Received: 23 June 2020 | Revised: 26 August 2020 | Accepted: 29 August 2020

DOI: 10.1002/jcc.26418

SOFTWARE NEWS AND UPDATES

Journal of COMPUTATIONAL CHEMISTRY WILEY

MOCASSIN: Metropolis and kinetic Monte Carlo for solid electrolytes

Sebastian Eisele[1,2] | Steffen Grieshammer[1,2]

[1]Forschungszentrum Julich Helmholtz-Institut Munster Institut fur Energie- und Klimaforschung Elektrochemische Verfahrenstechnik (Ringgold: 551614)

[2]Institute of Physical Chemistry, RWTH Aachen University, Aachen, Germany

Correspondence
Sebastian Eisele, Forschungszentrum Julich Helmholtz-Institut Munster Institut fur Energie- und Klimaforschung Elektrochemische Verfahrenstechnik (Ringgold: 551614).
Email: s.eisele@fz-juelich.de, s.grieshammer@fz-juelich.de

Funding information
Deutsche Forschungsgemeinschaft, Grant/Award Number: GR-5011/1-1

Abstract

The combination of density functional theory and Monte Carlo simulations is a powerful approach for the atomistic modeling of defect transport in solid electrolytes. The present contribution introduces the MOCASSIN software (Monte Carlo for Solid State Ionics) for kinetic and Metropolis Monte Carlo simulations of crystalline materials. MOCASSIN combines model building, visualization, and simulation, aiming to provide accessible MC for end users. Developed for the investigation of solid electrolytes, MOCASSIN is ideal for screening common variation parameters, such as temperature and doping fraction. The input effort is minimized using space groups for processing symmetry. The graphical interface for model building allows complex model input, including multiple mobile species, multiple migration paths, small polaron hopping, vehicle movements, multiple complex migration mechanisms, and custom interaction clusters. The software is provided free of charge for noncommercial usage.

KEYWORDS

ionic conductivity, ionic diffusion, kinetic Monte Carlo, Metropolis Monte Carlo, solid electrolytes

1 | INTRODUCTION

Monte Carlo (MC) is a powerful widely used numeric solution method for statistical problems. It samples the occurrence of events proportional to a probability function *P* and yields expected values for properties of interest, where the accuracy of the results increases with the number of sampled events. Since its formal introduction,[1] it has been adapted for a broad range of applications (e.g., see References 2–6 for further information).

In the field of particle physics and chemistry, MC methods coexist with molecular dynamics (MD) as the primary approaches for simulating the statistical properties of interacting ensembles that cannot be solved analytically (e.g., see References 7–12 for further information). MC often has the advantage of relative simplicity, lower resource requirements, and reduced computational cost compared to classical MD. Naturally, this is not always true, but it has led to the development of multiple generic solutions for Metropolis Monte Carlo (MMC) simulations of soft matter, liquids, solids, and gaseous phases in 3D space (e.g., see References 13,14). Another approach, using fixed position grids, is especially interesting for periodic high-symmetry systems and enables straightforward kinetic Monte Carlo (KMC). This allows the simulation of dynamic system properties based on state change events. Ionic mobility, ionic conductivity, and diffusion are most relevant for crystalline solid electrolytes. Since classical MD is computationally expensive and suitable potential sets are not always available, a combined density functional theory (DFT)/MC approach enables the broad simulation of defective materials on a first-principles basis. For this purpose, reliable and effective MC software is required that allows the direct modeling of arbitrary crystal structures based on first-principles energy calculations. Although several KMC codes exist (e.g., see References 14–17), we found the application or the adaptation of available software for KMC in solid electrolytes problematic. Common issues are: (i) outdated or no longer maintained source codes; (ii) efficiency issues simulating highly

interacting systems; or (iii) program designs not targeting large crystalline systems and thus inconvenient to use. Therefore, we developed *Monte Carlo for Solid State Ionics* (MOCASSIN) as a convenient solution, designed to make fixed-position MMC/KMC simulations of defects in solid electrolytes accessible. From a functionality viewpoint, the software has four major objectives; (i) usage of ideal crystal geometry and space group symmetry to minimize input effort; (ii) support of energy models based on first-principles calculations with acceptable efficiency for strongly interacting systems; (iii) convenient definition of multiple complex migration processes; and (iv) focus on a simple-to-use program for end users. This leads to the core development guidelines: (i) accessibility is favored over universality; (ii) default use cases are covered by a graphical interface; and (iii) input for solver routines is generated by the model tools, keeping code in performance languages and manual input files to a minimum. Since completion of the core functionality, MOCASSIN has been used successfully for in-house projects investigating fast ion conductors.[18–20]

With MOCASSIN, we aim to improve accessibility to MC and reduce the need for system-specific developments. We do not aim at providing a truly universal solid-state MC program; instead, MOCASSIN targets the simulation of ionic transport in crystals with moderately complex models based on data from first-principles calculations. Here, we present some of the top-tier conceptual ideas and validation examples relevant for researchers and potential users. In-depth details of processing logic or components are not presented. We want to illustrate how MOCASSIN achieves model freedom, low input effort, and widespread applicability, combined with high levels of automation.

2 | METHODS

2.1 | Monte Carlo in solid electrolytes

Monte Carlo simulations for solid-state materials have a broad usage spectrum. We focus on MMC and null-event KMC, an alternative to the rejection-free KMC principle,[21] as applied in MOCASSIN.

2.1.1 | Metropolis Monte Carlo

Metropolis Monte Carlo allows equation of state calculations through statistical sampling of state distributions.[22] Fixed-position MMC simulates the distribution of particles by swapping particles between sites with a certain probability. The method samples states proportional to a probability function $P(E)$ from the configuration space $\Omega(E)$. In thermal equilibrium, the probability function is given by Boltzmann statistics $P(T, E)$ (1).

$$P(T,E_i) = \exp\left(-\frac{E_i}{k_B T}\right) \tag{1}$$

Effectively, MMC efficiently samples part of the configuration space with an unknown factor m as given in (2).

$$m \cdot M(E) = \Omega(E) \cdot \exp\left(-\frac{E_i}{k_B T}\right) \tag{2}$$

After reaching thermal equilibrium and given sufficient sample sizes, MMC in crystals yields average ensemble properties, including the interaction contribution to inner energy, defect distribution, and occurrence of phase separations.

2.1.2 | Null-event algorithm for kinetic Monte Carlo

There are two general ways of implementing KMC routines: with and without event rejection.[21] The rejection-free principle requires current event options and affiliated rates to be known for correct selection of the next event. Thus, huge rate catalogues are necessary, which are updated after each simulation cycle. The null-event KMC algorithm is a trial and error solution analogous to MMC. Instead of creating and updating rate catalogues, it statistically accepts or rejects uniformly selected events to achieve proper evolution of the system. There are three major steps involved: (i) select one of the possible events in the system using a uniform random distribution; (ii) calculate the affiliated probability of success P in range [0…1] according to an arbitrary probability function; (iii) accept the event if P is unity or draw a uniform real number $RN \in [0.0…1.0)$ and accept the event if $P \geq RN$, otherwise reject the event (null event). As a result, the null-event routine replaces expensive rate catalog updating with rejection overhead. In theory, the algorithmic efficiency depends solely on the accepted/rejected event ratio for a given problem, making the rejection-free routine the common choice. In practice, computer hardware complicates the issue as huge rate catalogues cause CPU bottlenecking due to memory limitations. This makes the null-event principle especially competitive in cases with huge, strongly interdependent event sets, where branch traversals and updating of rate catalogues represented by binary trees are memory-limited and thus expensive.

2.1.3 | Kinetic Monte Carlo of ionic transport

The idea of kinetic Monte Carlo is to assign a defined time step to each event and limit the affiliated spatial component. In crystalline solids, it therefore permits simulation of dynamic defect transport through the lattice, that is, diffusion coefficients D_i, ionic conductivity σ_i, and ionic mobility u_i. The transport of ions is a thermally activated process with a migration barrier E_{mig}. In the context of transition state theory, E_{mig} depends on the three occurring states of the system during transport: initial state S_0, unstable transition state S_1, and final state S_2. E_{mig} for forward or backward migration is then $E(S_1) - E(S_0)$ or $E(S_1) - E(S_2)$, respectively. Importantly, the states must obey a detailed balance to ensure meaningful evolution of the system.[21] This can be achieved by describing the states using exact energy

calculations for each configuration. However, the number of permutations often renders this route impractical. Thus, KMC usually obtains $E(S_0)$ and $E(S_2)$ by interaction summation. One model for consistent $E(S_1)$ definition uses a base value $E_{1,\text{base}}$, depending on the transition site environment, and an additive part to account for the energetic differences between S_0 and S_2. A linear interpolation as given in (3), is a basic and commonly used description of $E(S_1)$.[23]

$$E(S_1) = \frac{E(S_0) + E(S_2)}{2} + E_{1,\text{base}} \tag{3}$$

In contrast to MD, the migration rate of atoms Γ_0 in KMC is usually not only subjected to the energetic landscape and temperature. Transport events occur statistically and thus the attempt rate ν_0, that is, the number of attempts per time, is required. Values are either dynamically calculated or a constant ν_0 is used. Within crystalline solids, choosing $\nu_0 \in [10^{12}...10^{13}]$ for event attempt rates is a common approximation.[21] Then, the actual migration rate of ions Γ_0 (4) is the product of ν_0 and $P(E, T)$.

$$\Gamma_0 = \nu_0 \cdot \exp\left(-\frac{E_{\text{mig}}}{k_B T}\right) \tag{4}$$

In complex cases with multiple mobile species or transport mechanisms, individual attempt rates for each event ν_i may be required. The distinct rates ν_i can be described by products of a factor $f_i \in [0...1]$ with the highest rate ν_0 yielding the individual rate Γ_i (5).

$$\Gamma_i = \nu_0 \cdot f_i \cdot \exp\left(-\frac{E_{\text{mig}}}{k_B T}\right) \text{ with } f_i = \frac{\nu_i}{\nu_0} \tag{5}$$

Consequently, this unifies the calculation of the event time step Δt. Assuming a sequential occurrence of events, (6) describes Δt as a function of active event sites in the lattice N_{site}, the average number of events per site $\langle N_{\text{event}} \rangle$, and the base attempt frequency ν_0.

$$\Delta t = \frac{1}{N_{\text{site}} \cdot \langle N_{\text{event}} \rangle \cdot \nu_0} \tag{6}$$

For atomistic transport simulations, N_{site} is usually constant and equivalent to the number of mobile atoms N_{mobile}, while $\langle N_{\text{event}} \rangle$ is the average number of migration directions per atom. The latter can be a function of the current system state and requires dynamic runtime adjustments.

As in MD, the trajectory of particles in KMC is individually trackable for each species. In this way, particle tracking yields individual displacements $\vec{r}_{i,j}$ for each particle, species ensemble displacements $\vec{R}_i$, and mean square displacements $\left\langle \vec{R}_i^2 \right\rangle$ (MSD). Tracer diffusion coefficients D_i^* and diffusion correlation factors $f_{\text{cor},i}$ can be calculated from the MSD time derivative. Additionally, fixed-position KMC allows more specific tracing operations: (i) tracking of the local particle flow at each lattice position, without needing arbitrary tracking volumes; and (ii) direct separation of contributions from transport mechanisms.

It is possible to calculate the mobility u_i and the ionic conductivity σ_i of an individual mobile species by three distinct concepts. The first approach is calculation of the mobility from the diffusion coefficient using the Nernst-Einstein relation. However, this requires the Haven ratio H_R to be known and is thus not feasible in the general case.[24] The second solution is based on linear response theory, stating that the drift velocity $v_{d,i}$ shows a response linear to the magnitude of an external electric potential. Thus, u_i is the scalar projection of the average ensemble displacement $\left\langle \vec{R}_i \right\rangle$, per electric field modulus $|E|$ and time t, onto the normalized electric field $\vec{E}_n$ (7). The ionic conductivity of a species σ_i is then calculated from the mobility u_i, species charge q_i, and the particle density per unit volume c_i (8). Importantly, this approach assumes a linear electric potential gradient within the material and remains valid only if the response (ionic drift velocity) is proportional to the force (external electric field).

$$u_i = \frac{\left\langle \vec{R}_i \right\rangle \cdot \vec{E}_n}{|E| \cdot t} \tag{7}$$

$$\sigma_i = u_i \cdot q_i \cdot c_i \tag{8}$$

Choosing a proper field strength is a question of the balance between statistically meaningful drift and maintaining linear response behavior. The electric field causes an additive energetic contribution ΔE_{field} that affects the potential energy change between states. $\Delta E_{\text{field}}(S_0 \rightarrow S_2)$ is described by a scalar projection of the shift of the charge focal point onto the electric field (9), which is a finite sum in the case of point charges.

$$\begin{aligned} \Delta E_{\text{field}} &= \vec{E} \cdot \int \left(\vec{r}_2 - \vec{r}_0\right) \mathrm{d}q \\ &= \vec{E} \cdot \sum_i \left(\vec{r}_{2,i} - \vec{r}_{0,i}\right) \cdot q_i \end{aligned} \tag{9}$$

On average, half of the $\Delta E_{\text{field}}(S_0 \rightarrow S_2)$ can affect the barrier E_{mig}, as only half of the potential change can be converted into kinetic energy (10).[25,26]

$$\begin{aligned} &E_{\text{mig}}(|E| \neq 0) \\ &= E_{\text{mig}}(|E| = 0) - \frac{1}{2} \cdot \Delta E_{\text{field}}(S_0 \rightarrow S_2) \end{aligned} \tag{10}$$

The last evaluation option determines the linear Onsager transport coefficients[27,28] L_{ij} by evaluating the affiliated Kubo-Green relations.[29,30] Given sufficient statistical input, both individual species movement and correlation effects between species can be analyzed. The calculation is straightforward for cubic systems as shown in (11) and (12).[31] However, application of the method becomes significantly more complicated in the general case.

$$L_{ij} = \frac{\left\langle \vec{R}_i \cdot \vec{R}_j \right\rangle}{6 \cdot V \cdot k_B \cdot t} \tag{11}$$

$$\sigma_i = L_{ii} \cdot q_i^2 \tag{12}$$

2.1.4 | Geometry in fixed-position Monte Carlo

In this section, we present basic aspects of the symmetry operations and integer lattice representation applied in the simulation of crystalline materials. In-depth explanations of space groups and the underlying mathematics are available in the literature (e.g., see Reference 32).

Space groups and symmetry operations

The 230 space groups describe all possible symmetries within crystallographic structures. A group consists of a set of symmetry operations where the entity operation is always included. Each symmetry operation is a Euclidean mapping of one object configuration to another and thus an affine transformation, preserving all distances and angles of the object on transformation. Transformation produces either congruent objects, operations of the first kind, or it affects orientation and therefore yields enantiomorphous objects, operations of the second kind. The operations are commonly found in an "x, y, z" notation with component coefficients c_{ij} and translation shifts w_i as shown in (13) (parenthesis added for readability). They can be translated into general 4×4 affine transformation matrices T_s for homogeneous coordinates in row vector notation (14).

$$\begin{matrix}(c_{xx}x + c_{xy}y + c_{xz}z + w_x),\\ (c_{yx}x + c_{yy}y + c_{yz}z + w_y),\\ (c_{zx}x + c_{zy}y + c_{zz}z + w_z)\end{matrix} \tag{13}$$

Here, (x, y, z) coordinates refer to the unit cell coordinate space and not a Cartesian system. For Cartesian coordinates in homogeneous notation, calculating the composed $(M_f T_s M_c)$ transform is required. Here, M_f transforms from the Cartesian to the cell coordinate system, T_s executes the symmetry operation and M_c transforms the result back to Cartesian coordinates.

$$\begin{aligned}\vec{x} \cdot T_s &= (x\,y\,z\,1)\begin{pmatrix} c_{xx} & c_{yx} & c_{zx} & 0\\ c_{xy} & c_{yy} & c_{zy} & 0\\ c_{xz} & c_{yz} & c_{zz} & 0\\ w_x & w_y & w_z & 1\end{pmatrix}\\ &= \vec{x}' \begin{cases} x' = c_{xx}x + c_{xy}y + c_{xz}z + w_x\\ y' = c_{yx}x + c_{yy}y + c_{yz}z + w_y\\ z' = c_{zx}x + c_{zy}y + c_{zz}z + w_z\end{cases}\end{aligned} \tag{14}$$

Using symmetry operations for symmetry extension or reduction requires more work than transforming each vector. A transform and filter approach can be used, which is effective due to the limited number of operations per group. This requires a differentiation between isolated points and geometric paths, that is, sequences of connected points. Starting with the former, extending any cell position (x, y, z) into the translation invariant set is straightforward: (i) transform the original vector with all available symmetry operations; (ii) trim each resulting vector back into the [0...1) coordinate range of the unit cell; and (iii) extract the unique entries from the resulting collection. In contrast, a path is subjected to the Euclidean mapping operations as an entity. This slightly modifies the routine; (i) describe the reference path by absolute points; (ii) transform each point of the sequence with the operations; (iii) for each result, calculate a transform shifting the path origin point into the [0...1) range and apply it to the sequence; and (iv) filter the results for unique entries. In this case, attaching tags to each point of a sequence and including these in the equality comparison opens up another area of application. By tagging positions with occupation sets and evaluating all resulting path permutations for equivalency, the symmetry-reduced set of unique path occupations in accordance with the space group symmetry is obtained through the filter step.

2.1.5 | Integer lattice coordinates

Floating-point coordinates are well suited for treating of geometric data: (i) transformation between different coordinate spaces is simple; (ii) equivalent geometries can be generated when required; and (iii) the data remains human readable during I/O operations. However, frequently needed operations, for example, finding the target of a relative vector, require numeric searches. This produces computational overhead during the simulation and can often be avoided by using integer mappings, making geometric data available through multidimensional lookup tables. Importantly, mapping for fixed positions can be done without using approximated grids. For crystals, an efficient solution to map 3D data exactly is 4D integer transformation, taking advantage of the system's translation invariance. The affiliated coordinates are (z_a, z_b, z_c, z_p) with $z_i \in$? to describe both positions and relative vectors. The first three entries (z_a, z_b, z_c) describe the integer offset of the unit cell origin point as multiples of (a, b, c) cell directions, respectively. The fourth entry z_p indexes the atomic site in the unit cell and the affiliated coordinate vector $\vec{x}_0$ of the (0, 0, 0) origin cell entry is stored in a reference list. The general operation to restore any 3D cell coordinate vector $\vec{x}$ from a 4D integer tuple is shown in (15).

$$\begin{aligned}&\vec{x}(z_a, z_b, z_c, z_p) =\\ &z_a \vec{a} + z_b \vec{b} + z_c \vec{c} + \vec{x}_0(z_p)\end{aligned} \tag{15}$$

There are certain advantages to this layout: (i) dynamic definition of the 4D space for arbitrary geometry only requires the unit cell definition; (ii) access of positional data is an $O(1)$ index operation; and (iii) the 4D integer tuple is a native candidate for SIMD (single

instruction multiple data) vector instructions. The obvious downside is the cryptic nature of the data, rendering it impractical for user interaction and data visualization.

2.2 | The MOCASSIN solution

The following section describes MOCASSIN as a computational solution. Both the model processing system and simulation code are discussed on the conceptual and feature level, with a special focus on issues relevant to potential users. Illustration of technical details is limited to general framework choices and reasoning for existing limitations of the MOCASSIN processing components.

Based on the intended purpose, MOCASSIN components fall into four distinct categories: (i) model processing, (ii) simulation, (iii) data evaluation, and (iv) user interaction. These categories have diverse requirements concerning performance, functionality, and development effort. Figure 1 presents a simplified illustration of the general MOCASSIN solution layout including the core managed, unmanaged, and database components. The model processing system is the core of MOCASSIN, which manages the user model and packages data for simulation. Object-oriented programming (OOP), proper exception handling, good memory management, strong typing, and access to reliable data storage technologies, are some of the priorities that outweigh any performance concerns. The simulator code focuses on performance, Linux/Windows platform portability, and communication with the model processing system. After careful consideration, we decided on platform-portable C# for the main framework (.NET Core), Windows-specific C# for the graphical interface (.NET Framework) and C (2011 standard) for the simulator. The C#/C data exchange is realized through local SQLite databases serving as containers.

2.2.1 | Feature overview

The main features of MOCASSIN are accessible MMC and KMC simulation of equilibrium defect distribution and ionic transport in crystals with minimized input effort. Thus, MOCASSIN conveniently provides its core functionalities through a graphical interface to assist in efficient model building, as well as the creation and simulation of large parameter studies.

Model building supports up to 63 custom species and evaluates subsequent model components based on charge conservation, mass conservation, and vacancy behavior. Mobility is an implicit model property and all 63 species can thus be mobile within a single model. The structure is defined through space group selection, cell parameters, and reference sites, with attached particle sets to describe possible occupations. The required space group database is included. Energy modeling is based on interaction range cutoffs, supporting radial filtering with automatic detection and symmetry reduction of pair interactions. Cluster geometries with up to eight positions around a center site are supported. Parameterization of the resulting energy contributions is achieved through program-supplied templates. Transition modeling supports an arbitrary number of complex process definitions per model, with up to eight positions per KMC path, and automated deduction of physically meaningful behavior. This allows the definition of vacancy mechanisms, interstitialcy mechanisms, migration of multibody vehicles, small polaron hopping, and MMC site exchanges in a uniform style. By means of geometry bindings, process definitions support multiple reference paths and attempt rates can be customized. For example, modeling oxygen migration in the fluorite structure of ceria requires a single reference event to model all 48 nearest-neighbor migration options in the unit cell.

For a convenient definition of parameter studies, MOCASSIN implements a basic inheritance hierarchy for simulation sets with simulation core definitions, job groups with collective settings, and

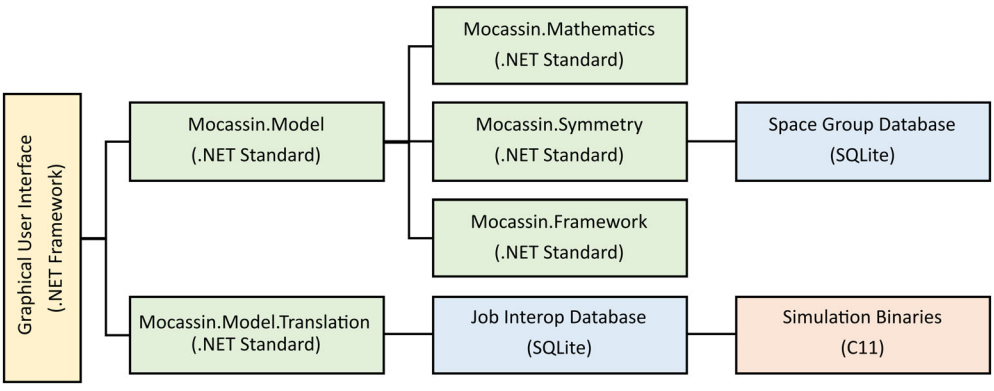

FIGURE 1 Simplified schematic representation of the MOCASSIN solution layout showing the major processing components. The Windows-only WPF user interface (yellow) is built upon a fully portable set of .Net Standard 2.0 libraries (green) that form the application programming interface (API) for model processing. Storage of space group information and data exchange with the C simulator binaries (orange) is realized through SQLite databases (blue). All components are native 64-bit with data layouts tailored to solver execution on typical Linux-based x86-64 high-performance clusters [Color figure can be viewed at wileyonlinelibrary.com]

individual job overwrites. Explicit duplication functions and optional automated implicit duplications simplify sample collection for statistical deviation evaluations. Fast generation of simulation supercells is provided through a doping system, supporting automated charge balancing and execution order. Simulation startup data are stored in SQLite databases, supporting later storage of collected simulation results to avoid huge collections of loose files.

The graphical user interface provides convenience functionalities to support the user. Data are stored in solution files that can contain an arbitrary number of independent model projects. Each project is divided into the reference model, an arbitrary number of model parameterizations, and a set of simulation build templates. Controls are provided through dynamic work tabs and tab-hosting subwindows. Numeric fields support expression parsing for a more natural input. Limited 3D visualization of sites, interactions, and migration pathways is possible through a DirectX10/11 viewer with image export in multiple formats. Additionally, a validation system assists in the identification of obvious errors and informs about potentially problematic implications of model definitions.

2.2.2 | Core processing components

The C# MOCASSIN processing core is a pipeline system with three consecutive components: (i) the model management system, which validates and stores the reference model; (ii) the model context system that manages data relations between the model and its P1 symmetry extension; and (iii) the model translator for conversion of model context data into simulation job databases. Figure 2 illustrates the typical workflow during definition of a MOCASSIN project. The following section describes important steps and presents an overview of the core processing principles.

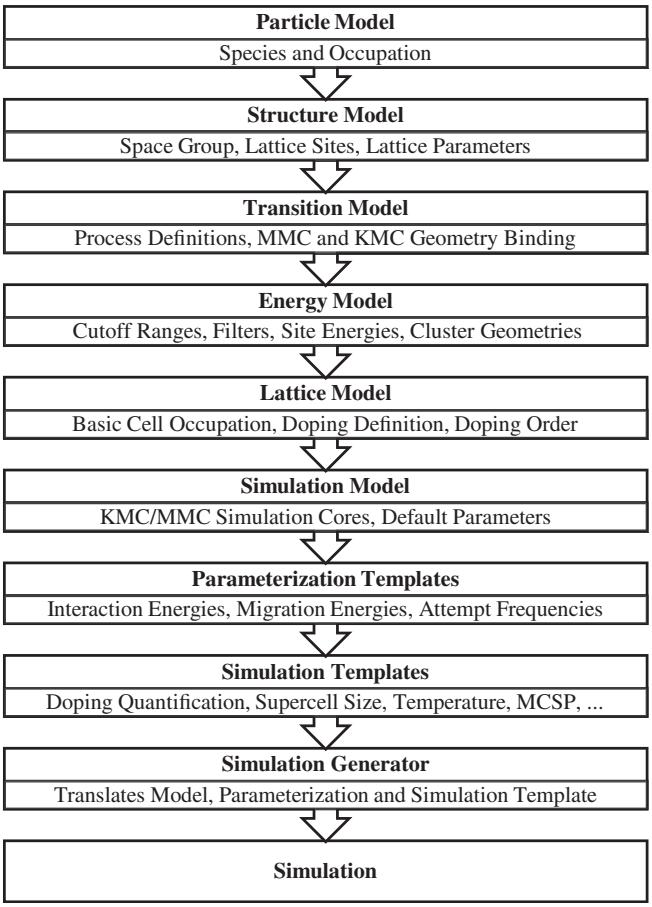

FIGURE 2 Simplified MOCASSIN workflow from definition of particles to translation of the reference model into simulation databases

2.2.3 | Particles and structure definition

The first step in a MOCASSIN project is the definition of particles and particle sets. Particles have three important properties: (i) the charge state; (ii) an element or pseudo-element symbol required to tag multiple charge states of the same species; and (iii) a flag that indicates whether the particle is a vacancy pseudoparticle. Particle sets define unique collections of particles and provide information about possible site occupations. Unit cells are defined by a space group selection, the lattice parameters, and a set of reference sites. Each reference site includes cell coordinates and a particle set specifying allowed occupations. Additionally, a flag differentiates between regular lattice sites (stable) and transition sites (unstable).

2.2.4 | Transition and energy modeling

The modeling of both MMC occupation swaps and KMC transport processes follows a unified principle based on transition events. The key idea of the transition system is to abstract Monte Carlo events into simple base components and separate the process from the affiliated geometry. There are three major steps involved: (i) define required local occupation changes from one particle to another; (ii) describe process chains by a linear set of local changes and their sequential dependencies; and (iii) bind the process chain to geometric reference paths to create MMC or KMC events. This abstraction allows the definition of various processes: e.g., basic MMC exchanges, vacancy migration, vehicle migration, small polaron hopping, or concerted mechanisms such as interstitialcy. Considering the sequential dependencies and boundary conditions, for example, conservation laws, model processing can automatically determine sets of physically valid change rules for each combination of process chain and geometry. This abstract transition modeling is explained in the following using practical examples.

For an oxygen vacancy mechanism, a donor-acceptor analogy best describes the occupation change objects. There are three particles involved in this process: an oxygen ion O^{2-} (donor state), an oxygen vacancy V (acceptor state) pseudoparticle and a null pseudoparticle $\emptyset$, representing a noninteracting void state. The states S_0, S_1, S_2 are defined by the occupation of start position P_0, transition

position P_1, and end position P_2. The matrix in (16) shows the occupation for all positions involved for each state of the oxygen migration process.

$$\begin{bmatrix} & P_0 & P_1 & P_2 \\ S_0 & O^{2-} & \emptyset & V \\ S_1 & \emptyset & O^{2-} & \emptyset \\ S_2 & V & \emptyset & O^{2-} \end{bmatrix} \tag{16}$$

As seen in (16), two unique donor-acceptor changes are required to describe the local changes. Positions P_0 and P_2 both have O^{2-} as the donor state and V as the acceptor state, and P_1 has O^{2-} and $\emptyset$, respectively. Other local changes are inferred using mass and charge conservation laws, yielding (O^{2-}, V) and $(O^{2-}, \emptyset)$ as donor-acceptor pairs. The first pair is technically sufficient in the context of a vacancy mechanism. However, the system is flexible enough to handle charge transport and thus enforces an explicit definition to prevent ambiguity. After identification of the basic particle changes, it is possible to define the linearized process chain independent of geometry. To connect the local changes into a process chain, MOCASSIN defines "dynamic" ($\Leftrightarrow$) and "static" ($\nLeftrightarrow$) position linkers, which describe the dependent or independent occurrence of sequential site changes, respectively.

In the case of the oxygen vacancy mechanism, changing P_0 implies that P_2 performs the affiliated change and vice versa. Thus, the process requires the passing of information through P_1 yielding a chain with two ($\Leftrightarrow$) linkers as show in (17).

$$\left(O^{2-}, V\right) \Leftrightarrow \left(O^{2-}, \emptyset\right) \Leftrightarrow \left(O^{2-}, V\right) \tag{17}$$

By trying to advance all S_0 permutations to their S_1 and S_2 states, it is possible to deduce the physically valid S_0, S_1, S_2 state sets using boundary conditions. For example, the S_0 state with two ions (18) violates mass conservation $\Delta m = 0$ for S_1, as P_1 cannot be occupied by both ions at once. Only two valid sets exist, one identical to (16), and the other encoding the $S_2 \rightarrow S_0$ backwards migration, which must be modeled analogously to obey the detailed balance rule.

$$\begin{bmatrix} \Delta m & & P_0 & P_1 & P_2 \\ 0 & S_0 & O^{2-} & \emptyset & O^{2-} \\ -O^{2-} & S_1 & \emptyset & O^{2-} & \emptyset \\ 0 & S_2 & O^{2-} & \emptyset & O^{2-} \end{bmatrix} \tag{18}$$

More complex migration process chains and affiliated valid state matrices are illustrated in (19) and (20): in (19), concurrent charge transport between A and A^- and ion B vacancy migration, and in (20) the oxygen interstitialcy mechanism. Importantly, (19) has two valid transport cases describing either association/dissociation or forward/backward movement.

$$\begin{gathered} (A^-, A) \Leftrightarrow (e^-, \emptyset) \Leftrightarrow (A^-, A) \\ \nLeftrightarrow (B, V) \Leftrightarrow (B, \emptyset) \Leftrightarrow (B, V) \\ \begin{bmatrix} & P_0 & P_1 & P_2 & P_3 & P_4 & P_5 \\ S_0 & A^- & \emptyset & A & B & \emptyset & V \\ S_1 & A & e^- & A & \emptyset & B & \emptyset \\ S_2 & A & \emptyset & A^- & V & \emptyset & B \end{bmatrix} \\ \begin{bmatrix} & P_0 & P_1 & P_2 & P_3 & P_4 & P_5 \\ S_0 & A & \emptyset & A^- & B & \emptyset & V \\ S_1 & A & e^- & A & \emptyset & B & \emptyset \\ S_2 & A^- & \emptyset & A & V & \emptyset & B \end{bmatrix} \end{gathered} \tag{19}$$

$$\begin{gathered} \left(O^{2-}, V\right) \Leftrightarrow \left(O^{2-}, \emptyset\right) \Leftrightarrow \left(O^{2-}, V\right) \\ \Leftrightarrow \left(O^{2-}, \emptyset\right) \Leftrightarrow \left(O^{2-}, V\right) \\ \begin{bmatrix} & P_0 & P_1 & P_2 & P_3 & P_4 \\ S_0 & O^{2-} & \emptyset & O^{2-} & \emptyset & V \\ S_1 & \emptyset & O^{2-} & \emptyset & O^{2-} & \emptyset \\ S_2 & V & \emptyset & O^{2-} & \emptyset & O^{2-} \end{bmatrix} \end{gathered} \tag{20}$$

In the last step, transition process chains are bound to geometric paths of matching size to form KMC transitions. For MMC, setting the exchanging sublattices provides sufficient geometric information to define all equivalent swaps.

The next step is the definition of the energy model. Interactions are geometric paths and thus require an evaluation of symmetry for positional and occupational uniqueness. To this end, MOCASSIN provides three basic model components: (i) the radial interaction cutoff ranges around sites; (ii) interaction filters to control each environment; and (iii) multibody geometries around sites forming interaction clusters. MOCASSIN implicitly defines that the environments of all stable sites are subjected to the same cutoff and filter set to ensure a detailed balance, that is, $E(AB) = E(BA)$. For unstable positions this is not the case, as an unstable A' can only interact with a stable site B during temporary transition state occupation. Using the internal default $E(S_1)$ calculation, the A'B interaction exists purely as an additive contribution to E_{mig} and thus there is no BA' interaction that could compromise $E(S_0)$ and $E(S_2)$ in an unphysical manner. Based on the environment boundaries, MOCASSIN finds the symmetry-reduced set of stable and unstable site pair interactions, and affiliated unique occupation permutations. The same is done for clusters. However, the user defines which reference paths the system should process as clusters.

MOCASSIN translates the transition and energy models into symmetry-reduced parameterization templates, which ensure safe setting of the following quantities: (i) attempt frequencies for each KMC event and affiliated transition rule; (ii) energies of the found pair interaction permutations of stable and unstable sites; and (iii) energies for unique configuration permutations of user-defined clusters. By providing a template selection at simulation building and detecting zero-valued interactions as part of the optimization, MOCASSIN allows the effects of different energy parameterizations to be quickly compared without redefinition of the underlying model.

2.2.5 | Lattice, simulations, and job templates

Analogous to the energy model, the lattice definition system sets boundary conditions for supercell generation. The user defines a default host occupation and additional doping objects describing a particle substitution for a sublattice. Doping objects are combined to allow automatic charge compensation by counter substitutions. Doping levels are later quantified individually for each simulation and the resulting population tables are distributed into a host supercell matrix using uniform random placement. For example, in a first step, barium zirconate $BaZrO_3$ can be doped with trivalent yttrium Y^{3+} on zirconium sites and vacancies V on oxygen sites according to (21).

$$\begin{aligned} &2Zr_{Zr}^{x} + 6O_{O}^{x} + Y_2O_3(s) \\ &\rightarrow 5O_{O}^{x} + 2Y_{Zr}^{\prime} + V_{O}^{\cdot\cdot} + 2ZrO_2(s) \end{aligned} \tag{21}$$

The vacancies can then be partially refilled by hydration (22), creating mobile protons at interstitial positions around the oxygen sites. In this way, creating doped simulation supercells follows chemical intuition and the system allows the definition of large supercell collections without redundant input.

$$V_{O}^{\cdot\cdot} + H_2O(g) \rightleftharpoons O_{O}^{x} + 2H_{i}^{\cdot} \tag{22}$$

The reference simulations are created through simulation base objects that serve as the blueprints for later quantified simulation settings. They define either a KMC or MMC simulation with default parameters, for example, minimal Monte Carlo steps (MCS), and set the allowed transitions during simulation.

The last model step is the definition of job templates. These templates quantify all the parameters of a simulation not defined by the model or the parameterization template. This includes supercell size, doping level, and common variation parameters such as temperature or electric field strength. A simple data hierarchy allows the simulation base default values to be overwritten for a group of simulation jobs. Each group contains multiple jobs that can again overwrite the settings inherited from their parent group. Optional implicit job duplication by multiplicators creates a set of equivalent jobs for statistical sampling of parameter sets.

Ultimately, the simulation build system requires the general model, a parameterization template, and a job template to create processable data for the simulator. All the jobs defined by the job template are written into a single SQLite database, from which the simulator can query the required data on simulation startup.

2.2.6 | Model limitations

MOCASSIN comes with a set of both soft and hard limitations. Soft limits exist for most model components in the form of general data validation rules and can be disabled. They serve the purpose of catching obvious input mistakes, prevent physically meaningless models, and provide notification of potentially critical resource requirements. In contrast, hard limits cannot be disabled trivially. The most notable limitations are: (i) the number of custom species definitions is limited to 63; (ii) geometric paths are limited to eight positions, or eight plus a center site for cluster interactions; and (iii) the maximum number of interaction clusters per position is 256. All limitations represent a balance between performance and memory requirements on the one hand, and model freedom, on the other hand. The species index limitation allows single bytes to represent particles in simulation data structures. While the technical limit is 256, a further restriction to 64 leads to 64-bit mask encoding for species sets. Effectively, the number drops to 63, as the null species $\emptyset$ is implicitly required. The byte representation of species allows occupation states of up-to eight positions to fit a 64-bit integer without any encoding. This enables efficient lookup containers for permutations and direct by-byte particle access. Limiting the number of interaction clusters per position to 256 was chosen to support indexing with a single byte and this restriction does not affect pair interactions. In our experience, these limitations, as well as other restrictions not detailed here, do not effectively restrict model freedom or applicability of the framework.

2.2.7 | The simulator

This section gives an overview of the MOCASSIN MMC/KMC null-event solver written in C. Cycle routines and key implementation choices are presented. Special emphasis is given to the optimization of the rejection principle to process highly interacting systems with acceptable performance.

For the proper statistical generation of pseudorandom numbers, we decided to use the 32-bit version of the permuted congruential generator (PCG32) by M. E. O'Neill,[33] using the recommended XSH RR output function. Prevention of modulo bias for generation of ceiled numbers is implemented as shown within the PCG C/C++ source code. Each simulator instance receives a seeded PCG32 state from the C# model builder to ensure reproducibility.

Efficient implementation of rejection-free KMC is usually done using binary trees for the rate catalogues. However, we expect several issues with this approach for solid electrolytes with large interaction cutoffs. While rate catalogues are comparatively simple to implement for compile-time known problems, realizing a generic solver for dynamic data without corrupting efficiency is more demanding. The effort of tree updating affiliated with rejection-free KMC events is $\langle A \rangle \cdot \log(N)$. Here, N is the number of existing branches and $\langle A \rangle$ is the average number of branches that require updating after each step. Thus, tree updating scales $O(\log(N))$ with system size but has $O(\langle A \rangle)$ dependency comparing different models. For 3D networks with a cutoff range R, we can approximate $\langle A \rangle \propto R^3$, as a direct result of increasing interdependency between events. Thus, we expect the factor $\langle A \rangle$ to quickly become an issue when extending beyond nearest neighbor models. Providing any quantitative prediction is difficult; $\langle A \rangle$ additionally depends on

mobile particle density, memory access is expected to be limiting when the binary trees become large, and synthetic benchmarks rarely yield genuine performance data. Ultimately, we expect the advantages/disadvantages of rejection-free or null-event principles to be case-specific when simulating arbitrary solid electrolytes. Thus, implementation of the MMC compatible null-event routine in MOCASSIN was pursued.

One important issue of C/C# interoperability is an efficient and portable layout for multidimensional arrays. Built-in rectangular arrays in C are a compile-time feature requiring constant sizes. Thus, we implemented a portable, basic rectangular array system for MOCASSIN. A typical row-major layout accesses the data with the products of the dimension size values l_k as shown in (23).

$$\text{Index of } (x_0,\ldots,x_n) = x_n + \sum_{i=0}^{n-1} x_i \cdot \prod_{k=i+1}^{n} l_k \tag{23}$$

This allows the formatting of C array structures in the C# system, which can then be directly passed through the SQLite databases. Within the C code, a set of small access structures, factory functions, and preprocessor macros forms an efficient generic array system, providing type safety and on/off switching of boundary checks with compiler flags.

The implementation of the simulator in MOCASSIN uses a simulation context object to access the simulation data graph. The top-level simulation functions advance either KMC or MMC simulations by one cycle and write the results to the context. In general, the principle behind both MMC and rejection KMC is similar and enables most of the underlying functionality to be designed in a compatible fashion. A flow diagram illustrating the general KMC cycle process with possible cycle outcomes is shown in Figure 3. Aside from statistical rejection or acceptance, there are four other outcomes: (i) skipping, (ii) site blocking, (iii) S_0 instability, and (iv) S_2 instability. The site blocking and skipping events both cause a time step without advancing the system. They represent jump rejection due to geometric blocking or a rate-induced skip, modeling weighted event selection with multiple attempt frequencies in accordance with (5). S_0 or S_2 instability occurs when the S_1 state energy is below the initial or final state energy of the system, respectively. While the latter blocks the migration, the former causes a migration without a timestep, which models the fact that unstable states would relax spontaneously. Using the default energy model (24), S_0 instability events emerge either due to extreme energetic state differences caused by the initial uniform defect distribution or the electric field influence ΔE_{field}, causing E_{mig} to become slightly negative.

$$E_{\text{mig}} = \frac{E(S_2) - E(S_0)}{2} + E_{\text{S1,base}} + \Delta E_{\text{field}} \tag{24}$$

The built-in calculation of the migration energy (24) supports replacement by an external C function. In general, interpretation of how many instability events are to be expected or acceptable is always up to the user.

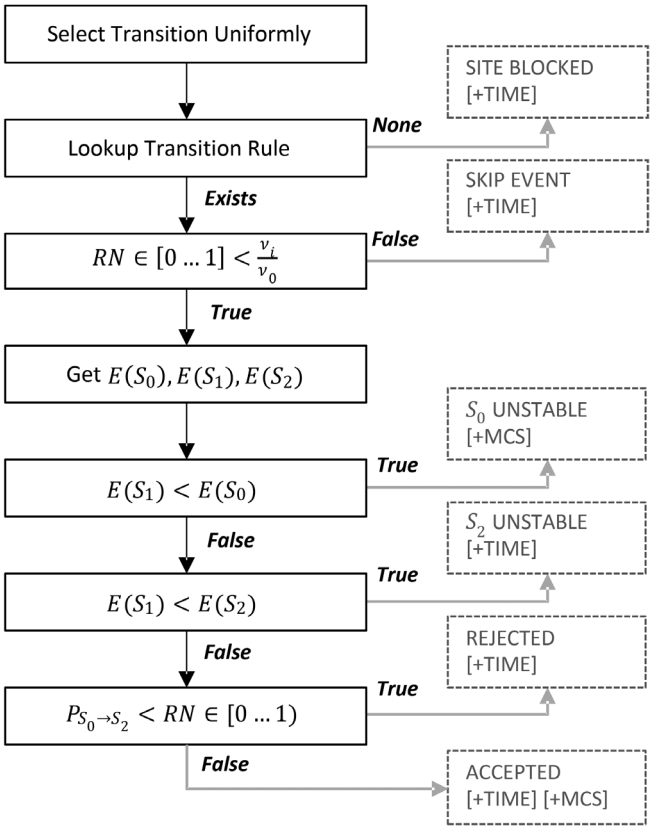

FIGURE 3 Flow diagram showing the basic KMC simulation cycle in MOCASSIN including possible cycle outcomes and consequences (gray)

KMC simulations are preceded by an initial equilibration that performs runtime optimization by dynamically adjusting the probability normalization K_{norm} based on the encountered maximal jump probability P_{max} (25), while correcting the affiliated time step Δt according to (26). Depending on the energy landscape, this step can drastically improve simulation performance. Assuming sufficient equilibration cycles, the statistical risk of unexpected critical overnormalization for the main run is negligible.

$$\begin{aligned} P_{\text{j,norm}} &= K_{\text{norm}} \cdot \exp\left(-\frac{E_{\text{mig}}}{k_{\text{B}}T}\right) \\ &= \frac{1}{P_{\text{max}}} \cdot \exp\left(-\frac{E_{\text{mig}}}{k_{\text{B}}T}\right) \end{aligned} \tag{25}$$

$$\Delta t_{\text{norm}} = K_{\text{norm}} \cdot \Delta t \tag{26}$$

The KMC routine is also capable of deliberate overnormalization, which can be adjusted by the user. This is useful for broad, fragmented jump probability distributions, where rare jumps with a low acceptance rate prevent proper normalization. The user can provide a fixed normalization probability $P_{\text{limit}} \in (0\ldots1)$ that serves as the new upper acceptance limit for the probability comparison. Automatic normalization then only considers the jump probability distribution below the new upper limit (27) and further corrects the time step as given in (28).

$$P_{j,\mathrm{norm}} = \frac{1}{P_{\mathrm{limit}}} \cdot K_{\mathrm{norm}} \cdot \exp\left(-\frac{E_{\mathrm{mig}}}{k_B T}\right) \tag{27}$$

$$\Delta t_{\mathrm{norm}} = \frac{1}{P_{\mathrm{limit}}} \cdot K_{\mathrm{norm}} \cdot \Delta t \tag{28}$$

The MMC simulation cycle is analogous to its KMC counterpart, without steps involving transition states as shown in Figure 4. As it is usually counterproductive and impossible to cache all MMC exchanges for the lattice, MOCASSIN uses a two-step MMC selection process for particle swaps. The selection pool stores MMC events as KMC-like transitions for each selectable particle of the lattice using information independent of the unit cell. Only the fourth coordinate of (0, 0, 0, *p*) is used when choosing a transition/position pair from the pool and defines the partner site. The second step selects the cell offset of the partner site by a uniform random pick of the (z_a, z_b, z_c) data. In this fashion, both KMC and MMC can use one transition/position pair selection pool implementation without violating uniformity of selection in either case.

The solver is mainly optimized by circumventing redundant runtime energy contribution lookups. Naively implemented, MMC and KMC pull contributions to energies or probabilities from data tables for each individual transition balance calculation, accumulating significant lookup overhead in each cycle. This is especially inefficient for null-event approaches, as statistical rejection is the most frequent cycle outcome, which does not affect the state of the system. Thus, MOCASSIN implements event-driven data pushing using an observer-pattern-like style for the entire supercell. Starting a simulation, the system analyzes pair interactions, clusters, and particle mobility in the lattice to separate constant and mutable contributions to the energy landscape. MOCASSIN then builds update instructions for each site (observable) encoding how to update the energy state of interacting sites (observers). During simulation, this dependency network keeps a cached energy landscape of relevant position/particle pairs, which invokes the local updating "lazy", that is, only when the site changes. The immutable contributions are used as the initial base values. Thus, state energy calculation for an event check is a cheap cache lookup operation, including a small correction calculation for the final state. The latter is necessary because calculating $E(S_2)$ based on the S_0 cache entries has virtual self-interactions if transition path particles themselves interact. As an explanation, a simple vacancy mechanism illustrates the issue in (29), where the real particles (R) interact with cache entries (C) that are not valid for all states. Here, the cached-retrieved $E(S_2)$ has a bias of a one AA and one VV virtual self-interaction, with two missing AV interactions that should exist instead.

$$\begin{bmatrix} & P_0(\mathrm{R}) & P_2(\mathrm{C}) & P_2(\mathrm{R}) & P_0(\mathrm{C}) \\ S_0 & \mathrm{A} & \mathrm{V} & \mathrm{V} & \mathrm{A} \\ S_1 & \emptyset & \mathrm{V} & \emptyset & \mathrm{A} \\ S_2 & \mathbf{V} & \mathbf{V} & \mathbf{A} & \mathbf{A} \end{bmatrix} \tag{29}$$

In most cases, the required correction is a constant value for each transition and MOCASSIN thus simply precalculates the values during initialization. The system drastically reduces the overhead of frequent null events in exchange for a minor cost increase of rarer MCS events. Importantly, the theoretical cost of optimized null events scales *O*(1) with local environment sizes. For an additional speed-up of pair interactions, MOCASSIN creates 3D change tables to perform each pair update with a single lookup.

Several additional optimization aspects are implemented in MOCASSIN. An important example is dynamic conversion of energy into Boltzmann probability. Modern C standard library implementations of exp() are fast and custom energy model functions are much more convenient to implement when working with energy values. Therefore, MOCASSIN avoids probability tables and calls exp() once per simulation cycle. However, in some situations high-precision exp() causes significant overhead at little benefit. Thus, MOCASSIN optionally provides the IEEE754-based exponential approximation fexp() by N. N. Shraudolph,[34] using the correction value for minimized root-mean square error (RMSE). The effective error for $P \in [0...1)$ typically vanishes behind statistical noise.

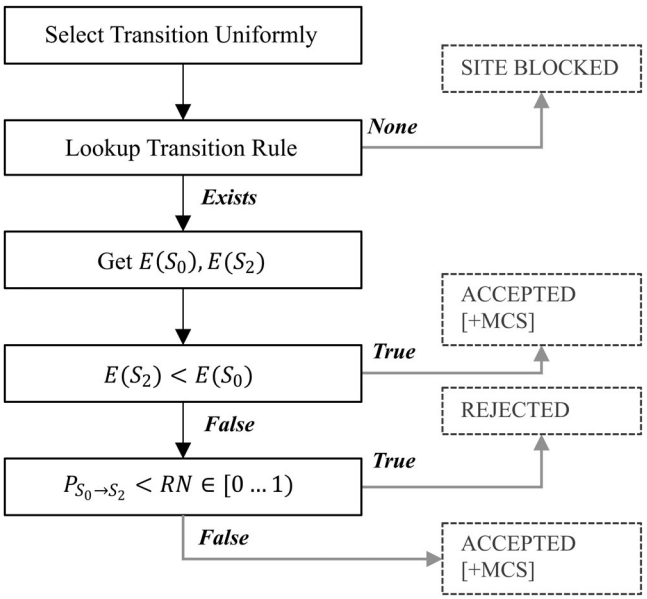

FIGURE 4 *Flow diagram showing the basic MMC simulation cycle in MOCASSIN including possible cycle outcomes and consequences (gray)*

2.3 | Application examples

2.3.1 | Basic vacancy diffusion

Initial validation of MOCASSIN is performed for analytically solvable test cases: (i) 2D square plane, (ii) 2D hexagonal plane, and (iii) 3D simple cubic lattice. The number of possible migration directions N_{dir} per

site in the lattice are four, six, and six, respectively. The possible transitions for each system are given a fixed barrier E_{mig} = 1 eV and a basic attempt frequency $\nu_0 = 10^{13}$ Hz per direction. The simulation temperature is varied from 273 to 1,273 K in steps of 50 K. Supercells have periodic boundary conditions and the vacancy fraction for the supercells is fixed at x_V = 0.01. All unit cells use $a = b = c = 1$ Å, with angles according to the cubic or hexagonal crystal system definition. The theoretical diffusion coefficients for vacancies $D_{V,theo}(T)$ and mobile atoms $D_{m,theo}(T)$ are calculated using (30), (31), and (32) using the correlation factors by Compaan and Haven.[35] Affiliated reference values and results for infinite temperature are summarized in Table 1.

$$D_{V,theo}(T) = D^0_{V,theo} \cdot \exp\left(-\frac{E_{mig}}{k_B T}\right) \tag{30}$$

$$D^0_{V,theo} = \frac{1}{2 \cdot d} \cdot l^2 \cdot \nu_0 \cdot N_{dir} \tag{31}$$

$$D_{m,theo}(T) = f_{cor} \cdot \frac{x_V}{x_m} \cdot D_{V,theo}(T) \tag{32}$$

The Win64 C simulator binaries for testing were built with the GNU GCC 9.2 compiler using the MSYS2/MinGW64 toolchain. The optimization level was set to "-O3" with native tuning enabled. All simulations were performed on a local Win64 machine using a single core of an AMD Ryzen 3700X eight-core processor (512 KB/4 MB/32 MB L1/L2/L3) with 4.3 GHz boost clock and 3,600 MHz DDR4 RAM. The average memory consumption for each run was around 10 MB, allowing all simulation data to fit into the CPU cache. The simulator was run for 10^5 MCS with autonormalization, followed by 10^9 main run MCS for data collection, using probability calculation with exp() and fexp(). The results are illustrated in Figures 5, 6, and 7.

Figure 5 shows perfect agreement between the analytical evaluation of diffusion and the MOCASSIN solver results for the 2D cases, with and without exponential approximation. As case-specific runtime values are difficult to compare, the performance results are provided as solver throughput. For this purpose, the total simulation cycle rate Γ_{tot} and MCS rate Γ_{MCS} are given, that is, the number of transition attempts and successful MCS yielded by the KMC solver per second, respectively. It can be expected that problems with both similar system sizes and complexity yield comparable simulation cycle rates.

As can be seen in Figure 6, the total cycle rate Γ_{tot} is almost equal to the acceptance rate Γ_{mcs}, that is, the pre-run autonormalization shifted both systems to ~99% statistical acceptance rate. The remaining ~1% of jumps events are site blocked by neighboring vacancies in accordance with the vacancy fraction. It can further be seen that exponential approximation yields ~25% higher performance. Calculating correlation factors from the simulated diffusion data using default exp() yields f_{cor} = 0.46 ± 0.01 for the square lattice and f_{cor} = 0.58 ± 0.02 for the hexagonal case. Applying the analytical coefficient $D_{V,theo}(T)$ as a reference for the vacancy movement, replacing the statistically weak data gained from the small vacancy ensemble, gives perfect results of f_{cor} = 0.467 ± 0.001 and f_{cor} = 0.560 ± 0.001. Exponential approximation yields almost equally good results of f_{cor} = 0.468 ± 0.002 and f_{cor} = 0.559 ± 0.003.

The results for the 3D cubic case in Figure 7 show perfect agreement with the analytic solutions as well. The correlation factors

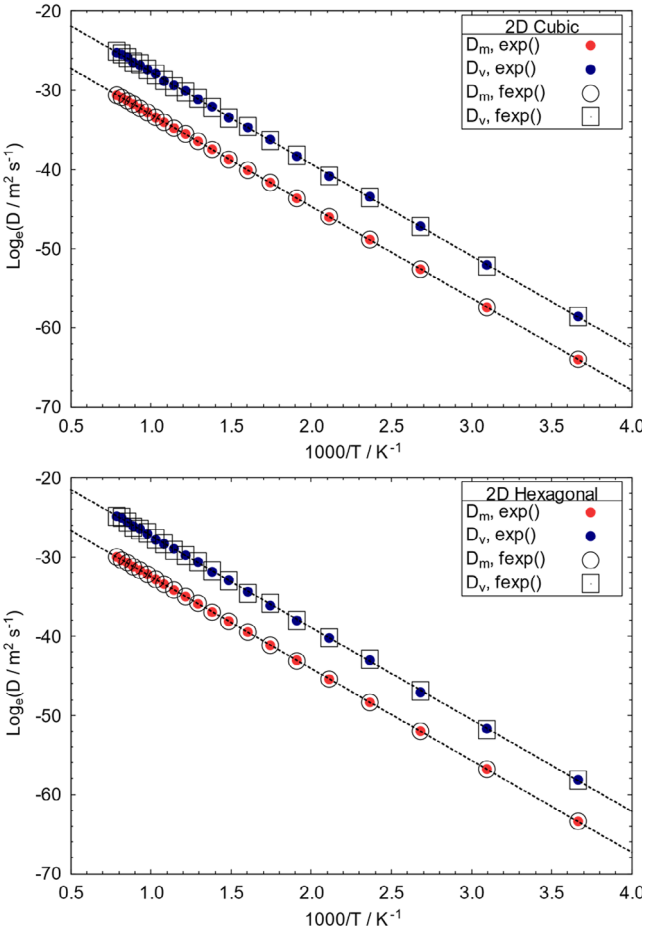

FIGURE 5 Results of 2D vacancy diffusion simulation in a 100 × 100 × 1 cubic supercell (top) and 100 × 100 × 1 hexagonal supercell (bottom) with affiliated analytic solutions (dashed lines). A $P\bar{1}$ pseudocubic unit cell with stable site (0.5, 0.5, 0.5) and a P6 hexagonal unit cell with stable site (0.0, 0.0, 0.5) are used, respectively [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 1 Results of theoretical vacancy diffusion evaluation in different symmetry systems using the correlation factors calculated by Compaan and Haven[35]

Lattice	N_{dir}	f_{cor}	$\frac{\nu_0 \cdot N_{dir}}{10^{13}\ s^{-1}}$	$\frac{l^2}{10^{-20}\ m^2}$	$\frac{D^0_{V,theo}}{10^{-7} m^2 s^{-1}}$	$\frac{D^0_{m,theo}}{10^{-10} m^2 s^{-1}}$
2D square	4	0.46705	4.000	1.000	1.000	4.718
2D hexagonal	6	0.56006	6.000	1.750	1.500	8.486
3D cubic	6	0.65549	6.000	1.000	1.000	6.621

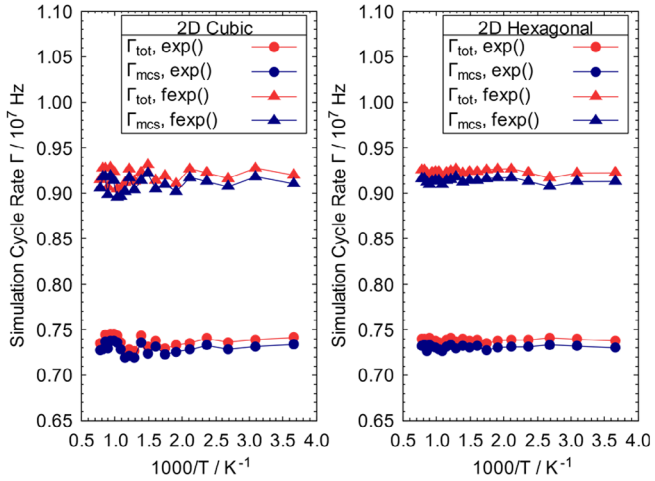

FIGURE 6 Performance results for 2D vacancy diffusion in 2D square lattice (left) and 2D hexagonal (right) network [Color figure can be viewed at wileyonlinelibrary.com]

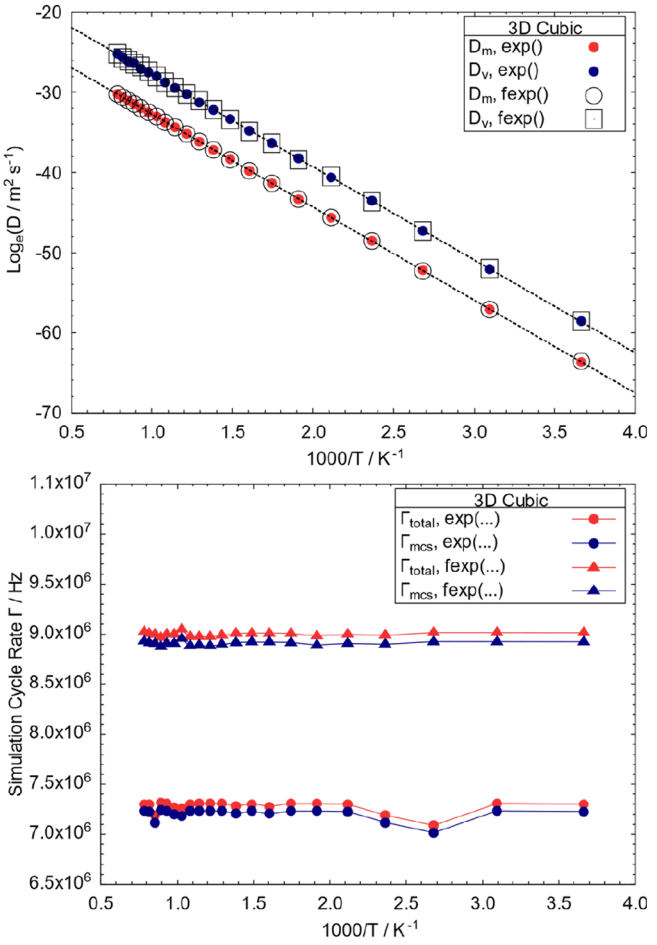

FIGURE 7 Results (top) and benchmark (bottom) of 3D vacancy diffusion in a 100 × 10 × 10 cubic supercell. A P1̄ pseudocubic unit cell with stable site placed at (0.5, 0.5, 0.5) is used. Results are in perfect agreement with the analytic solution (dashed lines) [Color figure can be viewed at wileyonlinelibrary.com]

inferred with $D_{V,theo}(T)$ as the vacancy reference using exp() and fexp() calculation are f_{cor} = 0.650 ± 0.001 and f_{cor} = 0.650 ± 0.003, respectively. Using the third dimension causes an unexpected 2–3% drop in solver performance compared to the 2D cases. This is most likely caused by the MOCASSIN movement tracking system, as the 3D cubic case was defined as three reference migrations in a P1̄ symmetry instead of using a cubic space group, resulting in slightly fragmented movement trackers in the memory.

Overall, both performance and ensemble results of the MC solver fulfill the expectations for a generic rejection routine solving simple, well-normalized problems. As the presented examples do not require the simulation to generate an interaction update network, they show the upper limit of what can be achieved in perfect, best-case scenarios and will be significantly lower for interacting systems.

2.3.2 | Ionic conductivity in yttrium-doped ceria

In previous publications,[19,36,37] we investigated acceptor-doped ceria by DFT and MC. This is an ideal model system for understanding ionic conductivity on the microscopic level and we thus use it as a test case for MOCASSIN. Doping ceria (fluorite structure, space group 225) with trivalent cations, for example, yttrium, by dissolving the affiliated oxide in the ceria host matrix yields mobile vacancies in the oxygen sublattice according to (33).

$$Y_2O_3 \overset{CeO_2}{\rightarrow} 2Y'_{Ce} + 3O^x_O + V^{\cdot\cdot}_O \tag{33}$$

It is known from experiments[38] that yttrium-doped ceria reaches a maximum oxygen ion conductivity around x_Y = 0.1, which was reproduced very well by existing in-house MC solutions and a DFT interaction model as described in a previous publication.[19] The basic migration barriers E_{CeCe} = 0.52 eV, E_{YCe} = 0.57 eV and E_{YY} = 0.82 eV are used, depending on the occupation of the two-cation sites adjacent to the transition site. The pair interaction energies are set according to Reference 19. An external electric field $E_x = 2 \times 10^7$ V/m is applied in the x-direction. A cerium oxide unit cell as shown in Figure 8, with $a = 5.411\,Å$ and Wyckoff sites $\vec{x}_O = (0.25, 0.25, 0.25)$ and $\vec{x}_{Ce} = (0,0,0)$, is used to create $10 \times 10 \times 10$ simulation supercells, including periodic boundary conditions and 24 unique migration paths per cell. The simulation temperature is set to 773 K, the dopant fraction is varied from x_Y = 0.02 to x_Y = 0.30, and 8×10^7 MCS are performed per simulation after 8×10^6 equilibration steps. Figure 8 shows the normalized occurrence of migration barriers recorded with MOCASSIN at higher temperature (1,273 K) to evaluate the expected normalization behavior. It can be seen that a few events with low energy barriers prevent efficient autonormalization and all simulations are thus shifted to a new upper acceptance limit of P_{limit} = 0.01 (equals ∼0.31 eV at 773 K). The shift decreases runtime by a factor of ∼100, introducing a bias indistinguishable from noise.

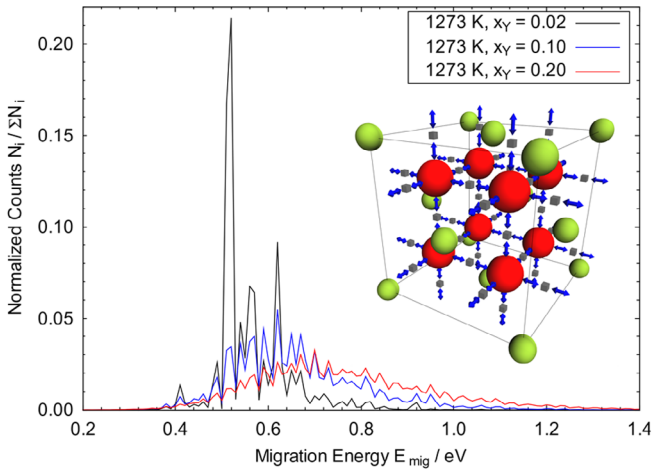

FIGURE 8 The normalized migration barrier histograms (attempts) in the ceria model at 1,273 K. Most jumps occur at E_{mig} > 0.4 eV indicating that normalization up to ~0.3 eV will not affect KMC results to a measurable amount. The insert shows the ceria unit cell as used for KMC simulation of oxygen (red) migration. There are a total of 48 possible nearest-neighbor migrations (blue, 36 shown) per unit cell. Each unstable site (gray boxes) has two neighboring symmetry equivalent ceria (green) sites, which are grouped into clusters defining the E_{edge} base barrier [Color figure can be viewed at wileyonlinelibrary.com]

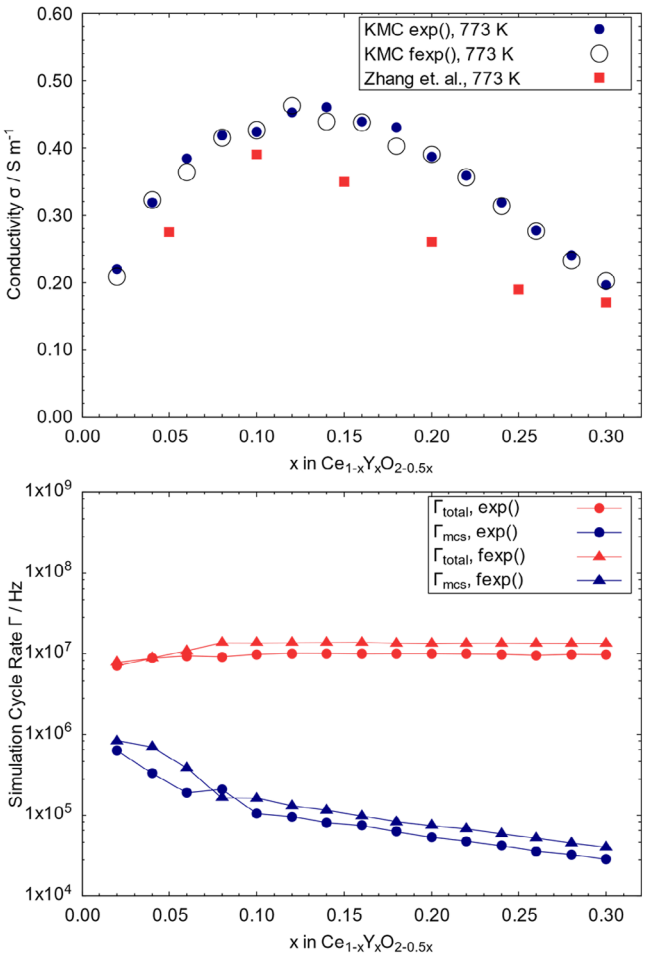

FIGURE 9 Oxygen ion conductivity for $Ce_{1-x}Y_xO_{2-0.5x}$ at 773 K (top) simulated with MOCASSIN compared to experimental results reported by Zhang et al. The performance analysis (bottom) shows the expected decrease of MCS rate with rising yttrium content due to the higher average barrier and broadening of the migration barrier distribution [Color figure can be viewed at wileyonlinelibrary.com]

The simulation results illustrated in Figure 9 (top) are in excellent agreement with experimental values reported by Zhang et al.[38] and no significant differences of the MOCASSIN solver compared to the previously used implementation[19] were observed. Additionally, Figure 9 (bottom) shows that MOCASSIN reaches the objective of *O* (1) null-event scaling with interaction range. The CPU time per simulation cycle increases with MCS rate only, while Γ_{tot} of the simulation caps at ~10 MHz (~13.5 MHz for fexp()) at low Γ_{MCS}. This is comparable to the interaction-free diffusion tests, even though the presented ceria model considers contributions from 148 pair interactions and one 3-body cluster per transition attempt.

2.4 | About program runtime

Concerning expected performance and runtime, it should be noted that MOCASSIN's caching optimization principle cannot circumvent the crucial issue of the rejection routine; broad, flattened rate histograms cause a significant decrease of the success rate, as an efficient bias-free normalization does not exist. In general, the program runtime varies from a few seconds to typically hours or rarely days depending on the complexity of the MC problem and the required number of MCS to reach a steady state. For example, the runtime of the performed ceria simulations using regular exp() calls varies from 3 to 53 min for the lowest to the highest yttrium fraction, respectively.

In theory, the solver algorithm scales *O*(1) with supercell size for both KMC and MMC, however, there are two important factors to consider: (i) the number of interactions affecting each transitions attempt, usually scaling R^3 with the interaction cut-off radii, which increases both the number of energy update calls after an MCS and the memory size of the caching system; and (ii) memory consumption in general, which causes a significant performance loss as soon as the simulation data no longer fits into the CPU cache and the RAM is accessed frequently. Ultimately, the flexibility of MOCASSIN renders it difficult to provide a conclusive prediction about expected runtime for a specific MC problem.

3 | CONCLUSIONS

We introduced the MOCASSIN program as a versatile tool for fixed-position MMC and KMC studies of solid electrolytes. The system supports simulations including multiple mobile species, complex migration

mechanisms, custom cluster definitions, and polaron hopping in fixed-position lattices of arbitrary crystal structures. The graphical interface, job templating systems, and symmetry processing components conveniently assist in screening studies of solid electrolyte materials based on energy models from first-principles calculations. The rejection-based solver achieves practical performance, even for long-range interaction models. The SQLite container concept keeps startup data and solver results together, reducing the file clutter typically associated with parameter sweeps.

We showed that MOCASSIN uses the space group symmetry to automate the translation process from a symmetry-reduced, high-level reference model to an efficient, low-level data layout for high-performance computing. This minimizes the input effort to a minimalistic model set and allows the system to perform various consistency checks and automatic component detections for the user. In combination with the flexible transition definition system and templating approach for parameterizations and job collections, MOCASSIN users can create simulation studies of solid electrolytes efficiently, requiring no experience with Monte Carlo implementation.

The functionality of MOCASSIN's solver system was successfully validated with diffusion in three model systems and comparison of a complex ceria migration model to both experiment and previous MC implementations. It additionally showed that MOCASSIN's optimizations realize the objective of $O(1)$ scaling of null events with environment size, reducing the performance impact of null events in highly interacting systems.

ACKNOWLEDGMENTS

The authors of MOCASSIN acknowledge the work of all creators/contributors of third-party software projects used in the development of MOCASSIN, including SQLite, ReactiveX, SharpDX, Helix Toolkit, NCalc, Json.Net, AvalonEdit, Entity Framework Core, and PCG. Thanks are due to John P. Arnold (Prof. Dr. Manfred Martin's group, RWTH Aachen University, Germany) for his code contributions and Lukas Eisele (Artiso Solutions GmbH, Blaustein, Germany) for his development insights and discussion time. The authors further acknowledge the feedback from all testers and beta version users, and also thank the RWTH Aachen ITC for providing the CLAIX2016 and CLAIX2018 HPC infrastructure. This research was partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), Grant No. GR-5011/1-1. Open access funding enabled and organized by Projekt DEAL.

DATA AVAILABILITY

MOCASSIN is currently a closed-source project, copyrighted by Forschungszentrum Jülich GmbH/DE and RWTH Aachen University/DE. Until the official release of the software and/or the source code, end user versions, including NuGet packages for .NET Core project integration, can be obtained from the authors free of charge for research purposes upon personal request. All components are provided "as is" without any warranties.

ORCID

Sebastian Eisele https://orcid.org/0000-0001-7138-0627
Steffen Grieshammer https://orcid.org/0000-0001-7583-6417

REFERENCES

[1] N. Metropolis, S. Ulam, *J. Am. Stat. Assoc.* **1949**, *44*, 335.
[2] M. D. Towler, *Phys. Stat. Sol. B* **2006**, *243*, 2573.
[3] J. M. Hammersley, D. C. Handscomb, *Monte Carlo methods*, Springer, Dordrecht **1964**.
[4] M. H. Kalos, P. A. Whitlock, *Monte Carlo Methods*, 2nd ed., Wiley, Hoboken **2009**.
[5] K. Binder, *Monte Carlo Methods in Statistical Physics*, 2nd ed., Springer, Berlin **1986**.
[6] R. Y. Rubinstein, D. P. Kroese, *Simulation and the Monte Carlo Method*, 3rd ed., John Wiley & Sons, Hoboken, New Jersey **2016**.
[7] M. P. Allen, D. J. Tildesley, *Computer Simulation of Liquids*, Clarendon Press, Oxford **1987**.
[8] K. Binder, A. Baumgärtner, *Applications of the Monte Carlo Method in Statistical Physics*, 2nd ed., Springer-Verlag, Berlin, New York **1987**.
[9] D. Frenkel, B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*, 2nd ed., Academic Press, San Diego, London **2002**.
[10] W. Krauth, *Statistical Mechanics: Algorithms and Computations*, Oxford University Press, Oxford **2006**.
[11] D. C. Rapaport, *The Art of Molecular Dynamics Simulation*, 2nd ed., Cambridge University Press, Cambridge **2004**.
[12] G. E. Murch, A. S. Nowick, *Diffusion in Crystalline Solids*, Academic Press, Orlando, Florida **1984**.
[13] R. Jurij, L. Per, *J. Comput. Chem.* **2015**, *36*, 1259.
[14] J. A. Purton, J. C. Crabtree, S. C. Parker, *Mol. Simul.* **2013**, *39*, 1240.
[15] M. J. Hoffmann, S. Matera, K. Reuter, *Comput. Phys. Commun.* **2014**, *185*, 2138.
[16] M. Leetmaa, N. V. Skorodumova, *Comput. Phys. Commun.* **2014**, *185*, 2340.
[17] C. Garcia Cardona, E. B. Webb III; G. J. Wagner, V. Tikare, E. A. Holm, S. J. Plimpton, A. P. Thompson, A. Slepoy, X. W. Zhou, C. C. Battaile, M. E. Chandross, Sandia Report SAND2009-6226, **2009**.
[18] F. M. Draber, C. Ader, J. P. Arnold, S. Eisele, S. Grieshammer, S. Yamaguchi, M. Martin, *Nat. Mater.* **2020**, *19*, 338.
[19] S. Grieshammer, S. Eisele, J. Koettgen, *J. Phys. Chem. C* **2018**, *122*, 18809.
[20] J. Schuett, T. K. Schultze, S. Grieshammer, *Chem. Mater.* **2020**, *32*, 4442.
[21] A. F. Voter, Introduction to the kinetic Monte Carlo method. in *Radiation Effects in Solids*, Springer, Dordrecht **2007**, p. 1.
[22] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, *J. Chem. Phys.* **1953**, *21*, 1087.
[23] A. van der Ven, G. Ceder, M. Asta, P. D. Tepesch, *Phys. Rev. B* **2001**, *64*, 184307.
[24] G. Murch, *Solid State Ionics* **1982**, *7*, 177.
[25] E. J. W. Verwey, *Physica* **1935**, *2*, 1059.
[26] N. F. Mott, R. W. Gurney, *Electronic Processes in Ionic Crystals*, 2nd ed., Oxford University Press, Oxford **1950**.
[27] L. Onsager, *Phys. Rev.* **1931**, *37*, 405.
[28] L. Onsager, *Phys. Rev.* **1931**, *38*, 2265.
[29] R. Kubo, *J. Phys. Soc. Jpn.* **1957**, *12*, 570.
[30] M. S. Green, *J. Chem. Phys.* **1954**, *22*, 398.
[31] A. R. Allnatt, E. L. Allnatt, *Philos. Mag. A* **1984**, *49*, 625.
[32] M. I. Aroyo, *International Tables for Crystallography. Volume A, Space-Group Symmetry*, 6th ed., Wiley, Chichester, West Sussex **2016**.
[33] M. E. O'Neill, *HMC-CS-2014-0905*, Claremont, CA **2014**.
[34] N. N. Schraudolph, *Neural Comput.* **1999**, *11*, 853.
[35] K. Compaan, Y. Haven, *Trans. Faraday Soc.* **1956**, *52*, 786.

[36] S. Grieshammer, B. O. H. Grope, J. Koettgen, M. Martin, *Phys. Chem. Chem. Phys.: PCCP* **2014**, *16*, 9974.
[37] J. Koettgen, S. Grieshammer, P. Hein, B. O. H. Grope, M. Nakayama, M. Martin, *Phys. Chem. Chem. Phys.: PCCP* **2018**, *20*, 14291.
[38] T. Zhang, J. Ma, L. Kong, S. Chan, J. Kilner, *Solid State Ionics* **2004**, *170*, 209.

How to cite this article: Eisele S, Grieshammer S. MOCASSIN: Metropolis and kinetic Monte Carlo for solid electrolytes. *J Comput Chem*. 2020;41:2663–2677. https://doi.org/10.1002/jcc.26418

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of this article.