Stochastic Cell Cycle Modeling for Budding Yeast

Tae-Hyuk Ahn*, Pengyuan Wang†, Layne T. Watson*+,
Yang Cao*, Clifford A. Shaffer*, and William T. Baumann‡
Departments of Computer Science*, Mathematics+, and Electrical and Computer Engineering‡
Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061
Cisco Systems, Inc. †
E-mail contact: thahn@cs.vt.edu

Keywords: Stochastic simulation algorithm (SSA), cell cycle, budding yeast, parallel computing, load balancing

Abstract

The budding yeast cell cycle provides an excellent example of the need for modeling stochastic effects in mathematical modeling of biochemical reactions. The continuous deterministic approach using ordinary differential equations is adequate for understanding the average behavior of cells, while the discrete stochastic approach accurately captures noisy events in the growth-division cycle. This paper presents a stochastic approximation of the cell cycle for budding yeast using Gillespie's stochastic simulation algorithm. To compare the stochastic results with the average behavior, the simulation must be run thousands of times. A load balancing algorithm reduces the cost for making those runs by 14% when run on a parallel supercomputer.

1. Introduction

The cell-division cycle is the sequence of events that take place in a eukaryotic cell leading to its replication. A growing cell replicates all its components and divides them into two daughter cells, so that each daughter has the information and machinery necessary to repeat the process [1]. The cell cycle of the unicellular budding yeast, *Saccharomyces cerevisiae*, has been extensively studied. Mathematical modeling and computational methods are needed to explain the detailed workings of complex yeast control systems. Deterministic mathematical modeling for the budding yeast cell cycle gives the average behavior of populations of dividing cells [2]. However, some major regulatory proteins occur in small numbers. As a result, individual cells have in ways different from the average. Thus, the stochastic approach provides more accurate results than does the deterministic one [3]. In addition, when cell cycle controls are compromised by mutation, random fluctuations must be accounted for when modeling the effects of the mutants. Therefore, it is desirable to translate the deterministic budding yeast model into a stochastic model, and simulate the model with an appropriate stochastic method.

Gillespie's stochastic simulation algorithm (SSA) ([4], [5]) is well known. It uses Monte Carlo methods to simulate the chemical reactions. The SSA is an asymptotically exact stochastic method to simulate chemical systems, but the SSA is often slow because it simulates every reaction. Since the SSA was first published, there have been many attempts to improve the computational efficiency ([6], [7], [8]). However, the core ideas remain the same.

Stochastic methods require that the model be cast in terms of population because they consider reactions with individual molecules. The problem is, however, that ODE models are usually based on concentration values. Therefore, a concentration-based formulation for a model has to be changed into a population-based formulation for simulation using a stochastic method. Previous work [9] explained the conversion process using JigCell [10] in detail. StochKit [11] is used to do stochastic simulation of the converted budding yeast model. StochKit supports various approximate simulation methods based on the SSA, but only the exact SSA is used to get precise results.

Because the SSA simulates every time step, the SSA is much slower than a deterministic simulation. Moreover, the simulation must be run thousands of times to generate enough data to determine the correct distribution of the behavior. Therefore, it is desirable to run many independent SSA simulations in parallel. StochKit supports MPI for parallel SSA runs, but the user assigns jobs for each processor. Sometimes, the processor times for individual runs are quite different. The result is poor parallel efficiency. This paper presents a load balancing algorithm to significantly improve the parallel efficiency.

The budding yeast cell cycle model is presented in Section 2, and the SSA is explained in Section 3. The load balancing algorithm is presented in Section 4. Section 5 gives new biological results, and Section 6 concludes.

2. Cell Cycle Model

The molecular machinery of eukaryotic cell cycle control is known in more detail for budding yeast, *Saccharomyces cerevisiae*, than for any other organism.

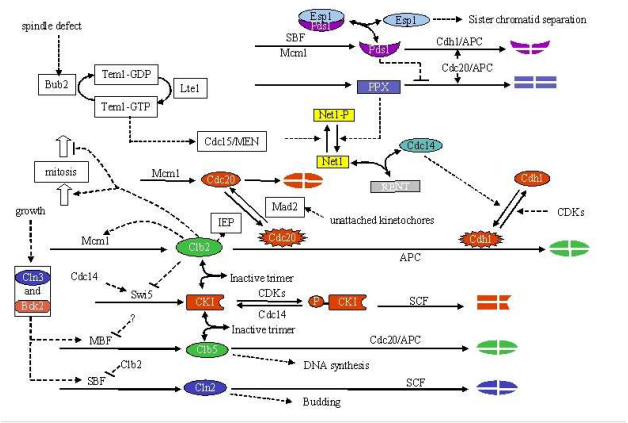

Figure 1. Wiring diagram of budding yeast.

Therefore, the unicellular budding yeast is an excellent organism for which to study cell cycle regulation. Molecular biologists have dissected and characterized individual components and their interactions to derive a consensus picture of the regulatory network of budding yeast. Figure 1 shows the wiring diagram for the budding yeast model [10]. The diagram should be read from bottom-left toward top-right. Solid arrows represent biochemical reactions, and dashed lines represent how components may influence one another.

Computational biologists traditionally formulate the complex regulatory network as a set of nonlinear ordinary differential equations (ODEs) according to the wiring diagram [2]. This formulation is first translated to a population-based model using primitive reaction equations as discussed in Section 5.1. The population-based formulation is then simulated with an appropriate stochastic method, as explained in the next section.

3. SSA

Suppose a biochemical system or pathway involves N molecular species $S_1, \ldots, S_N$. $X_i(t)$ denotes the number of molecules of species S_i at time t. People would like to generate the evolution of the state vector $X(t) = (X_1(t), ..., X_N(t))$ given that the system was initially in the state vector $X(t_0)$. Suppose the system is composed of M reaction channels $R_1, \ldots, R_M$. In a constant volume Ω, assume that the system is well-stirred and in thermal equilibrium at some constant temperature. There are two important quantities in reaction channels R_j: the state change vector $v_{\cdot j} = (v_{1j}, ..., v_{Nj})$, and propensity function a_j. v_{ij} is defined as the change in the S_i molecules' population caused by one R_j reaction, and $a_j(x)dt$ gives the probability that one R_j reaction will occur in the next infinitesimal time interval $[t, t + dt)$.

The SSA simulates every reaction event ([4], [5]). With $X(t) = x$, $p(\tau, j|x, t)d\tau$ is defined as the probability that the next reaction in the system will occur in the infinitesimal time interval $[t + \tau, t + \tau + d\tau)$, and will be an R_j reaction. By letting $a_0(x) \equiv \sum_{j=1}^{M} a_j(x)$, the equation

$$p(\tau, j|x, t) = a_j(x) \exp(-a_0(x)\tau)$$

can be obtained. A Monte Carlo method is used to generate τ and j. On each step of the SSA, two random numbers r_1 and r_2 are generated from the uniform (0,1) distribution. From probability theory, the time for the next reaction to occur is given by $t + \tau$, where

$$\tau = \frac{1}{a_0(x)} \ln\left(\frac{1}{r_1}\right).$$

The next reaction index j is given by the smallest integer satisfying

$$\sum_{j'=1}^{j} a_{j'}(x) > r_2 a_0(x).$$

After τ and j are obtained, the system states are updated by $X(t + \tau) := x + v_j$, and the time is updated by $t := t + \tau$. This simulation iteration proceeds until the time t reaches the final time.

4. Load Balancing Parallel Implementation

Since this is a stochastic algorithm, the time required for each run is a little bit different. When hundreds of trajectories are assigned statically to a processor, the total simulation times can be quite different across the processors, causing a serious load imbalance. This section presents a dynamic load balancing algorithm to more evenly distribute work to the processors.

The basic idea of the load balancing algorithm is a master/slave paradigm that dynamically adjusts the task chunk size. Pseudocode (with constants tuned for the problem at hand) for the load balancing follows.

```
Algorithm GetTask(n, p, task)
▷ Input: n = number of remaining tasks,
         p = number of processors
▷ Output: chunk = number of tasks for a processor
begin
    setPoint ← floor (n/p)
    if setPoint > 99 then
        chunk ← floor (setPoint × 0.8)
    else if setPoint > 5 then
        chunk ← floor (setPoint × 0.5)
    else then
        chunk ← 1
    end if
end
```

The idea of varying the task chunk size as the size of the task queue decreases is well known in parallel computing, where it is known as guided self-scheduling.

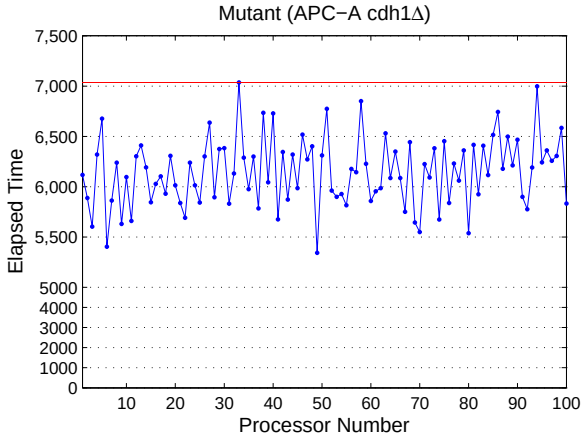

Figure 2. Static workload distribution.

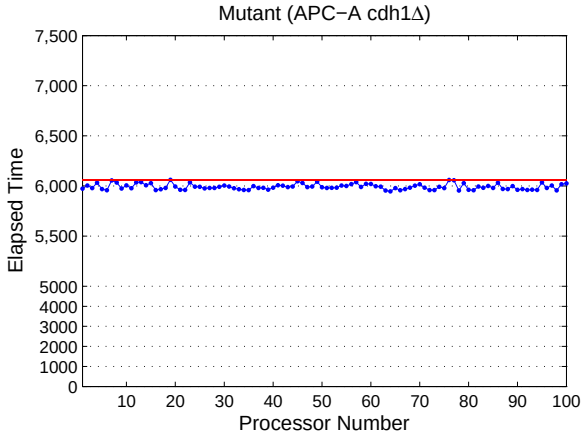

Figure 3. Dynamic workload distribution.

It is typical in guided self-scheduling algorithms to use constants tuned to the application, such as shown on lines 6-9. Figures 2 and 3 clearly show the advantage of the load balancing algorithm. Using Virginia Tech's 2200 processor (2.3 GHz PowerPC 970FX) System X, 100 worker processors were used for 10,000 stochastic simulations for the budding yeast double mutant APC-A Cdh1Δ. Figure 2 shows the time that each processor required to complete its assigned tasks. The variance between processors is high. Figure 3 shows equivalent times using guided self-scheduling. Here, the variance in processor times is negligible, even though individual runs of the simulation have large fluctuations in their times.

TABLE 1. EXECUTION TIMES FOR SIMPLE DISTRIBUTION.

Parallel Runtime	Time (seconds)
T_p(average)	6144.93
T_p(max)	7035.95
Wall clock time	**7040.72**

TABLE 2. EXECUTION TIMES FOR LOAD BALANCING DISTRIBUTION.

Parallel Runtime	Time (seconds)
T_p(average)	5992.43
T_p(max)	6061.78
Wall clock time	**6066.84**

Tables 1 and 2 show the average and total times for the static and dynamic workload distributions. The guided self-scheduling algorithm reduces system resource use by approximately 14%.

5. Cell Cycle Results

5.1. Implementation

As was mentioned briefly in Section 2, stochastic methods require the model to be in terms of population because they consider reactions with individual molecules. Because the original budding yeast model is based on normalized concentration values, a conversion process from an ODE model into a model in terms of number of molecules is needed. The conversion process is done using JigCell, and consists of two phases, *unit checking* and *model conversion* [9]. Unit checking verifies physical unit consistency inside the model. Model conversion converts the model by changing values of species and parameters based on the unit information.

After creating the population-based budding yeast model, there is a technical issue that must be addressed. An *event* is triggered when some condition is met. There are events defined to divide the cell or mark checkpoints within the cell cycle stages. A typical deterministic event has the form:

```
if (X > threshold)
  then (Event is triggered)
```

Because of the random nature of stochastic simulation, as illustrated in Figure 4, unwanted events can be triggered when the deterministic formulation for an event is used in the stochastic model. In Figure 4, an unwanted event (B) can be triggered with a wanted event (A) by using a deterministic event handling equation.

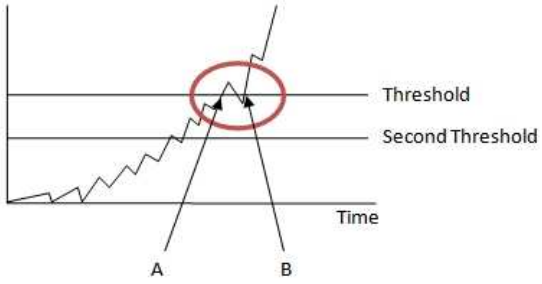

Figure 4. Event handling.

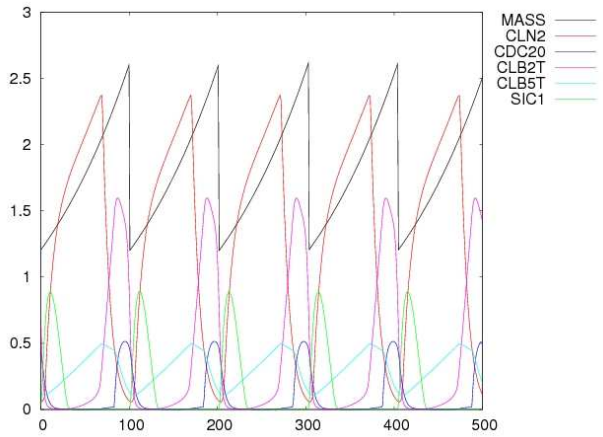

Figure 5. Deterministic cell cycles.

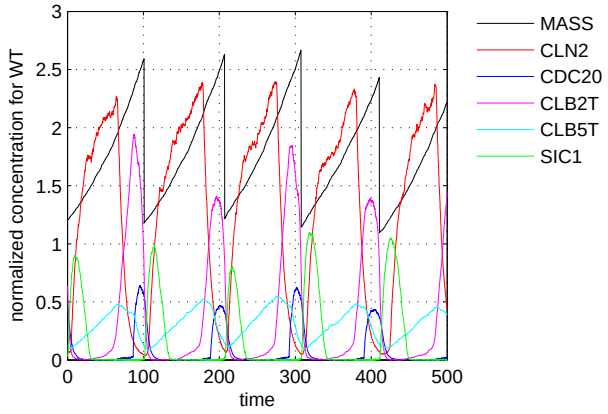

Figure 6. Stochastic cell cycles.

To prevent unwanted events, the event logic has to be rewritten to tolerate the situation where the value of X oscillates around the threshold. A second threshold value can be defined from the threshold and the direction of the test (greater than or less than). For budding yeast, this second threshold equals 0.5*threshold (for a greater than test) or 1.5*threshold (for a less than test). For instance, the event code above would be changed to:

```
if (X < second threshold)
  then (EventFlag ← TRUE)
if (X > threshold AND EventFlag = TRUE)
  then (event is triggered;
        EventFlag ← FALSE)
```

StochKit [11] was used to do stochastic simulation of the converted budding yeast model, using the SSA option for the most precise results. JigCell can generate the StochKit model file by using the population based budding yeast model file.

5.2. Wild Type Simulation Results

To compare the stochastic results with deterministic cell cycle simulation, mass and several representative species' trajectories are shown in Figures 5 and 6. The deterministic result is from the XPP ODE simulator using JigCell. For comparison, the stochastic simulation results are converted back to normalized concentrations.

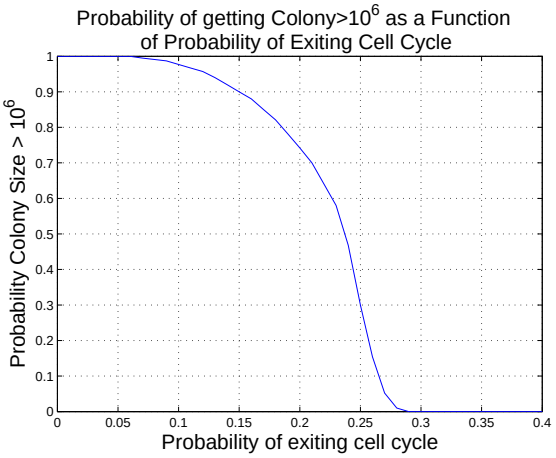

Figure 7. Probability of getting colony.

5.3. Mutants

For the mutants considered in this paper, stochastic simulations show that often they neither divide endlessly nor fail to divide at all, meaning that they divide for several cycles, then stop dividing. The number of cycles until stopping division is different for different runs of the simulation. For a statistical analysis, 10,000 independent simulations are executed from the same initial point using the load-balancing parallel algorithm.

Let the random variable X denote the number of cell divisions before the cell stops dividing, and assume that the probability p of not dividing is constant and independent of the cell's previous history. Then X has a modified geometric distribution given by

$\mathrm{P}(X = n) = p(1-p)^n$ where $n = 0, 1, ...$

Further, the probability of having n or more cycles is given by

$\mathrm{P}(X \geq n) = (1-p)^n$ where $n = 0, 1, ...$

If either probability is plotted on a log scale against n, it will be a straight line with a slope of $\log(1-p)$. In the analysis of the mutant simulations, a best least squares fit straight line is used to extract the slope and estimate the value of p.

In wet lab experiments, the viability of mutants is assessed by determining whether single mutant cells could grow into a colony. To determine the relationship between viability and the probability of ceasing to divide, simulation is used to determine the probability of one cell producing a colony of size greater than 10^6 in 32 division cycles. 32 division cycles correspond roughly to incubating the cells on a plate for two days. A 1 mm^3 colony has about 20×10^6 cells. Therefore, it is assumed

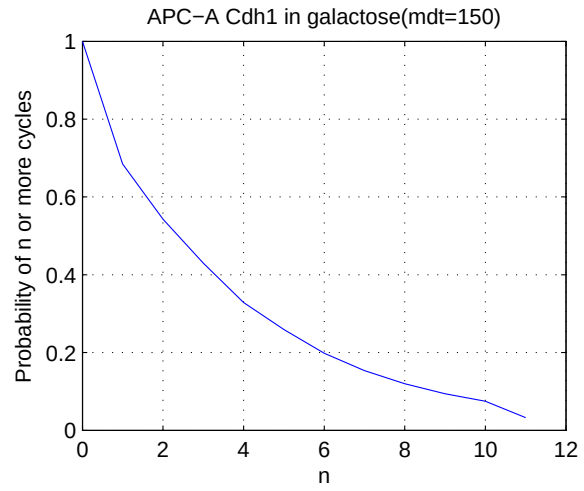

Figure 8. $P(X \geq n)$**.**

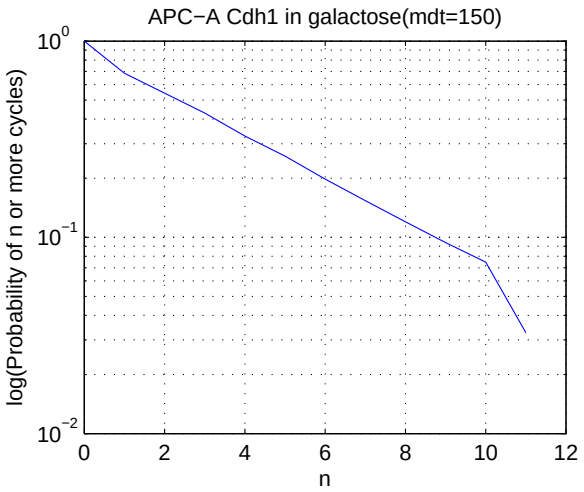

Figure 9. $\log(P(X \geq n))$

that 0.05 mm^3 can be visible. The results of these simulations are summarized in Figure 7. From this figure, the ability to observe colonies has a nearly switch-like characteristic with the switching point near $p = 0.25$.

It is interesting that some mutants have different fates in glucose and galactose. The budding yeast cell growth rate is different with glucose and galactose. In general, the mass doubling time is 90 minutes for glucose, and 150 minutes for galactose. For example, mutant APC-A Cdh1Δ has different fates in glucose and galactose. From Cross' paper [12], APC-A Cdh1Δ can be partially rescued in a poor galactose medium, with 8% viability of spores. In the deterministic model, the double mutant APC-A Cdh1Δ is inviable with telophase arrest, but if the mass doubling time is changed into 160 min, then the mutant becomes viable. The problem with the deterministic model is that only viable or inviable are evidenced from a single initial condition.

In a stochastic simulation using the observed mass doubling time of 150 min in glucose, the fluctuation in Clb2, sic1, and Cdc6 may allow some cells to achieve levels that enable them to exit from mitosis. This mutant is simulated using the parameters ka20$'' = 0$, kscdh $= 0$, init CDH1T $=$ CDH1 $= 0$ [2]. In glucose $(\text{MDT} = 90)$, this mutant does not divide and is inviable. In galactose $(\text{MDT} = 150)$, the probability of surviving at least n cycles has a modified geometric distribution as seen in Figures 8 and 9. From the log scale graph, the least squares fit slope is found to be $-1/9$, which gives an estimated probability to cease dividing of $p = 0.22$. From Figure 7, this implies that the probability of a single cell forming a colony is 65%, which does not match the experimental result. However, stochastic simulation explains mixed viability results that a deterministic simulation cannot.

6. Conclusions and Future Work

The budding yeast stochastic simulation results reported here, while limited, show important characteristic aspects of cell cycle empirical data, such as mixed mutant viability. Because random fluctuations are important to acculately simulate mutants, or where major regulatory proteins occur in small numbers, the stochastic approach is more realistic and accurate than the deterministic approach for modeling the budding yeast cell cycle. The guided self-scheduling load balancing algorithm is effective for managing the large numbers of SSA trajectories required by the stochastic approach.

The data collected here are only the first step toward calculating population doubling times, and probabilities for successful colony formation of the various mutants. Those probabilities are one of the tests for verifying that the simulations match experimental results. The results are also perhaps skewed by the fact that all simulation runs begin with a static initial condition that might not be representative of the true population. In future work, the author will calculate population statistics and use them to generate apppropriate initial conditions.

References

[1] Murray, A. and Hunt, T., 1993, *The Cell Cycle. An Introduction*, Oxford University Press, New York, USA.

[2] Chen, K. C., Calzone, L., Csikasz-Nagy, A., Cross, F. R., Novak, B, Tyson, J. J., 2004, "Integrative analysis of cell cycle control in budding yeast", *Mol. Biol. Cell*, **15**, 3841–3862.

[3] McAdams, H. H. and Arkin, A., 1997, "Stochastic mechanisms in gene expression", *Proc. Natl. Acad. Sci.*, **94**, 814–819.

[4] Gillespie, D. T., 1977, "Exact Stochastic Simulation of Coupled Chemical Reactions", *Journal of Physical Chemistry*, **81**, 2340–2361.

[5] Gillespie, D. T., 1976, "A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled

Chemical Reactions", *Journal of Computational Physics*, **22**, 403–434.
[6] Gibson, M. A. and Bruck, J., 2000, "Efficient exact stochastic simulation of chemical systems with many species and many channels", *Journal of Physical Chemistry*, **104**, 1876–1889.
[7] Gillespie, D. T., 2001, "Approximate accelerated stochastic simulation of chemically reacting systems", *Journal of Chemical Physics*, **115**, 1716–1733.
[8] Cao, Y., Gillespie, D. T. and Petzold, L. R., 2005, "The slow-scale stochastic simulation algorithm", *Journal of Chemical Physics*, **122**, 014116.
[9] Wang, P., Randhawa, R., Shaffer, C. A., Cao, Y., Baumann, W. T., 2008, "Converting macromolecular regulatory models from deterministic to stochastic formulation", In *Proceedings of the 2008 Spring Simulation Multiconference*, SpringSim '08. ACM, New York, 385-392.
[10] JigCell, *Virginia Tech*, http://jigcell.biol.vt.edu.
[11] Li H., Cao Y., Petzold L., Gillespie, D., 2007, "Algorithms and software for stochastic simulation of biochemical reacting systems", *Biotechnology Progress*, .
[12] Cross, F.R., 2003, "Two redundant oscillatory mechanisms in the yeast cell cycle", *Dev. Cell*, **4**,741-752.
[13] Ahn, T., Cao, Y., Watson, L. T., 2008, "Stochastic Simulation Algorithms for Chemical Reactions", In *Proceedings of the 2008 International Conference on BioInformatics & Computational Biology*, Arabnia, H. R, WORLDCOMP'08, 431-436.
[14] Rathinam, M., Petzold, L., Cao, Y. Gillespie, D., 2003, "Stiffness in stochastic chemically reacting systems: The implicit tau-leaping method", *Journal of Chemical Physics*, **119**, 12784–12794.
[15] Rao, C. V and Arkin, A. P, 2003, "Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm", *Journal of Chemical Physics*, **118**, 4999–5010.
[16] Tzafriri, A. R. and Edelman, E. R., 2004, "The total quasi-steady-state approximation is valid for reversible enzyme kinetics", *Journal of Theoretical Biology*, **226**, 303–313.

Biography

Tae-Hyuk Ahn is a Ph.D. student in the Department of Computer Science at Virginia Polytechnic Institute and State University. After he received a B.S. in Electrical Engineering from Yonsei University in Korea, he worked for Samsung SDS for 4 years. He received his M.S. in Electrical and Computer Engineering from Northwestern University. His research interests are stochastic modeling of biological system, numerical analysis, and parallel computation.

Pengyuan Wang received his Bachelor of Engineering degree in Computer Science and Technology from Beijing University of Technology in July 2006. In the same year he started graduate study at Virginia Tech, being a research assistant in the stochastic modeling group. He graduated with a Master of Science degree in Computer Science in June 2008. He is now at Cisco Systems, Inc.

Layne T. Watson is a Professor in the Departments of Computer Science and Mathematics at Virginia Polytechnic Institute and State University. He received a B.A. in psychology from the University of Evansville in 1969, and a Ph.D. in mathematics from the University of Michigan, Ann Arbor, in 1974. His current research interests include numerical analysis, nonlinear programming, mathematical software, solid mechanics, fluid mechanics, image processing, parallel computation, and bioinformatics. Dr. Watson is a Fellow of the IEEE and the National Institute of Aerospace. He has more than 250 refereed journal publications in the areas of numerical analysis, nonlinear programming, mathematical software, parallel computation, image processing, bioinformatics, and solid and fluid mechanics. Editorial service includes SIAM Journal on Optimization, ORSA Journal on Computing, Computational Optimization and Applications, Evolutionary Optimization, Engineering Computations, and International Journal of High Performance Computing Applications.

Yang Cao received his Ph.D. degree in computer science from the University of California, Santa Barbara in 2003. He is an Assistant Professor in the Computer Science Department at the Virginia Polytechnic Institute and State University. His research focuses on the development of multiscale, multiphysics stochastic modeling and simulation methods and tools that help biologists build, simulate and analyze complex biological systems.He has published around 30 refereed journal articles.

Clifford A. Shaffer (Senior member, IEEE and ACM) received his Ph.D. from the University of Maryland. He is a Professor of Computer Science at Virginia Tech. His current research interests are related to developing Problem Solving Environments for engineering and science applications. Specific topics include data structures and algorithms for visualization, collaborative computing, component architectures, and user interfaces for specifying models and computations.

William Baumann received his Ph.D. degree in electrical engineering from the Johns Hopkins University. He is an Associate Professor in the Department of Electrical and Computer Engineering at Virginia Tech. His current research interests are in modeling, analysis and identification of biological systems in both deterministic and stochastic settings.