JF may have overestimated $13549/8/4 * 0.918/43 = 9.04$ fold.

To understand this loss, we first invoked the `scipy` estimator for more reliable upper bounds, which returns 9917Å^{-1}. It accounts for 1.37 of the factor 9.04, and the remaining 6.62 must stem from the molecule's positional variation within a cell, and from simplifying the water molecule as a dipole. Therefore, we employ increasingly finer cells, with 20^3, 40^3, and 80^3 cells in total, and set the exclusion cells as 3^3, 5^3, and 9^3 so that the nearest molecular pairs treated by the cell-veto algorithm are invariant. We measured the distribution of cell-veto rejection events, namely unconfirmed events from occupied cells. The distribution of molecule-center separations is shown in Fig. 6.4.

The occupied event rates for 20^3, 40^3, and 80^3 cells are measured to be 2034Å^{-1}, 1133Å^{-1} and 701Å^{-1} respectively (The event rate for 40^3 cells agrees with total cell-veto event rates because $1133 = 9917 * 0.918/8$). If 80^3 cells are small enough to eliminate the molecular variation in position, it follows that a factor of $1133/701 = 1.62$ originates from finite cell size, and the remaining $701/(43 * 4) = 4.08$ is from treating water molecules as ideal dipoles.

In summary, for our standard 8000-molecule-40^3-cell test case, a typical upper bound of the derivative given by the charge-dipole estimator (here we take 13549Å^{-1}) implies

- A total of $13549 * 0.918 = 12438$Å^{-1} cell-veto events,

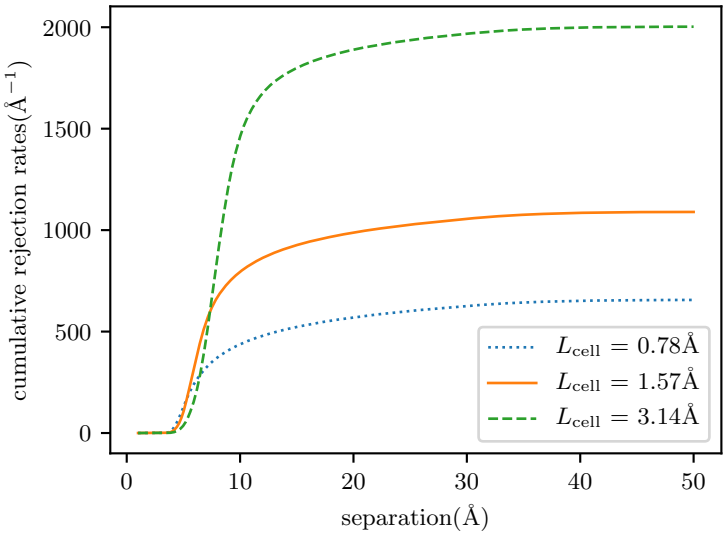


Figure 6.4: Cumulative distribution of molecule-center separations for rejected cell-veto events. Variable cell sizes are tested and excluded layers are set so that nearest possibility remains the same. Events from empty cells are not counted.

- $13549/8 = 1694$Å^{-1} occupied-cell events,
- 43Å^{-1} confirmed events.

The empty-cell event is considered as inevitable, yet the large gap between 43 and 1694 can be divided into the following facts

- Estimator overestimates by 1.37 times.
- For ideal dipole, there is a factor 4 for its orientation.
- Water molecule inner structure accounts for 4.08.
- The large cell size brings a factor 1.62.

6.2.3 Dipole factor vs. atomic factor

When we propose dipole factor in section 3.5 the complexity per sweep to sample a dipole system reduces from $\mathrm{O}(N^{4/3})$ to $\mathrm{O}(N \log N)$. However, such a speedup is not expected to manifest in a system of 8000 molecules because $N^{1/3}/\log N = 1.54$ for $N = 8000$, and we are concerned that a complicated dipole algorithm introduces another factor which is more than 1.54.

The dipole factorization has more computational burdens than the atom factorization. The event time sampling involves the interactions between the active atom and the entire target molecule, meaning three times as much computation for water as two atoms. To confirm an event, it demands night pairs of Coulomb interaction and has to perform an elaborate lifting scheme while one pair is sufficient in atomic factors.

Also, as discussed in section 6.2.1, it is more complicated to bound from above the charge-water derivative than the charge-charge derivative.

Thus we tested the atom factorization for 8000 molecules through JF. While the dipole factorization using the cell-veto algorithm is a fundamental feature in JF, the implementation of atom factorization for our benchmark model proves tricky. It is unrealistic to have one single-occupant cell holding either a hydrogen atom or an oxygen atom because the typical H-O separation 1.013Å is too short that a vast number of cells are needed. In the end, it is done by setting up three cell occupancies holding the first hydrogen atom, the second hydrogen, and the oxygen atom, respectively. Any type of active atom interacts with all three sets of cells[2].

In a system of the same setting, but with unit charges, our estimator tells us that the corresponding $Q_{\text{Clb}}^{\text{cell}} = 12228\text{Å}^{-1}$. For three charged sets, the total reduced rate is $12228\text{Å}^{-1} * (2|c_{\text{H}}| + |c_{\text{O}}|) = 20054\text{Å}^{-1}$, and this number is comparable with reduced cell-veto derivative bound 9917Å^{-1} for dipole factors. We expect that simpler computation in the atomic factor case can compensate for its larger event rate.

However, atomic factorization takes more than three times longer as the dipole case, contrary to what we believed. We found that the confirmed event rate in the dipole factor case is around 84Å^{-1}, including 64Å^{-1} Coulomb events, and this number rises to 578Å^{-1} with 546Å^{-1} Coulomb events in the atomic factor case. Confirmed events imply lifting moves that require recalculation of all candidate events, which seems inevitable. In our testing model, there are $5^3/8 \approx 15$ particles treated as excluded neighbors for one set of cell, and typical surplus atoms is 13, so there are $15*3(\text{Coulomb nearby}) + 15(\text{LJ nearby}) + 13(\text{Coulomb surplus}) + 2(\text{intra} - \text{molecule}) \approx 75$ pair interactions to be updated, accounting for the slowdown in atomic factorization. This measurement reveals that it is the confirmed cell-veto event that takes the majority of the time.

In the future, instead of trashing all physical events for each lifting move, we can stash the candidate events for each atom in hope that it will be reused later. For example, keeping a list for every atom within the active molecule until a new molecule becomes active will avoid around 40% recalculation if the inside-first scheme is adopted, according to our tests.

6.3 Water system benchmark

In this section we are going to start with relatively small systems, from 8 molecules to 216, with the box side length fixed to 18.975Å to keep the density $1\text{g}/\text{cm}^{-3}$ for 216 molecules (so for less molecules this means a dilute system). We measure the total polarization as the criteria of the system's relaxation, which is defined for N molecules as

$$\mathbf{P} = \sum_{i}^{N} c_{\text{H}}(\mathbf{r}_{i,\text{OH1}} + \mathbf{r}_{i,\text{OH2}}), \tag{6.3.1}$$

where $\mathbf{r}_{i,\text{OH1}}$ and $\mathbf{r}_{i,\text{OH2}}$ are separations between oxygen atoms, and the first or the second hydrogen atom respectively. Periodic boundaries are taken into account so

[2] In addition, a small change is added to avoid intra-molecule Coulomb interaction.

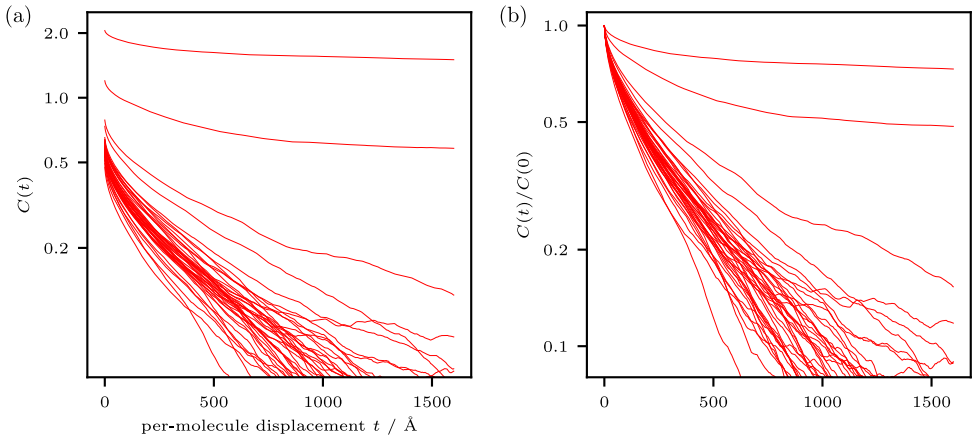


Figure 6.5: Normalized and non-normalized autocorrelation function of $\mathbf{P}$ for eight molecules. (a) Non-normalized autocorrelation denoted as $C_{\mathbf{P}}$. (b) Normalized autocorrelation referred to as $\tilde{C}_{\mathbf{P}}$. The time unit here is defined as total displacement divided by eight.

that $\mathbf{r}_{i,\mathrm{OH1}}$ and $\mathbf{r}_{i,\mathrm{OH1}}$ do not cross the entire system. The corresponding autocorrelation function is defined as

$$C_{\mathbf{P}}(t) = \langle \mathbf{P}(s) \cdot \mathbf{P}(s+t) \rangle_s, \tag{6.3.2}$$

where the average is taken over all valid samples, and the normalized autocorrelation is

$$\tilde{C}_{\mathbf{P}}(t) = C_{\mathbf{P}}(t)/C_{\mathbf{P}}(0). \tag{6.3.3}$$

6.3.1 Chain state

We observe that $C_{\mathbf{P}}$ for an 8-molecule simulation behaves strangely. Shown in Fig. 6.5 are $C_{\mathbf{P}}$ and $\tilde{C}_{\mathbf{P}}$ of 40 independent runs each with 10^6Å total displacement and starting with one equilibrated state prepared by LAMMPS. Two $\tilde{C}_{\mathbf{P}}$ curves deviate from others with much slower decay, and their corresponding absolute values show that those chains bear higher polarizations than the rest.

Then we pick one of the outliers and plot $\mathbf{P}$ as a function of time in Fig. 6.6. We find that one of its components presents a step-like shape: At some point, it increases to an unusually high level, stays for a period, and finally drops back to normal. The other abnormal $C_{\mathbf{P}}$ curves also exhibit such an excitation. The runs whose autocorrelations $C_{\mathbf{P}}$ are normal, however, do not feature the step-like polarization signal.

It seems that the system transitions to an unusual excited state at some point. This state has a distinguished polarization but has low free energy because its duration comprises less than one-hundredth of the total simulation. Due to the symmetry in the positive and negative direction, and in x, y, and z coordinates, similar excitations are also observed with $P_i < 0$ and $i \in \{x, y, z\}$. To capture the excited state's nature, we record the trajectories of runs and make them into animation. In Fig. 6.7, we present three snapshots during the simulation. The times are carefully picked to have one in the normal state, one in rising slope, and a third in a highly polarized state.

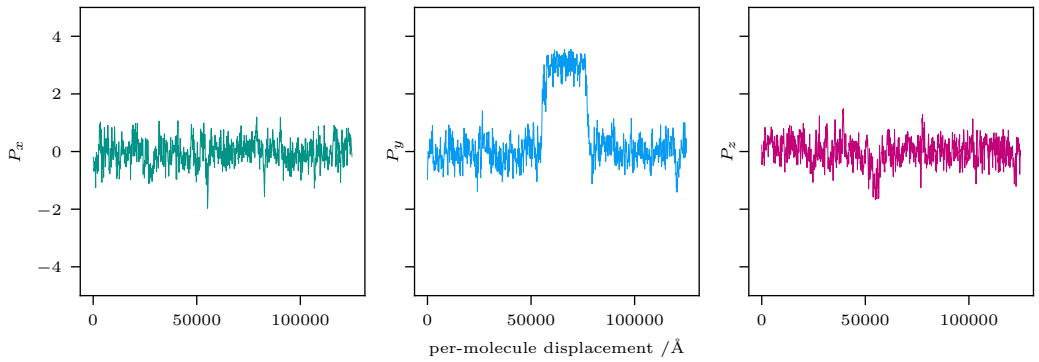


Figure 6.6: $\mathbf{P}$ as a function of t for the abnormal case in Fig. 6.5 whose $C_{\mathbf{P}}(0) \approx 2$.

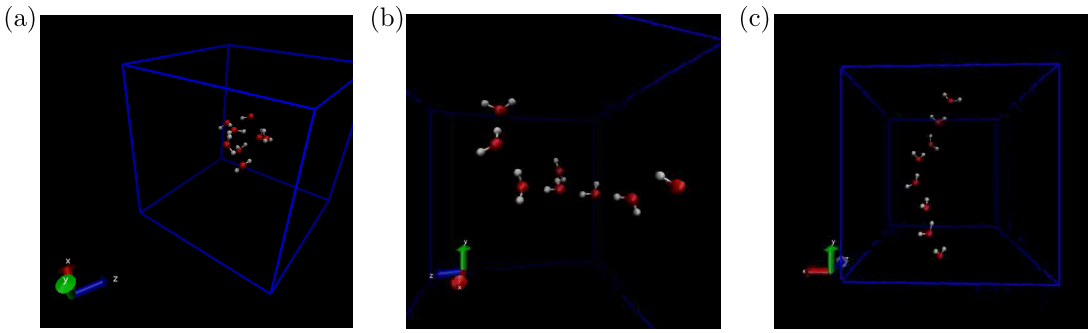


Figure 6.7: Visualization of an eight-molecule simulation that transitions to a high-polarization state. (a) A normal state, where molecules cluster into a drop. (b) An intermediary state. (c) The high-polarization state, in which eight molecules form a chain crossing the simulated box.

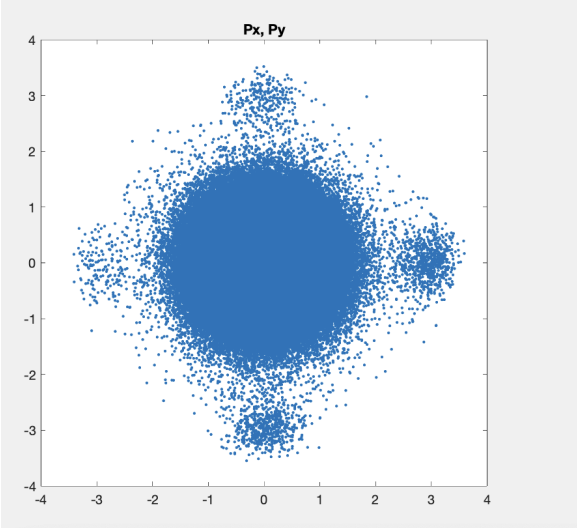


Figure 6.8: Scatter plot of $\mathbf{P}$ on $x-y$ plane through LAMMPS. Eight molecules are simulated in a system of side length 18.975Å for twenty hours. The central cluster represents non-excited polarization while four lobes around are excited chain states.

The chain state shown in Fig. 6.7 (c) lays out molecules in order, thus has lower entropy. This chain state cannot exist without periodicity in the simulation box, so the system topology is crucial to its stability. A simulation conducted by LAMMPS suggests that the chain state has similar total energy as the drop state, and there is an energy barrier in between. We plot the polarizations on $x-y$ plane in Fig. 6.8 acquired by 20 hours of LAMMPS. Admittedly, LAMMPS is much faster than JF in reproducing the 8-molecule chain state.

6.3.2 Molecular translation

We start increasing the molecule number in order to approach realistic situations. The box side length remains as $L = 19$Å while the numbers of water molecules N are taken to be 8, 27, 64, and 212. $L = 19$Å corresponds to $N = 216$ at the density $1\mathrm{g/cm}^3$. However, $N = 212$ is used instead because the less packed state is believed to speed up the relaxation. Here we fix the chain length $\ell = 0.2N$, and the cyclic switching scheme is employed. As discussed in section 6.3.1, the presence of outliers due to the chain state may heavily drive the mean $\tilde{C}_{\mathbf{P}}$ upwards. Thus we take the median of $\tilde{C}_{\mathbf{P}}$ from many runs as the measured normalized autocorrelation. To gain uncertainty for their median, we apply the bootstrapping method. If there are n runs, then n samples are drawn with replacement from them, and we can get the median from the drawn samples. Repeat this process, and the standard deviation of repetitions is the desired error.

The autocorrelation of $\mathbf{P}$ shown in Fig. 6.9 suggests a significant slowdown for large system size, making it prohibitively slow for realistic cases. We notice that the initial part of $\tilde{C}_{\mathbf{P}}$ decays faster but faces a slower decay afterwards.

The presence of two time scales usually implies an underlying factor impeding the

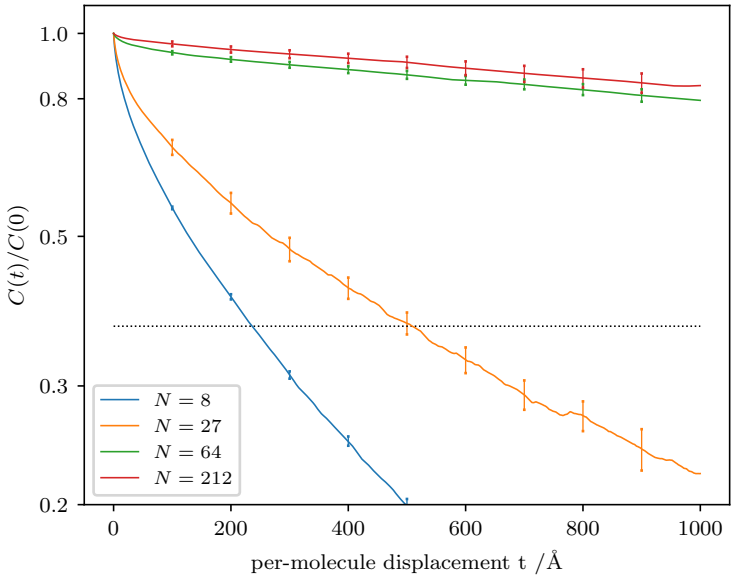


Figure 6.9: $\tilde{C}_{\mathbf{P}}$ with variable molecule numbers through the cyclic direction switching scheme with $\ell = 0.2N$.

relaxation of the algorithm. Here, ECMC seems to shake molecules during a short period but is hindered immediately by the configuration structure. To better understand the reason, we implement molecule translation, that is, moving a single molecule as a whole, just like the root-mode dipole motion described in section 5.1.4. What is new is that the cell-veto algorithm is included in the root-mode scheme, and the corresponding event handler and water-water Monte Carlo estimator are also implemented.

We alternate atom motion and molecule motion, and switch the moving direction right after each turn of the molecule move. Thus the pattern becomes $X_aX_mY_aY_mZ_aZ_mX_aX_m\ldots$ where a means atom mode, and m means molecule mode. The atom mode persists ℓ_a while molecule mode lasts ℓ_m (ℓ_m is the total translation displacement, and the sum of individual atom displacement will be $3\ell_m$). We fix $\ell_a = 0.2N$ while varying ℓ_m. Samples are only drawn in the atom mode, and only the atom mode is taken into sampling interval.

Fig. 6.10 shows the change of $\tilde{C}_{\mathbf{P}}$ with different lengths of molecule mode sandwiched in atom mode, as well as the errors. With the help of the molecule mode, $\tilde{C}_{\mathbf{P}}$ decays faster and there is sign that the second time scale is eliminated. The total performance in terms of $\tilde{C}_{\mathbf{P}}$, however, does not improve with consideration of the molecule mode cost.

However, with increasing molecule numbers, the improvement by introducing the molecule translation diminishes, as shown in Fig. 6.11. Long simulation time also disables us to precisely quantify their accelerations[3], but no evidence shows that $X_aX_mY_aY_m\ldots$ scheme can help reduce the second time scale to the level of LAMMPS. Other schemes like $(X_aY_a)^n(Y_aZ_a)^n(Z_aX_a)^n$ are also tested whereas the improve-

[3]It takes more than two days to run $N = 212$ by 10^6Å, to make up Fig. 6.11 (d)

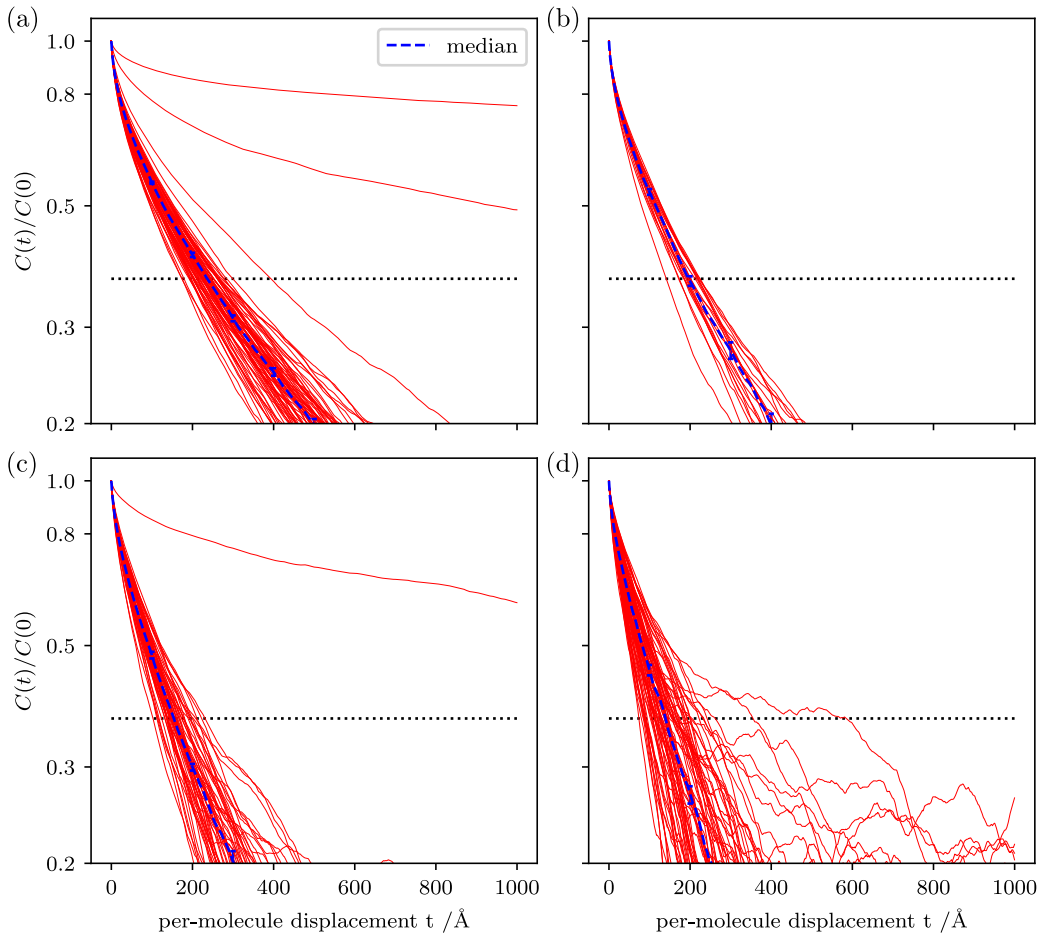


Figure 6.10: Normalized $C_{\mathbf{P}}$ of eight-molecule simulations with variable molecule translation lengths between cyclic atom mode. (a) Pure atom mode $\ell_m = 0$. (b) $\ell_m = 1$Å . (c) $\ell_m = 5$Å . (d) $\ell_m = 20$Å . Error bars are obtained through bootstrapping method. Dotted lines are for $y = 1/e$.

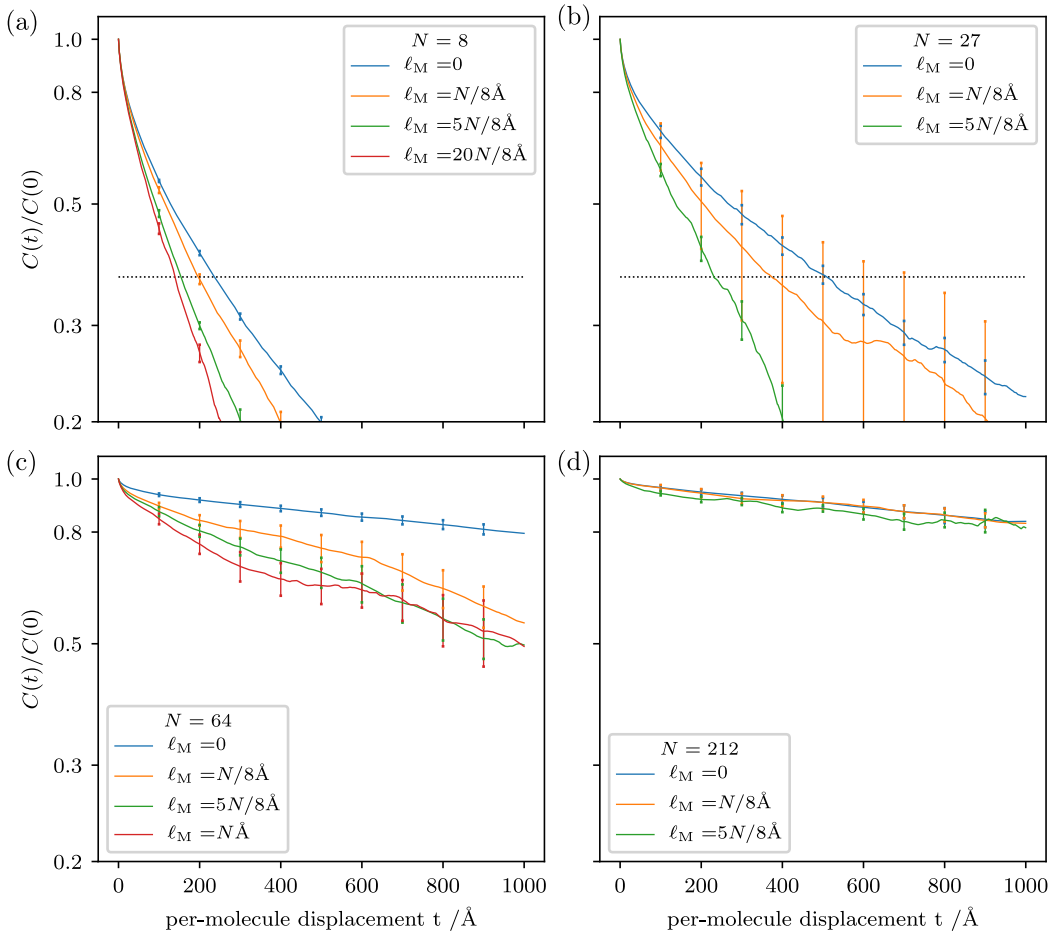


Figure 6.11: Normalized $C_{\mathbf{P}}$ for simulations of 8, 27, 64 and 212 molecules. Molecule translations of varied durations are inserted between atom modes, while atom modes all last $\ell_a = N/5$Å. (a) $N = 8$. (b) $N = 27$. (c) $N = 64$. (d) $N = 212$. Dotted lines in (a) and (b) are for $y = 1/e$.

ments are also limited. More tests and theoretical analyses are needed to reveal the reason of the double time scales.

6.3.3 Discussions

Apart from the above results, we also performed other measurements though the data are not ready to be published. First, ECMC can still achieve short correlation times for the density mode of water systems, so it is the collective rotation instead of the translation that poses challenges. The animation of a 216-molecule simulation displays restricted movements of dense molecules. Each molecule vibrates around its initial position, while the orientation and separations relative to the neighbors are almost frozen. It does not help by changing the lifting schemes proposed in section 3.5. On the other hand, traditional Monte Carlo also faces slow relaxation of the molecular polarization if only the atomic displacement is applied. It achieves a significant speedup by the proposal of molecular rotation.

To close the gap between JF with translational motion and LAMMPS, we analyze their procedures and list the differences here. The previous measurement in Fig. 3.19 only confirms the relaxation of molecule polarization through the motion along axes, but it does not reveal the relaxation speed.

First, LAMMPS is highly optimized, while JF-V1.0 remains immature. LAMMPS is mainly written in C++ and is compiled before running, but JF-V1.0 uses python. Moreover, LAMMPS features massive parallelization and even GPU acceleration. When running on a personal computer, multiple cores are used at the same time. It will be a good comparison by writing LAMMPS' algorithm and ECMC in the same language and the same style.

Some may argue that the ten-fold overestimation of the cell-veto Coulomb event rate is the main reason, as discussed in section 6.2.2. Further parameter adjustment may bring a slight speedup, while the unconfirmed events caused by the freedom of dipole rotation and particle movement within a cell are inevitable. We believe that a more profound mechanism exists that impedes the collective molecular rotation, given the presence of the second time scale and our test that molecular rotations can also speed up traditional Monte Carlo.

Next, trajectories given by molecular dynamics suggest that the motion in arbitrary directions may imply a faster equilibrium, so we can lift the restriction that all displacement must be along axes, which motivates chapter 7. In chapter 7, we will test the idea where the proposed direction of relaxation changes gradually, and the reduction in the mixing time is confirmed for dipoles with relatively narrow potentials. One can further exploit this freedom since there are numerous combinations of proposed directions.

Furthermore, lifting schemes provide another way to realize motions in general directions. One example is the reflective chain in the initial proposal [76] of ECMC, and it renders ECMC more similar to physical collisions. Intuitively, reflective chains enforce consistent rotation for a single planar dipole, whereas its generalization to higher dimensions is unclear.

Chapter 7

Sequential Monte Carlo

The dynamics of water molecule rotations obtained in section 6.3 bring a novel question for the ECMC. Simple complements through molecular rotation help, but do not significantly eliminate the slow time scale in the relaxation of the polarization. In fact, much freedom of the ECMC has not been exploited, and we have so far been restricting on the motion in positive senses of three orthogonal axes, namely $+x$, $+y$, and $+z$. It is intuitive to think that the molecular dynamics is faster due to its motion in all spatial directions.

On the other hand, sequential proposals in conventional Monte Carlo methods are regarded as the first approach leading to irreversibility, as discussed in section 1.4.1.1. Metropolis et al. mentioned the sequential scheme of updating particles in the original paper [36] of Monte Carlo method (it said "...we move each of the particles in succession according to..."). Orkoulas et al. formulated sequential updates in spin system [53] and then applied them to various other scenarios [55, 100, 101]. The speedup was demonstrated by both theoretical analyses and simulations, while they only studied the sequentiality in terms of spin or particle order.

In this chapter, we combine the two ideas above and apply the sequential proposal in the choice of directions, to study the possible gain and underlying dynamics if the proposal direction evolves gradually. In view of the difficulties experienced in chapter 6, we considered a system of a two-dimensional dipole as the testbed for this method and found that the evolution of the dipole angle exhibits distinct patterns, that is, the so-called "zigzag" and "excursion". The order of directions imposes an additional level of irreversibility to the existing schemes of ECMC. Finely tuning the speed of the direction change can achieve several times of speedup, and the generalization to higher dimensions and multiple molecules is possible in the future.

7.1 Single-dipole system

7.1.1 Configuration

We extract from more realistic models a simple dipole placed on a plane, as shown in Fig. 7.1 (a) . Two atoms of the dipole are represented by positions $\mathbf{x}_1$ and $\mathbf{x}_2$. We impose hard-disk potential between any two atoms and another intra-dipole hard

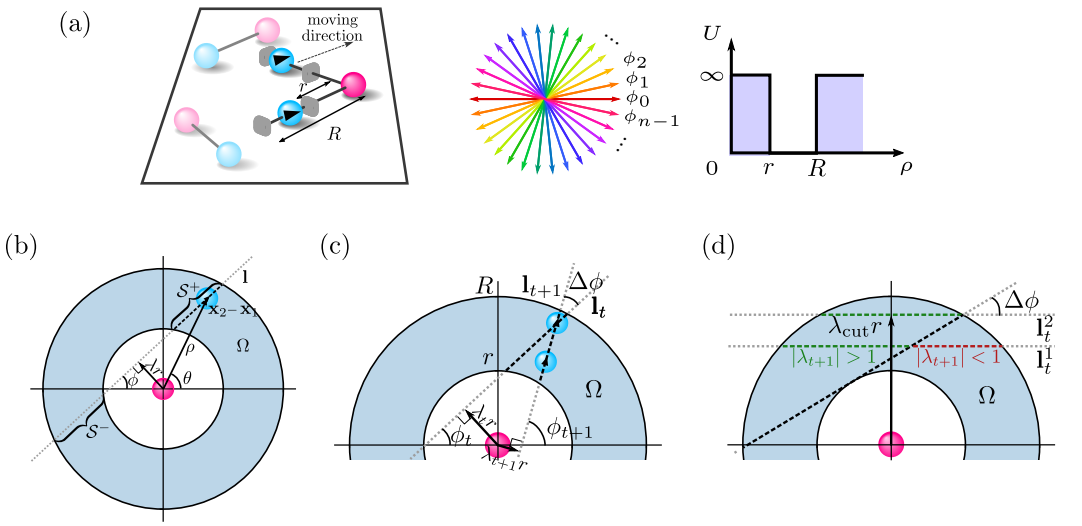


Figure 7.1: Configurations and notations of the two-dimensional dipole system. (a) Left: Dipoles on a plane, with the active particle moving in a given direction. Particles interact with hard-disk potential and intra-dipole hard repulsive potential. Middle: The series of direction $\phi_0, \phi_1, \ldots$. Right: The potential between two particles of a dipole. (b)∼(d) Important notations, with dotted lines representing the direction of motion. (b) and (c) illustrate the case with $|\lambda| < 1$ and (d) is for $|\lambda| > 1$. (Adapted from Ref [3].)

potential so that the energy of the studied single-dipole system can be expressed as

$$U(\mathbf{x}_1, \mathbf{x}_2) = \begin{cases} 0 & \text{if } r \le \rho \le R, \\ \infty & \text{otherwise,} \end{cases} \tag{7.1.1}$$

where $\rho = |\mathbf{x}_1 - \mathbf{x}_2|$ is the distance between two atoms. The potential is illustrated in the right panel of Fig. 7.1 (a) .

We can reparametrize any valid state as where θ represents the angle of dipole orientation and (ρ, θ) becomes the polar representation of $\mathbf{x}_2 - \mathbf{x}_1$. In light of periodic boundary condition, $\mathbf{x}_1$ must be in $S^1 \times S^1$, the two-dimentional torus. The condition for a valid state is simply expressed as $\rho \in [r, R]$ and $\theta \in [0, 2\pi)$, and the probability is evenly distributed (Euclidean measure) within a ring Ω, as shown in Fig. 7.1 (b) .

7.1.2 Sequential Monte Carlo method

Among $(\mathbf{x}_1, \rho, \theta)$, the parameter $\mathbf{x}_1$ is trivially distributed in $S^1 \times S^1$, and we are not interested in the equilibration of this part. $\mathbf{x}_1$ corresponds to the collective translation of the molecular systems, and ECMC has proved efficient in decorrelating density modes. On the other hand, the θ coordinate is a simplification of molecular orientations whose mixing presents a realistic challenge. Although such a ring's stationary distribution can be obtained analytically, we hope that this simplified model serves as a prototype to reveal new mechanisms of Monte Carlo with sequential directions.

Here we list three algorithms that respect the global balance condition for the distribution of $\mathbf{x} = \mathbf{x}_2 - \mathbf{x}_1$.

The most straightforward algorithm is the direct application of the Metropolis-Hastings method with symmetric proposals as described in Algo. 1.

Algorithm 1 Metropolis-Hastings algorithm

initialize $\mathbf{x} \leftarrow \mathbf{x}_0$
for $1 \le i \le N_s$ **do**
  draw $\Delta\mathbf{x} \in [-\epsilon, \epsilon]^2$
  $\mathbf{x}' = \mathbf{x} + \Delta\mathbf{x}$
  if $\mathbf{x}'$ is in the ring Ω **then**
    $\mathbf{x} \leftarrow \mathbf{x}'$
  end if
  print $\mathbf{x}$
end for

The ECMC algorithm simulates with consecutive chains that follow a determined series of directions. Each time we draw a chain length ℓ from a certain distribution and a direction from the pre-determined set. The active particle is advanced until it hits the hard wall induced by the other particle, or the cumulative displacement reaches ℓ. Previously, the directions of motion were kept along positive senses of axes, but now we extend to the sequence $\{\mathbf{e}_0, \mathbf{e}_1, ...\}$ with $\mathbf{e}_k = (\cos\phi_k, \sin\phi_k)$, $\phi_k = \pi n k/180$, where n is the change in degree between two consecutive directions. We denote the series as $\mathcal{D}_n$ and for the moment we focus on $n \in \mathbb{Z}$. And also, it is clear that moving $\mathbf{x}_1$ in $\mathbf{e}$ is equivalent to moving $\mathbf{x}$ in $-\mathbf{e}$, so the ECMC for sampling $\mathbf{x}$ becomes Algo. 2. It is noteworthy that in Algo. 2, the function `distance_before_collision` must consider both the inner circle and the outer circle of Ω. Here, samples of $\mathbf{x}$ are drawn at the end of chains, which is valid when the chain length is unrelated to the collision. Taking $n = 90$ is similar our conventional ECMC implementation that alternates between $+x$ and $+y$ directions.

Algorithm 2 Event-chain Monte Carlo with finite chain length

initialize $\mathbf{x} \leftarrow \mathbf{x}_0$
for $1 \le i \le N_s$ **do**
  draw chain length ℓ
  $\mathbf{e} \leftarrow (\cos\phi_i, \sin\phi_i)$
  while $\ell > 0$ **do**
    $s = \min(\ell, \text{distance\_before\_collision}(\mathbf{x}, \mathbf{e}))$
    $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{e} * s$
    $\ell \leftarrow \ell - s$
    $\mathbf{e} \leftarrow -\mathbf{e}$
  end while
  print $\mathbf{x}$
end for

Usually, Algo. 2 samples ℓ from a distribution with a characteristic length ℓ_0.

To further simplify the model, we set up $\ell_0 \to \infty$ to eliminate the finite-chain-length effect, which amounts to sampling $\mathbf{x}$ along the cut $\mathbf{l}$, that is, the ring Ω cut by the line $\mathbf{x} + \alpha\mathbf{e}$. Thus, we have the direct sampling Algo. 3. The pesudocode for Algo.

Algorithm 3 Direct sampling

```
initialize x ← x_0
for 1 ≤ i ≤ N_s do
  e ← (cos φ_i, sin φ_i)
  l ← cut_line(Ω, x, e)
  x ← random point from l
  print x
end for
```

3 simulates in the normal reference frame, and the sampling directions will evolve gradually following the series $\phi_0, \phi_1, ...$, as shown in the middle panel of Fig. 7.1 (a) . In practice, one can also operate in the reference frame always with $\phi = 0$. In this case, one has to rotate $\mathbf{x}$ clockwise $\Delta\phi = n\pi/180$ before each iteration. We will use the two interpretations to describe the evolution later in Fig. 7.3. In the rest of this chapter, Algo. 3 will be used as the common method to study the effect of sequential choices of the direction. We vary the set $\mathcal{D}_n$ to see the resulting physical and algorithmic consequences.

Each iteration in Algo. 3 is taken as one time unit. Also, rolled-out angles are labeled with tilde so we have $\tilde{\phi}_t = \pi n t/180$, the corresponding $\tilde{\theta}_t$ and ρ_t. We define the impact parameter λ_t, i.e., the component of $\mathbf{x}$ normal to $\mathbf{e}_t$, as

$$\lambda_t = \sin(\theta_t - \phi_t)\rho_t/r. \tag{7.1.2}$$

Obviously, $\lambda_t \in [-R/r, R/r]$. When $\mathbf{x}$ is on the left side of the direction $\mathbf{e}_t$, λ_t is positive; otherwise, it is negative. Moreover, λ_t has a crucial threshold ± 1 which will be manifest later. For now, one can distinguish that if $|\lambda_t| < 1$, as illustrated in Fig. 7.1 (b) and (c), $\mathbf{x}$ will be resampled from a segment formed between the two circles; if $|\lambda_t| > 1$, as shown in Fig. 7.1 (d) , $\mathbf{x}$ is free along a chord of the outer circle. These two cases do not only differ in the program but have distinct features in dynamics.

7.1.3 Validation

We verify our Algo. 3 here using the distributions of ρ and λ, which can be obtained analytically, as well as the homogeneous distribution over Ω.

The stationary probability density of ρ should be proportional to ρ itself, so we have

$$\pi(\rho) = \frac{2\rho}{r^2(\eta^2 - 1)}, \tag{7.1.3}$$

where $\eta = R/r$ is the ratio of the outer and inner circles of the ring.

As for the distribution of λ, it should also be proportional to the length of $\mathbf{l}$, namely

$$\pi(|\lambda|) = \begin{cases} C\sqrt{\eta^2 - \lambda^2}, & \text{if } 1 \le |\lambda| \le R/r; \\ C\left(\sqrt{\eta^2 - \lambda^2} - \sqrt{1 - \lambda^2}\right), & \text{if } 0 \le |\lambda| \le 1. \end{cases} \tag{7.1.4}$$

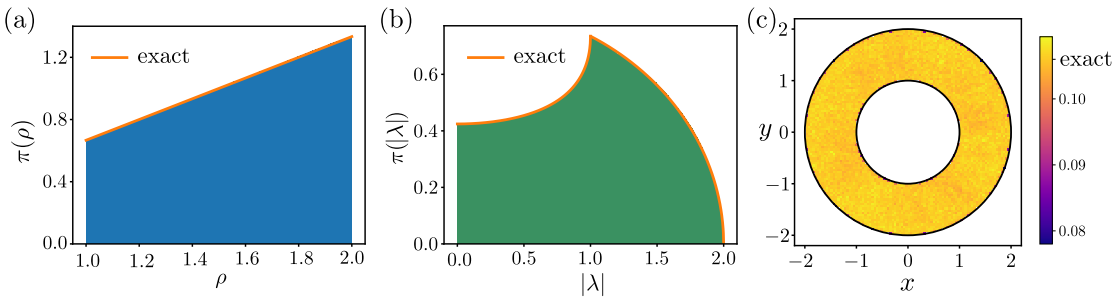


Figure 7.2: Probability distributions of ρ, $|\lambda|$ and $\mathbf{x} = (x, y)$ with $\eta = 2.0$. Results are simulated through Algo. 3 and compared with analytic solutions. Statistics are drawn from 10^8 samples and put into 400 bins in (a) and (b), and into 100^2 grids in (c).

The normalizing factor is $C = 4/\pi(\eta^2 - 1)$.

We performed a long simulation using the code of Algo. 3. The statistics of ρ, λ and $\mathbf{x}$ are plotted in Fig. 7.2. The agreement between the simulated and analytical distributions validates the global balance condition under the sequential method.

7.2 Rotational dynamics

We track the evolutions of $\tilde{\theta}$ and λ as shown in Fig. 7.3 that exhibit distinct patterns which are never observed in traditional Monte Carlo simulations. In this section we study the typical dynamics presented under moderate choices of η and $\mathcal{D}_n$, which consist of "zigzags" and "excursions" in the curves of λ, as shown in Fig. 7.3 (a) and (b). We also observed exotic behaviors when using large n values. For example, $n = 85$ leads to "amplitude-modulated" waves of λ, while we will not cover it in this thesis.

7.2.1 Zigzag and excursion

In Fig. 7.3 (a) and (b), the rolled-out dipole angle $\tilde{\theta}$ can rise and fall with fluctuation during a long simulation. The rising parts of $\tilde{\theta}$ correspond to $|\lambda| > 1$ and the falling parts correspond to $|\lambda| < 1$. When $\tilde{\theta}$ increases, λ behaves like a one-dimensional random walk, but when $\tilde{\theta}$ decreases, λ undergoes a more deterministic evolution that drives it between ± 1. Patterns of decreasing $\tilde{\theta}$ contain more than a drifted random walk but a series of steps, each corresponding to a λ period (see the middle panel of Fig. 7.4).

Given λ_t, the dipole separation $\mathbf{x}_t$ lies on the segment parallel to $\mathbf{e}_t$. It is possible to infer the distribution of λ_{t+1} through rotating $\mathbf{x}_t$ by $\Delta\phi$. The drift term for λ_{t+1} is

$$\mathbb{E}\lambda_{t+1} = \begin{cases} \lambda_t \cos\Delta\phi - A_t \sin\Delta\phi & \text{if } |\lambda| < 1 \text{ and } \mathbf{x}_t \in \mathcal{S}^+, \\ \lambda_t \cos\Delta\phi + A_t \sin\Delta\phi & \text{if } |\lambda| < 1 \text{ and } \mathbf{x}_t \in \mathcal{S}^-, \\ \lambda_t \cos\Delta\phi & \text{if } |\lambda| \geq 1, \end{cases} \tag{7.2.1}$$

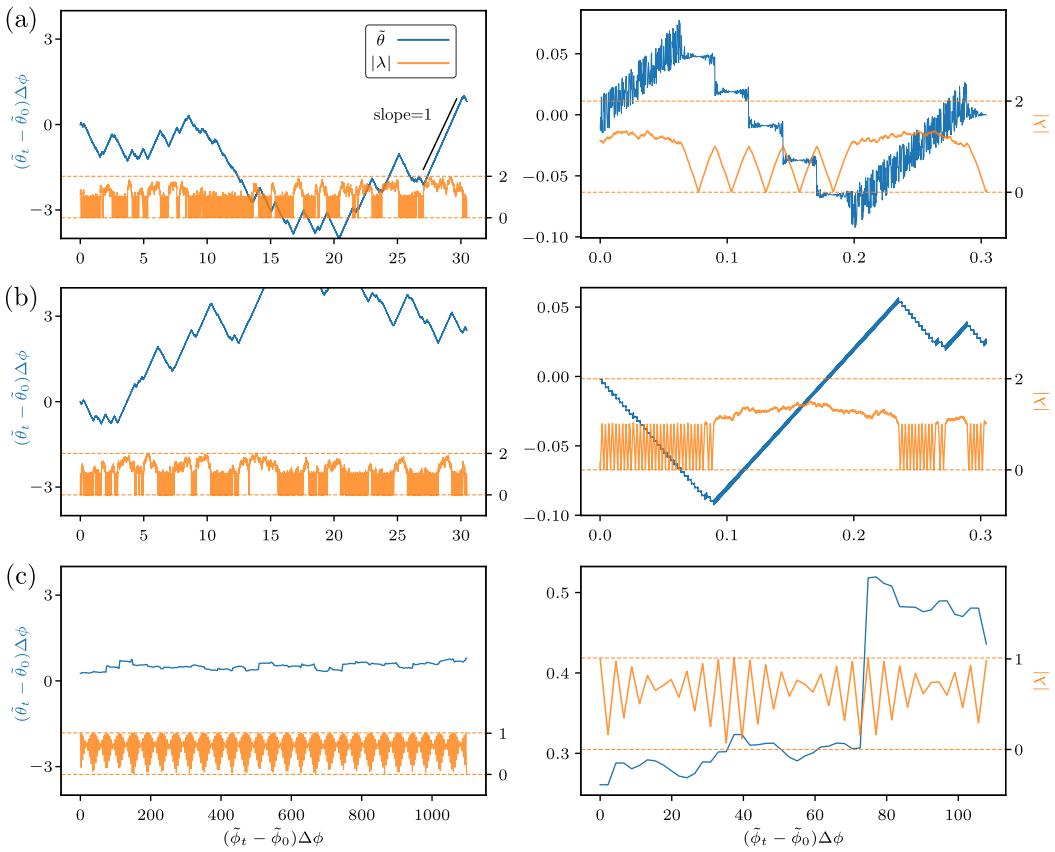


Figure 7.3: Sample trajectories of the rolled-out dipole angle $\tilde{\theta}$ and the impact parameter λ, with respect to $\tilde{\phi}$. Axes are rescaled with $\Delta\phi$ to highlight the similarity with different values of $\Delta\phi$. The right column is the initial potions of those in the left column. (a) $\eta = 2.0$, $\Delta\phi = \pi/180$. (b) $\eta = 2.0$, $\Delta\phi = 0.1\pi/180$. (c) $\eta = 1.01$, $\Delta\phi = 85\pi/180$. (Styles are from Ref [3].)

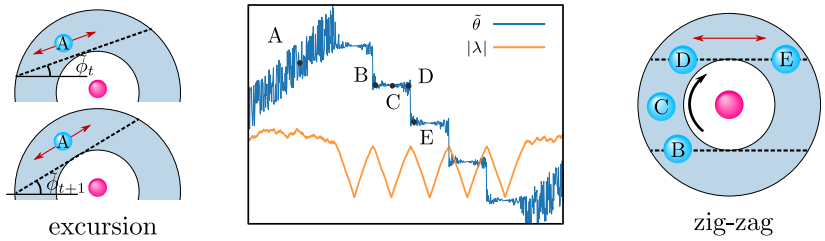


Figure 7.4: The states of $\mathbf{x}$ corresponding to zigzags and excursions. The $\tilde{\theta}$ and λ curves are extracted from Fig. 7.3 (a) . On the left are consecutive states for an excursion, while on the right we show states of a zigzag period in the reference frame with $\phi = 0$. (Adapted from Ref [3].)

where $A_t = (B_t + C_t)/2$ with $B_t = \sqrt{\eta^2 - \lambda_t^2}$ and $C_t = \sqrt{1-\lambda^2}$. $\mathcal{S}^+$ and $\mathcal{S}^-$ are the part of $\mathbf{l}_t$ where $\cos(\theta - \phi) > 0$ and the part where $\cos(\theta - \phi) < 0$, as shown in Fig. 7.1 (b) . And the variance of λ_{t+1} is

$$\sigma^2(\lambda_{t+1}) = \begin{cases} \frac{1}{12}(B_t - C_t)^2 \sin^2 \Delta\phi & \text{if } |\lambda_t| < 1, \\ \frac{1}{3} B_t^2 \sin^2 \Delta\phi & \text{if } |\lambda_t| \geq 1. \end{cases} \tag{7.2.2}$$

According to Eq. 7.2.1, for small $\Delta\phi$, the drift term $\mathbb{E}\lambda_{t+1} - \lambda_t$ is proportional to $(\Delta\phi)^2$ when $|\lambda_t| > 1$, while it is proportional to $\Delta\phi$ if $|\lambda_t| < 1$, which agrees with the observation that the former case is driven by random walk, whereas the latter is dominated by motion with less randomness.

In the case of $|\lambda| > 1$, given that the drift and the variance are both proportional to $(\Delta\phi)^2$, we can write down the equation for λ as

$$\lambda_{t+1} = \lambda_t + D_1 \Delta\phi^2 + D_2 \Delta\phi \omega', \tag{7.2.3}$$

where $D_1 = -1/2$, $D_2 = \sqrt{(\eta^2 - \lambda_t^2)/3}$, ω' is a uniform random number within $[-1/2, 1/2]$, in agreement with Eqs. 7.2.1 and 7.2.2. After a long period Δt, the change of λ should take the form of discrete Langevin equation

$$\lambda_{t+\Delta t} = \lambda_t + D_1 \Delta\phi^2 \Delta t + \bar{D}_2 \Delta\phi \sqrt{\Delta t}\omega. \tag{7.2.4}$$

Here, $\bar{D}_2$ is an effective value of D_2 during the period, and ω is a Gaussian random number with zero mean and unit variance. It is not difficult to notice the rescaling invariance of Eq. 7.2.4 by letting $\Delta\phi \to k\Delta\phi$, $t \to k^{-2}t$ (the same for Δt) and $\lambda_t \to \lambda_{k^{-2}t}$. Thus, in Fig. 7.3 (a) and (b), we set the axes as $(\tilde{\phi}_t - \tilde{\phi}_0)\Delta\phi$ and $(\tilde{\theta}_t - \tilde{\theta}_0)\Delta\phi$, then the lengths and frequencies of excursions are comparable for different $\Delta\phi$ values. In addition, the $|\lambda| > 1$ condition for excursions suggests that, $\mathbf{x}$ remains in a circular segment of Ω, as illustrated in the left panel of Fig. 7.4. As the circular segment rotates following the change of $\tilde{\phi}$, the dipole angle should rotate in the same speed, which results in a slope equal to 1 in the rising side of $\tilde{\theta} \sim \tilde{\phi}$ curve.

As for the zigzags, the $|\lambda|$ curve consists of a succession of V-shapes, meaning that λ goes between ± 1 in a more deterministic manner. The corresponding values of $\tilde{\theta}$ take downward steps with relatively fixed widths and heights. If we track $\mathbf{x}$ in this period, we find that it stays outside the circular segments of $|\lambda| > 1$ but jumps between $\mathcal{S}^+$ and $\mathcal{S}^-$. The right panel of Fig. 7.4 shows the correspondence of critical moments of λ and $\mathbf{x}$. When $\mathbf{x}$ is in the middle of $\mathcal{S}^+$ or $\mathcal{S}^-$, $\tilde{\theta}$ remains roughly the same; when it hits the edge of $|\lambda| = 1$, $\tilde{\theta}$ may jump from $\mathcal{S}^+$ or $\mathcal{S}^-$ to the other, giving rise to a sharp drop of the $\tilde{\theta}$ value.

Analytical approaches exist to approximate the zigzag curves. Taking Eq. 7.2.1 but neglecting the variance given by Eq. 7.2.2 leads to a reasonable estimate since the diffusion term is minimum, as shown by the black curve of Fig. 7.5. If we further expand Eq. 7.2.1, say, for the first case, we get

$$\lambda_{t+1} = \lambda_t(1 - \Delta\phi^2/2) - A_t \Delta\phi. \tag{7.2.5}$$

Dropping the term $\Delta\phi^2$, we have

$$-\frac{\Delta\lambda}{A(\lambda)} = \Delta\phi. \tag{7.2.6}$$

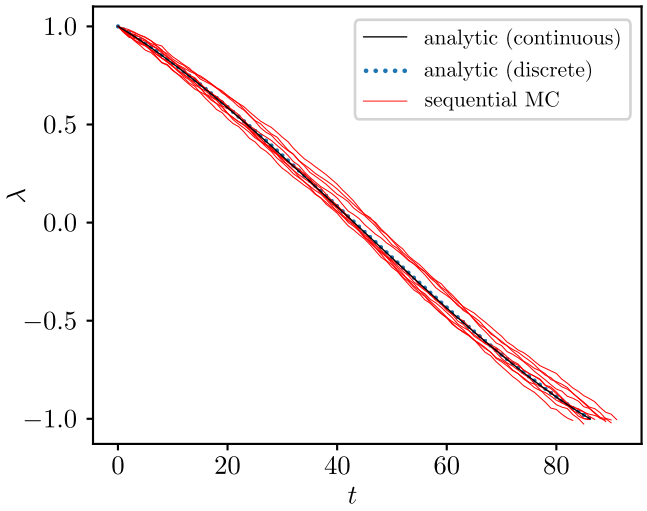


Figure 7.5: Simulated and ayalytical results for λ with $\eta = 2.0$ and $\Delta\phi = \pi/180$. All tests start with $\lambda = 1$. Red curves are simulated by Algo. 3, that is, our sequential Monte Carlo method. Blue dots are the discrete version of analytical results given by Eq. 7.2.1. Black curve is the continuous version given by Eq. 7.2.8.

This can further be approximated by a continuous differential equation

$$-\frac{\mathrm{d}\lambda}{\frac{1}{2}\left(\sqrt{\eta^2-\lambda^2}-\sqrt{1-\lambda^2}\right)} = \mathrm{d}\phi. \tag{7.2.7}$$

The solution of λ has an analytic expression

$$\phi - \phi_0 = \frac{-\lambda\sqrt{\eta^2-\lambda^2} - \eta^2 \arcsin\frac{\lambda}{\eta} + \lambda\sqrt{1-\lambda^2} + \arcsin\lambda}{\eta^2 - 1}. \tag{7.2.8}$$

Replacing ϕ with $t \sim \phi/\Delta\phi$, we reach a continuous relation of λ and t. Fig. 7.5 starts with $\lambda = 1$ and shows a good agreement between the Monte Carlo method, the discrete and the continuous analytical solutions.

7.2.2 Large-time limit

In this section, we study the change of the dipole angle $\tilde{\theta}$. In general, the expectation of $\tilde{\theta}$ can increase or decrease, depending on whether the initial $\mathbf{x}$ belongs to $\mathcal{S}^-$ or $\mathcal{S}^+$ region. To simplify the analysis, we assume an equilibrated initial distribution of $\mathbf{x}$.

First, if $\mathbf{x}_t$ and the induced θ_t have equilibrated uniform distributions, $\mathbf{x}_{t+1}$ and θ_{t+1} are uniformly distributed too. One can argue that $\pi(\tilde{\theta}_{t+1} - \tilde{\theta}_t)$ is symmetric around zero because the contributions from the points with $\tilde{\theta}_t = \tilde{\phi}_t + \phi'$ and those with $\tilde{\theta}_t = \tilde{\phi}_t - \phi'$ have opposite effects to the change of $\tilde{\theta}$. The zero mean is confirmed by Fig. 7.6 (a) . Furthermore, the change in the rolled-out angle after more than one iteration has zero mean as well, because

$$\mathbb{E}(\tilde{\theta}_t - \tilde{\theta}_0) = \sum_{i=1}^{t} \mathbb{E}(\tilde{\theta}_i - \tilde{\theta}_{i-1}) = 0. \tag{7.2.9}$$

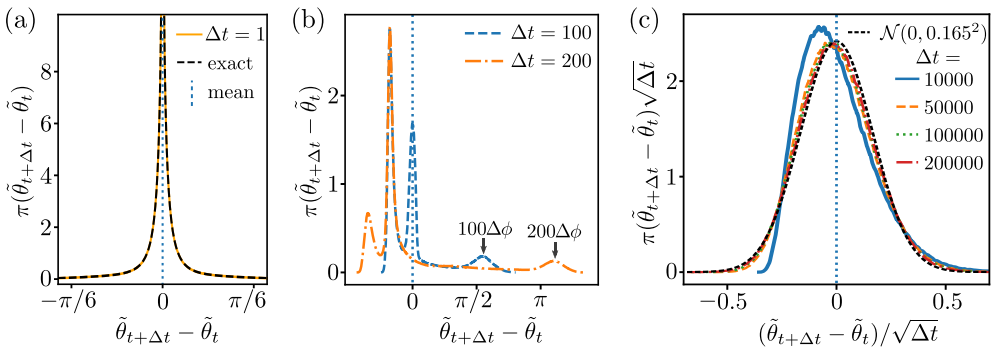


Figure 7.6: Distributions of $\tilde{\theta}_{t+\Delta t} - \tilde{\theta}_t$ for different values of Δt. $\eta = 1.1$ and $\Delta\phi = \pi/180$ are taken throughout. Blue dotted vertical lines indicate their zero mean values. Statistics are gained by 10^6 initial points randomly drawn from Ω. (a) $\Delta t = 1$, with the dashed line obtained by numerical integral. (b) Moderate Δt's. (c) Large Δt limit. (From Ref Ref [3].)

Each term of $\mathbb{E}(\tilde{\theta}_i - \tilde{\theta}_{i-1})$ vanishes because $\tilde{\theta}_{i-1}$ and $\mathbf{x}_{i-1}$ have uniform distrubtions. However, $\tilde{\theta}_t - \tilde{\theta}_0$ is no longer symmetric due to the shift of ϕ, as shown in Fig. 7.6 (b) and (c).

As Δt increases, $\pi(\tilde{\theta}_{t+\Delta t} - \tilde{\theta}_t)$ becomes asymmetric, with main part around zero and a small bump on the right. The position of the small bump agrees with the change of ϕ. It represents trajectories that stay in excursions, so the magnitude of the bump diminishes for larger Δt since the probability for always staying $|\lambda| > 1$ decays. For large Δt limit, the small bump is no longer visible, and the overall shape of $\pi(\tilde{\theta}_{t+\Delta t} - \tilde{\theta}_t)$ tends to a Gaussian distribution because it consists in contributions from different times that are uncorrelated. For $\eta = 1.1$ and $\Delta\phi = \pi/180$, Fig. 7.6 (c) tells us that $\pi(\tilde{\theta}_{t+\Delta t} - \tilde{\theta}_t)$ approaches a Gaussian distribution with zero mean and $0.165^2\Delta t$ variance.

7.3 Mixing of orientation

To measure the advantage of our method with sequential direction choices, we take the mixing time as the criterion. The mixing time is defined upon total variation distance given by Eq. 1.3.5, and by Levin's convention [29], it is the time at which the total variation distance to the equilibrated state reduces to $1/4$. And for our single dipole system, we are interested in algorithms that can fast explore the space of its orientation, so we project the probability of $\mathbf{x}$ onto θ component. The target distribution is $\pi(\theta) = 1/2\pi$. The mixing time in our case can be written as

$$t_{\text{mix}} = \min\left\{t \ : \|\pi(\theta_t) - \frac{1}{2\pi}\|_{\text{TVD}} \leq \frac{1}{4}\right\}. \tag{7.3.1}$$

Many factors affect the mixing time. First, we study the effect of the initial state. Levin and Peres define the mixing time as Eq. 7.3.1 given by the worst-case initial state. In order to identify this worst case, we tried different choices of initial θ and

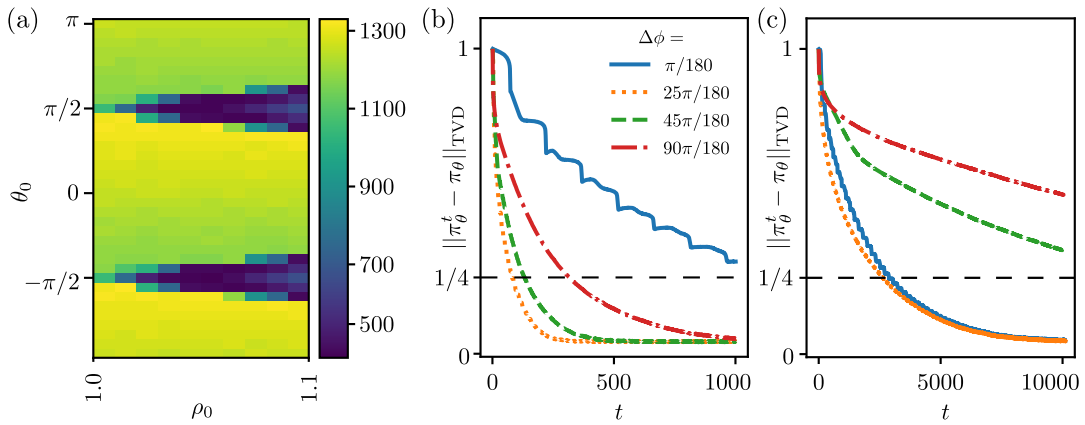


Figure 7.7: (a) Mixing times for various initial points, with $\eta = 1.1$ and $\Delta\phi = \pi/180$. (b) and (c) show the total variation distance with respect to time, for different values of $\Delta\phi$. (b) is for $\eta = 1.1$ and (c) is for $\eta = 1.01$. (Panel (a) is adapted from Ref [3].)

ρ, and the results are shown in the colormap of Fig. 7.7 (a) . According to Fig. 7.7 (a) , the initial state leading to the worst mixing time occurs near the sides with $|\lambda| = 1$, but the mixing times are very close for all initial points with $|\lambda| < 1$. Thus, we take $\theta_0 = 0$ and $\rho_0 = (r + R)/2$ for following measurements (ϕ_0 is always 0).

Shown in Fig. 7.7 (b) and (c) are total variation distances from $\pi(\theta_t)$ to the target $1/2\pi$, with increasing t. It first confirms that our method can reach the theoretical equilibrium, and different choices of $\Delta\phi$ have influence over the speed. For the case of $\eta = 1.1$ and $\eta = 1.01$, $\Delta\phi = 25°$ is optimal compared with $1°$, $45°$ and $90°$.

Furthermore, we investigate the effect of the series $\mathcal{D}_n$, i.e., $\Delta\phi = n\pi/180$. We wonder whether the elements of $\mathcal{D}_n$ have the major influence, or its order makes the main contribution. Hence, we perform the direct sampling Algo. 3 with $\mathcal{D}_n$, either sequentially or randomly, which are denoted as $(\mathcal{D}_n, \mathcal{S})$ and $(\mathcal{D}_n, \mathcal{R})$. Also, we tested the variant that randomly chooses directions with no constraint, i.e., directions from the unit circle, which is denoted as $\mathcal{D}_\infty$.

In Fig. 7.8 (a) , we plot the mixing times with respect to various values of $\mathcal{D}_n$, and they are chosen sequentially. We find that if n is very small, ϕ changes slowly, and the mixing time is large. However, t_{mix} has multiple peaks around certain values of n, and those peaks coincide with the inverse number[1] of elements in set $\mathcal{D}_n$, denoted as $|\mathcal{D}|^{-1}$. This coincidence implies that periodicity is a factor in choices of $\mathcal{D}$, and a large period (or aperiodicity) is preferred.

Then we compare the sequential selection of $\mathcal{D}_n$ and the random selection in Fig. 7.8 (b) , with $\eta = 1.005$. We find that sequential order always outperforms random order, and mixing times of random order are lower-bounded by that of $\mathcal{D}_\infty$. Compared with t_{mix} of $\mathcal{D}_\infty$ that is also the minimum for $(\mathcal{D}_n, \mathcal{R})$, $(\mathcal{D}_n, \mathcal{S})$ can reach a lower mixing time that is around 1.5 times faster. In Fig. 7.8 (c) we study the cases with $\eta = 1.01$ and $\eta = 1.05$. Peaks are less intense with increasing η, while the optimal $\mathcal{D}_n$'s in sequential order can all outperform random direction schemes.

[1]Note that set elements cannot repeat, and directions that differ by multiples of π are regarded identical. So $|\mathcal{D}_1| = 180$, $|\mathcal{D}_{30}| = 6$, and $|\mathcal{D}_{90}| = 2$.

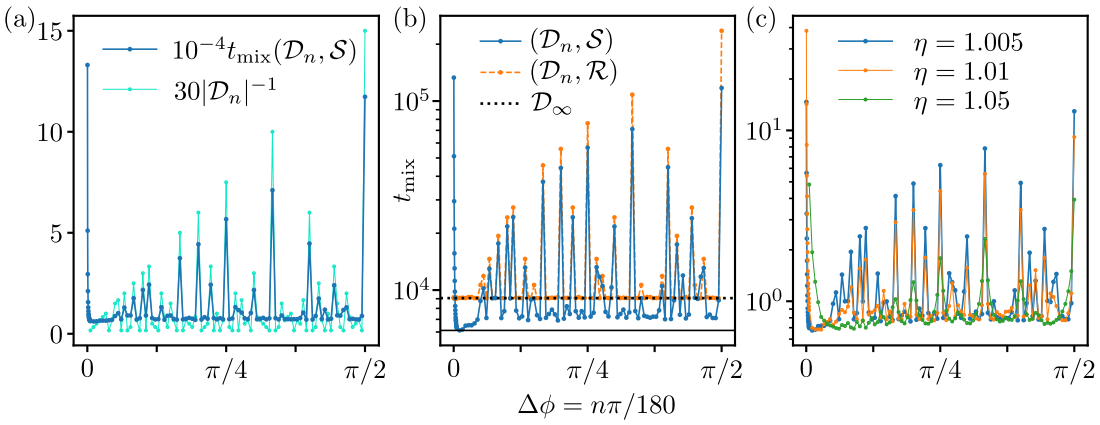


Figure 7.8: Mixing times with respect to $\mathcal{D}_n$, for random order ($\mathcal{R}$) and sequential order ($\mathcal{S}$). Dark blue lines in three panels are t_{mix}'s for $(\mathcal{D}_n, \mathcal{S})$. We compare it with : (a) inverse number of set elements $|\mathcal{D}_n|^{-1}$; (b) random order $(\mathcal{D}_n, \mathcal{R})$, and random arbitrary directions $\mathcal{D}_\infty$; (c) other η values. (Adapt from Ref [3].)

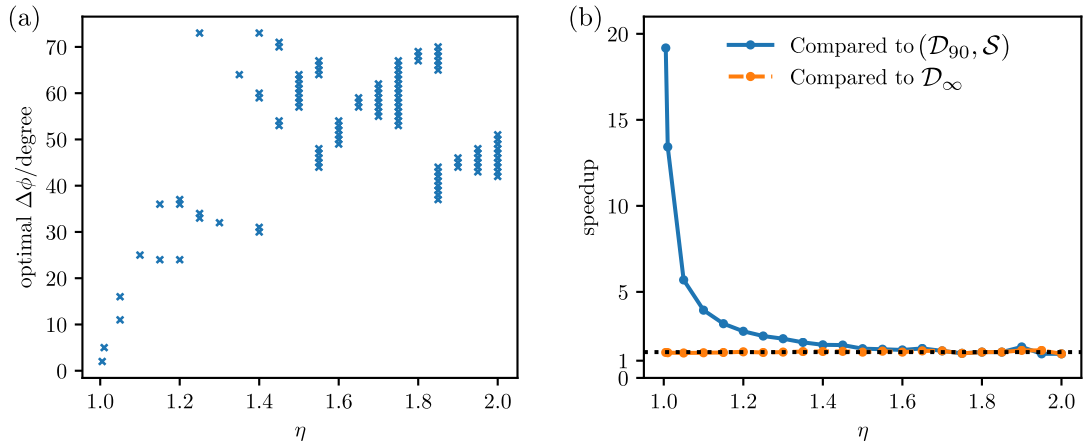


Figure 7.9: Optimal values of $\Delta\phi$ and overall accelerations using sequential order of $\mathcal{D}_n$, for $1.0 < \eta \leq 2.0$. (a) Values of $\Delta\phi$ that give smallest t_{mix}. (b) Speedup compared with $(\mathcal{D}_{90}, \mathcal{S})$ and $\mathcal{D}_\infty$. (Panel (b) is adapted from Ref [3].)

Finally, we measure the optimal choices of $\mathcal{D}_n$ that give the lowest mixing times and the overall speedup, as plotted in Fig. 7.9. The optimal $\Delta\phi$'s present a complicated pattern, and it is hard to extract a formula. Small values of η favor relatively small $\Delta\phi$, while when $\eta > 1.4$ the optimal $\Delta\phi$'s are between $\pi/4$ and $\pi/2$. As for the speedup, sequential directions can indeed accelerate, especially compared with directions constrained in $+x$ and $+y$ for small η. When we compare with arbitrary random directions, the sequential method has still around 1.5 times speedup for all $\eta < 2.0$. So far, we have only understood the mechanism for small $\Delta\phi$. Further theories need to be established for moderate values of $\Delta\phi$ and the optimal choice.

General Conclusion

During the past three years we aimed to apply the ECMC algorithm to long-range systems, in hope that ECMC that exhibits exciting speedup in hard-disk and spin systems can also outperform widespread molecular dynamics in biochemical simulations. The bottleneck to simulating a realistic system is the long-range Coulomb interaction, while ECMC with factorized filter provides an innovative approach by sampling from pairwise Coulomb factors.

The currently most efficient molecule dynamics have always invoked the N-body Coulomb computation. First, we rewrote the Coulomb interaction in a two-body form, since the ECMC's advantage relies on the pairwise factorization. We proved that the line-charge method proposed by Kapfer and Krauth [95] is an alternative of tin-foil electrostatics. With its inspiration, we proposed several other methods of fast convergence such as kite-charge and volume-charge models.

Second, for dipole systems, we exploited the freedom of factorization in ECMC to find that dipole factors are able to achieve $\mathrm{O}(N \log N)$ time complexity for a sweep. Interesting lifting schemes appeared for dipole systems, and we proposed two schemes in addition to the simplest ratio scheme. In particular, the inside-first lifting scheme which maximally favors next active atoms to be located within the same molecule, has a bounded inter-molecule lifting rate regardless of molecule numbers. We demenstrated these concepts in [1].

Then, we developed a Python application called JELLYFYSH for ECMC simulation of the particle systems, with belief that a universal scientific platform will greatly benefit researchers. For the first time, we implemented a complete prototype version of ECMC with the dipole factorization and the cell-veto algorithm. The structured codes allow us to easily customize physical settings and enrich the function of JF. The specification of JF was published in [2].

Furthermore, we studied the efficiency of ECMC through JELLYFYSH benchmark. It shows that Coulomb events cost most of the time, and in particular, it is the confirmed events that predominate time consumption due to the recalculation of candidate events. For this reason, dipole factors outperform atomic factors by three times. The autocorrelation of the total polarization presents double time scales, and the second significant slows down a water system's rotation. Enforcing molecule translation helps to alleviate the slowdown while the mechanism is still to be discovered.

Finally, our proposal of sequential Monte Carlo provides an approach to overcome the slow rotation. Tests over a planar dipole showed a speedup depending on the potential width and they led to asymmetric evolutions of the dipole orientation. We found that the inverse number of unique directions coincides with the speedup,

implying that the periodicity plays an important role in the optimal set of directions. Relaxing the dipole in an ordered sequence of directions is always preferred, compared to taking directions randomly from the sequence. We published the results in [3].

There are many open problems to be studied in the future. ECMC with pure atomic motions proved fast in mixing single-molecule orientations, while it is inefficient for the bulk-molecular rotation, thus the reason remains unknown for the efficiency gap between molecular dynamics and ECMC, just as rotating molecule can significantly speed up the decorrelation in traditional Monte Carlo methods. And also, we are not clear about the role of factorizations and lifting schemes in ECMC's dynamics. The correlation time drops dramatically when the optimal factorization is chosen as in the study of factor field, and here the three lifting schemes proposed in section 3.5 may bring distinct dynamics. Advantages of ECMC are more evident for atomic systems, where open questions include parallelization and the factor field beyond one dimension.

Bibliography

[1] M. F. Faulkner, L. Qin, A. C. Maggs & W. Krauth; *All-atom computations with irreversible Markov chains*; The Journal of chemical physics **149**, p. 064113 (2018).

[2] P. Höllmer, L. Qin, M. F. Faulkner, A. C. Maggs & W. Krauth; *JeLLyFysh-Version1. 0-a Python application for all-atom event-chain Monte Carlo*; Computer Physics Communications p. 107168 (2020).

[3] L. Qin, P. Hoellmer & W. Krauth; *Fast sequential Markov chains*; arXiv preprint arXiv:2007.15615 (2020).

[4] J. M. Kosterlitz & D. J. Thouless; *Ordering, metastability and phase transitions in two-dimensional systems*; Journal of Physics C: Solid State Physics **6**, p. 1181 (1973).

[5] B. Halperin & D. R. Nelson; *Theory of two-dimensional melting*; Physical Review Letters **41**, p. 121 (1978).

[6] D. R. Nelson & B. Halperin; *Dislocation-mediated melting in two dimensions*; Physical Review B **19**, p. 2457 (1979).

[7] A. Young; *Melting and the vector Coulomb gas in two dimensions*; Physical Review B **19**, p. 1855 (1979).

[8] B. Alder & T. Wainwright; *Phase transition in elastic disks*; Physical Review **127**, p. 359 (1962).

[9] E. P. Bernard & W. Krauth; *Two-step melting in two dimensions: First-order liquid-hexatic transition*; Physical review letters **107**, p. 155704 (2011).

[10] E. Bernard; *Algorithms and applications of the Monte Carlo method: Two-dimensional melting and perfect sampling*; Ph.D. thesis (2011).

[11] M. Engel, J. A. Anderson, S. C. Glotzer, M. Isobe, E. P. Bernard & W. Krauth; *Hard-disk equation of state: First-order liquid-hexatic transition in two dimensions with three simulation methods*; Physical Review E **87**, p. 042134 (2013).

[12] D. E. S. Research; *Molecular Dynamics Simulations Related to SARS-CoV-2*; (2020); d. E. Shaw Research Technical Data.

[13] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao *et al.*; *Anton, a special-purpose machine for molecular dynamics simulation*; Communications of the ACM **51**, pp. 91–97 (2008).

[14] S. Plimpton; *Fast parallel algorithms for short-range molecular dynamics*; Journal of computational physics **117**, pp. 1–19 (1995).

[15] D. Frenkel & B. Smit; *Understanding molecular simulation: from algorithms to applications*; volume 1 (Elsevier) (2001).

[16] B. Leimkuhler & C. Matthews; *Molecular Dynamics.* (Springer) (2016).

[17] G. Zhong & J. E. Marsden; *Lie-poisson hamilton-jacobi theory and lie-poisson integrators*; Physics Letters A **133**, pp. 134–139 (1988).

[18] C. L. Brooks III, B. M. Pettitt & M. Karplus; *Structural and energetic effects of truncating long ranged interactions in ionic and polar fluids*; The Journal of chemical physics **83**, pp. 5897–5908 (1985).

[19] T. Darden, D. York & L. Pedersen; *Particle mesh Ewald: An N· log (N) method for Ewald sums in large systems*; The Journal of chemical physics **98**, pp. 10089–10092 (1993).

[20] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee & L. G. Pedersen; *A smooth particle mesh Ewald method*; The Journal of chemical physics **103**, pp. 8577–8593 (1995).

[21] R. W. Hockney & J. W. Eastwood; *Computer simulation using particles* (crc Press) (1988).

[22] Y. Shan, J. L. Klepeis, M. P. Eastwood, R. O. Dror & D. E. Shaw; *Gaussian split Ewald: A fast Ewald mesh method for molecular simulation*; The Journal of chemical physics **122**, p. 054101 (2005).

[23] E. A. Proctor, F. Ding & N. V. Dokholyan; *Discrete molecular dynamics*; Wiley Interdisciplinary Reviews: Computational Molecular Science **1**, pp. 80–92 (2011).

[24] M. A. Khan & M. C. Herbordt; *Parallel discrete molecular dynamics simulation with speculation and in-order commitment*; Journal of computational physics **230**, pp. 6563–6582 (2011).

[25] M. N. Bannerman, R. Sargant & L. Lue; *DynamO: a free O(N) general event-driven molecular dynamics simulator*; Journal of computational chemistry **32**, pp. 3329–3338 (2011).

[26] B. D. Lubachevsky; *Simulating colliding rigid disks in parallel using bounded lag without time warp*; Distributed Simulation **22**, pp. 194–202 (1990).

[27] B. D. Lubachevsky; *Simulating billiards: Serially and in parallel*; International Journal in Computer Simulation **2**, pp. 373–411 (1992).

[28] B. Li, S. Todo, A. C. Maggs & W. Krauth; *Multithreaded event-chain Monte Carlo with local times*; arXiv preprint arXiv:2004.11040 (2020).

[29] D. A. Levin & Y. Peres; *Markov chains and mixing times*; volume 107 (American Mathematical Soc.) (2017).

[30] W. Krauth; *Statistical mechanics: algorithms and computations*; volume 13 (OUP Oxford) (2006).

[31] P. L'Ecuyer, M. Mandjes, B. Tuffin *et al.*; *Importance sampling and rare event simulation*; Rare event simulation using Monte Carlo methods pp. 17–38 (2009).

[32] C. Mak; *Stochastic potential switching algorithm for Monte Carlo simulations of complex systems*; The Journal of chemical physics **122**, p. 214110 (2005).

[33] R. H. Swendsen & J.-S. Wang; *Nonuniversal critical dynamics in Monte Carlo simulations*; Physical review letters **58**, p. 86 (1987).

[34] E. Marinari & G. Parisi; *Simulated tempering: a new Monte Carlo scheme*; EPL (Europhysics Letters) **19**, p. 451 (1992).

[35] K. Hukushima & K. Nemoto; *Exchange Monte Carlo method and application to spin glass simulations*; Journal of the Physical Society of Japan **65**, pp. 1604–1608 (1996).

[36] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller & E. Teller; *Equation of state calculations by fast computing machines*; The journal of chemical physics **21**, pp. 1087–1092 (1953).

[37] W. K. Hastings; *Monte Carlo sampling methods using Markov chains and their applications*; (1970).

[38] A. A. Barker; *Monte carlo calculations of the radial distribution functions for a proton? electron plasma*; Australian Journal of Physics **18**, pp. 119–134 (1965).

[39] S. L. Adler; *Over-relaxation method for the Monte Carlo evaluation of the partition function for multiquadratic actions*; Physical Review D **23**, p. 2901 (1981).

[40] C. Whitmer; *Over-relaxation methods for Monte Carlo simulations of quadratic and multiquadratic actions*; Physical Review D **29**, p. 306 (1984).

[41] F. R. Brown & T. J. Woch; *Overrelaxed heat-bath and Metropolis algorithms for accelerating pure gauge Monte Carlo calculations*; Physical Review Letters **58**, p. 2394 (1987).

[42] P. J. Green & X.-l. Han; *Metropolis methods, Gaussian proposals and antithetic variables*; in *Stochastic Models, Statistical methods, and Algorithms in Image Analysis,* pp. 142–164 (Springer) (1992).

[43] R. M. Neal; *Suppressing random walks in Markov chain Monte Carlo using ordered overrelaxation*; in *Learning in graphical models*, pp. 205–228 (Springer) (1998).

[44] E. Luijten; *Introduction to cluster Monte Carlo algorithms*; in *Computer Simulations in Condensed Matter Systems: From Materials to Chemical Biology Volume 1*, pp. 13–38 (Springer) (2006).

[45] M. Michel; *Irreversible Markov chains by the factorized Metropolis filter: algorithms and applications in particle systems and spin models*; Ph.D. thesis (2016).

[46] U. Wolff; *Collective Monte Carlo updating for spin systems*; Physical Review Letters **62**, p. 361 (1989).

[47] V. Cataudella, G. Franzese, M. Nicodemi, A. Scala & A. Coniglio; *Critical clusters and efficient dynamics for frustrated spin models*; Physical review letters **72**, p. 1541 (1994).

[48] S. Liang; *Application of cluster algorithms to spin glasses*; Physical review letters **69**, p. 2145 (1992).

[49] C. Dress & W. Krauth; *Cluster algorithm for hard spheres and related systems*; Journal of Physics A: Mathematical and General **28**, p. L597 (1995).

[50] A. Jaster; *An improved Metropolis algorithm for hard core systems*; Physica A: Statistical Mechanics and its Applications **264**, pp. 134–141 (1999).

[51] J. I. Siepmann & D. Frenkel; *Configurational bias Monte Carlo: a new sampling scheme for flexible chains*; Molecular Physics **75**, pp. 59–70 (1992).

[52] K. S. Turitsyn, M. Chertkov & M. Vucelja; *Irreversible Monte Carlo algorithms for efficient sampling*; Physica D: Nonlinear Phenomena **240**, pp. 410–414 (2011).

[53] R. Ren & G. Orkoulas; *Acceleration of Markov chain Monte Carlo simulations through sequential updating*; The Journal of chemical physics **124**, p. 064109 (2006).

[54] P. H. Peskun; *Optimum monte-carlo sampling using markov chains*; Biometrika **60**, pp. 607–612 (1973).

[55] R. Ren & G. Orkoulas; *Parallel markov chain monte carlo simulations*; (2007).

[56] Z. Lei & W. Krauth; *Mixing and perfect sampling in one-dimensional particle systems*; EPL (Europhysics Letters) **124**, p. 20003 (2018).

[57] H. Suwa & S. Todo; *Markov chain Monte Carlo method without detailed balance*; Physical review letters **105**, p. 120603 (2010).

[58] S. Todo & H. Suwa; *Geometric allocation approaches in Markov chain Monte Carlo*; in *Journal of Physics: Conference Series*, , volume 473p. 012013 (IOP Publishing) (2013).

[59] Y. Mori & H. Okumura; *Simulated tempering based on global balance or detailed balance conditions: S uwa–t odo, heat bath, and m etropolis algorithms*; Journal of computational chemistry **36**, pp. 2344–2349 (2015).

[60] S. G. Itoh & H. Okumura; *Replica-permutation method with the Suwa–Todo algorithm beyond the replica-exchange method*; Journal of chemical theory and computation **9**, pp. 570–581 (2013).

[61] S. Duane, A. D. Kennedy, B. J. Pendleton & D. Roweth; *Hybrid monte carlo*; Physics letters B **195**, pp. 216–222 (1987).

[62] S. Brooks, A. Gelman, G. Jones & X.-L. Meng; *Handbook of markov chain monte carlo* (CRC press) (2011).

[63] B. J. Culpepper, J. Sohl-Dickstein & B. A. Olshausen; *Building a better probabilistic model of images by factorization*; in *2011 International Conference on Computer Vision,* (IEEE) (2011).

[64] P. Diaconis, S. Holmes & R. M. Neal; *Analysis of a nonreversible Markov chain sampler*; Annals of Applied Probability pp. 726–752 (2000).

[65] F. Chen, L. Lovász & I. Pak; *Lifting Markov chains to speed up mixing*; in *Proceedings of the thirty-first annual ACM symposium on Theory of computing,* pp. 275–281 (1999).

[66] J. Sohl-Dickstein, M. Mudigonda & M. R. DeWeese; *Hamiltonian Monte Carlo without detailed balance*; arXiv preprint arXiv:1409.5191 (2014).

[67] M. Vucelja; *Lifting—a nonreversible Markov chain Monte Carlo algorithm*; American Journal of Physics **84**, pp. 958–968 (2016).

[68] F. Faizi, G. Deligiannidis & E. Rosta; *Efficient Irreversible Monte Carlo Samplers*; Journal of Chemical Theory and Computation **16**, pp. 2124–2138 (2020).

[69] H. C. Fernandes & M. Weigel; *Non-reversible Monte Carlo simulations of spin models*; Computer Physics Communications **182**, pp. 1856–1859 (2011).

[70] Y. Sakai & K. Hukushima; *Eigenvalue analysis of an irreversible random walk with skew detailed balance conditions*; Physical Review E **93**, p. 043318 (2016).

[71] Y. Sakai & K. Hukushima; *Dynamics of one-dimensional Ising model without detailed balance condition*; Journal of the Physical Society of Japan **82**, p. 064003 (2013).

[72] K. Hukushima & Y. Sakai; *An irreversible Markov-chain Monte Carlo method with skew detailed balance conditions*; in *Journal of Physics: Conference Series,* , volume 473p. 012012 (IOP Publishing) (2013).

[73] Y. Sakai & K. Hukushima; *Irreversible simulated tempering*; Journal of the Physical Society of Japan **85**, p. 104002 (2016).

[74] Y. Wu, H. L. Tepper & G. A. Voth; *Flexible simple point-charge water model with improved liquid-state properties*; The Journal of chemical physics **124**, p. 024503 (2006).

[75] C. Chen, P. Depa, V. G. Sakai, J. K. Maranas, J. W. Lynn, I. Peral & J. R. Copley; *A comparison of united atom, explicit atom, and coarse-grained simulation models for poly (ethylene oxide)*; The Journal of chemical physics **124**, p. 234901 (2006).

[76] E. P. Bernard, W. Krauth & D. B. Wilson; *Event-chain Monte Carlo algorithms for hard-sphere systems*; Physical Review E **80**, p. 056704 (2009).

[77] E. A. Peters *et al.*; *Rejection-free Monte Carlo sampling for general potentials*; Physical Review E **85**, p. 026703 (2012).

[78] J. Harland, M. Michel, T. A. Kampmann & J. Kierfeld; *Event-chain Monte Carlo algorithms for three-and many-particle interactions*; EPL (Europhysics Letters) **117**, p. 30001 (2017).

[79] M. Michel, S. C. Kapfer & W. Krauth; *Generalized event-chain Monte Carlo: Constructing rejection-free global-balance algorithms from infinitesimal steps*; The Journal of chemical physics **140**, p. 054116 (2014).

[80] M. Isobe & W. Krauth; *Hard-sphere melting and crystallization with event-chain Monte Carlo*; The Journal of chemical physics **143**, p. 084509 (2015).

[81] S. C. Kapfer & W. Krauth; *Soft-disk melting: From liquid-hexatic coexistence to continuous transitions*; arXiv preprint arXiv:1406.7224 (2014).

[82] M. Michel, J. Mayer & W. Krauth; *Event-chain Monte Carlo for classical continuous spin models*; EPL (Europhysics Letters) **112**, p. 20003 (2015).

[83] Z. Lei & W. Krauth; *Irreversible Markov chains in spin models: Topological excitations*; EPL (Europhysics Letters) **121**, p. 10008 (2018).

[84] Y. Nishikawa, M. Michel, W. Krauth & K. Hukushima; *Event-chain algorithm for the Heisenberg model: Evidence for $z \simeq 1$ dynamic scaling*; Physical Review E **92**, p. 063306 (2015).

[85] S. C. Kapfer & W. Krauth; *Irreversible local markov chains with rapid convergence towards equilibrium*; Physical review letters **119**, p. 240603 (2017).

[86] Z. Lei, W. Krauth & A. C. Maggs; *Event-chain Monte Carlo with factor fields*; Physical Review E **99**, p. 043301 (2019).

[87] T. A. Kampmann, H.-H. Boltz & J. Kierfeld; *Monte Carlo simulation of dense polymer melts using event chain algorithms*; The Journal of chemical physics **143**, p. 044105 (2015).

[88] M. Hasenbusch & S. Schaefer; *Testing the event-chain algorithm in asymptotically free models*; Physical Review D **98**, p. 054502 (2018).

[89] M. Michel & S. Sénécal; *Forward Event-Chain Monte Carlo: a general rejection-free and irreversible Markov chain simulation method*; arXiv preprint arXiv:1702.08397 (2017).

[90] R. Weigel; *Equilibration of orientational order in hard disks via arcuate event-chain Monte Carlo*; Ph.D. thesis; MS thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (2018).

[91] A. Bouchard-Côté, S. J. Vollmer & A. Doucet; *The bouncy particle sampler: A nonreversible rejection-free Markov chain Monte Carlo method*; Journal of the American Statistical Association **113**, pp. 855–867 (2018).

[92] J. Bierkens, P. Fearnhead, G. Roberts *et al.*; *The zig-zag process and super-efficient sampling for Bayesian analysis of big data*; The Annals of Statistics **47**, pp. 1288–1320 (2019).

[93] S. C. Kapfer & W. Krauth; *Sampling from a polytope and hard-disk Monte Carlo*; in *Journal of Physics: Conference Series, ,* volume 454p. 012031 (IOP Publishing) (2013).

[94] S. W. de Leeuw, J. W. Perram & E. R. Smith; *Simulation of electrostatic systems in periodic boundary conditions. I. Lattice sums and dielectric constants*; Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences **373**, pp. 27–56 (1980).

[95] S. C. Kapfer & W. Krauth; *Cell-veto Monte Carlo algorithm for long-range systems*; Physical Review E **94**, p. 031302 (2016).

[96] C. Bender & S. Orszag; *Advanced Mathematical Methods for Scientists and Engineers I: Asymptotic Methods and Perturbation Theory*; Advanced Mathematical Methods for Scientists and Engineers (Springer) (1978); ISBN 9780387989310.

[97] H. J. Berendsen, D. van der Spoel & R. van Drunen; *GROMACS: a message-passing parallel molecular dynamics implementation*; Computer physics communications **91**, pp. 43–56 (1995).

[98] R. Salomon-Ferrer, D. A. Case & R. C. Walker; *An overview of the Amber biomolecular simulation package*; Wiley Interdisciplinary Reviews: Computational Molecular Science **3**, pp. 198–210 (2013).

[99] E. Gamma; *Design patterns: elements of reusable object-oriented software* (Pearson Education India) (1995).

[100] C. O'Keeffe & G. Orkoulas; *Parallel canonical Monte Carlo simulations through sequential updating of particles*; The Journal of chemical physics **130**, p. 134109 (2009).

[101] G. Orkoulas; *Spatial updating Monte Carlo algorithms in particle simulations*; Molecular Simulation **36**, pp. 516–525 (2010).

Appendix A

Publication I

All-atom computations with irreversible Markov chains

Michael F. Faulkner, Liang Qin, A. C. Maggs, and Werner Krauth
J. Chem. Phys. 149, 064113 (2018)

This article contains the published result described in chapter 3. First, it gives the basics of ECMC and the cell-veto framework to deal with long-range interactions. Then it introduces our new work in extending ECMC into Coulomb systems. The methods for three-dimensional Coulomb systems require two additional elements — the computation of Coulomb potential derivatives, and the dipole factorization. Our Coulomb computations are based on Ewald's formula of electrostatics and the line-charge method. We adapted Ewald's formula to the sum of two-body interactions, and proposed other methods that all converge to tin-foil electrostatics.

The dipole factor is an application of the many-body ECMC method. It groups all pairs of Coulomb interactions between two dipoles into one factor. By exploiting the charge-dipole interaction, it is the first Monte Carlo algorithm that achieves $\mathrm{O}(N \log N)$-per-sweep complexity. Two specific lifting schemes are also proposed favoring intra-dipole or inter-dipole probabilities flows.

We test our methods with Coulomb atom, dipole, and SPC/Fw water systems.

THE JOURNAL OF CHEMICAL PHYSICS **149**, 064113 (2018)

All-atom computations with irreversible Markov chains

Michael F. Faulkner,[1] Liang Qin,[2] A. C. Maggs,[3] and Werner Krauth[2,4]
[1]*H. H. Wills Physics Laboratory, University of Bristol, Tyndall Avenue, Bristol BS8 1TL, United Kingdom*
[2]*Laboratoire de Physique Statistique, Département de physique de l'ENS, Ecole Normale Supérieure, PSL Research University, Université Paris Diderot, Sorbonne Paris Cité, Sorbonne Universités, UPMC Université Paris 06, CNRS, 75005 Paris, France*
[3]*CNRS UMR7083, ESPCI Paris, PSL Research University, 10 rue Vauquelin, 75005 Paris, France*
[4]*Max-Planck-Institut für Physik komplexer Systeme, Nöthnitzer Str. 38, 01187 Dresden, Germany*

(Received 17 April 2018; accepted 21 July 2018; published online 13 August 2018)

We apply the irreversible event-chain Monte Carlo (ECMC) algorithm to the simulation of dense all-atom systems with long-range Coulomb interactions. ECMC is event-driven and exactly samples the Boltzmann distribution. It neither uses time-step approximations nor spatial cutoffs on the range of the interaction potentials. Most importantly, it need not evaluate the total Coulomb potential and thus circumvents the major computational bottleneck of traditional approaches. It only requires the derivatives of the two-particle Coulomb potential, for which we discuss mutually consistent choices. ECMC breaks up the total interaction potential into factors. For particle systems made up of neutral dipolar molecules, we demonstrate the superior performance of dipole–dipole factors that do not decompose the Coulomb potential beyond the two-molecule level. We demonstrate that these long-range factors can nevertheless lead to local lifting schemes, where subsequently moved particles are mostly close to each other. For the simple point-charge water model with flexible molecules (SPC/Fw), which combines the long-ranged intermolecular Coulomb potential with hydrogen–oxygen bond-length vibrations, a flexible hydrogen–oxygen–hydrogen bond angle, and Lennard-Jones oxygen–oxygen potentials, we break up the potential into factors containing between two and six particles. For this all-atom liquid-water model, we demonstrate that the computational complexity of ECMC scales very well with the system size. This is achieved in a pure particle–particle framework, without the interpolating mesh required for the efficient implementation of other modern Coulomb algorithms. Finally, we discuss prospects and challenges for ECMC and outline several future applications. *Published by AIP Publishing.* https://doi.org/10.1063/1.5036638

I. INTRODUCTION

A. Irreversible Markov processes

Numerical methods are ubiquitous in the natural sciences, with Markov-chain Monte Carlo (MCMC)[1] and molecular dynamics[2] playing central roles. Markov-chain Monte Carlo applies to any computational science problem that can be formulated as an (perhaps fictitious) equilibrium-statistical-physics system and whose solution requires sampling its probability distribution. As in physical and chemical systems, equilibrium within the computational context usually means that all probability flows vanish. This requirement is enforced by the detailed-balance condition, an essential ingredient of most Markov-chain Monte Carlo methods and notably of the Metropolis algorithm.[3] Monte Carlo algorithms usually take much time to approach equilibrium[4] and, once in equilibrium, to generate independent samples. This is, in part, due to the fact that detailed balance leads to time-reversible Markov-chain dynamics, which is diffusive and therefore slow.

In recent years, a new class of irreversible "event-chain" Monte Carlo (ECMC) algorithms has been proposed.[5,6] ECMC algorithms violate detailed balance but satisfy a weaker global-balance condition. Configurations at large times sample the equilibrium distribution, but the asymptotic steady state comes with non-vanishing probability flows. In particle systems with periodic boundary conditions, for example, atoms may continue to move preferentially in certain directions. In continuous spin systems, likewise, configurations realize the equilibrium distribution even though spins rotate in a preferred way.[7–9] ECMC moves (displacements of particles, rotations of spins, etc.) are infinitesimal and persistent: An "active" particle moves directly from one event to the next, that is, it continues to move until a proposed move is vetoed by a unique "target" particle, which in turn becomes the active particle. This passing of the active-particle label is called a lifting,[10,11] and this concept overcomes the characteristic rejections of randomly proposed finite moves in the Metropolis algorithm. The ECMC algorithm was instrumental in the solution of the hard-disk melting problem, after decades of debate.[12,13] For soft potentials,[6] it can decorrelate with a smaller dynamical exponent than the local Metropolis algorithm.[7,9] Furthermore, in a one-dimensional particle system, ECMC was demonstrated to mix on shorter time scales than Markov chains that satisfy detailed balance.[14,15]

In ECMC, the traditional Metropolis acceptance criterion based on the change in potential is replaced by a consensus rule. This is the essence of the factorized Metropolis filter, which applies to translation-invariant systems with pairwise interactions between particles[6] and, more generally, to

models whose interactions can be split into sets of independent factors.[16] The ECMC algorithm does not compute the total system potential energy. This makes it very appealing for long-range-interacting systems, where this computation is costly. For Coulomb systems, ECMC altogether avoids traditional algorithms for the electrostatic potential,[17] the dominant computational bottleneck for long-range-interacting models. Rather, the cell-veto algorithm[18] efficiently establishes consensus on the acceptance or the rejection of a proposed move, even if all particles interact with each other. This is the starting point for the present work.

Generally, computations in statistical physics fall into two categories. They either aim at thermodynamic averages (energy, specific heat, spatial correlation functions, etc.) or at dynamic properties (time correlations, nucleation barriers, coarsening, etc.). In principle, the computation of thermodynamic averages is the realm of Markov-chain Monte Carlo, whereas the analysis of dynamical behavior calls on molecular dynamics, as it solves Newton's equations of motion. Specifically, however, the field of large-scale all-atom computations with long-ranged interactions is today dominated by molecular dynamics for both categories. The dominance of molecular dynamics is rooted in two facts: First, traditional Monte Carlo methods usually update just $\mathcal{O}(1)$ particles at a time, and the acceptance/rejection step is then followed by the exact computation of the change in potential. The best currently known algorithm[19] for the change in potential after such a local update in a Coulomb system is of complexity $\mathcal{O}(\sqrt{N})$ so that one Monte Carlo sweep (a sequential update of all N particles) requires $\mathcal{O}(N^{3/2})$ computations. In molecular dynamics, by contrast, the discretized Newton's equations update all particle positions simultaneously, and the necessary computation of the forces on all particles comes at a cost of $\mathcal{O}(N \log N)$, much less than for a Monte Carlo sweep. Second, Newtonian dynamics conserves momentum and explores phase space more efficiently than the local Metropolis algorithm. This advantage of molecular dynamics over Monte Carlo is, for example, brought out by the different scaling of the velocity auto-correlation functions in the context of long-time tails.[20,21]

The time evolution of molecular dynamics has physical meaning, but from an algorithmic point of view, it is constrained by the requirement that it must implement Newton's law. As a result, there is no additional freedom to accelerate the exploration of phase space. By contrast, Monte Carlo dynamics is non-physical and only constrained by the global-balance condition. A well-chosen Monte Carlo dynamics can considerably speed up the sampling of the equilibrium distribution. Those equilibrium samples may also serve as starting configurations for parallel molecular-dynamics calculations that give access to high-precision dynamical correlation functions. Furthermore, if more complex out-of-equilibrium rare-event physical phenomena (such as protein folding) are of interest, the time scales of long-time features can be accessed by the inspection of the rare events produced by parallel simulation on N_{proc} processors. Similar to the half-life analysis of radioactive substances composed of large numbers of atoms, a rare event that takes place on a time scale τ on a single processor will then take place on a time scale τ/N_{proc} on one of the N_{proc} processors.

In this work, we develop the framework for the application of ECMC to classical long-range-interacting all-atom systems. In particular, we demonstrate efficient ECMC methods that rigorously sample the canonical ensemble, without even evaluating the total potential. The factorizations that we implement with the cell-veto algorithm allow us to move a single particle from one event to the next in a computer run time that is independent of the number of point charges in a system. For a local charge-neutral system (for example, collections of charge-neutral many-atom molecules), the mean-free path (the mean distance between events) decreases only logarithmically with the number of point charges in the system. This implies that the computational effort required to move every particle in a simulation a constant distance scales as only $\mathcal{O}(N \log N)$, with no approximation and without the demanding interpolation onto the mesh that is introduced in many modern electrostatic simulations.

We validate our algorithm through explicit comparisons with a standard Metropolis algorithm and with molecular-dynamics simulations, each performed with Ewald summations. We focus on two conceptual issues. One is the computation of Coulomb pair-event rates, that is, essentially, the derivatives of the two-particle Coulomb potential with respect to the position of the "active" particle. In the simplest version of ECMC, this corresponds to the probability with which an active particle will stop and induce a lifting to another particle. The other issue concerns the factorization schemes of the system potential in which we lump together different interactions that partially compensate each other so that the ECMC mean-free path between events is much increased. We first apply our ECMC algorithm to a pair of like Coulomb point charges and then to systems of charge-neutral dipoles in a three-dimensional simulation box with periodic boundary conditions. We finally demonstrate the perfect agreement of thermodynamic observables between ECMC and conventional Monte Carlo and molecular dynamics for up to 256 water molecules at the standard density and temperature. The freedom offered by ECMC in choosing dynamics, factor decompositions, and lifting schemes leaves ample room for improvements. We expect it to be widely applicable to all-atom simulations of charged systems.

B. All-atom molecular simulations

Of great importance in soft-matter research, biological physics, and related fields, the all-atom approach projects the full quantum-mechanical many-body system onto the reduced classical degrees of freedom of the atomic positions. The projection yields the potential energy as a function of all the particle positions, and the Monte Carlo method can then, in principle, be applied directly. Molecular dynamics also starts from the atomic potential, as the forces in Newton's equations are given by its spatial derivatives. Present-day parametrized empirical force-field models[22,23] further break up the potentials and make them amenable to practical computations. For example, separate terms in the potential typically describe deviations of chemical bonds from their equilibrium values, with individual contributions for stretching, bending, and torsion. Likewise, distinct intermolecular

potentials capture longer-ranged features of the interactions; for example, dispersion forces, hard-core repulsions, and long-ranged charge–charge and dipolar interactions. The all-atom reduction from quantum mechanics to a classical interacting system is approximate and not uniquely defined. Various force-field models are used in a number of code bases,[24,25] which are also implemented in other prominent codes.[26–28] The parameters in each force-field model are optimized to reproduce thermodynamic and structural features over a reduced range of temperatures and pressures. Different potential functions coexist even for the description of simple molecules such as water.[29] We use in this work an all-atom potential for water that features two-body bond stretching, three-body bending as well as long-ranged Coulomb interactions, and a Lennard-Jones (LJ) potential.[30]

Modern codes generally compute the long-ranged Coulomb potential through variants of the Ewald algorithm applied to a discretized analog of the continuous position space. The Fourier contribution to the potential is evaluated by first interpolating each point charge to multiple points on a mesh and then solving the Poisson equation via fast Fourier transform, which, combined, is of complexity $\mathcal{O}(N \log N)$ per computation of the potential energy. Numerous formulations of this algorithm have been developed starting with the particle–particle–particle–mesh method.[31] More recent generations combine the particle–mesh philosophy with the Ewald formula, to create the particle–mesh–Ewald method[32] together with many variants[33–35] which, together, remain the workhorse of modern simulation codes. The charge interpolation onto the mesh generally presents the main computational workload. These methods use intricate strategies to maintain a high level of accuracy. Mesh interpolation leads to very large self-energy artifacts which have to be subtracted with great care in order not to modify the physical interactions.

Alternative approaches exist for the computation of the Coulomb potential and the electrostatic forces on particles. The hierarchical multipole-moment expansion,[36] for example, expands the interactions of a particle with all the other particles in terms of spherical harmonics, and therefore avoids Fourier transforms and mesh interpolations. However, the expansion converges only with high orders of the multipole moments so that one molecular-dynamics time step, although it is of complexity $\mathcal{O}(N)$, comes with a prohibitive prefactor. Local algorithms that propagate electric fields rather than solve the Poisson equation also bypass the fast Fourier transform.[37–39] This is an advantage in architectures where the Fourier transform involves large-scale non-local information transfers. In these algorithms, the complexity of a single-particle update is $\mathcal{O}(1)$ but the use of a background mesh to discretize the electrostatic degrees of freedom again leads to costly interpolations from the continuum charges to the discrete space.[40,41] In contrast to well-established methods, ECMC is directly formulated in continuous space, and its successful implementation only relies on translational invariance on all length scales. In essence, ECMC requires no discretization of the simulation box, and the total Coulomb potential and forces may remain unknown throughout the simulation.

All-atom molecular-dynamics simulations must take into account a variety of time scales and lengths. Indeed, the high-precision time integration of intramolecular spring forces requires a discretization time in the femtosecond range. The physics associated with the much longer time scales that one wishes to study include density fluctuations (which relax on the picosecond time scale), Debye-layer equilibration (nanoseconds), and conformation changes (milliseconds). At the same time, the precise rendering of dielectric and screening properties requires high-quality computations, and the long-ranged nature of the interaction calls for large system sizes in order to overcome finite-size effects. In order to efficiently manage both the stiffness (the presence of many relevant time scales) and long-ranged potentials, interactions are often broken up and sophisticated multiple time-step algorithms are implemented.[42,43] Use of a thermostat[44] is crucial in order to counteract a drift of the system energy and to connect the potential-energy surface with the system temperature. The ECMC algorithm considers the same potentials as its competitors, but it is fundamentally event-driven.[45] This is exceptional in the presence of continuous potentials, whereas the event-driven formulation for hard-sphere[5] or for stepped potentials[46] finds its correspondence in event-driven molecular-dynamics algorithms.[47,48] In the absence of discrete-time approximations, the exact Boltzmann distribution is sampled at any given temperature. This renders the thermostat unnecessary. In our application, the triggering of events remains well balanced between intramolecular, short-range intermolecular, and long-ranged intermolecular Coulomb events.

The remainder of this work is split into two parts. Part I corresponds to Sec. II, where we review recent advances in ECMC. We present the literature in a consistent mathematical language, which provides a unified framework for the application of ECMC to Coulomb systems. In Part II (Secs. III–V), we apply this framework to all-atom computations of various systems with Coulomb interactions. We demonstrate both the convergence to the Boltzmann distribution and the algorithmic scaling discussed in Sec. I A. The precise comparison of computer run time with established molecular-dynamics and Monte Carlo algorithms is, however, not presented in this work.

II. ECMC ALGORITHM

ECMC[5,6] is an irreversible continuous-time Markov process: Its moves are thus infinitesimal. Analogously, Newton's differential equations are of course also defined in continuous time. The molecular-dynamics algorithms that solve Newton's equations must be time-discretized for all systems except for hard spheres[2] or for stepwise constant potentials.[47,48] By contrast, in ECMC, discretization is generally avoided through the event-driven approach. In the present section, we discuss the essential issues of the algorithm's setup and implementation as well as its complexity.

A. Factors, factorized Metropolis filter

In ECMC, the interactions in an N-particle system are split into a finite or infinite set of factors $M = (I_M, T_M) \in \mathcal{P}(\{1, \ldots, N\}) \times \mathcal{T}$, where $\mathcal{P}$ is the power set of the indices

(comprising all indices, pairs of indices, triplets, etc), and $\mathcal{T}$ is a set of interaction types. We refer to I_M as the index set of the factor and to T_M as its type. The total potential U, which is a function of all particle positions $\{\mathbf{r}_1, \ldots, \mathbf{r}_N\}$, is written as a sum over factor potentials U_M

$$U(\{\mathbf{r}_1, \ldots, \mathbf{r}_N\}) = \sum_{M \in \mathcal{M}} U_M(\{\mathbf{r}_i : i \in I_M\}), \tag{1}$$

where U_M only depends on the factor indices I_M and is of type T_M. In Eq. (1), the set $\mathcal{M} = \{M : U_M \neq 0\} \subset \mathcal{P}(\{1, \ldots, N\}) \times \mathcal{T}$ only contains factors that have a non-zero contribution for some values of the positions. In a system with only pair interactions, a non-zero factor may be $(\{i,j\}, \text{pair})$. The corresponding factor potential would then be $U_{(\{i,j\},\,\text{pair})}(\mathbf{r}_i, \mathbf{r}_j)$, and the total potential in Eq. (1) then becomes $U = \sum_{i<j} U_{(\{i,j\},\,\text{pair})}(\mathbf{r}_i, \mathbf{r}_j)$, which is normally written as $U = \sum_{i<j} U_{\text{pair}}(\mathbf{r}_i, \mathbf{r}_j)$.

In this work, we use more general factorizations. The Lennard-Jones factor, that we write as $(\{i,j\}, \text{LJ})$, has a factor potential

$$U_{(\{i,j\},\text{LJ})}(\mathbf{r}_{ij}) = k_{\text{LJ}} \left[\left(\frac{\sigma}{|\mathbf{r}_{ij}|} \right)^{12} - \left(\frac{\sigma}{|\mathbf{r}_{ij}|} \right)^{6} \right], \tag{2}$$

where $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ is the shortest separation vector from particle i to particle j, possibly corrected for periodic boundary conditions. The Lennard-Jones factor "LJ" in Eq. (2) may be replaced by two types, namely, the type LJ_6 (describing the $1/|\mathbf{r}_{ij}|^6$ part of the Lennard-Jones interaction) and the type LJ_{12} (describing its $1/|\mathbf{r}_{ij}|^{12}$ part).[6] For two indices i and j, this yields two factors, namely, $(\{i,j\}, \text{LJ}_6)$ and $(\{i,j\}, \text{LJ}_{12})$. Likewise, the bending energy in a water molecule with particles i, j, k will correspond to a factor index $I_M = \{i,j,k\}$ and to a factor type given by the specific function chosen for this interaction. A similar approach was introduced for modeling neighboring beads in a polymer.[16] In Secs. IV B and V, we consider factors that lump together all of the Coulomb interactions between the four particles comprising two distinct dipoles and even between the six particles of two water molecules, respectively. The factor corresponding to the latter case is given by $(\{i,j,k,l,m,n\}, \text{Coulomb})$. (For simplicity of notation, we do not differentiate in this work the Coulomb types for two, four, and six particles.) As mentioned, the set of factors can be infinite,[18] even for finite N. As an example, in a finite periodic system, one can view the three-dimensional Coulomb interaction between particles i and j as a sum of interactions between i and each periodic copy of j indexed by an image index $\mathbf{n} \in \mathbb{Z}^3$. For the case of the above two-water-molecule Coulomb interaction, we would then have $M = (\{i,j,k,l,m,n\}, \text{Coulomb}_{\mathbf{n}})$. The type set $\mathcal{T}$ would then contain all of the separate-image Coulomb interactions,

$$\{\text{Coulomb}_{\mathbf{n}} : \mathbf{n} \in \mathbb{Z}^3\} \subseteq \mathcal{T}, \tag{3}$$

where the set of Coulomb types may be a proper or an improper subset of $\mathcal{T}$. We will treat such factor types in Sec. III.

Given the potential factorization enforced by Eq. (1), the Boltzmann weight $\pi(c) = \exp[-\beta U(c)]$ of configuration $c = \{\mathbf{r}_1, \ldots, \mathbf{r}_N\}$ reduces to a product over factor weights $\pi_M(c_M) = \exp[-\beta U_M(c_M)]$,

$$\pi(c) = \prod_M \pi_M(c_M) = \prod_M \exp[-\beta U_M(c_M)], \tag{4}$$

where c_M is the factor configuration, that is, the configuration c restricted to the indices of factor M. The traditional Metropolis filter,[1] which defines the acceptance probability for a move from configuration c to configuration c' in the Metropolis algorithm, does not factorize in a similar fashion,

$$p^{\text{Met}}(c \to c') = \min[1, \exp(-\beta \Delta U)], \tag{5}$$

$$= \min\left[1, \prod_M \exp(-\beta \Delta U_M)\right], \tag{6}$$

where $\Delta U_M = U_M(c'_M) - U_M(c_M)$ is the factor-potential difference between factor configurations c_M and c'_M. The recent factorized Metropolis filter[6] inverts the order of the product and the minimization and thus casts the acceptance probability of a move into the same factorized form as the Boltzmann weight,

$$p^{\text{Fact}}(c \to c') = \prod_M \min[1, \exp(-\beta \Delta U_M)]. \tag{7}$$

The factorized filter in Eq. (7) and the Boltzmann weight are now written as analogous products. Strictly speaking, M is a generalized index denoting a factor $(\exp(-\beta \Delta U_M)$ or $\min[1, \exp(-\beta \Delta U_M)])$. It is for simplicity that we refer to M as a "factor" rather than a "generalized index for the Boltzmann factor and the filter factor."

The factorized Metropolis filter satisfies the detailed-balance condition,

$$\pi(c) p^{\text{Fact}}(c \to c') = \pi(c') p^{\text{Fact}}(c' \to c). \tag{8}$$

This is evident if there is only a single factor [$U = U_M$ in Eq. (1) so that Eqs. (5) and (7) are identical] because the Metropolis algorithm itself is well known to satisfy it,

$$\underbrace{\pi(c) p^{\text{Met}}(c \to c')}_{\mathcal{F}^{\text{Met}}_{c \to c'}} = \underbrace{\pi(c') p^{\text{Met}}(c' \to c)}_{\mathcal{F}^{\text{Met}}_{c' \to c}}. \tag{9}$$

If there is more than one factor, p^{Fact} also satisfies detailed balance because the Boltzmann weight π of Eq. (4) and the factorized Metropolis filter p^{Fact} of Eq. (7) factorize (that is, break up) in exactly the same way and Eq. (7), on the level of a single factor, is again equivalent to the Metropolis algorithm.

Applying the Metropolis filter p^{Met} of Eq. (5) is equivalent to drawing a Boolean random variable,

$$X^{\text{Met}}(c \to c') = \begin{cases} \text{"True"} & \text{if } \text{ran}(0,1) < p^{\text{Met}}(c \to c'), \\ \text{"False"} & \text{else}, \end{cases} \tag{10}$$

where "True" means that the move from configuration c to configuration c' is accepted. Similarly, the factorized Metropolis filter p^{Fact} could be applied by drawing a single Boolean random variable with p^{Fact} replacing p^{Met} in Eq. (10). However, because $p^{\text{Fact}} \leq p^{\text{Met}}$, this would yield a less efficient algorithm.

We rather view the factorized Metropolis filter as a conjunction of Boolean random variables,

$$X^{\text{Fact}}(c \to c') = \bigwedge_{M \in \mathcal{M}} X_M(c_M \to c'_M). \tag{11}$$

Now, $X^{\text{Fact}}(c \to c')$ is "True" if the independently drawn factorwise Booleans X_M are all "True",

$$X_M = \begin{cases} \text{"True"} & \text{if } \text{ran}_M(0,1) < \mathrm{e}^{-\beta \Delta U_M}, \\ \text{"False"} & \text{else}, \end{cases} \tag{12}$$

where the uniform random variables $\text{ran}_M(0,1)$ are mutually independent for all M.

The conjunction of Eq. (11) formulates the consensus principle: In order to be accepted, the move $c \to c'$ must be independently accepted by all factors M. For example, for a homogeneous N-particle system with pair factors ($\{i,j\}$, pair), the move of a single particle k must be individually accepted by the factors ($\{k,j\}$, pair) $\forall j \neq k$. In other words, the move of particle k must be accepted by all other particles, each through its individual Metropolis filter.

For a continuously varying potential, the acceptance probability of a single factor M has the following infinitesimal limit:

$$\begin{aligned} \min[1, \exp(-\beta \Delta U_M)] &= \exp(-\beta \Delta U_M^+) \\ &\xrightarrow{\Delta U_M \to \mathrm{d}U_M} 1 - \beta \mathrm{d}U_M^+, \end{aligned} \tag{13}$$

where

$$x^+ = \max(0, x) \tag{14}$$

is the unit ramp function of a real number x. In this limit, the factorized Metropolis filter becomes

$$p^{\text{Fact}}(c \to c') = 1 - \beta \sum_M \left[\mathrm{d}U_M(c_M \to c'_M)\right]^+, \tag{15}$$

and the total rejection probability for the move becomes a sum over factors

$$1 - p^{\text{Fact}}(c \to c') = \beta \sum_M \left[\mathrm{d}U_M(c_M \to c'_M)\right]^+. \tag{16}$$

In ECMC, the infinitesimal limit generally corresponds to the continuous-time displacement of a particle k at position $\mathbf{r}_k = (x_k, y_k, z_k)$ and it is usually along a coordinate axis. Supposing that this displacement is in direction $\hat{\mathbf{e}}_x$, the differential of the factor potential becomes

$$\mathrm{d}U_M = \tilde{q}_{M,k}\, \mathrm{d}x_k, \tag{17}$$

where

$$\tilde{q}_{M,k}(\{\mathbf{r}_i : i \in I_M\}) = \frac{\partial U_M}{\partial x_k}, \quad (k \in I_M) \tag{18}$$

is the factor derivative with respect to particle k. We then define the factor event rate with respect to particle k as

$$q_{M,k} = \beta [\tilde{q}_{M,k}]^+, \tag{19}$$

so that each of the terms $\mathrm{d}U_M^+$ becomes

$$\beta \mathrm{d}U_M^+ = q_{M,k}\, \mathrm{d}x_k. \tag{20}$$

The event rate $q_{M,k}$ yields the probability of an event being triggered by particle k within factor M. The total event rate

$$Q_k(\{\mathbf{r}_1, \ldots, \mathbf{r}_N\}) = \sum_{M=(I_M,T_M):k \in I_M} q_{M,k}(\{\mathbf{r}_i : i \in I_M\}) \tag{21}$$

with respect to a particle k naturally involves only event rates for factors that contain k in their index set.

B. Lifting and factorization schemes

The lifting concept[10] is central to ECMC. It lends persistence to the individual Monte Carlo moves and thereby allows one to take the zero-displacement limit. It is in this limit that the sampling of factors becomes unique. We now describe the implementation of a lifted irreversible Markov chain for the simulation of pair-interacting particles,[6] starting with a single pair. We then generalize[16] the method to complex multi-particle potentials.

In a standard Markov-chain Monte Carlo algorithm, the rejection of a move of some particle at time s imposes that the state $c(s+1)$ of the Markov chain at time $s+1$ remains unchanged with respect to the state $c(s)$ at time s. A new move is then proposed. For a local Monte Carlo algorithm in a particle system, this new move normally consists in an independently sampled displacement applied to another randomly chosen particle. In order to converge toward the correct stationary distribution π, we recall that the Markov chain must satisfy the global-balance condition,

$$\mathcal{F}_c = \sum_{c''} \mathcal{F}_{c'' \to c} = \sum_{c''} \pi(c'') p(c'' \to c) = \pi(c), \tag{22}$$

meaning that the total flow $\mathcal{F}_c$ into a configuration c must equal its Boltzmann weight.[49] The detailed-balance condition of Eq. (8) is only a special solution of Eq. (22). In addition to the global-balance condition, the Markov chain must also be irreducible and aperiodic. These two conditions are easily satisfied;[4] the former guarantees that any configuration will eventually be visited, while the latter guarantees that the large-time limit has no hidden periodicities.

In ECMC, any physical configuration c (that is, any set of particle coordinates) is augmented (or "lifted"[11]) to include the so-called lifting variable describing which particle is "active",

$$c \equiv \{\mathbf{r}_1, \ldots, \mathbf{r}_N\} \mapsto (c, a). \tag{23}$$

In principle, the Boltzmann weight now depends on a, but, for simplicity, we require $\pi[(c,a)] = \pi(c)/N$ and absorb the normalization factor $1/N$ into the zero of the potential and omit it in the following.

In ECMC, furthermore, the particle a (the active particle) remains active for subsequent moves as long as they are accepted, and the displacement (in the case that we will treat) is always the same.[50] For simplicity of notation, in the following, the displacement η is applied in the $\hat{\mathbf{e}}_x$ direction for all moves so that the position $\mathbf{r}_a$ is updated to $\mathbf{r}_a + \eta \hat{\mathbf{e}}_x$ for accepted moves. When a displacement $\mathbf{r}_a \to \mathbf{r}_a + \eta \hat{\mathbf{e}}_x$ is rejected by a target particle t, the state of the lifted Markov chain changes in the augmented space as

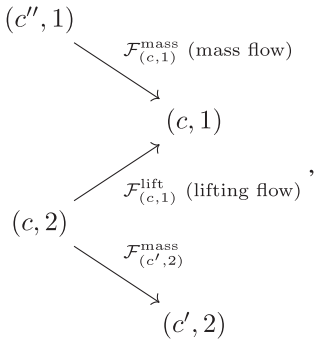
$$(c,a) \to (c,t), \tag{24}$$

but the physical configuration c remains unchanged. Liftings thus replace rejections, and the resulting ECMC algorithm is rejection-free on the augmented space. The global-balance condition must be written in terms of the augmented configurations, and the probability flow $\mathcal{F}_{(c,a)}$ into each lifted configuration (c, a) is then given by the sum of the mass flow $\mathcal{F}^{\text{mass}}_{(c,a)}$, that is, the flow corresponding to a particle displacement and the lifting flow $\mathcal{F}^{\text{lift}}_{(c,a)}$. This sum must equal the statistical weight of (c, a) that, as discussed, equals $\pi(c)$,

$$\mathcal{F}_{(c,a)} = \mathcal{F}^{\text{mass}}_{(c,a)} + \mathcal{F}^{\text{lift}}_{(c,a)} = \pi(c). \tag{25}$$

In order to assure irreducibility of the Markov chain, one may change the direction of motion, most simply by selecting from the set $\{\hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z\}$ in a way that does not need to be random (see the discussion in Sec. V B). In ECMC, the process in between two changes of direction is the eponymous "event chain." The length ℓ of an event chain (the cumulative sum of the displacements) and the distribution of ℓ are essential parameters for the performance of the algorithm.

To demonstrate that ECMC satisfies the global balance condition and to study the conditions on the lifting probabilities, we first consider a system of two particles $\{1, 2\}$. We may suppose, without restriction, that the active particle is 1 so that, at a given time, the lifted configuration is $(c, 1)$. This lifted configuration can only be reached from two other lifted configurations, one that differs in the configuration variable and the other in the lifting variable (see Fig. 1). The lifted configurations and the corresponding flows are



, (26)

where $c = \{\mathbf{r}_1, \mathbf{r}_2\}$, $c'' = \{\mathbf{r}_1 - \eta\hat{\mathbf{e}}_x, \mathbf{r}_2\}$, and $c' = \{\mathbf{r}_1, \mathbf{r}_2 + \eta\hat{\mathbf{e}}_x\}$. The mass flow of the lifted algorithm from $(c'', 1)$ to $(c, 1)$ equals the total Metropolis flow from the non-lifted configuration c'' to c. Because of detailed balance, the latter equals the (non-lifted) Metropolis flow from c to c'' so that

$$\mathcal{F}^{\text{mass}}_{(c,1)} = \underbrace{\mathcal{F}^{\text{Met}}_{c''\to c} = \mathcal{F}^{\text{Met}}_{c\to c''}}_{\text{see Eq. (9)}} = \pi(c)p^{\text{Met}}(c \to c''). \tag{27}$$

The lifting flow in Eq. (26) equals the rejection probability of the Metropolis move $c \to c'$. Because of translational invariance (c'' is a translated version of c'), it agrees with the Metropolis rejection probability of the move back from c to c'',

$$\begin{aligned}\mathcal{F}^{\text{lift}}_{(c,1)} &= \pi(c)\left[1 - p^{\text{Met}}(c \to c')\right],\\ &= \pi(c)\left[1 - p^{\text{Met}}(c \to c'')\right].\end{aligned} \tag{28}$$

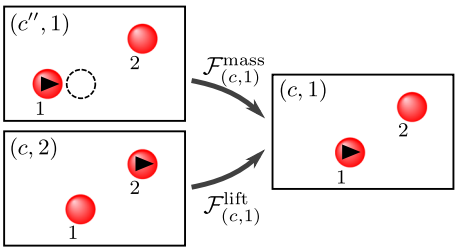


FIG. 1. Mass flow [from $(c'', 1)$] and lifting flow [from $(c, 2)$] into a lifted configuration $(c, 1)$, corresponding to an accepted and a rejected particle move, respectively [see Eq. (26)]. The total flow should equal the Boltzmann weight $\pi(c)$ in order to satisfy the global balance condition of Eq. (22).

$\mathcal{F}^{\text{mass}}_{(c,1)}$ and $\mathcal{F}^{\text{lift}}_{(c,1)}$ thus add up to the Boltzmann weight $\pi(c)$ and global balance is satisfied. The validity of the lifted algorithm (which only satisfies global balance, but breaks detailed balance) hinges on the fact that the underlying Metropolis algorithm satisfies detailed balance and on the translation invariance of the system.

In the infinitesimal limit, for N particles and a particle-pair factorized potential, the total probability flow into a lifted configuration (c, a) has up to N components, namely, $N - 1$ lifting flows from (c, k) to (c, a) for $k \neq a$ and one mass move from (c', a) to (c, a), where c' is again the non-lifted configuration with x_a replaced by $x_a - \mathrm{d}x$. This corresponds to one lifting flow $\mathcal{F}^{\text{lift}}(k \to a)$ equivalent to that in Eq. (26) per target particle $k \neq a$ and a mass flow that is the infinitesimal analog of that in Eq. (26). Furthermore, a particle-pair potential may be further factorized according to multiple factor types T_M; there then exist $N - 1$ lifting flows for each factor M consisting of two particles ($|I_M| = 2$, with I_M the index set of M). Of course, factors that do not contain a in their index set do not contribute to this flow.

Factors M with more than two particles ($|I_M| > 2$) can also be handled within the lifting framework[16] because, by translational invariance, the sum over the factor derivatives with respect to particle k satisfies

$$\sum_{k\in I_M} \partial_{x_k} U_M(\{\mathbf{r}_i : i \in I_M\}) = 0. \tag{29}$$

It is useful to separate the particle indices $k \in I_M$ of a factor M into two sets I^+_M (with positive factor derivatives) and I^-_M (negative factor derivatives) such that

$$k^+ \in I^+_M \Leftrightarrow \quad \partial_{x_{k^+}} U_M > 0, \tag{30}$$

$$k^- \in I^-_M \Leftrightarrow \quad \partial_{x_{k^-}} U_M < 0, \tag{31}$$

where the factor derivatives satisfy

$$\sum_{k^+\in I^+_M} \partial_{x_{k^+}} U_M = -\sum_{k^-\in I^-_M} \partial_{x_{k^-}} U_M \tag{32}$$

[see Fig. 2(a)].

The mass flow into a lifted configuration (c, k^+) with $k^+ \in I^+_M$ by itself satisfies global balance,

$$\begin{aligned}\mathcal{F}^{\text{mass}}_{(c,k^+)} &= \pi_M(c'')p^{\text{Met}}_M(c'' \to c)\\ &= \pi_M(c)p^{\text{Met}}_M(c \to c'') = \pi_M(c),\end{aligned} \tag{33}$$

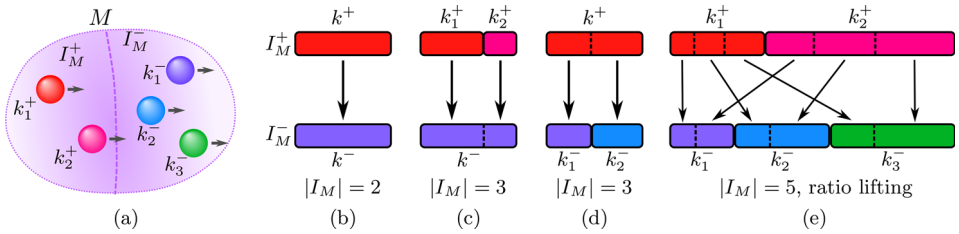


FIG. 2. Factors and lifting schemes. (a) A factor M consisting of $|I_M| = 5$ particles split into non-empty sets I_M^+ (particles that increase the factor potential) and I_M^- [see Eqs. (30) and (31)]. [(b)–(e)] Lifting schemes. Unit branching $\gamma_{k^+\to k^-} = 1$ and $\gamma_{k_1^+\to k^-} = 1, \gamma_{k_2^+\to k^-} = 1$ for a pair-particle factor [(b)] and a three-particle factor with $|I_M^-| = 1$ [(c)], and "ratio" lifting scheme for $|I_M| = 3$, $|I_M^+| = 1$ [(d)] and for $|I_M| > 3$ [(e), see Eq. (40)].

so that there can be no additional lifting moves into (c, k^+). This implies that lifting moves are always as $(c, k^+) \to (c, k^-)$, that is, from an active particle in I_M^+ to a target particle in I_M^-. By contrast, the mass flow into the configuration (c, k^-) is smaller than $\pi_M(c)$,

$$\begin{aligned}\mathcal{F}^{\text{mass}}_{(c,k^-)} &= \pi_M(c'')p_M^{\text{Met}}(c'' \to c)\\ &= \pi_M(c)p_M^{\text{Met}}(c \to c'')\\ &= \pi_M(c)(1 + \underbrace{\beta\partial_{x_{k^-}} U_M}_{<0 \text{ (see Eq. (31))}}\,\mathrm{d}x).\end{aligned} \tag{34}$$

The total lifting flow into (c, k^-) comes from all lifted configurations (c, k^+) with $k^+ \in I_M^+$,

$$\mathcal{F}^{\text{lift}}_{(c,k^-)} = \pi_M(c)\beta \sum_{k^+\in I_M^+} \partial_{x_{k^+}} U_M \,\mathrm{d}x\, \gamma_{k^+\to k^-}, \tag{35}$$

where $\gamma_{k^+\to k^-}$ is the lifting probability from k^+ to k^- once the displacement of k^+ has been rejected. In order for global balance to hold, Eqs. (34) and (35) must add up to $\pi(c)$ for all $k^- \in M^-$. Therefore, and for the algorithm to be rejection-free, one needs[16]

$$\forall k^- \in I_M^- : \underbrace{\partial_{x_{k^-}} U_M}_{<0} + \sum_{k^+\in I_M^+} \underbrace{\partial_{x_{k^+}} U_M}_{>0} \gamma_{k^+\to k^-} = 0, \tag{36}$$

$$\forall k^+ \in I_M^+ : \sum_{k^-\in I_M^-} \gamma_{k^+\to k^-} = 1. \tag{37}$$

Equations (36) and (37) can be visualized as $|I_M^+|$ intervals of length $\partial_{x_{k^+}} U_M$ placed on the upper row of a two-row table and of $|I_M^-|$ intervals of length $|\partial_{x_{k^-}} U_M|$ on the lower row [see Figs. 2(b)–2(e)]. The total lengths of the two rows are equal [see Eq. (32)], and $\gamma_{k^+\to k^-}$ is the fraction of the interval k^+ on the upper row that lifts into k^- on the lower row. Equation (36) describes a conservation of the interval lengths from the upper row to the lower row.

For a pair factor ($|I_M| = 2$), each row has one element and the lifting is unique [$\gamma = \gamma_{k^+\to k^-} = 1$; see Fig. 2(b)]. For a three-particle factor ($|I_M| = 3$), if $|I_M^+| = 2$, again clearly $\gamma_{k^+\to k^-} = 1$ for each one of the particles $k^+ \in I_M^+$ [see Fig. 2(c)]. If $|I_M^+| = 1$ and $|I_M^-| = 2$, then Eq. (36) yields the unique branching probabilities[16] from k^+ to k_1^- and k_2^-,

$$\gamma_{k^+\to k_1^-} = -\frac{\partial_{x_{k_1^-}} U_M}{\partial_{x_{k^+}} U_M} \propto |\partial_{x_{k_1^-}} U_M|, \tag{38}$$

$$\gamma_{k^+\to k_2^-} = -\frac{\partial_{x_{k_2^-}} U_M}{\partial_{x_k^+} U_M} \propto |\partial_{x_{k_2^-}} U_M|, \tag{39}$$

which is readily understood from Fig. 2(d). Analogously, for factors with $|I_M| > 3$, the "ratio" lifting corresponds to cutting up each element in the upper row of the table into pieces of length proportional to the elements in the lower row so that

$$\gamma_{k^+\to k^-} = \frac{\left|\partial_{x_{k^-}} U_M\right|}{\sum_{k^-\in I_M^-}\left|\partial_{x_{k^-}} U_M\right|} \tag{40}$$

[see Fig. 2(e)]. For factors with more than three particles ($|I_M| > 3$), the "ratio" lifting scheme is not unique.[16] We will make use of this freedom, in Secs. IV and V, for factors with up to six particles corresponding to the atoms of two H_2O molecules.

C. Event-driven and cell-veto methods

The implementation of ECMC differs notably from that of the Metropolis algorithm, both because of the continuous-time nature of the Markov chain, which can be simulated without approximations using the event-driven approach,[45] and because of the consensus property, which can be checked in $\mathcal{O}(1)$ operations via the cell-veto method, even for infinite-ranged interactions.[18] (This method can be understood as a "thinning" of the underlying nonhomogeneous Poisson process.[51]) The event-driven formulation of ECMC and the efficient establishment of the consensus are explored in the present section. The intent is to overcome the limitations of time-driven ECMC which considers a finite move $\eta\hat{\mathbf{e}}_x$ of the active particle,

$$\{\mathbf{r}_1, \ldots, \mathbf{r}_a, \ldots, \mathbf{r}_N\} \to \{\mathbf{r}_1, \ldots, \mathbf{r}_a + \eta\hat{\mathbf{e}}_x, \ldots, \mathbf{r}_N\}. \tag{41}$$

This move is either accepted (and then repeated) or it leads to a rejection (by a factor $M \in \mathcal{M}$ containing particle a), and it gives rise to a lifting (or possibly to multiple simultaneous liftings). The complexity of time-driven ECMC is $\mathcal{O}(|\{M : a \in I_M\}|)$ per displacement $\eta\hat{\mathbf{e}}_x$. Time-driven ECMC has a discretization error, as it becomes inconsistent if more than one factor simultaneously rejects the move in Eq. (41). The parameter η must be small enough for multiple rejections to be rare. Time-driven ECMC is thus slow, especially for long-ranged interactions, and inexact. It is useful only for testing.

The finite-move ECMC can be implemented as an event-driven, rather than as a time-driven, algorithm,[45,52] and because all factors are independent, we may consider a single one of them. In the above time-driven ECMC, if the move

in Eq. (41) (the first move, $m = 1$) is accepted, another displacement of magnitude η is attempted. The lth move is

$$\{\mathbf{r}_1, \ldots, \mathbf{r}_a + (l-1)\eta\hat{\mathbf{e}}_x, \ldots, \mathbf{r}_N\} \to \{\mathbf{r}_1, \ldots, \mathbf{r}_a + l\eta\hat{\mathbf{e}}_x, \ldots, \mathbf{r}_N\}. \tag{42}$$

After $m-1$ acceptances, finally, the mth such move is rejected (and leads to a lifting). The parameter m is itself a random variable distributed with a factor-dependent probability

$$p_M(m) = \underbrace{\prod_{l=1}^{m-1} \mathrm{e}^{-\beta\Delta U_M^+(l)}}_{\text{accepted; see Eq. (13)}} \overbrace{\left[1 - \mathrm{e}^{-\beta\Delta U_M^+(m)}\right]}^{\text{move } m \text{ rejected}}, \tag{43}$$

where $\Delta U_M^+(l)$ is the change ΔU_M^+ corresponding to the lth move in Eq. (42). The variable m can be sampled from Eq. (43) and the move $\mathbf{r}_a \to \mathbf{r}_a + (m-1)\eta\hat{\mathbf{e}}_x$ accepted in one step. Although the right-hand side of Eq. (43) gives a probability distribution for the displacement of the active particle a, it only depends on the positive increments of the factor potential. In the continuum limit $\eta \to 0$, the second term on the right-hand side becomes $\beta \mathrm{d}U_M^+(\mathbf{r}_a + \eta_M\hat{\mathbf{e}}_x)$, that is, the factor event rate of Eq. (20), where η_M is the total displacement before a rejection by factor M takes place. In this limit, the exponent in the first term on the right-hand side contains the integral of the factor event rate for the displacement of $\mathbf{r}_a$ between 0 and η_M. This gives the probability density[45]

$$p_M(U_M^+) = \beta \exp(-\beta U_M^+). \tag{44}$$

In Eq. (44), the exponential distribution is sampled by

$$\beta U_M^+ = -\log\{\mathrm{ran}_M(0,1)\}, \tag{45}$$

where

$$\underbrace{\beta U_M^+(\mathbf{r}_a + \eta_M\hat{\mathbf{e}}_x)}_{\text{sampled via Eq. (45)}} = \int_0^{\eta_M} \underbrace{\beta[\partial_{x_a} U_M(\{\mathbf{r}_a + \eta\hat{\mathbf{e}}_x, \mathbf{r}_k : k \in I_M\})]^+}_{\text{factor event rate, see Eq. (20)}} \mathrm{d}\eta. \tag{46}$$

In other words, βU_M^+ is the cumulative event rate of Eq. (20). Equation (46) is an implicit relation for the limiting displacement η_M at which the rejection takes place as a function of the sampled value of βU_M^+. For a two-particle factor $M = (\{a,k\}, \text{pair})$, the integration of the pair event rate in Eq. (46) consists in the replacement of the potential U_M by a related potential which is zero at $\mathbf{r}_a$ and where all the negative increments are replaced by horizontal lines (see Fig. 3).

As mentioned, the factors are independent and each concerned factor M provides a value η_M. The next event takes place at

$$\eta = \min_{M: a \in I_M} \eta_M \tag{47}$$

and the factor which realizes this minimum (that is, η),

$$\mathrm{argmin}_{M: a \in I_M} \eta_M, \tag{48}$$

is the one in which the lifting takes place. For a continuous potential, this factor is uniquely defined and possible simultaneous events, due to finite-precision arithmetic, are too rare to play a role.

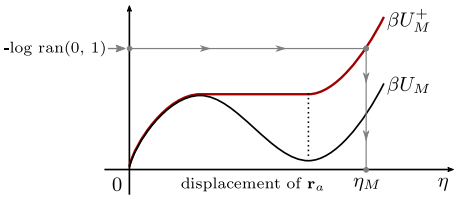


FIG. 3. Event-driven ECMC[45] for a two-particle factor M. The integral of the factor derivative multiplied with β equals βU_M, whereas the integral of the event rate (*in red*) must equal βU_M^+, which is sampled from Eq. (45). The calculation of the displacement η_M from the sampled value of $\beta U_M^+ = -\log\ \mathrm{ran}(0,1)$ is indicated by arrows.

The integration of the factor event rate in Eq. (46) can be tedious if it cannot be cast into an explicit analytical form. This will, for example, be the case for the Coulomb potential in the merged-image framework of Sec. III C. In addition, the inversion of the factor potential [the computation of η_M in Eq. (46)] can be non-trivial. Finally, this calculation must in principle be redone for all the factors that contain the active particle a. For a long-ranged potential, this requires $\mathcal{O}(N) = \mathcal{O}(|\{M \in \mathcal{M} : i \in I_M\}|)$ event-rate integrations and inversions per event. The cell-veto algorithm,[18] by use of a comparison function, avoids the integration and the inversion of the event rate, and it moreover reduces the overall complexity of ECMC to $\mathcal{O}(1)$ per event.

We again first consider a pair factor ($\{1,2\}$, pair), with 1 being the active particle. The lifted position is $(c, 1)$ [with $c = (\mathbf{r}_1, \mathbf{r}_2)$] and the displacement is again in direction $\hat{\mathbf{e}}_x$ (as in the situation in Fig. 1). We embed the two particles in disjoint cells $\mathcal{C}_1$ and $\mathcal{C}_2$ (see Fig. 4). The potentials that we consider here are singular only at $\mathbf{r}_1 = \mathbf{r}_2$ so that the event rate for factor M may be bounded by a constant "cell-event" rate $q_{T_M}^{\mathrm{cell}}(\mathcal{C}_1, \mathcal{C}_2)$,

$$q_{M,1}(\mathbf{r}_1, \mathbf{r}_2) \le q_{T_M}^{\mathrm{cell}}(\mathcal{C}_1, \mathcal{C}_2) \quad \forall \mathbf{r}_1 \in \mathcal{C}_1, \mathbf{r}_2 \in \mathcal{C}_2, \tag{49}$$

where the right-hand side only depends on the factor type. This factor-type dependence may take into account separate cell schemes that could, for example, correspond to Coulomb interactions between isolated charges, dipole–dipole interactions, or to the Lennard-Jones potential. (We recall that we do not differentiate the different Coulomb types for 2, 4, 6 particles to ease notation.) In this work, the condition $\mathcal{C}_1 \neq \mathcal{C}_2$ is adequate to ensure a reasonable value of the cell-event rate. In other cases,[18] one must exclude a local set of cells and treat

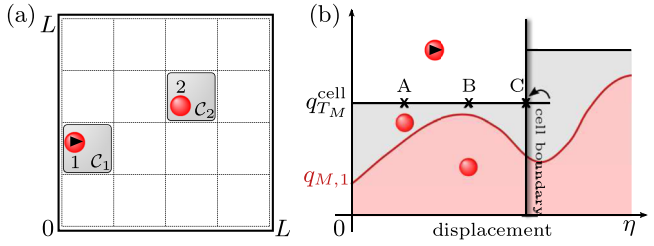


FIG. 4. Cell-veto algorithm for a two-particle factor M. (a) Active particle 1 in cell $\mathcal{C}_1$ and target particle 2 in cell $\mathcal{C}_2$. (b) The event-rate $q_{M,1}(\mathbf{r}_1, \mathbf{r}_2)$ is bounded from above by the cell-event rate $q_{T_M}^{\mathrm{cell}}(\mathcal{C}_1, \mathcal{C}_2)$, which can be sampled trivially. A cell event may either be rejected (at point "A") or confirmed (at point "B") as a particle event [see Eq. (50)], while a cell event taking place outside $\mathcal{C}_1$ (at point "C") means that the active particle 1 will advance toward the cell boundary.

local neighbors outside the cell-veto framework. Cell-event rates are easily tabulated in advance of the ECMC computation proper.

The probability of the event taking place for an infinitesimal displacement $\mathrm{d}x$ equals $q_{M,1}(\mathbf{r}_1, \mathbf{r}_2)\mathrm{d}x$. Since

$$q_{M,1}(\mathbf{r}_1,\mathbf{r}_2)\,\mathrm{d}x = \underbrace{q^{\mathrm{cell}}_{T_M}(\mathcal{C}_1,\mathcal{C}_2)\,\mathrm{d}x}_{\text{infinitesimal}}\ \underbrace{\frac{q_{M,1}(\mathbf{r}_1,\mathbf{r}_2)}{q^{\mathrm{cell}}_{T_M}(\mathcal{C}_1,\mathcal{C}_2)}}_{\leq 1}, \tag{50}$$

the event can initially be sampled as a "cell event" with the constant infinitesimal probability $q^{\mathrm{cell}}_{T_M}(\mathcal{C}_1,\mathcal{C}_2)\mathrm{d}x$, before being confirmed with the finite probability $q_{M,1}(\mathbf{r}_1,\mathbf{r}_2)/q^{\mathrm{cell}}_{T_M}(\mathcal{C}_1,\mathcal{C}_2) \leq 1$. We may suppose that the cell event takes place at a lifted configuration $(c', 1)$ with

$$c' = (\mathbf{r}_1 + \eta\hat{\mathbf{e}}_x, \mathbf{r}_2), \tag{51}$$

$$\pi(\eta) = \exp\left[-\eta q^{\mathrm{cell}}_{T_M}(\mathcal{C}_1,\mathcal{C}_2)\right], \tag{52}$$

where η can be sampled via

$$\eta = -\log[\,\mathrm{ran}(0,1)]/q^{\mathrm{cell}}_{T_M}(\mathcal{C}_1,\mathcal{C}_2). \tag{53}$$

Three outcomes are possible for the sampled values of η and the subsequent confirmation step. First, the cell event may correspond to a configuration c' [in Eq. (51)] that is already outside the active-particle cell ($c' \notin \mathcal{C}_1$). In this case, the move is $(c, 1) \to (c'', 1)$, where c'' is the configuration intersecting the trajectory of particle 1 with the boundary of $\mathcal{C}_1$. Such a cell-boundary event moves the particle, but does not trigger a lifting. Second, the cell event may take place at a configuration $c' \in \mathcal{C}_1$ but fails to be confirmed as an event (because a uniform random number $\mathrm{ran}\left\{0, q^{\mathrm{cell}}_{T_M}(\mathcal{C}_1,\mathcal{C}_2)\right\} > q_{M,1}(\mathbf{r}_1,\mathbf{r}_2)$) [see the second term on the right-hand side of Eq. (50)]. In this case, the move is $(c, 1) \to (c', 1)$ and no lifting takes place. Third, a cell event may take place at a position $c' \in \mathcal{C}_1$ and it is confirmed as an event. This event induces a lifting $(c', 1) \to (c', 2)$ [see Fig. 4(b)]. In this whole process, the factor derivative $\tilde{q}_{M,1}$ is evaluated only when a cell event is triggered from the exponential distribution in Eq. (52). The costly integration of the factor event rate in Eq. (46) is thus avoided.

For an N-particle system, the cell-veto algorithm organizes the search of the next lifting in $\mathcal{O}(1)$ operations. It suffices to choose a regular grid of cells such that, normally, only a single particle belongs to each cell. (Exceptional double-cell occupancies can be handled easily.[18]) In this case, the total event rate with respect to the factor type T_M for an active particle in $\mathcal{C}_a$ is bounded by the total cell event rate,

$$Q^{\mathrm{cell}}_{T_M}(\mathcal{C}_a) = \sum_{\text{cells } \mathcal{C}_t \neq \mathcal{C}_a} q^{\mathrm{cell}}_{T_M}(\mathcal{C}_a,\mathcal{C}_t). \tag{54}$$

In a translationally invariant system, the total cell event rate does not depend on the active cell so that $Q^{\mathrm{cell}}_{T_M}(\mathcal{C}_a) \equiv Q^{\mathrm{cell}}_{T_M}$, a constant that is computed before the ECMC simulation starts from the total number of cells that scales as $\mathcal{O}(N)$. The next cell event is obtained from an exponential distribution with parameter $Q^{\mathrm{cell}}_{T_M}(\mathcal{C}_a)$. This event corresponds to cell $\mathcal{C}_t$ with probability $\propto q^{\mathrm{cell}}_{T_M}(\mathcal{C}_a,\mathcal{C}_t)$, posing a discrete sampling problem that can be solved in $\mathcal{O}(1)$ by Walker's algorithm.[18,53]

The cell-veto algorithm samples the Boltzmann distribution without performing the event-rate integration in Eq. (46). It requires only $\mathcal{O}(1)$ factor-potential evaluations per event in an N-particle system. As a consequence, the total potential of Eq. (1) is not updated and the potential remains unknown as the Markov chain evolves. This is what sets ECMC apart from traditional simulation approaches.

III. ECMC COULOMB ALGORITHMS

In a three-dimensional simulation box with periodic boundary conditions, the Coulomb potential is only conditionally convergent for a charge-neutral system, and it is infinite for a system with a net charge. Finiteness of the potential can be recovered in both cases if each point charge is compensated by a background charge distribution. Traditionally, this is chosen as uniform within the simulation box.[17] The precise association of each background charge with its point charge is not unique. This leads to different electrostatic boundary conditions, which are linked to the polarization state of the simulation box. Consistency imposes a distinct fluctuation theorem[17] for each choice of the boundary condition when computing macroscopic physical properties such as the dielectric constant. Alternatively to the uniform compensating background charge, in ECMC, a line-charge model was introduced.[18] In this model, the background charge distribution is one-dimensional and the factor derivatives are absolutely convergent. The potential for different variants of the line-charge model can be absolutely or conditionally convergent.

As discussed in Sec. II A, ECMC allows for different Coulomb factor sets that may influence the convergence properties of the algorithm, although the steady state is invariably given by the Boltzmann distribution. Roughly, there are two inequivalent Coulomb factorizations.[18] First, the periodic two-particle problem can be embedded on a three-dimensional torus and the potential merged from all the topologically inequivalent minimal paths between particles [see Fig. 5(a)]. For two particles, $\{1,2\}$, this "merged-image" system has a single factor $(\{1,2\}, \mathrm{Coulomb})$. For N particles, this gives the factor set

$$\{(\{i,j\}, \mathrm{Coulomb}) : i < j \in \{1,\ldots,N\}\}. \tag{55}$$

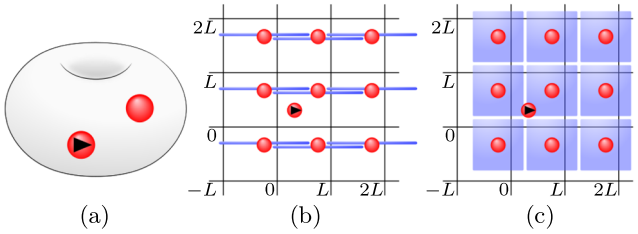


FIG. 5. Periodic two-particle Coulomb system. (a) Toroidal representation corresponding to a merged-image factor. (b) Line-charge representation. The target point-charge particle and each of its copies are compensated by line charges of length $2L$. The active particle inside the central simulation box $[0,L)^3$ is not replicated. (c) Compensating volume-charge representation corresponding to "tin-foil" boundary conditions.

In general, the merged-image factors may comprise more than two particles, but they do not distinguish between the different images of a local configuration (for example, an H_2O molecule). Second, we may picture the three-dimensional periodic system as an infinite number of periodic images of the simulation box indexed by an integer vector $\mathbf{n} \in \mathbb{Z}^3$. For two particles already, this "separate-image" system has an infinite number of factors and for N particles, the factor set is

$$\{(\{i,j\}, \text{Coulomb}_{\mathbf{n}}) : i < j \in \{1,\dots,N\}, \mathbf{n} \in \mathbb{Z}^3\}. \quad (56)$$

More generally, an individual "separate-image" factor may describe an image of certain particles inside the simulation box.

The aim of this section is threefold. First, we present the tin-foil and the line-charge Coulomb formulations and then demonstrate that, although the potentials differ, the Coulomb factor derivatives (that for pair factors yield the pair event rates) are identical. Second, we discuss two efficient algorithms for the merged-image Coulomb derivatives of a pair of particles, one algorithm from the tin-foil perspective and the other summing up line-charge derivatives. Third, we set up an ECMC simulation for two particles in a periodic three-dimensional simulation box in order to validate that the merged-image and the separate-image factor sets indeed show indistinguishable equilibrium properties. We then discuss possible applications for both factorizations.

A. Tin-foil electrostatics within ECMC

The traditional treatment of electrostatic interactions with periodic boundary conditions is based[17] on a large spherical aggregate of images of the three-dimensional cubic simulation box. The polarization state of the simulation box is expressed through electrostatic boundary conditions. With "tin-foil" boundary conditions, the potential of N particles $i \in \{1,\dots,N\}$ of charge c_i (in units where the Coulomb potential between two point charges in free space is $U_{ij} = c_i c_j/|\mathbf{r}_{ij}|$), is[17]

$$U_{\mathrm{C}}(\{\mathbf{r}_1,\dots,\mathbf{r}_N\},\{c_1,\dots,c_N\}) = \frac{1}{2}\sum_{i=1}^{N} c_i \psi(\mathbf{r}_i) + U_{\text{self}}(\alpha), \quad (57)$$

with the electrostatic potential ψ

$$\psi(\mathbf{r}_i) = \sum_{j\neq i=1}^{N} c_j \left[\sum_{\mathbf{n}\in\mathbb{Z}^3} \frac{\operatorname{erfc}(\alpha|\mathbf{r}_{ij}+\mathbf{n}L|)}{|\mathbf{r}_{ij}+\mathbf{n}L|} + \frac{4\pi}{L^3} \sum_{\mathbf{q}\neq(0,0,0)} \frac{\mathrm{e}^{-\mathbf{q}^2/(4\alpha^2)}}{\mathbf{q}^2} \cos\left(\mathbf{q}\cdot\mathbf{r}_{ij}\right) \right], \quad (58)$$

where the Fourier-space sum is over $\mathbf{q} = 2\pi\mathbf{m}/L$ with $\mathbf{m} \in \mathbb{Z}^3$. The self-energy contribution $U_{\text{self}}(\alpha)$ is independent of the particle positions and drops out of our considerations, which are only concerned with derivatives of the potential. The left-hand side of Eq. (57) is independent of the convergence factor $\alpha > 0$, which however influences the speed of evaluation of Eq. (58). Direct evaluation of the sums for N point charges leads to an optimal choice $\alpha \sim N^{1/6}/L$, and a scaling in operations $\mathcal{O}(N^{3/2})$. The particle–mesh Ewald method uses an interpolating mesh to approximate the Fourier sum, leading to $\mathcal{O}(N \log N)$ operations to evaluate the potential. In merged-image ECMC, we only use Eq. (58) for $N = 2$ with $\alpha = \mathcal{O}(1/L)$ and evaluate the derivative of the Coulomb potential to machine precision with $\mathcal{O}(1)$ effort.

We continue, as in Sec. II B, with a two-particle factor $(\{1,2\},\ \text{Coulomb})$. The tin-foil factor derivative is given by

$$\tilde{q}_{(\{1,2\},\ \text{Coulomb}),1}(\mathbf{r}_{12},\{c_1,c_2\}) = \tilde{q}_{\text{Real}}(\mathbf{r}_{12}) + \tilde{q}_{\text{Four.}}(\mathbf{r}_{12}), \quad (59)$$

with the real-space derivative $\tilde{q}_{\text{real}}$,

$$\tilde{q}_{\text{real}}(\mathbf{r}_{12}) = c_1 c_2 \sum_{\mathbf{n}\in\mathbb{Z}^3} \frac{x_{12}+n_x L}{|\mathbf{r}_{12}+\mathbf{n}L|^2} \left[\frac{\operatorname{erfc}(\alpha|\mathbf{r}_{12}+\mathbf{n}L|)}{|\mathbf{r}_{12}+\mathbf{n}L|} + \frac{2\alpha \mathrm{e}^{-\alpha^2|\mathbf{r}_{12}+\mathbf{n}L|^2}}{\pi^{1/2}} \right], \quad (60)$$

and the Fourier-space derivative $\tilde{q}_{\text{Four.}}$,

$$\tilde{q}_{\text{Four.}}(\mathbf{r}_{12}) = c_1 c_2 \frac{4\pi}{L^3} \sum_{\mathbf{q}\neq 0} q_x \frac{\mathrm{e}^{-\mathbf{q}^2/(4\alpha^2)}}{\mathbf{q}^2} \sin(\mathbf{q}\cdot\mathbf{r}_{12}). \quad (61)$$

For two particles and, more generally, for pair factors in an N-particle system, the merged-image Coulomb pair-event rate, from Eq. (59), is given by

$$q_{(\{1,2\},\ \text{Coulomb}),1}(\mathbf{r}_{12},\{c_1,c_2\}) = \beta\left[\tilde{q}_{(\{1,2\},\ \text{Coulomb}),1}(\mathbf{r}_{12},\{c_1,c_2\})\right]^{+}. \quad (62)$$

In Secs. IV and V, we will consider dipole–dipole factors with an index set comprising the four or six particles of two molecules and the "Coulomb" type corresponding to all the Coulomb interactions between the two molecules. The factor potential in this case is the sum over Coulomb pairs within the factor, and the factor derivatives needed in Eq. (36) are the sum of a finite number of pairwise Coulomb derivatives as in Eq. (59). The evaluation of the dipole–dipole factor derivatives remains of complexity $\mathcal{O}(1)$ because the number of elements in each factor remains finite as $N \to \infty$. In ECMC, at most a single factor has to be evaluated precisely for each move (see Sec. II C), whereas in traditional MCMC or MD computations the Coulomb potential in Eq. (57) or its derivatives are computed for all N particles.

B. Line-charge model

In a large periodically reproduced aggregate of the simulation box, the sum over the Coulomb derivatives between a charged active particle and multiple target images (without neutralizing backgrounds) is ill-defined. However, the compensating uniform volume charge is not the only option to regularize the sum, as the line-charge model[18] and its variants provide alternatives to tin-foil electrostatics. Here, straight lines of charges are associated with each copy of the target particle, and aligned with its direction of motion [in our example $\hat{\mathbf{e}}_x$, see Fig. 5(b)]. Although the merged-image line-charge potential, in its simplest version, is itself not absolutely convergent, its factor derivatives are unequivocally defined and equivalent to those obtained with tin-foil boundary conditions. By itself, the line charge neutralizes the charge of the target particle and (because it is centered) also creates an object with zero dipole moment. Previous work[18] used line charges of length

L. Here, we consider lengths pL with integer p [see Fig. 5(b)]. The line charges are replicated over a cubic lattice indexed by the lattice vector $\mathbf{n}$. Lines of different images meet [see Fig. 5(b)]. The Coulomb potential of the line-charge model naturally differs from the one of the tin-foil model because the background charge distributions are manifestly different. However, the merged-image Coulomb derivative of the line-charge model, relevant to ECMC, is identical to the tin-foil expression.

Explicitly, the contribution to the Coulomb derivative from an image $\mathbf{n}$ [with $\mathbf{n} = (0, 0, 0)$ the original simulation box] is

$$\tilde{q}_{(\{1,2\},\,\mathrm{Coulomb}_{\mathbf{n}}),1}(\mathbf{r}_{12}) = c_1c_2\left\{\frac{x_{12}+n_xL}{|\mathbf{r}_{12}+\mathbf{n}L|^3} + \frac{1}{pL}\left[\frac{1}{|\mathbf{r}_{12}+L(\mathbf{n}+p\hat{\mathbf{e}}_x)/2|} - \frac{1}{|\mathbf{r}_{12}+L(\mathbf{n}-p\hat{\mathbf{e}}_x)/2|}\right]\right\}. \tag{63}$$

The line charge generates an electrostatic potential at large separations, $\mathbf{r} = L\mathbf{n}$, which varies with a quadrupolar form. Thus, in any given direction the Coulomb derivative decays as $1/|\mathbf{r}|^4$. For this reason, the sum over the images of the Coulomb derivatives of Eq. (63) converges absolutely. The merged-image Coulomb derivative, in the line-charge formulation, is thus

$$\overbrace{\tilde{q}_{(\{1,2\},\,\mathrm{Coulomb}),1}(\mathbf{r}_{12},\{c_1,c_2\})}^{\text{tin-foil expression, Eq. (59)}} = \underbrace{\sum_{\mathbf{n}}\tilde{q}_{(\{1,2\},\,\mathrm{Coulomb}_{\mathbf{n}}),1}(\mathbf{r}_{12})}_{\text{sum over line charges, Eq. (63)}}. \tag{64}$$

To show this, we first consider the target particle 2 in the simulation box and all its images to be surrounded by a cube of neutralizing charge of volume L^3 centered on the particle 2 and its images. This volume-charge model [see Fig. 5(c)] is closely connected to the line-charge model [see Fig. 5(b)]. The point charge and associated volume charge have vanishing charge, dipole, and quadrupole moments (whereas the line-charge model, in its simplest form, has a finite quadrupole moment). We now compare spherical (radius $R \gg L$) and cubic aggregates (of side $2R$) of target images and study the electrostatic potential within the central simulation box. In this process, the active particle is not replicated, and it remains within the simulation box. Due to the vanishing quadrupole moment of the volume charges, the difference in the electrostatic potential on the particle 1 in the spherical and cubic aggregates decreases at least as fast as $1/R^2$. However the electrostatic potential in the center of the spherical aggregate corresponds to a zero-polarization state which is identical to the tin-foil expression of Eq. (58).

We now find explicit integral expression for the Coulomb derivative of an aggregate of line charges and volume charges and show that the difference is zero in the limit of a large assembly. We again consider the interaction between an active particle and the cubic aggregate of the $(2K+1)^3$ copies of the target particle (the central simulation box and its images). (The active particle is placed inside the simulation box.) The Coulomb potential between the active particle and a single target particle is

$$U_{12} = 4\pi c_1\int_{-\infty}^{\infty}\frac{\mathrm{d}^3\mathbf{q}}{(2\pi)^3}\,e^{i\mathbf{q}\cdot\mathbf{r}_{12}}\frac{\rho_2(\mathbf{q})}{|\mathbf{q}|^2}, \tag{65}$$

where $\rho_2(\mathbf{q})$ is the structure factor of the target particle and the background. We now sum over the images, separated by a multiple of the simulation box size L along each axis. This requires evaluating the sum

$$D_K(q_x) = \sum_{l=-K}^{K}\mathrm{e}^{iq_xlL} = \frac{\sin\left[q_xL(K+1/2)\right]}{\sin(q_xL/2)}, \tag{66}$$

and analogously for q_y and q_z. With the product

$$\tilde{D}_K(\mathbf{q}) = D_K(q_x)D_K(q_y)D_K(q_z), \tag{67}$$

this gives the potential of the active particle in the aggregate of the target particle and its images,

$$U_K = 4\pi c_1\int_{-\infty}^{\infty}\frac{\mathrm{d}^3\mathbf{q}}{(2\pi)^3}\tilde{D}_K(\mathbf{q})\,\mathrm{e}^{i\mathbf{q}\cdot\mathbf{r}_{12}}\,\frac{\rho_2(\mathbf{q})}{|\mathbf{q}|^2}. \tag{68}$$

Equation (66) is the Dirichlet kernel which converges, in a weak sense, to a sum of δ-functions in the limit of large K,

$$D_K(q_x)\xrightarrow[K\to\infty]{}\frac{2\pi}{L}\sum_{m=-\infty}^{\infty}\delta\left(q_x-m\frac{2\pi}{L}\right), \tag{69}$$

and similarly for q_y and q_z. The width of the central peak of D_K scales as $1/K$ for large K. Integrals over sufficiently well-behaved objects become summations in the limit of large K,

$$\int\frac{\mathrm{d}^3\mathbf{q}}{(2\pi)^3}\tilde{D}_K(\mathbf{q})f(\mathbf{q}) \to \frac{1}{L^3}\sum_{\mathbf{q}=2\pi\mathbf{m}/L}f(\mathbf{q}). \tag{70}$$

For the volume-charge model, the structure factor is

$$\rho_2(\mathbf{q}) = c_2\left(1-\operatorname{sinc}\frac{q_xL}{2}\operatorname{sinc}\frac{q_yL}{2}\operatorname{sinc}\frac{q_zL}{2}\right), \tag{71}$$

where the first term on the right-hand side describes the point charge and the product of cardinal sine functions, $\operatorname{sinc}(q_x) = \sin(q_x)/q_x$, etc., the uniform background volume charge.

From Eqs. (68) and (71), the potential of a finite cubic array of images of the particle 2, with active particle 1, is

$$U_K^{\mathrm{volume}} = c_1c_2\int\frac{\mathrm{d}^3\mathbf{q}}{2\pi^2}\tilde{D}_K(\mathbf{q})\frac{\mathrm{e}^{i\mathbf{q}\cdot\mathbf{r}_{12}}}{|\mathbf{q}|^2} \times\left(1-\operatorname{sinc}\frac{q_xL}{2}\operatorname{sinc}\frac{q_yL}{2}\operatorname{sinc}\frac{q_zL}{2}\right). \tag{72}$$

For line charges of length pL, we find

$$U_K^{\mathrm{line}} = c_1c_2\int\frac{\mathrm{d}^3\mathbf{q}}{2\pi^2}\tilde{D}_K(\mathbf{q})\frac{\mathrm{e}^{i\mathbf{q}\cdot\mathbf{r}_{12}}}{|\mathbf{q}|^2}\left(1-\operatorname{sinc}\frac{pq_xL}{2}\right). \tag{73}$$

The volume-charge model is equivalent to the tin-foil Coulomb potential. The line-charge model, whose potential is not absolutely convergent, is nevertheless equivalent for ECMC because, as we will see, the integrals in Eqs. (72) and

(73) yield uniquely defined and equivalent Coulomb derivatives for large K. The difference between the two is given by

$$\Delta U_K = (U_K^{\text{line}} - U_K^{\text{volume}}) = c_1 c_2 \int \frac{\mathrm{d}^3\mathbf{q}}{2\pi^2} \tilde{D}_K(\mathbf{q}) \frac{\mathrm{e}^{i\mathbf{q}\cdot\mathbf{r}_{12}}}{|\mathbf{q}|^2} \times \left(\operatorname{sinc}\frac{q_x L}{2} \operatorname{sinc}\frac{q_y L}{2} \operatorname{sinc}\frac{q_z L}{2} - \operatorname{sinc}\frac{p q_x L}{2} \right).$$

The Dirichlet kernels imply that the integral in this equation is dominated by contributions near $\mathbf{q} = 2\pi\mathbf{m}/L$. However, the function $\operatorname{sinc}(q_i L/2)$ also has zeros at these same points (except when $q_i = 0$, where the sinc function is equal to one). For large K, the potential differences is thus dominated by a sum over q_y, q_z, with $q_x = 0$. This implies that the potential on the active particle equals (to within a constant) the tin-foil potential for motion parallel to the line-charges, but the difference of potentials is corrugated in the perpendicular y–z plane. This is a consequence of the fusion of multiple aligned line charges into a single uniform line when p is integer [see Fig. 5(b)].

We examine the derivative of ΔU_K to show that the Coulomb derivatives converge to the same value,

$$\partial_{x_1} \Delta U_K = \int \frac{\mathrm{d}^3\mathbf{q}}{2\pi^2} \tilde{D}_K(\mathbf{q}) \frac{q_x \sin(\mathbf{q}\cdot\mathbf{r}_{12})}{|\mathbf{q}|^2} \times \left(\operatorname{sinc}\frac{q_x L}{2} \operatorname{sinc}\frac{q_y L}{2} \operatorname{sinc}\frac{q_z L}{2} - \operatorname{sinc}\frac{p q_x L}{2} \right), \quad (74)$$

which suppresses the contributions which remained for the calculation of the potential, due to the factor $q_x \sin(q_x x)$ near $q_x = 0$.

Finally, we consider explicitly the possible divergence at $|\mathbf{q}| = 0$ in Eq. (74), due to the presence of the term $1/|\mathbf{q}|^2$. We expand all the trigonometric functions in the integrand, ΔI_K, to find

$$\Delta I_K \xrightarrow[\mathbf{q}\to 0]{} \text{const} \times \frac{q_x^2\left[(p^2-1)q_x^2 - q_y^2 - q_z^2\right]}{|\mathbf{q}|^2} \tilde{D}_K(\mathbf{q}).$$

Even this contribution is thus driven to zero for large K. We conclude that in a periodic three-dimensional system, the line-charge model becomes equivalent to the volume-charge model, and therefore to tin-foil electrostatics. The line charges must lie parallel to the direction of motion but can of course be switched at will. In contrast, the volume-charge model gives the tin-foil Coulomb derivatives in all directions.

C. Algorithms for Coulomb derivatives

The merged-image Coulomb derivatives are best computed from the tin-foil expressions of Eq. (62). To accelerate the evaluation, we reduce the Fourier-space component of Eq. (61) to a sum over non-negative components (m_x, m_y, m_z),

$$\tilde{q}_f(\mathbf{r}_{12}) = A_{xyz} \sin(\lambda_{12}^x) \cos(\lambda_{12}^y) \cos(\lambda_{12}^z), \quad (75)$$

where $\lambda_{12}^x = 2\pi m_x x_{12}/L$, and similarly in y and z and where

$$A_{xyz} = \frac{16 c_1 c_2 m_x}{L^2 |\mathbf{m}|^2 2^{\delta_{m_y,0}+\delta_{m_z,0}}} \exp\left(-\frac{\pi^2|\mathbf{m}|^2}{\alpha^2 L^2}\right) \quad (76)$$

is a position-independent tensor that can be computed before the simulation starts. In Eq. (75), repeated indices (x, y, z) are summed over non-negative integers (m_x, m_y, m_z).

The merged-image Coulomb derivatives can also be computed from the sum of the line-charge derivatives [see the right-hand side of Eq. (64)]. Because of the symmetry of the line charges, the quadrupolar contribution to the derivative is an odd function of x so that forward and backward terms cancel, and that the sum converges as $1/K^2$ for large K. The convergence may be accelerated using Richardson extrapolation[54] (see Fig. 6). Denoting the finite line-charge sum over the range $\mathbf{n} \in [-K, K]^3$ as S_K and assuming that

$$S_K = S_\infty + \frac{A}{K^p}, \quad (77)$$

one may eliminate A as

$$S'_{K+1} = \frac{(K+1)^p S_{K+1} - K^p S_K}{(K+1)^p - K^p}. \quad (78)$$

The sequence $(S'_{K+1} - S_\infty)$ then decays as $1/K^{p+1}$. The transformation of Eq. (78) can be iterated, each time gaining one power in the asymptotic behavior of the sequence. The merged-image line-charge derivatives converge to the tin-foil expression of Eq. (59), confirming that the two algorithms compute the same object and that individual factors in the line-charge model may be used to simulate tin-foil potentials.

As in the line-charge model, one may sum up the associated point charges and their compensating volume charges explicitly, rather than proceeding through Fourier transformation. However, the analytic formulas are difficult to work with. A further possibility consists in compensating each point charge with more than one line charge. Remarkably, four line charges arranged on a square of side $L/\sqrt{12}$ in the y–z plane, cancel dipole and quadrupole moments in the multipole expansion and lead to an absolutely converging sum for the electrostatic potential. One may also construct more elaborate sheets

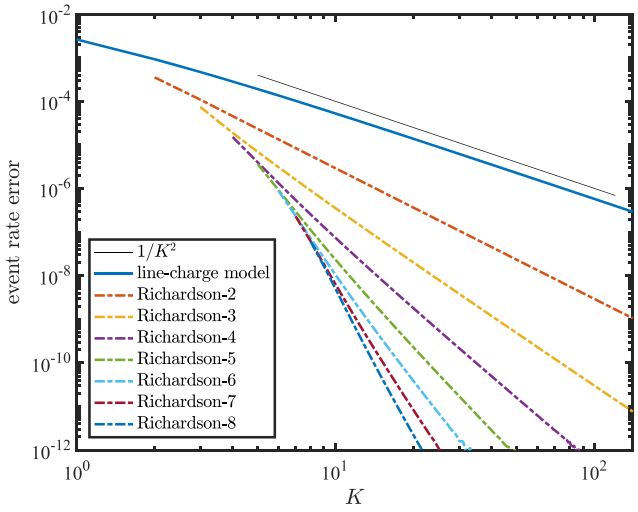


FIG. 6. Comparison of the tin-foil expression for the Coulomb factor derivative and the sum over line charges for a given value of $\mathbf{r}_{12}$ [see Eq. (64)] as a function of the cutoff K. The 8-fold iterated Richardson extrapolation for the line-charge expression agrees with the tin-foil expression to within 10^{-12} for $K \approx 20$.

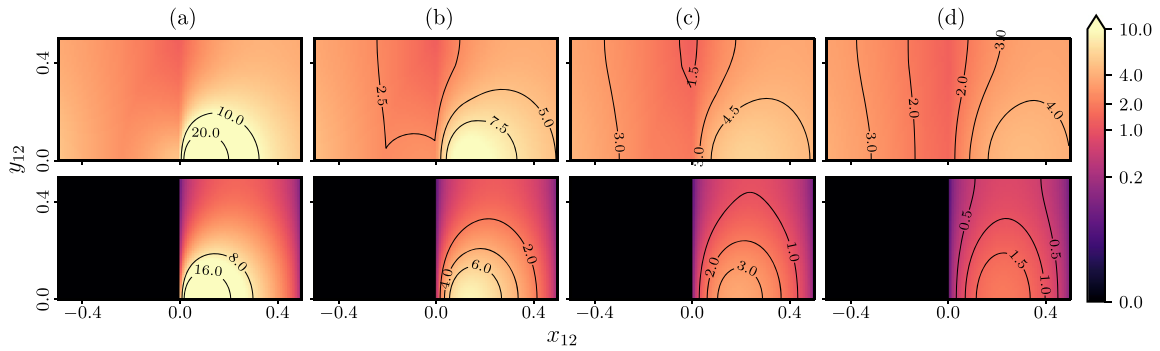


FIG. 7. Comparison of the sum of the separate-image event rates $\sum_{\mathbf{n}} \beta[\tilde{q}_{(\{1,2\},\,\mathrm{Coulomb}_{\mathbf{n}}),1}(\mathbf{r}_{12})]^+$ (upper row) and the merged-image event rate $\beta[\tilde{q}_{(\{1,2\},\,\mathrm{Coulomb}),1}(\mathbf{r}_{12})]^+$ (lower row). In all panels $\mathbf{r}_{12} = (x_{12}, y_{12}, z_{12})$ with (a) $z_{12} = 0.1$, (b) $z_{12} = 0.2$, (c) $z_{12} = 0.3$, and (d) $z_{12} = 0.4$. $L = 1$ and $\beta c_1 c_2 = 1$ throughout.

and volumes of screening charges to cancel higher orders in the multipole expansion. All of these screening objects presented here regularize the sum of the pair derivatives over images and allow for separate-image factor sets [analogous to Eq. (56); see Sec. III D]. Although the sequence S_K decays faster, the Coulomb event rate is not reduced by these different objects.

D. Separate-image ECMC

As we have seen, all the Coulomb interactions in a finite system with periodic boundary conditions can be image-merged into a single Coulomb type that sums over all the inequivalent minimal paths between two points on a torus, and that correspond to images in the rolled-out representation of periodic boundary conditions. For two particles 1 and 2, this is expressed through a single factor $M = (\{1,2\},\ \mathrm{Coulomb})$. The corresponding factor derivatives can then be computed with the traditional tin-foil expression [Eq. (59)] or within the line-charge framework [Eq. (64)]. The choice of one over the other is a matter of efficiency only (the algorithmic complexity being the same). Each of the formulations suggests other choices for the interaction types. In the line-charge formulation, the choice of an infinite set of types $\{\mathrm{Coulomb}_{\mathbf{n}} : \mathbf{n} \in \mathbb{Z}^3\}$ suggests itself. For two particles 1 and 2, the set of separate-image factors is $\{(\{1,2\}, \mathrm{Coulomb}_{\mathbf{n}}) : \mathbf{n} \in \mathbb{Z}^3\}$. Within ECMC, these images are statistically independent but only one of them must be computed precisely for each event. This is because, as in Sec. II, we can use a variant of the cell-veto algorithm (supplemented with an asymptotic bounding function[18]), in order to sample the relevant image index $\mathbf{n}$ and to then compute the corresponding factor derivative of $M_{\mathbf{n}}$.

Separate-image Coulomb factors generally come with larger pair event rates, as the contributions from different images do not compensate (see Fig. 7). On the other hand, evaluating a separate-image Coulomb derivative [as in Eq. (63)] to machine precision requires just a few operations, many fewer than what is required for its merged-image counterpart. Details of the separate-image Coulomb factors can influence the efficiency of the algorithm. As an example, the terminal point of the line charge is a singular point of Eq. (63) and should not approach another point charge in the system. This motivates our choice of length $2L$ (or multiples thereof), as the terminal point of one line charge then coincides with the position of an image of the original particle. For the Coulomb potential, the nonphysical line-charge singularity, confounded with the singularity of the point charge, no longer disrupts the ECMC dynamics.

The dynamic behavior of the different factor sets for the Coulomb problem has not yet been explored in detail. As a first step, for a system of two like Coulomb charges, merged-image and separate-image ECMC was validated against the regular tin-foil Metropolis algorithm (see Fig. 8). All three methods clearly sample the Boltzmann distribution in the asymptotic steady state.

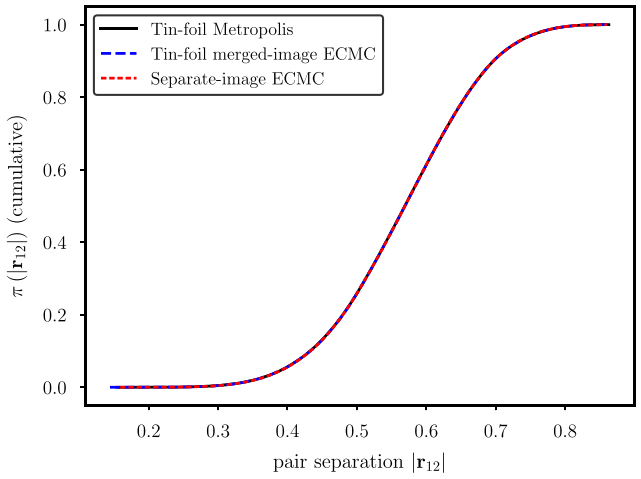


FIG. 8. Cumulative histogram of the pair separation $|\mathbf{r}_{12}|$ for two particles of equal charge in a periodic three-dimensional simulation box ($\beta c_1 c_2 = 2$, $L = 1$).

IV. DIPOLE–DIPOLE FACTORS

In ECMC, one may tailor the factor sets to the problems at hand. In electrostatic systems made up of local dipoles, specific "dipole–dipole" Coulomb factors may thus contain all the atoms distributed over two molecules that can be far apart from each other. These factors yield much smaller

event rates than "particle–particle" pair factors. In addition, a special "inside-first" lifting scheme can direct most of the lifting flow from the active particle to a target particle situated on the same molecule. Even for a non-local factor made up of two distant dipoles, the lifting flow will thus mostly be between an active particle and a target particle on the same molecule (the probability of an intramolecular lifting grows like $\log N$, whereas all the intermolecular liftings remain constant). We expect such a local lifting scheme for extended factors to show interesting dynamic properties. In the present section, we explore dipole–dipole factors in a simple model of charge-neutral two-particle molecules before employing them, in Sec. V, to a model of liquid water. We expect dipole–dipole factors and their variants to have useful applications in ECMC.

Concretely, for a simple model of two-particle dipoles in a three-dimensional periodic simulation box, the dipole–dipole factor for the particles $\{1,2,3,4\}$ is given by

$$(\{1,2,3,4\},\ \text{Coulomb}), \tag{79}$$

[see Fig. 9(b)], where the corresponding Coulomb factor potential is

$$U_{(\{1,2,3,4\},\ \text{Coulomb})}(\mathbf{r}_1,\dots,\mathbf{r}_4) = \sum_{i=1}^{2}\sum_{j=3}^{4} U_{\text{C}}(\mathbf{r}_{ij},\{c_i,c_j\}). \tag{80}$$

The factor of Eq. (79) thus comprises the four Coulomb potentials between these particles, using the Coulomb potential of Eq. (57). The model excludes, as is usual,[30] Coulomb interactions within a dipole. For the same four particles, one may also use the "particle–particle" factors

$$\{(\{1,3\},\ \text{Coulomb}), (\{1,4\},\ \text{Coulomb}) \\ (\{2,3\},\ \text{Coulomb}), (\{2,4\},\ \text{Coulomb})\}, \tag{81}$$

with the "particle–particle" factor potential

$$U_{(\{i,j\},\text{Coulomb})}(\mathbf{r}_{ij},\{c_i,c_j\}) = U_{\text{C}}(\mathbf{r}_{ij},\{c_i,c_j\}) \tag{82}$$

[see Fig. 9(a)]. We suppose that the particle 1 is active. The dipole–dipole event rate

$$\beta[\tilde{q}_{(\{1,2,3,4\},\ \text{Coulomb}),1}]^+ \tag{83}$$

then allows the interactions $U_{\text{C}}(\mathbf{r}_{13})$ and $U_{\text{C}}(\mathbf{r}_{14})$ to compensate each other (and to give the event rate corresponding to a point charge interacting with a dipole), while the particle–particle event rate

$$\beta[\tilde{q}_{(\{1,3\},\ \text{Coulomb}),1}]^+ + \beta[\tilde{q}_{(\{1,4\},\ \text{Coulomb}),1}]^+ \tag{84}$$

remains much larger (corresponding to a point charge separately interacting with two isolated point charges) because the unit-ramp functions are both non-negative [see Eq. (14)] and one of them is usually zero.

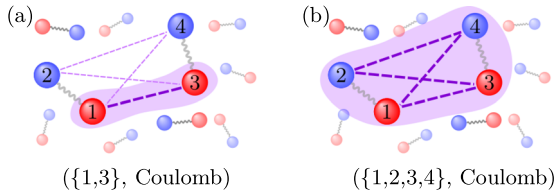


FIG. 9. Model of two-particle dipoles. (a) Particle–particle factor associating two point charges that belong to different dipoles. (b) "dipole–dipole" factor comprising four Coulomb interactions.

A. Event-rate scaling for Coulomb factors

We now consider a homogeneous system of dipoles of size $|\mathbf{d}| \sim d$ small compared to the simulation box (see Fig. 10). For concreteness, we suppose that particle 1 is the active particle. The event rate, whose scaling with system size we compute in the present section, is the result of the interaction between the particle 1 and the distant dimer (in Fig. 9 made up of particles 3 and 4). As there is no Coulomb interaction between particles on the same dipole, the position of particle 2 (the dipole partner of particle 1) does not come into play for the event rate. We will see in Sec. IV B that this is no longer true for the lifting rates, which are influenced both by the distant dimer and by the local dimer of particle 1, that is, by the position of particle 2.

The electrostatic potential at a distance $\mathbf{r}$ from a point charge c_k, within the merged-image (tin-foil) formulation in a box of side L, is given by the scaling form

$$\psi_L(\mathbf{r}) = \frac{c_k}{|\mathbf{r}|} f_E(\mathbf{r}/L), \tag{85}$$

which generalizes Coulomb's law valid in free space. The function $f_E(\mathbf{x})$ is smooth and remains $\mathcal{O}(1)$ for all $\mathbf{x} \in [-1/2, 1/2]^3$. For separations such that $|\mathbf{r}|/L \ll 1$, the potential given by Eq. (85) has the expansion[55]

$$\psi(\mathbf{r}) = c_k\left(\frac{1}{|\mathbf{r}|} + \frac{\text{const}}{L} + \frac{2\pi|\mathbf{r}|^2}{3L^3} + \cdots\right). \tag{86}$$

The nth-order derivatives of $f_E(\mathbf{r}/L)$ are also smooth and have an amplitude which scale as L^{-n}. The Coulomb derivative between an active particle and a particle k, separated by a vector $\mathbf{r} \in [-L/2, L/2]^3$, also has the scaling form

$$\beta\tilde{q}_{(\{1,k\},\text{Coulomb}),1} = \frac{l_{\text{B}}}{|\mathbf{r}|^2} f_E^1(\mathbf{r}/L). \tag{87}$$

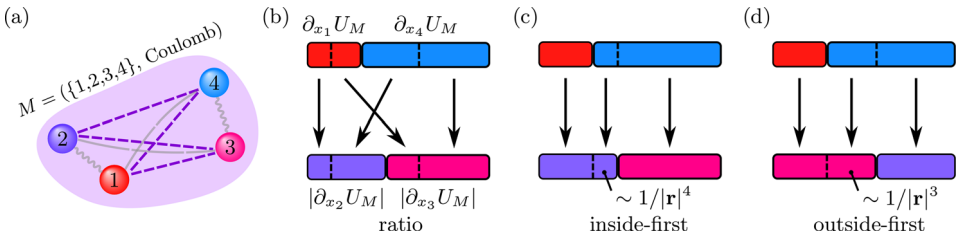


FIG. 10. Lifting schemes for a dipole–dipole factor. (a) Dipole–dipole factor with four Coulomb interactions. It is assumed that $\partial_{x_1}U_M > 0$, $\partial_{x_4}U_M > 0$ and $\partial_{x_2}U_M < 0$, $\partial_{x_3}U_M < 0$. (b) "ratio" lifting, (c) "inside-first" lifting (d) "outside-first" lifting.

Here, we have introduced the characteristic Bjerrum length $l_B = |e^2|\beta$, with e being the elementary charge, the distance at which the Coulomb interaction equals the thermal energy and used f_E^1 as a new scaling function, which again remains $\mathcal{O}(1)$. An explicit form for Eq. (87) at small separations can be found from Eq. (86). For a constant number density ρ of particles within the simulation cell, the mean total Coulomb event rate per particle, $\langle Q_{\text{p–p}} \rangle$, is given by the integral

$$\langle Q_{\text{p–p}} \rangle = \sum_{k\neq 1} \left\langle q_{(\{i,k\},\,\text{Coulomb}),1} \right\rangle \tag{88}$$

$$= \int_{[-L/2,L/2]^3} \frac{l_B \rho}{|\mathbf{r}|^2} f_E^1(\mathbf{r}/L)\, \mathrm{d}^3\mathbf{r}$$

$$= l_B \rho L \int_{[-1/2,1/2]^3} \frac{1}{\mathbf{x}^2} f_E^1(\mathbf{x})\, \mathrm{d}^3\mathbf{x} \sim l_B \rho L. \tag{89}$$

This mean total event rate thus diverges as $\mathcal{O}(L)$. The reciprocal of $\langle Q_{\text{p–p}} \rangle$ sets the scale for the mean-free path due to charge–charge interactions, and it is of length scale $\mathcal{O}(1/L)$. The result agrees with the naive free-space argument[18] based on the bare $1/|\mathbf{r}|$ Coulomb interaction. At constant density, the divergence of Eq. (89) in $L \sim N^{1/3}$ implies that the active and target particles are often widely separated from each other. With pair factors, one thus expects a complexity of $\mathcal{O}(N^{4/3})$ for an $\mathcal{O}(1)$ displacement of all particles in the system.

The scaling form of the potential can also be used to determine the event rate for dipole–dipole factors [as in Fig. 9(b)], the interaction of point charges with dipoles, or the interaction of pairs of well-separated dipoles. The potential at a distance $\mathbf{r}$ from a dipole in the periodic box is found from Eq. (85) by applying the operator $(-\mathbf{d}\cdot\nabla)$, with $\mathbf{d}$ the dipole moment. Using again $|\mathbf{d}| \sim d$ implies that the event rate of the dipole–dipole factor, resulting from the interaction of the active particle 1 with the dipole at a distance $\mathbf{r}$ corresponds to a particle–dipole Coulomb interaction. The dipole–dipole event rate, for two dipoles separated by a vector $\mathbf{r} \in [-L/2, L/2]^3$ is given by

$$\beta\tilde{q}_{(\{1,2,3,4\},\,\text{Coulomb}),1} \sim \frac{d l_B}{|\mathbf{r}|^3} f_E^1(\mathbf{r}/L), \tag{90}$$

where $\mathbf{r}$ denotes the vector from the active particle to the dipole. Equation (90) implies that ECMC with dipole–dipole factors has a much lower mean total Coulomb event rate $\langle Q_{\text{Coulomb}} \rangle$,

$$\langle Q_{\text{Coulomb}} \rangle = \int_{[-L/2,L/2]^3} \frac{l_B \rho d}{|\mathbf{r}|^3} f_E^2(\mathbf{r}/L)\, \mathrm{d}^3\mathbf{r}$$

$$= l_B \rho d \int_{[-1/2,1/2]^3} \frac{1}{|\mathbf{x}|^3} f_E^2(\mathbf{x})\, \mathrm{d}^3\mathbf{x}, \tag{91}$$

where f_E^2 is another scaling function. The second integral in Eq. (91) is weakly divergent near the origin (which simply means that in ECMC very nearby dipoles have to be treated individually). Excluding a region of radius $\mathcal{O}(d/L)$, the mean total Coulomb event rate using dipole–dipole factors is

$$\langle Q_{\text{Coulomb}} \rangle \sim l_B \rho d \log(L/d). \tag{92}$$

This much reduced total event rate, obtained by limiting the contributions from large distances, is our main motivation for using dipole–dipole factors.

The scaling obtained in Eqs. (90) and (92) is independent of the specific definition of the dipole model. It only relies on the use of dipole–dipole factors connecting two charge-neutral molecules that may be far apart (see Sec. V, where the dipoles are realized by H_2O molecules). The scaling is also insensitive to the introduction of screening charge distributions, and it holds both for the merged-image and for the separate-image factor sets. Adapting this factorization framework to systems composed of molecules that behave as approximate higher-order multipoles would further improve the scaling.

B. Dipole–dipole lifting schemes

We now consider lifting schemes for dipole–dipole factors, and for concreteness, we consider a four-particle system of particles $\{1, 2\}$, forming a charge-neutral dipole $\mathbf{d}_{12}$ and particles $\{3, 4\}$, forming an analogous dipole $\mathbf{d}_{34}$. In this two-dipole system, particle 1, for example, not only interacts with a charge-neutral dipole $\mathbf{d}_{34}$, but is itself inside such a dipole $\mathbf{d}_{12}$. Although the Coulomb lifting rate is oblivious to the position of 2 (as there is no Coulomb interaction between particles 1 and 2), particle 2 is part of the dipole–dipole factor, and its position influences the relative lifting rates.

We obtain the derivatives with respect to particles 1 and 2 for the factor $M = (\{1, 2, 3, 4\}, \text{Coulomb})$ as follows:

$$\beta\tilde{q}_{M,1} = l_B \left[a \frac{|\mathbf{d}_{34}|}{|\mathbf{r}|^3} + \mathcal{O}\left(\frac{|\mathbf{d}_{34}|^2}{|\mathbf{r}|^4} \right) + \mathcal{O}\left(\frac{|\mathbf{d}_{34}|}{L^3} \right) \right], \tag{93}$$

$$\beta\tilde{q}_{M,2} = l_B \left[-a \frac{|\mathbf{d}_{34}|}{|\mathbf{r}|^3} + \mathcal{O}\left(\frac{|\mathbf{d}_{34}|^2}{|\mathbf{r}|^4} \right) + \mathcal{O}\left(\frac{|\mathbf{d}_{34}|}{L^3} \right) \right]. \tag{94}$$

The dominant terms in these two equations are equal in magnitude yet opposite in sign, reflecting that particles 1 and 2 interact with the same distant dipole $\mathbf{d}_{34}$, are of opposite sign, and close to each other (on the dipole $\mathbf{d}_{12}$). For the factor derivatives with respect to particles 3 and 4, we find

$$\beta\tilde{q}_{M,3} = l_B \left[\tilde{a} \frac{|\mathbf{d}_{12}|}{|\mathbf{r}|^3} + \mathcal{O}\left(\frac{|\mathbf{d}_{12}|^2}{|\mathbf{r}|^4} \right) + \mathcal{O}\left(\frac{|\mathbf{d}_{12}|}{L^3} \right) \right], \tag{95}$$

$$\beta\tilde{q}_{M,4} = l_B \left[-\tilde{a} \frac{|\mathbf{d}_{12}|}{|\mathbf{r}|^3} + \mathcal{O}\left(\frac{|\mathbf{d}_{12}|^2}{|\mathbf{r}|^4} \right) + \mathcal{O}\left(\frac{|\mathbf{d}_{12}|}{L^3} \right) \right]. \tag{96}$$

[For ease of notation, we used here Eq. (86) for small $|\mathbf{r}|/L$ rather than the full scaling form.]

The coefficient a (and analogously for $\tilde{a}$) reflects the orientation of $\mathbf{d}_{34}$ with respect to the distance vector between the two dipoles (see Fig. 10). Remarkably, the factor derivatives of M with respect to the particles within each dipole ($\tilde{q}_{M,1} + \tilde{q}_{M,2}$ and $\tilde{q}_{M,3} + \tilde{q}_{M,4}$) cancel at order $1/|\mathbf{r}|^3$ and leave a remainder of $1/|\mathbf{r}|^4$. This dipole–dipole compensation to order $1/|\mathbf{r}|^3$ of the factor derivatives is a general feature for pairs of local dipoles (that can be composed of more than two atoms) inside a factor and occurs in the same manner with the full scaling functions in the merged-image potential.

We recall from Eq. (29) that the four factor derivatives exactly sum up to zero. As illustrated in Sec. II B (see Fig. 2), the lifting scheme corresponds to arranging the indices $k^+ \in I_M^+$ on the upper row of a two-row table and the indices $k^- \in I_M^-$ on the lower row. In a factor with large separation $|\mathbf{r}|$, each row contains one element corresponding to each of the two dipoles (see Fig. 10).

The "ratio" lifting scheme is as described in Sec. II B. All elements fall off as $\mathcal{O}(1/|\mathbf{r}|^3)$ [see Eq. (90)], and both rows contain elements representing each dipole. From Eqs. (94) and (96), this leads to comparable proportions of intra- and inter-molecular liftings. Both rates fall off at the same rate, but their coefficients are different reflecting the orientations of the dipoles. The total inter- and intra-dipole lifting rates both scale as $\log L$ [see Fig. 10(b) and Table I].

In the "inside-first" lifting scheme, the elements corresponding to each dipole are aligned with each other. The two match to order $\sim 1/|\mathbf{r}|^3$. The mismatch in bar length is $\mathcal{O}(1/|\mathbf{r}|^4)$ in Eqs. (93) and (94). In the full scaling picture, the difference in length of the elements can be computed analogously. Coulomb liftings thus occur mostly within a dipole, and long-ranged inter-dipole liftings remain bounded in number for large system sizes [see Fig. 10(c) and Table I].

Finally, the "outside-first" lifting scheme consists in vertically aligning elements corresponding to different dipoles. Aligned elements are of length $\sim|a|$ and $\sim|\tilde{a}|$ so that intra- and inter-dipole lifting rates again both fall off as $\mathcal{O}(1/|\mathbf{r}|^3)$. The situation is analogous to the one for the "ratio" lifting, and the "outside-first" scheme remains strongly non-local [see Fig. 10(d) and Table I].

In contrast to the above dipole–dipole factors, the "particle–particle" factor, as argued in Eqs. (87) and (89), produces events which occur at the scale of the simulation box at a rate which decreases as only $1/|\mathbf{r}|^2$, leading to a total event rate increasing linearly with L. The lifting flow is between one dipole and the other, and the intra-dipole lifting rate is zero (see Table I).

C. Validation of factors and liftings

The dipole–dipole factors and their different lifting schemes can be checked for consistency for two charge-neutral dipoles with a short-ranged vibrational intra-dipole potential, a repulsive potential between oppositely charged particles (needed to keep dipoles apart from each other), and intermolecular Coulomb interactions. With particles numbered as in Fig. 9, the model corresponds to a factor set

$$\{(\{1,2\},\text{bond}),(\{3,4\},\text{bond}),(\{1,4\},\text{rep}),(\{2,3\},\text{rep}),(\{1,2,3,4\},\text{Coulomb})\}, \tag{97}$$

with the harmonic bond factor potential,

$$U_{(\{i,j\},\text{bond})}(\mathbf{r}_{ij}) = \frac{1}{2}k_b\left(|\mathbf{r}_{ij}| - r_0\right)^2, \tag{98}$$

with $k_b > 0$, a short-range repulsive potential

$$U_{(\{i,j\},\text{rep})}(\mathbf{r}_{ij}) = \frac{1}{2}k_2\left(\frac{r_0}{|\mathbf{r}_{ij}|}\right)^6, \tag{99}$$

with $k_2 > 0$, and a scalar separation r_0, in addition to the dipole–dipole Coulomb factor potential of Eq. (80).

The dipole–dipole Coulomb factor differs from the particle–particle Coulomb factors in the set,

$$\{(\{1,2\},\text{bond}),(\{3,4\},\text{bond}),(\{1,4\},\text{rep}),(\{2,3\},\text{rep}),(\{1,3\},\text{Coulomb}),(\{1,4\},\text{Coulomb}),(\{2,3\},\text{Coulomb}),(\{2,4\},\text{Coulomb})\}, \tag{100}$$

where the factor potentials corresponding to bond vibrations and the repulsion between unlike charges are as in Eqs. (98) and (99) and the Coulomb factor potentials are those of Eq. (82). In addition, since $|I_M| = 2$ for each particle-factorized factor M, we have no freedom in choosing a lifting scheme (see Sec. IV B).

The "ratio," "inside-first" and "outside-first" lifting schemes for the dipole–dipole factor are easily implemented and compared to the particle–particle lifting scheme. By construction, they yield identical thermodynamic correlations (see Fig. 11). Although the event rates are fixed by the decomposition of the total potential into factors, the different lifting schemes may differ in their dynamical behavior.

TABLE I. Coulomb lifting rates for two dipoles separated by a distance $|\mathbf{r}|/L \ll 1$, together with full integrated rate in simulation box of size L^3: One particle–particle and three dipole–dipole schemes ("ratio," "outside-first," and "inside-first"). q_{intra}: lifting rate to the non-active particle within the active dipole. q_{inter}: lifting rate to the triggering dipole. $\langle Q_{\text{intra}}\rangle$ and $\langle Q_{\text{inter}}\rangle$ denote the mean total event rates (using the full scaling form, as in Sec. IV A), integrated over the simulation box.

Lifting scheme	q_{intra}	q_{inter}	$\langle Q_{\text{intra}}\rangle$	$\langle Q_{\text{inter}}\rangle$	Lifting
Particle	0	$1/\lvert\mathbf{r}\rvert^2$	0	L	inter-dipole
Ratio	$1/\lvert\mathbf{r}\rvert^3$	$1/\lvert\mathbf{r}\rvert^3$	$\log L$	$\log L$	inter + intra
Outside-first	$1/\lvert\mathbf{r}\rvert^3$	$1/\lvert\mathbf{r}\rvert^3$	$\log L$	$\log L$	inter + intra
Inside-first	$1/\lvert\mathbf{r}\rvert^3$	$1/\lvert\mathbf{r}\rvert^4$	$\log L$	const	inter-dipole

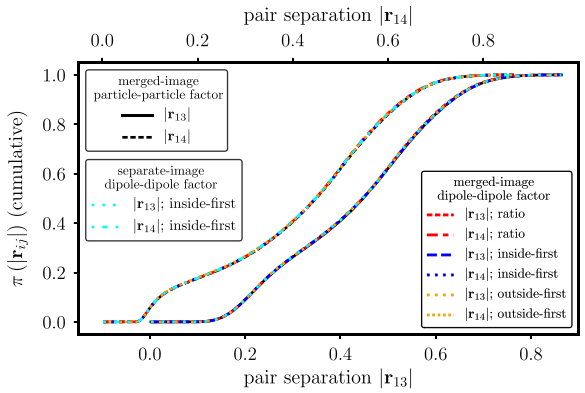


FIG. 11. Cumulative histograms of the distances $|\mathbf{r}_{13}|$ [like charges, see Fig. 10(a)] and $|\mathbf{r}_{14}|$ [opposite charges, see Fig. 10(a)] for the particle–particle factor set of Eq. (100) and also for the factor set of Eq. (97) using dipole–dipole Coulomb factors, using the three lifting schemes of Figs. 10(b)–10(d). Also separate-image dipole–dipole factors with inside-first lifting. Periodic cubic simulation box with $L = 1$, $c_i = \pm 1$ point charges, $\beta = 1$, $k_b = 400$, $k_2 = 1$, and $r_0 = 0.1$.

V. LIQUID WATER AND DIPOLE–DIPOLE FACTORS

To explore ECMC in a realistic context, we implement in this section the simple point-charge water model with flexible molecules (SPC/Fw), a well-studied all-atom model for liquid water.[30] This model combines the long-ranged Coulomb potential with hydrogen–oxygen bond-length vibrations, a flexible hydrogen–oxygen–hydrogen angle, and a specific oxygen–oxygen interaction of the Lennard-Jones type. The SPC/Fw model is closely related to one used in molecular-dynamics simulations of solvated peptides.[43]

Naturally, each water molecule is charge-neutral and dipolar so that the dipole–dipole factorization of Sec. IV applies. This realizes a mean-free path for a single particle as $\sim 1/\log N$ in the thermodynamic limit (an earlier ECMC Coulomb algorithm[18] had obtained a mean-free path scaling as $\sim 1/N^{1/3}$).

A. Factors in the SPC/Fw water model

To simulate liquid water with the SPC/Fw potential, we use the following type set:

$$\{\text{bond},\ \text{bending},\ \text{LJ},\ \text{Coulomb}\}. \tag{101}$$

As an example, the factor set for two water molecules, containing particles $\{1,2,3\}$ and $\{4,5,6\}$, respectively, [and with 2 and 5 being the oxygens, see Fig. 12(a)] is

$$\begin{aligned}\{&(\{1,2\},\ \text{bond}),(\{2,3\},\ \text{bond}),\\ &(\{4,5\},\ \text{bond}),(\{5,6\},\ \text{bond}),(\{2,5\},\ \text{LJ}),\\ &(\{1,2,3\},\ \text{bending}),(\{4,5,6\},\ \text{bending}),\\ &(\{1,\ldots,6\},\ \text{Coulomb})\}.\end{aligned} \tag{102}$$

This factor set [see Fig. 12(b)] trivially generalizes to more than two H_2O molecules.

In Eq. (102), the "bond" factor potential of Eq. (98) describes oxygen–hydrogen bond vibrations with the equilibrium bond distance $r_0 = 1.012$ Å and $k_b = 1059.162$ kcal mol^{-1} rad^{-2}, that correspond to the SPC/Fw parameters. The "bending" factor potential describes the fluctuations in the bond angle within each H_2O molecule,

$$U_{(\{i,j,k\},\,\text{bending})}(\mathbf{r}_i,\mathbf{r}_j,\mathbf{r}_k) = \frac{1}{2}k_a\left(\phi_{\{i,j,k\}} - \phi_0\right)^2,$$

where $\phi_{\{1,2,3\}}$ and $\phi_{\{4,5,6\}}$ denote the internal angle between the two legs of each H_2O molecule (see Fig. 12). We adopt the SPC/Fw parameters: $\phi_0 = 113.24°$ and $k_a = 75.90$ kcal mol^{-1} Å^{-2}. The specific Lennard-Jones interaction between oxygen atoms corresponds to the "LJ" factor potential

$$U_{(\{2,5\},\,\text{LJ})}(\mathbf{r}_{25}) = k_{\text{LJ}}\left[\left(\frac{\sigma}{|\mathbf{r}_{25}|}\right)^{12} - \left(\frac{\sigma}{|\mathbf{r}_{25}|}\right)^{6}\right], \tag{103}$$

where $k_{\text{LJ}} = 0.62$ kcal mol^{-1} and $\sigma = 3.165$ Å are prescribed in the SPC/Fw model. The Lennard-Jones potential is truncated beyond 9.0 Å. This truncation, however, is unnecessary if the cell-veto algorithm is used. Finally, the dipole–dipole "Coulomb" factor potential, in direct generalization of Eq. (80), is given by

$$U_{(\{1,\ldots,6\},\,\text{Coulomb})}(\mathbf{r}_1,\ldots,\mathbf{r}_6) = \sum_{i=1}^{3}\sum_{j=4}^{6} U_{\text{C}}(\mathbf{r}_{ij},\{c_i,c_j\}). \tag{104}$$

Here, the Coulomb potential of Eq. (57) is used with the SPC/Fw parameters $c_1 = c_3 = c_4 = c_6 = 0.41e$ and $c_2 = c_5 = -0.82e$ (with e the elementary charge).

The type set of Eq. (101) is by no means unique. We could also break up the Lennard-Jones interaction into two types, corresponding to the two components of the Lennard-Jones potential (as discussed in Sec. II A). Also, instead of the merged-image Coulomb type, we could adopt any of the variants of the separate-image type, resulting in a type set,

$$\{\text{bond},\ \text{bending},\ \text{LJ},\ \text{Coulomb}_{\mathbf{n}} : \mathbf{n} \in \mathbb{Z}^3\}.$$

Finally, it is possible to break up the "bond" and "bending" factors into $N_{\text{H}_2\text{O}} - 1$ equal terms in order to construct a unique dipole–dipole factor for each pair of H_2O molecules in such a way that the type set only contains a single element. All these choices are correct, but they may differ in the ease of implementation and in the speed with which they approach equilibrium.

B. Intrinsic rotations

Our version of ECMC is formulated in terms of displacements that, for a given event chain, are along one of the directions $\{\hat{\mathbf{e}}_x,\hat{\mathbf{e}}_y,\hat{\mathbf{e}}_z\}$. Each individual event chain can strain the system, but is unable to rotate it, as the coordinates perpendicular to the direction of motion remain unchanged. The flexible SPC/Fw H_2O molecule may itself get strained in a single event chain. Applying strain subsequently in different directions is known to be equivalent to a rotation on all levels, and, in particular, on the level of a single molecule. This guarantees that the algorithm is irreducible and can attain all of configuration space.

The rotation that is induced through subsequent event chains in the three directions can be illustrated in an

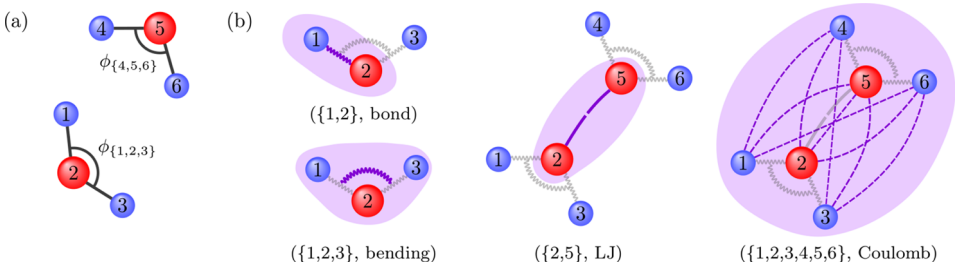


FIG. 12. SPC/Fw water model and ECMC factors. (a) Two H_2O molecules, with particles $\{1,2,3\}$ and $\{4,5,6\}$, respectively (2 and 5 being the oxygens). Each of the molecules has a finite dipole moment. (b) "bond," "bending," "LJ," and "Coulomb" factors implementing the SPC/Fw model. Factors contain between two and six particles.

ECMC simulation of a single H_2O molecule, using only the intramolecular factor types in Eq. (101). The rotational dynamics of such a single molecule is easily tracked through the equilibrium autocorrelation function of the dipole moment $\mathbf{d} = \mathbf{r}_{21} + \mathbf{r}_{23}$ (see Fig. 12), given by

$$A(s) = \langle \mathbf{d}(s') \cdot \mathbf{d}(s'+s) \rangle \sim \exp(-s/\lambda) \quad \text{for } s \to \infty,$$

where the variables s and $s + s'$ denote the ECMC displacement (proportional to the time of the continuous Markov process). $A(s)$ decays exponentially at large s with a rate that gives the autocorrelation length λ of molecular orientation.

At temperature 300 K, the cumulative chain length it takes to rotate the molecule around itself is about one to two orders of magnitude larger than the H_2O molecule itself (see Fig. 13). In the limit of large chain lengths ℓ, the autocorrelation length of the dipole moment is proportional to ℓ. This simply means that lengthening an already long chain does not add to the internal strain of the water molecule, as a local equilibrium is reached.

The sequence of chain directions need not be random: The switching of directions merely renders the Markov chain irreducible, whereas global balance is satisfied for any infinitesimal move (without the return move necessary for detailed balance). As a deterministic, cyclic, sequence $\hat{\mathbf{e}}_x\hat{\mathbf{e}}_y\hat{\mathbf{e}}_z\hat{\mathbf{e}}_x\hat{\mathbf{e}}_y \ldots$ avoids repetitions, we find it to decorrelate the dipole moment faster than a uniform random sampling of chain directions (see Fig. 13). The rotations of molecules are thus generated as a byproduct of the switching of event-chain directions. In practical applications, it remains to be seen whether the rotations of molecular ensembles decay particularly slowly. In this case only, the ECMC algorithm will need to be modified in order to explicitly take into account rotations.

C. ECMC for liquid water

The SPC/Fw potential is adapted for liquid water at standard temperature 300 K and density 1 g/cm^3. An ECMC simulation at these conditions is easily set up with factors

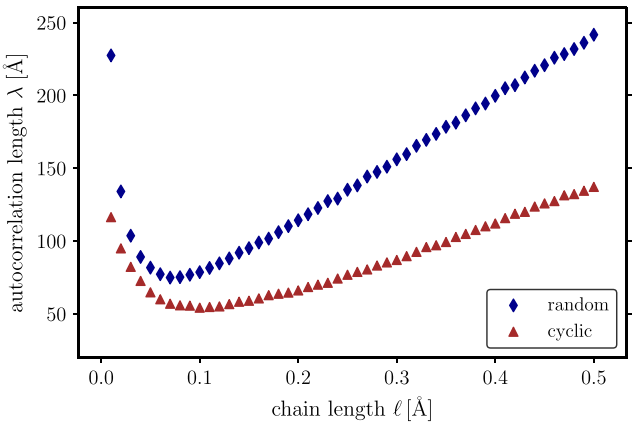


FIG. 13. Autocorrelation length λ for the dipole moment in ECMC of a single H_2O molecule (fixed chain length ℓ) for the cyclic sequence of event-chain directions ($\hat{\mathbf{e}}_x\hat{\mathbf{e}}_y\hat{\mathbf{e}}_z\hat{\mathbf{e}}_x \ldots$) and for their random resampling.

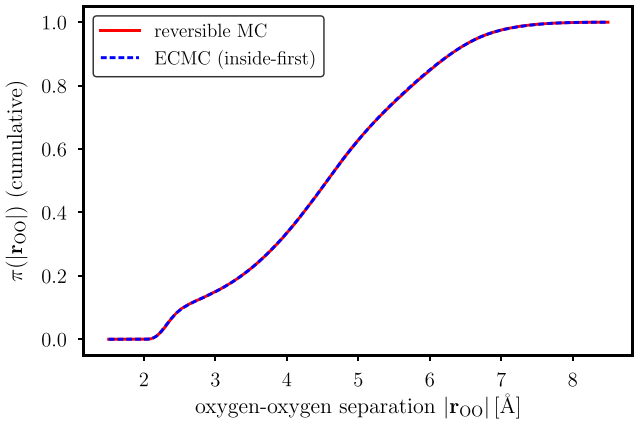


FIG. 14. Cumulative histogram of the oxygen–oxygen separation $|\mathbf{r}_{OO}|$ for 32 H_2O molecules at standard density and temperature via conventional reversible Monte Carlo and ECMC using the factor set of Eq. (102) with the inside-first lifting scheme. The random choice of directions was used with a fixed value of $\ell = 0.5$ Å. The maximal difference between the two distributions is smaller than 10^{-3}.

(including the dipole–dipole Coulomb factor) as in Eq. (102) generalized for $N_{H_2O} > 2$. The "ratio," "outside-first," and "inside-first" lifting schemes are taken over from the dipole case discussed in Sec. IV. However, the dipole is now constructed from three particles. For a far distant pair of H_2O molecules, the factor derivatives with respect to the hydrogen positions are usually of the same sign and of opposite sign to that of the oxygen. In the notations of Fig. 12 and using $M = (\{1, \ldots, 6\}, \text{Coulomb})$, we thus have that to order $1/r^3$,

$$\partial_{x_1} U_M \sim \partial_{x_3} U_M \sim -\frac{1}{2}\partial_{x_2} U_M. \tag{105}$$

This can again be used in the inside-first lifting scheme to keep most of the lifting flow inside the molecule of the active particle. Care must be exercised in these lifting schemes to arrange the particles in a fixed order that is independent of which particle is active (it is incorrect to place the active particle systematically on the left-most position on the upper row of the table in Fig. 10).

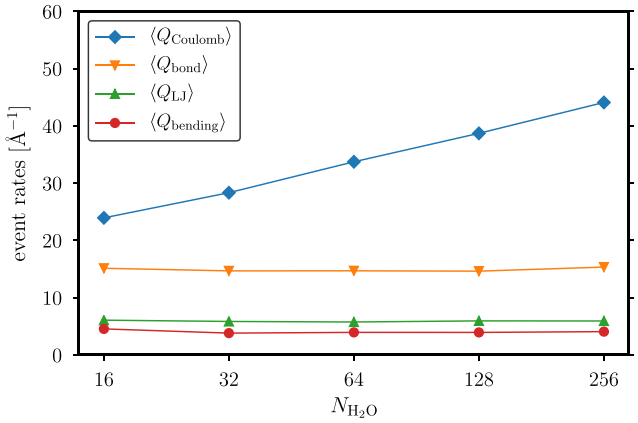


FIG. 15. Ensemble-averaged total "Coulomb," "bond," "LJ," and "bending" event rates as a function of the number of H_2O molecules. The Coulomb event rate scales logarithmically. Event rates depend on the choice of factors but are independent of the lifting scheme.

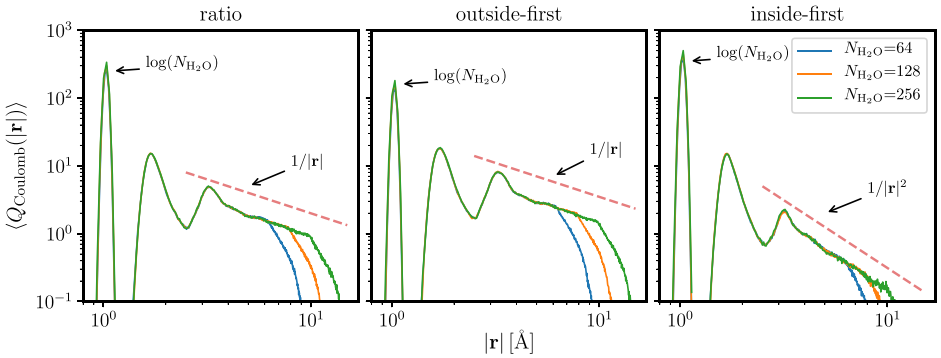


FIG. 16. Histogram of distance $|\mathbf{r}|$ between the active and the target particle for the Coulomb events for the "ratio," "outside-first," and "inside-first" lifting schemes and for 64, 128, and 256 H_2O molecules. The integral of each histogram corresponds to $\langle Q_{\text{Coulomb}}\rangle$ in Fig. 15. Dashed lines indicate $|\mathbf{r}|^{-z}$ with exponents z: 1, 1, 2, which corresponds to $q_{\text{inter}} \sim |\mathbf{r}|^{-(z+2)}$ in Table I.

For long simulation times, the ECMC algorithm exactly samples the Boltzmann distribution of this model, and thermodynamic observables can be compared with Metropolis Monte Carlo using the Ewald summation for the Coulomb potential. This can be verified for the oxygen–oxygen distances that agree to very high precision, demonstrating that the irreversible ECMC converges toward the same steady state as reversible Monte Carlo algorithms (see Fig. 14). To make sure that equilibrium is reached, the initial configurations were chosen randomly in a very dilute system and slowly compressed toward the target density.

In the liquid-water simulation for $N_{H_2O} > 2$, the factors $M = (I_M, T_M)$ belong to four different types (that is, $T_M \in \mathcal{T}$ and $|\mathcal{T}| = 4$), into which the ensemble-averaged total event rate [see Eq. (21)] can be split,

$$\frac{1}{N}\sum_{k=1}^{N}\langle Q_k(\{\mathbf{r}_1,\ldots,\mathbf{r}_N\})\rangle = \sum_{T\in\mathcal{T}}\langle Q_T\rangle. \tag{106}$$

$\langle Q_{\text{Coulomb}}\rangle$ agrees with the definition in Sec. III [see Eq. (91)]. The three local factor types naturally give constant scaling of their associated mean event rates $\langle Q_{\text{bond}}\rangle$, $\langle Q_{\text{LJ}}\rangle$, and $\langle Q_{\text{bending}}\rangle$ with system size, whereas $\langle Q_{\text{Coulomb}}\rangle$ clearly features $\log N_{H_2O}$ scaling with the number of H_2O molecules (see Fig. 15). The logarithmic scaling of the total Coulomb event rate validates the prediction of Eq. (92). The total event rate increases by 5 $Å^{-1}$ when N_{H_2O} doubles.

Finally we study the lifting flows for the dipole–dipole factors under the "ratio," "inside-first," and "outside-first" schemes (see Sec. IV B). As discussed in Sec. IV A, the event rates are independent of the lifting schemes for a given factor set. However, the probability distributions of the distance $|\mathbf{r}|$ between the active and the target particles are different (see Fig. 16). First, the peak at the oxygen–hydrogen bond length corresponding to a lifting within the molecule increases logarithmically with system size. Second, with increasing system size the distribution of event distances develops a power-law tail. In both the "ratio" and the "outside-first" lifting schemes, the tail of the probability distribution decreases as $|\mathbf{r}|^{-1}$. The "inside-first" scheme decays as $|\mathbf{r}|^{-2}$. These results, corresponding to the evolution of q_{inter} in $|\mathbf{r}|^{-3}$ and $|\mathbf{r}|^{-4}$, are summarized in Table I.

Remarkably, the "inside-first" lifting scheme induces mostly local lifting flows, even for Coulomb factors that associate H_2O molecules that are far distant from one another. Most of the liftings are local and the central peak increases as $\log N_{H_2O}$. We expect a local lifting to keep the dynamics of the system coherent and to lead to faster convergence toward equilibrium. It appears also possible to replace the interaction with far-away H_2O molecules by the interaction with an effective medium (given that the lifting flow remains local). In the "ratio" and "outside-first" lifting schemes, this would probably not be possible as the lifting flow toward far-away dipoles is of the same order of magnitude as the local flow.

VI. CONCLUSIONS

In this work, we have outlined the ECMC framework for all-atom computations. Our algorithm advances a single particle in the presence of long-ranged electrostatic interactions in $\mathcal{O}(1)$ operations, with a mean-free path which decreases as $\mathcal{O}(1/\log N)$. This gives an overall complexity of $\mathcal{O}(N\log N)$ to advance N particles, each by $\mathcal{O}(1)$. This speed can be achieved for locally charge-neutral systems, where particles can be grouped into local dipoles. The algorithm can take into account the presence of free point charges, and its performance worsens only gradually with their number. The algorithm is manifestly translation-invariant and event-driven. It is free of discretization errors and exactly samples the Boltzmann distribution, without needing a thermostat. Its outstanding property is that it neither computes total forces nor determines the system potential.

ECMC breaks with tradition in two ways. First, as a Markov-chain algorithm, it offers the freedom to choose among a variety of moves. Our approach of advancing single particles may be a first step only. Nevertheless, as we have shown, it effectively rotates dipoles and flexible water molecules in three-dimensional space and samples the entire configuration space. We have explored the great freedom to choose factors and liftings that suit the problem at hand. Second, ECMC breaks with tradition in that it is purely particle–particle: It treats electrostatic interactions between point charges, but is oblivious to the electrostatic field. This aspect liberates it from the interpolating mesh that in traditional particle–particle–particle–mesh methods approximates the Coulomb field. Rather, the algorithm is based on the interaction of pairs of particles and, more generally, of factors that may comprise pairs of local dipoles or even more complex objects.

In this work, we have checked that thermodynamic quantities from ECMC agree with those obtained with methods

that satisfy detailed balance. As a next step for analyzing ECMC in all-atom systems, it will be important to study its relaxation dynamics in detail. This dynamics will certainly depend on the choice of factors and, for example, for the case of dipole–dipole factors treated here, on the choice of liftings. The inside-first lifting scheme yields mostly local dynamics, and we would expect it to lead to a faster decay of correlation functions. Besides this, we have discussed that the length and the probability distribution of the event-chain parameter ℓ and even the sequence of the directions of the event-chain can significantly influence the ECMC dynamics although, as we have verified extensively, the steady state is always given by the Boltzmann distribution. We would hope that, in addition to the overall favorable algorithmic scaling, the fast decay of density fluctuations carries over from short-range-interacting particle and spin systems. The influence of different factorization and lifting schemes on the dynamics of ECMC will also have to be understood. From an algorithmic implementation point of view, we think that the parallelization of the method[56] will have to be dealt with carefully. ECMC simulations of water will permit standardized comparison of run times with the ones of traditional molecular-dynamics algorithms. Other applications, such as solvated peptides[43] and polarizable models, appear within reach.

ACKNOWLEDGMENTS

M.F.F. acknowledges financial support from EPSRC Fellowship No. EP/P033830/1 and hospitality at Ecole normale supérieure. This work was initiated at the Aspen Center for Physics, which is supported by the National Science Foundation Grant No. PHY-1066293. We thank Matthew Downton for illuminating discussions and Matthias Staudacher for helpful comments. W.K. acknowledges support from the Alexander von Humboldt Foundation and thanks the Santa Fe Institute for hospitality.

[1] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, J. Chem. Phys. **21**, 1087 (1953).
[2] B. J. Alder and T. E. Wainwright, J. Chem. Phys. **27**, 1208 (1957).
[3] W. Krauth, *Statistical Mechanics: Algorithms and Computations* (Oxford University Press, 2006).
[4] D. A. Levin, Y. Peres, and E. L. Wilmer, *Markov Chains and Mixing Times* (American Mathematical Society, 2008).
[5] E. P. Bernard, W. Krauth, and D. B. Wilson, Phys. Rev. E **80**, 056704 (2009).
[6] M. Michel, S. C. Kapfer, and W. Krauth, J. Chem. Phys. **140**, 054116 (2014).
[7] Y. Nishikawa, M. Michel, W. Krauth, and K. Hukushima, Phys. Rev. E **92**, 063306 (2015).
[8] M. Michel, J. Mayer, and W. Krauth, Europhys. Lett. **112**, 20003 (2015).
[9] Z. Lei and W. Krauth, Europhys. Lett. **121**, 10008 (2018).
[10] P. Diaconis, S. Holmes, and R. M. Neal, Ann. Appl. Probab. **10**, 726 (2000).
[11] F. Chen, L. Lovász, and I. Pak, in *Proceedings of the 17th Annual ACM Symposium on Theory of Computing* (ACM, 1999), Vol. 275.
[12] E. P. Bernard and W. Krauth, Phys. Rev. Lett. **107**, 155704 (2011).
[13] M. Engel, J. A. Anderson, S. C. Glotzer, M. Isobe, E. P. Bernard, and W. Krauth, Phys. Rev. E **87**, 042134 (2013).
[14] S. C. Kapfer and W. Krauth, Phys. Rev. Lett. **119**, 240603 (2017).
[15] Z. Lei and W. Krauth, e-print arXiv:1806.06786 (2018).
[16] J. Harland, M. Michel, T. A. Kampmann, and J. Kierfeld, Europhys. Lett. **117**, 30001 (2017).
[17] S. W. de Leeuw, J. W. Perram, and E. R. Smith, Proc. R. Soc. A **373**, 27 (1980).
[18] S. C. Kapfer and W. Krauth, Phys. Rev. E **94**, 031302 (2016).
[19] H. Berthoumieux and A. C. Maggs, Europhys. Lett. **91**, 56006 (2010).
[20] B. J. Alder and T. E. Wainwright, Phys. Rev. A **1**, 18 (1970).
[21] J. P. Wittmer, P. Polińska, H. Meyer, J. Farago, A. Johner, J. Baschnagel, and A. Cavallo, J. Chem. Phys. **134**, 234901 (2011).
[22] A. R. Leach, *Molecular Modelling: Principles and Applications*, 2nd ed. (Pearson Prentice Hall, 2009).
[23] T. Schlick, *Molecular Modeling and Simulation: An Interdisciplinary Guide* (Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002).
[24] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman, J. Am. Chem. Soc. **117**, 5179 (1995).
[25] B. R. Brooks, C. L. Brooks, A. D. Mackerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, and M. Karplus, J. Comput. Chem. **30**, 1545 (2009).
[26] D. Van der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. C. Berendsen, J. Comput. Chem. **26**, 1701 (2005).
[27] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, J. Comput. Chem. **26**, 1781 (2005).
[28] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao, M. P. Eastwood, J. Gagliardo, J. P. Grossman, C. R. Ho, D. J. Ierardi, I. Kolossváry, J. L. Klepeis, T. Layman, C. McLeavey, M. A. Moraes, R. Mueller, E. C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, and S. C. Wang, in *Proceedings of the 34th Annual International Symposium on Computer Architecture, ISCA '07* (ACM, New York, NY, USA, 2007), pp. 1–12.
[29] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, J. Chem. Phys. **79**, 926 (1983).
[30] Y. Wu, H. L. Tepper, and G. A. Voth, J. Chem. Phys. **124**, 024503 (2006).
[31] R. W. Hockney and J. W. Eastwood, *Computer Simulation using Particles* (Taylor & Francis, Inc., Bristol, PA, USA, 1988).
[32] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen, J. Chem. Phys. **103**, 8577 (1995).
[33] Y. Shan, J. L. Klepeis, M. P. Eastwood, R. O. Dror, and D. E. Shaw, J. Chem. Phys. **122**, 054101 (2005).
[34] H. Limbach, A. Arnold, B. Mann, and C. Holm, Comput. Phys. Commun. **174**, 704 (2006).
[35] C. Sagui and T. Darden, J. Chem. Phys. **114**, 6578 (2001).
[36] L. Greengard and V. Rokhlin, J. Comput. Phys. **73**, 325 (1987).
[37] A. C. Maggs and V. Rossetto, Phys. Rev. Lett. **88**, 196402 (2002).
[38] A. C. Maggs, J. Chem. Phys. **117**, 1975 (2002).
[39] A. C. Maggs, J. Chem. Phys. **120**, 3108 (2004).
[40] J. Rottler and A. C. Maggs, J. Chem. Phys. **120**, 3119 (2004).
[41] J. Rottler and A. C. Maggs, Phys. Rev. Lett. **93**, 170201 (2004).
[42] J. A. Morrone, R. Zhou, and B. J. Berne, J. Chem. Theory Comput. **6**, 1798 (2010).
[43] D. E. Shaw, P. Maragakis, K. Lindorff-Larsen, S. Piana, R. O. Dror, M. P. Eastwood, J. A. Bank, J. M. Jumper, J. K. Salmon, Y. Shan, and W. Wriggers, Science **330**, 341 (2010).
[44] D. Frenkel and B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*, Computational Science Series (Elsevier Science, 2001).
[45] E. A. J. F. Peters and G. de With, Phys. Rev. E **85**, 026703 (2012).
[46] E. P. Bernard and W. Krauth, Phys. Rev. E **86**, 017701 (2012).
[47] B. J. Alder and T. E. Wainwright, J. Chem. Phys. **31**, 459 (1959).
[48] M. N. Bannerman and L. Lue, J. Chem. Phys. **133**, 124506 (2010).
[49] To simplify, we do not distinguish between the filter, which is, strictly speaking, the acceptance probability of a proposed move $c \to c'$, and the probability to move from c to c'. In our context, the difference between the two is at most a constant factor.
[50] Strictly speaking, the characteristics of the displacement are also to be included among the lifting variables.
[51] P. A. W. Lewis and G. S. Shedler, Nav. Res. Logist. Q. **26**, 403 (1979).
[52] A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, J. Comput. Phys. **17**, 10 (1975).

[53]A. J. Walker, ACM Trans. Math. Software **3**, 253 (1977).
[54]C. Bender and S. Orszag, *Advanced Mathematical Methods for Scientists and Engineers I: Asymptotic Methods and Perturbation Theory*, Advanced Mathematical Methods for Scientists and Engineers (Springer, 1978).
[55]L. M. Fraser, W. M. C. Foulkes, G. Rajagopal, R. J. Needs, S. D. Kenny, and A. J. Williamson, Phys. Rev. B **53**, 1814 (1996).
[56]S. C. Kapfer and W. Krauth, J. Phys.: Conf. Ser. **454**, 012031 (2013).

Appendix B

Publication II

JeLLyFysh-Version1.0 — a Python application for all-atom event-chain Monte Carlo

Philipp Höllmer, Liang Qin, Michael F. Faulkner, A.C. Maggs, Werner Krauth
Computer Physics Communications (2020) 107168

This paper summarizes our work described in chapters 4 and 5. There, we lay the foundation of JF, an open-source python application for all-atom simulations. All operations become events in JF. We design the mediator architecture to organize application modules and to realize the event processing. It introduces various modules, including those essential to prepare events and those to implement long-range methods in chapter 3.

This paper also features an introduction to JF's development and usage. We give examples, with the same physical systems as in appendix A, to show the functions of event handlers and the way to write configuration files, and to verify JF with high precision.

Computer Physics Communications 253 (2020) 107168

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

JELLYFYSH-Version1.0 — a Python application for all-atom event-chain Monte Carlo☆,☆☆

Philipp Höllmer [a,b], Liang Qin [a], Michael F. Faulkner [c], A.C. Maggs [d], Werner Krauth [a,e,*]

[a] Laboratoire de Physique de l'Ecole normale supérieure, ENS, Université PSL, CNRS, Sorbonne Université, Université Paris-Diderot, Sorbonne Paris Cité, Paris, France
[b] Bethe Center for Theoretical Physics, University of Bonn, Nussallee 12, 53115 Bonn, Germany
[c] H. H. Wills Physics Laboratory, University of Bristol, Tyndall Avenue, Bristol BS8 1TL, United Kingdom
[d] CNRS UMR7083, ESPCI Paris, PSL Research University, 10 rue Vauquelin, 75005 Paris, France
[e] Max-Planck-Institut für Physik komplexer Systeme, Nöthnitzer Str. 38, 01187 Dresden, Germany

ARTICLE INFO

Article history:
Received 15 August 2019
Received in revised form 10 December 2019
Accepted 16 December 2019
Available online 20 January 2020

Keywords:
Monte Carlo algorithm
Irreversible Markov chain
N-body simulation
Event-chain algorithm
Long-range potentials
Python application

ABSTRACT

We present JELLYFYSH-Version1.0, an open-source Python application for event-chain Monte Carlo (ECMC), an event-driven irreversible Markov-chain Monte Carlo algorithm for classical N-body simulations in statistical mechanics, biophysics and electrochemistry. The application's architecture mirrors the mathematical formulation of ECMC. Local potentials, long-range Coulomb interactions and multi-body bending potentials are covered, as well as bounding potentials and cell systems including the cell-veto algorithm. Configuration files illustrate a number of specific implementations for interacting atoms, dipoles, and water molecules.

Program summary

Program title: JELLYFYSH-Version1.0
Program files doi: http://dx.doi.org/10.17632/srrjt9493d.1
Licensing provisions: GNU GPLv3
Programming language: Python 3
Nature of problem: Event-chain Monte Carlo (ECMC) simulations for classical N-body simulations in statistical mechanics, biophysics and electrochemistry.
Solution method: Event-driven irreversible Markov-chain Monte Carlo algorithm.
Additional comments: The application is complete with sample configuration files, docstrings, and unittests. The manuscript is accompanied by a frozen copy of JELLYFYSH-Version1.0 that is made publicly available on GitHub (repository https://github.com/jellyfysh/JeLLyFysh, commit hash d453d497256e7270e8babc8e04d20fb6d847dee4).

1. Introduction

Event-chain Monte Carlo (ECMC) is an irreversible continuous-time Markov-chain algorithm [1,2] that often equilibrates faster than its reversible counterparts [3–7]. ECMC has been successfully applied to the classic N-body all-atom problem in statistical physics [8,9]. The algorithm implements the time evolution of a piecewise non-interacting, deterministic system [10]. Each straight-line, non-interacting leg of this time evolution terminates in an event, defined through the event time at which it takes place and through the out-state, the updated starting configuration for the ensuing leg. An event is chosen as the earliest of a set of candidate events, each of which is sampled using information contained in a so-called factor. The entire trajectory samples the equilibrium probability distribution.

ECMC departs from virtually all Monte Carlo methods in that it does not evaluate the equilibrium probability density (or its ratios). In statistical physics, ECMC thus computes neither the total potential (or its changes) nor the total force on individual point masses. Rather, the decision to continue on the current leg of the non-interacting time evolution builds on a consensus which is established through the factorized Metropolis algorithm [2]. A veto puts an end to the consensus, triggers the event, and terminates the leg (see Fig. 1). In the continuous problems for which ECMC has been conceived, the veto is caused by a single factor.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
☆☆ The review of this paper was arranged by Prof. D.P. Landau.
* Corresponding author at: Laboratoire de Physique de l'Ecole normale supérieure, Ecole normale supérieure, 24 rue Lhomond, 75005 Paris, France.
E-mail address: werner.krauth@ens.fr (W. Krauth).

https://doi.org/10.1016/j.cpc.2020.107168

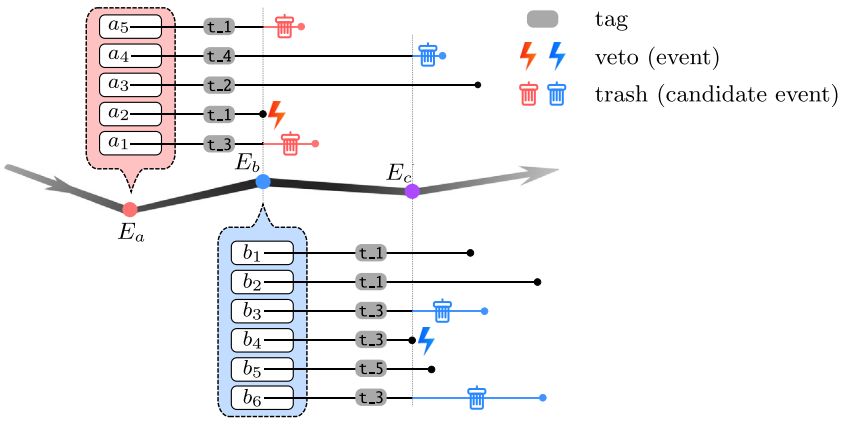


Fig. 1. ECMC time evolution. At events $E_a, E_b, E_c, \ldots$, a number of factors $(\{a_1, a_2, \ldots, a_5\}, \{b_1, b_2, \ldots, b_6\}, \ldots)$ are activated. For each leg $((E_a \to E_b), (E_b \to E_c), \ldots)$, each factor must at all times independently accept the continued non-interacting evolution, and must determine a candidate event time at which this is no longer the case. The earliest candidate event time (which determines the veto) and its out-state yield the next event (the event E_b is triggered by a_2). In JF-V1.0, after committing an event to the global state, candidate events with certain tags are trashed (tags `t_1`, `t_3` at E_b) or maintained active (tags `t_2`, `t_4` at E_b), and others are newly activated. JF introduces non-confirmed events and also pseudo-factors, which complement the factors of ECMC, and which may also trigger events.

The resulting event-driven ECMC algorithm is reminiscent of molecular dynamics, and in particular of event-driven molecular dynamics [11–13], in that there are velocity vectors (which appear as lifting variables). These velocities do not correspond to the physical (Newtonian) dynamics of the system. ECMC differs from molecular dynamics in three respects: First, ECMC is event-driven, and it remains approximation-free for any interaction potential [14], whereas event-driven molecular dynamics is restricted to hard-sphere or piecewise constant potentials. (Interaction potentials in biophysical simulation codes have been coarsely discretized [15] in order to fit into the event-driven framework [16–18].) Second, in ECMC, most point masses are at rest at any time, whereas in molecular dynamics all point masses typically have non-zero velocities. In ECMC, an arbitrary fixed number of independently active point masses with identical non-zero velocity vectors may be chosen. In most present applications of ECMC, only a single independent point mass is active at any time. The ECMC dynamics is thus very simple, yet it mixes and relaxes at a rate at least as fast as in molecular dynamics [4,6,7]. Third, ECMC by construction exactly samples the Boltzmann (canonical) distribution, whereas molecular dynamics is in principle micro-canonical, that is, energy-conserving. Molecular dynamics must therefore generally be coupled to a thermostat in order to sample the Boltzmann distribution. The thermostat also eliminates the drift in physical observables that is caused by integration errors. In contrast, ECMC is free from truncation and discretization errors.

ECMC samples the equilibrium Boltzmann distribution without being itself in equilibrium, as it violates the detailed-balance condition. Remarkably, it establishes the aforementioned consensus and proceeds from one event to the next with $\mathcal{O}(1)$ computational effort even for long-range potentials, as was demonstrated for soft-sphere models, the Coulomb plasma [4,19], and for the simple point-charge with flexible water molecules (SPC/Fw) model [20,21].

JELLYFYSH (JF) is a general-purpose Python application that implements ECMC for a wide range of physical systems, from point masses interacting with central potentials to composite point objects such as finite-size dipoles, water molecules, and eventually peptides and polymers. The application's architecture mirrors the mathematical formulation that was presented previously (see [21, Sect II]). The application can run on virtually any computer, but it also allows for multiprocessing and, in the future, for parallel implementations. It is being developed as an open-source project on GitHub. Source code may be forked, modified, and then merged back into the project (see Section 6 for access information and license issues). Contributions to the application are encouraged.

The present paper introduces the general architecture and the key features of JF. It accompanies the first public release of the application, JELLYFYSH-Version1.0 (JF-V1.0). JF-V1.0 implements ECMC for homogeneous, translation-invariant N-body systems in a regularly shaped periodic simulation box and with interactions that can be long-range. In addition, the present paper presents a cookbook that illustrates the application for simplified core examples that can be run from configuration files and validated against published data [21]. A full-scale simulation benchmark against the Lammps application is published elsewhere [22].

The JF application presented in this paper is intended to grow into a basis code that will foster the development of irreversible Markov-chain algorithms and will apply to a wide range of computational problems, from statistical physics to field theory [23]. It may prove useful in domains that have traditionally been reserved to molecular dynamics, and in particular in the all-atom Coulomb problem in biophysics and electrochemistry.

The content of the present paper is as follows: The remainder of Section 1 discusses the general setting of JF as it implements ECMC. Section 2 describes its mediator-based architecture [24]. Section 3 discusses how the eponymous events of ECMC are determined in the event handlers of JF. Section 4 presents system definitions and tools, such as the user interface realized through configuration files, the simulation box, the cell systems, and the interaction potentials. Section 5, the cookbook, discusses a number of worked-out examples for previously presented systems of atoms, dipoles or water molecules with Coulomb interactions [21]. Section 6 discusses license issues, code availability and code specifications. Section 7 presents an outlook on essential challenges and a preview of future releases of the application.

1.1. Configurations, factors, pseudo-factors, events, event handlers

In ECMC, configurations $c = \{\mathbf{s}_1, \ldots, \mathbf{s}_i, \ldots, \mathbf{s}_N\}$ are described by continuous time-dependent variables where $\mathbf{s}_i(t)$ represents the position of the ith of N point masses (although it may also stand for the continuous angle of a spin on a lattice [3]). JF is an event-driven implementation of ECMC, and it treats point masses and certain collective variables (such as the barycenter of a composite point object) on an equal footing. Rather than the time-dependent variables $\mathbf{s}_i(t)$, its fundamental particles (`Particle` objects) are individually time-sliced positions (of the

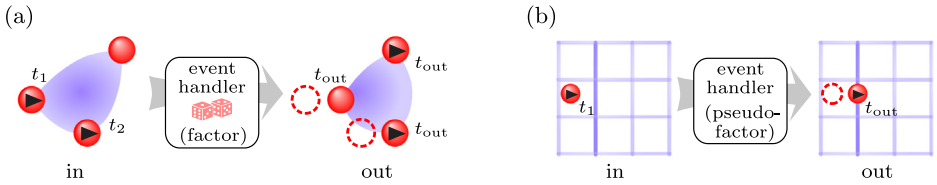


Fig. 2. Factors and pseudo-factors. (a) In-state and sampled out-state (each with two active units) for a three-unit factor M (implementing, for example, the inter-molecular bending potential U_M of Section 4.4.5). (b) In- and out-states for a cell-boundary event handler realizing a pseudo-factor. Times at which units are time-sliced are indicated. t_{out} is the event time.

point masses or composite point objects). Non-zero velocities and time stamps are also recorded, when applicable. The full information can be packed into units (`Unit` objects), that are moved around the application (see Section 1.2).

Each configuration c has a total potential $U(\{\mathbf{s}_1, \ldots, \mathbf{s}_N\})$, and its equilibrium probability density π is given by the Boltzmann weight

$$\pi(\{\mathbf{s}_1, \ldots, \mathbf{s}_N\}) = \exp\left[-\beta U(\{\mathbf{s}_1, \ldots, \mathbf{s}_N\})\right], \tag{1}$$

that is sampled by ECMC (see [21]). The total potential U is decomposed as

$$U(\{\mathbf{s}_1, \ldots, \mathbf{s}_N\}) = \sum_{M\in\mathcal{M}} U_M(\{\mathbf{s}_i : i \in I_M\}), \tag{2}$$

and the Boltzmann weight of Eq. (1) is written as a product over terms that depend on factors M, with their corresponding factor potentials U_M. A factor $M = (I_M, T_M)$ consists of an index set I_M and of a factor type T_M, and $\mathcal{M}$ is the set of factors that have a non-zero contribution to Eq. (2) for some configuration c. In the SPC/Fw water model, for example, one factor M with factor type T_M = Coulomb might describe all the Coulomb potentials between two given water molecules, and the factor index set I_M would contain the identifiers (indices) of the involved four hydrogens and two oxygens (see Section 5.3).

ECMC relies on the factorized Metropolis filter [2], where the move from a configuration c to another one, c', is accepted with probability

$$p^{\text{Fact}}(c \to c') = \prod_{M\in\mathcal{M}} \min\left[1, \exp\left(-\beta \Delta U_M\right)\right], \tag{3}$$

where $\Delta U_M = U_M(c'_M) - U_M(c_M)$. Rather than evaluating the right-hand side of Eq. (3), the product over the factors is interpreted as corresponding to a conjunction of independent Boolean random variables

$$X^{\text{Fact}}(c \to c') = \bigwedge_{M\in\mathcal{M}} X_M(c_M \to c'_M). \tag{4}$$

In this equation, $X^{\text{Fact}}(c \to c')$ is "True" (the proposed Monte Carlo move is accepted) if the independently sampled factorwise Booleans X_M are all "True". Equivalently, the move $c \to c'$ is accepted if it is independently accepted by all factors. This realizes the aforementioned consensus decision (see Fig. 1).

For an infinitesimal displacement, the random variable X_M of only a single factor M can be "False", and the factor M vetoes the consensus, creates an event, and starts a new leg. In this process, M requires only the knowledge of the factor in-state (based on the configuration c_M, and the information on the move), and the factor out-state (based on c'_M) provides all information on the evolution of the system after the event. The event is needed in order to enforce the global-balance condition (see Fig. 2a). In this process, lifting variables [25], corresponding to generalized velocities, allow one to repeat moves of the same type (same particle, same displacement), as long as they are accepted by consensus.[1] Physical and lifting variables build the overcomplete description of the Boltzmann distribution at the base of ECMC, and they correspond to the global physical and global lifting states of JF, its global state.

JF, the computer application, is entirely formulated in terms of events, beyond the requirements of the implemented event-driven ECMC algorithm. The application relies on the concept of pseudo-factors, which complement the factors in Eq. (2), but are independent of potentials and without incidence on the global-balance condition (see Fig. 2b). In JF, the sampling of configuration space, for example, is expressed through events triggered by pseudo-factors. Pseudo-factors also trigger events that interrupt one continuous motion (one "event chain" [1]) and start a new one. Even the start and the end of each run of the application are formulated as events triggered by pseudo-factors.

In ECMC, among all factors M in Eq. (2), only those for which U_M changes along one leg can trigger events. In JF, these factors are identified in a separate element of the application, the activator (see Section 2.4), and they are realized in yet other elements, the event handlers. An event handler may require an in-state. It then computes the candidate event time and its out-state (from the in-state, from the factor potential, and from random elements). The complex operation of the activator and the event handlers is organized in JF-V1.0 with the help of a tag activator, with tags essentially providing finer distinction than the factor types T_M. A tagger identifies a certain pool of factors, and also singles out factors that are to be activated for each tag. The triggering of an event associated with a given tag entails the trashing of (non-triggered) candidate events with certain tags, while other candidate events are maintained (see Fig. 1). Also, new candidate events have to be computed by event handlers with given tags. This entire process is managed by the tag activator.

1.2. Global state, internal state

In the event-driven formulation of ECMC, a point mass with identifier σ and with zero velocity is simply represented through its position $\mathbf{s}_\sigma$, while an active point mass (with non-zero velocity) is represented through a time-sliced position $\mathbf{s}_\sigma(t_\sigma)$, a time stamp $\sigma(t_\sigma)$ and a velocity $\mathbf{v}_\sigma$:

$$\mathbf{s}_\sigma(t) = \begin{cases} \mathbf{s}_\sigma & \text{if } \mathbf{v}_\sigma = 0 \\ \mathbf{s}_\sigma(t_\sigma) + (t - t_\sigma)\mathbf{v}_\sigma & \text{else (active point mass).} \end{cases} \tag{5}$$

An active point mass thus requires storing of a velocity $\mathbf{v}_\sigma$ and of a time stamp t_σ, in addition to the time-sliced position $\mathbf{s}_\sigma(t_\sigma)$. In JF, the global state traces all the information in Eq. (5). It is broken up into the global physical state, for the time-sliced positions $\mathbf{s}_\sigma$, and the global lifting state, for the non-zero velocities $\mathbf{v}_\sigma$ and the time stamps t_σ.

JF represents composite point objects as trees described by nodes. Leaf nodes correspond to the individual point masses.

[1] For concreteness, the lifting variables in this paper are referred to as "velocities", although they are not derived from mechanical equations of motions and their conservation laws. The concept of lifting variables is more general [25].

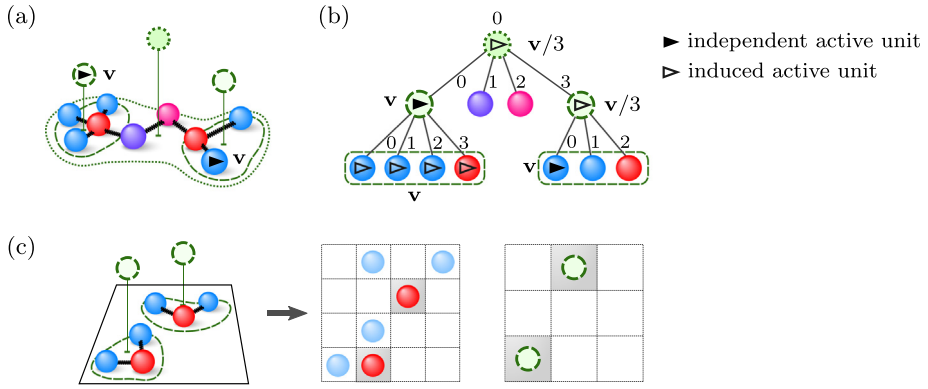


Fig. 3. Tree representation of composite point objects in JF-V1.0. (a) Molecule with functional parts. (b) Tree representation, with leaf nodes for the individual atoms and higher-level nodes for barycenters. Nodes each have a particle (a `Particle` object) containing a position vector and charge values. A unit (a `Unit` object), associated with a node, copies out the particle's identifier and its complete global-state information. (c) Internal representation of composite point objects with separate cell systems for particle identifiers on different levels. On the leaf level, only one kind of particle is tracked.

A tree's inner node may represent, for example, the barycenter of a part of a molecule, and the root node that of the entire molecule (see Fig. 3a–b). The velocities inside a composite point object are kept consistent, which means that the global lifting state includes non-zero velocities and time stamps of inner and root nodes. The storing element of the global state in JF is the state handler (see Section 2.3). The global state is not directly accessed by other elements of the application, but branches of the tree can be extracted (copied) temporarily, together with their unit information. Independent and induced units differentiate between those that appear in ECMC and those that are carried along in order to assure consistency (see Fig. 3).

For internal computations, the global state may be supplemented by an internal state that is kept, not in the state handler, but in the activator part of the application (see Section 2.4). In JF-V1.0, the internal state consists of cell-occupancy systems, which associate identifiers of composite point objects or point masses to cells. (An identifier is a generalized particle index with, in the case of a tree, a number of elements that correspond to the level of the corresponding node.) In JF, cell-occupancy systems are used for book-keeping, and also for cell-based bounding potentials. JF-V1.0 requires consistency between the time-sliced particle information and the units. This means that the time-sliced position $\mathbf{s}_\sigma(t_\sigma)$ and the time-dependent position $\mathbf{s}_\sigma(t)$ in Eq. (5) belong to the same cell (see Fig. 2b). Several cell-occupancy systems may coexist within the internal state (possibly on different tree-levels and with different cell systems, see Fig. 3c and Section 5.3.4). ECMC requires time-slicing only for units whose velocities are modified. Beyond the consistency requirements, JF-V1.0 performs time-slicing also for unconfirmed events, that is, for triggered events for which, after all, the out-state continues the straight-line motion of the in-state (see Section 3.1.2).

1.3. Lifting schemes

In its lifted representation of the Boltzmann distribution, ECMC introduces velocities for which there are many choices, that is, lifting schemes. The number of independent active units can in particular be set to any value $n_{\text{ac}} \geq 1$ and then held fixed throughout a given run. This generalizes easily from the known $n_{\text{ac}} = 1$ case [26]. A simple n_{ac}-conserving lifting scheme uses a factor-derivative table (see [21, Fig. 2]), but confirms the active out-state unit only if the corresponding unit is not active in the in-state (its velocity is `None`).

For $|I_M| > 3$, the lifting scheme (the way of determining the out-state given the in-state) is not unique, and its choice influences the ECMC dynamics [21]. In JF-V1.0, different lifting-scheme classes are provided in the JF `lifting` package. They all construct independent-unit out-state velocities for independent units that equal the in-state velocities. This appears as the most natural choice in spatially homogeneous systems [1].

1.4. Multiprocessing

In ECMC, factors are statistically independent. In JF, therefore, the event handlers that realize these factors can be run independently on a multiprocessor machine. With multiprocessor support enabled, candidate events are concurrently determined by event handlers on separate processes, using the Python `multiprocessing` module. Candidate event times are then first requested in parallel from active event handlers, and afterwards the out-state for the selected event. Given a sufficient number of available processors, out-states may be computed for candidate events in advance, before they are requested (see Section 2.1). The event handlers themselves correspond to processes that usually last for the entire duration of one ECMC run. When not computing, event handlers are either in `idle` stage waiting to compute a candidate event time or in `suspended` stage waiting to compute an out-state.

Using multiple processes instead of threads circumvents the Python global interpreter lock, but the incompressible time lag due to data exchange slows down the multiprocessor implementation of the mediator with respect to the single-processor implementation.

1.5. Parallelization

ECMC generalizes to more than one independent active unit, and a sequential, single-process ECMC computation remains trivially correct for arbitrary n_{ac} (although JF-V1.0 only fully implements the $n_{\text{ac}} = 1$ case). The relative independence of a small number of independent active units in a large system, for $1 \ll n_{\text{ac}} \ll N$, allows one to consider the simultaneous committing in different processes of n_{pr} events to the shared global state. (A conflict arises if this disagrees with what would result by committing them in a single process.) If $n_{\text{pr}} \ll N$, conflicts between processes disappear (for short-range interacting systems) if nearby active units are treated in a single process (see Fig. 4a). The parallel implementation of ECMC, for short-range interactions, is conceptually much simpler than that of event-driven molecular dynamics [27–29], and it may well extend to long-range interacting system.

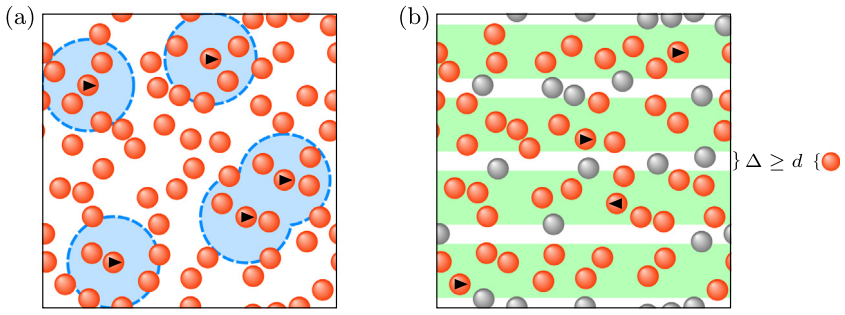


Fig. 4. Parallel ECMC with local potentials (interaction range d). (a) Multiprocess version with $n_{ac} \ll N$ active units. Nearby active units avoid conflict in a single process. (b) Domain decomposition with separated stripes. Particles in between stripes are immobile. The separation region (of width Δ) is wider than d, so that all of the conflict between stripes is avoided (see [30]).

An alternative type of parallel ECMC, domain decomposition into n_{ac} stripes, was demonstrated for two-dimensional hard-spheres systems, and considerable speed-up was reached [30]. Here, stripes are oriented parallel to the velocities, with one active unit per stripe. Stripes are isolated from each other by immobile layers of spheres [30], which, however, cause rejections (or reversals of one or more components of the velocity). The stripe decomposition eliminates all scheduling conflicts. As with any domain decomposition [27], it is restricted to physical models with short-range interactions. It is not implemented in JF-V1.0 (see Fig. 4b).

2. JF architecture

JF adopts the design pattern based on a mediator [24], which serves as the central hub for the other elements that do not directly connect to each other. In this way, interfaces and data exchange are particularly simple. The mediator design maximizes modularity in view of future extensions of the application.

2.1. Mediator

The mediator is doubled up into two modules (with `SingleProcessMediator` and `MultiProcessMediator` classes). The `run` method of either class is called by the executable `run.py` script of the application, and it loops over the legs of the continuous-time evolution. The loop is interrupted when an `EndOfRun` exception is raised, and a `post_run` method is invoked. For the single-process mediator, all the other elements are instances of classes that provide public methods. In particular, the mediator interacts with event handlers. For the multi-process mediator, each event handler has its own autonomous iteration loop and runs in a separate process. It exchanges data with the mediator through a two-way pipe. Receiving ends on both sides detect when data is available using the pipe's `recv` method.

In JF-V1.0, the same event-handler classes are used for the single-process and multi-process mediator classes. The multi-process mediator achieves this through a monkey-patching technique. It dynamically adds a `run_in_process` method to each created instance of an event handler, which then runs as an autonomous iteration loop in a process and reacts to shared flags set by the mediator. The multi-process mediator in addition decorates the event handler's `send_event_time` and `send_out_state` methods so that output is not simply returned (as it is in the single-process mediator) but rather transmitted through a pipe. Only the mediator accesses the event handlers, and these re-definitions of methods and classes (which abolish the need for two versions for each event-handler class) are certain not to produce undesired side effects.

On one leg of the continuous-time evolution, the mediator goes through nine steps (see Fig. 5). In step 1, the active global state (the part of the global state that appears in the global lifting state) is obtained from the state handler. (In the tree state handler of JF-V1.0, branches of independent units are created for all identifiers that appear in the lifting state.) Knowing the preceding event handler (which initially is `None`) and the active global state, it then obtains from the activator, in step 2, the event handlers to activate, together with their in-state identifiers. For this, the activator may rely on its internal state, but not on the global state, to which it has no access. In step 3, the corresponding in-states are extracted (that is, copied) from the state handler. In step 4, candidate event times are requested from the appropriate event handlers and pushed into the scheduler's `push_event` method. In step 5, the mediator obtains the earliest candidate event time from the scheduler's `get_succeeding_event` method and asks its event handler for the event out-state (step 6) to be committed to the global state (step 7). The activator, in step 8, determines which candidate events are to be trashed (in JF-V1.0: based on their tags), that is, which candidate event times are to be eliminated from the scheduler. Also, the activator collects the corresponding event handlers, as they become available to determine new candidate events. In the optional final step 9, the mediator may connect (*via* the input–output handler) to an output handler, depending on the preceding event handler. A mediating method defines the arguments sent to the output handler (for example the extracted global state), and considerable computations may take place there.

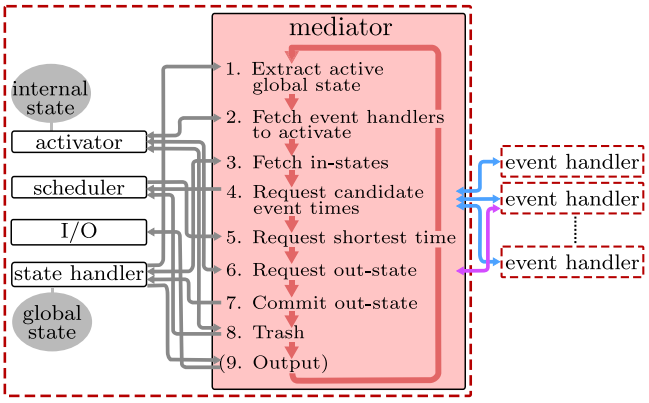


Fig. 5. JF architecture, built on the mediator design pattern. The iteration loop takes the system from one event to the next (for example from E_a to E_b in Fig. 1). All elements of JF interact with the mediator, but not with each other. The multi-process mediator interacts with event handlers running on separate processes, and exchanges data *via* pipes.

The multi-process mediator uses a single pipe to receive the candidate event time and the out-state from an event handler. In order to distinguish the received object, the mediator assigns four different stages to the event handlers (`idle`,

event_time_started, suspended, out_state_started stages). The assigned stage determines which flags can be set to start the send_event_time or send_out_state methods. It also determines the nature of the data contained in the pipe. In the idle stage, the mediator can set the starting flag after which the event handler will wait to receive the in-state through the pipe. This starts the event_time_started stage during which the event handler determines the next candidate event time and places it into the pipe. After the mediator has recovered the data from the pipe, it places the event handler into the suspended stage. If requested (by flags), the event handler can then either compute the out-state (out_state_started stage), or else revert to the event_time_started stage.

The strategy for suspending an event handler or for having it start an out-state computation (before the request) can be adjusted to the availability of physical processors on the multi-processor machine. However, in JF-V1.0, the communication *via* pipes presents a computational bottleneck.

2.2. Event handlers

Event handlers (instances of a number of classes that inherit from the abstract EventHandler class) provide the send_event_time and send_out_state methods that return candidate events. These candidate events either become events of a factor or pseudo-factor or they will be trashed (see Fig. 6).[2]

When realizing a factor or a pseudo-factor, event handlers receive the in-state as an argument of the send_event_time method. The send_out_state method then takes no argument. In contrast, event handlers that realize a set of factors or pseudo-factors request candidate event times without first specifying the complete in-state, because the element of the set that triggers the event is yet unknown at the event-time request (see Section 3.2.2 for examples of event handlers that realize sets of factors). The send_event_time method then takes the part of the in-state which is necessary to calculate the candidate event time. Also, it may return supplementary arguments together with the candidate event time, which are used by the mediator to construct the full in-state. The in-state is then an argument of the send_out_state method, as it was not sent earlier.

In JF-V1.0, each run requires a start-of-run event handler (an instance of a class that inherits from the abstract StartOfRunEventHandler class), and it cannot terminate properly without an end-of-run event handler. Section 3 discusses several event-handler classes that are provided.

2.3. State handler

The state handler (an instance of a class that inherits from the abstract StateHandler class) is the sole separate element of JF to access the global state. In JF-V1.0, the global physical state (all positions of point masses and composite point objects) is contained in an instance of the TreePhysicalState class represented as a tree consisting of nodes (each node corresponds to a Node object). Each node contains a particle (a Particle object) which holds a time-sliced position. In JF-V1.0, each leaf node may in addition have charges as a Python dictionary mapping the name of the charge onto its value.

Each tree is specified through its root node. Root nodes can be iterated over (in JF-V1.0, they are members of a list). Each node is connected to its parent and its children, which can also be iterated over. In JF-V1.0, the children are again members of a list.

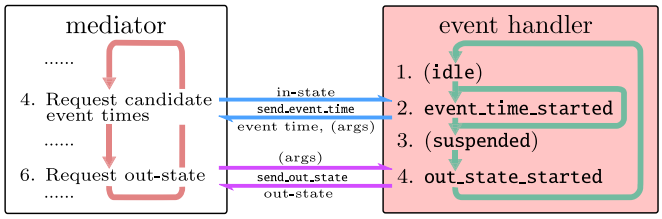


Fig. 6. Basic stages of event handlers for factors and pseudo-factors (stages 1 and 3 relevant for the multi-process mediator only). In the idle and suspended stages, the event handler is halted (*via* flags controlled by the multi-process mediator), thus liberating resources for other candidate-event-time computations. With the multi-process mediator, candidate out-states may be computed before the out-state request arrives.

These lists imply unique identifiers of nodes and their particles as tuples. The first entry of the tuple gives a node's root-node list index, followed by the indices on lower levels down to the node itself (see Fig. 3).

The global lifting state is stored in JF-V1.0 in a Python dictionary mapping the implicit particle identifier onto its time stamp and its velocity vector. This information is contained in an instance of the TreeLiftingState class. Both the physical and lifting states are combined in the TreeStateHandler which implements all methods of a state handler.

To communicate with other elements of the JF application (such as the event handlers and the activator) *via* the mediator, the state handler combines the information of the global physical and the global lifting state into units (that is, temporary Unit objects, see Fig. 7). For a given node in the state handler, its physical-state and lifting-state information is mirrored (that is, copied) to a unit containing its implicit identifier, position, charge, velocity and time stamp. All other elements can access, modify, and return units. This provides a common packaging format across JF. The explicit identifier of a unit allows the program to integrate changed units into the state handler's global state.

In the tree state handler of JF-V1.0, the local tree structure of nodes can be extracted into a branch of cnodes, that is, nodes containing units.[3] Each event handler only requires the global state reduced to a single factor in order to determine candidate event times and out-states. As a design principle in JF-V1.0, the event handlers keep the time-slicing of composite point objects and its point masses consistent. Information sent to event handlers *via* the mediator is therefore structured as branches, that is, the information of a node with its ancestors and descendants. The state handler's extract_from_global_state method creates a branch for a given identifier of a particle by constructing a temporary copy of the immutable node structure of the state handler using cnodes. Out-states of events in the form of branches can be committed to the global state using the insert_into_global_state method.

The extract_active_global_state method, the first of two additional methods provided by the state handler, extracts the part of the global state which appears in the global lifting state. The tree state handler constructs the minimal number of branches, where each node contains an active unit, so that all implicit identifiers appearing in the global lifting state are represented. The activator may then determine the factors which are to be activated. The method is also used to time-slice the entire global state (see Section 3.2.2). Second, the extract_global_state method extracts the full global state. (For

[2] A candidate event time may stem from a bounding potential, and not be confirmed for the factor potential. In JF-V1.0, unconfirmed and confirmed events are treated alike.

[3] The distinction between particles and units, as well as between nodes and cnodes stresses that the state handler can only be accessed by the mediator, although information on the physical and the lifting state must of course travel throughout the application.

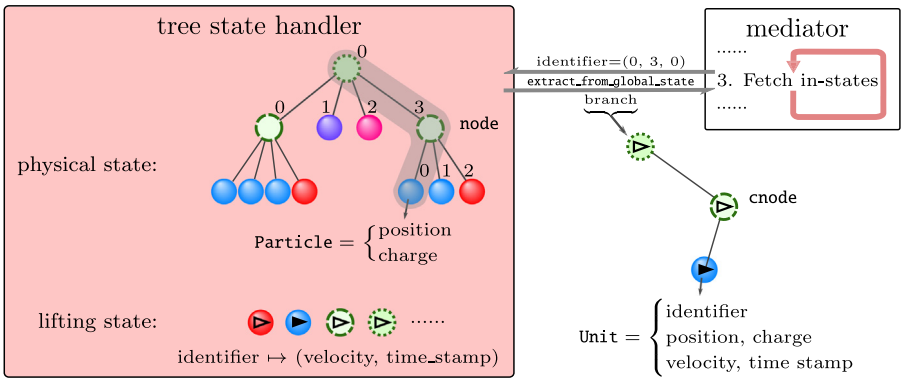


Fig. 7. Inner storage of the tree state handler and example of its `extract_from_global_state` method, applied to the global state of Fig. 3b.

the tree state handler of JF-V1.0, this corresponds to a branch for each root node.) This method does not copy the positions and velocities.

In JF-V1.0, the global physical state is initialized *via* the input handler within the input–output handler (see Section 2.6). The initial lifting state, however, is set *via* the out-state of the start-of-run event handler, which is committed to the global state at the beginning of the program (see Section 3.2.2). This means that, in JF-V1.0, the lifting state cannot be initialized from a file.

2.4. Activator

The activator, a separate element of the JF application, is an instance of a class that inherits from the abstract `Activator` class. At the beginning of each leg, the activator provides to the mediator the new event handlers which are to be run, using the `get_event_handlers_to_run` method. (As required by the mediator design pattern, no data flows directly between the activator and the event handlers, although it initially obtains their references, and subsequently manages them.) The activator also returns associated in-state identifiers of particles within the global state. The extracted parts of the global state of these in-state identifiers are needed by the event handlers to compute their candidate event times (the identifier may be `None` if no information is needed).

Once the mediator has committed the preceding event to the global state *via* the state handler, the activator finally prepares (in the `get_trashable_events` method) a list of trashable candidate events.

In JF-V1.0, the activator is an instance of the `TagActivator` class (that inherits from the `Activator` class). The tag activator's operations depend on the interdependence of tags of event handlers and their events. Event handlers receive their tag by instances of classes located in the activator and derived from the abstract `Tagger` class that are called "taggers".

A tagger centralizes common operations for identically tagged event handlers (see Fig. 8). On initialization, the tagger receives its tag (a string-valued `tag` attribute) and an event handler (that is, a single instance), of which it creates as many identical event-handler copies as needed (using the Python `deepcopy` method). Each tagger provides a `yield_identifiers_send_event_time` method which generates in-state identifiers based on the branches containing independent active units (this means that the taggers are implemented especially for the `TreeStateHandler`. The `TagActivator`, however, can be used with any state handler since it just transmits the extracted active global state). These in-states are passed (after extracting the part of the global state related to the identifiers from the state handler) to the `send_event_time` method of the tagger's event handlers. The number of event handlers inside a tagger should meet the maximum number of events with the given tag simultaneously in the scheduler. In this paper, event handlers (and their candidate events) are referred to by tags, although in JF they do not have the `tag` attribute of their taggers.

On initialization, a tagger also receives a list of tags for event handlers that it creates, as well as a list of tags for event handlers that need to be trashed. The tag activator converts this information of all taggers into its internal `_create_taggers` and `_trash_taggers` dictionaries. Additionally, the tag activator creates an internal dictionary mapping from an event handler onto the corresponding tagger (`_event_handler_tagger_dictionary`).

A call of the `get_event_handlers_to_run` method is accompanied by the event handler which created the preceding event and by the extracted active global state. The event handler is first mapped onto its tagger. The taggers returned by the `_create_taggers` dictionary then generate the in-state identifiers, which are returned together with the corresponding event handlers (in a dictionary). For the initial call of the `get_event_handlers_to_run` method no information on the preceding event handler can be provided. This is solved by initially returning the start-of-run event handler. Similarly the `_trash_taggers` dictionary is used on each call of `get_trashable_events`. The corresponding event handlers are then also liberated, meaning that the activator can return them in the next call of the `get_event_handlers_to_run` method.[4] For this, the activator internally splits the pool of all event handlers of a given tag into those with a scheduled candidate event and the ones that are available to take on new candidate events.

The activator also maintains the internal state. In JF-V1.0, the internal state consists of cell-occupancy systems. Therefore, the internal state is an instance of a class that inherits from the `CellOccupancy` class, which itself inherits from the abstract `InternalState` class. Taggers may refer to internal-state information to determine the in-states of their event handlers. The cell-occupancy system does not double up on the information available in the state handler. It keeps track of the identifier of a particle (which may correspond to a point mass or a composite point object), but does not store or copy the particle itself (see Section 4.3). The mediator can access the internal state *via* the `get_info_internal_state` method (see Fig. 8). To acquire consistency between the global state and the internal state (and between a particle and its associated unit), a pseudo-factor triggers an event for each active unit tracked by the cell-occupancy system that crosses a cell boundary (see Fig. 2b). The internal state is updated in each call of the `get_event_handlers_to_run` method.

[4] The action of the `_create_taggers` and `_trash_traggers` dictionaries can be overruled with the concept of activated and deactivated taggers. Event handlers out of deactivated taggers are not returned to the mediator.

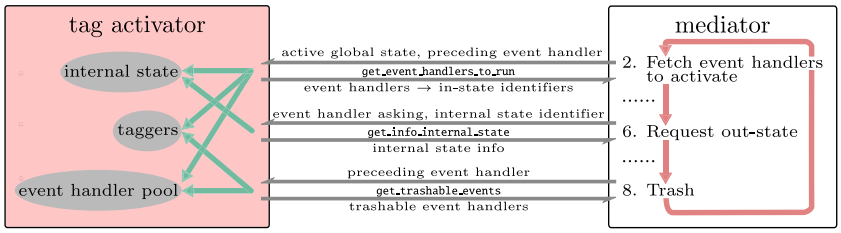


Fig. 8. Tag activator, and its complex interaction with the mediator. It readies event handlers and in-state identifiers, provides internal-state information for an out-state request, and identifies the trashable candidate events, as a function of the preceding event.

2.5. Scheduler

The scheduler is an instance of a class inheriting from the abstract `Scheduler` class. It keeps track of the candidate events and their associated event-handler references. Its `get_succeeding_event` method selects among the candidate events the one with the soonest candidate event time, and it returns the reference of the corresponding event handler. Its `push_event` method receives a new candidate event time and event-handler reference. Its `trash_event` method eliminates a candidate event, based on the reference of its event handler. In JF-V1.0, the scheduler is an instance of the `HeapScheduler` class. It implements a priority queue through the Python `heapq` module.

2.6. Input–output handler

The input–output handler is an instance of the `InputOutputHandler` class. The input–output handler connects the JF application to the outside world, and it is accessible by the mediator. The input–output handler breaks up into one input handler (an instance of a class that inherits from the abstract `InputHandler` class) and a possibly empty list of output handlers (instances of classes that inherit from the abstract `OutputHandler` class). These are accessed by the mediator only *via* the input–output handler. Output handlers can also perform significant calculations.

The input handler enters the initial global physical state into the application. JF-V1.0 provides an input handler that enters protein-data-bank formatted data (`.pdb` files) as well as an input handler which samples a random initial state. The initial state (constructed as a tree for the case of the tree state handler) is returned when calling the `read` method of the input–output handler, which calls the `read` method of the input handler.

The output handlers serve many purposes, from the output in `.pdb` files to the sampling of correlation functions and other observables, to a dump of the entire run. They obtain their arguments (for example the entire global state) *via* its `write` method. The `write` method of the input–output handler receives the desired output handler as an additional argument through the mediating methods of specific event handlers. These are triggered for example after a sampling or an end-of-run event. The corresponding event handlers are initialized with the name of their output handlers.

3. JF event-handler classes

Event-handler classes differ in how they provide the `send_event_time` and `send_out_state` methods. Event handlers split into those that realize factors and sets of factors and those that realize pseudo-factors and sets of pseudo-factors. The first are required by ECMC while the second permit JF to represent the entire run in terms of events.

3.1. Event handlers for factors or sets of factors

Event handlers that realize a factor M, or a set of factors, are implemented in different ways depending on the analytic properties of the factor potential U_M and on the number of involved independent units.

3.1.1. Invertible-potential event handlers

In JF, an invertible factor potential U_M (an instance of a class that inherits from the abstract `InvertiblePotential` class) has its event rate integrated in closed form along a straight-line trajectory (as in Fig. 1). The sampled cumulative event rate (U_M^+ in [21, Eq. (45)]) provides the `displacement` method. Together with the time stamp and the velocity of the active unit, this determines the candidate event time. In JF-V1.0, the two-leaf-unit event handler (an instance of the `TwoLeafUnitEventHandler` class) is characterized by two independent units at the leaf level. It realizes a two-particle factor with an invertible factor potential. The in-state (an argument of the `send_event_time` method) is stored internally, and it remains available for the subsequent call of the `send_out_state` method. Because of the two independent units, the lifting simply consists in these two units switching their velocities (using the internal `_exchange_velocity` method) and keeping the velocities of all induced units consistent.

3.1.2. Event handlers for factors with bounding potential

For a factor potential U_M that is not inverted (by choice or by necessity because it is non-invertible), the cumulative event rate U_M^+ is unavailable (or not used) and so is its `displacement` method. Only the `derivative` method is used. To realize such a factor without an inverted factor potential, an event handler then uses the `displacement` method of an associated bounding potential whose event rate at least equals that of U_M and that is itself invertible. A non-inverted U_M may be associated with more than one bounding potential, each corresponding to a different event handler (the molecular Coulomb factors in Section 5.2 associate the Coulomb factor potential in the same run with different bounding potentials). In JF-V1.0, a number of event handlers are instances of classes that inherit from the `EventHandlerWithBoundingPotential` class, and that realize factors with bounding potentials. Each of these event handlers translates the sampled displacement of the bounding potential into a candidate event time. On an out-state request (*via* the `send_out_state` method), the event handler confirms the event with a probability that is given by the ratio of the event rates of the factor potential and the bounding potential. The out-state consists of independent units together with their branches of induced units. For two independent units, the lifting simply consists in the application of a local `_exchange_velocity` method, which exchanges independent-unit velocities and enforces velocities for the induced units. For more than one independent unit, the out-state calculation requires a lifting. For an unconfirmed event, no lifting takes place. In JF-V1.0, confirmed and unconfirmed

events have time-sliced out-states. The inefficient treatment of unconfirmed events is the main limitation of this version of the application.

A special case of a bounding potential is the cell-based bounding potential which features piecewise cell-bounded event rates. The two independent units are localized within their respective cells, and the bounding potential's rate is for all positions of the units larger than the factor potential event rate. In JF-V1.0, the constant cell-bounded event rate is determined for all pairs of cells on initialization (see Section 4.4.4). The resulting displacement may move the independent active unit outside its cell. The proposed candidate event will then, however, be preempted by a cell-boundary event and therefore trashed (see Section 3.2.1).

3.1.3. Cell-veto event handlers

Cell-veto event handlers (instances of a number of classes that inherit from the abstract `CellVetoEventHandler` class) realize sets of factors, rather than a single factor. The factor in-states (for each element of the set) are not transmitted with the candidate-event-time calculations. Instead, the branch of the independent active unit is an argument of the `send_event_time` method. The sampled factor in-state is transmitted with the out-state request. The cell-veto event handler implements Walker's algorithm [31] in order to sample one element in the set of factors in $\mathcal{O}(1)$ operations.

Cell-veto event handlers are instantiated with an estimator (see Section 4.6). In addition, they obtain a cell system which is read in through its `initialize` method (see Section 4.2). The estimator provides upper limits for the event rate (in the given direction of motion) for the independent active unit anywhere in one specific cell (called the "zero-cell", see Section 4.3), and for a target unit in any other cell, except for a list of excluded cells. These upper limits can be translated from the zero-cell to any other active-unit cell, because of the homogeneity of the simulation box. In JF-V1.0, the cell systems for the cell-veto event handler can be on any level of the particles' tree representation (see Section 5.3.4, where a molecule-cell system tracks individual water molecules on the root level, while an oxygen-cell system tracks only the leaf nodes corresponding to oxygens).

A Walker sampler is an instance of the `Walker` class in the `event_handler` package. It provides the total event rate (`total_rate`), which, for a homogeneous periodic system, is a constant throughout a run. On a candidate-event-time request, a cell-veto event handler computes its displacement no longer through the `displacement` method of a factor potential or a bounding potential, but simply as an exponential random number divided by the total event rate. (The particularly simple `send_event_time` method of a cell-veto event handler is implemented in the abstract `CellVetoEventHandler` class, see [21] for a full description.) The Walker sampler's `sample_cell` method samples the cell of the target unit in $\mathcal{O}(1)$. It is returned, together with the candidate event time, as an argument of the `send_event_time` method. The out-state request is accompanied by the branch of the independent unit in the target cell, if it exists. Confirmation of events and, possibly, lifting are handled as in Section 3.1.2.

3.2. Event handlers for pseudo-factors or sets of pseudo-factors

The pseudo-factors of JF unify the description of the ECMC time evolution entirely in terms of events. The distinction between event handlers that realize pseudo-factors and those that realize sets of pseudo-factors remains crucial. In the former, the factor in-state is known at the candidate-event-time request. It is transmitted at this moment and kept in the memory of the event handler for use at the out-state request. For a set of pseudo-factors, the factor in-state can either not be specified at the candidate-event-time request, or would require transmitting too much data (one in-state per element of the set). It is therefore transmitted later, with the out-state-request (see Fig. 9).

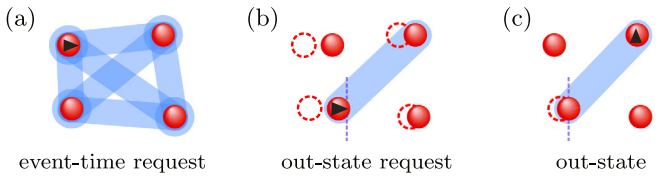


Fig. 9. Set of pseudo-factors realized by the end-of-chain event handler. (a) Set of end-of-chain pair pseudo-factors for four point masses coupling the final active unit of the old chain and the beginning active unit of the new chain. (b) At the event time, the realized pseudo-factor with the incoming active unit and the outgoing unit is known. (c) A new event chain is started. The outgoing active unit is shown.

3.2.1. Cell-boundary event handler

In the presence of a cell-occupancy system, JF-V1.0 preserves consistency between the tracked particles of the global physical state and the corresponding units (which must both belong to the same cell). This is enforced by a cell-boundary event handler, an instance of the `CellBoundaryEventHandler` class. This event handler has a single independent unit and realizes a pseudo-factor with a single identifier. A cell-boundary event leads to the internal state being updated (see Section 2.4).

On instantiation, a cell-boundary event handler receives a cell system. (Each cell-occupancy system requires one independent cell-boundary event handler.) A candidate-event-time request by the mediator is accompanied by the in-state contained in a single branch and a single unit on the level tracked by the cell-occupancy system. An out-state request is met with the cell-level-unit's position corresponding to the minimal position in the new cell.

3.2.2. Event handlers for sampling, end-of-chain, start-of-run, end-of-run

Sampling event handlers are instances of classes that inherit from the abstract `SamplingEventHandler` class. Sampling event handlers are expected to produce output (they inherit from the `EventHandlerWithOutputHandler` class and are connected, on instantiation, with their own output handler which is used in the mediating method of this event handler). Several sampling event handlers may coexist in one run. Their output handler is responsible for computing physical observables at the sampling event time (see Section 2.6). JF-V1.0 implements sampling events as the time-slicing of all the active units. A sampling event handler thus realizes a set of single-unit pseudo-factors, and the in-state is not specified at the candidate-event-time request. In JF-V1.0, the candidate event times of the sampling event handler are equally spaced. The out-state request is accompanied by branches of all independent active units, which are then all time-sliced simultaneously. Sampling candidate events are normally trashed only by themselves or by an end-of-run event.

End-of-chain event handlers are instances of classes that inherit from the abstract `EndOfChainEventHandler` class. They effectively stop one event chain and reinitialize a new one. This is often required for the entire run to be irreducible (see [21]). The end-of-chain event handler clearly realizes a set of pseudo-factors, rather than a single pseudo-factor (see Fig. 9a). An end-of-chain event handler implements a method to sample a new direction of motion. In addition, it implements a method to determine a new chain length (that gives the time of the next end-of-chain event) and, finally, the identifiers of the next independent active cnodes. For this, the end-of-chain event handler is aware of all the possible cnode identifiers (see Section 4.2).

On an event-time request, the end-of-chain event handler returns the next candidate event time (computed from the new chain length) and the identifier of the next independent active cnode. The out-state request is accompanied by the current and the succeeding independent active units and their associated branches (see Fig. 9b). For the out-state, the event handler determines the next direction of motion (see Fig. 9c).

A start-of-run event handler (an instance of a class that inherits from the abstract `StartOfRunEventHandler` class) is the sole event handler whose presence is required. The start-of-run event is the first one to be committed to the global state, because its candidate event time is set equal to the initial time of the run (usually zero) and because the activator will initially only activate the start-of-run event handler. The start-of-run event handler serves two purposes. First, it sets the initial lifting state. Second, the activator uses the start-of-run event handler as an entry point. Its tag (the `start_of_run` tag in the configuration files of Section 5) is then used to determine the events that should be activated and created thereafter.

The end-of-run event handler (an instance of a class that inherits from the abstract `EndOfRunEventHandler` class) terminates a run by raising an end-of-run exception and thus ends the mediator loop. An end-of-run event handler is usually connected, on instantiation, with its own output handler. In JF-V1.0, its `send_event_time` method returns the total run-time, which transits from the configuration file. On the `send_out_state` request, all active units are time-sliced. The end-of-run output handler may further process the global state which it receives *via* the mediating method of the end-of-run event handler.

3.3. Event handlers for rigid motion of composite point objects, mode switching

The event handlers of JF-V1.0 are generally suited for the rigid motion of composite point objects (root mode), that is, for independent non-leaf-node units (as implemented in Section 5.2.4). This is possible because all event handlers keep the branches of independent units consistent. As the subtree-node units of an independent-unit node move rigidly, the displacement is not irreducible. Mode switching into leaf mode (with single active leaf units) then becomes a necessity in order to have all factors be considered during one run and to assure the irreversibility of the implemented algorithm. In JF-V1.0, the corresponding event handlers are instances of the `RootLeafUnitActiveSwitcher` class. On instantiation, they are specified to switch either from leaf mode to root mode or vice versa.

These event handlers resemble the end-of-chain event handler, but only one of them is active at any given time. They provide a method to sample the new candidate event time based on the time stamp of the active independent unit at the time of its activation. An out-state request from one of these event handlers is accompanied by the entire tree of the current independent active unit of one mode and met with the tree of the independent active unit on the alternate mode.

4. JF run specifications and tools

The JF application relies on a user interface to select the physical system that is considered, and to fully specify the algorithm used to simulate it. Inside the application, some of these choices are made available to all modules (rather than having to be communicated repeatedly by the mediator). The application also relies on a number of tools that provide key features to many of its parts.

4.1. Configuration files, logging

The user interface for each run of the JF application consists of a configuration file that is an argument of the executable `run.py` script.[5] It specifies the physical and algorithmic parameters (temperature, system shape and size, dimension, type of point masses and composite point objects, and also factors, factor potentials, lifting schemes, total run time, sampling frequency, etc.).

A configuration file is composed of sections that each correspond to a class requiring input parameters. The `[Run]` section specifies the mediator and the setting. The ensuing sections choose the parameters in the `__init__` methods of the mediator and of the setting. Each section contains pairs of properties and values. The property corresponds to the name of the argument in the `__init__` method of the given class, and its value provides the argument (see Fig. 12). The content of the configuration file is parsed by the `configparser` module and passed to the JF factory (located in the `base.factory` module) in `run.py`. Standard Python naming conventions are respected in the classes built by the JF factory, which implies the naming conventions in the configuration file (see Section 6.3 for details). Within the configuration file, sections can be written in any order, but their explicit nesting is not allowed. Nesting is, however, implicit in the structure of the configuration file.

The JF application returns all output *via* files under the control of output handlers. Run-time information is logged (the Python `logging` module is used). Logged information can range from identification of CPUs to the initialization information of classes, run-time information, etc. Logging output (to standard output or to a file) can take place on a variety of levels from `DEBUG` to `INFO` to `WARNING` that are controlled through command-line arguments of `run.py`. An identification hash of the run is part of the logging output. It also tags all the output files so that input, output and log files are uniquely linked (the Python `uuid` module is used).

4.2. Globally used modules

JF-V1.0 requires that all trees representing composite point objects are identical and of height at most two. Furthermore, in the *NVT* physical ensemble, the particle number, system size and temperature remain unchanged throughout each run. After initialization, as specified in the configuration file, these parameters are stored in the JF `setting` package and the modules therein, which may be imported by all other modules, which can then autonomously construct identifiers. Helper functions for periodic boundary conditions (if available) and for the sampling of random positions are also accessible.

JF-V1.0 implements hypercubic and hypercuboid setting modules. Both settings define the inverse temperature and also the attributes of all possible particle identifiers, which are broadcast directly by the `setting` package. In contrast, the parameters of the physical system are accessed only using the modules of the specific setting (for example the `setting.hypercubic_setting` module).[6] The `setting` package and its modules are initialized by classes which inherit from the abstract `Setting` class. The `HypercuboidSetting` class defines only the hypercuboid setting, the `HypercubicSetting` class, however, sets up both the `hypercubic_setting` and the `hypercuboid_setting` modules together with the `setting` package. This allows modules that are specifically implemented for a hypercuboid setting to be used with the hypercubic setting.

[5] Configuration files follow the INI-file format and, in JF, feature the extension `.ini`.

[6] Attributes in the `setting` package are copied to the modules for convenience.

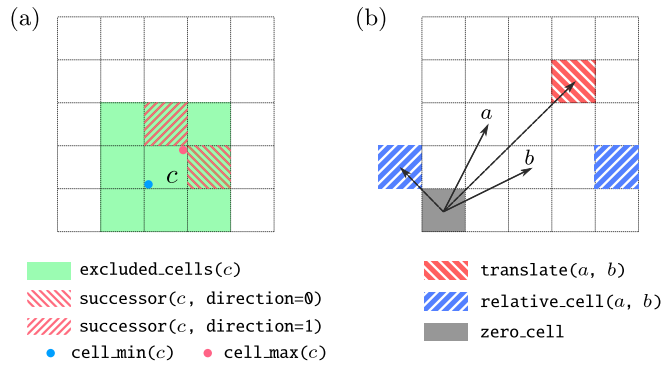


Fig. 10. Cell methods. (a) `excluded_cells`, `successor`, `cell_min` and `cell_max` methods required by the abstract `Cells` class. Horizontal and vertical directions are indexed as 0 and 1, respectively. (b) `translate` and `relative_cell` methods (illustrated by vectors) required by the `PeriodicCells` class, in addition to the methods of the `Cells` class. Periodic boundary conditions are required, and the two blue (north-east hatched) cells are identical. The periodic-cell system's origin is given by the `zero_cell` property.

Each setting can implement periodic boundaries, by inheriting from the abstract `PeriodicBoundaries` class and by implementing its methods. Since many modules of JF only rely on periodic boundaries but not on the specific setting, the `setting` package gives also access to the initialized periodic boundary conditions. Similarly, a function to create a random position is broadcast by the setting package. All the configuration files in Section 5 are for a three-dimensional cubic simulation box, that is, use the hypercubic setting with `dimension` = 3.

Additional useful modules are located in the JF `base` package. The abstract `Initializer` class located in the `initializer` module enforces the implementation of an `initialize` method. This method must be called ahead of other public methods of the inheriting class. The `strings` module provides functions to translate strings from snake to camel case and vice versa, as well as to translate a package path into a directory path. Helper functions for vectors, such as calculating the norm or the dot product, are located in the `vectors` module.

4.3. Cell systems and cell-occupancy systems

A cell-occupancy system is an instance of a class that inherits from the abstract `CellOccupancy` class, located in the activator. Any cell-occupancy system is associated with a cell system, itself an instance of a class that inherits from the abstract `Cells` class.

In JF-V1.0, the cell system consists of a regular grid of cells that are referred to through their indices. Cells can be iterated over with the `yield_cells` method. For a given cell, the excluded cells are accessed by the `excluded_cells` method, the successor cell in a suitably defined direction by the `successor` method and the lower and the upper bound position in each direction through the `cell_min` and `cell_max` methods (see Fig. 10a). Finally, the `position_to_cell` method returns the cell for a given position. Cell systems with periodic boundary conditions are described as periodic cell systems (instances of classes that inherit from the abstract `PeriodicCells` class, which itself inherits from the `Cells` class). Their `zero_cell` property corresponds to the cell located at the origin. Their `relative_cell` method receives a cell and a reference cell, and establishes equivalence between the relative and the zero-cell. The inverse to this is the `translate` method (see Fig. 10b).

A cell-occupancy system (which is located in the activator) associates the identifiers of cell-based particles and of surplus particles with a cell. It also stores active cells, that is, cells that contain an active unit (see Fig. 11). Cell-based and surplus particles in the state handler correspond to units with zero velocity, so that there is no real distinction between units and particles for them. The cell-occupancy system inherits from the abstract `InternalState` class and therefore provides `__getitem__` and `update` methods. The former returns a particle identifier based on a cell, whereas the latter updates the cell occupancies based on the currently active units. This keeps the internal state consistent with the global state. Moreover, the cell-occupancy may iterate over surplus particle identifiers *via* the `yield_surplus` method. The active cells and the corresponding identifiers of the active units are generated using the `yield_active_cells` method (see Fig. 11).

JF-V1.0 implements the `SingleActiveCellOccupancy` class which features only a single active cell and which keeps the active unit identifier among its private attributes. The cell-based particle identifiers are stored in an internal `_occupant` list, and surplus-particle identifiers are stored in an internal `_surplus` dictionary mapping the cell indices onto the surplus-particle identifiers.

The stored cell-occupancy system can address different levels of composite particles: one cell-occupancy system may track particles (and units) associated to root nodes, while another may track particles associated with leaf nodes. This is set on initialization *via* the `cell_level` property which equals the length of the particle identifier tuple. The concerned cell system is itself set on initialization. An indicator charge allows one to select specific particles on a given level for tracking.

A single run can feature several internal states stored within the activator. These instances may rely on different cell-occupancy systems and cell systems. For consistency between internal states and the global state, each cell-occupancy system requires its own cell-boundary event handler.

4.4. Inter-particle potentials and bounding potentials

In JF, potentials play a dual role, not only as factor potentials U_M in event handlers but also as bounding potentials for factor potentials U_M. Potentials are located in the JF `potential` package. They inherit from the abstract `Potential` class and provide a `derivative` method. They may also inherit from the abstract `InvertiblePotential` class, and must then additionally provide a `displacement` method. In JF-V1.0, derivatives and displacements are with respect to the positive change of the active unit along one of the coordinates (indicated through the `direction` argument). For a potential $U(\mathbf{r}_j - \mathbf{r}_i)$ and `direction` $= 0$, the derivative is, for example, given by $\left[\partial/\partial x_i U(\mathbf{r}_j - \mathbf{r}_i)\right]$.

4.4.1. Inverse-power-law potential, Lennard-Jones potential

The inverse-power-law potential (an instance of the `InversePowerPotential` class that inherits from the abstract `InvertiblePotential` class) concerns the separation vector $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ (without periodic boundary conditions, in d-dimensional space) between a unit j and an active unit i as

$$U_{(\{i,j\},\,\mathrm{inv})}(\mathbf{r}_{ij}, c_i, c_j) = c_i c_j k \left|\frac{1}{\mathbf{r}_{ij}}\right|^p . \tag{6}$$

Here, k and $p > 0$ correspond to the `prefactor` and `power` parameters set on initialization. The charges c_i and c_j are entered into the methods of the potential as parameters `charge_one` and `charge_two`. This allows one instance of the `InversePowerPotential` class to be used for different charges. The `derivative` method is straightforward, while the `displacement` method distinguishes the repulsive ($c_i c_j k > 0$) and the attractive ($c_i c_j k < 0$) cases.

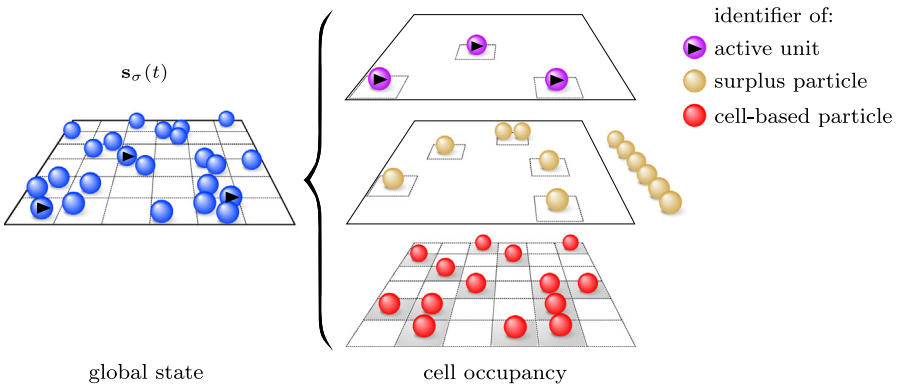


Fig. 11. Cell-occupancy system, an internal state of the activator, with active units accounted for differently from surplus and cell-based particles. Only a fixed number of cell-based particle identifiers are allowed per cell (here one per cell). Surplus-particle identifiers may be iterated over from the outside of the cell-occupancy system with the `yield_surplus` method. In JF-V1.0, surplus particles form an internal dictionary mapping the cell onto the particle identifier.

The Lennard-Jones potential (an instance of the `LennardJonesPotential` class) implements the Lennard-Jones potential

$$U_{(\{i,j\},\mathrm{LJ})}(\mathbf{r}_{ij}) = k_{\mathrm{LJ}}\left[\left(\frac{\sigma}{|\mathbf{r}_{ij}|}\right)^{12} - \left(\frac{\sigma}{|\mathbf{r}_{ij}|}\right)^{6}\right], \tag{7}$$

where $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ is the separation vector (without periodic boundary conditions, in d-dimensional space) between a unit j and an active unit i. The parameters `prefactor` and `characteristic_length` set on instantiation correspond to k_{LJ} and σ. This Lennard-Jones potential provides a straightforward `derivative` method. Its `displacement` method relies on an algebraic inversion.

4.4.2. Displaced-even-power-law potential

An instance of the `DisplacedEvenPowerPotential` class that inherits from the abstract `InvertiblePotential` class, the displaced-even-power-law potential, concerns the separation vector $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ (without periodic boundary conditions, in d-dimensional space) between a unit j and an active unit i

$$U_{(\{i,j\},\,\mathrm{depp})}(\mathbf{r}_{ij}) = k_{\mathrm{depp}}\left(|\mathbf{r}_{ij}| - r_0\right)^p, \tag{8}$$

where $k_{\mathrm{depp}} > 0$, $p \in \{2, 4, 6, \dots\}$, and r_0, respectively, are the parameters `prefactor`, `power`, and `equilibrium_separation` parameters set on instantiation. The `derivative` and `displacement` methods are provided analytically.

4.4.3. Merged-image Coulomb potential and bounding potential

An instance of the `MergedImageCoulombPotential` class that inherits from the abstract `Potential` class, the merged-image Coulomb potential is defined for a separation vector $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ (with periodic boundary conditions in a three-dimensional cubic simulation box of side L) between a unit j and an active unit i as

$$U_{\mathrm{C}}(\mathbf{r}_{ij}, c_i, c_j) = \sum_{\mathbf{n}\in\mathbb{Z}^3} c_i c_j / |\mathbf{r}_{ij} + \mathbf{n}L|. \tag{9}$$

The conditionally convergent sum in Eq. (9) can be consistently defined in terms of "tin-foil" boundary conditions [32]. It then yields an absolutely convergent sum, partly in real space and partly in Fourier space (see [21, Sect. IIIA]),

$$U_{\mathrm{C}}(\mathbf{r}_{ij}, c_i, c_j) = c_i c_j \left[\sum_{\mathbf{n}\in\mathbb{Z}^3} \frac{\operatorname{erfc}(\alpha|\mathbf{r}_{ij} + \mathbf{n}L|)}{|\mathbf{r}_{ij} + \mathbf{n}L|} + \frac{4\pi}{L^3}\sum_{\mathbf{q}\neq(0,0,0)} \frac{\mathrm{e}^{-\mathbf{q}^2/(4\alpha^2)}}{\mathbf{q}^2}\cos\left(\mathbf{q}\cdot\mathbf{r}_{ij}\right)\right], \tag{10}$$

with α a tuning parameter and $\mathbf{q} = 2\pi\mathbf{m}/L$, $\mathbf{m} \in \mathbb{Z}^3$. JF-V1.0 provides this class for a cubic simulation box only. Its parameters are optimized to reach machine precision for its `derivative` method. Summations over $\mathbf{n}$ and $\mathbf{m}$ are taken within spherical cutoffs, namely for all $|\mathbf{n}| \leq$ `position_cutoff` and $|\mathbf{m}| \leq$ `fourier_cutoff` excluding $\mathbf{m} = (0, 0, 0)$. (The potential in Eq. (10) differs from the tin-foil Coulomb potential in a constant self-energy term that does not influence the derivatives.)

The merged-image Coulomb potential is not invertible. When it serves as a factor potential, bounding potentials provide the required `displacement` method. JF-V1.0 provides a merged-image Coulomb bounding potential as an instance of the `InversePowerCoulombBoundingPotential` class, with

$$U_{\mathrm{C,Bounding}}(\mathbf{r}_{ij}, c_i, c_j) = c_i c_j k_{\mathrm{C}} / |\mathbf{r}_{ij,0}|. \tag{11}$$

Here, $\mathbf{r}_{ij,0}$ is the minimum separation vector, that is, the vector between $\mathbf{r}_i$ and the closest image of $\mathbf{r}_j$ under the periodic boundary conditions. (The merged-image Coulomb bounding potential thus involves no sum over periodic images.) The constant k_{C} must satisfy

$$k_{\mathrm{C}} \geq \max_{\mathbf{r}\in[-L/2,L/2]^3} \frac{|\mathbf{r}|^3}{x}\frac{\partial U_{\mathrm{C}}(\mathbf{r}, 1, 1)}{\partial x}, \tag{12}$$

so that the factor-potential event rate is bounded. A value $k_{\mathrm{C}} \gtrsim 1.5836$ (the parameter `prefactor`) is appropriate for a cubic simulation box. The merged-image Coulomb bounding potential is closely related to the inverse-power-law potential of Eq. (6) with $p = 1$, although the restriction to the minimum separation vector makes that the latter cannot be used directly.

4.4.4. Cell-based bounding potential

A cell-based bounding potential is an instance of a class that inherits from the abstract `InvertiblePotential` class. It bounds the derivative of the factor potential inside certain cell regions by constants. These constants can be computed analytically on demand or even sampled using a separate Monte Carlo algorithm. On initialization, a cell-based bounding potential receives an estimator (see Section 4.6). Also the information about the cell system is transmitted. Then, the cell-based bounding potential iterates over all pairs of cells (making use of periodic boundary conditions) and determines an upper and a lower bound derivative for the factor units being in those cells for each possible direction of motion using the estimator. Here, the cell-based bounding potential is not applied to excluded cells, where the cell-bounded event rate diverges, is simply too large, or otherwise inappropriate.

The constant-derivative bound leads to a piecewise linear invertible bounding potential. The call of the `displacement`

method is accompanied by the direction of motion, the charge product, the sampled potential change and the cell separation. In JF-V1.0, any cell-based bounding potential requires a cell-boundary event handler, that detects when the displacement proposed by the `displacement` method in fact takes place outside the cell for which it is computed.

4.4.5. Three-body bending potential

The SPC/Fw water model of Section 5.3 includes a bending potential (an instance of the `BendingPotential` class), which describes the fluctuations in the bond angle within each molecule. For the three units i, j, and k within such a molecule (with j being the oxygen), it is given by

$$U_{(\{i,j,k\},\,\mathrm{bending})}(\mathbf{r}_{ij}, \mathbf{r}_{jk}) = \frac{1}{2} k_b \Big[\phi_{\{i,j,k\}}(\mathbf{r}_{ij}, \mathbf{r}_{jk}) - \phi_0\Big]^2 . \tag{13}$$

Here, $\phi_{\{i,j,k\}}(\mathbf{r}_{ij}, \mathbf{r}_{jk})$ denotes the internal angle between the two hydrogen–oxygen legs. The constants k_b and ϕ_0 are set on initialization of the potential (see [21]). The `derivative` method is provided explicitly for this potential, which is, however, not invertible.

In JF-V1.0, an associated piecewise linear bounding potential is constructed dynamically by an event handler.[7] Here, the event handler speculates on a constant bounding derivative through its position between two subsequent time-sliced positions of the active unit: $q_{\mathrm{bounding}} = \max\{q(\mathbf{r}), q(\mathbf{r} + \mathbf{v}\Delta t)\} + \mathrm{const}$ where $q(\mathbf{r})$ is the potential derivative at $\mathbf{r}$. The interval length $|\mathbf{v}\Delta t|$ and the constant offset are input from the configuration file. Fine-tuning provides an efficient bounding potential that does not under-estimate the event rate, yet limits the ratio of unconfirmed events.

4.5. Lifting schemes

Event handlers with more than two independent units require a lifting scheme (an instance of a class that inherits from the `Lifting` class). The event handler calls a method of the lifting scheme to compute its out-state. At first, the event handler prepares factor derivatives of relevant time-sliced units. The derivative table (see [21, Figs 2 and 10]) is filled with unit identifiers, factor derivatives and activity information through its `insert` method. Finally, the event handler calls the `get_active_identifier` method that returns the identifier of the next independent active unit. The lifting scheme's `reset` method deletes the derivative table. It is called before the first derivative is inserted. JF-V1.0 implements the ratio, inside-first and outside-first lifting schemes for a single independent active unit (see [21, Sect. IV]).

4.6. Estimator

Estimators (instances of a class that inherits from the abstract `Estimator` class) determine upper and lower bounds on the factor derivative in a single direction between a minimum and maximum corner of a hypercuboid for the possible separations. For this, they provide the `derivative_bound` method. Both upper and lower bounds are useful when the potential can have either positive and negative charge products (as happens for example for the merged-image Coulomb potential as a function of the two charges). In general, an estimator compares the factor derivatives for different separations in the hypercuboid to obtain the bounds. These are corrected by a prefactor and optionally by an empirical bound, which are set on instantiation (together with the factor potential).

JF-V1.0 provides estimators which either regard regularly or randomly sampled separations within the hypercuboid. The inner-point and boundary-point estimators vary the separation evenly within the hypercuboid or on the edge of the hypercuboid, respectively. For these separations, the factor potential derivatives (optionally including charges) are compared. Two more estimators consider the interaction between a charged active unit and two oppositely charged target units within a dipole. Here, the factor derivative is summed for the two possible active-target pairs. A Monte-Carlo estimator distributes both the separation and the dipole orientation randomly. The dipole-inner-point estimator varies the separations evenly but aligns the dipole orientation along the direction of the gradient of the factor derivative. The implemented estimators are appropriate for the cookbook examples of Section 5, where the upper and lower bounds on the factor derivatives (and equivalently on the event rates) must be computed for a small number of cell pairs only.

5. JF cookbook

The configuration files[8] in JF-V1.0 introduce the key features of the application by constructing runs for two charged point masses, for two interacting dipoles of charges, and for two interacting water molecules (using the SPC/Fw model). All configuration files are for a three-dimensional cubic simulation box with periodic boundary conditions, and they reproduce published data [21].

As specified in their `[Run]` sections, the configuration files use a single-process mediator (an instance of the `SingleProcessMediator` class), and the `setting` package is initialized by an instance of the `HypercubicSetting` class (see for example Fig. 12). All configuration files in the directory use a heap scheduler (an instance of the `HeapScheduler` class), a tree state handler (instance of the `TreeStateHandler` class), as well as a tag activator (an instance of the `TagActivator` class) in order to activate event handlers, trash candidate events and prepare in-states.

The `start_of_run`, `end_of_run`, `end_of_chain`, and `sampling` event handlers (that realize common pseudo-factors) are implemented in largely analogous sections across all the configuration files, although their parent sections (that define the corresponding taggers) provide different tag lists for trashing and activation of event handlers. The corresponding tagger sections are presented in detail in Section 5.1.1, and only briefly summarized thereafter.

5.1. Interacting atoms

The configuration files in the `coulomb_atoms` directory of JF-V1.0 implement the ECMC sampling of the Boltzmann distribution for two identical charged point masses. They interact with the merged-image Coulomb pair potential and are described by a Coulomb pair factor. One of the two point masses is active, and it moves either in the $+x$, $+y$, or $+z$ direction. Statistically equivalent output is obtained for the merged-image Coulomb pair potential (the factor potential) associated with the inverse-power bounding potential (Section 5.1.1), or else with a cell-based bounding potential, either realized directly (Section 5.1.2), or through a cell-veto event handler (Section 5.1.3). Although the configuration files use the language of Section 1.2 for the representation of particles, all trees and branches are trivial, and each root node is also a leaf node.

[7] Instance of the `FixedSeparationsEventHandlerWithPiecewiseConstantBoundingPotential` class.

[8] Configuration files in the `src/config_files/2018_JCP_149_064113` directory tree are described in this section.

(a)

```
class SingleProcessMediator:
      def __init__(self, input_output_handler:  InputOutputHandler, state_handler:  StateHandler,
                         scheduler:  Scheduler, activator:  Activator):
```

(b)

```
[section] property value

[Run]
mediator = single_process_mediator
setting = hypercubic_setting

[HypercubicSetting]
system_length = 1
beta = 2
dimension = 3

[SingleProcessMediator]
state_handler = tree_state_handler
scheduler = heap_scheduler
activator = tag_activator
input_output_handler = input_output_handler

[TagActivator]
taggers =
         coulomb (factor_type_map_in_state_tagger),
         sampling (no_in_state_tagger),
         end_of_chain (no_in_state_tagger),
         start_of_run (no_in_state_tagger),
         end_of_run (no_in_state_tagger)

[Coulomb]
event_handler = two_leaf_unit_bounding_potential
                _event_handler

[TwoLeafUnitBoundingPotentialEventHandler]
potential = merged_image_coulomb_potential
bounding_potential = inverse_power_coulomb
                     _bounding_potential

[Sampling]
event_handler = fixed_interval_sampling_event_handler

[FixedIntervalSamplingEventHandler]
sampling_interval = 0.56789
output_handler = separation_output_handler

[EndOfChain]
event_handler = same_active_periodic_direction
                _end_of_chain_event_handler

[SameActivePeriodicDirectionEndOfChainEventHandler]
chain_length = 0.78965

[TreeStateHandler]
physical_state = tree_physical_state
lifting_state = tree_lifting_state

[InputOutputHandler]
output_handlers = separation_output_handler
input_handler = random_input_handler
```

Fig. 12. Configuration file `coulomb_atoms/power_bounded.ini`. (a) A typical `__init__` method of a JF class. (b) Excerpts of the configuration file (some lines split for clarity). Sections with properties and values that correspond to the argument names in the `__init__` methods of JF classes.

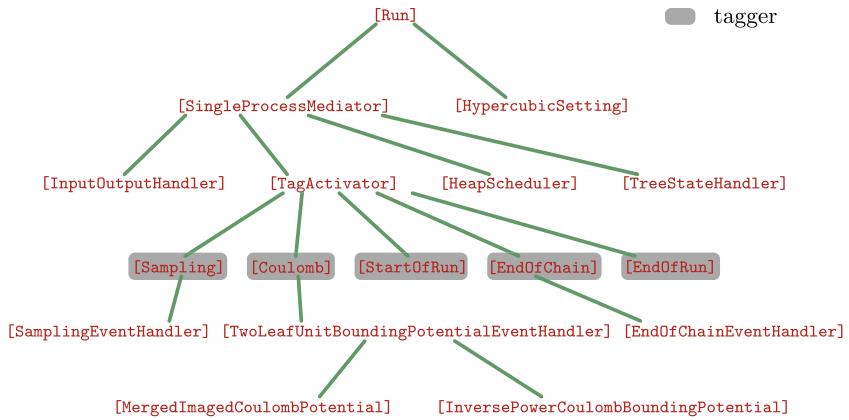


Fig. 13. Tree representation of the sections in the configuration file `coulomb_atoms/power_bounded.ini`. Only part of the tree is shown and names of event handlers for sampling and end-of-chain are shortened. The children of the `[TagActivator]` section correspond to all the declared taggers, which point towards sections for their associated event-handler classes.

5.1.1. Atomic factors, inverse-power Coulomb bounding potential

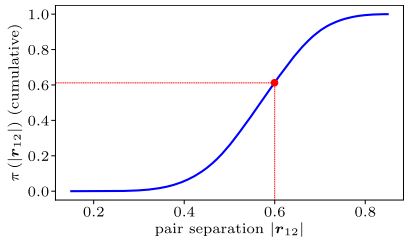
The configuration file `coulomb_atoms/power_bounded.ini` implements a single Coulomb pair factor with the merged-image Coulomb factor potential that is associated with its inverse-power Coulomb bounding potential. The same event handler realizes this factor for any separation of the point masses. The activator requires no internal state.

Although it would be feasible to directly implement (that is, hard-wire) all event handlers for this simple system, the tag activator is used. All event handlers are thus accessed *via* taggers that are listed, together with their tags, in the `[TagActivator]` section (see Fig. 13 for a tree representation of the sections). The `coulomb` tagger is an instance of the `FactorTypeMapInStateTagger` class, indicating that its event handlers require a specific in-state created from a pattern stored in a file indicated in the `[FactorTypeMaps]` section. This pattern mirrors the factor index sets and factor types for a system with two root nodes. The entry `[0, 1], Coulomb` in this file indicates that, for two point masses, a Coulomb potential would act between particles 0 and 1. From this information, the tagger's `yield_identifiers_send_event_time` method generates all the in-state identifiers for any number of point masses.

The `[Coulomb]` section specifies input for the `coulomb` tagger's tag lists (the `creates` list and the `trashes` list). Here, a `coulomb` event creates and trashes only `coulomb` candidate events (see the configuration file of Section 5.3.1 for different tag lists for the same `coulomb` event handlers).

The `[Coulomb]` section further specifies that the `coulomb` event handler is an instance of the `TwoLeafUnitBoundingPotentialEventHandler` class and that, for two point masses, only one `coulomb` event handler is needed. The corresponding section[9] specifies the factor potential to be an instance of the `MergedImageCoulombPotential` class. It specifies the bounding potential as an instance of the `InversePowerCoulombBoundingPotential` class. The `sampling`, `end_of_chain`, `start_of_run` and `end_of_run` taggers are all instances of the `NoInStateTagger` class (their event handlers require no in-state), and also provide their event handlers and their tag lists, which are then transmitted to the tag activator. Each of these taggers' `yield_identifiers_send_event_time` methods yields

[9] The `[TwoLeafUnitBoundingPotentialEventHandler]` section. The section name may be replaced by an alias to respect the tree structure of the configuration file (see Section 5.2.1).



method	$\pi(\lvert\mathbf{r}_{12}\rvert < 0.6)$
①	0.61305 ± 0.00003
②	0.61306 ± 0.00006
③	0.61299 ± 0.00008
④	0.61307 ± 0.00012

Fig. 14. Cumulative histogram of the pair separation $|\mathbf{r}_{12}|$ (nearest image) for two charges in a periodic three-dimensional cubic simulation box with periodic boundary conditions ($\beta c_1 c_2 = 2$, $L = 1$). ①: Reversible Markov-chain Monte Carlo (see [21, Fig. 8]) ②: Method of Section 5.1.1 ③: Method of Section 5.1.2 ④: Method of Section 5.1.3, each with standard errors for $\pi(|\mathbf{r}_{12}| < 0.6)$.

the in-state identifiers needed by the taggers' event handlers in order to realize corresponding factors or pseudo-factors.

The configuration file's `[InputOutputHandler]` section specifies the input–output handler. It consists of the separation-output handler (an instance of the `SeparationOutputHandler` class), which is connected to the `sampling` event handler. In the present example, it samples the nearest-image separation (under periodic boundary conditions) of any two point masses. The initial global physical state is created randomly by the random-input handler (an instance of the `RandomInputHandler` class). The configuration file `coulomb_atoms/power_bounded.ini` reproduces published data (see Fig. 14,②).

The configuration file `coulomb_atoms/power_bounded.ini` can be modified for N point masses. In the `[RandomInputHandler]` section, the number of root nodes must then equal N. In the `[Coulomb]` section, the number of event handlers must be set to at least $N-1$ (this instructs the `Coulomb` tagger to deep-copy the required number of event handlers). Without changing the factor-type map with respect to the $N = 2$ case, each event handler will be presented with the correct in-state corresponding to a pair of units with one of them being the active unit. The complexity of the implemented algorithm is $\mathcal{O}(N)$ per event.

5.1.2. Atomic factors, cell-based bounding potential

The configuration file `coulomb_atoms/cell_bounded.ini` implements a single Coulomb pair factor with the merged-image Coulomb potential, just as the configuration file of Section 5.1.1. However, a cell-occupancy internal state associates the factor potential with a cell-based bounding potential. The target (non-active) unit may be cell-based or surplus (see Fig. 11). The target unit may also be in an excluded nearby cell of the active cell (see Fig. 10), for which the cell-based bounding potential cannot be used. In consequence, three taggers correspond to distinct event handlers that together realize the Coulomb pair factor. The consistency requirement of JF-V1.0 assures that particles and units are always associated with the same cell.

Taggers and their tags are listed in the `[TagActivator]` section. The `coulomb_cell_bounding` tagger, for example, appears as an instance of the `CellBoundingPotentialTagger` class. The `coulomb_cell_bounding` event handler then realizes the Coulomb factor unless the cell of the target particle is excluded with respect to the active cell and unless it is a surplus particle (in these cases the tagger does not generate any in-state for its event handler). Otherwise, the Coulomb pair factor is realized by a `coulomb_surplus` or a `coulomb_nearby` event handler. (For two units, as the active unit is taken out of the cell-occupancy system, no surplus candidate events are ever created.)

The cell-occupancy systems (an instance of the `SingleActiveCellOccupancy` class) are also declared in the `[TagActivator]` section and further specified in the `[SingleActiveCellOccupancy]` section. The associated cell system is described in the `[CuboidPeriodicCells]` section. The internal state, set in the `[SingleActiveCellOccupancy]` section, has no charge value. This indicates that the identifiers of all particles at the cell level (here `cell_level = 1`) are tracked (see Section 5.3.2 for an example where this is handled differently).

The `coulomb_nearby` tagger, an instance of the `ExcludedCellsTagger` class, yields the identifiers of particles in excluded cells of the active cell, by iterating over excluded cells and by checking whether they contain appropriate identifiers. In the same way, the `coulomb_surplus` tagger relies on the `yield_surplus` method of the cell-occupancy system to generate in-states.

To keep the internal state consistent with the global state, a cell-boundary event handler is used in the `CellBoundaryTagger` class (together, this builds `cell_boundary` candidate events). The cell-boundary tagger just yields the active-unit identifier as the in-state used in the corresponding event handler. The configuration file `coulomb_atoms/cell_bounded.ini` reproduces published data (see Fig. 14,③).

To adapt the configuration file for $N > 2$ point masses (from the $N = 2$ case that is provided), in the `[RandomInputHandler]` section, `number_of_root_nodes` must be set to N. The number of `coulomb_cell_bounding`, `coulomb_nearby`, and `coulomb_surplus` event handlers must be increased. Surplus particles can now exist. The number of event handlers to allow for depends on the cell system, whose parameters must be adapted in order to limit the number of surplus particles, and also to retain useful cell-based bounds for the Coulomb event rates.

5.1.3. Atomic factors, cell-veto

The configuration file `coulomb_atoms/cell_veto.ini` implements a Coulomb pair factor together with the merged-image Coulomb potential. A cell-occupancy internal state is used. The Coulomb pair factor is then realized, among others, by a cell-veto event handler, which associates the merged-image Coulomb potential with a cell-based bounding potential.

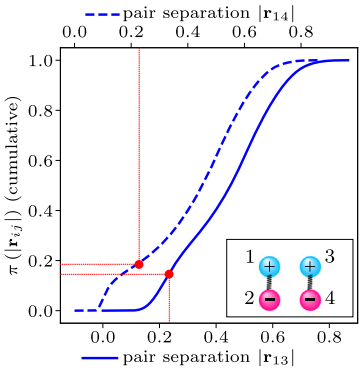
All the Coulomb pair factors of the active particle with target particles that are neither excluded nor surplus are taken together in a set of Coulomb factors, and realized by a single `coulomb_cell_veto` event handler. The candidate event time can be calculated with the branch of the active unit as the in-state, which is implemented in the `CellVetoTagger` class. (The cell-veto tagger returns the identifier of the active unit.) The event handler returns the target cell (in which the target unit is to be localized) together with the candidate event time. The out-state request is accompanied by the branch of the target unit (if it exists), and the out-state computation is in analogy with the case studied in Section 5.1.2.

The configuration file features the `coulomb_cell_veto` tag together with the `coulomb_nearby`, `coulomb_surplus`, `cell_boundary`, `sampling`, `end_of_chain`, `start_of_run`, and `end_of_run` tags. The configuration file reproduces published data (see Fig. 14,④).

To adapt the configuration file for N point masses, the number of root nodes must be set to N in the `[RandomInputHandler]` section. The number of event handlers for the `coulomb_nearby` and `coulomb_surplus` events might have to be increased. However, a single cell-veto event handler realizes any number of factors with cell-based target particles whereas in Section 5.1.2 each of them required its own event handler.

5.2. Interacting dipoles

The configuration files in the `dipoles` directory of JF-V1.0 implement the ECMC sampling of the Boltzmann distribution for two identical finite-size dipoles, a model that was introduced previously [21]. Point masses in different dipoles interact



method	$\pi(\lvert\mathbf{r}_{13}\rvert < 0.22)$	$\pi(\lvert\mathbf{r}_{14}\rvert < 0.22)$
①	0.12128 ± 0.00003	0.18453 ± 0.00003
②	0.12129 ± 0.00003	0.18458 ± 0.00004
③	0.12129 ± 0.00003	0.18456 ± 0.00005
④	0.12108 ± 0.00037	0.18412 ± 0.00053
⑤	0.12130 ± 0.00002	0.18457 ± 0.00003

Fig. 15. Cumulative histogram of the pair separation $|\mathbf{r}_{13}|$ and $|\mathbf{r}_{14}|$ (nearest image) for two dipoles (see the inset) in a periodic three-dimensional cubic simulation box with periodic boundary conditions ($\beta c_i c_j = \pm 1$, $L = 1$). ①: Reversible Markov-chain Monte Carlo (see [21, Fig. 11]) ②: Method of Section 5.2.1 ③: Method of Section 5.2.2 ④: Method of Section 5.2.3 ⑤: Method of Section 5.2.4, each with standard errors for $\pi(|\mathbf{r}_{13}| < 0.22)$ and $\pi(|\mathbf{r}_{14}| < 0.22)$.

via the merged-image Coulomb potential (pairs 1–3, 1–4, 2–3, 2–4 in Fig. 15). Point masses within each dipole interact with a short-range potential (pairs 1–2 and 3-4). A repulsive short-range potential between oppositely charged atoms in different dipoles counterbalances the attractive Coulomb potential at small distances (pairs 1–4 and 2–3).

Each dipole is a composite point object made up of two oppositely charged point masses. It is represented as a tree with one root node that has two children. The number of root nodes in the system is set in the `[RandomInputHandler]` section of the configuration file, where the dipoles are created randomly through the `fill_root_node` method in the `DipoleRandomNodeCreator` class. In the `setting` package, the input handler specifies that there are two root nodes (`number_of_root_nodes` = 2). Each of them contains two nodes (which is coded as `number_of_nodes_per_root_node` = 2) and the number of node levels is two (`number_of_node_levels` = 2). As these numbers are set in the `setting` package, all the JF modules can autonomously construct all possible particle identifiers.

Statistically equivalent output is obtained for pair factors for all interactions (Section 5.2.1), for dipole–dipole Coulomb factors and their factor potential associated with a cell-based bounding potential (Section 5.2.2), for dipole–dipole Coulomb factors with the cell-veto algorithm (Section 5.2.3), and by alternating between concurrent moves of the entire dipoles with moves of the individual point masses (Section 5.2.4). The latter example showcases the collective-motion possibilities of ECMC integrated into JF. All configuration files here implement the short-range potential as an instance of the `DisplacedEvenPowerPotential` class with `power` = 2 and the repulsive short-range potential as an instance of the `InversePowerPotential` class with `power` = 6.

5.2.1. Atomic Coulomb factors

The configuration file `dipoles/atom_factors.ini` implements for each concerned pair of point masses a Coulomb pair factor, with the merged-image Coulomb potential associated with the inverse-power Coulomb bounding potential. Several event handlers that are instances of the same class realize these factors, and the number of event handlers must scale with their number. No internal state is declared. Pair factors are implemented for each pair of point masses that interact with a harmonic or a repulsive potential. One of the four point masses is active at each time, and it moves either in the $+x$, $+y$, or $+z$ direction. The configuration file represents composite point objects as trees with two levels (see Section 1.2). Positions and velocities are kept consistent on both levels, although the root-unit properties are not made use of. The tree structure only serves to identify leaf units on the same dipole.

In the configuration file, taggers and tags are listed in the `[TagActivator]` section. The `coulomb`, `harmonic`, and `repulsive` taggers are separate instances of the same `FactorType` `MapInStateTagger` class, and the corresponding sections set up the corresponding event handlers. Both the `harmonic` and the `repulsive` event handlers are instances of the `TwoLeafUnitEventHandler` class. Aliasing nevertheless assures a tree-structured configuration file (the `harmonic` tagger is for example declared with a `HarmonicEventHandler` class which is an alias for the `TwoLeafUnitEventHandler` class). The `coulomb` tagger and its event handlers are treated as in Section 5.1.1.

The sampling, start-of-run, end-of-run and end-of-chain pseudo-factors are realized by event handlers that are set up in the same way as in all other configuration files. However, the parent sections differ: the parent of the `[InitialChainStartOfRunEventHandler]` section sets the `start_of_run` tagger, which specifies that after the `start_of_run` event, new `coulomb`, `harmonic`, `repulsive`, `sampling`, `end_of_chain`, and `end_of_run` event handlers must be activated. The tag lists thus differ from those of the `[StartOfRun]` section in other configuration files. The configuration file `dipoles/atom_factors.ini` reproduces published data (see Fig. 15,②).

5.2.2. Molecular Coulomb factors, cell-based bounding potential

The configuration file `dipoles/cell_bounded.ini` implements for each pair of dipoles a Coulomb four-body factor. (The sum of the merged-image Coulomb potentials for pairs 1–3, 1–4, 2–3, 2–4 in Fig. 15 constitutes the Coulomb factor potential.) The event rates for such factors decay much faster with distance than for Coulomb pair factors, and the chosen lifting scheme considerably influences the dynamics (see [21, Sect. IV]). The configuration file installs a cell-occupancy internal state on the dipole level (rather than for the point masses). A cell-bounded event handler then realizes a Coulomb four-body factor with its factor potential associated with an orientation-independent cell-based bounding potential for dipole pairs that are not in excluded cells relative to each other. The configuration file furthermore implements pair factors for the harmonic and the repulsive interactions. One of the four point masses is active at each time, and it moves either in the $+x$, $+y$, or $+z$ direction.

The configuration file's `[TagActivator]` section defines all taggers and their corresponding tags. Among the taggers for event handlers realizing the Coulomb four-body factor, the `coulomb_cell_bounding` tagger differs markedly from the set-up in Section 5.1.2, as the event handler[10] is for a pair of composite point objects. The lifting scheme is set to `inside_first`

[10] Set in the `[TwoCompositeObjectCellBoundingPotentialEventHandler]` section.

_lifting. The bounding potential is defined in the [CellBoundingPotential] section. A dipole Monte Carlo estimator is used for simplicity (see Section 4.6). As it obtains an upper bound for the event rate from random trials for each relative cell orientations, its use is restricted to there being only a small number of cells. The coulomb_nearby and coulomb_surplus taggers are for event handlers realizing the Coulomb four-body factor when the bounding potential cannot be used. In this case, the merged-image Coulomb potential is summed not only for the factor potential, but also for the bounding potential.[11] The standard sampling, end_of_chain, end_of_run, and start_of_run taggers as well as the ones responsible for the harmonic and repulsive potentials are set up in a similar way as in Section 5.2.1.

The [TagActivator] section defines the internal state that is used by the coulomb_cell_bounding, coulomb_nearby, and coulomb_surplus taggers. The [SingleActiveCellOccupancy] section specifies the cell level (cell_level = 1 indicates that the particle identifiers have length one, corresponding to root nodes, rather than length two, which would correspond to the dipoles' leaf nodes). Positions and velocities must thus be kept consistent on both levels. The cell-occupancy system requires the presence of a cell_boundary event handler, again on the level of the root nodes. This event handler is aware of the cell level, and it ensures consistency of the events triggered by the cell-based bounding potential with the underlying cell system. The configuration file dipoles/cell_bounded.ini reproduces published data (see Fig. 15,③).

5.2.3. Molecular coulomb factors, cell-veto

The configuration file dipoles/cell_veto.ini implements the same factors and pseudo-factors and the same internal state as the configuration file of Section 5.2.2. A single cell-veto event handler then realizes the set of factors that relate to cells that are not excluded for any number of cell-based particles, whereas in the earlier implementation, the number of cell-bounded event handlers must exceed the possible number of particles in non-excluded cells of the active cell. This is what allows to implement ECMC with a complexity of $\mathcal{O}(1)$ per event.

The configuration file resembles that of Section 5.2.2. It mainly replaces the latter file's coulomb_cell_bounding event handlers with a coulomb_cell_veto event handler. Slight differences reflect the fact that a cell-veto event handler uses no displacement method of the bounding potential but obtains the displacement from the total event rate (see the discussion in Section 3.1.3). The configuration file dipoles/cell_veto.ini reproduces published data (see Fig. 15,④).

5.2.4. Atomic Coulomb factors, alternating root mode and leaf mode

The configuration file dipoles/dipole_motion.ini implements two different modes. In leaf mode, at each time one of the four point masses is active, and it moves either in the $+x$, $+y$, or $+z$ direction (see Fig. 16a). In root mode, at each time the point masses of one dipole moves as a rigid block, in the same direction (see Fig. 16b). (The root mode, by itself, does not assure irreducibility of the Markov-chain algorithm, as the orientation and shape of any dipole molecule would remain unchanged throughout the run.)

JF-V1.0 represents the dipoles as trees, and both modes are easily implemented. In leaf mode, the Coulomb factors are realized by coulomb_leaf event handlers that are instances of the same class[12] as the coulomb_nearby event handlers in Sections 5.2.2 and 5.2.3. The root mode, in turn, is patterned after the simulation of two point masses (as in Section 5.1.1): all inner-dipole potentials are constant. The inter-dipole Coulomb potentials sum up to an effective two-body potential, the factor potential of a two-body factor realized in a Coulomb-dipole event handler. The repulsive short-range potential between oppositely charged atoms in different dipoles also translates into a potential between the dipoles in rigid motion, and serves as a factor potential of a two-body factor, realized in a specific event handler.

Taggers and their tags are listed in the [TagActivator] section. The harmonic_leaf, repulsive_leaf (leaf-mode) taggers, as well as all those related to event handlers that realize pseudo-factors are as in Section 5.2.1. The coulomb_leaf tagger corresponds to the coulomb_nearby tagger in Section 5.2.2. The coulomb_root and repulsive_root taggers are analogous to those in Section 5.1.1 for the two-atom case.

As all other operations that take place in JF, the switches between leaf mode and root mode are also formulated as events. They are related to two pseudo-factors and realized by a leaf_to_root event handler and by a root_to_leaf event handler, respectively. (These two event handlers are aliases for instances of the RootLeafUnitActiveSwitcher class.) The root_to_leaf and leaf_to_root taggers, in addition to the create and trash lists, set up separate activate and deactivate lists (see Section 2.4). The configuration file reproduces published data (see Fig. 15,⑤). Of particular interest is that the tree representation of composite point objects preserves consistency between leaf-node units and root-node units: the event handlers return branches of cnodes for all independent units (see Fig. 7) whose unit information can be integrated into the global state.

5.3. Interacting water molecules (SPC/Fw model)

The configuration files in the water directory implement the ECMC sampling of the Boltzmann distribution for two water molecules, using the SPC/Fw model that was previously studied with ECMC [21]. Molecules are represented as composite point objects with three charged point masses, one of which is positively charged (representing the oxygen) and the two others are negatively charged (representing the hydrogens). Point masses in different water molecules interact *via* the merged-image Coulomb potential. In addition, point masses within each molecule interact with a three-body bending interaction, and a harmonic oxygen–hydrogen potential. Finally, any two oxygens interact through a Lennard-Jones potential [21].

In the tree state handler (defined in the [TreeStateHandler] section, a child of the [SingleProcessMediator] section), water molecules are represented as trees with a root node and three children (the leaf nodes of the tree). The total number of water molecules (that is, of root nodes) is set in the [RandomInputHandler] section of each configuration file. The molecules are created through the fill_root_node method in the WaterRandomNodeCreator class. There are two node levels (number_of_node_levels = 2) and three nodes per root node (number_of_nodes_per_root_node = 3). The charges of a molecule are set in the [ElectricChargeValues] section (a descendant of the [WaterRandomNodeCreator] section).

All the configuration files in the water directory of JF-V1.0 implement the pair harmonic factors that are realized through harmonic event handlers. The corresponding taggers are defined in the [Harmonic] sections, with the displaced even-power potential and its parameters set in the [HarmonicEventHandler] and [HarmonicPotential] sections. The configuration files furthermore implement the taggers corresponding to the three-body bending factors in their [Bending] sections. The bending event handler has three independent units (attached to

[11] The tree structure of the configuration file is hidden in this case, as the JF factory (which builds instances of classes based on its content) creates separate instances for all the descendants of a section, not requiring the use of aliases.

[12] Instances of the TwoCompositeObjectSummedBoundingPotentialEventHandler class.

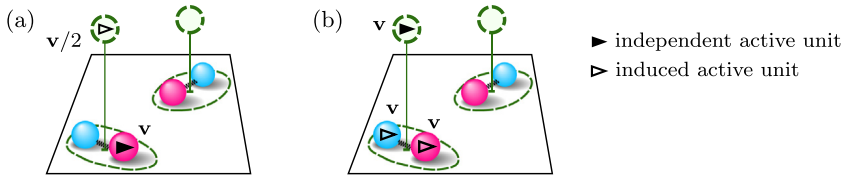
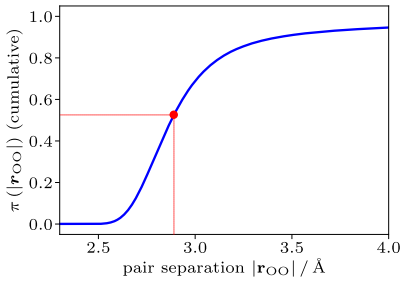


Fig. 16. Two moves implemented in `dipoles/dipole_motion.ini`. (a) In leaf mode, a single independent active leaf unit has velocity **v**. The corresponding dipole center (the active root unit) is induced to move at **v**/2. (b) In root mode, one dipole (independent active root unit) has velocity **v**, and both its active leaf units have induced velocity **v**.



method	$\pi(\lvert\mathbf{r}_{OO}\rvert < 2.9\,\text{Å})$
①	0.5382 ± 0.0003
②	0.5385 ± 0.0003
③	0.5390 ± 0.0002
④	0.5400 ± 0.0010
⑤	0.5398 ± 0.0010

Fig. 17. Cumulative histogram of the oxygen–oxygen pair separation $|\mathbf{r}_{OO}|$ for two SPC/Fw water molecules in a periodic cubic simulation box. ①: Reversible Markov-chain Monte Carlo (see [21, Fig. 14]) ②: Method of Section 5.3.1 ③: Method of Section 5.3.2 ④: Method of Section 5.3.3 ⑤: Method of Section 5.3.4, each with standard errors for $\pi(|\mathbf{r}_{OO}| < 2.9\text{Å})$.

branches). It thus requires a lifting scheme (which is chosen in the `[BendingEventHandler]` section), which is, however, unique (see [21, Fig. 2]). In all these configuration files, one of the six point masses is active, and it moves either in the $+x$, $+y$, or $+z$ direction (the optional rigid displacement of the entire water molecule, could be set up as in Section 5.2.4).

Statistically equivalent output is obtained for a simple set-up featuring pair factors for the Coulomb potential and a Lennard-Jones interaction that is inverted (Section 5.3.1), or for a molecular-factor Coulomb potential associated with a power-law bounding potential and a cell-based Lennard-Jones bounding potential (Section 5.3.2). In addition, the cell-veto algorithm for the Coulomb potential coupled to an inverted Lennard-Jones potential (Section 5.3.3) is also provided. Finally, cell-veto event handlers take part in the realization of complex molecular Coulomb factors and also realize Lennard-Jones factors between oxygens (Section 5.3.4). This illustrates how multiple independent cell-occupancy systems may coexist within the same run.

5.3.1. Atomic Coulomb factors, Lennard-Jones inverted

The configuration file `water/coulomb_power_bounded_lj_inverted.ini` implements pair Lennard-Jones, harmonic and Coulomb factors. The Coulomb factors are realized for any distance of the point masses by event handlers that associate the merged-image Coulomb potential with its inverse-power Coulomb bounding potential. The Lennard-Jones potential is inverted. This configuration file needs no internal state.

In the configuration file, the `[TagActivator]` section lists all the taggers together with their tags, which in addition to the taggers related to pseudo-factors, are reduced to `coulomb`, `harmonic`, `bending`, and `lennard_jones`. The merged-image Coulomb potential, with its associated power-law bounding potential (both for attractive and repulsive charge products), is specified in the `[Coulomb]` section of the configuration file. The Lennard-Jones potential is invertible and its `displacement` method is used rather than that of a bounding potential. The output handler is defined in the `[OxygenOxygenSeparationOutputHandler]` section, a child of the `[InputOutputHandler]` section. It obtains all the units, extracts the oxygens through their unit identifier, and records the oxygen–oxygen separation distance. This reproduces published data (see Fig. 17,②).

5.3.2. Molecular Coulomb factors, Lennard-Jones cell-bounded

The configuration file `water/coulomb_power_bounded_lj_cell_bounded.ini` for the water system corresponds to pair factors for the Lennard-Jones and the harmonic potentials and to molecular factors for the Coulomb interaction. The Coulomb factor potential is the sum of the merged-image Coulomb potential for the nine relevant pairs of point masses (pairs across two molecules). It is realized in a particular event handler,[13] analogously to how this is done for the Coulomb interaction in Sections 5.2.2 and 5.2.3. The associated bounding potential (both for attractive and repulsive charge combinations) is given by the sum over all the individual pairs. Although the Lennard-Jones interaction can be inverted, the configuration file sets up a cell-occupancy internal state that tracks the identifiers for the oxygens. As in previous cases, this leads to three types of events, corresponding to the nearby, surplus, and cell-based particles, in addition to cell-boundary events.

Taggers and their tags are listed in the `[TagActivator]` section. Taggers are generally utilized as in other configuration files. The internal state is specified in the `[TagActivator]` section. As set up in the `[SingleActiveCellOccupancy]` section, it features an `oxygen_indicator` charge (set in the `[OxygenIndicator]` section). The oxygen-indicator charge is nonzero only for the oxygens. In consequence, the oxygen cell system (defined in the `[OxygenCell]` section) tracks only oxygens. This reproduces published data (see Fig. 17,③).

5.3.3. Molecular Coulomb cell-veto, Lennard-Jones inverted

The configuration file `water/coulomb_cell_veto_lj_inverted.ini` for the water system corresponds to the same factors as in Section 5.3.2. As a preliminary step towards the treatment of all long-range interactions with the cell-veto algorithm, in Section 5.3.4, molecular Coulomb factors are realized here (for non-excluded cells of the active cell) with a cell-veto event handler.

Taggers and their tags are listed in the `[TagActivator]` section, and they are generally similar to those of other configuration files. In addition, the internal state for the Coulomb system is defined in the `[TagActivator]` section and further described in the `[SingleActiveCellOccupancy]` section. The latter describes the cell level (which serves for the water molecules) as on the root node level (`cell_level` = 1), the barycenter of the leaf-node positions of each water molecule. (Root-node and leaf-node positions are set in the random input handler, which itself uses a water random node creator.)

The event handlers consistently update all leaf-node positions and root-node positions from a valid initial configuration obtained in an instance of the `WaterRandomNodeCreator` class.

[13] An instance of the `TwoCompositeObjectSummedBoundingPotentialEventHandler` class.

Consistency will be deteriorated over long runs, but this is of little importance for the simple example case presented here. The configuration file reproduces published data (see Fig. 17,④).

5.3.4. Molecular Coulomb cell-veto, Lennard-Jones cell-veto

The configuration file `water/coulomb_cell_veto_lj_cell_veto.ini` offers no new factors compared to Sections 5.3.2 and 5.3.3, but it uses, for illustrative purposes, two cell-occupancy systems and two cell-veto event handlers. As nearby and surplus particles are excluded from the cell-veto treatment, this implies two sets of cell-veto, nearby, and surplus event handlers in addition to two cell-boundary event handlers. For the molecular Coulomb factors, the cell-veto event handler receives as a factor potential the sum of pairwise merged-image Coulomb potentials with attractive and repulsive charge combinations. The corresponding cell-occupancy system tracks the barycenter of individual water molecules, and consistency between root-node units and leaf-node units is of importance. Although the Lennard-Jones potential can be inverted, the configuration file sets up a second cell-occupancy system for the Lennard-Jones potential. The cell-occupancy system tracks only leaf-node particles that correspond to oxygen atoms.

Taggers and their tags are listed in the `[TagActivator]` section. This section is of interest as it sets up the internal state as two cell-occupancy systems, both instances of the same `SingleActiveCellOccupancy` class. They require different parameters, and are therefore presented under aliases, in the `[OxygenCell]` and `[MoleculeCell]` sections. Each of these cell-occupancy systems uses a separate cell system instance of the same class. As the two cell systems have the same parameters, they do not need to be aliased in the configuration file. The configuration file reproduces published data (see Fig. 17,⑤).

6. License, GitHub repository, Python version

JF, the Python application described in this paper, is an open-source software project that grants users the rights to study and execute, modify and distribute the code. Modifications can be fed back into the project.

6.1. License information, used software

JF is made available under the GNU GPLv3 license (for details see the JF `LICENSE` file). The use of the Python `MDAnalysis` package [33,34] for reading and writing `.pdb` files, of the Python `Dill` package [35,36] for dumping and restarting a run of the application, and of the Python `Matplotlib` [37] and `NumPy` [38,39] packages for the graphical analysis of output is acknowledged.

6.2. GitHub repository

`JeLLyFysh`, the public repository for all the codes and the documentation of the application, is part of a public GitHub organization.[14] The repository can be forked (that is, copied to an outside user's own public repository) and from there studied, modified and run in the user's local environment. Users may contribute to the JF application *via* pull requests (see the JF `README.md` and `CONTRIBUTING.md` files for instructions and guidelines). All communication (bug reports, suggestions) take place through GitHub "Issues", that can be opened in the repository by any user or contributor, and that are classified in GitHub projects on `JeLLyFysh`.

[14] The organization's url is https://github.com/jellyfysh.

6.3. Python version, coding conventions

JF-V1.0 is compatible with Python 3.5 (and higher) and with PyPy 7 (and higher), a just-in-time compiling Python alternative to interpreted CPython (see the JF documentation for details). JF code adheres to the PEP8 style guide for Python code, except for the linewidth that is set to 120 (see the `CONTRIBUTING.md` file for details).

Following the PEP8 Python naming convention, JF modules and packages are spelled in snake case and classes in camel case (the `state_handler` module thus contains the `StateHandler` class). In configuration files, section titles are in camel case and enclosed in square brackets (see Fig. 13).

Versioning of the JF project adopts two-to-four-field version numbers defined as Milestone.Feature.AddOn.Patch. Version 1.0, as described, represents the first development milestone which reproduces published data [21]. Patches and bugfixes of this version will be given number 1.0.0.1, 1.0.0.2, etc. (Finer-grained distinction between versions is obtained through the hashes of master-branch GitHub commits.) New configuration files and required extensions are expected to lead to versions 1.0.1, 1.0.2, etc. Version 1.1 is expected to fully implement different dimensions and arbitrary rectangular and cuboid shapes of the JF `potential` package. Versions 1.2 and 1.3 will consistently implement $n_{\mathrm{ac}} \geq 1$ independent active particles (on a single processor) and eliminate unnecessary time-slicing for some events triggered by pseudo-factors and for unconfirmed events. All development from Versions 1.0 to 2.0 can be undertaken concurrently. Fully parallel code is planned for Version 3.0. In JF development, two-field versions (2.0, 3.0, etc.) may introduce incompatible code, while three- and four-field version numbers are intended to be backward compatible.

7. Conclusions, outlook

As presented in this paper, JF is a computer application for ECMC simulations that we hope will be useful for researchers in different fields of computational science. The JF-V1.0 constitutes its first development milestone: built on the mediator design pattern, it systematically formulates the entire ECMC time evolution in terms of events, from the start-of-run to the end-of-run, including sampling, restarts (that is, end-of-chain), and the factor events. A number of configuration files validate JF-V1.0 against published test cases for long-range interacting systems [21].

For JF-V1.0, consistency has been the main concern, and code has not yet been optimized. Also, the handling of exceptions remains rudimentary, although this is not a problem for the cookbook examples of Section 5.

All the methods are written in Python. Considerable speed-up can certainly be obtained by rewriting time-consuming parts of the application in compiled languages, in particular of the `potential` package. One of the principal limitations of JF-V1.0 is that pseudo-factor-related and unconfirmed events are time-sliced, leading to superfluous trashing and re-activation of candidate events. Optimized bounding potentials for many-particle factor potentials also appear as a priority.

The consistent implementation of an arbitrary number n_{ac} of simultaneously active particles is straightforward, although it has also not been implemented fully in JF-V1.0. (As mentioned, this is planned for JF (Version 2.0)). This will enable full parallel implementations on multiprocessor machines. Simplified parallel implementations for one-dimensional systems and for hard-disk models in two dimensions are currently being prototyped. The parallel computation of candidate events (using the `MultiProcessMediator` class implemented in JF-V1.0) is at present rather slow. Bringing the full power of parallelization and of multi-process ECMC to real-world applications appears as its outstanding challenge for JF.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

P.H. acknowledges support from the Bonn-Cologne Graduate School of Physics and Astronomy honors branch, Germany and from Institut Philippe Meyer. L.Q. and M.F.F. acknowledge hospitality at the Max-Planck-Institut für Physik komplexer Systeme, Dresden, Germany. M.F.F. acknowledges financial support from EPSRC fellowship EP/P033830/1 and hospitality at Ecole normale supérieure. W.K. acknowledges support from the Alexander von Humboldt Foundation. All program files are freely available at http://dx.doi.org/10.17632/srrjt9493d.1, including the configuration files with which all figures can be reproduced.

References

[1] E.P. Bernard, W. Krauth, D.B. Wilson, Phys. Rev. E 80 (2009) 056704, http://dx.doi.org/10.1103/PhysRevE.80.056704, arXiv:0903.2954.
[2] M. Michel, S.C. Kapfer, W. Krauth, J. Chem. Phys. 140 (5) (2014) 054116, http://dx.doi.org/10.1063/1.4863991, arXiv:1309.7748.
[3] Y. Nishikawa, M. Michel, W. Krauth, K. Hukushima, Phys. Rev. E 92 (6) (2015) 063306, http://dx.doi.org/10.1103/PhysRevE.92.063306, arXiv:1508.05661.
[4] S.C. Kapfer, W. Krauth, Phys. Rev. Lett. 119 (2017) 240603, http://dx.doi.org/10.1103/PhysRevLett.119.240603, arXiv:1705.06689.
[5] Z. Lei, W. Krauth, Europhys. Lett. 121 (2018) 10008, http://dx.doi.org/10.1209/0295-5075/121/10008, arXiv:1711.08375.
[6] Z. Lei, W. Krauth, Europhys. Lett. 124 (2) (2018) 20003, http://dx.doi.org/10.1209/0295-5075/124/20003, arXiv:1806.06786.
[7] Z. Lei, W. Krauth, A.C. Maggs, Phys. Rev. E 99 (4) (2019) 043301, http://dx.doi.org/10.1103/PhysRevE.99.043301, arXiv:1812.02494.
[8] E.P. Bernard, W. Krauth, Phys. Rev. Lett. 107 (2011) 155704, http://dx.doi.org/10.1103/PhysRevLett.107.155704, arXiv:1102.4094.
[9] S.C. Kapfer, W. Krauth, Phys. Rev. Lett. 114 (2015) 035702, http://dx.doi.org/10.1103/PhysRevLett.114.035702, arXiv:1406.7224.
[10] J. Bierkens, A. Bouchard-Côté, A. Doucet, A.B. Duncan, P. Fearnhead, T. Lienart, G. Roberts, S.J. Vollmer, Statist. Probab. Lett. 136 (2018) 148–154, http://dx.doi.org/10.1016/j.spl.2018.02.021, arXiv:1701.04244.
[11] B.J. Alder, T.E. Wainwright, J. Chem. Phys. 27 (1957) 1208–1209, http://dx.doi.org/10.1063/1.1743957.
[12] B.J. Alder, T.E. Wainwright, J. Chem. Phys. 31 (1959) 459–466, http://dx.doi.org/10.1063/1.1730376.
[13] M.N. Bannerman, L. Lue, J. Chem. Phys. 133 (12) (2010) 124506, http://dx.doi.org/10.1063/1.3486567.
[14] E.A.J.F. Peters, G. de With, Phys. Rev. E 85 (2012) 026703, http://dx.doi.org/10.1103/PhysRevE.85.026703, arXiv:1112.1263.
[15] F. Ding, D. Tsao, H. Nie, N.V. Dokholyan, Structure 16 (7) (2008) 1010–1018, http://dx.doi.org/10.1016/j.str.2008.03.013.
[16] D. Shirvanyants, F. Ding, D. Tsao, S. Ramachandran, N.V. Dokholyan, J. Phys. Chem. B 116 (29) (2012) 8375–8382, http://dx.doi.org/10.1021/jp2114576.
[17] E.A. Proctor, F. Ding, N.V. Dokholyan, Wiley Interdiscip. Rev.: Comput. Mol. Sci. 1 (1) (2011) 80–92, http://dx.doi.org/10.1002/wcms.4.
[18] E.A. Proctor, N.V. Dokholyan, Curr. Opin. Struct. Biol. 37 (2016) 9–13, http://dx.doi.org/10.1016/j.sbi.2015.11.001.
[19] S.C. Kapfer, W. Krauth, Phys. Rev. E 94 (2016) 031302, http://dx.doi.org/10.1103/PhysRevE.94.031302, arXiv:1606.06780.
[20] Y. Wu, H.L. Tepper, G.A. Voth, J. Chem. Phys. 124 (2) (2006) 024503, http://dx.doi.org/10.1063/1.2136877.
[21] M.F. Faulkner, L. Qin, A.C. Maggs, W. Krauth, J. Chem. Phys. 149 (6) (2018) 064113, http://dx.doi.org/10.1063/1.5036638, arXiv:1804.05795.
[22] L. Qin, P. Höllmer, A.C. Maggs, W. Krauth, Benchmarking ECMC against lammps, 2020, manuscript in preparation.
[23] M. Hasenbusch, S. Schaefer, Phys. Rev. D 98 (2018) 054502, http://dx.doi.org/10.1103/PhysRevD.98.054502, arXiv:1806.11460.
[24] E. Gamma, R. Helm, R. Johnson, J.M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, first ed., Addison-Wesley Professional, 1994.
[25] P. Diaconis, S. Holmes, R.M. Neal, Ann. Appl. Probab. 10 (2000) 726–752, http://dx.doi.org/10.1214/aoap/1019487508.
[26] J. Harland, M. Michel, T.A. Kampmann, J. Kierfeld, Europhys. Lett. 117 (3) (2017) 30001, http://dx.doi.org/10.1209/0295-5075/117/30001, arXiv:1611.09098.
[27] S. Miller, S. Luding, J. Comput. Phys. 193 (1) (2003) 306–316, http://dx.doi.org/10.1016/j.jcp.2003.08.009, arXiv:physics/0302002.
[28] M.C. Herbordt, M.A. Khan, T. Dean, 2009 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2009, pp. 129–136, http://dx.doi.org/10.1109/ASAP.2009.39.
[29] M.A. Khan, M.C. Herbordt, J. Comput. Phys. 230 (17) (2011) 6563–6582, http://dx.doi.org/10.1016/j.jcp.2011.05.001.
[30] S.C. Kapfer, W. Krauth, J. Phys. Conf. Ser. 454 (1) (2013) 012031, http://dx.doi.org/10.1088/1742-6596/454/1/012031, arXiv:1301.4901.
[31] A.J. Walker, ACM Trans. Math. Software 3 (3) (1977) 253–256, http://dx.doi.org/10.1145/355744.355749.
[32] S.W. de Leeuw, J.W. Perram, E.R. Smith, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 373 (1752) (1980) 27–56, http://dx.doi.org/10.1098/rspa.1980.0135.
[33] N. Michaud-Agrawal, E.J. Denning, T.B. Woolf, O. Beckstein, J. Comput. Chem. 32 (10) (2011) 2319–2327, http://dx.doi.org/10.1002/jcc.21787.
[34] R.J. Gowers, M. Linke, J. Barnoud, T.J.E. Reddy, M.N. Melo, S.L. Seyler, J. Domaski, D.L. Dotson, S. Buchoux, I.M. Kenney, O. Beckstein, in: S. Benthall, S. Rostrup (Eds.), Proceedings of the 15th Python in Science Conference, 2016, pp. 98–105, http://dx.doi.org/10.25080/Majora-629e541a-00e.
[35] M.M. McKerns, M.A. Aivazis, Pathos: a framework for heterogeneous computing, 2010, URL http://trac.mystic.cacr.caltech.edu/project/pathos.
[36] M.M. McKerns, L. Strand, T. Sullivan, A. Fang, M.A. Aivazis, Building a framework for predictive science, in: S. van der Walt, J. Millman (Eds.), Proceedings of the 10th Python in Science Conference, 2011, pp. 67–78,
[37] J.D. Hunter, Comput. Sci. Eng. 9 (3) (2007) 90–95, http://dx.doi.org/10.1109/mcse.2007.55.
[38] T.E. Oliphant, A Guide to NumPy, Trelgol Publishing, USA, 2006, URL https://web.mit.edu/dvp/Public/numpybook.pdf.
[39] S. van der Walt, S.C. Colbert, G. Varoquaux, Comput. Sci. Eng. 13 (2) (2011) 22–30, http://dx.doi.org/10.1109/MCSE.2011.37, arXiv:1102.1523v1.

Appendix C

Publication III

Fast sequential Markov chains

Liang Qin, Philipp Höllmer, and Werner Krauth
arXiv:2007.15615 (2020)

This paper is the published result that we obtained in chapter 7. Usually, the naive implementation of ECMC relaxes particles along orthogonal coordinate axes. Here, we propose a scheme with gradual direction changes in order for fast mixing of the dipole angle. A simple dipole with hard potential on a plane is selected for testing purposes. We find that, in contrast to traditional Monte Carlo methods, the sequential Monte Carlo method exhibits distinct patterns in the evolution of the dipole angle, though the estimate of the angle change remains zero. Further tests show that the efficiency of this scheme agrees with the number of elements of the direction set, and the sequential order is always preferable over the random choice from the set.

Fast sequential Markov chains

Liang Qin,[1,*] Philipp Höllmer,[2,†] and Werner Krauth[1,‡]

[1] *Laboratoire de Physique de l'Ecole normale supérieure, ENS, Université PSL, CNRS, Sorbonne Université, Université Paris-Diderot, Sorbonne Paris Cité, Paris, France*

[2] *Bethe Center for Theoretical Physics, University of Bonn, Nussallee 12, 53115 Bonn, Germany*

(Dated: August 7, 2020)

We discuss non-reversible Markov chains that generalize the sweeps commonly used in particle systems and spin models towards a sequential choice from a set of directions of motion. For a simplified dipole model, we show that direction sweeps leave the stationary probability distribution unchanged, but profoundly modify the trajectory of the Markov chain. Choosing a larger direction set can lead to much shorter mixing times. The sequential order is faster than the random sampling from the set. We discuss possible applications of sequential Monte Carlo in polymer physics and molecular simulation.

Since its introduction in 1953, the Markov-chain Monte Carlo (MCMC) method [1] has developed into an essential tool in science and engineering as well as into a ramified mathematical research discipline. [2] The concept of sampling is central to the method. Sampling may consist in generating (representative) integrands or addends to approximate a high-dimensional integral or a sum. Sampling may also refer to the evaluation of representative Feynman diagrams in order to approximate, with purely statistical errors, a perturbative series in quantum field theory. [3, 4] The randomness in MCMC, its trademark property, corresponds to samples j at Monte-Carlo time step t being produced from samples i at time step $t-1$ with the independent probabilities contained in a transition matrix $P = P(i,j)$. The stationary distribution π of samples satisfies the global-balance condition $\pi P = \pi$.

The randomness of the moves may often be reduced without destroying the convergence to π. This was understood for particle systems in the original 1953 reference where, instead of moving randomly sampled particles, "... we move each of the particles in succession ..." (see [1, p.22]). The sequential MCMC algorithm for particle systems is non-reversible, [5, 6] as it violates the detailed-balance condition $\pi_i P(i,j) = P(j,i)\pi_j$ that characterizes reversible Markov chains. However, it still satisfies the global-balance condition that is a prerequisite for convergence towards π. The sequential algorithm is somewhat more efficient. [7–9] In the Ising model and related systems, sequential spin updates (so called "sweeps") also break detailed balance, and they also lead to moderate speedups compared to the detailed-balance chains that flip spins randomly. [10] In the event-chain Monte Carlo (ECMC) algorithm, [11, 12] randomness is drastically reduced in that a particle move, if accepted, is identically repeated for the same particle. If the move is rejected for a given particle, it is identically repeated for the particle that caused the rejection. Lifting variables [13] allow one to formulate the sequential Markov chains and, in particular, ECMC as time-independent transition matrices acting on an extended configuration space.

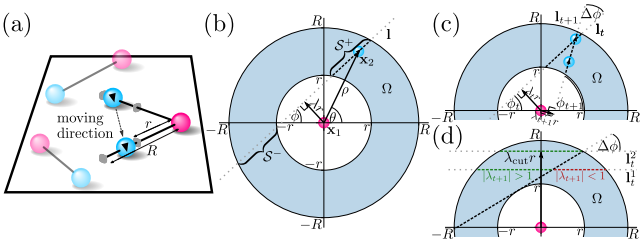


FIG. 1. Ring representation of dipole. (a): Dipole atom moving along a fixed direction. (b): Single dipole, with segments $\mathcal{S}^-$ and $\mathcal{S}^+$ for impact parameter $|\lambda| < 1$. (c): Intersection of $\mathbf{l}_t$ and $\mathbf{l}_{t+1}$ at $\mathbf{x}_2(t+1)$. (d): For $\phi_t = 0$: lines $\mathbf{l}_t$; impact parameter $\lambda_{\rm cut}$ for which all $\mathbf{x}_{t+1}$ satisfy $\lambda_{t+1} > 1$.

In the present paper, we reduce randomness in MCMC through a sequential selection of directions from a possibly large set $\mathcal{D}$. Specifically, we study a model of a two-dimensional simple dipole. For k steps, all moves of the two atoms composing these dipoles are in direction $\pm\phi$, after which all moves are in direction $\pm(\phi+\Delta\phi)$, etc. (Directions are parametrized by an angle with respect to a fixed axis, say, the x-axis.) We show the special interest of choosing small $\Delta\phi$, where, paradoxically, the mixing time of the algorithm can be much shortened. This sequential MCMC algorithm leaves the stationary distribution strictly unchanged. For a single dipole, the non-trivial random walk of the sequence of samples exhibits a statistical force that may accelerate ECMC simulations of long-range interacting N-body systems of extended elements, e.g., complex liquids made up of dipoles, such as water molecules (see [14, 15]). It may also apply to reversible or non-reversible MCMC simulations of polymers, dipolar systems or glass models.

For concreteness, we consider a two-dimensional dipole with atoms at $\mathbf{x}_1$ and $\mathbf{x}_2$ and a flat inner-dipole interaction

$$U(\rho) = \begin{cases} 0 & \text{if } \rho \in [r, R], \\ \infty & \text{otherwise,} \end{cases} \tag{1}$$

with $\rho = |\mathbf{x}_2 - \mathbf{x}_1|$. This dipole can be envisaged as part of a more complex system of many dipoles with hard-sphere interactions between any two atoms. Periodic boundary conditions are assumed throughout (see Fig. 1a). A standard reversible Metropolis sampling of this system would propose a random displacement of one of the two atoms in a way that satisfies detailed balance. The move would be rejected if it proposes a dipole extension outside the range allowed by eq. (1). The sampling can be easily extended to the presence of other dipoles with hard-sphere interactions between atoms. The ECMC algorithm for this system would move an atom in a given direction until $\rho = r$ or until $\rho = R$ and then move the other atom in the same direction (hard-sphere interactions with other dipoles are implemented easily).

With periodic boundary conditions, the configuration space for a single dipole is given by $T^2 \times \Omega$ with a uniform Euclidean measure, where T^2 is the two-dimensional torus, and Ω a two-dimensional ring of inner radius r and outer radius R. With the potential of eq. (1), the Boltzmann weight π factors out translations in T^2. We may thus set $\mathbf{x}_1 = (0, 0)$. The position of the atom 2 is then described by the dipole length ρ and the dipole angle θ (see Fig. 1b). The uniform Euclidean measure translates into a dipole-length distribution $\pi(\rho) = 2\rho/\left[r^2\left(\eta^2 - 1\right)\right]$ for $\rho \in [r, R]$ with $\eta = R/r$. The dipole angle θ is uniformly distributed in $(-\pi, \pi]$.

In our sequential Markov-chain algorithm, we consider a direction set $\mathcal{D} = \{\phi_0, \phi_1 ... \phi_{n-1}\}$, but propose them in a specific time order $t = 0, 1, 2, \ldots$, rather than to sample from this set at each t. Any direction set spanning $\mathbb{R}^2$ and any sequence taken from this set yields the same stationary distribution π as long as the choice of directions is independent of the configuration. Any continuous step-size distribution implies that the Markov chain converges towards π. For concreteness, we analyze the case where $\phi_{k+1} - \phi_k = \Delta\phi$ is constant, with positive $\Delta\phi$. Each value ϕ_l is repeated for k Monte Carlo moves, and we consider the limit $k \to \infty$, in which local equilibrium is reached. For simplicity of notation, we increment the Monte Carlo time t by one at the choice of a new direction (rather than to increment it by k). In the limit $k \to \infty$, the Metropolis algorithm (with an additional non-hopping condition that corresponds to small step size) and ECMC both realize a direct sample [16] under the constraint given by the direction of motion.

We now analyze the motion of the dipole for a given direction ϕ. Because of $\mathbf{x}_1 = (0, 0)$, this is equivalent to the move of atom 2 on a straight line. At time t, the dipole moves on the line $\mathbf{l}_t$ (which forms the angle ϕ_t) from $\mathbf{x}_2(t)$ to $\mathbf{x}_2(t+1)$ (see Fig. 1b). Because of the implicit limit $k \to \infty$, $\mathbf{x}_2(t+1)$ is a uniform random point on the segment of $\mathbf{l}_t \cap \Omega$ that also contains $\mathbf{x}_2(t)$. The nature of this move depends on the impact parameter:

$$\lambda_t = \sin(\theta_t - \phi_t)\rho_t/r, \tag{2}$$

in other words on the signed distance (in units of r) of $\mathbf{l}_t$ to the origin. (In a reference frame where $\phi = 0$, λ is positive for $\mathbf{x}_2$ in the upper half plane and negative in the lower half plane.)

If $|\lambda_t| > 1$, the line $\mathbf{l}_t$ forms a single segment $\mathcal{S}_t$ in Ω. In contrast, if $|\lambda_t| < 1$, there are two such segments, namely $\mathcal{S}_t^-$ (to the left, in the reference frame in which $\phi = 0$) and $\mathcal{S}_t^+$ (to the right). Then, if $|\lambda_{t+1}| < 1$ (in addition to $|\lambda_t| < 1$), the dipole is trapped in its segment from time t to $t+1$, and the impact parameter decreases if in $\mathcal{S}^+$ and increases if in $\mathcal{S}^-$. This trapping gives rise to a deterministic counter-rotation of the dipole angle with respect to the rotation of the directions, as we will describe later.

The evolution of the impact parameter λ can be described through drift and diffusion terms. The expectation (mean value) of λ_{t+1}, for a fixed value of λ_t, is

$$\mathbb{E}\,\lambda_{t+1} = \begin{cases} \lambda_t \cos\Delta\phi - A_t \sin\Delta\phi & \text{if } \mathbf{x}_2(t) \in \mathcal{S}^+ \,, \\ \lambda_t \cos\Delta\phi + A_t \sin\Delta\phi & \text{if } \mathbf{x}_2(t) \in \mathcal{S}^- \,, \\ \lambda_t \cos\Delta\phi & \text{if } |\lambda_t| > 1, \end{cases} \tag{3}$$

where $A_t = \frac{1}{2}(B_t + C_t)$ with $B_t = \sqrt{\eta^2 - \lambda_t^2}$ and $C_t = \sqrt{1 - \lambda_t^2}$. The fluctuation term (variance) of the impact parameter λ_{t+1} is

$$\sigma^2(\lambda_{t+1}) = \begin{cases} \frac{1}{12}\sin^2(\Delta\phi)\,(B_t - C_t)^2 & \text{if } |\lambda_t| < 1 \,, \\ \frac{1}{3}\sin^2(\Delta\phi)\,B_t^2 & \text{if } |\lambda_t| > 1 \,. \end{cases} \tag{4}$$

For small $|\Delta\phi|$, the drift $\mathbb{E}\,(\lambda_{t+1}) - \lambda_t$ (from eq. (3)) is proportional to $(\Delta\phi)^2$ for $|\lambda| > 1$, and proportional to $\Delta\phi$ for $|\lambda| < 1$. The fluctuation term of eq. (4) is always proportional to $(\Delta\phi)^2$. This can be compared to the discrete-time Langevin equation

$$\xi_{t+1} = \xi_t + D^{(1)}(\xi_t, t_t)\tau + \sqrt{D^{(2)}(\xi_t, t_t)\tau}\, w_t, \tag{5}$$

where $D^{(1)}$ and $D^{(2)}$ are Kramers–Moyal expansion coefficients, τ is the time step, and w_t are Gaussian random numbers (see [17, eq. (3.138)]). For $|\lambda_t| > 1$, the impact parameter performs what we can call an "excursion", a random walk in the quantity $\tau = (\Delta\phi)^2$. Such an excursion corresponds to a total number Δt of time steps (that is, changes of $\Delta\phi$) that scales as $\Delta t \sim \text{const}/(\Delta\phi)^2$ and a total angle that diverges as $\tilde{\phi}_t - \tilde{\phi}_0 = \Delta t \Delta\phi \propto 1/\Delta\phi$. ($\tilde{\phi}$ differs from ϕ in that it is rolled out, rather than folded into the interval $(-\pi, \pi]$.) During the excursion, the dipole rotates (for $\Delta\phi > 0$) in the positive sense (see inset of Fig. 2a). Thus, the difference in the rolled-out dipole angle $\tilde{\theta}$ during an excursion diverges as $\sim \text{const}/\Delta\phi$. This is evidenced by typical trajectories of $\tilde{\theta}_t \Delta\phi$ as a function of $\tilde{\phi}_t \Delta\phi$ for different values of $\Delta\phi$ (see Fig. 2a and b).

For $|\lambda_t| < 1$, the fluctuation term of eq. (4) is negligible for small $|\Delta\phi|$, and eq. (3) (for the upper case) leads

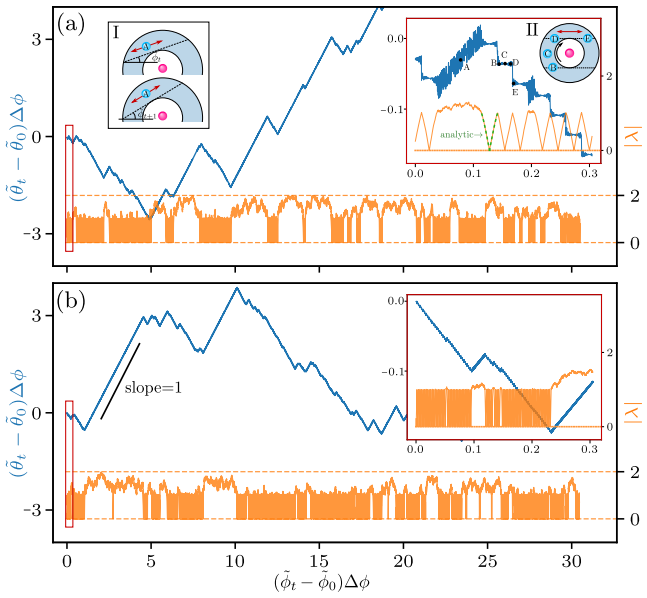


FIG. 2. Sample trajectories of the rolled-out dipole angle $\tilde{\theta}$ (blue, upper) and of $|\lambda|$ (yellow, lower) for $\eta = 2$. Axes are rescaled with $\Delta\phi$. Insets: Initial portion of the trajectories. (a): $\Delta\phi = \pi/180$. Sketch I illustrates "excursions" , sketch II "zigzags". (b): $\Delta\phi = 0.1\pi/180$.

to the non-linear differential equation $\mathrm{d}\lambda = -A(\lambda)\mathrm{d}\phi$. Its solution agrees with the trajectories (for $|\lambda| < 1$) obtained by numerical simulation (see inset of Fig. 2a). The half-period (in $\tilde{\phi}$), that is, the difference in $\tilde{\phi}$ which corresponds to one move from $\lambda = 1$ to $\lambda = -1$ (or vice versa), is always smaller than π. In the reference frame of constant $\tilde{\phi}$, it rotates in the mathematically negative sense (see the trajectory from B to point D in the inset of Fig. 2a). After one such half-period (which corresponds to roughly constant $\tilde{\theta}$), the dipole either jumps to the other segment (from $\mathcal{S}^+$ to $\mathcal{S}^-$ or from $\mathcal{S}^-$ to $\mathcal{S}^+$) and continues to rotate in the negative sense (creating a "zigzag" for $|\lambda| < 1$) or else goes over to an excursion with $|\lambda| > 1$. In trajectories of $\tilde{\theta}_t\Delta\phi$ as a function of $\tilde{\phi}_t\Delta\phi$, the zigzags become ever steeper as $\Delta\phi \to 0$ unlike the excursions, that approach a well-defined limit (see Fig. 2a and b, again).

One zigzag corresponds to a constant negative rotation of $\tilde{\theta}$ (as $\Delta\phi \to 0$) whereas one excursion corresponds to a positive rotation diverging as $\sim \mathrm{const}/\Delta\phi$. Nevertheless, positive and negative rotations balance, and the expectations $\mathbb{E}\left(\tilde{\theta}_{t+\Delta t} - \tilde{\theta}_t\right)$ are all zero. This is trivial for $\Delta t = 1$, as the distribution $\pi(\tilde{\theta}_{t+1} - \tilde{\theta}_t)$ is symmetric around zero because of detailed balance (see Fig. 3a). Furthermore, $\tilde{\theta}_t - \tilde{\theta}_0 = \tilde{\theta}_t - \tilde{\theta}_{t-1} + \cdots + \tilde{\theta}_1 - \tilde{\theta}_0$ yields

$$\mathbb{E}\left(\tilde{\theta}_t - \tilde{\theta}_0\right) = \mathbb{E}\left(\tilde{\theta}_t - \tilde{\theta}_{t-1}\right) + \mathbb{E}\left(\tilde{\theta}_{t-1} - \tilde{\theta}_{t-2}\right) + \ldots + \mathbb{E}\left(\tilde{\theta}_2 - \tilde{\theta}_1\right) + \mathbb{E}\left(\tilde{\theta}_1 - \tilde{\theta}_0\right) = 0, \quad (6)$$

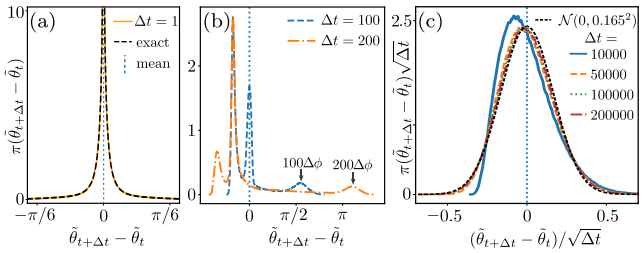


FIG. 3. Distributions $\pi(\tilde{\theta}_{t+\Delta t} - \tilde{\theta}_t)$ for $\eta = 1.1$ and $\Delta\phi = \pi/180$. (a): $\Delta t = 1$. (b): Distributions for moderate Δt. (c): Rescaled large-Δt distributions compared to a Gaussian.

because the expectation of a sum of (possibly dependent) random variables equals the sum of expectations. For $\Delta t > 1$, the distribution of the rolled-out dipole angle can be highly asymmetric, although the random walk remains unbiased (see Fig. 3b). For $\Delta t \lesssim \mathrm{const}/(\Delta\phi)^2$, the distribution peaks for large $\tilde{\theta}$ that corresponds to trajectories that remain on long excursions. For $\Delta t \gg \mathrm{const}/(\Delta\phi)^2$, the distribution approaches a Gaussian (and becomes again symmetric), because excursions and zigzags then compensate for each interval $[t, t + \Delta t]$ (see Fig. 3c).

The vanishing of $\mathbb{E}\left(\tilde{\theta}_{t+\Delta t} - \tilde{\theta}_t\right)$ implies that many ($\propto 1/\Delta\phi$) zigzags follow one another in between two excursions of the dipole. Microscopically, this can be understood through the existence of a cutoff value λ_{cut} of the impact parameter (see Fig. 1c). A dipole with $|\lambda_t| > \lambda_{\mathrm{cut}}$ leads to $|\lambda_{t+1}| > 1$, while a dipole with $\lambda_{\mathrm{cut}} > |\lambda_t| > 1$ may either produce $|\lambda_{t+1}| > 1$ or $|\lambda_{t+1}| < 1$, in which case it may remain trapped in its corresponding segment.

We quantify the convergence of the complicated random walk composed of zigzags and excursions through the total variation distance (TVD) [2, 18],

$$||\pi^t - \pi||_{\mathrm{TVD}} = \frac{1}{2}\int \mathrm{d}\overline{\mathbf{x}}|P^t\pi_0(\overline{\mathbf{x}}) - \pi(\overline{\mathbf{x}})|, \quad (7)$$

where $\overline{\mathbf{x}}$ stands for all the variables describing the configuration space and π_0 is the most unfavorable initial probability distribution. The time t after which the TVD reaches $\epsilon = 1/4$ is defined as the mixing time $t_{\mathrm{mix}} = t_{\mathrm{mix}}(1/4)$. For $\epsilon' < \epsilon$, $t_{\mathrm{mix}}(\epsilon')$ is easily bounded through the value for $\epsilon = 1/4$. [2] In our case, the distribution π^t depends on three variables θ, ρ, and ϕ. However, ρ relaxes very quickly and ϕ is a lifting variable without influence on π. We thus consider the TVD for the distribution π_θ, that only depends on θ (that is, we integrate over ρ and ϕ). Finally, we use a δ-function at most unfavorable ρ_0 and θ_0 as an initial distribution π_0. With these approximations, we systematically study t_{mix} for our sequential MCMC as a function of the set $\mathcal{D}$ characterized by $\Delta\phi$. We compare the sequential algorithm with random discrete MCMC where the new direction is randomly sampled from $\mathcal{D}$ and with random continuous MCMC where $\Delta\phi = \mathrm{ran}(0, \pi)$.

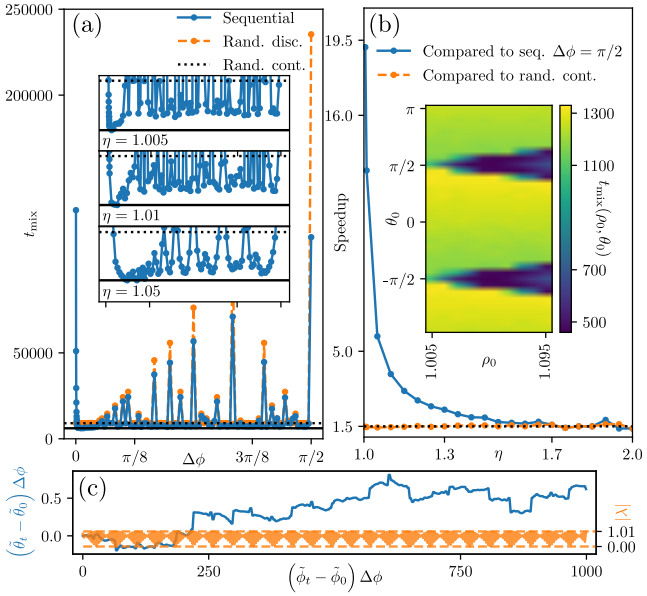


FIG. 4. Mixing times and trajectories for general $\Delta\phi$ (a): $t_{\rm mix}$ at $\eta = 1.005$ compared to mixing times of random discrete and continuous MCMC. Inset: $t_{\rm mix}$ for small $\Delta\phi$ and $\eta \gtrsim 1$. (b): Optimal speedup with respect to $\Delta\phi = \pi/2$ and to random continuous MCMC. Inset: Dependence on ρ_0 and θ_0 for $\eta = 1.1$ and $\Delta\phi = \pi/180$. (c): $\tilde{\theta}_t$ and λ_t for $\eta = 1.01$ and $\Delta\phi = 85\pi/180$.

Several properties stand out (see Fig. 4). First, we find that accessing the elements of $\mathcal{D}$ sequentially is generically better than randomly sampling the direction from $\mathcal{D}$ (see Fig. 4a). (This no longer holds as $\Delta\phi \to 0$ where the mixing time naturally diverges for sequential MCMC.) The benefits of sequential MCMC are reminiscent of what is known for the sequential particle labeling [7] or for spin sweeps [10]. The mixing time is very sensitive to the size of $\mathcal{D}$. Second, we find that sequential MCMC is faster than the random continuous choice of directions except when $|\mathcal{D}|$ is very small. Third, we confirm that for small $\Delta\phi$, the very special Markov chain described in the core of this paper indeed has the smallest mixing time. This, in our model, can of course only be observed for $\eta - 1 \ll 1$ because the speedup for small $\Delta\phi$ is cut off by the divergence of $t_{\rm mix}$ for $\Delta\phi \to 0$ (see inset of Fig. 4a). Fourth, we find that the sequential Markov chains for generic $\Delta\phi$ remain peculiar (as they are for small $\Delta\phi$). They feature intriguing patterns for λ_t and the rolled-out dipole angle $\tilde{\theta}_t$ (see Fig. 4c).

In conclusion, we have discussed in this paper a class of Markov chains with reduced randomness. Our example consisted in the choice of directions from a set. Sequentially accessing the elements of this set left the stationary probability distribution unchanged. It profoundly changed the properties of the Markov chain and clearly reduced mixing times. In real-world MCMC, mixing times and correlation times are often counted in weeks or months. There, the potential benefits of what corresponds in our simplified dipole model to small $\Delta\phi$ may not be cut off by the $\Delta\phi \to 0$ divergence of the mixing time. It will thus be fascinating to understand the usefulness of sequential MCMC in applications such as polymer physics and, also, systems of long-range interacting extended molecules at the core of the JeLLyFysh project. [15]

ACKNOWLEDGEMENTS

W.K. acknowledges support from the Alexander von Humboldt Foundation. We thank A. C. Maggs for helpful discussions.

* liang.qin@phys.ens.fr
† hoellmer@physik.uni-bonn.de
‡ werner.krauth@ens.fr

[1] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, J. Chem. Phys. **21**, 1087 (1953).
[2] D. A. Levin, Y. Peres, and E. L. Wilmer, *Markov Chains and Mixing Times* (American Mathematical Society, 2008).
[3] N. V. Prokof'ev and B. V. Svistunov, Phys. Rev. Lett. **81**, 2514 (1998).
[4] K. Van Houcke, E. Kozik, N. Prokof'ev, and B. Svistunov, Physics Procedia **6**, 95 (2010).
[5] H. Suwa and S. Todo, Phys. Rev. Lett. **105**, 120603 (2010).
[6] K. S. Turitsyn, M. Chertkov, and M. Vucelja, Physica D: Nonlinear Phenomena **240**, 410 (2011).
[7] C. J. O'Keeffe and G. Orkoulas, J. Chem. Phys. **130**, 134109 (2009).
[8] S. C. Kapfer and W. Krauth, Phys. Rev. Lett. **119**, 240603 (2017).
[9] Z. Lei and W. Krauth, EPL **124**, 20003 (2018).
[10] B. A. Berg, *Markov Chain Monte Carlo simulations and their statistical analysis: with web-based Fortran code* (World Scientific, 2004).
[11] E. P. Bernard, W. Krauth, and D. B. Wilson, Phys. Rev. E **80**, 056704 (2009).
[12] M. Michel, S. C. Kapfer, and W. Krauth, J. Chem. Phys. **140**, 054116 (2014).
[13] P. Diaconis, S. Holmes, and R. M. Neal, Ann. Appl. Probab. **10**, 726 (2000).
[14] M. F. Faulkner, L. Qin, A. C. Maggs, and W. Krauth, J. Chem. Phys. **149**, 064113 (2018).
[15] P. Höllmer, L. Qin, M. F. Faulkner, A. Maggs, and W. Krauth, Comput. Phys. Commun. **253**, 107168 (2020).
[16] W. Krauth, *Statistical Mechanics: Algorithms and Computations* (Oxford University Press, 2006).
[17] H. Risken, *The Fokker-Planck Equation* (Springer Berlin Heidelberg, 1989).
[18] P. Diaconis, J. Stat. Phys. **144**, 445 (2011).

RÉSUMÉ

Cette thèse étudie le comportement de la chaîne d'événements de Monte Carlo (ECMC) dans les systèmes de particules à longue portée. Nous proposons la formule d'échantillonnage ECMC dans un système de Coulomb dans le cadre des conditions périodique aux limites. En regroupant toutes les paires coulombiennes entre deux molécules, le facteur dipolaire induit des taux d'événements plus faibles. En combinaison avec la "cell-veto" méthode, nous obtenons un algorithme O(NlogN)-par-balayage pour les systèmes dipolaires. Nous développons également une application scientifique appelée JeLLyFysh pour les simulations moléculaires par ECMC. JeLLyFysh est conçu de manière très extensible et est open-source en ligne. A l'aide de JeLLyFysh, nous établissons un profil des performances de l'ECMC pour les grands systèmes d'eau. La dynamique qui en résulte implique qu'un schéma plus sophistiqué est nécessaire pour équilibrer la polarisation. Ainsi, nous testons la stratégie d'échantillonnage avec changement séquentiel de direction. L'évolution du dipôle présente une dynamique particulière, et l'ensemble des choix de direction ainsi que l'ordre de sélection s'avèrent tous deux cruciaux pour atteindre la distribution stationnaire de l'orientation du dipôle.

MOTS CLÉS

Méthode de Monte Carlo, chaîne de Markov irréversibles, interaction coulombienne, système de longue portée, simulation moléculaire

ABSTRACT

This thesis studies the behavior of event-chain Monte Carlo (ECMC) in long-range particle systems. We propose the formula for ECMC to sample in a Coulomb system under periodic boundary conditions. By grouping all Coulomb pairs between two molecules, the dipole factor induces smaller event rates. Together with the cell-veto method, we obtain an O(NlogN)-per-sweep algorithm for dipole systems. Also, we develop a scientific application called JeLLyFysh for molecular simulations through ECMC. JeLLyFysh is designed in a highly extensible way, and is open-source online. Using JeLLyFysh, we profile the performance of ECMC for large water systems. The resulting dynamics imply that a more sophisticated scheme is needed to equilibrate the polarization. Thus, we test the sampling strategy with sequential direction changes. The dipole evolution exhibits distinct dynamics, and the set of direction choices and the order of selection prove both crucial in mixing the dipole's orientation.

KEYWORDS

Monte Carlo method, irreversible Markov chain, Coulomb interaction, long-range system, molecular simulation