Efficient Kinetic Monte Carlo Simulation

Tim P. Schulze

Mathematics Department 121 Ayres Hall 1403 Circle Drive Knoxville, TN 37996-1300 USA

Abstract

This paper concerns Kinetic Monte Carlo (KMC) algorithms that have a single-event execution time independent of the system size. Two methods are presented—one that combines the use of inverted-list data structures with rejection Monte Carlo and a second that combines inverted lists with the Marsaglia-Norman-Cannon algorithm. The resulting algorithms apply to models with rates that are determined by the local environment but are otherwise arbitrary, time-dependent and spatially heterogeneous. While especially useful for crystal growth simulation, the algorithms are presented from the point of view that KMC is the numerical task of simulating a single realization of a Markov process, allowing application to a broad range of areas where heterogeneous random walks are the dominate simulation cost.

Key words: Kinetic Monte Carlo, stochastic simulation, Markov process
PACS:

1 Introduction

The term "Kinetic Monte Carlo" (KMC) often refers to the stochastic simulation of crystal growth and evolution[1–3], where atoms are deposited on or hop between predefined lattice locations with rates that depend on the local crystal configuration. As explained further below, this is closely related to other types of dynamic Monte Carlo simulations, where the distribution/ensemble one is sampling evolves in time. A distinct feature of KMC, however, is that the distribution of rates is derived from, or coupled to, the evolution of an underlying state-space that is spatially heterogeneous. This lends KMC simulations a character that resembles the solution of space-time partial differential equations. While the algorithms presented below were developed with application to crystal growth in mind, they are presented from the point of view that KMC is the numerical task of simulating a single realization of a Markov process. These same algorithms should therefore be applicable to a broad range of systems that are modeled as heterogeneous random walks.

The specific algorithms under consideration have single event execution times that are independent of the size of the system. As explained further below, this is only possible when the required changes to the system state are in some sense local. These algorithms generalize a known algorithm that uses an inverted-list data structure to achieve a fixed cost for local updates in the special case where the model involves a small set of distinct rates [4,5]. To achieve algorithms that work for arbitrarily large sets of rates, an inverted-list data structure is combined with one of two standard Monte Carlo techniques: rejection or the (rejection-free) Marsaglia-Norman-Cannon [6,7] algorithm for sampling a fixed distribution/ensemble. The first of these has been applied to the simulation of epitaxial growth [8], but only briefly discussed; the second is entirely new. We begin by introducing KMC from a general point of view and reviewing the most commonly used algorithms.

2 KMC algorithms

Kinetic Monte Carlo models usually fall into the category of discrete-space, continuous-time Markov processes, where the system passes through a sequence of states $\{x_{t_k} \in \mathcal{X}\}$ drawn from a model dependent *state-space* $\mathcal{X}$ at transition times $\{t_0 < t_1 < \ldots < t_k < \ldots\}$. Equivalently one can view the sequence $\{x_k\}$ as a Markov chain and associate with this an inhomogeneous Poisson process with *rate* $Q(t)$ that generates a sequence of waiting times $\{\delta t_k\}$ for the intervals between transitions. The goal of KMC simulations is to produce a sample of these two sequences from the set of all possible realizations as quickly as possible. Generating the first of the two sequences is the principal topic of this paper; the second is relatively straight-forward, although omitting or approximating it can sometimes lead to considerable computational savings.

Common examples for the state-space include the set of occupation arrays $\mathcal{L} = \{L = \{L_{ijk} \in \{0, 1\}\}\}$ describing a simple cubic crystal or the set of height arrays $\mathcal{H} = \{H = \{H_{ij} \in \mathbb{Z}\}\}$ describing the surface of the crystal. We will represent the general case as $\mathcal{X} = \{X_i\}$ using a single index to enumerate the states. The sequence $\{x_k = X_{i_k}\}$ can then be mapped onto a sequence of these state-space indices $\{i_k\}$. The model is completed by a matrix $\{q_{ij} \geq 0\}$ that gives the expected *rates* for the system to move from an arbitrary state X_i to X_j. The possible outcomes are assumed independent, so that the rates sum to give

$$Q = \sum_{X_j \in \mathcal{X}} q_{ij},$$

the rate for the overall process at the current time step. The rates q_{ij} may depend on time; when they do not, the Markov process is stationary. Equiva-

lently, transition *probabilities* can be defined from these rates by

$$p_{ij} = q_{ij}/Q \quad \Rightarrow \quad \sum_{X_j \in \mathcal{X}} p_{ij} = 1.$$

Since transitions from X_i to itself do no affect the dynamics one can take $q_{ii} = p_{ii} = 0$.

For KMC, the transition matrix $\{q_{ij}\}$ is typically sparse, as most states are not accessible from a given state X_i. At any given time-step, we can therefore restrict our attention to a much smaller set of *events* where the transition rate q_{ij} out of the current state is nonzero. Thus, while events can be identified by a pair of state-space labels (i, j), we will find it more useful to enumerate the set of currently accessible states $\{E_n \in \mathcal{X}\}_{n=1}^{N}$ or, equivalently, the corresponding set of possible changes to the current state $\{\Delta X_n\}_{n=1}^{N}$ using an *event label*.[1] This leads naturally to a corresponding *rate list* $\{q_n\}_{n=1}^{N}$. Note that the rate list $\{q_n\}$ will typically change even when the transition matrix $\{q_{ij}\}$ is stationary and/or the number of events N is constant.

In addition to sparse transition matrices, most KMC models are *local* in the sense that the change in state is spatially localized. For example, if the state-space is the set of singly-defined surfaces for a simple cubic lattice $\mathcal{H}$, the events in a local model typically correspond to changes in the height H_{ij} confined to a region, $|i - i^*|^2 + |j - j^*|^2 < K^2$, that lies within K lattice sites of a particular lattice point (i^*, j^*). If the model does not have this feature, the best one can hope for is an $O(N)$ update cost, as every rate in the rate list must be updated and the cost of updating the system state will typically be $O(N)$ as well. The algorithms discussed in Section 3 are of little use in the latter situation, as they are aimed at providing a fixed update cost for local models.

In order to build efficient algorithms, it is important that the event labels reflect the local nature of the model in two ways: 1) a minimal subset of rates should require changing after any given event, and 2) this subset should be identifiable from the event labels, *i.e.* the event labels should tell you where the "neighboring" events are. This implies explicit knowledge of the mapping between the event numbers and events $n \leftrightarrow \Delta X_n$. In a simple case like the state-space $\mathcal{H}$, we can permanently associate events with specific lattice coordinates (i^*, j^*), so that the mapping never changes and can be computed using formulas or stored. Note that the lattice coordinates (i^*, j^*) refer to a single lattice site, where as the state-space coordinates (i, j) used above number entire states. More complicated cases may require updating the mapping $n \leftrightarrow \Delta X_n$ in some fashion. Below, we shall assume the existence of such a map and refer to event labels using a single index n.

[1] the state-space is typically a vector space so that $\Delta X \in \mathcal{X}$

The essential tasks involved in generating the sequence of states $\{x_k\}$ are sampling from a discrete distribution $\{q_n\}$ (see Figure 1) and updating various data structures to reflect the chosen event. These tasks are repeated at each time step. The second step involves updates to both structures needed for the sampling algorithm and structures associated with the rate list $\{q_n\}$ and system state x_k. It is the latter task that distinguishes KMC from other dynamic Monte Carlo simulations, where the distribution $\{q_n\}$ need not be tied to a state space. Dynamic Monte Carlo is itself distinct from stationary Monte Carlo simulations and Monte Carlo integration, where one repeatedly samples the same distribution and can therefore afford much higher overhead. Stationary Monte Carlo is much more studied [9], offering many techniques which can potentially be adapted to the non-stationary case and to KMC.

Fig. 1. Like other MC methods, KMC must repeatedly sample from discrete distributions/ensembles. For KMC, this distribution corresponds to a list of rates $\{q_n\}$ that typically changes from one time-step to the next and a corresponding set of events $\{\Delta X_n\}$ that represent local changes to the system state.

We are interested in systems where the number of events N is very large, the number of time-steps is even larger, and ensemble averaging is often desirable—thus, the need for fast algorithms. The first condition, in particular, may not apply so broadly. In the chemical kinetics literature, for example, where the system is assumed to be continuously stirred, events correspond to distinct chemical processes and the number of these is usually less than 100. We shall address algorithms which are, in principle, exact, although they may correspond to a model that is an approximation. In many applications, we have already mentioned that it is common to use some form of nearest-neighbor model where a limited number of rates $K << N$ are affected by any given transition. Many of the algorithms we discuss exploit this fact and it is assumed in our assessments of computational costs . It is also possible to develop algorithms that rely on further approximations to the underlying stochastic process [12–14], but we do not address this here.

There are two well-known basic strategies for KMC simulation which we shall refer to as *rejection* and *rejection free* algorithms. The first method relies on repeatedly sampling from a uniform distribution and correcting for this by rejecting an appropriate number of events. One can therefore directly implement any number of schemes for generating uniformly distributed random numbers

and there is essentially no manipulation of data involved. The weakness of this algorithm lies in the cost of generating extra random numbers, which becomes intolerable for systems with highly disparate rates. Rejection-free algorithms, which are known variously as the Bortz-Kalos-Lebowitz (BKL) [4], N-Fold Way [2], or Gillespie algorithm [10,11] in different communities, take what can be viewed as the opposite strategy: they avoid rejection entirely by sampling uniformly from the interval $[0, Q)$ rather than $[0, N)$, and determining which sub-interval $[q_n, q_{n+1})$ the random number falls into. Note that the analogous task for rejection based algorithms can be accomplished by simply rounding up to an integer, whereas the rejection-free techniques require some form of searching and/or sorting. There is a surprising variety of approaches to the latter task. Next, we review some established approaches and then, in Section 3, introduce two new algorithms.

2.1 Review of KMC algorithms

The basic KMC algorithm consists of two steps: event selection and updating of data structures. If one is also generating the sequence of waiting times $\{\delta t_k\}$, an independently generated random number $u \in (0, 1]$ is transformed to an appropriately distributed waiting time $\delta t_k = -(1/Q)\ln(u)$. It is frequently sufficient, however, to accumulate time using expected waiting time $1/Q$ or to simply keep track of the number of iterations.

2.1.1 Rejection

We begin by reviewing the simplest rejection-based algorithm. To select an event, let $\hat{q}$ be an upper bound, preferably sharp, for the current set of rates $\{q_n\}$ and perform the following steps:

Algorithm 1

(1) Choose a random number $r \in [0, N)$,
(2) Consider $n = \mathrm{Int}(r) + 1$,
(3) Select event n if $n - r < q_n/\hat{q}$,
(4) Repeat until successful.

Assuming the model has the local property discussed above, the cost of this algorithm is proportional to the ratio of attempted to accepted events. We call the inverse of this the *efficiency* $\mathcal{E} = Q/\hat{Q}$ of the simulation, with Q being the exact sum of rates and $\hat{Q} = N\hat{q}$ the overestimate. The rejection method thus

[2] The N here refers to the number of distinct rates, which we call M below, and does not indicate a factor of improvement in performance over the rejection algorithm.

becomes extremely inefficient when $Q << N\hat{q}$, a situation frequently encountered, for example, when rates are derived from a Boltzmann-like distribution:

$$q \propto \exp\left(-\Delta U/k_B T\right)$$

that depends sensitively on an energy barrier ΔU scaled by temperature. If one assumes that a typical rate $\bar{q}$ and the upper bound $\hat{q}$ approach a limit with increasing system size, then the efficiency is independent of system size and this algorithm has a fixed, but typically large, cost as $N \to \infty$. Indeed there are many variations on the rejection algorithm that scale well with increasing system size, but these statements are of little practical value unless they can be combined with a statement addressing efficiency.

The inefficiency when there is one or more large rates q_n is analogous to the problem that occurs in MC integration of a sharply peaked function. When integrating, one corrects for this by using some form of "importance" sampling: concentrating the sample points in the region of the peak. For example one could partition the rates into two sets—one where the rates are large and another where they are small—and apply the rejection method with two estimates $\hat{q}_L$ and $\hat{q}_S$. As we shall see in Section 3, this generalizes to something analogous to Lebesgue integration. The partitioning clearly requires more work and, unlike the integration example, in KMC the distribution changes after each sample is taken so that at least some of this work must be repeated. Strategies for doing this ultimately rely on techniques for rejection-free sampling, which are reviewed next.

2.1.2 Rejection-free

Rejection-free algorithms [4]-[11] are more common in KMC due to highly disparate rates. A basic version of this algorithm requires these steps:

Algorithm 2

(1) Calculate the sum $Q = \sum_{n=1}^{N} q_n$, retaining the partial sums Q_n.
(2) Choose a random number $r \in [0, Q)$,
(3) Search the list of partial sums until $Q_{n-1} \leq r < Q_n$.
(4) Select event n.

Since Q in step 2 is the (numerically) exact sum of the rates, the efficiency of this algorithm $\mathcal{E} = 1$ and every attempt is successful, but the number of operations involved in each attempt is proportional to the number of events N, involving the generation of a random number and $O(N)$ floating point operations.

The cost of each iteration can be reduced if the rates are *sorted*, from largest to smallest, to favor early termination of the search in step 3. Techniques that accomplish something along these lines are incorporated into the algorithms discussed in Section 3. For local models, one can greatly improve upon the basic rejection-free algorithm by using a *partitioned* search strategy. A simple version of this is to separate the rates into to M sets, compute sums $Q_1, Q_2, \ldots, Q_M$, apply Algorithm 2 to these sets and apply it a second time to the elements of the selected set [15]. This two-level procedure can be repeated, leading to a large number of possible tree-searching algorithms, including options where branches have an unequal number of sub-levels.

The binary search, where one repeatedly subdivides events into two equal sets to the greatest extent possible, is popular. For a detailed description of how to implement this method, see Blue *et. al.* [16]; we will restrict our discussion here to an evaluation of the computational cost of the method. If we let $\hat{N}$ be the smallest power of 2 greater than N, this search can be implemented as an L-level binary search, where $L = \log_2 \hat{N}$ by setting the extra rates, where $n > N$, to zero. In this method, one must make one floating point decision per level and for each of the K rates that change, one must perform L sums from the bottom of the tree to the root. Thus, the cost of this method is $O(\log N), \quad N \to \infty$.

A frequently occurring special case is when the number of distinct rates $M << N$ is small. One often finds $M \approx K$ because both the number of distinct rates and the number of neighbors influenced by a transition depend on the definition of "nearest" neighbor. This case is amenable to a "binning" algorithm, where a set of M *event lists* $\{\{e_{ml}\}_{l=1}^{c_m}\}_{m=1}^{M}$ contain event labels corresponding to common rates r_m and multiplicities c_m. Apparently overlooked, at least in the epitaxy literature, is that this is the approach adopted in the original algorithm presented by Bortz, Kalos and Lebowitz [4] and that this approach actually scales better than the binary search as the system size becomes large. This is pointed out in Schulze [5], where the technique is rediscovered and a direct comparison of the two methods is made. Briefly, this data structure allows the sum of rates $Q = \sum_{m=1}^{M} c_m r_m$ to be computed more efficiently and a 2-level method requires no searching at the final level, as the rates within that level are uniform. When $K > 1$, however, one must locate the neighbors within the sub-lists in order to update them. Searching the lists is, at best, an $O(N^{1/2})$ operation and this is the reason the method is sometimes viewed as being less efficient than the binary search [3]. However, this too can be done efficiently by using an *inverted-list* data structure e_n^{-1} to store the event list coordinates (m, l) of event n. The update cost is then independent of the number of events N, scaling instead with the number of distinct rates M, as maintaining the binned event lists and inverted event list can be done without searching. This method can, itself, be generalized to a multi-level technique if the number M is also large. The inverted list is the es-

sential tool needed to build the strategies presented in the next section. These algorithms will address the general case of arbitrary rates that do not cluster at particular values.

3 Fixed cost algorithms for arbitrary rate profiles

As emphasized above, it is not sufficient to simply indicate a scaling with increasing system size, $N \to \infty$, as it is easy to invent methods that scale well, but necessarily involve a large number of operations, rendering the methods less than optimal for the size of problem one actually wishes to simulate. For example, we have seen that the simplest rejection method of sampling from a uniform bounding distribution is $O(1)$, but performs poorly due to disparate rates. We now discuss two general-purpose strategies that have a small, fixed cost with increasing system size.

3.1 Inverted-list rejection algorithm

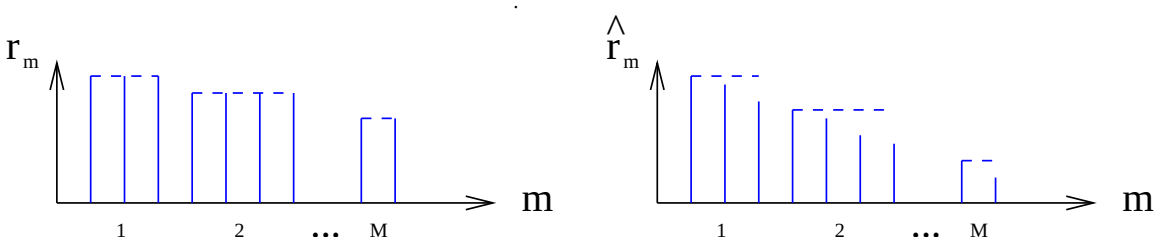

Fig. 2. Illustration of the reorganization of the rate list $\{q_n\}$ into a) ordered sets of distinct rates $\{r_m\}$ when this is possible, and b) ordered sets of rate categories $R_m = \{r|\hat{r}_{m-1} \leq r < \hat{r}_m\}$ when it is not.

There are many ways to combine rejection with one or more levels of a tree search. The first method we discuss generalizes the two-level inverted-list algorithm [4,5] to the case where the number of distinct rates is arbitrary. In the special case of this algorithm discussed above, it is useful to reorganize the rates $\{q_n\}$ into an ordered set of distinct rates $\{r_1 > r_2 > \ldots r_m > \ldots > r_M\}$ with multiplicities $\{c_m\}$. Similarly, as illustrated in Figure 2, we will now take $M << N$ to be a smaller number of rate *categories* $R_m = \{r|\hat{r}_{m-1} \leq r < \hat{r}_m\}$, representing c_m rates, and implement rejection on the lower level of the search by using the upper bounds $\hat{r}_m$.

The algorithm requires these steps:

Algorithm 3

(1) Compute the partial sums $\hat{R}_m = \sum_{i=1}^{m} \hat{r}_i c_i$

(2) Generate a random number $r \in \left[0, \hat{R}_M\right)$.
(3) Search the list of partial sums until $\hat{R}_{m-1} \le r < \hat{R}_m$,
(4) Select event e_{ml} from this sub-list by computing

$$l = \mathrm{Int}\left(\frac{(\hat{R}_m - r)}{\hat{r}_m}\right) + 1.$$

(5) Reject (goto 2) if

$$\mathrm{Frac}\left(\frac{(\hat{R}_m - r)}{\hat{r}_m}\right) \ge \frac{q_{ml}}{\hat{r}_m}$$

(6) For events that have their rates changed from category $\hat{R}_{m_i}$ to $\hat{R}_{m_f}$:
   (a) Move them to the end of sublist m_f; add one to c_{m_f}; update $\{e_n^{-1}\}$.
   (b) Move the event listed as $e_{m_i c_{m_i}}$ into the vacated position in sublist m_i; reduce c_{m_i} by one; update $\{e_n^{-1}\}$.

For a given set of rates, the expected number of attempts per accepted event is again the reciprocal of the efficiency:

$$\mathcal{E} = \frac{Q}{\hat{R}_M} = \frac{Q}{Q + \sum_{m=1}^{M}\sum_{l=1}^{c_m}(\hat{r}_m - q_{ml})}$$

which depends in a complicated way on the partitioning of the rates. The sums in the denominator of this expression are reminiscent of the error formulas for a numerical quadrature, with the important difference that the approximation to the true set of rates must be an upper bound.

Assuming these estimates, the computational cost can be minimized by finding the number of categories M and placement of category boundaries $\{\hat{r}_m\}$ that minimize

$$\min_M \left[M \min_{\{\hat{r}_m\}} \frac{Q + \sum_{m=1}^{M}\sum_{l=1}^{c_m}(\hat{r}_m - q_{ml})}{Q} \right].$$

The inner minimization problem will minimize the number of random numbers used per accepted move for a given number of categories M. As explained earlier, the cost of a linear search through the categories, which must be done for both accepted and rejected moves, scales with M, so that the total cost of the algorithm is increased by this factor and this must also be considered in the overall optimization of the algorithm. It is anticipated that the optimal M will not be large; if it were large, one could use a binary search on the categories and the factor of M would then become $\log M$.

The performance of this algorithm can be analyzed further for special cases, but in practice is model-dependent, requiring choices that work well for typical rate profiles $\{q_n\}$, as these change from one event to the next. For a fixed set of choices for M and $\{\hat{r}_m\}$, the inverted-list method will always outperform a binary search in the limit $N \to \infty$, but this will not be the case at low values

of N, as the inverted-list method has somewhat higher overhead. The exact size of N for which this cross-over occurs and the factor of improvement for larger N will depend on how efficiently the partitioning is done. Indeed, it also depends on how efficiently all aspects of the algorithm are implemented. An improvement that reduces an operation performed by both algorithms—a faster random number generator, for example—will enhance the factor of improvement of the faster method. In an application where N is large and the partitioning is difficult to optimize, one may find the method discussed below to be preferable.

3.2 Digit sorting algorithm

In the above strategy, an interval $[0, \hat{Q})$, is partitioned into subintervals corresponding to events with approximately equal rates. The partitions are then repartitioned into equal intervals, one for each event. It is the equal sized partitions at the lower level that allow for search-free event selection. In this section, we adapt the Marsaglia-Norman-Cannon [6,7] digit-sorting strategy for sampling a stationary distribution so that it may be applied to KMC. The key once again is to rely on the inverted list technique, so that one can avoid repeating the considerable overhead that is required to initialize this algorithm.

The idea here is to partition the interval $[0, Q)$ into subintervals that correspond to bits in the binary representation of the rates q_n. Each of the subintervals is again repartitioned into equal intervals, one for each event with a nonzero bit corresponding to that interval, but an event may now be represented in more than one of the original partitions. Combining this with an inverted list for each bit will result in a rejection-free method with a fixed cost as the number of events increases. A modified version of this algorithm can be made to work with decimal (or any other base) integers.

Let $q_{n,b} \in \{0, 1\}$ be the bth bit in the B-bit binary representation of the transition rate q_n represented to a fixed binary accuracy with respect to a floating binary point and let $c_b = \sum_n q_{n,b}$ be the number of rates with non-zero bth bits. We now maintain B event lists $e_{l,b}$, one for each bit, that compactly store event labels n of the c_b events with non-zero bth bits and B inverted event lists $e^{-1}_{n,b}$ that indicates where, and if, event n is represented in event sublist b.

The algorithm requires these steps:

Algorithm 4

(1) Compute/update the partial sums $S_b = \sum_{i=1}^{b} c_i * 2^{B-i}$,

(2) Generate a random number $r \in [0, S_B)$.
(3) Search the list of partial sums until $S_{b-1} \leq r < S_b$,
(4) Select event $e_{l,b}$ from list b by computing

$$l = \text{Int}\left(\frac{r - S_{b-1}}{2^{B-b}}\right) + 1$$

(5) For events that have one or more bits $q_{n,b}$ of their rates change, do one of the following:
   (a) Add to the end of list b; add one to c_b; update $e_{n,b}^{-1}$, or
   (b) Remove event $e_{l,b}$ from list b; move the event listed as $e_{c_b,b}$ into the vacated position; reduce c_b by one; update both elements of $e_{n,b}^{-1}$.

This algorithm has substantially larger memory requirements than Algorithm 3, as one must maintain an inverted event list corresponding to each bit instead of a single inverted list. Similarly, the fixed update cost includes an inner loop over the number of bits B in the representation of the rates. These disadvantages are offset by lack of rejection and could potentially be further mitigated by exploiting the binary representation of rates to perform arithmetic and store events more efficiently.

4 Conclusion

In summary, this paper discusses two algorithms for KMC simulations designed to give a small, fixed single-event execution time as the system size is increased. These algorithms address systems with large event lists and disparate rates that do not cluster at a small set of discrete values. The algorithms will be applicable to many systems that are modeled using inhomogeneous random walks with rates that depend on the local environment.

References

[1] F.F. Abraham and G.W. White, "Computer simulation of vapor deposition on two-dimensional lattices," *J. Appl. Phys.* **41** (1970) 1841–1849.

[2] G.H. Gilmer and P. Bennema, "Simulation of crystal growth with surface diffusion," *J. Appl. Phys.* **43** (1972) 1347–1360.

[3] M. Kotrla, "Numerical simulations in the theory of crystal growth," *Comp. Phys. Comm.* **97** (1996) 82–100.

[4] A.B. Bortz, M.H. Kalos and J.L. Lebowitz, "New algorithm for Monte-Carlo simulations of Ising spin systems," J. Comput. Phys. **17** (1975) 10–18.

[5] T.P. Schulze, "Kinetic Monte-Carlo with minimal searching," *Phys. Rev. E* **65** (2002) Art. No. 036704.

[6] G, Marsaglia, "Generating discrete random variables in a computer," *Comm. of the ACM* **6** (1963) 37–38.

[7] J.E. Norman and L.E. Cannon, "A computer program for the generation of random variables from any discrete distribution," *J. Stat. Comput. Simul.* **1** (1972) 331–348.

[8] T.P. Schulze, "A hybrid scheme for simulating epitaxial growth" *J. Cryst. Growth* **263** (2004) 505–615.

[9] J.E. Gentle, *Random Number Generation and Monte Carlo Methods,* Springer (2003).

[10] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *J. Phys. Chem.* **81** (1977) 2340–2361.

[11] D. T. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions," *J. Comp. Phys.* **22** (1976) 403–434.

[12] M. Rathinam, L. R. Petzold, Y. Cao, and Daniel T. Gillespie, "Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method," *J. Chem. Phys.* **119** (2003) 12784–12794.

[13] W. E, D. Liu, and E. Vanden-Eijnden, "Nested stochastic simulation algorithm for chemical kinetic systems with disparate rates," *J. Chem. Phys.* **123** (2005) 194107.

[14] J. Devita, P. Smereka and L. Sander, " Multiscale kinetic Monte Carlo for simulating epitaxial growth," *Physical Review B* **72** (2005) article no. 205421.

[15] P.A. Maksym, "Fast Monte-Carlo simulation of MBE growth," *Semiconductor Sci. and Tech.* **3** (1988) 594–596.

[16] J. L. Blue, I. Beichl and F. Sullivan, "Faster Monte-Carlo simulations," *Phys. Rev. E* **51** (1995) 867–868.