Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 204 (2005) 100–120
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Queue-based method for efficient simulation of biological self-assembly systems

Farokh Jamalyaria [a], Rori Rohlfs [a,b], Russell Schwartz [a,b,*]

[a] School of Computer Science, Carnegie Mellon University, 4400 Fifth Avenue, Pittsburgh, PA 15213, USA
[b] Department of Biological Sciences, Carnegie Mellon University, 4400 Fifth Avenue, Pittsburgh, PA 15213, USA

Received 14 June 2004; received in revised form 16 August 2004; accepted 1 October 2004
Available online 2 November 2004

Abstract

Self-assembly systems are central to a broad range of critical biological processes. Developing methods for quantitative simulation of self-assembly dynamics on cellular scales is therefore an essential sub-step in the broader goal of building predictive models of cellular function. Yet several aspects of self-assembly systems challenge key assumptions of conventional methods for biochemical simulation. Innovations are thus required in the rapid, quantitative simulation of self-assembly on cellular scales. In this paper, we describe a novel discrete-event queuing strategy for time- and memory-efficient quantitative simulation of self-assembly systems in continuous time. The method will typically allow simulation of interactions of even large, complex assembly structures in space and amortized run time per time step linear in system size. It can therefore be expected to extend the applicability of quantitative discrete-event methods to biologically important systems and scales inaccessible to prior techniques. In addition to presenting the method, we provide empirical evidence for its efficiency on two model systems.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Self-assembly; Queue; Discrete event; Simulation

1. Introduction

Self-assembly systems play crucial roles in numerous cellular functions (see, for example [18].) These systems include relatively small heterocomplexes such as the ribosome, regular structures of hundreds of proteins such as viral capsids, or much larger structures such as microtubules. Self-assembly is an essential step in such biological processes as DNA replication and transcription; protein translation, degradation,

* Corresponding author. Tel.: +1 412 268 3971; fax: +1 412 268 7129.
E-mail address: russells@andrew.cmu.edu (R. Schwartz).

0021-9991/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2004.10.004

and transport; and cell movement and shape control. Properly modeling the quantitative behavior of self-assembly systems is therefore essential to the broader goal of building predictive models of overall cell behavior. Self-assembly systems are also implicated in various pathological conditions, such as viral infection. Viral capsids – enclosures of typically several hundred proteins that protect the viral genome – are particularly impressive examples of how nature can produce complex but regular structures at extremely high rates and fidelity in difficult environments. Self-assembly has also emerged as a promising technology for nanometer-scale fabrication [17]. This fact provides even greater impetus to understand the self-assembly systems found in nature, which are far more sophisticated than anything human engineers can currently construct, and to develop simulation tools that can be used for rapid in silico prototyping of hypothetical engineered systems.

Despite the tremendous value that predictive, quantitative models of self-assembly behavior would have, existing simulation techniques poorly model their behavior in the cellular environment. The number of assembly intermediates (partially formed structures) is frequently exponential in the number of monomers in a complete assembly, making the differential equation models commonly used for biochemical simulation intractable for non-trivial systems without substantial simplifications. Furthermore, the large number of intermediates means that most intermediates are unpopulated at any given time in a cell-scale system, also compromising the accuracy of continuous models on small scales. Discrete models avoid the problem of having to maintain concentrations for all possible intermediates, but can quickly become bogged down by the large number of individual proteins that can be found in even a cell-scale system. Improvements in biochemical simulation methods are therefore needed to make quantitative self-assembly simulation tractable at cellular scales.

Aspects of this problem have been explored by several largely disjoint bodies of research. Much practical work in the biological literature has focused on icosahedral virus capsid assembly, due to its practical importance, its relative complexity, and its familiarity and tractability to experimentalists. Icosahedral virus capsids are computationally among the most challenging self-assembly systems because their relatively large sizes (typically several hundred proteins per capsid and possibly hundreds or thousands of capsids in an infected cell) and their relatively unconstrained growth patterns make them unusually difficult to model by conventional methods. The difficulties they pose for existing modeling methods, however, also make them an ideal model system for research on simulation methods for cellular biochemistry. Many simulation techniques have been applied to this system, including non-quantitative discrete-event models of varying levels of detail [1,2,11,15], simplified differential equation models [5,19,20], and detailed Brownian dynamics-like methods [10,13,14]. All of these techniques have limitations, though, either in the degree of simplification they require or in the range of systems they can model. In particular, none is well suited for quantitative simulations of virus-sized systems on the scale of tens of thousands of monomers required to simulate a single infected cell.

In the chemical engineering community, such molecular reaction systems are commonly modeled by a technique called the N-fold way [3,7]. The N-fold way depends on the fact that by neglecting consideration of explicit space in the model, waiting times between individual molecular reaction events can be modeled as exponentially distributed random variables. Various properties of these variables then allow substantial simplifications in sampling event times. In an N-fold way simulation, one repeatedly samples waiting times for all possible events, chooses the one with minimum time, and updates system state accordingly. For a system of n monomers and m distinct structures, an N-fold way simulation requires time $O(m^2 f(n))$ per simulation step, where $f(n)$ is the time to sample possible events between two structures. Recent work has improved on the classic N-fold way methods for particular use with assembly systems, reducing run time to $O(mf(n))$ at the cost of raising memory usage to $O(m^2)$ [8,9]. These quadratic dependencies, in either time or memory, mean that the method in either form will be difficult to apply to complicated systems with large numbers of geometrically distinct intermediate structures on the scales typical of a cellular system.

Mathematically equivalent systems are also familiar to the computer science literature, although for very different applications. This particular representation, in which a system transitions between discrete states with exponential waiting times between states, is an example of a continuous-time Markov model (CTMM) (see, for example [12]). These models are extensively used in computer system modeling among other areas and there is thus a great deal of theoretical work in the computer science literature on their behavior and simulation. The time evolution of such systems is described by a series of differential equations known as the Kolmogorov equations, which can be analytically solved or numerically integrated for many simple systems. For more complicated systems, particularly those in which the state set is too large to enumerate explicitly, it may not be possible to find analytical solutions or feasible to perform numerical integration of state distributions over time. Simulation may thus be necessary to understand model behavior.

The purpose of the present work is to extend the applicability of these quantitative discrete event models to yield more efficient simulations of computationally challenging biological self-assembly systems on cellular scales. We aim to do this by uniting disparate contributions from the various fields that have explored similar discrete-event models. Our problems are motivated by cell-scale biology and specifically by biological systems such as virus capsids that are especially challenging to existing computational methods. Our models are essentially identical to those used in the chemical engineering N-fold way method. Our computational strategies draw on standard techniques from the computer science literature for efficient simulation, particularly of CTMMs. The remainder of this paper describes and analyzes our technique and presents empirical results. We first describe a representation of generic self-assembly systems as a CTMM (an N-fold way model). We then develop a queue-based simulation strategy for fast, memory-efficient simulation of complex self-assembly systems using this N-fold way model. Finally, we provide empirical evidence for the efficiency and correctness of these methods on two simple model systems.

2. Methods

2.1. Computational model

We first define some basic terminology for describing our model of self-assembly systems. The basic building blocks of an assembly system are *subunits*, which generally correspond to individual proteins in a biological assembly system. A subunit has a collection of *binding sites*, each of which has some propensity for binding to any other binding site. The process of pairwise interaction between two binding sites is called a *binding interaction*. Connected components of subunits joined by bonds form an *assembly*. The current state of any simulation is then defined by the set of assemblies it contains. These objects are illustrated in Fig. 1. We assume below that our system contains n subunits comprising m distinct assemblies.

The collective potential binding interactions among all distinct types of binding sites are captured by a *binding matrix*. Element B_{ij} of the binding matrix corresponds to the binding affinity of binding site type i for binding site type j. These affinities are expressed as parameters of an exponential distribution describing the waiting time between binding events for two isolated subunits. The matrix is therefore symmetric. Similarly, an *unbinding matrix* describes waiting times for breaking any given bond type. These can also be characterized as parameters of exponential waiting time distributions. These concepts are illustrated in Fig. 2. For certain kinds of assemblies, a given pair of binding sites may need two distinct binding values: one corresponding to binding times when the binding sites are fixed in space relative to one another by other binding interactions and the other corresponding to binding times when the two binding sites are on assemblies freely moving relative to one another in the simulation space. Fig. 3 illustrates the distinction. This split of values is required because the entropy, and thus free energy, of binding is substantially lower when two binding sites are forced into the correct relative positions to one another for binding, which would be expected to reduce the time to bind. Note that the model would never require more than two such

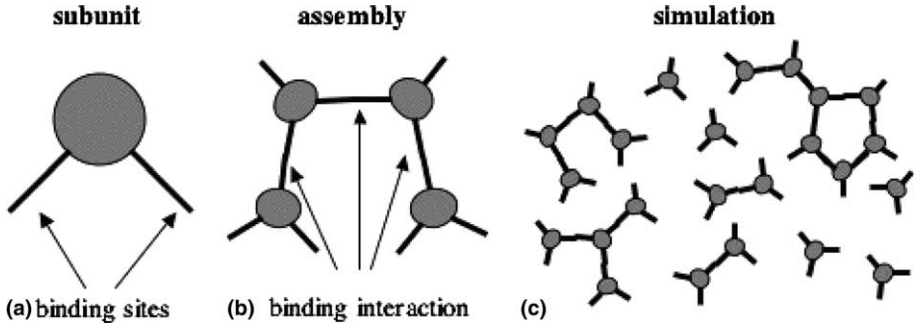

Fig. 1. Illustration of the components of an assembly simulation. (a) An assembly subunit with two binding sites; (b) an assembly formed by four assembly subunits engaged in binding interactions to one another; (c) a simulation consisting of a number of assemblies. Note that the discrete-event techniques examined here do not keep track of the positions of different assemblies in space, only of the relative subunit positions within each assembly.

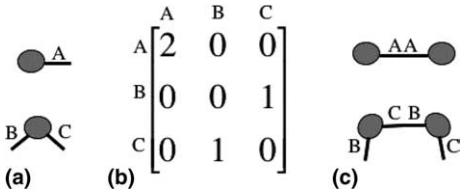

Fig. 2. Illustration of a binding matrix and the subunit binding patterns consistent with it. (a) A pair of subunits with binding sites. Edges are labeled according to binding site types A, B, and C. (b) A binding matrix defining possible interactions among the three binding site types and the relative rates of those interactions. (c) Pairings consistent with the matrix: a symmetric homodimer formed by joining two A binding sites and an asymmetric homodimer formed by joining a B and a C binding site.

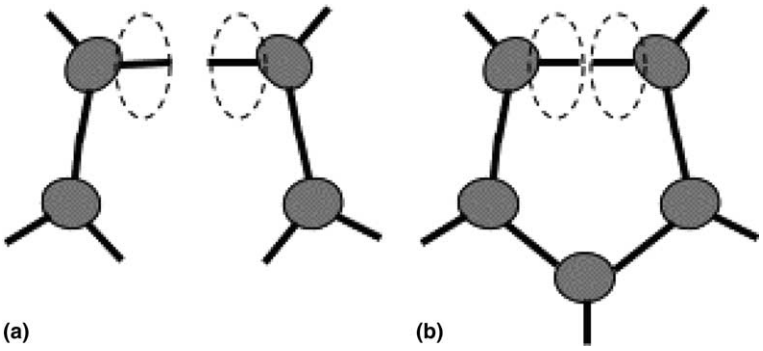

Fig. 3. Distinction between spatially fixed and unfixed binding interactions. (a) A pair of potential binding partners unfixed in space; (b) potential binding partners fixed in space by other interactions.

binding values per pair of binding sites, although more sophisticated models allowing for flexible binding interactions might. Simulations then consist in the abstract of repeatedly forming or breaking binding interactions, with the waiting time to the next event at any given time distributed according to the minimum of the waiting times to all possible individual events, as in a standard N-fold way simulation.

2.2. Algorithms

Our method implements an N-fold way simulation more efficiently by using an event queue to maintain the possible next events at any given time. A simulation constructed in this manner proceeds through an *event loop* in which we repeatedly pull the minimum-time event (that scheduled to occur nearest in the future) off of the queue, update system time, implement whatever actions the event requires, and then sample a waiting time (the time until the event occurs) for each potential new event that might then become possible.

The central issue in designing such a method is determining the queuing policy, which specifies when we place events in the queue, which events we place there, and how we deal with events that preclude others already in the queue. The first innovation of our method over prior work comes from the recognition that correctness of the method does not require storing times for all $O(m^2)$ possible pairwise or single-assembly interactions; we need only guarantee that the next event for each assembly is stored in the queue. Upon processing a next event, we need only sample new waiting times for the $O(m)$ events that involve subunits affected by the event just processed, analogous to what is done in the prior work [8,9]. Unlike in that work, however, we only store the one minimum-time new event per affected assembly in the queue, i.e. the unique event for each assembly that has the smallest waiting time. If no other event interacts with those assemblies prior to their minimum-time events coming to the top of the queue, then we can be sure that those events were not precluded and will appear with the correct waiting time distributions.

A complication arises in handling the *invalidation* of pending events in the queue by other events that reach the top of the queue first. The issue occurs because a reactant involved in some event could be affected by other events that occur between the first event's *posting time* (when it is placed on the queue) and its *activation time* (when it is supposed to occur). For example, suppose our queue contains an event indicating that assemblies A and B will bind at time t, but some other event produces an assembly C which is selected to bind with B at time $t - \Delta t$. In that case, by the time we reach the A/B binding event in the queue, that event will no longer be usable because B will have bound with C. We solve this with a lazy evaluation strategy, in which each assembly maintains a *valid time* reporting the last time new events were sampled for it. Any event posted to the queue prior to the valid time of one of its reactants is then *invalid* and does not modify the system state when it comes up. If an invalid event was posted before the valid times of all of its reactants, then we can be sure that each such substituent has already had some other events sampled for it. If the invalid event was posted after the valid time of one of the reactants for which it was sampled, then that event serves as a place-holder indicating that that reactant has not experienced any events between its valid time and the invalid event's activation time. The waiting time to that reactant's next event must then be resampled when the invalid event is processed.

The second innovation in our algorithm is a periodic requeuing operation used to maintain constant queue size. Every $O(m)$ steps of the algorithm, we delete the entire pending event queue and reinitialize it as at the beginning of the algorithm. As we initialize the queue with only one event per assembly and add only a constant number of events to the queue per event removed, this strategy guarantees queue size remains $O(m)$. Less obvious is that it has no asymptotic effect on amortized run-time of the method for reasonable queue data structures. Note that requeuing requires $O(m^2)$ time to recompute possible next events and, for any reasonable queuing method, $o(m^2)$ time to place $O(m)$ of these events in the queue. Thus, the amortized cost of requeuing is $O(m)$ per step, adding no asymptotic increase over the existing $O(m)$ sampling cost per step.

Fig. 4 provides pseudocode for the resulting queue-based algorithm with $O(m)$ requeuing. Queuing policy in the figure is described in terms of the *ValidTime* of each assembly (when its events were last resampled, invalidating its previous events), the *PostTime* of each event (when it was placed in the queue), and the

```
Initialize(t_0):
  t ← t_0
  step ← 0
  q ← ∅
  Sample all possible events
  place the minimum-time event involving each assembly a_i in q
  ValidTime[a_i] ← t for all a_i
End Initialize

Simulate(t_max,k):
  Initialize(0)
  while (t < t_max)
    step ← step + 1
    if (step > km) then Initialize(t)
    e ← pop(q)
    t ← ActiveTime(e)
    if (e is a binding interaction between a_i and a_j)
      if (PostTime(e) ≥ ValidTime(a_i) ⋀ PostTime(e) ≥ ValidTime(a_j))
        bind a_i to a_j to create a_k
        sample all possible events involving a_k and store minimum in q
        ValidTime[a_k] ← t
      else if (PostTime(e) ≥ ValidTime(a_i) ⋀ a_i ∈ Parents(e))
        sample all possible events involving a_i and store minimum in q
        ValidTime[a_i] ← t
      else if (PostTime(e) ≥ ValidTime(a_j) ⋀ a_j ∈ Parents(e))
        sample all possible events involving a_j and store minimum in q
        ValidTime[a_j] ← t
      end if
    else if (e is an unbinding interaction for assembly a_k)
      if (PostTime(e) ≥ ValidTime(a_k))
        break a_k into two assemblies a_i and a_j
        sample all possible events involving a_i or a_j, store min. for each in q
        ValidTime[a_i] ← t
        ValidTime[a_j] ← t
      end if
    end if
  end while
End Simulate
```

Fig. 4. Pseudocode for our queue-based algorithm self-assembly simulation with requeuing in the assembly centered variant of the algorithm. *Initialize( )* encodes the queue initialization used when first starting and when requeuing simulations. *Simulate( )* runs the overall simulation for a time $t_{\max}$. We focus here on the issues of queuing events and establishing their validity, omitting details on sampling event times.

ActiveTime of each event (when it is set to occur). It also occasionally depends on the *Parent* of an event, defined as the assemblies whose minimum event time distributions were sampled to yield the event in question. Collectively, our innovations give a significant advantage for complex assembly systems having large numbers of distinct intermediates. They allow us to keep the O(m) space complexity of the classic N-fold way method and get the O(m) per-step time complexity of the recent innovations by Laurenzi et al. [8,9].

These performance improvements are, however, conditional on the frequency with which events are invalidated by the queuing method, an issue examined in depth below.

We can illustrate the method with a simple example, shown in Table 1. Suppose we examine a dimer system made up of three subunits, A, B, and C, where each one can bind to either of the other two. We can walk through a few possible steps of the algorithm for this system:

(1) We initialize the queue by sampling times for all possible events. There are initially three possible events: A + B → AB, B + C → BC, and A + C → AC. Suppose our random sampling gives event activation times of $t = 1$ for A + B → AB, $t = 2$ for B + C → BC, and $t = 3$ for A + C → AC. Then we must place the minimum time event for each assembly in the queue. For both A and B, the minimum time event is A + B → AB at $t = 1$. For C, it is B + C → BC at $t = 2$. We therefore initially fill our queue with A + B → AB and B + C → BC.
(2) To take the first step of the simulation, we then extract the minimum time event from the queue, A + B → AB at $t = 1$, and implement that event. Our current state is then an AB dimer and free monomer C at time $t = 1$. We then sample among possible events for the new AB dimer. There is only one such event: AB → A + B. We will suppose this event is sampled with waiting time 2, which, added to the current time of 1, gives the new event an activation time of 3. This is the only event sampled, so it is added to the queue.
(3) We again extract the minimum time event from the queue: B + C → BC at $t = 2$. This event is invalid for B because the event was sampled at time 0 but B was last modified at time 1. We therefore discard the event without implementing it. The event is valid for C, so we must sample new possible events for C. Because C has no binding partners available, though, there are no possible new events for C and we add nothing to the queue. We do, however, update the system time to that of the discarded invalid event, $t = 2$.
(4) We now extract the minimum time event: AB → A + B at time 3. The event is valid for AB so we break AB into A and B and update the system time to 3. We must then sample possible next events for A and B. Suppose we now choose times $t = 7$ for A + B → AB, $t = 4$ for B + C → BC, and $t = 5$ for A + C → AC. Then B + C → BC is the minimum time event for B and A + C → AC is the minimum time event for A, so we place those two events in the queue to prepare the system for the next step.

Table 1
Illustration of a possible series of steps of the queuing algorithm on a simple heterodimer system of three monomers

Time	System state	Events sampled	Resulting queue
0	A	A + B → AB ($t = 1$)	A + B → AB ($t = 1$)
	B	B + C → BC ($t = 2$)	B + C → BC ($t = 2$)
	C	A + C → AC ($t = 3$)	
1	AB	AB → A + B ($t = 3$)	B + C → BC ($t = 2$)
	C		AB → A + B ($t = 3$)
2	AB	none	AB → A + B ($t = 3$)
	C		
3	A	A + B → AB ($t = 7$)	B + C → BC ($t = 4$)
	B	B + C → BC ($t = 4$)	A + C → AC ($t = 5$)
	C	A + C → AC ($t = 5$)	

Each row describes the system at a particular point in time, the events sampled at that time, and the state of the queue following the sampling.

We could continue this process indefinitely, periodically emptying the queue and reinitializing it by considering all possible events from the current state.

2.3. Theoretical analysis

In this section, we analyze our methods theoretically. We first seek to establish the correctness of the methods by showing that event distributions yielded by our queue-based method are described by the same CTMM as that describing a classic queue-less N-fold way simulation of the same assembly system. Second, we examine the memory and run-time efficiency of the methods. For simplicity, we assume that the only event types are binding and unbinding events. We thus neglect single-molecule events, such as conformational shifts that may change subunit binding characteristics. Accounting for such events would not negatively impact any of the theoretical results we present below.

We first note an important property of exponential distributions, the "memory-less property", that allows us both to perform the "resampling" operation that is the basis of much of our algorithm and to use events pending in the queue. Suppose we examine our queue at some time t and we are interested in some potential next event e with exponentially distributed waiting time w that was sampled at time $u < t$ but is observed not to have have occurred yet as of time t. The memory-less property of exponential random variables tells us the following:

$$Pr\{w = t + s | w > t\} = Pr\{w = s\} = Pr\{w = u + s | w > u\} \tag{1}$$

Informally, this means that if we have observed a pending event at time t that was originally sampled from an exponential distribution and placed on the queue at time $u < t$, then its additional waiting time after time t will be distributed identically to what it would be if we resampled it at time t from the same exponential distribution. This is the key property of exponential random variables that makes both the classic N-fold way and our queuing algorithm possible.

We now first prove a basic proposition that says that we can add "null" states to a continuous-time Markov model that immediately transition back to the starting state without affecting the overall time progress of the model. A null state q_* is modeled as a state that is reached from some starting state q by a waiting time exponentially distributed with parameter λ_* and then immediately returns with zero waiting time. The idea of a null state is illustrated in Fig. 5. We wish to show that the distributions of waiting times by which we leave state q to enter any non-null state are unaffected by the presence of the null state. This is established by the following lemma:

Lemma 1. *Assume we are given a CTMM M containing some state q with neighbors $q_1,\ldots,q_k$. Further assume we construct a second CTMM M_* from M by adding a "null state" q_*. q transitions to q_* with rate λ_* and q_* transitions only to q with infinite rate. For any $q_i \in \{q_1,\ldots,q_k\}$, the probability q_i is the next non-null state reached after q and the distribution of waiting times until q_i is reached are the same for M as for M_*.*

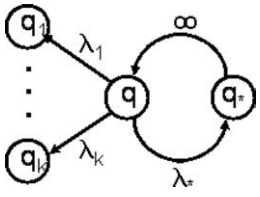

Fig. 5. Illustration of a subset of a CTMM containing a "null state" q_* reachable from some state q. We assume q can transition to some other set of states, $q_1,\ldots,q_k$, which each has some other set of potential transitions (not shown). Nodes correspond to distinct states of the CTMM. Edges are labeled with their rates, expressed as parameters of exponential distributions.

Proof. Define $\exp(\lambda)$ to be an exponentially distributed random variable with parameter λ, λ_* to be the rate of the $q \to q_*$ transition, and Λ to be the sum of the rates of all other transitions out of q. We now consider the probability p_i that our next non-null state is some q_i with parameter λ_i. We can enter q_i next either by transitioning directly from q to q_i or by transitioning some number of times from q to q_* and then from q to q_i. These possibilities yield the following probabilities:

$$p_{\mathrm{i}} = \frac{\lambda_{\mathrm{i}}}{\Lambda + \lambda_*} + \frac{\lambda_*}{\Lambda + \lambda_*} p_{\mathrm{i}}, \tag{2}$$

$$p_{\mathrm{i}}\left(1 - \frac{\lambda_*}{\Lambda + \lambda_*}\right) = \frac{\lambda_{\mathrm{i}}}{\Lambda + \lambda_*}, \tag{3}$$

$$p_{\mathrm{i}}\left(\frac{\Lambda}{\Lambda + \lambda_*}\right) = \frac{\lambda_{\mathrm{i}}}{\Lambda + \lambda_*}, \tag{4}$$

$$p_{\mathrm{i}} = \frac{\lambda_{\mathrm{i}}}{\Lambda}. \tag{5}$$

The probability of choosing q_i directly from q in M is also λ_i/Λ, satisfying the first part of the lemma. Using a similar argument, we can express the waiting time to enter any state other than q_* as follows:

$$w = \exp(\Lambda + \lambda_*) + \frac{\lambda_*}{\Lambda + \lambda_*} w, \tag{6}$$

$$w\left(\frac{\Lambda}{\Lambda + \lambda_*}\right) = \exp(\Lambda + \lambda_*), \tag{7}$$

$$w = \frac{\Lambda + \lambda_*}{\Lambda} \exp(\Lambda + \lambda_*), \tag{8}$$

$$w = \exp(\Lambda), \tag{9}$$

$\exp(\Lambda)$ is also the waiting time to leave state q in M, satisfying the second part of the lemma. □

We require one more lemma to allow us to show correctness of our method. This lemma asserts an invariant on our queue state that will allow us to argue that it models the same process as the classic N-fold way method. We define an event to be valid for assembly X if the event's posting time is not less than X's valid time. We can then state the following:

Lemma 2. *For any possible next event e_i with rate λ_i, either e_i is present in the queue with waiting time sampled from* $\exp(\lambda_i)$ *or there is some event e_j with rate λ_j that is valid for each parent of e_i and was chosen to have a shorter waiting time than e_i with probability equal to* $Pr\{\exp(\lambda_j) < \exp(\lambda_i)\}$.

Proof. We establish this by structural induction on the system state over time. As the base case, we consider the queue immediately after an initialization or reinitialization operation. At this time, all possible events are sampled and the minimum-time event for each subunit X is stored in the queue. Suppose we consider some arbitrary possible event e_i. Then e_i will be placed in the queue unless at least one other e_j for each of the subunits involved in e_i is sampled to have shorter waiting time. Thus, if e_i appears in the queue it will have a waiting time distributed as $\exp(\lambda_i)$. If it does not appear in the queue, then some e_j sampled from $\exp(\lambda_j)$ must have been found to have a shorter waiting time with probability equal to $Pr\{\exp(\lambda_j) < \exp(\lambda_i)\}$.

For the inductive step, we must consider all the ways system state (the distributions of particles) or queue state (the set of events and event times in the queue) can change by handling some event:

(1) valid unbinding event: In this case, one reactant, Z, is removed, eliminating possible events it might have participated in and two products, X and Y, are created, enabling possible new events. The algorithm invalidates any events referring to Z, converting them to "null" events and thus leaving us with a Markov model that yields equivalent event-time distributions to one without any Z events. The algorithm will sample minimum-time events for X and Y, guaranteeing that any possible event referring to either either occurs in the queue, having been sampled correctly, or is preceded by another correctly sampled event valid for the same parent that appears in the queue.
(2) valid binding event: In this case two reactants, X and Y, are removed from the simulation, removing any possible events they might have been involved in, and one product Z is created, possibly enabling new events. The algorithm will invalidate any events referring to X or Y, converting them into "null" events that do not change event time distributions. It will also sample possible events for Z and store the minimum-time event, guaranteeing that any new event created by the processing of the current event is either in the queue with the correct sampling distribution or occurs after another event valid for common parent Z that is in the queue and was sampled with its correct distribution.
(3) invalid unbinding event: In this case, system state does not change. The unbinding event could not be valid for it parent so its removal does not affect the validity of the lemma.
(4) invalid binding event that does not result in resampling: the system state does not change in this case. The removed event could not be valid for its parent or else it would induce resampling. Thus, if the lemma was valid for the queue prior to the removal of this event, it must be valid after it.
(5) invalid binding event that results in resampling: The system state does not change in this case. The invalid event is valid for its parent and invalid for its other reactants. After removal of the event, the algorithm invalidates any existing events involving the parent, resamples all possible events involving the parent and stores the one with minimum time in the queue. Thus, for any possible event involving the parent, either the event is in the queue or another valid such event with lesser waiting time is in the queue with correctly sampled waiting times.

Together, these cases consider all possible ways the system or queue state can change. Since the lemma is true as of queue initialization and each possible update preserves its validity, the lemma is valid. □

The preceding arguments now allow us to establish our primary result on the correctness of the method:

Theorem 1. *From any system state, the queue method will select the next state with identical probability distribution and identical waiting time distribution to a queue-less N-fold way simulation.*

Proof. A queue-less N-fold way simulation selects the next state by sampling waiting times to all possible next states and choosing the minimum. By Lemma 2, the queue will contain a set of events (some of them possibly null events) sampled from the correct exponential distributions such that at least one event in the queue has time less than or equal to that of any event not in the queue. No non-null event in the queue can encode any transition that is not possible given the current system state, since the event would have been invalidated by the disappearance of any of its reactants. Thus, the minimum-time event in the queue is distributed according to the minimum of the set of possible next events and a set of null events. By Lemma 1, the probability distributions with which such a CTMM selects

its next non-null state and waiting time to that state will be identical to those derived from a CTMM lacking the null events. Thus, the queue-based simulator will implement a CTMM equivalent to one that samples all possible next events of the system state and chooses the one with minimum waiting time. The queue-less and queue-based methods thus yield CTMMs with equivalent probability distributions of next states and waiting time distributions and therefore implement identical models of the physical process. □

Having established correctness of the method, we now seek to establish its efficiency in run-time and memory usage. We can first show the following:

Theorem 2. *Total memory usage is* $O(m)$.

Proof. Memory usage is dominated by the event queue and a constant amount of state per assembly. The requeuing method bounds the queue size by $O(m)$. There are $O(m)$ assemblies, so $O(m)$ storage is required to maintain their state. Thus, total memory usage is $O(m)$. □

We establish the run-time efficiency of the method in terms of the average time to compute possible events between two assemblies, $f(n)$, and a function $v(n)$ equal to the inverse of the fraction of events that are valid among those reaching the top of the queue.

Theorem 3. *The average run time per discrete event is* $O(mf(n)v(n))$.

Proof. On each event, our method does constant work to update state then samples new events for at most two assemblies at cost $O(mf(n))$ each. Adding the amortized $O(m)$ requeuing cost per step yields total cost per event of $O(mf(n))$. However, since our simulation state advances only on valid events, we must consider the true cost to be $O(mf(n)v(n))$ per discrete event. □

Thus, our method is superior to the classic N-fold way when $v(n) = o(m)$. In no event can it give asymptotically worse performance than the classic method, since the first event after requeuing is guaranteed to be valid. It is as efficient in run-time as the Laurenzi et al. method when $v(n) = O(1)$. We now develop a potential function argument based on bounds on the queue size to show that $v(n)$ is in fact $O(1)$ for the important special case of systems at equilibrium. To do this, we establish certain properties of binding and unbinding events, and we then combine them with properties of equilibrium systems to prove a lower bound on $1/v(n)$. We divide the events into five classes:

(1) valid unbinding event: an unbinding event valid for its parent,
(2) invalid unbinding events: an unbinding event not valid for its parent,
(3) valid binding events: a binding event valid for both of its reactants,
(4) fully invalid binding events: a binding event not valid for its parent or parents,
(5) semi-invalid binding events: a binding event that is valid for its parent but not for its other reactant.

We now examine how each possible event type can affect the queue size. The possibilities are expressed in the following lemma:

Lemma 3. *The possible net change in queue size from processing any single event are as follows*:

(1) *valid unbinding*: 0 *or* +1
(2) *invalid unbinding*: −1,
(3) *valid binding*: −1 *or* 0,
(4) *fully invalid binding*: −1.
(5) *semi-invalid binding*: −1 *or* 0.

Proof. We consider each case separately:

(1) valid unbinding: The event's product will be two subunits and the algorithm will enqueue at most two new events, one for each subunit. The gain of one or two events and the loss of the one event removed from the top of the queue give a net change of 0 or +1.
(2) invalid unbinding: The algorithm cannot sample for any subunits on an invalid unbinding event and it will therefore not enqueue any new events. The loss of one event from the top of the queue coupled with the gain of no events gives a net change of -1.
(3) valid binding: The product will be a single aggregate of the two involved subunits, for which the algorithm will enqueue at most a single future event. The net change is therefore -1 or 0.
(4) fully invalid binding: The algorithm will not resample in this case and so will create no new events. The net change is therefore -1.
(5) semi-invalid binding: The algorithm may sample events for either zero or one of the two binding partners, creating zero or one new event. The net change is therefore 0 or -1. □

We will now use the preceding lemma to establish some restrictions on proportions of events falling into the five event classes. We first define the following:

- t_{vu} = the number of valid unbinding events since the last requeuing,
- t_{iu} = the number of invalid unbinding events since the last requeuing,
- t_{vb} = the number of valid binding events since the last requeuing,
- t_{ib} = the number of fully invalid binding events since the last requeuing,
- t_{sb} = the number of semi-invalid binding events since the last requeuing,
- $t_v = t_{vu} + t_{vb}$, the number of valid events since the last requeuing,
- $t_i = t_{iu} + t_{ib}$, the number of invalid events since the last requeuing not counting semi-invalid binding events,
- t = the total number of events processed since the last requeuing neglecting semi-invalid binding events.

We will now bound the ratio of valid to fully invalid events, neglecting semi-invalid events for the moment:

Lemma 4. *If* $t > 3m$ *then* $\frac{t_v}{t_v+t_i} \geqslant \frac{1}{3}$.

Proof. All invalid events other than semi-invalid binding events decrease the queue size by at least 1. Only valid unbinding events can increase the queue size. The queue size is at most m at the time of requeuing and cannot fall below zero. Therefore, if we examine the net decrease in queue size, Δq, we can conclude

$$(t_{iu} + t_{ib}) - t_{vu} \leqslant \Delta q \leqslant m, \tag{10}$$

$$t_{vu} \geqslant (t_{iu} + t_{ib}) - m, \tag{11}$$

$$t_{vu} + t_{vb} \geqslant (t_{iu} + t_{ib}) - m, \tag{12}$$

$$t_v \geqslant t_i - m \tag{13}$$

If $t_v + t_i = km$ then $t_v \geqslant \frac{k-1}{2}m$.

$$\frac{t_v}{t_v + t_i} = \frac{t_v}{km} \geq \frac{(k-1)m/2}{km} = \frac{k-1}{2k}. \tag{14}$$

For $k > 3$, $\frac{t_v}{t_v+t_i} \geqslant \frac{1}{3}$. □

The most difficult portion of the proof deals with the fact that sampling the waiting time to the next valid or semi-invalid event requires sampling the minimum of several exponentials derived at different times from different system states. We will show that the expected value of this waiting time is equal to the expected value it would have if we assumed all of the variables were sampled simultaneously from a single system state. We will show this by considering the expected time to the next valid or semi-invalid event in terms of the expectation of the time if all events in the queue were sampled from the current state plus the expectation of a series of "corrections" to that time to account for the prior system states from which individual queue elements were sampled. We will then show that the expectation of these corrections is zero.

For the purposes of this proof, assume that our queue was produced over the course of a particular trajectory through the CTMM describing our system states. Assume we have a set of elements in the queue, $e_1,\ldots,e_k$ each sampled from the minimum of the distributions of possible next events for at least one assembly capable of undergoing events at the current time. The oldest element in the queue was introduced while the queue was in some state q_1 and successive elements were introduced over the course of a series of states $q_2,\ldots,q_{k-1}$ with the system currently in state q_k. We assume these states were encountered at times $t_1,\ldots,t_k$. Define T to be the random variable describing what the time to the next event would be if all events were sampled from state q_k as in a standard N-fold way simulation. Whichever assembly or assemblies had their next event times sampled at q_{k-1} may make slightly incorrect contributions to T because the set of possible binding partners for them may have changed between time t_{k-1} and time t_k. We will define the random variable expressing this error to be Δ_{k-1}. Likewise, we can define a $\Delta_1,\ldots,\Delta_{k-2}$. These random variables are likely to be dependent on both T and each other. Let $\tau = T + \Delta_1 + \cdots + \Delta_{k-1}$ be the random variable expressing the time to the next event. We will now show the following:

Lemma 5. *The expectation of Δ_i over all possible system trajectories and waiting time samples is zero for all i, provided the system has reached its stationary distribution.*

Proof. Consider a particular assembly i whose minimum time event is sampled at time t_i from state q_i and then remains in the queue up through the current time t_k. The probability of observing this trajectory is $\pi_i p_{ik}(t_k - t_i)$ where π_i is the stationary probability of choosing state q_i and $p_{ik}(t)$ is the probability that we transition from q_i to q_k over a time span t. By the detailed balance condition (a necessary property of a CTMM that reaches a stationary distribution) $\pi_i p_{ik}(t_k - t_i) = \pi_k p_{ki}(t_k - t_i)$. Let τ_i be the expected waiting time for subunit i's next event if sampled from state i and τ_k be its expected time if sampled from state k. Then for each trajectory from i to k yielding a contribution of $\pi_i p_{ik}(t)(\tau_i - \tau_k)$ to the expectation of Δ_i there is a contribution $\pi_k p_{ik}(t)(\tau_k - \tau_i)$ from the opposite trajectory. Applying the fact that the probabilities are equal, we conclude that these contributions cancel out, yielding an expectation of zero for Δ_i. □

We can then immediately conclude the following:

Corollary 1. *The expectation over all possible system trajectories and event samples of τ is equal to the expectation of T provided the system has reached its stationary distribution.*

Proof. By linearity of expectation

$$Ex[\tau] = Ex[T + \Delta_1 + \cdots + \Delta_{k-1}], \tag{15}$$

$$= Ex[T] + Ex[\Delta_1] + \cdots + Ex[\Delta_{k-1}], \tag{16}$$

$$= Ex[T] + 0 + \cdots + 0, \tag{17}$$

$$= Ex[T]. \qquad \square \tag{18}$$

The preceding proofs allow us to calculate the expected time to the next valid or semi-valid event by assuming that all valid or semi-valid events pending in the queue were sampled from the current state of the queue. We will use this fact to bound the expected waiting time until a valid or semi-valid event in terms of the expected waiting time to a valid event. This is applied in the proof of the following:

Lemma 6. *Consider a current system state q_i. Let X be the random variable describing the minimum of the times to all possible next events. Assume there are k assemblies available to participate in events at state q and let $Y_1, \ldots, Y_k$ be the random variables where Y_j describes the minimum of the the waiting times to all possible next events involving assembly j. Let Y be the minimum of the set $\{Y_1, \ldots, Y_k\}$. Then $Ex[Y] \geqslant (1/2)Ex[X]$.*

Proof. For any state q_i with stationary probability π_i, there is some set of possible next events with characteristic rates $\lambda_1, \ldots, \lambda_k$. The time X to the next event is exponentially distributed with parameter $\Lambda_i = \lambda_1 + \cdots + \lambda_k$ and thus has mean $1/\Lambda_i$. The time Y to the next valid or semi-valid event if all events in the queue were sampled from state q_i is the minimum of a set of exponentials that includes all of $\lambda_1, \ldots, \lambda_k$ and possibly one additional copy of each, as an event involving assemblies j and l could be counted once in sampling for Y_j and once in sampling for Y_l. No other events can be included in the distribution of Y and no event can be included more than twice as they can only be present if they are among some Y_i's next events and can be contributed by at most two reactants. Y is thus exponential with parameter less than or equal to $2(\lambda_1 + \cdots + \lambda_k) = 2\Lambda_i$. Its expected waiting time is thus at least $1/2\Lambda_i$. Summing over all possible system states j, the expected waiting time until the next valid event, $E[X]$, is $\sum_j \pi_j/\Lambda_j$ and the expected waiting time to the next valid or semi-valid event, $E[Y]$, is at least $\sum_j \pi_j/2\Lambda_j$. Thus, $E[Y] \geqslant (1/2)E[X]$. □

The preceding lemma establishes that once the system is at equilibrium (i.e. has reached its stationary distribution) the expected number of valid events is at least one half of the expected number of valid or semi-invalid events over any span of time. In other words, the expected number of valid events is at least as large as the expected number of semi-invalid events for systems at equilibrium. Finally, we have all of the pieces necessary to prove our primary theorem on the efficiency of the method:

Theorem 4. *For any system at equilibrium where at least* $3n$ *events have been executed since the last requeuing operation, the average over all possible system trajectories of the fraction of valid events is at least* $1/4$.

Proof. Lemma 4 showed that at least one third of valid or fully invalid events are valid. Lemma 6 showed that the expected valid fraction of valid or semi-invalid events is at least one half. Thus, the expected number of invalid or semi-invalid events per valid event is at most three, yielding an expected valid fraction of at least 1/4. □

From the above, it follows directly that the method yields amortized $O(m)$ run time per event for any system at equilibrium provided at least $3m$ steps are allowed to run between requeuing events. Together with our prior results, we can thus show that our method matches the best asymptotic run-time and the best asymptotic memory usage of any existing methods on equilibrium systems. Bringing both time and memory complexity simultaneously below $O(m^2)$ is a significant milestone in making cell-scale models of complex assembly systems computationally feasible.

Naturally, we are curious about non-equilibrium systems. The equilibrium assumption is necessary in the proofs only in bounding the ratio of valid to semi-invalid events. Informally, the assumption is important because it allows us to show that the delay between when an event is added to the queue and when it reaches the top does not change its expected contribution to overall waiting time. Systems in which at least a constant fraction of chosen events are unbinding events would yield linear run time even if they are not at equilibrium. If, however, the system were in a state in which the fraction of binding events approached one with increasing system size and in which the expected time to the next event were rapidly increasing (by more than any constant factor in system size) with each succeeding event, then our method could degrade to

$O(m^2)$ run time per event. We can construct artificial examples that function in this domain for a limited time, but it remains an open question whether such a transient state can continue for a large number of events ($\Omega(m)$ for example) or whether such behaviors occur at all in any realistic systems.

As an example of a bad case for our method, consider a heterodimer system containing subunits $a_1,\ldots,a_{n/2}$ and $b_1,\ldots,b_{n/2}$ initially unbound. Dimers are formed by binding some a_i to some b_j. Assume further that each b_j is identical, but that the a_i's have dramatically different binding affinities from one another such that the time for a_1 to bind any b_j is much shorter than that for a_2 to bind any b_j, the time for a_2 to bind any b_j is much shorter than that for a_3 to bind any b_j, and so on. Dissociation could be assumed to follow the opposite pattern, with the time for an a_1:b_j dimer to dissociate being much longer than for an a_2:b_j dimer, and so on. In such a situation, our method would be expected initially to fill the front of the queue with a_1:b_j binding events. The first of these would then be selected, invalidating the others. The method would then fill the queue with a_2:b_j binding events behind the remaining invalid a_1:b_j events, with the first a_2:b_j event being reached after the last a_1:b_j event was removed from the queue. Proceeding in this fashion, we would have to handle $O(n)$ invalid events for every valid event until all $n/2$ dimers had been formed.

2.4. Variations on the method

We have been describing here simulations in which we treat assemblies as our basic unit of structure – events describing interactions between species of assembly – analogous to the treatment of events in the prior work. Under certain circumstances, it may be reasonable to consider subunit-centered simulations. That is, we could define a binding event to be an event that joins two subunits, without regard for whether they are involved in larger assemblies, and store one such event per subunit in the queue when we sample new subunit events. If assemblies have size $O(n)$ then it might require $O(n^2)$ work to compute possible interactions between any two assemblies, potentially yielding $O(n^2)$ run time per simulation step in an assembly centered simulation. A subunit-centered simulation would require only $O(n)$ run time and $O(n)$ memory in this case. In cases where the cost of computing interactions between assemblies is constant, however, an assembly centered simulation would require only $O(m)$ time and space, which may be considerably less than $O(n)$ when assemblies can be large.

3. Empirical analysis

3.1. Implementation of model systems

In order to supplement our theoretical proofs of efficiency, we have developed prototypes of the simulator for two simple hard-coded systems. For the required priority queue, we have generally preferred a calendar queue [4], which gives an expected O(1) run time per event when waiting times are exponentially distributed and has well established theory on both analytical [6] and empirical [16] methods for parameter tuning. For the present work, though, we used instead a less efficient binary heap. This change does not affect our method's overall asymptotic performance and eliminates some issues of calendar queue parameter tuning that complicate analysis of empirical run times. For these examples, we perform requeuing every $4m$ steps.

We first implemented a simple homodimer assembly system, described by the reversible reaction $A + A \leftrightarrow B$. We used mean waiting times in both directions of 1 (in arbitrary units). In order to better model the situation of a typical non-trivial assembly system, we do not exploit the fact that there are only two types of assemblies in the simulation, instead treating each monomer or dimer as if it were a unique

assembly whose event times must be sampled independently from all other assemblies. This allows us to better stress the dependence of the systems on the number of assemblies by effectively making $m = O(n)$. For purposes of comparison, we have also implemented a traditional queue-less N-fold way simulation of the same system, also not exploiting the fact that most assemblies are identical.

We also created a model filament assembly system to explore behavior on a system with unbounded assembly sizes. Filaments are formed from chains of assembly subunits of arbitrary length. On each simulation event, either two filaments bind end-to-end or one filament breaks at an arbitrary point along its length. Each possible binding or breaking event again has waiting time exponentially distributed with parameter 1. We again did not exploit the fact that filaments of equal length have identical distributions in order to better model more difficult systems in which we cannot in general equate distinct assemblies even if they have identical sizes. In this case, though, the number of filaments of any given length is usually small after the simulation has run for a sufficient time, so that optimization would not be expected to lead to more than a constant factor improvement over large numbers of steps. We do, however, find that it can significantly reduce startup costs for simulations initiated from free monomers (data not shown).

Finally, we created a variant of the preceding filament simulations in which unbinding rate is held fixed but binding rate varies inversely with starting system size. Fixing both rates causes changes in system size to model different concentrations in a fixed volume of solution. Varying binding rate with system size causes changes in system size to model a change in volume of a fixed concentration of solution. While the former model is more useful for establishing the behavior of the method across different systems, the latter is more useful for showing how the performance of the method on a fixed system scales with the size of the model of that system. All performance tests were run on desktop computers with 2.80 GHz Pentium processors running Linux.

3.2. Results

Our empirical validation is intended to establish two points: that our methods are correct in both theory and implementation and that the critical issue of valid fractions of events is resolved as our theory predicts. We validate correctness of the implementation using the dimer system by comparing events processed in the queue-less N-fold way simulation to valid events processed in the queue-based simulation for equal amounts of simulation time. We compared the methods for systems of size 10 and 100 subunits at time points 1, 100, 250, 500, and 1000. Run-time was prohibitive for the queue-less method at larger system sizes. The resulting data are shown in Table 2. The nearly identical results confirm that the implementation of our queue-based method yields correct waiting times between events.

The central issue in establishing efficiency of our method is showing that the fraction of processed events that are valid converges on a positive constant for large system sizes. We evaluated these fractions for systems of 10, 100, 1000, and 5000 subunits at time points 1, 10, 100 and 250. Fig. 6(a) plots valid fraction as a function of system size at various stages of the program's run. For any one time point, the valid fraction rapidly converges to a constant in system size, confirming linear time per time step. The asymptotic valid fraction is approximately 0.51 for the larger time points, significantly better than our theoretical lower bound of 0.25. At time $t = 1$, where the system would be expected not to have reached equilibrium, the valid fraction is noticeably lower (about 0.45). It is not clear if it is converging to a lower constant or a decreasing function of n. Valid fractions at $t = 10$ are nearly indistinguishable from those at $t = 100$ and $t = 250$, which themselves align almost perfectly, suggesting that any transient effects disappear quickly from this system.

Our constant-volume filament-based system allows us to test whether our results on valid fractions hold in a more complicated system with very different equilibrium behavior. We examined valid fractions at the same time points and system sizes as for dimer systems. Fig. 6(b) shows valid fractions as functions of system size. For the larger time points, we again observe apparent convergence on a constant valid fraction, apparently also approximately 0.51. There is some variability between these points at smaller system sizes,

Table 2
Comparison of event counts between our queue-based method and a ''gold standard'' queue-less N-fold way simulation

Particles	Time	Valid events: queued	Events: non-queued
10	1	9	9
10	100	666	731
10	250	1734	1792
10	500	3549	3583
10	1000	7251	7154
100	1	135	139
100	100	9253	9022
100	250	23,104	22,747
100	500	45,863	45,528
100	1000	91,009	90,587

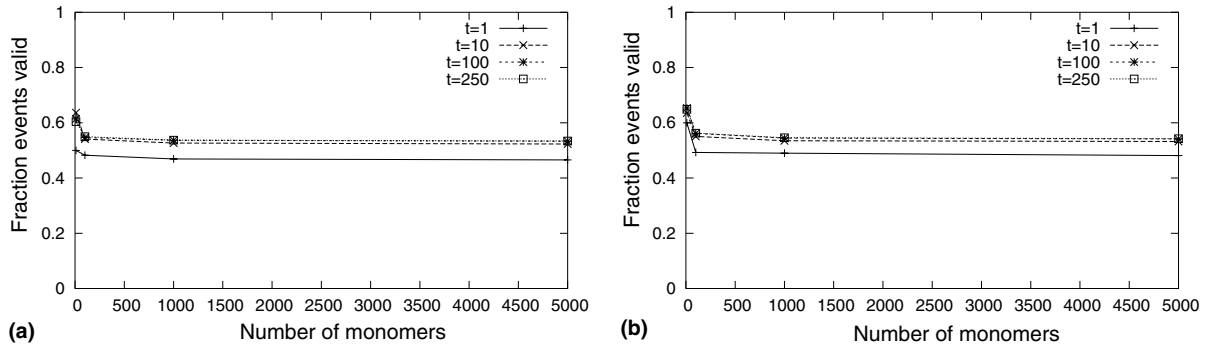

Fig. 6. Valid event fractions as function of system size at time points 1, 10, 100, and 250. (a) Results from the dimer system. (b) Results from the filament system.

suggesting the system is slower to reach equilibrium. At the non-equilibrium $t = 1$ time point, we again see a somewhat lower (about 0.44) but possibly asymptotically constant valid fraction, while $t = 10$ yields fractions only marginally below $t = 100$ and $t = 250$.

The non-equilibrium system allows us to examine practical efficiency for an instance in which our theoretical bounds do not apply. Systems with a significant fraction of unbinding events would be expected to yield linear run-time in any event, so we focused on the most extreme example of a system with low unbinding rate: one in which unbinding events are impossible. Our empirical tests unequivocally support the hypothesis that $v(n) = O(1)$ for such systems. We simulated each system for incrementally longer time spans until the fraction of valid events converged or the system became incapable of future events. Only filament assembly systems were considered, of sizes 100, 250, 500, 1000, and 5000 subunits. Fig. 7 plots valid fraction as a function of system simulation time. The plot shows apparent convergence to a constant valid fraction of about 0.37 over the course of those simulation runs. Thus, our system appears to exhibit linear run time even though it is a non-equilibrium system and therefore is not covered by our proof of efficiency.

Although the current prototype code is not highly optimized and we cannot attach great significance to exact run times and memory usage, we did perform some additional experiments to establish that true wall-clock run-time and memory usage reflect the theoretical results. Fig. 8 depicts run times and memory use as functions of system size for our fixed-concentration filament system run for one million valid events and 50, 100, 200, 400, and 800 monomers. Figs. 8(a) and (b) show results for our queue-based method and the

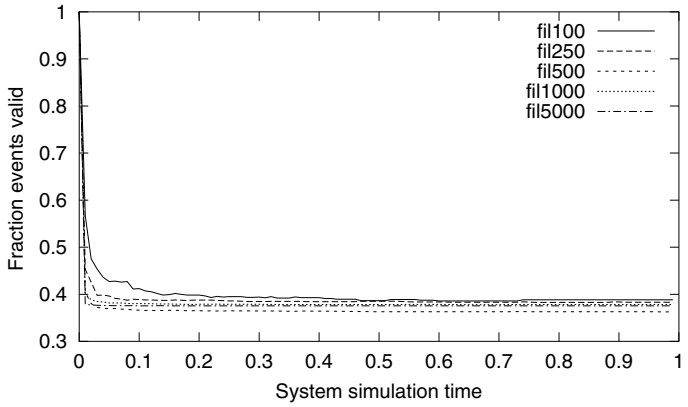

Fig. 7. Valid event fractions as function of filament system simulation time for a non-equilibrium system in which unbinding cannot occur.

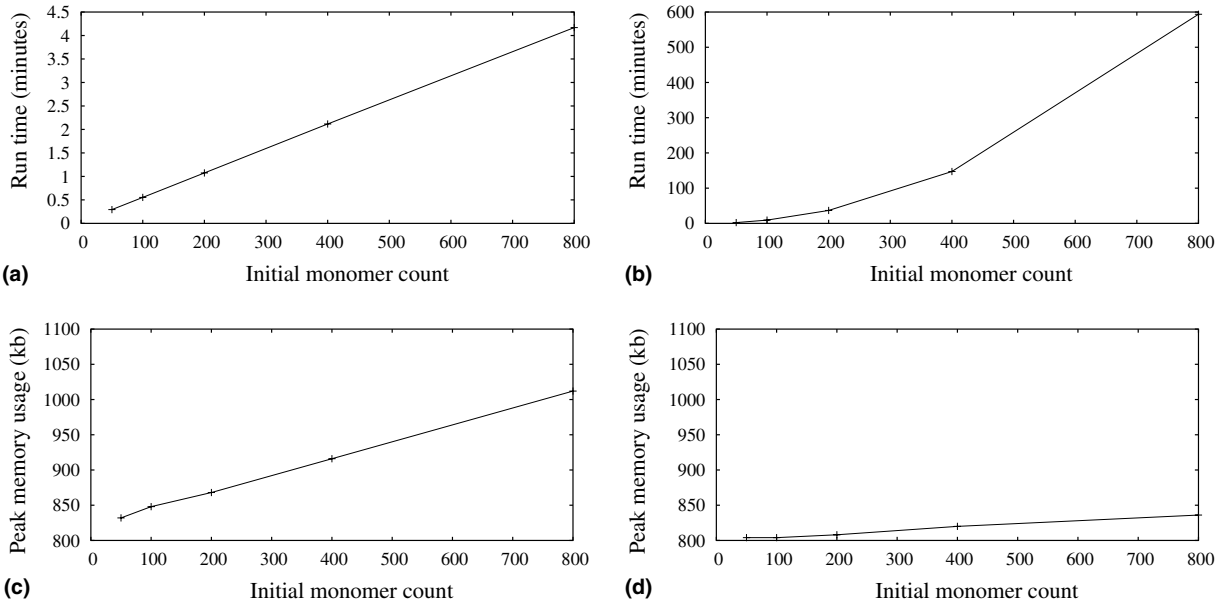

Fig. 8. Comparison of wall-clock run times and memory usage for a model filament system with queue-based and queue-less simulators. (a) Run times as functions of starting monomer count using our queue-based method; (b) run times using a queue-less implementation of the same system; (c) memory usage using our queue-based method; (d) memory usage using a queue-less implementation.

queue-less method, respectively, clearly establishing linear wall-clock run time for the queue method in comparison to the quadratic run time of the queue-less method. The two plots have different y-axis scales because of the very different run times of the two methods even for modest system sizes. Figs. 8(c) and (d) show memory usage for the two implementations. Both exhibit linear memory usage in system size, although with a constant approximately five times larger for the queue-based method. Fig. 9 shows log–

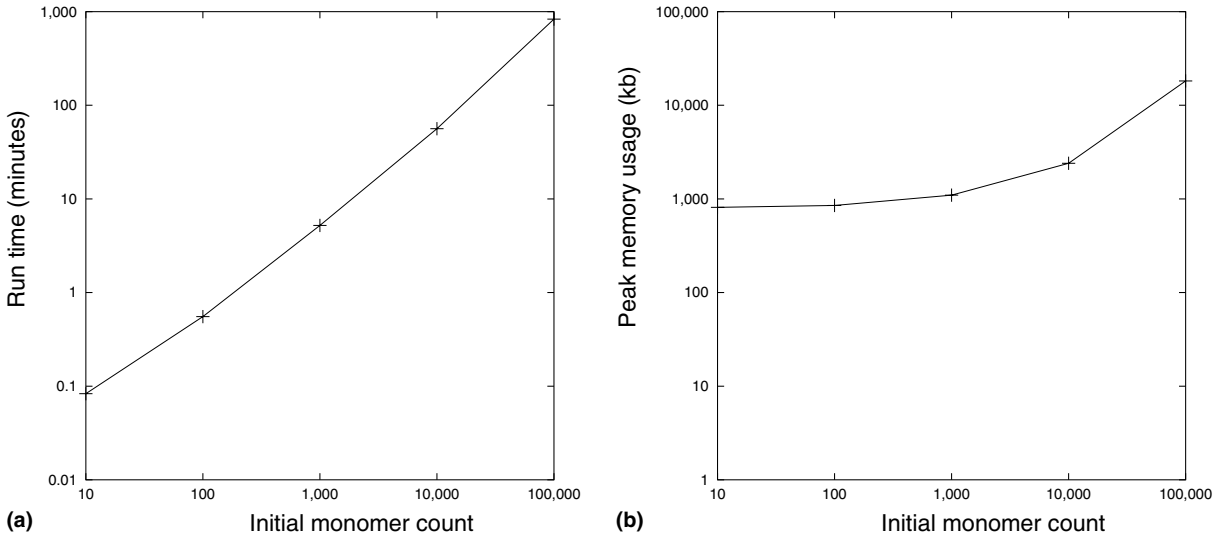

Fig. 9. Log–log plots of empirical run-time and memory usage of the queue-based simulator on a model filament system for a wide range of system sizes. (a) Run time as a function of starting monomer count; (b) memory usage as a function of starting monomer count.

log plots of run-time and memory usage of the queue method over a broader range of system sizes – 10, 100, 1,000, 10,000, and 100,000 monomers – extending well beyond the feasible range of the queue-less method. Fig. 9(a) shows clear linear growth in run time across this range of system sizes. Fig. 9(b) is consistent with a fixed memory cost added to a component proportional to system size. By the largest two system sizes, the O(m) component is dominant. Together, these plots show that the queue method should scale well across the feasible range of cell-scale systems, encompassing many that would be intractable for the queue-less method.

4. Discussion

We have presented a novel method for time- and space-efficient continuous-time simulation of self-assembly systems that improves on existing methods for an important class of systems. The method synthesizes theoretical contributions from the computer science and chemical engineering literature to tackle an important biological problem. For the first time, we show how to achieve simultaneous linear run-time and memory usage for a broad class of self-assembly systems. Our empirical results suggest the method will be feasible for cell-scale systems with large numbers of discrete assembly types, systems that would not have been approachable by prior methods requiring either quadratic run time or quadratic memory usage in the number of distinct species. Our methods are particularly well suited to systems of large or complicated assemblies, such as virus capsids, in which the numbers of distinct species co-existing at equilibrium could be on the order of the number of assembly subunits.

There are several open theoretical questions about our approach, particularly concerning the issue of valid event fractions. Although we show the valid fraction is bounded below by a fixed positive constant for systems at equilibrium, some important biological systems do not exist at thermodynamic equilibrium. Our approach could likely be extended to other systems by a more sophisticated argument allowing the bounding constant to vary depending on individual event times. One might also examine whether there

are bounds on how long a system can exist in domains for which our method requires super-linear run time per simulation step.

There are also practical issues in determining how our method can best be applied. It may be that the systems for which it performs poorly are sufficiently rare or artificial that it can be applied to general systems with little concern about encountering bad cases. These questions are likely to be resolved only with extensive experience with the performance on real-world systems. It might also be that other methods could be found that preserve the memory advantages of our queuing method but achieve efficiency in the domains where the queuing method performs poorly, either replacing our method or supplementing it in some problem domains.

While this paper is intended as a theoretical contribution, our primary goal is to address real-world problems. We plan to scale up to highly complex systems of biological interest, in particular the cellular cytoskeleton and virus capsids. In order to facilitate our own applied work and to make these tools available to a broader community, we are presently in the process of implementing a Java-based discrete event simulator that will implement our methods for a generalized representation of self-assembly systems. We hope that our theoretical contributions will be of use both to our own applications and to those of other researchers studying self-assembly dynamics.

Acknowledgments

This work was supported by US National Science Foundation Grant EIA-0320595. Rori Rohlfs was supported by the Merck Foundation Computational Biology and Chemistry Program at Carnegie Mellon University. We thank Peter Berget, Peter Prevelige, Roger Hendrix, and Mor Harchol-Balter for advice and support at various stages of this project.

References

[1] B. Berger, P.W. Shor, L. Tucker-Kellogg, J. King, Local rule-based theory of virus shell assembly, Proceedings of the National Academy of Sciences of the United States of America 91 (1994) 7732–7736.
[2] B. Berger, J. King, R. Schwartz, P.W. Shor, Local rule mechanism for selecting icosahedral shell geometry, Discrete Applied Mathematics 104 (2000) 97–111.
[3] K.L. Bortz, M.H. Kalos, J.L. Lebowitz, New algorithm for Monte-Carlo simulation of Ising spin systems, Journal of Computational Physics 17 (1975) 10–18.
[4] R. Brown, Calendar queues: a fast O(1) priority queue implementation for the simulation event set problem, Communications of the ACM 31 (1988) 1220–1227.
[5] D. Endres, A. Zlotnick, Model-based analysis of assembly kinetics for virus capsids and other spherical polymers, Biophysical Journal 83 (2002) 1217–1230.
[6] K.B. Erickson, R.E. Ladner, A. Lamarca, Optimizing static calendar queues, ACM Transactions on Modeling and Computer Simulation 10 (2000) 179–214.
[7] D.T. Gillespie, A general method for numerically simulating the stochastic time evolution of coupled chemical reactions, Journal of Chemical Physics 81 (1976) 2340–2361.
[8] I.J. Laurenzi, J.D. Bartels, S.L. Diamond, A general algorithm for exact simulation of multicomponent aggregation processes, Journal of Computational Physics 177 (2002) 418–449.
[9] I.J. Laurenzi, S.L. Diamond, Kinetics of random aggregation-fragmentation processes with multiple components, Physical Review E 67 (051103) (2003) 1–15.
[10] D.C. Rapaport, J.E. Johnson, J. Skolnick, Supramolecular self-assembly: molecular dynamics modeling of polyhedral shell formation, Computational Physics Communications 121–122 (1999) 231–235.
[11] V.J. Reddy, H.A. Giesing, R.T. Morton, A. Kumar, C.B. Post, C.L. Brooks, J.E. Johnson, Energetics of quasiequivalence: computational analysis of protein–protein interactions in icosahedral viruses, Biophysical Journal 74 (1998) 546–548.
[12] S.M. Ross, Probability Models, Academic Press, San Diego, CA, 2003.

[13] R. Schwartz, P.W. Shor, P.E. Prevelige Jr., B. Berger, Local rules simulation of the kinetics of virus capsid self-assembly, Biophysical Journal. 75:2626–2636..
[14] R. Schwartz, The local rules dynamics model for self-assembly simulation, Computer science Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2000 (also published as MIT-LCS-TR-800.).
[15] R. Schwartz, R.L. Garcea, B. Berger, ''Local rules'' theory applied to polyoma virus polymorphic capsid assemblies, Virology 268 (2000) 461–470.
[16] K.L. Tan, L. Thng, Snoopy calendar queue, in: Proceedings of the Winter Simulation Conference, 2000, pp. 487–495.
[17] G.M. Whitesides, Self-assembling materials, Scientific American (September) (1995) 146–149.
[18] G.M. Whitesides, B. Grzybowski, Self-assembly at all scales, Science 295 (2002) 2418–2421.
[19] A. Zlotnick, To build a virus capsid: an equilibrium model of the self-assembly of polyhedral protein complexes, Journal of Molecular Biology 241 (1994) 59–67.
[20] A. Zlotnick, J.M. Johnson, P.W. Wingfield, S.J. Stahl, D. Endres, A theoretical model successfully identifies features of hepatitis B virus capsid assembly, Biochemistry 38 (1999) 14644–14652.