DISSERTATION

STATISTICAL UPSCALING OF STOCHASTIC FORCING IN MULTISCALE, MULTIPHYSICS MODELING

Submitted by

Charles T. Vollmer

Department of Statistics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2019

Doctoral Committee:

Advisor: Don Estep

Simon Tavener
Jay Breidt
Dan Cooley

ABSTRACT

STATISTICAL UPSCALING OF STOCHASTIC FORCING IN MULTISCALE, MULTIPHYSICS MODELING

Modeling nuclear radiation damage necessarily involves multiple scales in both time and space, where molecular-level models have drastically different assumptions and phenomena than continuum-level models. In this thesis, we propose a novel approach to explicitly coupling these multiple scales of the microstructure damage of radiation in materials. Our proposed stochastic process is a statistical upscaling from physical first principals that explicitly couples the micro, meso, and macro scales of materials under irradiation.

ACKNOWLEDGEMENTS

I am very fortunate for many loving family and friends in my life. Without them, my life would not be rich and meaningful.

DEDICATION

I would like to dedicate this thesis to my mother and father. They planted the seeds responsible for this work, and nurtured their growth. This work would also not have been possible without the support of my wife, and children, throughout it all.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction and Background Literature Review

We are chiefly interested in modeling how the process of radiation damage proceeds in nuclear fuels, such as UO_2. In such fuels, particle irradiation induces microstructure changes in the lattice of the fuel rods as a result of atomic collisions. The creation of a large amount of these point defects in the fuel is a discrete process which occurs by so-called collision cascades (shown below, in Fig:**??**).

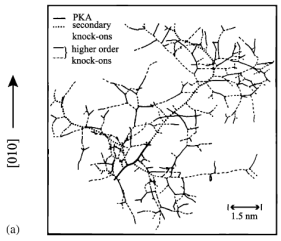

Figure 1.1: Example of a collision cascade

Microstructure damage in nuclear materials is a complex process that presents formidable modeling challenges due to its multiscale nature. Fundamentally driven by the atomic displacement collision cascades, microstructure damage occurs from various physical processes, acting through a range of scales [1]. At the molecular level, binary collision cascades have been the defacto manner of modeling the irradiation damage introduced by individual collision cascades [2–4]. Thus, we have long-standing and well-validated manners of simulating individual micro-scale events [5, 6].

Between the micro scale, which include atomic events that occur with nanoseconds, and the macro scale, which encompass long-term space and time behavior, exists a coupling scale: the meso scale. At this scale, the long-range spatial and time interactions of the behavior and evolution of microstructure damages such as void growth and shrinkage, void-void interactions, as well as spontaneous nucleation and growth of a large population of voids occur [7]. At the macro scale, the numbers of events and motion become so large that only the bulk properties of the statistical

fluctuations, eg. means, are significant. An example is how statistical mechanics transforms into thermodynamics. Thus, an intermediary scale is needed to appropriately handle the interactions between the large, fast statistical fluctuations and the smooth long-term behaviors of the continuum models.

At the macro level, continuum models of differential equations are used to model the material consequences of radiation damage, e.g. in terms of heat flow, stress, and so on. Currently, phase field models, such as Cahn-Hilliard equations, have been proposed to model the more advanced behavior and evolution of microstructure damages [8, 9]. Typical to this approach is the diffuse description of interfacial free energy. These phase field models, however, do not take collision cascades as a vacancy generation scheme, and do not model varying time nor spatial scales. While these models can be used to model both the micro and meso individually, complex issues arise, and stringent assumptions must be made in this approach, and the attempt to couple them together.

Our modeling approach, on the other hand, is explicitly based on coupling a mesoscale stochastic process to the microscale collision models. Our model utilizes these long-standing binary collision cascades as a source of vacancy generation and appropriately couples the micro and meso scales together into one stochastic process, whose properties hold over both small and large time and space scales.

1.1 Micro Scale Physics and Models

We can start out by defining a “collision spike." The atoms that have been knocked out of the lattice, see Fig:**??**, have a high amount of kinetic energy. Atoms physically collide into other atoms, thus causing a “cascade." When these cascading atoms become of sufficiently low energy they cause bursts of displacements in the solid known as collision spikes, as they deposit their last bit of energy over confined material volumes. This displacement causes a sort of pothole in the material, albeit at an atomic scale. (However, these newly displaced atoms, from the spike, are not of high enough energy to continue cascading, rather moving into a sort of shell around the pothole.

Not unlike a crater from a meteor.) After cooling, each collision spike leaves behind a vacancy-rich region surrounded by an interstitial-rich shell, Fig:**??**.

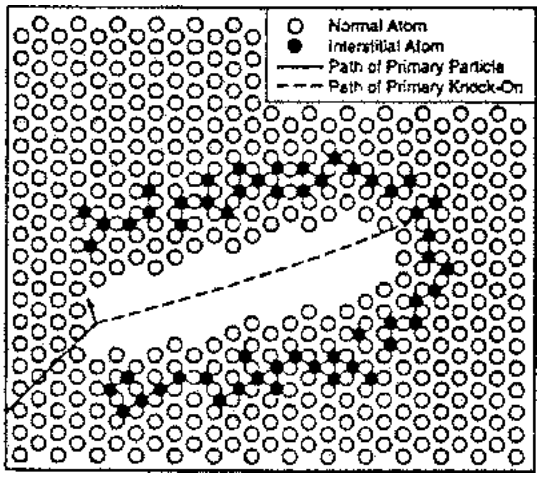

Figure 1.2: A thermal displacement spike: “collision spike."

1.1.1 Cascade Process Models

These binary collision cascades have been studied and well understood for six decades [2–4]. As clearly seen in Figure **??**, each branching point of a collision cascade represents a binary collision event and each branch corresponds to a mean free path of a displaced atom. A Monte Carlo computer program was created in the mid-80’s, called TRIM [5, 6], which generates these cascades in any desired solid lattice medium and corresponding ion. Our task is to utilize this widely-used code and to expand the model of this single cascade damage event to include the dynamics of multiple events over a large lattice volume, over large time intervals.

Our first assumption is that each cascade is represented by a single event, which we call a “spike." These spikes can be thought of as big pot holes in the atomic lattice. (Similarly, the collision spikes described above can also be thought of as pot holes in the atomic lattice. Then, these newly defined spikes are like a sum of all of the smaller collision spikes.) These spikes occur randomly in time and space and with a random strength (as shown below:)

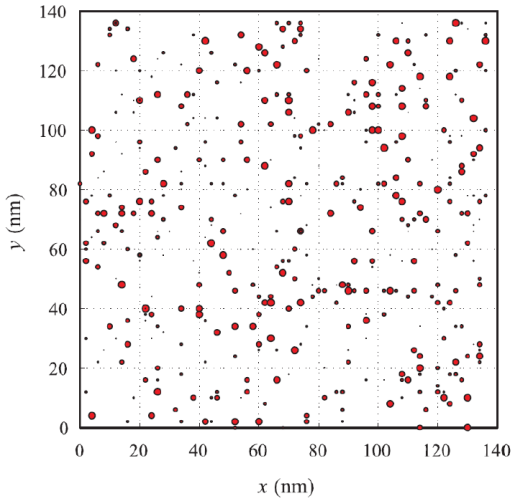

Figure 1.3: Example of random cascades in a unit area over period T.

1.2 Macro Scale Physics and Models

Phase-field models, and methods, have been a powerful tool in the materials science literature over the past two decades to mathematically characterize physical dynamical systems where interfacial boundaries are present. The physical problem we are trying to solve, microstructure damage of materials under irradiation, lends itself towards a mathematical characterization in terms of diffusion and interfacial motion. These are complex properties that are not easily represented using a single differential equation model, and is a large reason for the successful use of a phase field approach. Phase field models were brought into this area of research after other successful applications in materials science where these two descriptions, diffusion and interfacial motion, were paramount. In our work, we discuss the application of phase field modeling of microstructure evolution in our study of radiation effects in materials.

This class of problems presents many computational challenges even after mathematical modelers have appropriately captured the physics. This is because the interface motion calculations tend to be time consuming in just two or three dimensions. To help reduce the problem, an order parameter is used as a diffuse representation for the interface in the phase-field approach. This

greatly reduces the problem of interface motion in the classic sharp-interface description of these problems.

Another large advantage to using a phase field approach, in this context, is the model's ability to spatially-resolve realizations of the evolving microstructure. This lends it the potential to capture the complecting aspects of defect and microstucture damage in irradiated solids. Rokkam, et. al. [**?**, **?**], developed the framework and approach that we will be utilizing here within. Within their framework, they developed a method to merge defects diffusion and interface evolution into a type C framework that seeks to combine the dynamics of both Cahn-Hilliard and Allen-Cahn models [**?**, **?**, **?**, **?**, **?**].

1.2.1 Phase-field model description

The free-energy functional of these phase-field models takes the following form:

$$\Psi[c_v, \eta] = N \int_\Omega [h(\eta)\psi^m(c_v) + \omega(c_v, \eta) + \kappa_v |\bigtriangledown c_v|^2 + \kappa_\eta |\bigtriangledown \eta|^2 + \psi^{el}(c_v, \eta)]\, d\Omega \tag{1.1}$$

where the first two terms and last term account for the system bulk energy and elastic effects in the system, respectively. The two gradient terms, however, are what account for the interfacial energy contributions. The vacancy concentration field, $c_v(x,t)$, is modeled using the following generalized diffusion equation:

$$\frac{\partial c_v}{\partial t}(x,t) = \bigtriangledown \cdot M_v \bigtriangledown \frac{1}{N}\frac{\delta \Psi}{\delta c_v} + \xi(x,t) \tag{1.2}$$

and the long range order parameter of the system, $\eta(x,t)$, is modeled using the "phenomenological" Allen-Cahn equation:

$$\frac{\partial \eta}{\partial t}(x,t) = -L\frac{\delta \Psi}{\delta \eta} + \zeta(x,t). \tag{1.3}$$

It is easy to see that this modified Cahn-Hilliard equation, Eq. 1.2, does *not* account for the generation of vacancies or the direct formation of voids. Both of these phenomena occur as a result of the cascade process, detailed in Section **??**, in irradiated materials. To account for this, such terms can simply be added to the right-hand side of Eq. 1.2, to represent the production of vacancies due to the cascade process:

$$\frac{\partial c_v}{\partial t}(x,t) = \bigtriangledown \cdot M_v \bigtriangledown \frac{1}{N}\frac{\delta \Psi}{\delta c_v} + \xi(x,t) + P_v(x,t) \tag{1.4}$$

Of special note to us, as well, is that these explicit equations do *not* consider other vary common processes such as the spontaneous annihilation of vacancies, nor the direct formation of voids due to high-energy cascades. We will discuss how the addition of stochastic meso models to this framework can add the ability to explicitly incorporate these processes.

1.2.2 Phase-field Model Strengths and Deficiencies

This phase-field modeling framework has been shown to capture and reproduce several important dynamics of microstructure characteristics of void dynamics under irradiation. Chiefly among them are the nucleation and growth of voids in metals caused by the ongoing production of radiation defects; vacancy concentrations exceeding the thermal equilibrium values. Figure 1.4, below, shows the implementation of these phase field models representing this void nucleation.

Rokkam, et. al. [**?**, **?**], note that a major setback to the application and use of these models and modeling framework in real materials engineering has been the lack of material-specific model parameters. The derivation of these fundamental model parameters, and the subsequent quantification of the model predictions, is of paramount importance for the development of next-generation mesoscopically-informed materials modeling codes. These model parameters and model predictions must also include, and be based on, input parameter uncertainties. This is conceptually achieved by theoretically connecting these phase field models, described in Section **??** below, to their equivalent sharp interface models (free boundary problems which can be described using measurable quantities) and through an asymptotic analysis. This is achieved using an asymptotic

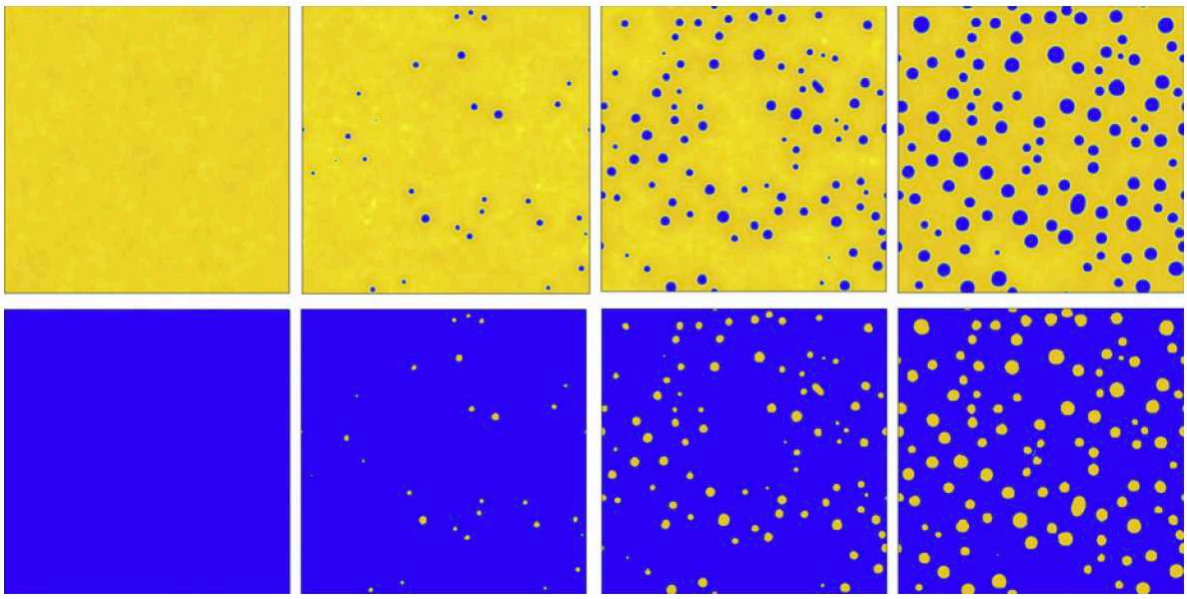

Figure 1.4: Snapshots showing the nucleation and growth of voids in the presence of on-going cascades modeled using Eq.**??**

matching technique. Finally, these model input parameters are generally obtained using live experiments, or from the outputs of lower-scale atomistic models.

Wang, et. al [], have investigated and performed an asymptotic analysis of a class of these models that have been used to model void formation under irradiation, by fixing the model parameters using conventional properties of the metal materials of interest. Then, using a stochastic collocation-based uncertainty quantification (UQ) method, they investigate the model's response to input material parameter uncertainties.

Figure 1.5, below, illustrates the use of these phase field models to model microstructure damage evolution in materials under irradiation; specifically void growth and the corresponding phase fields.

1.3 Meso Scale Physics and the Missing Models

Having understood all of the aforementioned physics and modeling approaches, the astute reader should undoubtedly be aware by now that the context of microstructure evolution in materi-

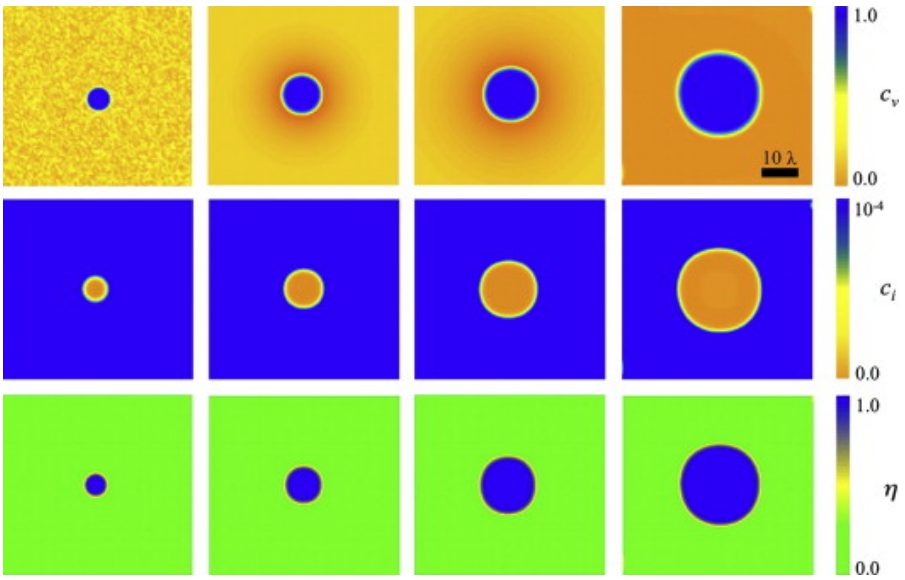

Figure 1.5: Snapshots showing the varying phase fields during the growth of voids in the presence of ongoing cascades modeled using Eq.**??**

als under irradiation is a complex class of phenomena that comprises great time scales, as well as spatial scales. What is missing from the current literature is work that bridges the divide between the time and space scales that are currently being addressed by the models detailed in Sections 1.1 and 1.2. Physics at this in-between scale, the meso scale, need to be captured and explicitly modeled.

The motivation of this entire body of work is to supplement the current modeling methodology with statistical and stochastic models that bridge these time and space scales, extending the statistical states of the micro processes to be fed into the models at the higher macro scales. The physics at these micro and macro scales are well understood and captured by the current mathematics. The physics at this meso scale is also well understood, but currently not being captured by mathematical models. We argue that the explicit physics do not need to, necessarily, be modeled mathematically, but rather a new approach devised: where we upscale the statistical properties of the modeled, micro phenomena, maintain the fidelity of the stochastic nature of the physics at the meso, and provide the upscaled statistical space and time states of the underlying physics to the models at the macro level.

1.3.1 The Physics Underlying the Meso Scale

The physical first principles begin with the most primal phenomena: the radiation damage event. As detailed in Section 1.1, these events comprise a highly energetic neutron particle with a high amount of kinetic energy that gets transferred into the lattice of the material, resulting in vacant atom sites and interstitial atoms. Although occurring in an uncorrelated fashion, these increases in vacancy concentrations lead to system states over their equilibrium values. New physical processes then start to govern the states of these systems, where the diffusion and interaction of these point defects lead to microstructural evolution.

Before the macroevolution of these microstructural damages, these diffusive movements of interstitial atoms, along with the varying interactions between the elements in the material system, encompass a meso scale of physics. Chiefly, void formation begins to result from the clustering of these vacancies, and it is driven by these meso sclae physical dynamics.

Dislocations first tend to absorb interstitials at a faster rate than vacancies, initially facilitating the process of void formation. This phenomenon is known as dislocation bias **??**. Nuclear transmutations cause the presence of gas atoms. These gas atoms available in the irradiated material permit the stabilization of vacancy cluster embryos; the formative stages of larger void states. This high rate of absorption of interstitials, by dislocations, effectively removes more interstitials from the system than vacancies. This explicitly causes a clustering of “free" vacancies; later void embryos. All of this process is further fomented by the tendency of interstitials -themselves- to cluster and form what are known as dislocation loops. And, finally, the presence of gas then further encourages the void nucleation process. Even without the presence of gas atoms, this supersaturation of vacancies leads to void formation.

It is easy to understand, under these conditions, how continuum-level phenomena such as volumetric swelling, density decrease, and dimensional instability begin to occur, having significant impact on the yielding behavior of the structure of the irradiated materials.

Nonetheless, it is self-evident that under these physics and conditions, that the macro-level modeling of void nucleation and growth are not capturing the physics of these meso-scale phe-

nomena. Classical nucleation theory, [], and reaction rate theory, [], can both be augmented by statistical and stochastic modeling, that respect the statistical properties of these phenomena while maintaining fidelity to the stochastic nature of the system, as well.

1.3.2 The Missing Meso Models

In the following chapters, we propose and outline multiple avenues of implementing this proposed approach, and analyze their impacts on the current modeling methodologies. We highlight their benefits, discuss their discrepancies as well as their deficiencies, and detail their implementations. The properties of these statistical methodologies and stochastic models are analyzed both numerically and analytically. Where appropriate, numerical simulations uncover space and time properties of the models themselves, as well as the properties of the predictions from the forward physical models. Theoretical properties are derived, under tangible assumptions for simplified versions of our desired stochastic models, and motivate the numerical simulations that are pertinent to the actual, more pragmatic use of these models in engineering materials.

Chapter 2

Stochastic Forcing through Statistical Upscaling

Modeling nuclear radiation damage necessarily implies multiple scales in both time and space, where molecular-level models have drastically different assumptions and phenomena than continuum-level models. In this paper, we propose a novel approach to explicitly coupling these multiple scales of the microstructure damage to engineering scales through a statistical upscaling of models using stochastic processes that take atomic collision cascade simulations from physical first principles of vacancy generation in the micro scale through first and second-order moment approximations to phase field models of void nucleation and growth in irradiated materials, in both space and time. The effect of radiation source on the dynamics of void nucleation and growth is illustrated.

2.1 Introduction

Structural damage in engineering systems resulting from radiation effects on materials is important in a number of applications including energy production, e.g. power plants, and devices operating under extreme conditions, e.g. satellites. The process leading to damage is multiscale in nature and presents a number of modeling and numerical challenges. In this paper, we tackle the critical problem of coupling a microscale model for radiation collision damage to a macroscale model for material void dynamics.

The source of damage originates with atomistic-scale collision cascades initiated by a neutron that collides with an atom occupying a place in a lattice structure, which leads to a cascade of subsequent collisions. The result is a large cohort of atoms that are displaced from their lattice positions as well as interstitial molecules isolated in vacancies, see Fig. 2.1. The physics at this scale is well understood and there are well validated codes that produce high fidelity simulations of individual collision cascades [].

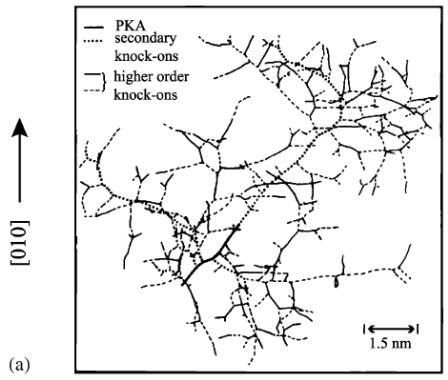

Figure 2.1: Example of a collision cascade.

Interestingly, if the collision cascades are sufficiently large and sufficiently dense in the material, they can combine to form void regions that are much larger than molecular scale and evolve over much longer time scales than individual damage events. Eventually, such voids can affect the structural properties of the material, leading to heat and stress risers in large structures. Void dynamics are believed to result from motion to lower potential energy and a common model is the Cahn-Hilliard partial differential equations. One formulation uses a phase field variation as a way to treat the sharp interface boundaries of voids in a mathematically consistent and numerically accessible fashion. Such Cahn-Hilliard phase field models reproduce void dynamics on an macroscale consistent with energy principles [].

However, this leads to the multiscale modeling problem of combining the atomistic and macroscale models to form a multiscale model of the structural damage that incorporates the initial damage events and the subsequent void dynamics. This is the problem tackled in this manuscript.

A common approach to create such models is to represent the damage as a forcing function for macroscale partial differential equation (PDE) which is evaluated at each point in a numerical discretization of the PDE []. Such models have produced results that appear reasonable when the PDE discretization is fixed. But, there are significant mathematical difficulties with this approach, including the fact that this leads to arbitrarily rough forcing for the PDE and the model changes when the discretization changes. One practical consequence is that common consistency validation experiments cannot be performed.

In this paper, we investigate a statistical upscaling approach. We model both the geometry of the collision cascades and the position and time of cascade events as stochastic and then use statistical techniques to upscale the atomistic scale collisons to create a forcing function representing statistical properties of the stochastic microscale behavior for use in the PDE model. The upscaling involves using long time behavior of stochastic processes to bridge to the long time scale for void migration.

The numerical results show that integral quantities such as the void volume fraction in time as well as the morphology of the concentration fields depends on the stochastic nature of the forcing function in the PDE model. We investigate the impact of the properties and behavior the stochastic effects in defect generation have on void volume fraction, as well as the nucleation and growth dynamics simulated by the phase field approach. We demonstrate that the model behaves consistently as the PDE discretization is changed.

OUTLINE

2.2 Phase Field Cahn-Hilliard Model for Void Dynamics

The PDE model consists of a phase field model for void formation in irradiated solids defined in a type C framework [**?**] that combines both Cahn-Hilliard and Allen-Cahn dynamics [9]. The phase field variation is introduced to handle the sharp interface between voids and the material. The model reproduces several important characteristics of void dynamics under irradiation including nucleation and void growth. The solutions of the phase field models can be related to solutions of the original sharp interface models using formal analytical analysis [**?**, **?**, **?**]. We present a brief overview.

2.2.1 Integral Form of the Free Energy Functional

The material of interest is assumed to be a single crystal metal that has a matrix phase and a void phase. The matrix phase is subject to point defects or vacancies arising from radiation

collisions while the voids form as a result of a "supersaturation" of vacancies in regions consisting of a mixture of vacancies and lattice atoms.

The phase field formulation expresses the total free energy functional of a system in terms of a deconstruction of the constituent phases and interfaces. The phase field formulation involves two descriptive variables. We describe the distribution of vacancies using a vacancy field, denoted $c_v(x,t)$, valued between 0 (void) and 1 (matrix). Since the system is heterogeneous with different phases occupying different regions of the material, we also use a "long range order" parameter, $\eta(x,t)$. This non-conserved phase field variable or order parameter varies smoothly across interfaces and takes on constant values within regions of a particular phase. Employing the Cahn-Hilliard definition of free energy, the total free energy functional is used to derive the kinetic equations for both the conserved and non-conserved phase field variables of the system, assuming they are smooth functions:

$$\Psi[c_v,\eta] = N\int_\Omega [h(\eta)\psi^m(c_v) + \omega(c_v,\eta) + \kappa_v|\bigtriangledown c_v|^2 + \kappa_\eta|\bigtriangledown \eta|^2 + \psi^{el}(c_v,\eta)]\, d\Omega \qquad (2.1)$$

The first two terms on the right correspond to the system bulk energy; the free-energy density of the homogeneous system, the gradient terms describe the interfacial energy contributions, and last term accounts for the interaction of point defects with the stress field, i.e., the elastic effects [9]. The corresponding energy terms are akin to the gradient terms in the spinodal decomposition theory of Cahn and Hilliard [**?**, **?**] and the theory of antiphase boundary of Allen and Cahn [**?**]. The gradient energy terms account for the field inhomogeneity that is present across the diffuse void-matrix interface. When there is no stress on the system, the first two energy terms define two stable wells: one at $c_v = c_v^0$ and $\eta = 0$, corresponding to the matrix phase with vacancy concentration equal to the thermal concentration, and another at $c_v = 1$ and $\eta = 1$ corresponding to the void phase.

2.2.2 Kinetic Equations of the Phase Field Model

The kinetic equations for the space and time evolution of the phase field variables, $c_v(x,t)$ and $\eta(x,t)$, are derived following the standard procedures in the phase field approach:

$$\frac{\partial c_v}{\partial t}(x,t) = \bigtriangledown \cdot M_v \bigtriangledown \frac{1}{N}\frac{\delta \Psi}{\delta c_v} + \xi(x,t) + P_v(x,t); \tag{2.2}$$

$$\frac{\partial \eta}{\partial t}(x,t) = -L\frac{\delta \Psi}{\delta \eta} + \zeta(x,t). \tag{2.3}$$

Equation (2.2) is a form of generalized diffusion equation or modified Cahn-Hilliard equation [**?**]. The last term in (2.2) accounts for the generation of vacancies, or the direct formation of voids, as a result of the atomic collision cascade process in irradiated materials. Thus, $P_v(x,t)$ represents the production of vacancies due to the cascade process. Equation (2.3) is similar to the phenomenological Allen-Cahn equation [**?**]. The final forms of the kinetic equations may be solved by evaluating the variational derivatives of the free energy functional, (2.1), with respect to $c_v(x,t)$ and $\eta(x,t)$ and substituting the results into (2.2) and (2.3). The solution of these final kinetic equations, the partial differential equations, describes the time evolution of the void phase and the vacancy concentration field.

2.3 Statistical Upscaling of the Source

In classic applications, the Cahn-Hilliard model is used to describe the void dynamics in material starting from a mixed solid/void initial state, while the model (2.2) is driven by the source term $P_v(x,t)$ quantifying the introduction of spatially distributed vacancies over time. The result is a multiscale model combining the very fast, molecular scale collision cascades generating atomic vacancies with the complex dynamics of much larger void regions evolving on much longer time scales. In concrete terms, the spatial and temporal scales of a reasonably accurate discretization of (2.2)–(2.3) are much larger than the spatial and temporal scales of individual cascades. In this section, we describe a statistical approach to upscaling the molecular scale events to be included into a driving term for the macroscale model.

2.3.1 Simulating Collision Cascades on the Molecular Scale

There are useful analytical expressions describing the physical mechanisms of radiation/particle interactions based on transport theory valid in the linear cascade regime. However, such results

tend to be complicated and require strong simplifying assumptions; e.g. infinite medium and non-near-surface phenomena. An alternative solution that also covers the linear cascade regime is the so-called binary-collision approximation (BCA), which have become the defacto manner of modeling the radiation damage introduced by individual collision cascades [2–4]. This has yielded long-standing and well-validated manners of simulating individual micro-scale events using simulation software such as SRIM (*Stopping Range of Ions in Matter*) by Ziegler et. al. [5, 6]. We use a variation of SRIM known as TRIM (*Transport of Ions in Matter*) to compute individual collision events.

We use TRIM [5] to generate binary collision cascades for copper at a given energy of the primary knock-on atoms. In particular, we simulate the introduction of a pure copper ion Cu (29), with mass = 062.93 amu, initial energy of either 10Kev or 100KeV, and ion angle-to-surface of 0 degrees, into a pure Cu lattice cube of depth, width and length of 500 angstroms with density of 8.92 g/cm^3. We discuss the choice of 500 angstroms below. In such a lattice, the displacement energy of a Cu molecule is 25 eV, the lattice binding energy of Cu is 3 eV, and the surface binding energy of the Cu is 3.52 eV.

A typical simulation of a collision cascade is presented in Fig. 2.1. Computing the source term involves simulating many, e.g. 100,000, collision cascades. For cascade simulation j, we record the number of atoms displaced from the lattice d_j, and the 3-dimensional range projections (X_j, Y_j, Z_j) measured in angstroms that coarsely quantify the size of the cascade. That is all the specific characteristics of simulated collision cascades used in the statistical upscaling.

2.3.2 A Microscale Spatial Model for Collision Cascades

Recall that a collision cascade results in a molecular scale region in the material consisting of atoms that are displaced from their lattice positions as well as interstitial molecules isolated in vacancies, see Fig. 2.1. The Cahn-Hilliard model is not derived at a scale that represents the dynamics of individual molecules or the geometric complexities of collision cascades. Rather, the descriptive variables are densities of atom types, e.g., vacancy, interstitial, or matrix, over

much larger nominal cells of a material. The first step in the model is to build density functions associated with each collision cascade that quantify the number of displaced and interstitial atoms respectively in the region of the cascade. We use Gaussian Radial Basis Functions to represent the density centered at the location of each collision cascade. Because the cascades are stochastic, this amounts to replacing the geometry of collision cascades by first order spatial moments.

A Model for the Fraction of Molecules Affected by a Collision Cascade

We define a volume for cascade j as,

$$v_j = \frac{4}{3}\pi\, X_j Y_j Z_j \text{ ang}^3,$$

where recall that X_j, Y_j, and Z_j are the range projections of cascade j. To characterize the number of molecules affected by a collision cascade, we define the nondimensional **Cascade Fractional Density** (cfd) for each cascade j:

$$\bar{c}_j = (\text{cfd})_j = \frac{d_j}{v_j \rho},$$

where ρ is the atom density of the material in units of atoms/ang^3. For pure copper,

$$\rho = \frac{4}{a^3} = \frac{4}{3.615^3} \approx \frac{4}{47.24163} \frac{\text{atoms}}{\text{ang}^3}.$$

The quantities d_j, v_j, and $\bar{c}_j$ are the basic building blocks for the model we construct computed from SRIM simulations. Since the collision cascades computed using SRIM are stochastic, the quantities d_j, v_j, and $\bar{c}_j$ are random variables. We plot some histograms reflecting the distributions of these quantities. We show the distribution of d_j in Fig. 2.2, the distribution of v_j in Fig. 2.2, and finally the distribution of the cfd $\bar{c}_j$ in Fig. 2.3.

Representing a Collision Cascade as a Density

We represent the displaced vacancy and interstitial molecules involved in each collision cascade as densities. We employ Gaussian radial basis functions:

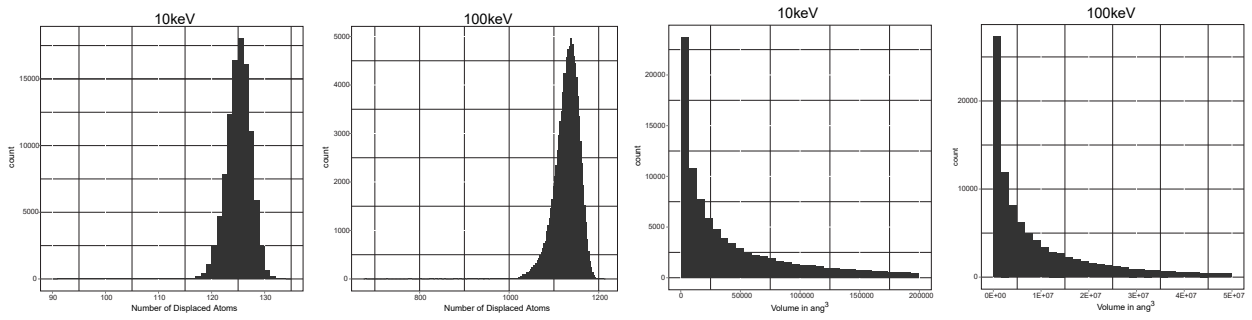

Figure 2.2: Left two plots: Histograms for the numbers of displaced atoms d_j at two energies. Right two plots: Histograms for the effective volumes of collision cascades v_j at two energies

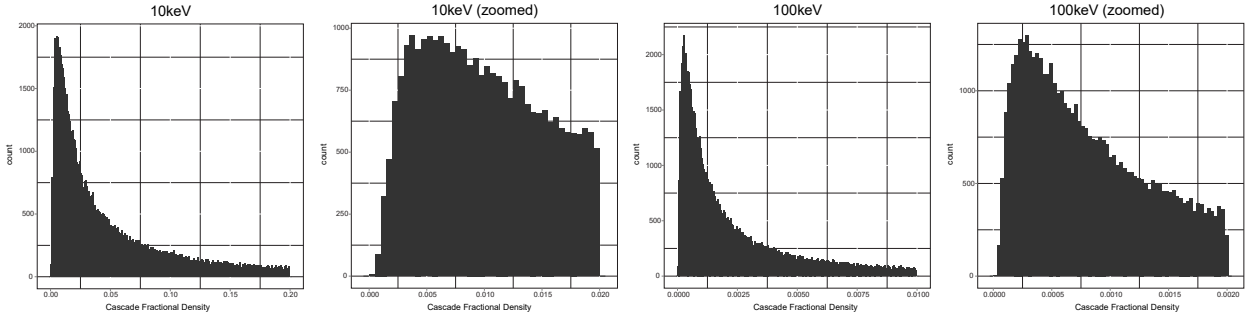

Figure 2.3: Histograms for the Collision Fractional Density $\bar{c}_j$ at two energies, along with plots zoomed in the regions near the origin.

$$p_v^{casc}(x - x_c) = A_v \exp\left(-\frac{(x - x_c)^2}{\sigma_v^2}\right), \tag{2.4}$$

$$p_i^{casc}(x - x_c) = A_i \frac{(x - x_c)^2}{\sigma_i^2} \exp\left(-\frac{(x - x_c)^2}{\sigma_i^2}\right), \tag{2.5}$$

where x_c is the center of the collision cascade. The factors A_v and A_i correspond to the amplitude of concentrations of the displaced vacancy and interstitial atoms respectively, and σ_v and σ_i determine the spread of the density in space. (We use the subscripts v and i to denote *vacancies* and *interstitials*.) The model parameters A_v, A_i, σ_v and σ_i characterize the amount of radiation damage arising from a collision cascade.

Now, since each “event" can be written as the following two expressions (the first corresponding to the vacancies and the latter to the interstitial atoms), for each individual event:

$$\begin{cases} p_v^{casc}(x - x_c) = A_v \exp\left(-\frac{(x-x_c)^2}{\sigma_v^2}\right) \\ p_i^{casc}(x - x_c) = A_i \frac{(x-x_c)^2}{\sigma_i^2} \exp\left(-\frac{(x-x_c)^2}{\sigma_i^2}\right) \end{cases}$$

it follows, then, that we must have the above equations equated when integrated over their respective spaces such that when we specify, $\bar{c_j}$, and variance, σ, of our events by the prescribed physics of the particular material, that we can find our amplitudes, A_i and A_v, for each event since we must have the following:

$$\begin{cases} \bar{c_j} = \int_0^\infty A_v \, e^{-(x-x_c)^2/\sigma_v^2} \, 2\pi r dr \\ \bar{c_j} = \int_0^\infty A_i \, (x - x_c)^2 e^{-(x-x_c)^2/\sigma_i^2} \, 2\pi r dr \end{cases}$$

***note: this is for one dimension, while we can easily integrate over $4\pi r^2 dr$ for two dimensions**

and we can also quickly see then that we have a proportionality in the following:

$$\begin{aligned} \bar{c_j} &= \int_0^\infty A_v \, e^{-(x-x_c)^2/\sigma_v^2} \, 2\pi r dr \\ &= A_v \int_0^\infty e^{-(x-x_c)^2/\sigma_v^2} \, 2\pi r dr \\ &\Rightarrow \bar{c_j} \sim A_v \end{aligned}$$

Using the range projections X_j, Y_j, and Z_j of cascade j, we set $\sigma_{v,j} = \sqrt[3]{X_j Y_j Z_j}$. The interstitial atoms are mainly located on the edges of the collision cascade, so we set $\sigma_{i,j} = \frac{3}{2}\sigma_{v,j}$. To derive the amplitudes of the Gaussian functions, A_v and A_i, we use the following relationship between the $(cfd)_j = \bar{c}_j$ value of each cascade:

$$\bar{c}_j = \frac{\int A_v^j \exp -\frac{|\mathbf{x}-\mathbf{x_c}|^2}{\sigma_{v,j}^2} \, d\mathbf{x}}{\int \, d\mathbf{x}} \quad \text{and} \quad \bar{c}_j = \frac{\int A_i^j \frac{|\mathbf{x}-\mathbf{x_c}|^2}{\sigma_{i,j}^2} \exp -\frac{|\mathbf{x}-\mathbf{x_c}|^2}{\sigma_{i,j}^2} \, d\mathbf{x}}{\int \, d\mathbf{x}}.$$

For the dislocated atoms, we compute the integrals over domains of diameter $3\sigma_{v,j}$ to arrive at the approximation:

$$\bar{c}_j \approx \frac{2\pi A_{v,j} \int_0^{3\sigma_{v,j}} \exp -\frac{r^2}{\sigma_{v,j}^2} r \, dr}{\int_{-3\sigma_{v,j}}^{3\sigma_{v,j}} \int_{-3\sigma_{v,j}}^{3\sigma_{v,j}} dxdy}.$$

$$\frac{d_j}{V_j \cdot N} = \frac{2\pi A_{v,j} \int_0^{3\sigma} \exp -\frac{r^2}{\sigma_{v,j}^2} r dr}{\int_{-3\sigma}^{3\sigma} \int_{-3\sigma}^{3\sigma} dxdy} \tag{2.6}$$

and thus

$$\frac{d_j}{V_j \cdot N} = \frac{2\pi A_{v,j} \int_0^{3\sigma} \exp -\frac{r^2}{\sigma_{v,j}^2} d(r^2)}{36\sigma^2} \tag{2.7}$$

$$\frac{d_j}{V_j \cdot N} = \frac{2\pi A_{v,j} \left(-\frac{1}{2}\sigma^2 \left(\exp \frac{-r^2}{\sigma^2} \right]_0^{3\sigma} \right)}{36\sigma^2} \tag{2.8}$$

We obtain,

$$A_{v,j} \approx \bar{c}_j \frac{36}{(1 - e^{-9})\pi}.$$

We illustrate this density representation of each cascade in Fig. 2.4.

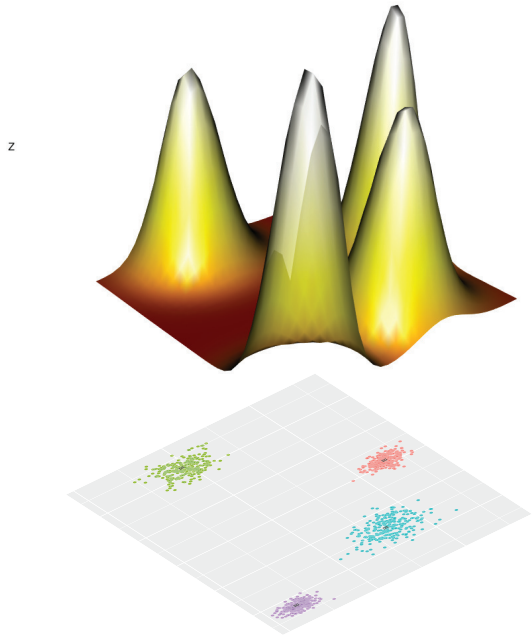

Figure 2.4: Radial Basis function approx. of Collision Cascades.

Statistical Distributions of Gaussian Parameters:

We plot the statistical distributions of each of the parameters from the Vacancy and Interstitial Gaussian structures. This allows us to visualize the statistical fluctuations that we will be introducing into the stochastic Core-Shell model of the source term of the Phase-Field equations.

2.3.3 Distribution of Vacancy Half-Widths

Following the spheroidal approximation of the binary collision cascade, we calculate the Gaussian Half-Widths as in (1).

We plot the distributions of the half-widths, $\sigma_{v,j}$ and $\sigma_{i,j}$, below:

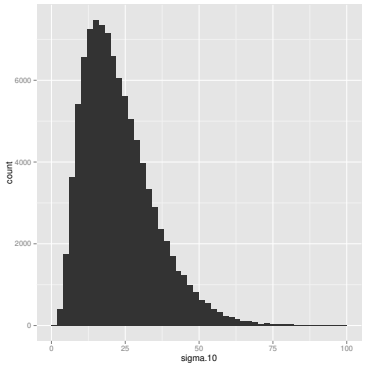

Figure 2.5: 10keV Initial PKA Energy

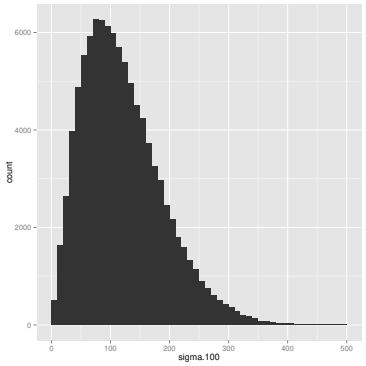

Figure 2.6: 100keV Initial PKA Energy

We omit the half-widths of the interstitials, since they are a linear scaling of $\sigma_{v,j}$.

2.3.4 Distribution of Vacancy Amplitudes:

Following the relationship between the average atomic displacement per cascade, we calculate the Gaussian Amplitudes following the equations (3), (4) and (5).

We plot the distributions of the Amplitudes, $A_{v,j}$, below:

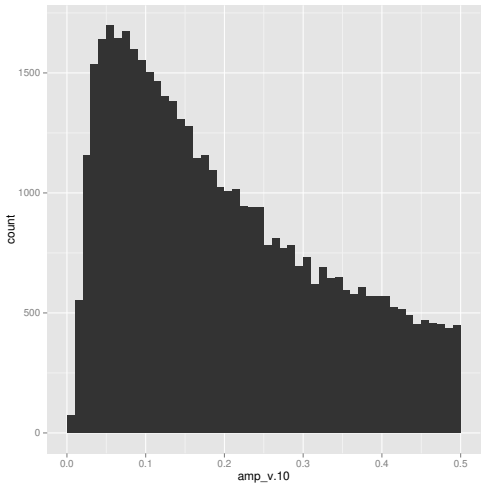

Figure 2.7: 10keV Initial PKA Energy

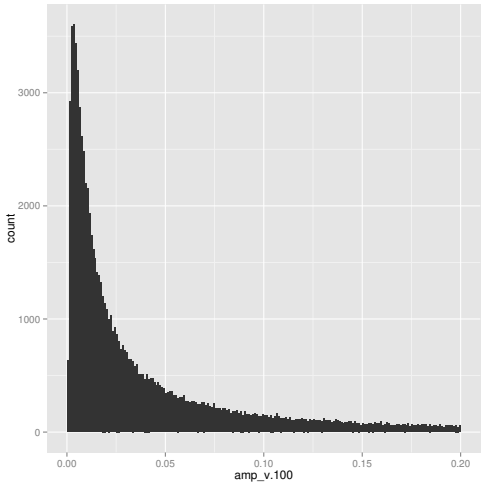

Figure 2.8: 100keV Initial PKA Energy

2.3.5 Damage Introduction Over Space

We incorporate the ongoing production of vacancies and interstitial molecules due to radiation into our concentration fields and models using stochastic source terms $P_v^{casc}(\mathbf{x}, \mathbf{t})$ and $P_i^{casc}(\mathbf{x}, \mathbf{t})$, which randomly introduce localized increases in point defects in our concentration fields which correspond to our spatially-resolved ensembles of vacancies and interstitials resulting from displacement cascades.

We thus define $P_v^{casc}(\mathbf{x}, \mathbf{t})$ and $P_i^{casc}(\mathbf{x}, \mathbf{t})$ as below,

$$P_v^{casc}(\mathbf{x}, \mathbf{t}) = \mathbf{p_v^{casc}}(\mathbf{x} - \mathbf{x_c})\delta(\mathbf{t} - \mathbf{t_c}) \tag{2.9}$$

$$P_i^{casc}(\mathbf{x}, \mathbf{t}) = \mathbf{p_i^{casc}}\xi_{\mathbf{b}}(\mathbf{x} - \mathbf{x_c})\delta(\mathbf{t} - \mathbf{t_c}) \tag{2.10}$$

where x_c is the center of a cascade occurring at time t_c, $p_v^{casc}(x - x_c)$ and $p_i^{casc}(x - x_c)$ are the mathematical functions characterizing the spatial distribution of defects generated due to cascade. Further, ξ_b, in equation 2.10, is a factor introduced to account for production bias in the number of interstitials produced during the cascade damage.

The binary collision cascades assume perfect material for each event so that, at it's most fundamental, we break up the mesoscale, described in detail below, into a grid of cells of sufficiently small size such that we may ensure that this assumption holds and that these point defects occur in a single cell and independently of one another. For the simulations shown in the rest of this paper, we chose to use cell sizes of 500 angstrom^2. This was a material-specific choice, as a binary collision cascade simulation almost surely is contained within this chosen size, for the material Cu with the initial energy that we used. Grid size may be appropriately chosen to satisfy the above assumptions according to the specific material and specific ion/energy simulated.

We then model the displacement collision cascades as discrete point process events and distribute their occurrence uniformly over space, Fig. 2.9. At each cascade occurrence event, we

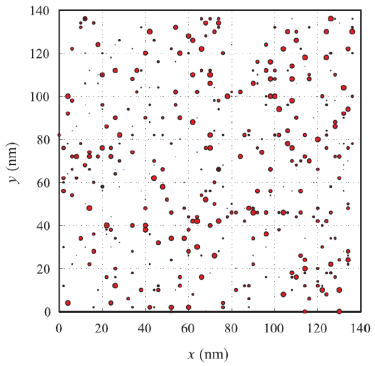

Figure 2.9: Example of random cascades in a unit area over period T.

uniformly draw spatial coordinates to distribute the cascade event. We also assume that no cascade can occur within an already-damaged cell, and thus do not distribute over already damaged cells.

Spatial Model: Construction of Mesoscale Grid Independent of PDE Discretization

The first step is to define a computational domain at the mesoscale on which to distribute the stochastic cascade occurrences. It is important to emphasize that this grid is explicitly defined firstly and independently of the PDE discretization. This mesoscale allows us to aggregate the molecular-level cascade events into a statistically upscaled representation, density, that the engineering-scale PDE can understand. We aggregate the independent cascade occurrences over both space and time, explained below. This mesoscale grid has units expressly larger than the individual cascade cells defined in Sec. 2.3.5. Fig. 2.10 shows both individual molecular cascades distributed across this mesoscale grid, as well as an aggregated representation to be introduced as the forcing terms to the concentration fields, Equations 2.9 and 2.10.

Spatial Model: Distribution of Cascades over Mesoscale Grid

We aggregate individual cascade events over the four adjacent quadrants for each node in the mesoscale grid. Fig. 2.11 shows the corresponding quadrants for each node. Thus, if the mesoscale computational domain is comprised of n^2 nodes, we distribute the individual collision cascades over the $4n^2$ mesoscale quadrants.

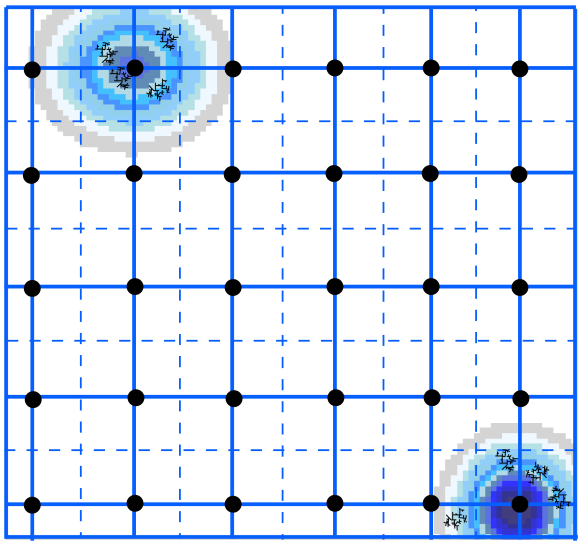

Figure 2.10: Mesoscale Computational Domain: solid blue lines define the mesoscale.

To distribute the collision cascades over the mesoscale, a pair of random variables are drawn at each cascade instance, $\{x_c\}$ and $\{y_c\}$, to determine the cascade center, specifying the spatial location of each cascade in the computational space domain. In order to mimic irradiation conditions which are uniform in space, we choose the random variables $\{x_c\}$ and $\{y_c\}$ to follow a uniform distribution over the spatial domain.

$$\{x_c\}, \{y_c\} \sim \mathcal{U}\{0, 2n\} \tag{2.11}$$

where the Cumulative Distribution Function for the random variable $\mathcal{U}\{a, b\}$ is defined as $\forall$ integer $k \in [a, b]$

$$F(k; a, b) = \frac{k-a+1}{b-a+1}$$

Here, n represents the number of nodes in the width of our mesoscale grid.

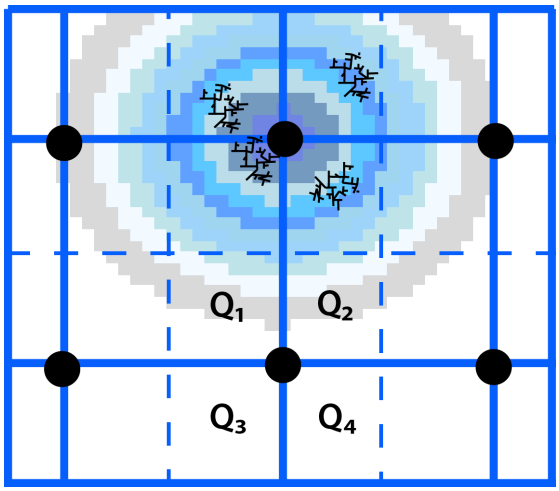

Figure 2.11: Dashed blue lines define the aggregation boundaries at the mesoscale.

Spatial Model: Aggregation of Cascades over Mesoscale Grid

After all the cascades have been introduced to the concentration fields, for each PDE timestep, a second aggregation occurs. At each meso node, we aggregate the cascades over its corresponding four quadrants shown in Fig. 2.11. We do this by equating the total damage of the collision cascades to the total damage of a larger Gaussian representation, Fig. 2.12.

This aggregation is achieved using Eq. 2.4 and Eq. 2.5 through the following relationships:

$$\int p_{v,j}^{nodeCasc}(x - x_c) = \sum_i \int p_{v,i}^{casc}(x - x_c) \tag{2.12}$$

and

$$\sigma_{v,j}^2 = \sum_i \sigma_{v,i}^2 \tag{2.13}$$

Spatial Model: PDE Discretization

For the PDE solver, we constrain the discretization to be strictly finer than the mesoscale grid defined in Sec. 2.3.5. This permits us to carry out experiments to verify common consistency in the solver. As we refine the discretization, we find that we achieve the same numerical solution, in expectation.

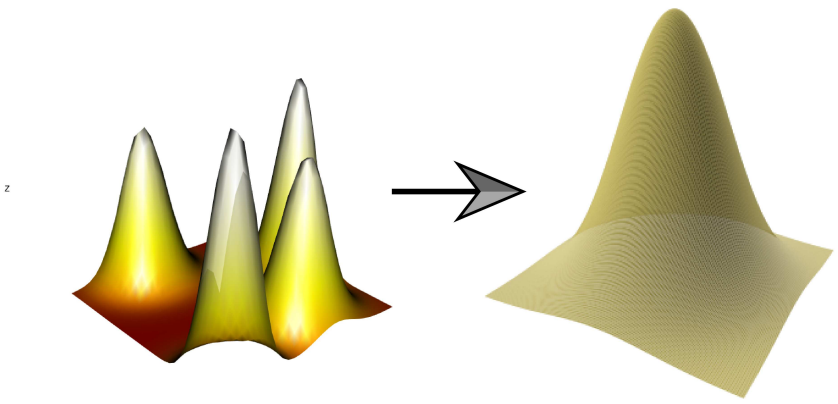

Figure 2.12: Aggregating cascades at each mesoscale node.

2.3.6 Damage Introduction Over Time on Microscale

We define a "waiting time" to be the time between an arrival event of a collision cascade occurrence and the immediate subsequent cascade arrival.

The first consideration to note is that irradiation is generated outside of the material domain that we would like to model. We assume that each introduction of irradiation damage to our domain is independent of one another. Therefore, the arrival time of each cascade event is independent of one another. A second chief consideration is to note that if t units of time has elapsed since the last cascade event, that the probability of waiting s more units of time until the subsequent event is the same as if we haven't waited at all. We assume this because each cascade arrival is independent of one another, and thus a cascade "doesn't know" about when the previous cascade arrived. In the probability literature, this is known as the **memory-less property**.

Since we assume that each cascade arrival time is independent, continuous, and memory-less, we have a natural model to impose on the waiting times: exponential. The exponential distribution is fully described by a single parameter, which is the mean of the distribution. We thusly set

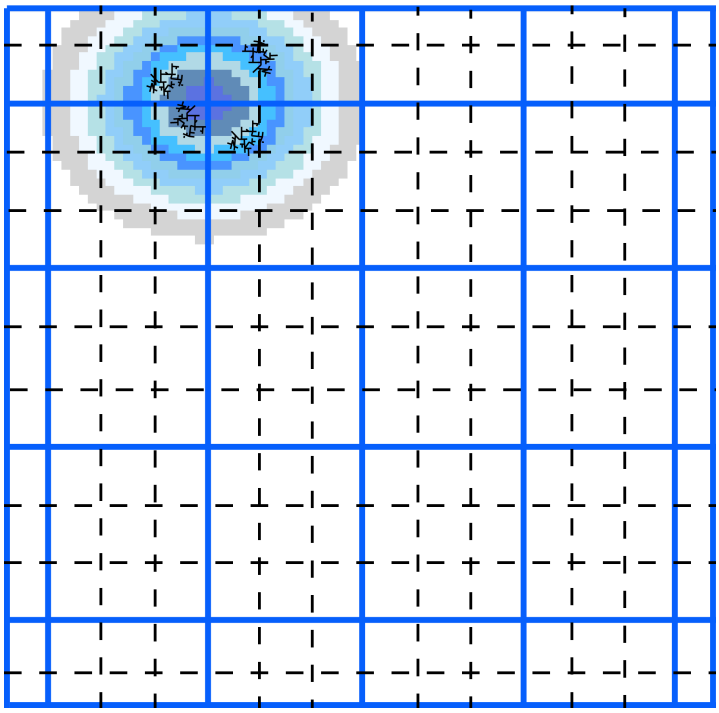

Figure 2.13: Dashed black lines define the PDE discretization, strictly smaller than the solid blue mesoscale.

the inverse of the prescribed material's average rate of occurrence as the parameter. We can then generate a new collision cascade at each time step, where the length of each time step is random and is distributed according to this prescribed exponential distribution.

A random variable T is said to have **an exponential distribution with rate** λ, or T = exponential(λ), if

$$P(T \leq t) = 1 - e^{-\lambda t} \quad \forall \ t \geq 0$$

Here, we have described the distribution by giving the **distribution function**, $F(t) = P(T \leq t)$, but we can also write the definition in terms of its **density function**, $f_T(t)$, which is the derivative of the distribution function

$$f_T(t) = \begin{cases} \lambda e^{-\lambda t} & \forall \ t \geq 0 \\ 0 & \forall \ t < 0 \end{cases}$$

Here, we specify the **arrival rate**, λ, as $\lambda = \bar{R_c}^{-1}$, $\forall\ i = 1, 2, ...$ where $\bar{R_c}$ is the average rate of collision occurrence prescribed by the physics of the material. Given the exponential model for waiting times, we can use it to simulate the process by generating new collisions at arrival times drawn from the exponential distribution.

We define the following in our model:

$$t_i = \text{the time of occurrence of the } i^{th} \text{ cascade}$$

and thus have that

$$t_{i+1} - t_i \sim exp(\lambda) \tag{2.14}$$

where $\lambda = \bar{R_c}^{-1}$, $\forall\ i = 1, 2, ...$ and $\bar{R_c}$ is the average rate of collision occurrence prescribed by the physics of the material.

A property of this model for arrival times is that, among all continuous distributions supported in $[0, \infty]$ that have a given mean of $1/\lambda$, the exponential distribution with mean $1/\lambda$ is the maximum entropy distribution. i.e. it maximizes the following:

$$H(X) = -\int_{-\infty}^{\infty} p(x) \log p(x) dx$$

over all $p(x)$.

This follows in accordance with the principle of maximum entropy, which states that if nothing is known about a distribution other than the fact that it belongs to a certain class, then the distribution with the largest entropy should be chosen as the default. The motivation for this principle is twofold: first, maximizing entropy minimizes the amount of prior information built into the distribution; second, many physical systems tend to move towards maximal entropy configurations over time.

Using this construction, we can take draws from our generated distribution of cascade vacancies, d_j, and generate the following marked point process:

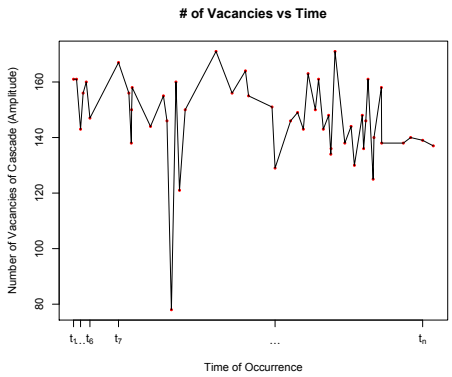

Figure 2.14: Time Series of Collision Cascade Occurrences

Cascade Vacancies, d_j, are iid, as well as the waiting times, $(t_i - t_{i-1})$.

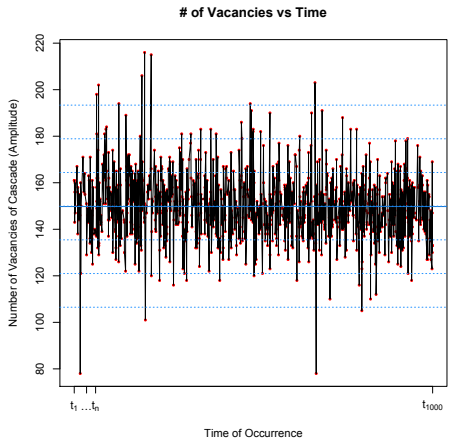

Figure 2.15: 1000 cascade occurrences

Here, we can see now a more dense picture of the process along with some descriptive statistics of the random variable d_j: its mean and 3 standard deviations in either direction.

2.3.7 Simulating Arrival Times over the PDE Timestep:

Since we impose exponential wait times between individual, independent cascade occurrences, we can exploit the fact that the number of cascade events over a larger time scale follows a Poisson Process. This is mathematically defined in the following:

Definition: Let $t_1, t_2, ...$ of independent exponential(λ) random variables. Let $T_n = t_1 + \cdots + t_n$ for $n \geq 1$, $T_0 = 0$, and define $N(s) = max\{n : T_n \leq s\}$.

Then $N(s)$ has a **Poisson distribution** with mean λs. Recall that X has a Poisson distribution with mean μ, or $X \sim Poisson(\mu)$, if

$$P(X = n) = e^{-\mu}\frac{\mu^n}{n!} \quad \forall \ n = 0, 1, 2, ...$$

This is the intuitive explanation for deeming the term "rate λ", the average number of arrivals in an amount of time s is λs, or λ per unit time.

We thusly have the following in the model, that if $N(T) =$ the number of occurrences until time T, then

$$N(t + \tau) - N(t) \sim Poisson(\tau\lambda), \quad \tau \geq 0 \tag{2.15}$$

where $N(t + \tau) - N(t)$ is independent of $N(r)$ for $0 \leq r \leq \tau$, and λ is defined in Equation 2.14.

This enables us to take advantage of the fact that the evolution of voids, whether through diffusion or reaction processes, takes place at a much slower speed than the occurrence of individual cascades, which happen within fractions of nanoseconds. We do this through aggregation: drawing multiple cascade parameters from their distributions at each of the larger PDE solver time steps, aggregating the damage in space from the micro to meso spatial scales. The Poisson Process ensures that the number of events within an interval is proportional to the length of the interval. The parameter of this Poisson Process is also directly determined by the parameter of the exponential arrival times.

2.3.8 Cumulative of Cascade Introductions over Time:

We can explore the behavior over a long run by looking at plots of the cumulative damage as a function of time as a result of random cascades from the above Figure 2.15.

We can easily see, here, that a simple exponential waiting time captures the space and time average constraints rather quickly. While we might expect our averages and statistical properties of our process to hold at infinity, we can see that we achieve our averages relatively quickly, as illustrated in Figure 2.16, below:

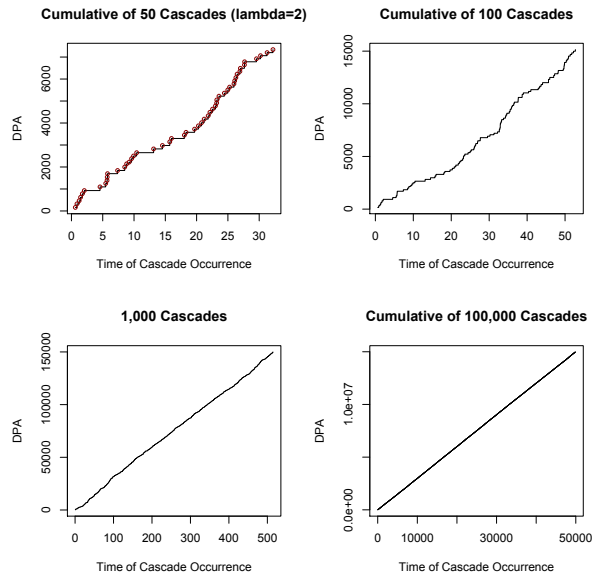

Figure 2.16: Cumulative of the source term corresponding to the Point Process displayed in Figure 2.15.

In fact, when using exponential waiting times, we found that by simply adjusting the parameter (the inverse of the prescribed average cascade occurrence rate), we do, indeed, see that this cumulative function has the same behavior, solely at a slightly different scale, yet within the same relative time frames: ie. we see the Cumulative function "stabilize" after just a few thousand cascade occurrences, regardless of the original parameter.

2.3.9 Practical Implementation of Stochastic Source

The following values are prescribed by the physics of the material:

- the cascade occurrence rate

$$\langle R_c \rangle = \bar{R}_c = \frac{1}{T\Omega} \int_T \int_\Omega \sum_k \delta(t - t_k)\delta(x - x_k) dt d\Omega = \frac{N}{T\Omega}; \;\; t_k \in [0, T], x_k \in \Omega$$

- the density average (i.e. the mean value of the "mark"/"strength"/"magnitude" of the cascades) over a reasonably long time period, T

$$\begin{cases} \langle P_v \rangle = \bar{P}_v = \frac{1}{T\Omega} \int_T \int_\Omega \sum_k A_{v,k}\, e^{-(x-x_k)^2/\sigma_{v,k}^2}\, \delta(t - t_k) dt d\Omega \\ \langle P_i \rangle = \bar{P}_i = \frac{1}{T\Omega} \int_T \int_\Omega \sum_k A_{i,k}\, (x - x_k)^2 e^{-(x-x_k)^2/\sigma_{i,k}^2}\, \delta(t - t_k) dt d\Omega \end{cases}$$

however....

we need to keep in mind that the data will get put into the C-H eqtns through an operator-splitting approach:

$$\begin{aligned} \frac{\partial c_\alpha}{\partial t} &= \frac{\partial c_\alpha}{\partial t}|_{diffreac} + \frac{\partial c_\alpha}{\partial t}|_{source} \\ \frac{\partial c_\alpha}{\partial t}|_{diffreac} &= \nabla M_\alpha \nabla \mu_\alpha(c_\alpha, c_\beta) - R_\alpha(c_\alpha, c_\beta), \\ \frac{\partial c_\alpha}{\partial t}|_{source} &= P_\alpha \end{aligned}$$

and, thus, the source term (the pieces of data that will actually be placed into the Cahn-Hilliard equations) are only just the sums of the delta functions:

$$\begin{aligned} P_v(r,t) &= \sum_k A_{v,k}\, e^{-(x-x_k)^2/\sigma_{v,k}^2}\, \delta(t - t_k) \\ P_i(r,t) &= \sum_k A_{i,k}\, (x - x_k)^2 e^{-(x-x_k)^2/\sigma_{i,k}^2}\, \delta(t - t_k) \end{aligned}$$

Now, since the averages of these terms are prescribed by the physics, we only need to generate a series of data which give us these sums, and which hold the desired properties.

Following the outline presented, we have a complete algorithm to simulate the introduction of atomic-level cascades through the PDE-scale source term stochastically, as given below:

Algorithm 1 Stochastic Cascade Generation as Defect Production Source

for each time step of the PDE solve **do**
  Draw $N(T) \sim$ Poisson($T\lambda$)
  Let mesoNodes[$1 \ldots N(T)$] be a new array

  for each i in N(T): **do**
    Draw int INDEX
    Set $\sigma_v = \sigma_v[INDEX]$
    Set $\sigma_i = \frac{3}{2}\sigma_v$
    Let $A_v =$ integrate (2.4)
    Let $A_i =$ integrate (2.5)
    Draw x_c^i from discrete unif(0,2n)
    Draw y_c^i from discrete unif(0,2n)
    Introduce averaged representation of void damage
    mesoNodes[i] = $\left(\left\lfloor \frac{x_c^i}{2} \right\rfloor , \left\lfloor \frac{y_c^i}{2} \right\rfloor \right)$
  end for

  for each (x,y) in mesoNodes: **do**
    Perform spatial aggregation of Sec. 2.3.5
  end for

  Return Concentration Fields to PDE solver
end for

The event of occurrence of a cascade is defined using cascade probability per unit volume per unit time, p_{vol}^{casc}, as an input parameter corresponding to the damage rate of the simulation. For a given probability, p_{vol}^{casc}, the average rate of production, λ in Equation ((2.14)), is thus completely specified. By determining the length of each PDE time-step (the numerical solver), τ from Equation (2.15), we have also thus completely specified the average rate of production over each PDE time-step.

By the properties of the Poisson Process, outlined above, at each step of the numerical solver, we thus take a random draw from the probability distribution Poisson($\tau\lambda$) and that draw represents the number of cascade events which randomly occurred over that time step. For each cascade in that PDE time step, then, we draw a random cascade size and structure given by the statistical distributions outlined, and introduce these fluctuations into the mesoscale concentration fields;

both vacancy and interstitial. Before returning the concentration fields to the PDE solver, we implement the aggregation from the microscale to the mesoscale in space. Thus, we completely simulate the microscale physics and explicitly couple them to the meso scale in both space and time with statistical fidelity.

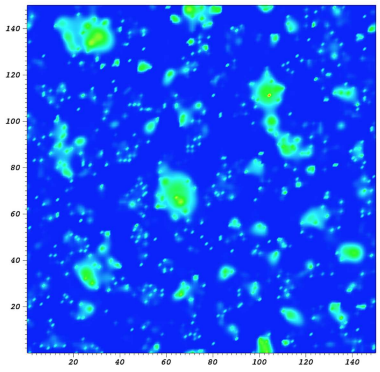

Figure 2.17: Vacancy Field in a unit area over period T.

To ensure the cascade term is applied only within the matrix volume, we exclude the likelihood of any cascade event happening within the non-matrix regions (i.e. voids).

2.4 Numerical Simulations

While in the classical models the source of void nucleation in irradiated alloys is attributed to thermal fluctuations, we investigate the dynamics of nucleation under conditions where source fluctuations dominate. It has been shown that, to a large extent, these conditions resemble the incubation, nucleation and growth behavior observed in the classical precipitation problem [9].

We thus have that $P_v^{casc}(\mathbf{x}, \mathbf{t})$ and $P_i^{casc}(\mathbf{x}, \mathbf{t})$, Equations (2.9) and (2.10), introduce the stochastic vacancy and interstitial source into the modeled concentration fields. These newly entering vacancy sources immediately affect the vacancy diffusion and nucleation of new voids governed by the phase field models. Here, we even have that the source may be assigned to a spatial region larger than the mesh size used. Convergence simulations for mesh size are also carried out and results shown.

2.4.1 Integral Quantities over Time

The description of cascade-induced fluctuations, outlined in the previous section, helps us to completely define the size, structure and rate of cascade based on A_v, A_i, σ_v, σ_i and p_{vol}^{casc} respectively. These quantities are determined by the physical material properties as well as the source production context. We can now also obtain the total magnitude of fluctuations induced by the cascade terms as,

$$< c_v >_{total}^{casc} = \sum_{x_c,t_c} A_v \exp\left(-\frac{(x-x_c)^2}{\sigma_v^2}\right) \tag{2.16}$$

$$< c_i >_{total}^{casc} = \sum_{x_c,t_c} A_i \xi_b \frac{(x-x_c)^2}{\sigma_i^2} \exp\left(-\frac{(x-x_c)^2}{\sigma_i^2}\right) \tag{2.17}$$

In actual practice, we compute this quantity directly from the numerical routines used to implement the cascade source so as to avoid any numerical interpolation errors.

The defect production rate is obtained as,

$$G_d = p_{vol}^{casc} \times \times \bar{p}_v^{casc}(x - x_c) \tag{2.18}$$

where $\bar{p}_v^{casc}$ is the Gaussian function defined in ((2.4)) using the means of the statistical distributions of A_v and σ_v^2.

This quantity, G_d, measures the rate at which atoms are being displaced to create point defects in the form of vacancies and interstitials. As such, G_d is reminiscent of dose rate (or dpa rate) used to measure the amount of radiation undergone by the material per unit volume per unit time.

Throughout time, there are two main integral quantities measured in each of the simulations to observe time behaviors of the model: the void volume fraction and void volume average across the domain. Using the terms defined above, in (2.16):

$$\pi_\Omega = \frac{< c_v >_{total}^{casc}}{\Omega} \tag{2.19}$$

ht

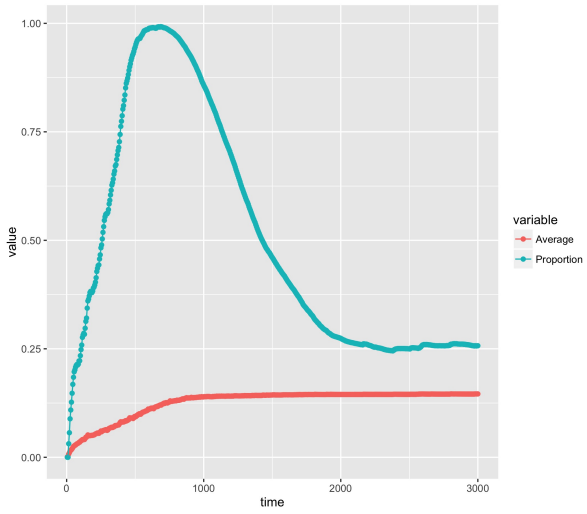

Figure 2.18: Integral Quantities against Time

and

$$\Pi_\Omega = \frac{\mathcal{N}_\ell}{\Omega} \tag{2.20}$$

where $\mathcal{N}_\ell$ is the number of damaged lattice sites on the PDE domain mesh.

Figure (2.18) illustrates the evolution of these integral quantities over time, highlighting the varying stages of void nucleation from incubation through nucleation, growth and stability.

We also highlight the stochastic nature of the forcing on the concentration fields over time, as seen in the evolution of the integral quantities in Figure (2.19). Although the Phase Field Models act as an averaging smoother, the stochastic fluctuations have an evident effect on forcing the PDEs.

2.4.2 Void Dynamics Under Stochastic Vacancy Generation

We run simulations under various parameter regimes for our stochastic forcing to illustrate the effects of the forcing function in the phase field equations on the behavior of the morphology, nucleation, and growth of void volume fractions (i.e. voids). We first illustrate the case in which

ht

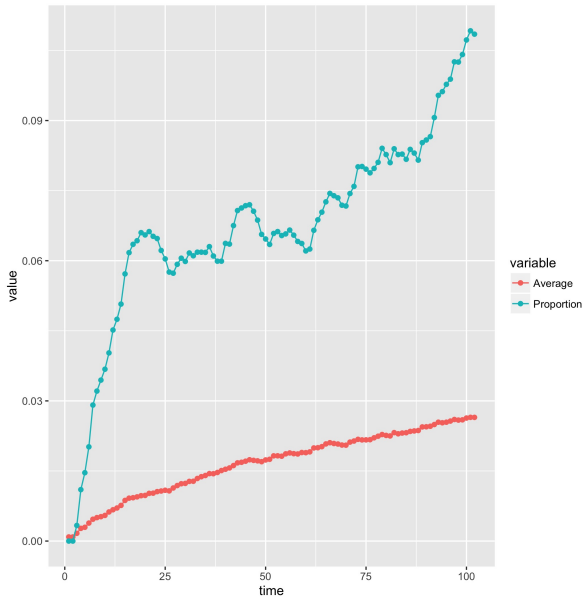

Figure 2.19: Stochastic Short-Term Variations Over Time

the domain of solution has the phase field, $\eta(\tilde{x}, 0)$, set equal to zero everywhere initially and the simulation is started with a supersaturation of vacancy production by irradiation.

As was previously observed in [9], while the average value of the vacancy defect source increases the background vacancy concentration, the highly random and varying nature of the source assists the nucleation process. After an initial incubation period, we observe small void nuclei beginning to appear in the system, growing by absorbing the continual entrance of new defects into the system.

The effect of the radiation source on the dynamics of the void nucleation and growth is illustrated in the following figures. In this example, we illustrate the effect of a larger given vacancy generation rate, λ, in (2.14), on the integral quantities of the void behavior and dynamics.

We also illustrate the effects of greater initial energies of our Primary Knockon Atoms (PKAs) in our micro-scale simulations of the binary collision cascades.

ht

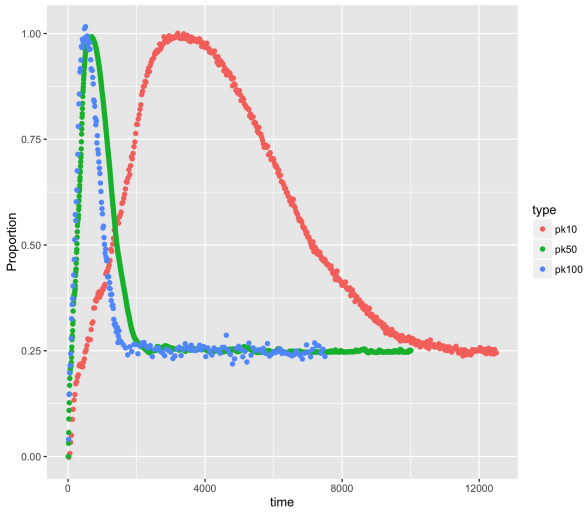

Figure 2.20: Varying PKA Initial Energies

2.5 Discussion of Results

As was hypothesized, the properties of the forcing term, 2.9, in our Phase Field Model, 2.2, had a large effect on the materials properties being modeled. Under our proposed modeling scheme for the forcing term of the differential equation, we are able to explicitly specify the physical properties and behavior of our desired material and radiation conditions. Small changes in the parameters of the proposed stochastic process, affected the void concentration field behavior, as well as the void nucleation stages and processes.

Our method allows us to specify the model from first principles at the micro, atomic scale from the original physical source of defect and damage processes driven by irradiation in the form of atomic collision cascades. We find that control over the initial energies of Primary Knockon Atoms (PKAs) affects the subsequent behaviors of void nucleation and growth in supersaturated material.

We also find that our manner of distributing our second moment approximations of the damage cascade events over the PDE domain lattice permits us to spatially resolve the coupling between our atomic and meso scales in a manner that maintains consistency with the PDE solver. We can refine the mesh of the solver domain and maintain consistent results of our solution.

2.6 Concluding Remarks

Our approach is a novel deconstruction of the additive forcing term of an adapted Cahn-Hilliard equation for Phase Field Models. Our deconstruction permits an explicit coupling to first principal physical processes, at the micro scale, through a statistical upscaling. We propose coupling well-founded simulations of binary collision cascades to represent the atomic displacement cascades in materials under irradiation. We then take second-order moment approximations of the damage events to create a spatially resolved coupling to a domain mesh through Gaussian radial basis functions that permit consistent solutions. Through a stochastic process, we resolve the time scales from the micro level to the PDE engineering scales that maintains statistical fidelity throughout the physical processes being modeled.

Our method and simulations bring a statistical and mathematical rigor to multi-scale modeling unseen in the current literature with far reaching implications beyond irradiated materials.

Chapter 3

Stochastic Mesoscale – *Probabilistic Cellular Automata (PCA)*

Currently, at the meso scale, phase field models, such as Cahn-Hilliard equations, have been proposed to model the more advanced behavior and evolution of microstructure damages [8, 9]. Typical to this approach is the diffuse description of interfacial free energy. These phase field models, however, do not take collision cascades as a vacancy generation scheme, and do not model varying time nor spatial scales. While these models can be used to model both the micro and meso individually, complex issues arise, and stringent assumptions must be made in this approach, and the attempt to couple them together.

Our modeling approach, on the other hand, is explicitly based on coupling a mesoscale stochastic process to the microscale collision models. Our model utilizes these long-standing binary collision cascades as a source of vacancy generation and appropriately couples the micro and meso scales together into one stochastic process, whose properties hold over both small and large time and space scales.

Since we have a multi scale problem where we have very fast individual events (collision cascades) generating atomic vacancies, and complex evolutionary behavior of accumulated vacancies which occur at a much slower scale, the task can seem onerous. This is one motivation for a simple model that captures the desired physical properties of the observed system. The time scale of void movements is much larger than the time of damage events, and is a chief consideration in our model. To address this, we explicitly model both time scales. While maintaining parsimony in the simplicity, we explicitly create a model for both the micro scale events over both space and time, along with a separate -yet intimately bound- model for the meso scale, over both space and time.

3.1 Description and Development

We now model the dynamical behavior that results from various fundamental physical processes of vacancy generation through collision cascades (considered point defects). These point defects undergo mutual recombination, annihilation, or segregate into clusters and thus form voids and dislocation loops. These voids then go through disparate phases, from the “precipitation" phase, the processes of void formation and growth, through to advanced evolution stages such as migrations and Ostwald ripening, in which smaller voids dissolve by releasing vacancies that are subsequently absorbed by larger voids.

Due to this complexity at the meso scale, we propose another model at this “higher" layer, which not only couples directly to the micro scale space/time model, but also exceptionally models the aforementioned behaviors and properties of the physical processes at this meso scale. To appropriately model the natural phenomena, we need to create a model over a large region into which will occur many damage events. Damage migration is an imminent concern and, as such, the meso scale model needs to incorporate a large enough area for damage events to accumulate and be appropriately modeled by a PDE.

We take the desired window size and partition it into a uniform grid of smaller cells of size 500×500 angstroms, which is the cell size in which we simulate the individual cascades. We then randomly choose a cell in the window using a uniform distribution for a damage cascade to occur at each time t_i, as depicted below (Figure 7):

3.1.1 Definitions:

To model the evolutionary behavior of these clusters of vacancies (voids), we propose a stochastic cellular automata model. This model provides a rich class of stochastic behaviors/properties with -few- parameters which can be determined by the physics of the given material. We find that the model allows for random behavior that satisfies our *a priori* model criteria.

We define the cellular automata rules by the principal characteristic that, at each iteration of the Markov Chain, the probability that any individual cell will have vacancy at the next step is

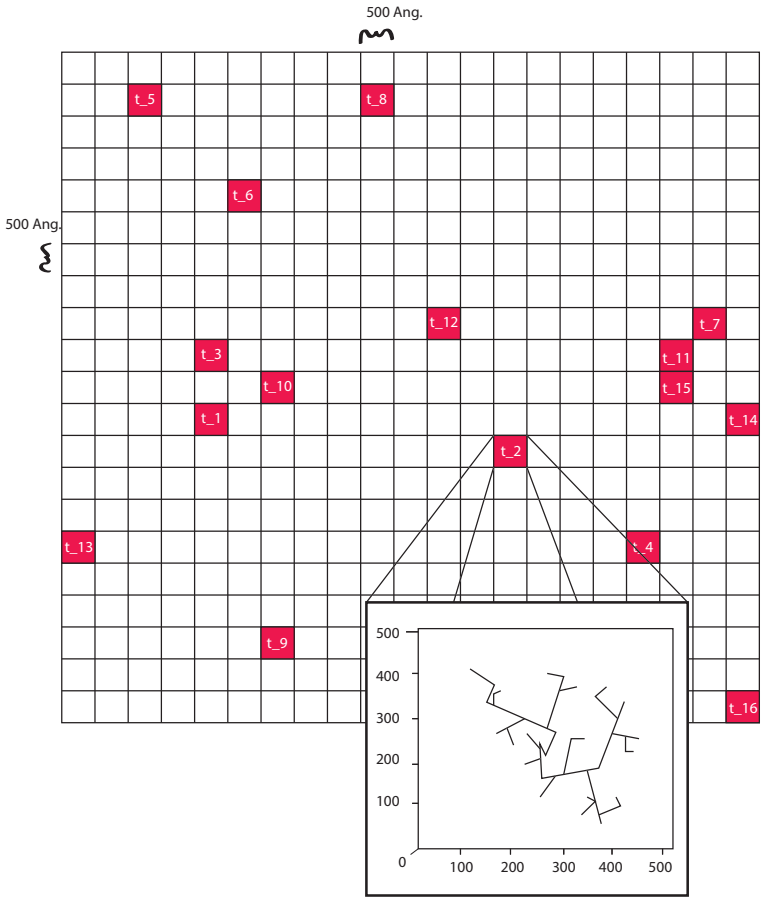

Figure 3.1: The Spatial Grid modeling the positions of the cascades from the Point Process and the Time Series.

dependent on each individual cell's immediate neighbors. Our approach is to allow certain cells to interact, namely cells which or damaged or have an adjacent cell which is damaged.

This approach is a variation of the well-studied Game of Life, by John Conway [10].

Definition 1. *A **cluster** is a group of adjacent cells which contain damage.*

Definition 2. *As shown above, an **extendedCluster** is a **cluster** along with all immediately adjacent non-damaged cells*

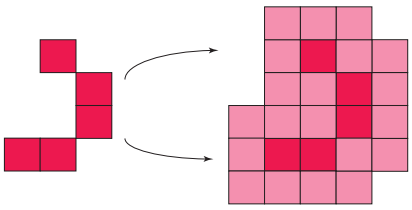

Figure 3.2: An example of a cluster and its corresponding extendedCluster.

In general, at each step of the Markov Chain, we draw a Bernoulli random variable (“flip a coin") for each cell in the extendedClusters to decide whether an individual cell receives any of the cluster’s total vacancy in the next iteration. The probability of success, for the Bernoulli random variable, is defined below. Here, a success would mean that damage vacancy from the cluster is distributed to the cell in the next step. Thus, a cluster’s total vacancy can migrate from the cluster at time T to include the cells in the extendedCluster at time $T + 1$. This allows for movement of the entire cluster as well as interesting behaviors.

3.2 Cluster Identification

A primary requisite of the model is identification of a local region of cells around a damaged cell. The goal is to determine cells which may interact. We define the following:

Definition 3. *A **cluster** is a group of adjacent cells which contain damage.*

Definition 4. *A **miniCluster** is the group of cells composed of a damaged cell (i, j) along with the cell to its immediate right, and the three adjacent cells in the row below.*

i.e. cells $(i, j), (i, j + 1), (i + 1, j - 1), (i + 1, j), (i + 1, j + 1)$ form a miniCluster if cell (i, j) is damaged (Figure 8)

The task is to take the grid of cells and to efficiently identify the clusters of cells, as defined above, which have vacancies. To accomplish this, we first identify the miniCluster for every cell in the grid with damage and create a graph structure out the miniClusters. With this graph structure,

we can identify every cluster using the very well-known and efficient Dijkstra's algorithm to find all the connected sub-graphs. With the Clusters identified, we can then subsequently model the stochastic behavior schemes for each of the vacancy clusters.

3.2.1 miniClusters

The first task to identify a cluster is to sweep through the entire grid of cells and create what we will define as a miniCluster. There is one miniCluster for every cell in our grid which has vacancy. Each miniCluster of a cell with vacancy is composed of the adjacent cells to the immediate right and below, as depicted in Figure 8 below.

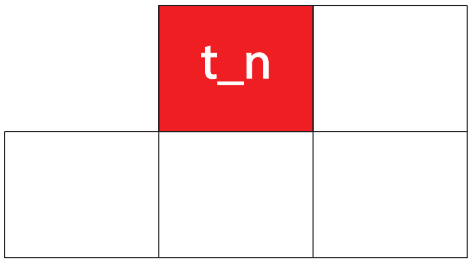

Figure 3.3: A miniCluster extends a vacancy cell to include adjacent neighbors to the right and below.

To illustrate how this efficiently helps with identification of the vacancy clusters in the grid, we study some of the cells in the grid depicted in Figure 7.

If we zoom in on the cluster located in the right-hand side of Figure 7, we find the following cluster:

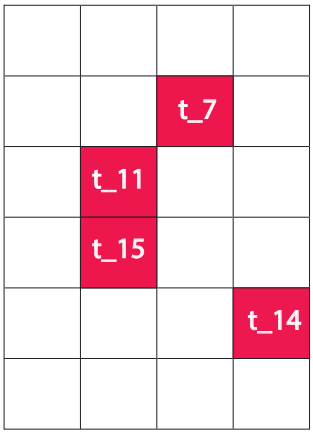

Figure 3.4: Close-up of Cluster from Figure 7.

Thus, we have four miniClusters from these four vacancy cells, and three of them are overlapping, as shown below:

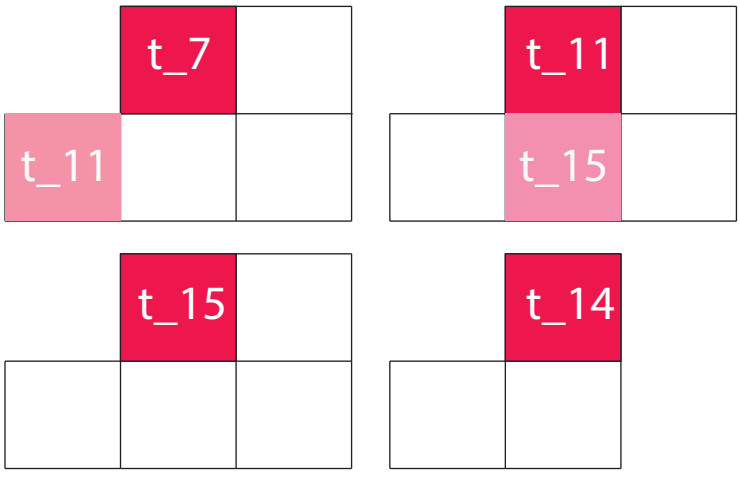

Figure 3.5: miniClusters created from the grid in Figure 9

3.2.2 Create a Graph of our miniClusters

After we have created a list of every miniCluster in the grid, we build a graph out of every one by the following algorithm. Using each vacancy cell as nodes, we make an edge between each pair of nodes if there exists a miniCluster in which both nodes are elements. Since there is at most 4 edges for each damaged cell, we can build the entire graph in linear time, $\mathcal{O}(n)$, where n is the number of vacancy cells in the grid, as depicted below:

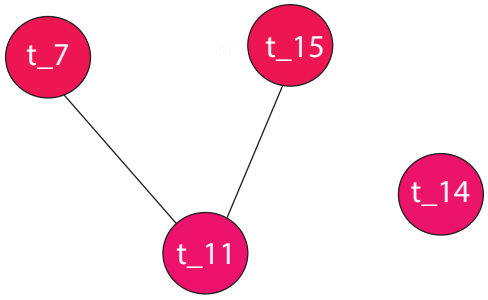

Figure 3.6: graph created from miniClusters in Figure 10

As explained above, it is clear in this example that we have a node for every vacancy cell and an edge between each pair of nodes which share a miniCluster.

A single graph -not necessarily entirely connected- is created from all of the damaged cells in our grid in this manner.

3.2.3 Traversing the Graph using Dijkstra's algorithm

Subequently, we need to traverse the graph to find all of the connected sub-graphs, which form the clusters. This problem, as presented above, is generally an NP-complete problem [**?**, **?**]. However, we can use Dijkstra's algorithm [**?**] and take further advantage of several fortunate features of this particular problem.

Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms [**?**].

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities [**?**].

We take advantage of two properties of this particular graph that immensely help with the theoretic runtime of computation:

1. Each edge can represent an equal weight (i.e. 1).

2. The entire graph is inherently sparse [**?**, **?**].

We iteratively find the shortest path and every node visited, removing each visited node, along with the starting node, from the candidate graph. The general runtime complexity of this particular algorithm is $\mathcal{O}(|N|^2)$, where $|N|$ is the number of nodes. However, an implementation-specific runtime is exploited due to the sparsity of the graph, which provides a drastically more efficient theoretic runtime of $\mathcal{O}(|E| + |N| \log |N|)$, where $|E|$ is the number of edges in the graph.

This improvement in the algorithm that takes advantage of these two properties is very important. It is easily seen in the following chart how much time and computation is saved.

3.2.4 Extending the Clusters

Once the clusters have been identified, we model the interactions between cells and the movement of damage vacancies in the lattice. We allow the vacancies to "migrate" to adjacent cells of the clusters. Thus, we create an "extendedCluster," depicted below:

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Figure 3.7: Growth rates of common functions measured in nanoseconds.

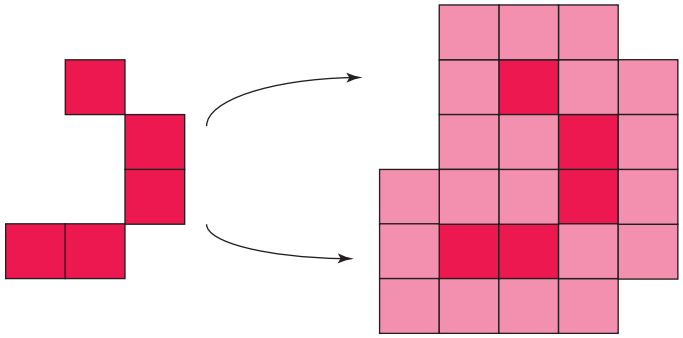

Figure 3.8: An example of an extendedCluster.

Definition 5. *As shown above, an **extendedCluster** is a **cluster** along with all immediately adjacent non-damaged cells*

We now illustrate how we model the movement of the damage vacancies and voids in the lattice material through the use of stochastic cellular automata rules. We establish several approaches to modeling the natural phenomena along with detailing the advantages and disadvantages of each approach.

When executing the behavioral rules of the individual clusters, there are two main considerations: whether to use Global Probability or Local Probability rules.

Definition 6. ***Global Probability Rules*** *are defined by the property that the probability that any individual cell has vacancy in the next time step is dependent on the* entire *cluster that is belongs to.*

Definition 7. ***Local Probability Rules*** *are defined by the property that the probability that any individual cell has vacancy in the next time step is dependent only on its immediate neighbors.*

The properties associated with each approach should “match" the physics of the particular problem and thus the decision as to which approach to use should be dictated by the physics itself.

In general, at each step of the Markov Chain, we “roll the dice" for each cell in the extended-Clusters to decide whether an individual cell receives any of the cluster’s total vacancy in the next iteration. Thus, a cluster’s total vacancy can migrate from the cluster at time T to include the cells in the extendedCluster at time $T + 1$. This allows for movement of the entire cluster as well as interesting behaviors. In the following section, we discuss the motivations and consequences of each modeling approach.

3.2.5 Global Probability Schemes

The assumptions in this approach are:

1. The probability of inclusion in the next step is proportional to the number of immediate neighbors with void; this places higher probability on those surrounded by more vacancy.

2. The larger a cluster is, the lower the probability of any individual cell in the extended cluster receives vacancy.

We then “roll the dice" on each cell in the extendedCluster from a Bernoulli random variable with each cell’s inclusion probability as the probability of success. Here, a success would mean that vacancy from the cluster is distributed to the cell in the next step. We illustrate the inclusion probabilities in the following figure:

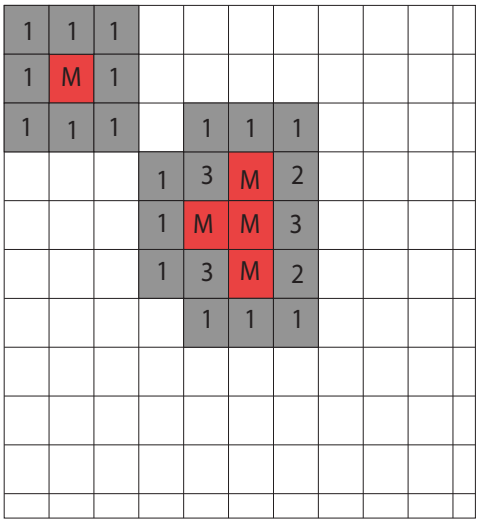

Figure 3.9: Clusters are in red, while extendedClusters include the grey cells. Inclusion probability weights are displayed inside of each cell.

Here, we have added “m” to the inclusion weight of each cell with void. Thus, when we perform the realization of the Bernoulli random variable on each cell according to

$$p_i = \frac{w_i^\alpha}{\sum_J w_j^\alpha}, \tag{3.1}$$

where w_i is the inclusion weight (which is the total number of neighbors of cell i with vacancy), and J is an arbitrary index over all cells in the extendedCluster, there are some immediate consequences which we further discuss in the following sections. This model has two tuning parameters, α and m, which control behavior of the model.

Global Probability Distribution Scheme 1: Maintain old vacancy cells

The main idea here, however, is that a cell with vacancy will always be a cell with vacancy. More precisely, we place the constraint that if a current cell has vacancy, then it maintains vacancy (not necessarily the same amount) in the next step of the chain. We then flip the coin on each cell in the extendedCluster according to the Bernoulli random variable with its corresponding inclusion probability, Equation (2) on previous page, to see if they receive vacancy and are included in the Cluster in the next step of the chain. Since we distribute the total cluster vacancy to new cells, the amount of vacancy in an old vacancy cell may be equal or less than the current time step.

In this approach we have to introduce a healing mechanism is needed or the grid of cells eventually all turn to vacancy as we continue to introduce damage to the window. This is illustrated in the following simulation:

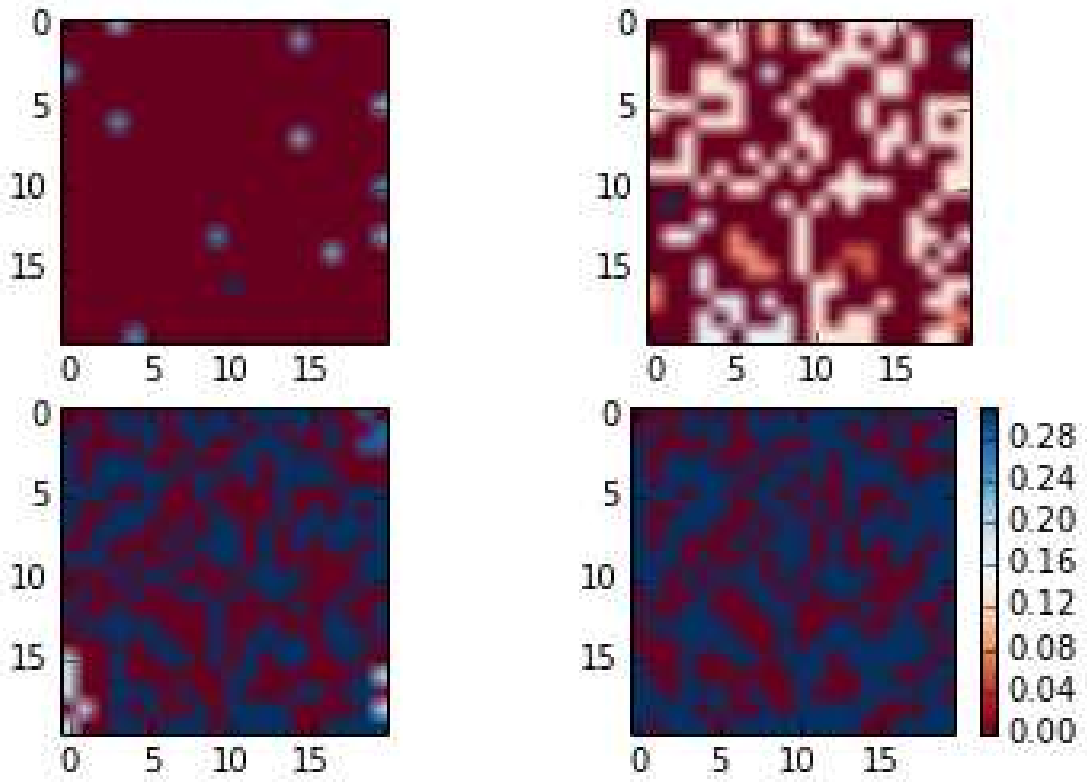

Figure 3.10: Simulation of Scheme 1 progression, without a healing mechanism.

Global Probability Distribution Scheme 2: no constraint on old vacancy cells

Now, we still flip a coin on the extendedCluster cells for inclusion in the vacancy cluster, but we also flip a coin on the old vacancy cells as well. However, as it seems very plausible that the old cells ought to have a higher probability of inclusion in the next step of the chain, we devise an inclusion scheme such that the parameter m allows varying weights to be placed onto current vacancy cells, is introduced. We illustrate in Figure 3.11.

Thus, each current vacancy cell has the following probability of having vacancy in the next time step:

$$p_i = \frac{w_i^\alpha}{\sum_J w_j^\alpha} = \frac{m^\alpha}{5m^\alpha + 27}, \tag{3.2}$$

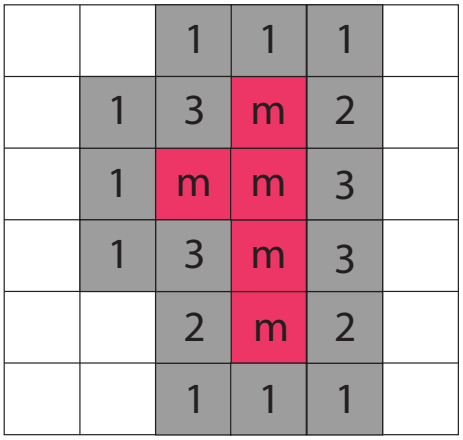

Figure 3.11: Parameter m allows control over the degree a current vacancy cell maintains vacancy.

while the cells in the extendedCluster (shown above in grey) have the following probability of inclusion in the next step:

$$p_i = \frac{w_i^\alpha}{\sum_J w_j^\alpha} = \frac{w_i^\alpha}{5m^\alpha + 27}, \tag{3.3}$$

where w_i is either 1, 2, or 3. This is a $much$ smaller probability than for the first scheme.

The consequences are shown below:

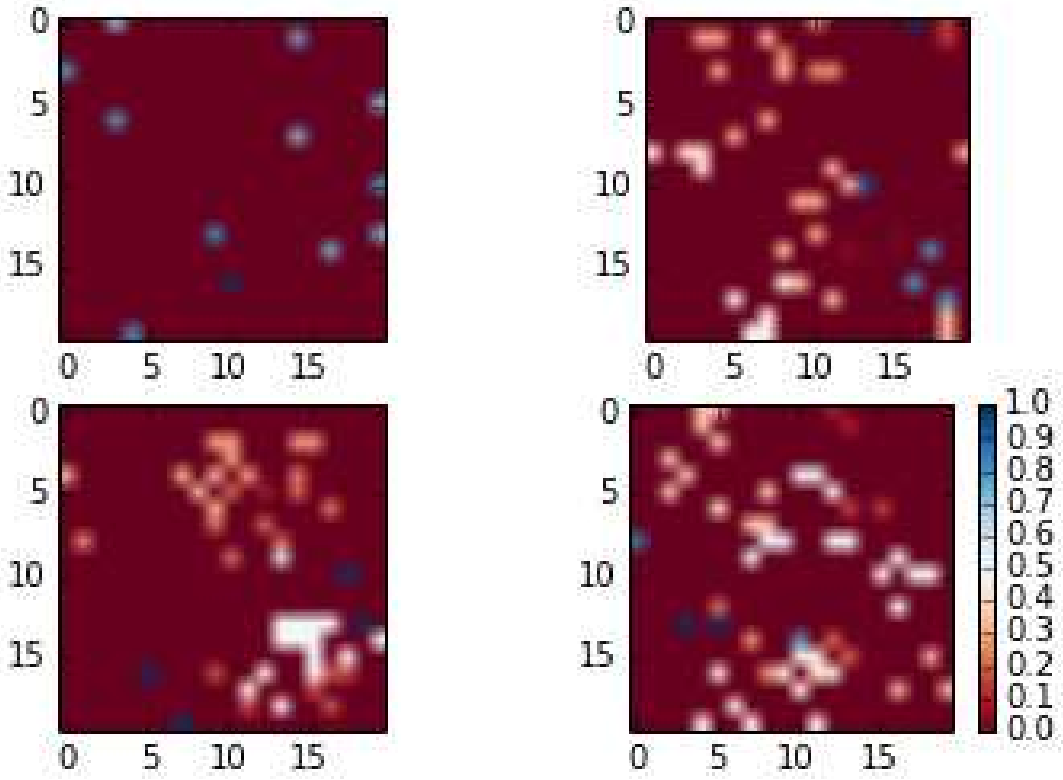

Figure 3.12: Simulation of Scheme 2 progression.

Observations for Scheme 2:

1. the probability of an extended cell joining the cluster $\to 0$ as $m \to \infty$

2. the expected size of the cluster is 1 as $m \to \infty$

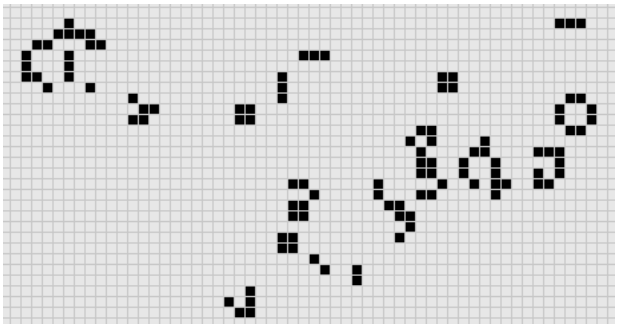

Figure 3.13: An single initial configuration of Conway's "Game of Life."

3.2.6 Local Probability Schemes

As defined earlier, these cluster behavior rules are defined by the principal characteristic that, at each iteration of the markov chain, the probability that any individual cell has vacancy at the next step is dependent only on each individual cell's immediate neighbors.

This approach is a variation of the well-studied Game of Life, by John Conway [10].

John Conway's "Game of Life"

Conway's "life", illustrated below, is a deterministic process on a two-dimensional lattice whose long-term properties and states only depends on the initial starting conditions. Each automata on the lattice takes the value of either a 0 or a 1.

The evolution of the initial state at each time transition from time t to $t + 1$ depends on the following deterministic rules:

1. Survivals. Every counter with two or three neighboring counters survives for the next generation.

2. Deaths. Each counter with four or more neighbors dies (is removed) from overpopulation. Every counter with one neighbor or none dies from isolation.

3. Births. Each empty cell adjacent to exactly three neighbors–no more, no fewer–is a birth cell. A counter is placed on it at the next move.

This process yields some fascinating “steady states" and shapes that never die out. Certain neighborhood configurations lead to perpetual behaviors which have been well studied and have puzzled mathematicians for decades.

Stochastic Game of Life

The Stochastic Game of Life is a stochastic analog of the aforementioned process. We define the probability rules as the following: let p_i be the inclusion probability for every cell i at each iteration:

$$p_i = \frac{\sum_J m_j}{\sum_J m_j + 1}, \tag{3.4}$$

where J is all adjacent neighbors of cell i, and m_j is the weight given to every cell, equaling the following:

$$m_j = \begin{cases} m, & \text{if vacancy}_j > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{3.5}$$

Using these inclusion probability for every cell in the grid, at each iteration we “flip the coin" for each cell. Thus, the inclusion probabilities are based solely on the immediate neighbors of every cell, and at each step of the chain, every cell is randomly drawn for inclusion in the next step.

We illustrate the probabilities for varying parameter values, m, below:

m=1

0	0	0	0	0	0
0	1/2	1/2	1/2	0	0
0	2/3	2/3	3/4	1/2	0
0	2/3	2/3	2/3	1/2	0
0	1/2	2/3	2/3	1/2	0
0	0	0	0	0	0

m=1/5

0	0	0	0	0	0
0	1/6	1/6	1/6	0	0
0	2/7	2/7	3/8	1/6	0
0	2/7	2/7	2/7	1/6	0
0	1/6	2/7	2/7	1/6	0
0	0	0	0	0	0

Figure 3.14: $m = 1$ on the left and $m = 1/5$ on the right.

We illustrate a simulation of this local probability scheme in Figure 18, using $m = 1/5$:

Observations for the Local Probability approach:

1. The probability of an extended cell joining the cluster $\to 0$ as $m \to 0$ but $\to 1$ as $m \to \infty$;

2. The expected size of the cluster is determined by the size of the cluster since each of the local probabilities is determined by the 8 adjacent neighbor cells;

3. There is great flexibility in both the movement and growth/shrinkage of clusters;

4. There are drastic interaction effects between the parameter m and the introduction of damage.

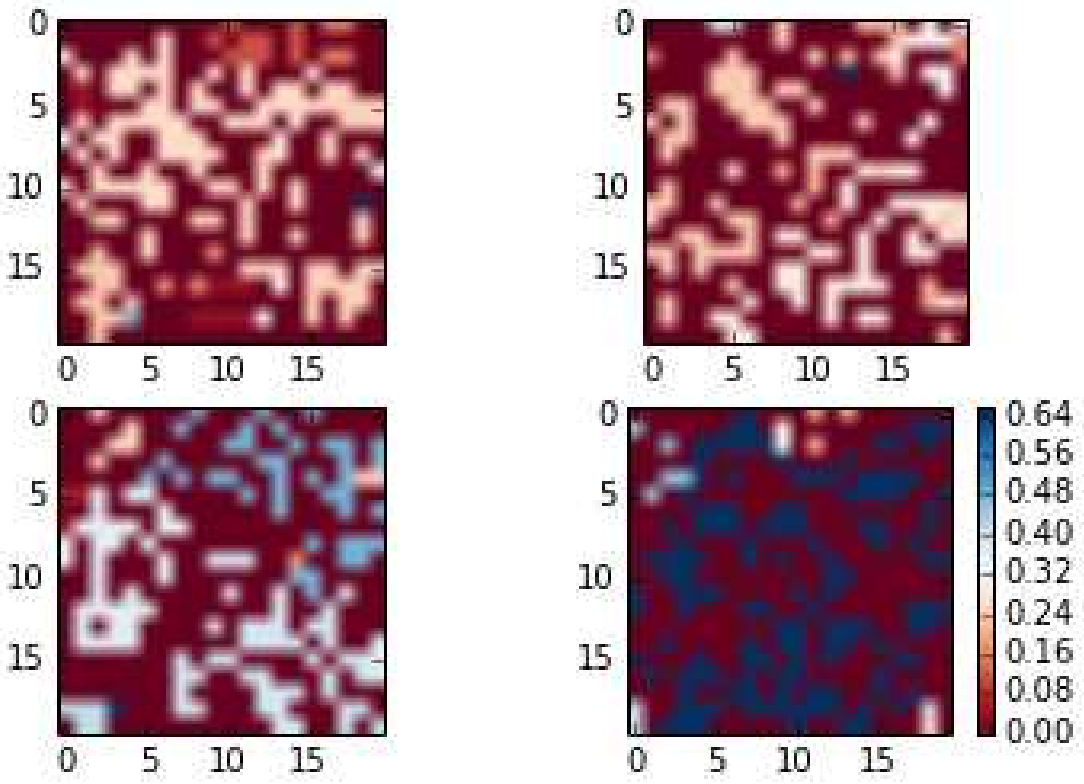

Figure 3.15: Read left to right, top to bottom: T=10, T=20, T=30, T=40

3.2.7 Using Local Probability Rules is the best model for Uranium damage

Our model is chiefly motivated by a natural phenomena with basic properties observed. Built from first principals, we choose the model that most appropriately satisfies these properties.

The first two models are deficient for satisfying the model criteria described in Section 2.1. The first model has predictable growth patterns, while the second has predictable abating patterns. Through trade-offs and dualities of these models, as we change the tuning parameters to allow for more flexible movement and migration behaviors, we inevitably force the models into undesirable states.

Our third model based on locally-driven probabilities, however, is a rich class of models that allows for great flexibility and user control. The model itself is not physics driven, but rather places the unique assumption that movements and interactions be random. The sole parameter presents a

simple model which affords complete situational discretion as to the tendency of cluster behavior, while still maintaining unpredictable movement. As was desired under the Global Probability Schemes, this last model also follows the principals relating to the size of a cluster and the amount of immediate neighbors with vacancy.

Not only is the third model simple, with the fewest assumptions imposed upon it, but it is also the most flexible model which satisfies all of the model criteria (Section 2.1).

Vacancy Distribution:

Since new cells may be included in the cluster for each time step, and some currently-damaged cells may not be included, we must redistribute the vacancies in each cluster at each step of the Markov Chain. We currently implement this by summing the total amount of proportion damaged over every cell in the cluster, and uniformly distributing this total cluster damage over the damaged cells in the next step. This is done for every cluster at every step.

3.2.8 Algorithm to Couple the Micro and Meso Models:

We now present an algorithmic view of the full model, below:

Algorithm 2 Coupling together the Micro and Meso Scales

initialize D_T = n $\times$ n zero matrix ▷ n is the number of rows/columns in our window

draw $N(T) \sim$ Poisson($T\lambda$) ▷ create an initial state of damage
for i in N(T) **do**
    $p_i =$ draw from cascade vacancy distribution
    $x_i =$ draw from discrete unif(0,n)
    $y_i =$ draw from discrete unif(0,n)
    $D_{T_{x_i,y_i}} = D_{T_{x_i,y_i}} + p_i$
end for

create list of all $clusters$

for each $cluster$ **do**
    create $extendedCluster$
end for

for each $extendedCluster$ **do**
    for each cell in $extendedCluster_j$ **do**
        draw from Bernoulli(p_i) ▷ p_i is defined in Eq. 3.4
    end for
end for

draw $N(T) \sim$ Poisson($T\lambda$) and repeat above steps

3.3 Time Series Representation of PCA:

This is the layer of the model that tracks the total proportion of damage within a window (defined here as a grid of "cells") over larger time steps than the individual cascade events. We first define the following:

$$\pi_j = \text{proportion of window damaged at time } T_j, \forall\ j = 1, 2, ...$$

which, mathematically, is determined as:

$$\pi_j = \pi_{(j-1)} + \frac{\sum_{i=j_1}^{j_n} m_i - \sum_{i \in I_j} m_i}{k},$$

where

j_1: indexes the first cascade occurrence during time period $(T_j : T_{j+1})$

j_n: indexes the last cascade occurrence during time period $(T_j : T_{j+1})$

I_j: is an arbitrary set of cascades that are healed at time T_j

k: the total number of atoms in the window (the grid of individual cells)

It is easy to see that π_j is the accumulation of additional collision cascades during the time period $(T_{j-1} : T_j)$, along with an instantaneous healing of a random subset of the previous vacancies, at T_j. This series is depicted in the following plot (Figure 6), where π_j is the value of the series at time T_j:

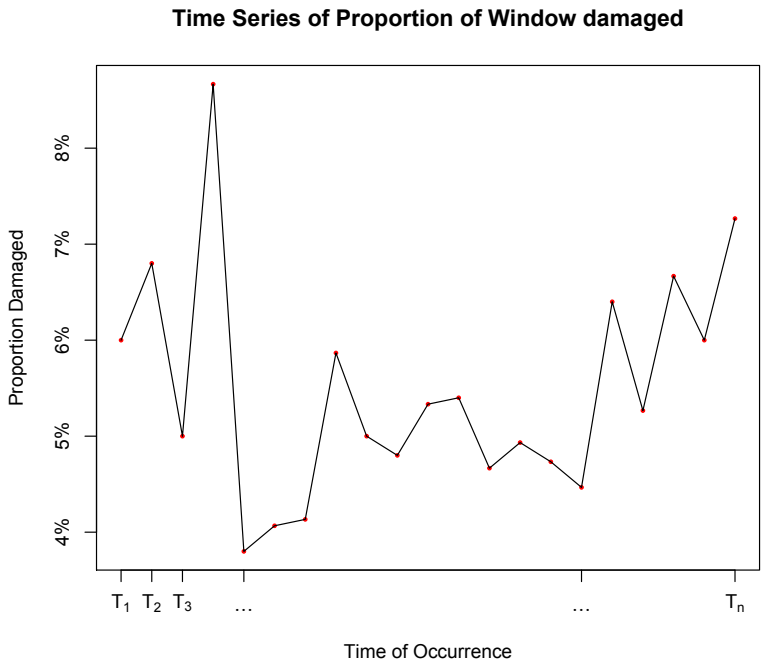

Figure 3.16: The time series representing percent of window damaged.

Modeling this Stochastic Process as a Markov Chain:

This process can be described using an additive markov-chain model:

$$\begin{cases} \hat{X}_{T+1} = X_T + D_T + H_T \\ X_{T+1} = HD_{T+1}(\hat{X}_{T+1}) \end{cases}$$

where the random matrices are defined as the following:

$X_T =$ is the current state, at time T, of proportions of damage in each cell,

where $X_{ii} =$ is the proportion of damage in cell i, i in the window

$D_T =$ is a grid of cells where D_{ii} is the proportion of damage in cell i, i at time T;

$H_T =$ is a grid of cells where H_{ii} is the proportion of healing in cell i, i at time T;

$HD_{T+1}(\hat{X}_{T+1})$ is a non-linear operator which executes the behavior rules of the cell-interactions.

*note: While π_j, of the previous page, represents the proportion of the entire window that is damaged, X_T is a matrix representation where each entry corresponds to the proportion damaged in an individual cell.

The following pseudo-code describes the simulation process of this Markov-chain:

Simulation of D_T:

Algorithm 3 Construction of D_T

initialize D_T = n $\times$ n zero matrix ▷ n is the number of rows/columns in our window
draw $N(T) \sim$ Poisson($T\lambda$)

for i in N(T) **do**
  $p_i =$ draw from cascade vacancy distribution
  $x_i =$ draw from discrete unif(0,n)
  $y_i =$ draw from discrete unif(0,n)
  $D_{T_{x_i,y_i}} = D_{T_{x_i,y_i}} + p_i$
end for

Since we are modeling the waiting times using an exponential distribution, the number of cascade events in any given time period, T, follows a Poisson distribution. As such, we draw from this distribution for each time step to determine the number of damage events.

We observe that the irradiation damage is independent of the current state because it is produced and driven by forces outside of the particular window. Thus, we model the phenomenon as a spatially uncorrelated process. We draw a damage proportion from the cascade vacancy distribution and randomly select the cells that receive the damage, using a uniform probability distribution.

Simulation of H_T:

We use the following pseudo-code to describe the simulation process of the random healing matrix:

Algorithm 4 Construction of H_T

initialize H_T = n $\times$ n zero matrix
draw n_T from a discrete unif(0, $\aleph_{T-1}$)
▷ where $\aleph_T$ is number of cells damaged at time T

for i in $\aleph_{T-1}$ **do**
    unif_sample= draw from Bernoulli(p= $n_T/\aleph_{T-1}$)
    if unif_sample ==0 **then**
        nothing
    else
        set $H_{T_{j,k}} = -X_{T_{j,k}}$ ▷ where (j,k) indexes the ith damaged cell in X_T
    end if
end for

This construction of the random healing matrix chooses a number of the damaged cells to heal, corresponding to the arbitrarily-indexed set I_j on page 9. The 'for' loop, in the algorithm, accomplishes this by sampling a random subset of the damaged cells to heal on average, the number of cells that heal is the mean of this discrete uniform distribution.

Simulation of HD_T:

To describe the non-linear operator that induces the spatial behavior/interaction rules of the damaged cells, we first establish the following assumptions:

- Only immediate neighbors can have vacancies migrate between them;
- Migration always occurs in the direction of the cell with the larger void fraction;
- The migration occurs with some probability and the amount of void migration is a random fraction of the vacancies.

To generate this operator -under the previously stated assumptions- we use the following pseudo-code:

We are now using the notation that $\pi_{i,j}$ is the proportion of damage in cell i,j.

Algorithm 5 Construction of HD_{T+1}

for each pair of $C_{i,j}$, $\forall i, j$ s.t. $\pi_{i,j}, \pi_{i^*,j^*} > 0$ **do**
  draw $X \sim Bernoulli(p^*)$
  if X == 0 **then**
    no migration
  else
    draw $p \sim unif(0,1)$ ▷ proportion to migrate, perhaps beta
    if $(\pi_{i,j} \geq \pi_{i^*,j^*})$ **then**
      set $\hat{\pi}_{i^*,j^*} = p \cdot \pi_{i^*,j^*}$
      if $(\hat{\pi}_{i^*,j^*} < 1 - \pi_{i,j})$ **then** ▷ Can't migrate more than (1 - π_{i^*,j^*})
        $\pi_{i,j} = \pi_{i,j} + \hat{\pi}_{i^*,j^*}$ && $\pi_{i^*,j^*} = \pi_{i^*,j^*} - \hat{\pi}_{i^*,j^*}$
      else
        $\pi_{i,j} = 1$ && $\pi_{i^*,j^*} = \pi_{i^*,j^*} - (1 - \pi_{i,j})$ ▷ If more told to migrate than possible,
      end if ▷ then migrate all possible
    else
      switch i,j
    end if
  end if
end for

Here, we are iterating over all possible adjacent cells that have some amount of vacancies within them. We then randomly draw a proportion of the damage in the less-damaged cells to migrate over to the more-damaged cells, thus concentrating the vacancies according to the rules/assumptions detailed on the previous page.

Chapter 4

Properties of *Probabilistic Cellular Automata*

During the 60's and 70's, of the twentieth century, several classes of interacting Markov processes were studied; broadly related to the models introduced in Chapter 4. The object of much research throughout the 70's and 80's, these processes encompass a large class of different models. Collectively referred to as infinite particle systems in the stochastic process literature, they comprise models such as the Ising and Gibbs models, additive processes, and cellular automata. In what follows, we outline the general development of these models and their characteristics which are most pertinent to model detailed in Chapter 4. We also outline some of the more relevant properties and results of these processes.

4.1 *Interacting Particle Systems*

This family of stochastic processes is seemingly amenable to a theoretic analysis in the probability theory framework. As with *PCA*, *interacting particle systems* are Markov processes on a state space with a product form. These processes are *continuous* in time and are defined through their updating *rate*. As was briefly described above, there are a few general results for these processes and then there exist more detailed analyses for more specific models.

Of the *interacting particle systems* that have more well-developed theoretical results, they tend to be of the class in which the updating procedure is sequential: at most one site of the network is updated when an *exponentially distributed* "clock rings." While an analogous discrete time processes description is possible, this is in large contrast to the framework of *cellular automata* in which *all* sites are updated at each discrete time step.

Let G be a countable set, which is usually $Z^d = \{(n_1, ..., n_d) : n_i \text{ are integers}\}$, a set of sites (locations in space) where the events of interest occur. Let S be a two element set $\{0, 1\}$, the set of states in which we can find the sites (infected or healthy, spin up or down, occupied or not, etc.).

Let $\chi = S^G$ be the set of all functions from G to S. If $\xi \in \chi$, then $\xi(x)$ gives the state at x. ξ describes the *configuration* of the system. An *interacting particle system* is a continuous-time, strong Markov stochastic process $(\xi_t)_{t\in\mathbb{R}^+}$ with state space ξ; defined on S^G. Let $\Lambda =$ the set of all subsets of G. If $a \in S$, the mapping $\xi \to \{x : \xi(x) = a\}$ gives a 1 to 1 correspondence of χ or Λ.

The process is described by giving a function c: $\Lambda \times \chi \to [0, \infty)$ which determines the evolution through the following:

$$\mathbb{P}(\xi_{t+\delta}(x) \neq \xi_t(x) \text{for all x} \in E | \xi_t = \xi) = \left(\sum_{F \supset E} c(F, \xi) \right) \delta + o(\delta), \tag{4.1}$$

as $\delta \to 0$. In words, $c(E, \xi)$ is the rate at which we flip the values at all the sites in E when the configuration is ξ. Most of the models in this class of processes that have been studied assume that $c(E, \xi) = 0$ unless $|E|$, the cardinality of a set, is either 1 or 2. The processes that were most heavily studied fall into the following two categories:

1. *Spin flip systems*: $c(E, \xi) = 0$ unless $|E| = 1$; only one site flips at a time and $c(\{x\}, \xi) = c(x, \xi)$.

2. *Particle motion systems*: $S = \{0, 1\}$ and $c(E, \xi)$ unless $E = \{x, y\}$ and $\xi(x) = 1$,$\xi(y) = 0$ or $|E| = 1$. In these models, ones mark the locations of particles and in the first case a particle jumps from x to y and in the second a particle appears or disappears.

An important question, and a natural question for an astute reader, is whether or not specifying the transition rates, (4.1), defines a unique Markov process.

4.1.1 Vasil'ev's model: Russian Lamps

Vasil'ev's model is based on a more general definition of *One sided nearest neighbor systems.* In these processes, we define $G = Z^1$ and $S = \{0, 1\}$. The transition probabilities are defined as

$$c(x, \xi) = f(\xi(x), \xi(x + 1)) > 0 \quad \forall \ \ x, \xi, \tag{4.2}$$

Surprisingly, though having been extensively studied, and having a deceptively simple definition, it is not known if the processes in this class always have a unique stationary distribution.

Vasil'ev describes the situation as a process over a row of lamps which is infinite in both directions. Each lamp continues to burn at the next instant if, at the given instant, it, together with its neighboring lamp, is on. The lamp can turn on with probability θ, a small positive number, in other cases. The formal definition is:

$$P[\xi_{t+1}(x_n) = 1] = \begin{cases} 1, & \xi_t(x_n) = \xi_t(x_{n+1}) = 1, \\ \theta, & \xi_t(x_n)\xi_t(x_{n+1}) = 0. \end{cases} \tag{4.3}$$

It is known that for any initial state of this medium, the probability distribution on the state space asymptotically approaches a linear combination of two fixed stationary distributions, one corresponding to the "all lamps on" state, while the other is regular [].

4.1.2 *The Ising model*

The previous definition of processes may appear to suggest that if G is finite and $c(x, \xi) > 0$ for all x, ξ, the process is automatically reducible, thus yielding a unique stationary distribution. However, the *Ising model* shows this intuition is not true.

In the *Ising model*, we think of the sites as iron atoms whose magnetic north pole may be pointed up or down $\{1, -1\}$. Thus, $G = Z^d$ and $S = \{-1, 1\}$. As is clear in the following definition of the spin rates, the sum in the exponent measures the extent to which $\xi(x)$ is in accordance with its neighbors:

$$c(x, \xi) = exp\left(-\beta \sum_{u,||u||=1} \xi(x)\xi(x+u)\right), \qquad \beta \geq 0. \tag{4.4}$$

If the alignment of the neighboring atoms is bad, then the flip rate is large and converse.

This model is easy to study in one dimension, but becomes much more interesting for $d \geq 2$, since there is more than one stationary distribution. Holley and Stroock [] have shown that there is

a unique transition matrix and stationary distributions for each value of β in the one-dimensional case.

4.1.3 *Contact processes*

In *contact processes*, we think of the sites as cells arranged in a lattice in a body. The states 0 and 1 correspond to the site being 'healthy' or 'infected.' Here we have that $G = Z^d$ and $S = \{0, 1\}$ and the transition probabilities are defined as,

$$c(x, \xi) = \begin{cases} 1, & \xi(x) = 1, \\ k\lambda, & \xi(x) = 0 \text{ and } \sum_{y \in E} \xi(x + y) = k. \end{cases} \tag{4.5}$$

Here, the flip rates say that infected sites recover at a rate of 1 while healthy cells are infected at a rate proportional to the number of sites in $x + E$ which are infected. This process shares similarities with the probabilistic cellular automata we have introduced.

Harris [] has derived conditions under which $p_t(\xi) = \mathbb{P}(\xi_t \neq \emptyset | \xi_0 = \xi)$ is increasing, subadditive, or submodular in ξ. In the case of general *contact processes*, conditions have been derived, [], that imply that $p_\infty(\xi) = 0$ for all finite ξ, or that the contrary is true. In other cases, conditions for ergodicity are given [].

4.2 *Probabilistic Cellular Automata*

Probabilistic cellular automata (PCA) are discrete-time Markov processes modeling the time evolution of a multicomponent system. A defining characteristic of these processes is the synchronous updating of the states of the components, which must take values in a finite set and interact with their neighbors according to a updating rule defined through a probability. Thus, they are a generalization of cellular automata (CA) where the updating rule is defined through a probability.

Analogous to their deterministic counterpart, PCA are a useful class of models for simulation and analysis of complex systems. The class of PCA comprise a large family of discrete-time

Markov stochastic processes where the transition probability has a product form (*synchronous updating*).

For a site to have its state updated, it is highly dependent on the state of its immediate neighborhood. While they are simple in definition, discrete in space and time, and the interactions are only local, the behaviors exhibited are complex and often illustrate non-equilibrium phenomena, *phase transitions*, and supercritical regimes. Critical values can show a transition from ergodic to non-ergodic regimes. In other words, if the free parameters are set above or below a certain critical threshold, at infinite time the process may or may not preserve part of the information of its initial condition. Thusly, the probability measure at infinite time may or may not depend on the initial state and it may or may not admit a unique, attracting invariant measure.

4.2.1 Stavskaya's probabilistic cellular automaton

Stavskaya's model is a two-state PCA defined on a one-dimensional periodic lattice of L cells specified by the familiar configuration $\xi(t) = (\xi_1(t), \xi_2(t), ..., \xi_L(t)) \in \{0,1\}^L$. The model evolves in discrete time $t \in \mathbb{N}$ according to the very simple rule: with probability $\epsilon \in [0,1]$, $\xi_{t+1}(x_n) = 1$, otherwise $\xi_{t+1}(x_n) = xi_t(x_n)xi_t(x_{n-1})$.

It is easy to convince yourself that $\mathbf{1} = (1,1,...,1)$ is an absorbing state of the model. It has also been shown that there exists a critical ϵ^* such that for all $\epsilon > \epsilon^*$ the only invariant measure of Stavskaya's PCA is $\delta_{\mathbf{1}}$, the measure concentrated in $\mathbf{1}$, and that for $\epsilon < \epsilon^*$ the invariant measures are translation-invariant convex combinations of the form $\alpha\mu_\epsilon + (1-\alpha)\delta_{\mathbf{1}}$, with $0 < \alpha < 1$ and μ_ϵ the measure that puts mass on configurations with density $0 < \mu_\epsilon(1) < 1$.

4.3 Definition of a *Generalized Probabilistic Cellular Automata (GPCA)*

Let $\mathbf{Z}$ be the 1-dimensional lattice of points $x = x_i$, where each x_i is an integer. In the Russian fashion, these sites, or particles, should be thought of as a row of lamps which can either be lit or not. Thusly, by a "random medium" we are referring to a system of automata, each of which can be

in one of two states, 0 or 1. The state of an automaton at time $t+1$ is probabilistically determined depending on the states of "neighboring" automata at time t.

Call x and y **neighbors**, and write $x \sim y$ if $|x-y|=1$.

If $x \notin E$ but $x \sim y$ for some $y \in E$, then x is a neighbor of $E \subset Z$.

E^N denotes the set of all neighbors of E and $E^+ = E \cup E^N$

N_x denotes the set of all neighbors of point x.

Let Ξ be the set of subsets of $\mathbf{Z}$ and Ξ_0 the set of finite subsets of $\mathbf{Z}$.

Each $\xi \in \Xi$ can be considered a map from $\mathbf{Z}$ into {0,1}, where $\xi(x)=1$ if $x \in \xi$ and $\xi(x)=0$ if $x \notin \xi$.

If $E \in \mathbf{Z}$, then $\xi(E)$ denotes the number of points of ξ in E.

Let $|\xi|$ be the number of points in ξ, $|\xi| \leq \infty$.

4.3.1 Definition of *GPCA*

We construct a discrete time Markov Process ξ_t whose values are subsets of $\mathbf{Z}$, the integers. We write $\xi(x)=1$ if $x \in \xi_t$ and 0 otherwise. The transition probability for a change in $\xi_t(x)$ depends on $\xi_t(y)$, y a neighbor of x. In the model, $\xi_t(x)$ can change from a 0 to 1 only if $\xi_t(y)=1$ for some $y \sim x$. Let $p_t(\xi) = P(\xi_t \neq \emptyset | \xi_0 = \xi)$.

In the example of a random medium, the automaton are arranged in a chain and are numbered as follows: $n = \ldots, -1, 0, 1, 2, \ldots$. For each x_n, we have that $\xi_t(N_x) = k$ for $k = 0, 1, 2$. The probability that at time $t+1$ automaton n is be in state $\xi_{t+1}(x_n) = 1$ is

$$P[\xi_{t+1}(x_n)=1] = \begin{cases} \frac{km}{1+km}, & \xi_t(x_n)=0, \\ \frac{1}{1+n}, & \xi_t(x_n)=1. \end{cases}$$

This is the "birth probability" (infection). The "death probability" (recovery) is

$$P[\xi_{t+1}(x_n) = 0] = \begin{cases} \frac{1}{1+km}, & \xi_t(x_n) = 0, \\ \frac{n}{1+n}, & \xi_t(x_n) = 1. \end{cases}$$

In addition, we define the following:

$\tau(x_n) = \inf\{s : \xi_s(x_n) = 1\}$: the first time $\xi_t(x_n) = 1$; if $\xi_0(x_n) = 1$ then $\tau(x_n) = 0$;

$P_t(\xi) = P(\xi_t \neq \emptyset | \xi_0 = \xi)$; survival probability of the chain;

$m_t(\xi) = E(|\xi_t||\xi_0 = \xi)$.

4.4 Analytic Space and Time Properties of the *GPCA*

The *GPCA* has properties that make it suitable for modeling non-equilibrium phenomena. Although simple, their behavior tends to be quite complex and difficult to analyze.

One interesting property to study is the transition of phases; eg., from an ergodic regime to one of non-ergodicity. By setting the sole free parameter above or below a threshold, the process may preserve the information of its initial configuration (non-ergodic). In other words, the probability measure at infinite time depends on the initial state of the process. However, if the parameter regime permits the process to be ergodic, then the stochastic process possesses a unique, attracting invariant measure. [] shows the connection between a PCA's non-ergotic regime and the existence of a phase transition, considered a statistical mechanics system.

In the quest to characterize the rate of convergence to these equilibrium states or the characterization of the particular invariant measures, the first step is understanding the rate of dispersion or spread.

4.4.1 *Infection rates*

By simplifying the stochastic process to the 1-dimensional lattice $\mathbf{Z}$, the relevant, Von Neumann, neighborhood of a cell becomes the cells to the left and right. This enables us to derive a

probabilistic lower bound on the number of time steps it takes before a cell is infected, which is dependent on the neighborWeight parameter, Eq. 3.5.

Lemma 1. *Let* $\Lambda = \max\{\frac{m}{1+m}, \frac{2m}{1+2m}\} = \frac{2m}{1+2m}$. *Then,* $P[\tau(x) \leq t] \leq 1 - (\frac{1}{1+2m})^t$, *if* $\xi_0(x) = 0$, $t > 0$.

Proof.

$$
\begin{aligned}
P[\tau(x) \leq t] &= 1 - P[\tau(x) > t] \\
&= 1 - P(\text{x is not infected from time 0 through t}) \\
&= 1 - P(\xi_t(x) = 0 | \xi_{t-1}(x) = 0) \times \\
&\qquad P(\xi_{t-1}(x) = 0 | \xi_{t-2}(x) = 0) \cdots P(\xi_1(x) = 0 | \xi_0(x) = 0) \\
&\leq 1 - \left(1 - \frac{2m}{1+2m}\right) \cdots \left(1 - \frac{2m}{1+2m}\right) \\
&= 1 - \left(\frac{1}{1+2m}\right)^t
\end{aligned}
$$

□

This gives a loose starting bound with which to build a tighter approximation of the true infection rates shown below.

4.4.2 *Infection chains*

From this idea of infection rates, we carry to the concept of an entire infection chain. Since we have restricted the model to a process in which no outside production of damage source is occurring, a cell only receives damage (infection) if damage passes through every single cell between it and the originally damaged cell.

This restriction enables us to define the *infection chain* in the statement of the following lemma, and we arrive at a tighter probabilistic lower bound for the time it takes for a cell to become “infected."

Lemma 2. *Given $x_0, x_1, \ldots, x_n$ consecutive points on the lattice* $\mathbf{Z}$ *such that $x_i \sim x_j$ for all $|i-j| = 1$, and $\xi_0(x_0) = 1$ and $\xi_0(x_i) = 0 \; \forall \; i \neq 0$, then we have that* $P\left(0 < \tau(x_1) < \cdots < \tau(x_n) \leq t\right) \leq 1 - \sum_{j=0}^{n-1} \left[\binom{t}{j} \left(\frac{m}{1+m}\right)^j \left(\frac{1}{1+m}\right)^{t-j} \right]$.

Proof. Let W_t denote the number of first infection within time t.

$$P\left(0 < \tau(x_1) < \cdots < \tau(x_n) \leq t\right) = P(\tau(x_n) \leq t)$$

We have

$$\begin{aligned} P(\tau(x_n) \leq t) &= 1 - P(\tau(x_n) > t) \\ &= 1 - P(W_t \leq n-1) \\ &= 1 - \sum_{j=0}^{n-1} P(W_t = j) \end{aligned}$$

For each $j = 0, \ldots, n-1$, we compute $P(W_t = j)$, $(0 \leq j \leq n-1)$. Let $t_1, \ldots, t_j$ be constants with $0 < t_1 < t_2 < \cdots < t_j < t$. Then there are $\binom{t}{j}$ possible combinations of $t_1, t_2, \ldots, t_j$, and

$$\begin{aligned} P(W_t = j) &= \sum_{t_j} \sum_{t_{j-1}} \cdots \sum_{t_1} \left[P(\tau(x_{j+1}) > t) P(\tau(x_j) = t_j) \ldots P(\tau(x_1) = t_1) \right] \\ &= \sum_{t_j} \cdots \sum_{t_1} P(\tau(x_{j+1}) > t | \tau(x_j) = t_j) \times \\ &\qquad P(\tau(x_j) = t_j | \tau(x_{j-1}) = t_{j-1}) \cdots P(\tau(x_1) = t_1 | \tau(x_0) = t_0) \end{aligned}$$

To find the above, we note that

$$P(\tau(x_{j+1}) > t | \tau(x_j) = t_j) \geq \prod_{T=t_j+1}^{t} \left[P(\xi_T(x_{j+1}) = 0 | \xi_{T-1}(x_j) = 1) \right] = \left(\frac{1}{1+m} \right)^{t-t_j},$$

and $\forall$ i with $1 \leq i \leq j$, we also have

$$\begin{aligned} P(\tau(x_i) = t_i | \tau(x_{i-1}) = t_{j-1}) &= P(\xi_{t_i}(x_i) = 1, \xi_t(x_i) = 0 for T < t_i | \xi_{t_{i-1}}(x_{i-1}) = 1) \\ &\geq P(\xi_{t_i}(x_i) = 1 | \xi_{t_{i-1}}(x_{i-1}) = 0) \times \\ &\prod_{T=t_{i-1}+1}^{t_i - 1} [P(\xi_t(x_i) = 0 | \xi_{T-1}(x_{i-1}) = 1)] \\ &= \frac{m}{1+m} \cdot \left(\frac{1}{1+m}\right)^{t_i - t_{i-1} - 1} \end{aligned}$$

We now simply multiply each of these $P(\tau(x_i) = t_i | \tau(x_{i-1})$ together from $i = 1, \ldots, j$ to find that

$$\begin{aligned} \prod_{i=1}^{j} [P(\tau(x_i) = t_i | \tau(x_{i-1})] &\geq \left(\frac{m}{1+m}\right)^j \cdot \left(\frac{1}{1+m}\right)^{t_i - t_0 - j} \\ &= \left(\frac{m}{1+m}\right)^j \cdot \left(\frac{1}{1+m}\right)^{t_i - j} \end{aligned}$$

Using the fact that $P(\tau(x_{j+1}) > t | \tau(x_j) = t_j) \geq \left(\frac{1}{1+m}\right)^{t - t_j}$, we have

$$\begin{aligned} P(W_t = j) &\geq \sum_{t_j} \cdots \sum_{t_1} \left[\left(\frac{1}{1+m}\right)^{t - t_j} \cdot \left(\frac{m}{1+m}\right)^j \cdot \left(\frac{1}{1_m}\right)^{t_j - j} \right] \\ &\geq \sum_{t_j} \cdots \sum_{t_1} \left[\left(\frac{m}{1+m}\right)^j \cdot \left(\frac{1}{1+m}\right)^{t-j} \right] \\ &\geq \binom{t}{j} \left(\frac{m}{1+m}\right)^j \cdot \left(\frac{1}{1+m}\right)^{t-j} \end{aligned}$$

Putting these inequalities together, we have

$$P(W_t \leq n-1) = \sum_{j=0}^{n-1} [P(W_t = j)]$$
$$\geq \sum_{j=0}^{n-1} \left[\binom{t}{j} \left(\frac{m}{1+m} \right)^j \cdot \left(\frac{1}{1+m} \right)^{t-j} \right]$$

It follows that

$$P(\tau(x_n) \leq t) = 1 - P(W_t \leq n-1)$$
$$\leq 1 - \sum_{j=0}^{n-1} \left[\binom{t}{j} \left(\frac{m}{1+m} \right)^j \cdot \left(\frac{1}{1+m} \right)^{t-j} \right],$$

and

$$P\left(0 < \tau(x_1) < \cdots < \tau(x_n) \leq t\right) \leq 1 - \sum_{j=0}^{n-1} \left[\binom{t}{j} \left(\frac{m}{1+m} \right)^j \left(\frac{1}{1+m} \right)^{t-j} \right].$$

□

Thus, we see that the probability of a single cell becoming infected through its chain of neighbors from the nearest cell that is infected is a geometric probability with parameters based on the transmission weights and the distance between the two points.

This result does not generalize easily into $d > 1$ dimensions. It is easy to see that for any cell to be infected in this case, where $\mathbf{Z}$ is our lattice on which our automaton exists, it must pass through a clear sequential series of neighbors. However, in any higher dimension the pathway of an infection chain can be much more complex.

Ideas taken from percolation theories use a directional lattice to come up with bounds on these rates and probabilities for other stochastic processes. []

4.4.3 *state maintenance probabilities*

We are interested not only in the time bounds with which infection occurs, but also the probabilities pertaining to the opposite; ie., no change in the system.

The probability that an automaton does not have any change at all in its system over one time step is straightforward to compute. This is shown to be geometric, proportional to the size of the density of active cells at the particular state.

Lemma 3. *The holding time in the state ξ is geometric with coefficient $p = 1 - P(\xi_{t+1} = \xi_t | \xi_t)$, where $P(\xi_{t+1} = \xi_t | \xi_t) = \left(\frac{1}{n}\right)^{|\xi|} \cdot \prod_{x \in \xi} \left(\frac{1}{1+\xi(N_x)m}\right)$.*

Proof. Since $\forall$ points $x \in \mathbf{Z}$,

$$P(\xi_{t+1}(x) = 1 | \xi_t(x) = 0) = \frac{\xi_t(N_x) \cdot m}{1 + \xi_t(N_x) \cdot m},$$
$$P(\xi_{t+1}(x) = 0 | \xi_t(x) = 1) = \frac{n}{1+n}.$$

we have that

$$P(\xi_{t+1}(x) = 0 | \xi_t(x) = 0) = \frac{1}{1 + \xi_t(N_x) \cdot m},$$
$$P(\xi_{t+1}(x) = 1 | \xi_t(x) = 1) = \frac{1}{1+n}.$$

and thus

$$\begin{aligned} P(\text{No change in time t+1}) &= \sum_{x \in \mathbf{Z}} P(\xi_{t+1}(x) = \xi_t(x) | \xi_t(x)) \\ &= \sum_{x \in \xi_t} P(\xi_{t+1}(x) = 1 | \xi_t(x) = 1) \cdot \sum_{x \notin \xi_t} P(\xi_{t+1}(x) = 0 | \xi_t(x) = 0) \\ &= \left(\frac{1}{n}\right)^{|\xi|} \end{aligned}$$

□

It is natural, and intuitive, that the state maintenance probability is dependent upon the total size of the density of infected cells (ie. the proportion of infected cells in the state of the automaton). This result also takes advantage of the strict assumptions which need to be relaxed for the full model of interest, **??**.

This probability is easy to derive in the case of a steady, absorbing, state, treated below. However, it is not immediately clear with what probability it actually enters into an absorbing state. The stricter assumptions allow us to make profound discoveries into this matter. Nonetheless, the treatment of the full model is planned for further analysis in the future; currently it lies not within reach.

4.4.4 *absorbing state phase transitions*

The idea of *state maintenance*, of course, is a parallel concept if the state of interest is an *absorbing state*. However, in that case, we have a vacuous result.

In all other initial conditions, we are interested in whether at infinite time there is an invariant probability measure.

Under the restricted stochastic process, we prevent the introduction of production source, so no outside damage (or “infection") is occurring. Under this strong assumption, the system has a predictable, guaranteed outcome. The following theorem proves the unique infinite-time state.

Theorem 1. *Recall that* $p_t(\xi) = P\{\xi_t \neq \emptyset | \xi_0 = \xi)$. *For a lattice with finite cells, and periodic boundaries, we have*

$$\lim_{t\to\infty} p_t(\xi) = 0$$

.

Proof. Without loss of generality, we can assume that there are $2N + 1$ cells, (from $-N$ to N), and, w.l.o.g., $m = 1$, $n = 1$, and $\alpha = 1$ for the transition probabilities.

Thus, we have

$$\text{P(cell i is occupied at time } T = t) = \frac{k}{1+k},$$

where k is the number of occupied cells among cells $i-1$, i, and $i+1$ at time $T = t-1$ and ξ_t represents the cells **distribution state** at time $T = t$. We discuss the possible states for ξ_t.

The possible number of occupied states is an integer value between 0 and $2N+1$. When the number of occupied cells is $0, 1, 2N$, or $2N+1$, we have only one possible distribution state for each of the four cases. We use $\mathcal{S}_0$,$\mathcal{S}_1$, $\mathcal{S}_N$, and $\mathcal{S}_{N+1}$ to denote these states, respectively.

For any other number of occupied cells (from 2 to 2N-1), we have multiple distribution states. For example, for 2 occupied cells we have N different distribution states possible (corresponding to the number of empty cells between these two occupied cells). We use $\mathcal{S}_{2,i}$ to denote each of these states (i from 0 to $N-1$). Thus, ξ_t can take the states: $\mathcal{S}_0$,$\mathcal{S}_1$, $\mathcal{S}_{2,i}$, $\mathcal{S}_{2,i+1}$, $\ldots$, $\mathcal{S}_{n,i}$, $\ldots$, $\mathcal{S}_N$, $\mathcal{S}_{N+1}$. We show that among these finite states, the only absorbing state is $\mathcal{S}_0$. For state $\mathcal{S}_0$ (no occupied cell), it is vacuously true that $P(\xi_{t+1} = \mathcal{S}_0 | \xi_t = \mathcal{S}_0) = 1$. For all states other than $\mathcal{S}_0$, we consider the transition of a single cell. For any cell x_i, we have

$$\begin{aligned} P(x_i \text{ empty at } T = t+1) &\geq P(x_i \text{ empty at } T = t+1 | x_{i-1}, x_i, x_{i+1} \text{ all occupied at } T = t) \\ &= 1 - \frac{3}{1+3} = \frac{1}{4} \end{aligned}$$

Thus, given any state ξ_t at time $T = t$, we have

$$\begin{aligned} P(\xi_{t+1} = \mathcal{S}_0 | \xi_t) &\geq P(\xi_{t+1} = \mathcal{S}_0 | \xi_t = \mathcal{S}_{2N+1}) \\ &= \prod_{i=-N}^{N} P(x_i \text{ empty at } t+1 | \xi_t = \mathcal{S}_{2N+1}) \\ &= \left(\frac{1}{4}\right)^{2N+1} \end{aligned}$$

Therefore, for all states other than $\mathcal{S}_0$, there is a positive probability of entering state $\mathcal{S}_0$, i.e., a probability not less than $\left(\frac{1}{4}\right)^{2N+1}$. Therefore, all other states are *not* absorbing.

We define a new state, $\mathcal{S}_{0'}$, which represents any state other than $\mathcal{S}_0$. It immediately follows from above that $\forall \mathcal{S}_i \in \mathcal{S}_{0'}$

$$P(\xi_{t+1} = \mathcal{S}_0 | \xi_t = \mathcal{S}_i) \geq \left(\frac{1}{4}\right)^{2N+1}$$

.

We immediately extend this transition probability to a matrix form for these two states, $\mathcal{S}_0$ and $\mathcal{S}_{0'}$:

$$P = \begin{array}{c} \\ \mathcal{S}_0 \\ \mathcal{S}_{0'} \end{array} \begin{array}{c} \begin{array}{cc} \mathcal{S}_0 & \mathcal{S}_{0'} \end{array} \\ \begin{pmatrix} 1 & 0 \\ q & 1-q \end{pmatrix} \end{array},$$

where $q \geq \left(\frac{1}{4}\right)^{2N+1} > 0$. This implies $1 - q \leq 1 - \left(\frac{1}{4}\right)^{2N+1}$. We then have

$$P^n = \begin{array}{c} \\ \mathcal{S}_0 \\ \mathcal{S}_{0'} \end{array} \begin{array}{c} \begin{array}{cc} \mathcal{S}_0 & \mathcal{S}_{0'} \end{array} \\ \begin{pmatrix} 1 & 0 \\ 1-(1-q)^n & (1-q)^n \end{pmatrix} \end{array}.$$

So, then we have

$$P(\xi_{t+n} = \mathcal{S}_{0'} | \xi_t = \mathcal{S}_{0'}) = (1-q)^n.$$

Since $1 - q \leq 1 - (\frac{1}{4})^{2N+1} < 1$, we see that

$$\lim_{n \to \infty} (1-q)^n = 0.$$

Thus

$$P(\xi_\infty = \mathcal{S}_{0'}|\xi_t = \mathcal{S}_{0'}) = 0,$$

which implies

$$P(\xi_\infty = \mathcal{S}_0|\xi_t = \mathcal{S}_{0'}) = 1.$$

We immediately have that

$$P(\xi_\infty = \mathcal{S}_0|\xi_t) = 1.$$

This means that with enough time, and considering any distribution state of the process at any time, the process eventually dies out. □

I believe that this simple proof is one straight out of *the book* :-)

4.5 Numeric Investigation of Space and Time Properties of the *GPCA*

Motivated by the results from numerical investigations of the models in Sec.4.1.1,4.1.2,4.1.3, we perform Monte Carlo numerical investigations of the long-term space and time properties of the proposed stochastic process; the *GPCA* model. We seek to identify and characterize many facets of its intricate behaviors, in both time and space. Of chief importance are the model's phase transitions: the transitions from parameterizations that provide ergotic long-term behaviors in state to parameterizations that induce absorbing, invariant state measures. To characterize these parameter regimes requires identifying and characterizing critical parameter values, along with understanding the invariant measures that result from the differing regimes.

4.5.1 Critical Behavior: State Phase Transitions

There are two clear absorbing states of our stochastic process: $\mathbf{1}$, where $\xi(x_n) = 1\ \forall n$, and $\mathbf{0}$, where $\xi(x_n) = 0\ \forall n$, if no outside damage is introduced into the system. As was shown in the simpler case for **??**, there exists a critical ϵ^* such that for $\epsilon > \epsilon^*$ the only invariant measure of Stavskaya's PCA is $\delta_{\mathbf{1}}$, the measure concentrated at $\mathbf{1}$, and that for $\epsilon < \epsilon^*$ the invariant measures are translation-invariant convex combinations of the form illustrated in Sec. **??**.

Early bounds on this critical value, ϵ^*, were provided as $0.09 < \epsilon^* < 0.323$, see [] and the upper bound was eventually confirmed but never improved, while the lower bound simply never received any reassessment for nearly 50 years. Finally, Mendonca [] was able to fully characterize this parameter. It should be noted that lower bounds on critical values of *interacting particle systems* are notoriously difficult to obtain.

We seek to similarly characterize the state phase transitions of the *GPCA* within the various parameter regimes pertinent to the physical applications and desired modeling.

Stationary Density of Inactive Cells: ***Absorbing state phase transitions***

The Monte Carlo simulations of the *GPCA* were run as follows. For a given m, as defined in Eq.3.5, the PCA is initialized with each $\xi_0(x_n) = 1$, drawn independently with probability π, for $0 \leq n \leq L$. The stationary state quantities are then measured after relaxing the system through $100L$ steps of the Monte Carlo chain, described in Alg. 3. Each Monte Carlo step is considered a synchronous update of all L cells of the automaton. Periodic boundary conditions are assumed.

The density of active cells is defined simply as in the following,

$$\rho_L = L^{-1} \sum_n \xi(x_n). \tag{4.6}$$

This amount of system relaxation was found to be sufficient for our purposes through empirical observation based on multiple runs of different parameter regime values. We run this full MCMC simulation thousands of times to be able to average values of each stochastic realization.

Figure 4.1 displays the density profile, ρ_L, of active cells for an automaton of $L = 160,000$ cells in the stationary state. Note that we plot the density profile of *inactive* cells, $1 - \rho_L$.

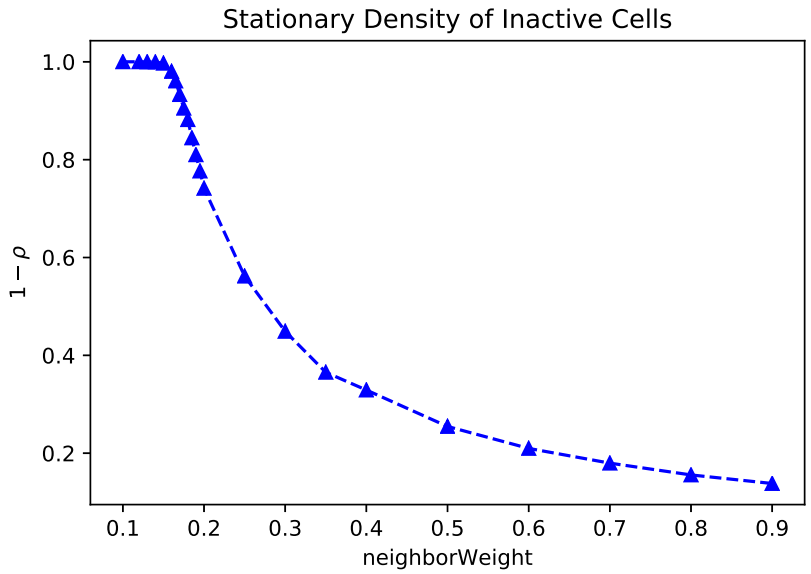

Figure 4.1: Stationary density of inactive cells, averaged over 10,000 realizations, in an automaton of $L = 160,000$ cells. The critical point is easy to, roughly, observe, as the state transitions from ergodicity to absorbing state.

We can numerically find the transition probability for which the steady state leaves the absorbing state of $\rho = 0$. Below, we zoom in on the plot at the critical parameter value:

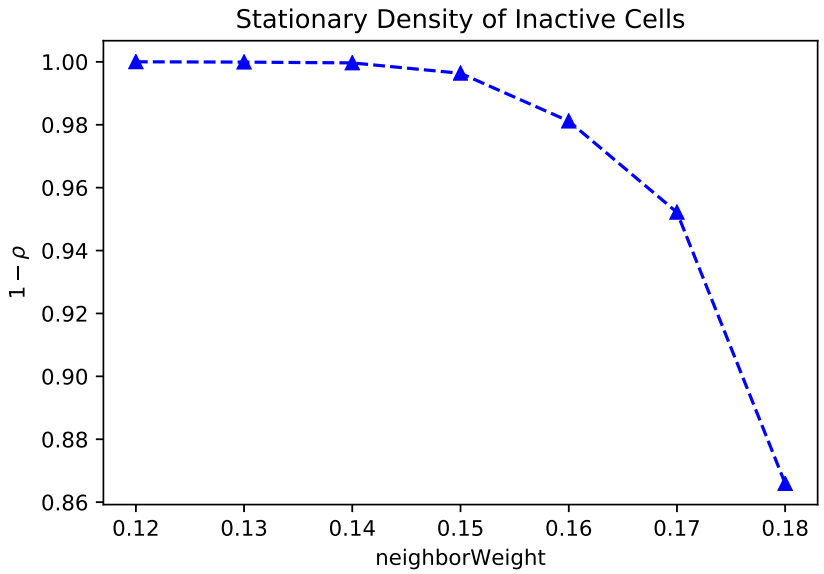

Figure 4.2: Stationary density of inactive cells, averaged over 10,000 realizations, in an automaton of $L = 160,000$ cells. The critical point is easy to observe, roughly, as the state transitions from ergodicity to absorbing state.

Lattice Size

We also investigate the impact that the domain of the automaton has on the critical values of phase transition. As noted earlier, we perform all simulations using *periodic boundary conditions* and in the following comparison, we maintain an equal proportion of initial damage distribution.

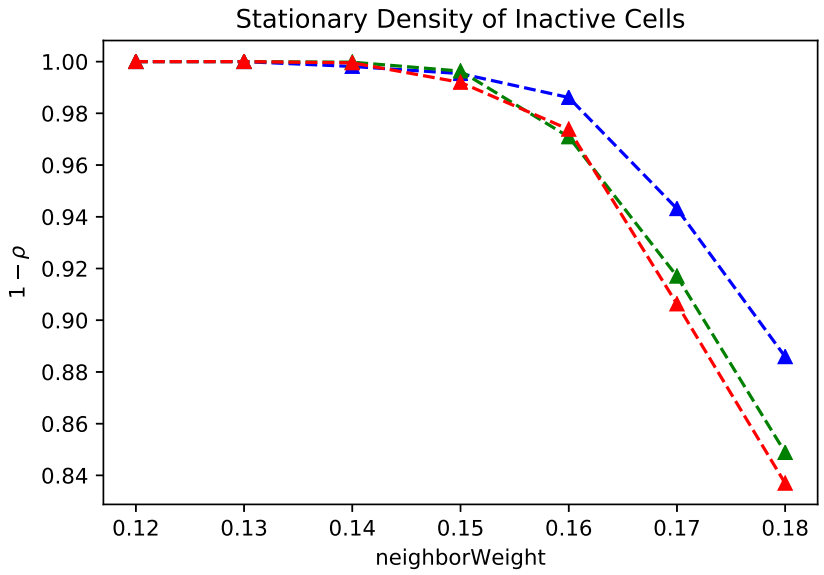

Figure 4.3: Stationary density of inactive cells, averaged over 10,000 realizations. Blue: an automaton of $L = 40,000$ cells; Green: $L = 90,000$; Red: $L = 160,000$. The automaton is initially damaged with 10% infected cells, distributed as detail above.

As we can see, the size of the domain on which the automaton exists does not seem to affect the critical value of the process, but rather the rate at which the steady states depend.

4.5.2 Rates of Convergence to Steady States

One of the most fundamental questions that needs to be answered if we want to actually apply the model to materials engineering is *how long does it take to reach these steady states?* We need to keep in mind that we are constantly trying to investigate whether our *GPCA* has *smooth* properties to feed to a partial differential equation. And if it does have smooth properties, do we need to complete millions of simulations? Or can we do better?

In the following, we perform analyses to examen the rates of convergence to our steady states, under varying parameter regimes, domain sizes, and initial conditions. We carry out the experiments in a similar fashion as those previously outlined: for a given m, as defined in Eq.3.5, the PCA is initialized with each $\xi_0(x_n) = 1$, drawn independently with probability π, for $0 \leq n \leq L$. The stationary state quantities are then measured after relaxing the system through $1000L$ steps of the Monte Carlo chain, described in Alg. 3. Each Monte Carlo step is considered a synchronous update of all L cells of the automaton. Periodic boundary conditions are assumed.

Steady State Convergence, under $m > \epsilon^*$

We initially use values of the transition probabilities that are strictly greater than the critical value, $m > \epsilon^*$. As seen in Figure 4.1, the value of the neighborWeight, $m = 0.2$, is suitably large enough to put us into the class of parameter values that assumes a steady state distribution of the *density of active cells*. We relax this system through $1000L$ steps of the Monte Carlo chain, for multiple realizations, and plot the average density of the active cells:

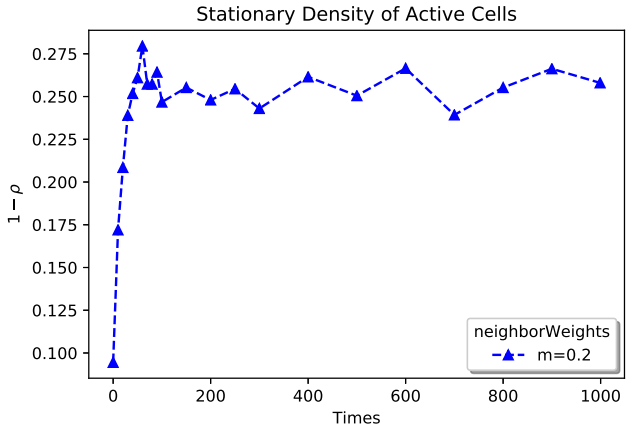

Figure 4.4: Stationary density of *active cells*, averaged over 10,000 realizations. The system is relaxed through $1000L$ steps of the MC chain. The automaton is initially damaged with 10% infected cells.

We can see that the steady state is actually achieved early on in the chain; the need to relax the system for this many time steps is simply not necessary. By simply zooming in on the beginning of the chain this becomes more clear:

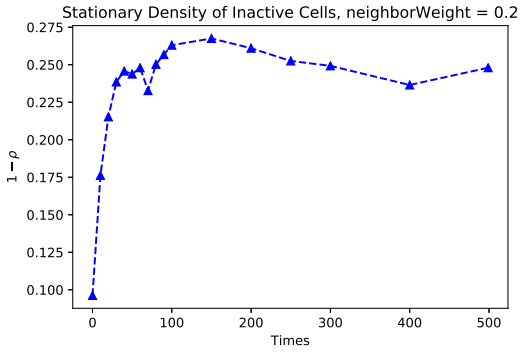

Figure 4.5: The system is relaxed through $500L$ steps of the MC chain.

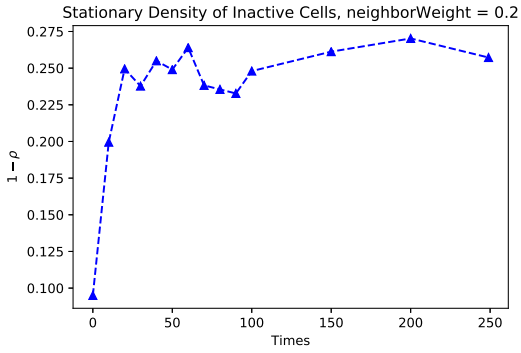

Figure 4.6: The system is relaxed through $250L$ steps of the MC chain.

Convergence Under Varying Transition Probabilities

This fast rate of convergence to the steady state is not only observed for one parameter value. We seem to observe this same rate of convergence under the entire class of parameters that achieve a steady state.

The following plot illustrates the rates of convergence for varying transition probabilities. Since it was observed that the chains converge quite quickly, we need only relax the system through $250L$ steps of the Monte Carlo chain.

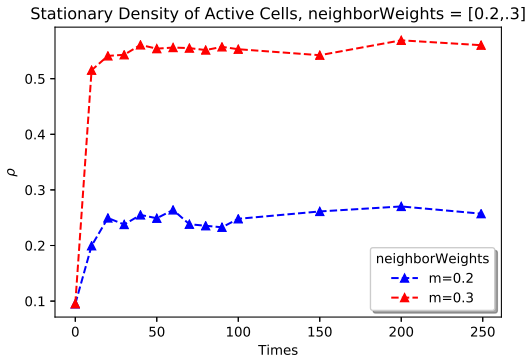

Figure 4.7: Stationary density of *active cells*, averaged over 10,000 realizations. The system is relaxed through $250L$ steps of the MC chain. The automaton is initially damaged with 10% infected cells.

In fact, we see that these changes converge simply within the first few dozen time steps:

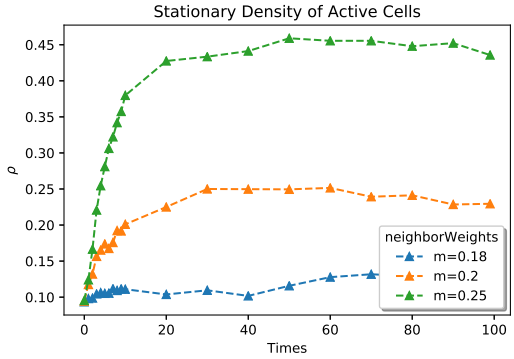

Figure 4.8: Stationary density of *active cells*, averaged over 10,000 realizations. The system is relaxed through $100L$ steps of the MC chain. The automaton is initially damaged with 10% infected cells.

Steady State Convergence, under $m < \epsilon^*$

For the class of parameter values where the transition probabilities are actually *less* than the critical value, $m < \epsilon^*$, we see that convergence can actually occur at a slightly slower rate.

In the following plots, we see that transition probability values *close to* the ϵ^* require closer to $250L$ steps to reach their steady states:

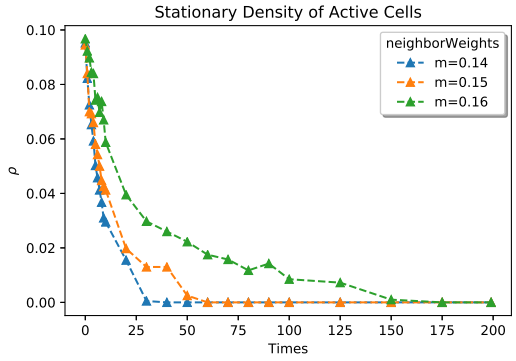

Figure 4.9: Stationary density of *active cells*, averaged over 10,000 realizations. The system is relaxed through $200L$ steps of the MC chain. The automaton is initially damaged with 10% infected cells.

The transition probabilities that are *further away* from ϵ^*, nonetheless, converge to $\mathbb{0}$ quickly.

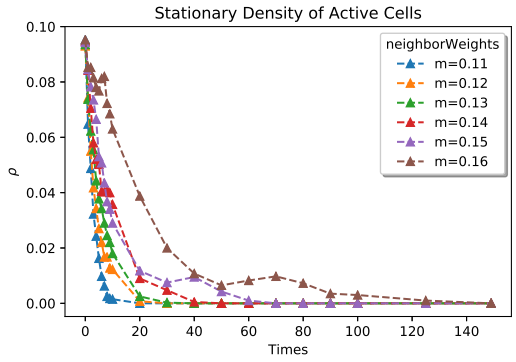

Figure 4.10: Stationary density of *active cells*, averaged over 10,000 realizations. The system is relaxed through $150L$ steps of the MC chain. The automaton is initially damaged with 10% infected cells.

Steady State Convergence for Transition Probabilities *close to* ϵ^*

The story definitely changes when we look at the parameter values on either side of the critical value, ϵ^*. Nonetheless, each of these automaton chains reach their steady state within the $100L$ Monte Carlo time steps.

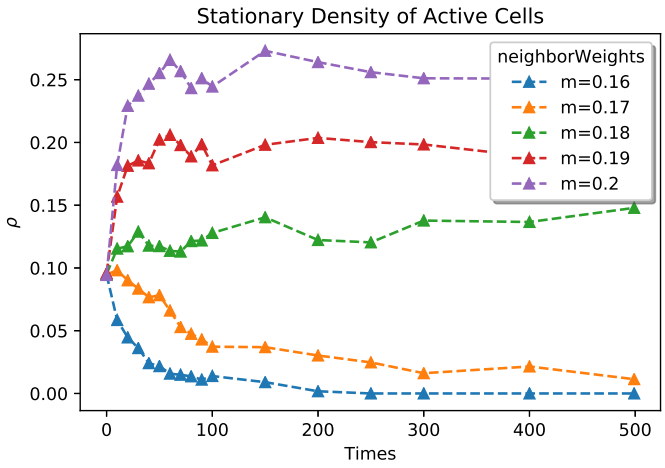

Figure 4.11: Under Varying Transition Probabilities *close* to ϵ^*. $500L$ MC steps.

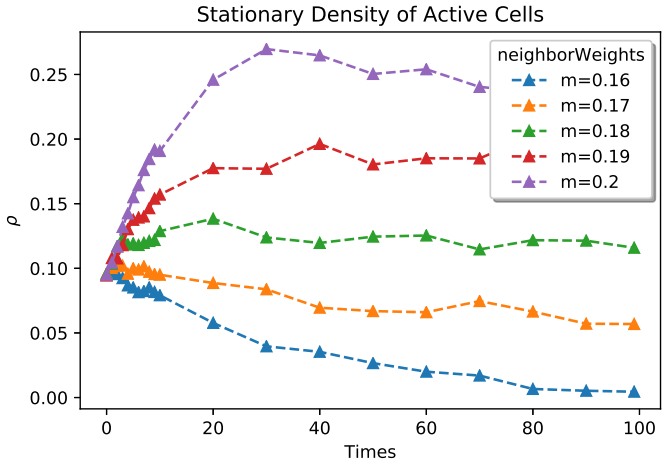

Figure 4.12: Under Varying Transition Probabilities *close* to ϵ^*. $100L$ MC steps.

Convergence Under Varying Domain Sizes

A natural question to ask pertaining to the convergence rates of these stochastic processes is whether the domain has an interaction effect on the rates. Since we are using *periodic boundary conditions*, and the finite automatons are reaching steady state, we investigate whether the size of the domain changes how many steps the chain requires to reach its steady state.

Figure 4.13, below, illustrates how the chains converge in *essentially* the same rates.

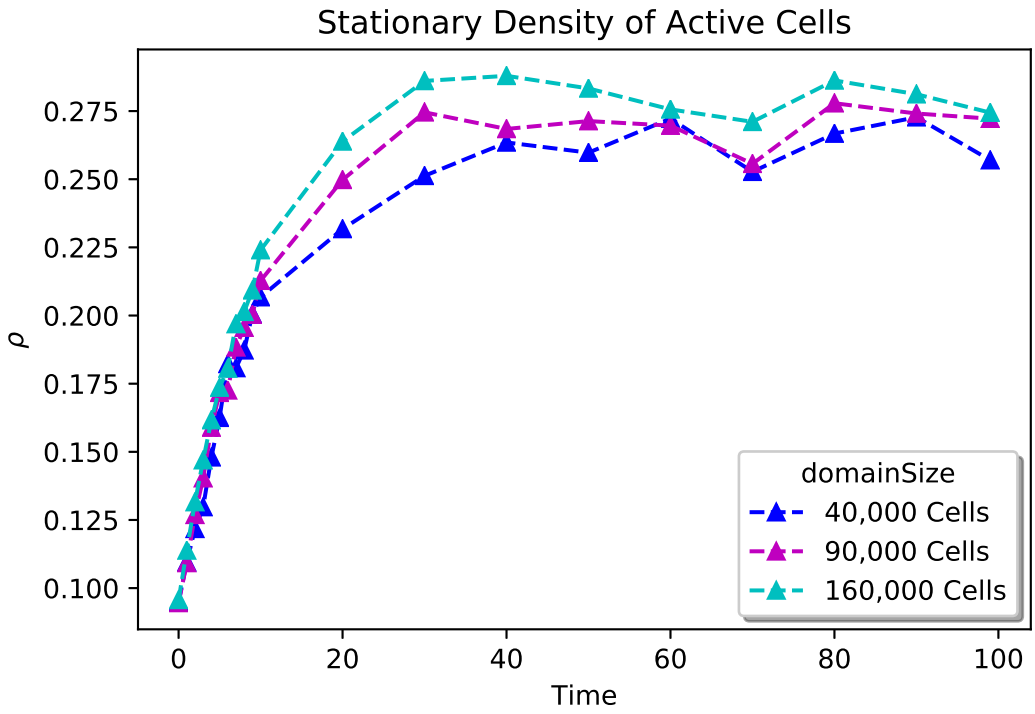

Figure 4.13: Convergence to Steady State under Varying Domain Sizes. The automaton is initially damaged with 10% infected cells.

Convergence Under Varying Initial Damage Domains

Another natural question is whether the initial damage to the domain affects the rate of converge, or even the steady state density? As before, for a given transition probability, m, the PCA is initialized with each $\xi_0(x_n) = 1$, drawn independently with probability π, for $0 \leq n \leq L$. We now vary these initial cell infection probabilities, π. The stationary state quantities are then measured after relaxing the system through $1000L$ steps of the Monte Carlo chain, described in Alg. 3. Each Monte Carlo step is considered a synchronous update of all L cells of the automaton. Periodic boundary conditions are assumed.

Figure **??** illustrates the convergence to the same steady state density, at the same rate, when the initial damage probabilities, π, are *less* than the steady state density rate, ρ.

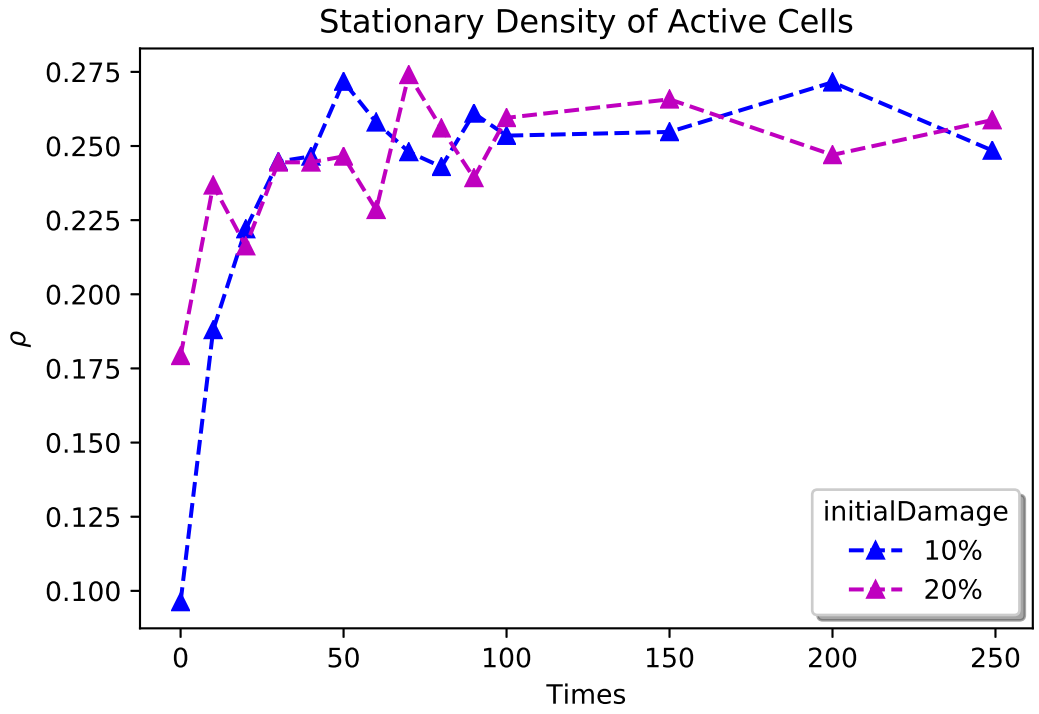

Figure 4.14: Convergence to Steady State under Varying Initial Damage Domains. The automaton is initially damaged with 10% and 20% infected cells.

We can also easily see that when the initial damage probabilities, π, are actually *more* than the steady state density rate, ρ, we have that the automaton *still converges* to the *same* steady state density, and at the *same* rate.

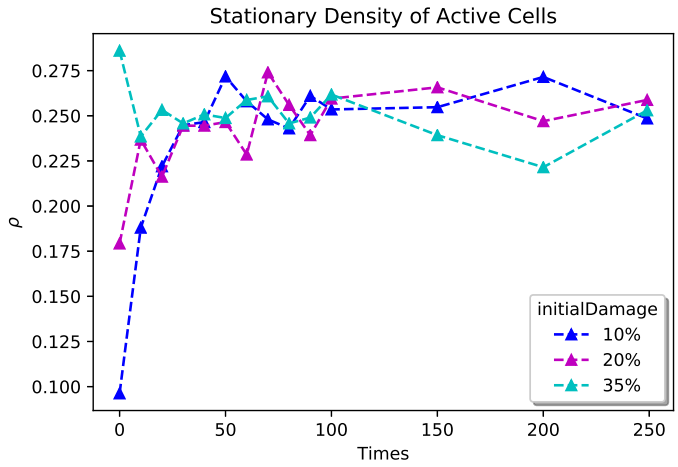

Figure 4.15: The domain is initially damaged with $\pi = .35 > \rho$.

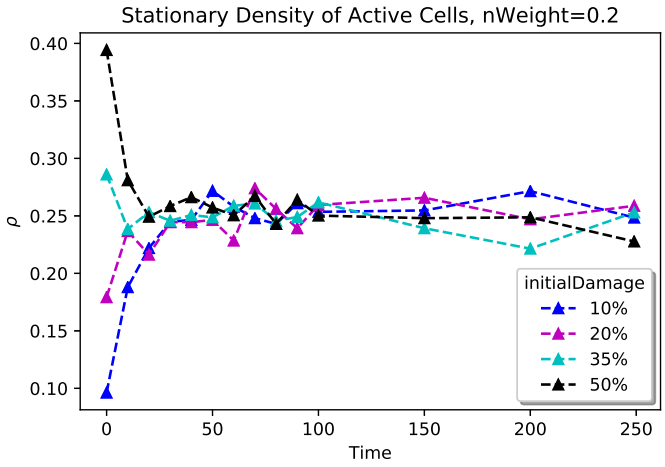

Figure 4.16: The domain is initially damaged with $\pi = .50 > \rho$.

4.5.3 Rates of Dispersion: *Infection rates*

The Monte Carlo simulations of the *GPCA* were run as follows. For a given m, as defined in Eq.3.5, the PCA is initialized with a single, circular cluster of active cells: each $\xi_0(x_n) = 1$, within a central, circular region of the automaton. The stationary state quantities are then measured after relaxing the system through $100L$ steps of the Monte Carlo chain, described in Alg. 3. Each Monte Carlo step is considered a synchronous update of all L cells of the automaton. Periodic boundary conditions are observed here within.

Figure 4.17 displays a single realization of an automaton of $L = 3,600$ cells.

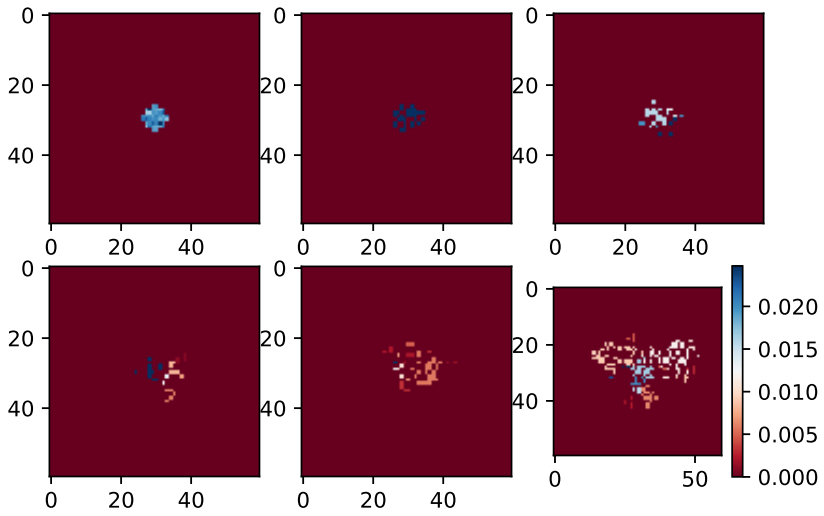

Figure 4.17: Realization of a single stochastic run; no healing or ongoing production of source.

We run this full MCMC simulation thousands of times to be able to average the automatons of each stochastic realization.

Figure 4.18 displays the density profile of active cells for an automaton of $L = 3,600$ cells.

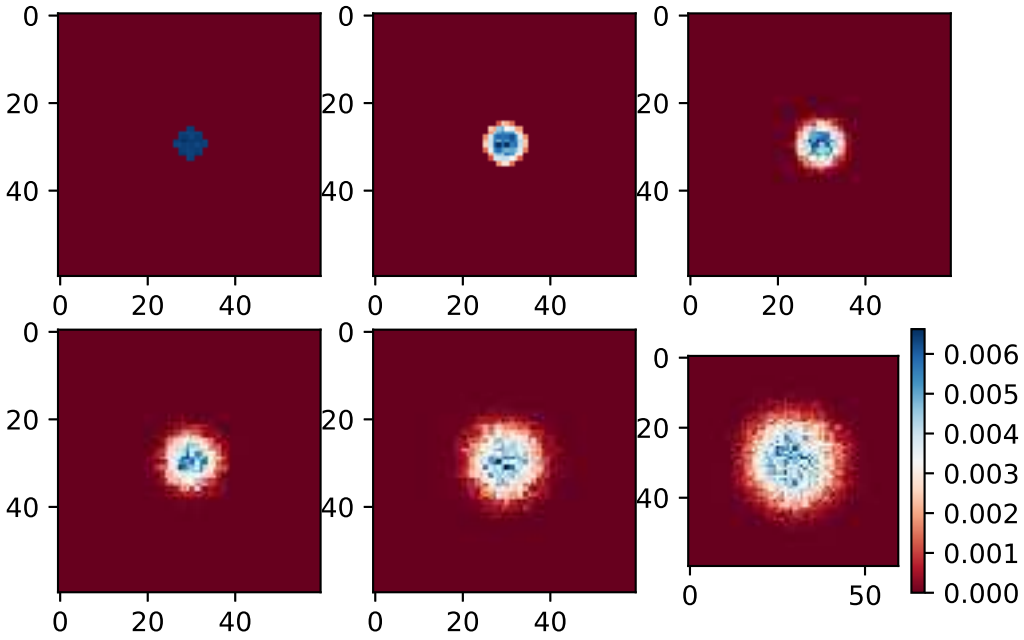

Figure 4.18: Average state of automaton, from 10,000 realizations, with neighborWeight = 0.4.

Figure 4.19 displays the marginal density profile of active cells for the above automaton.

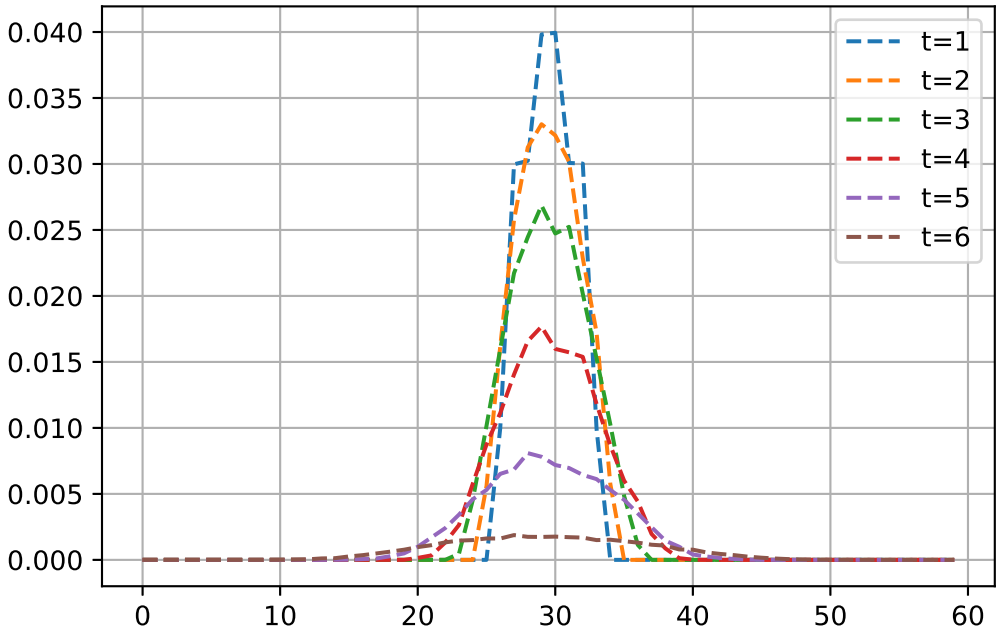

Figure 4.19: Average state of automaton, from 10,000 realizations, with neighborWeight = 0.4.

We run this full MCMC simulation thousands of times to be able to average the automatons of each stochastic realization.

Figure 4.20 displays the density profile of active cells for an automaton of $L = 3,600$ cells.

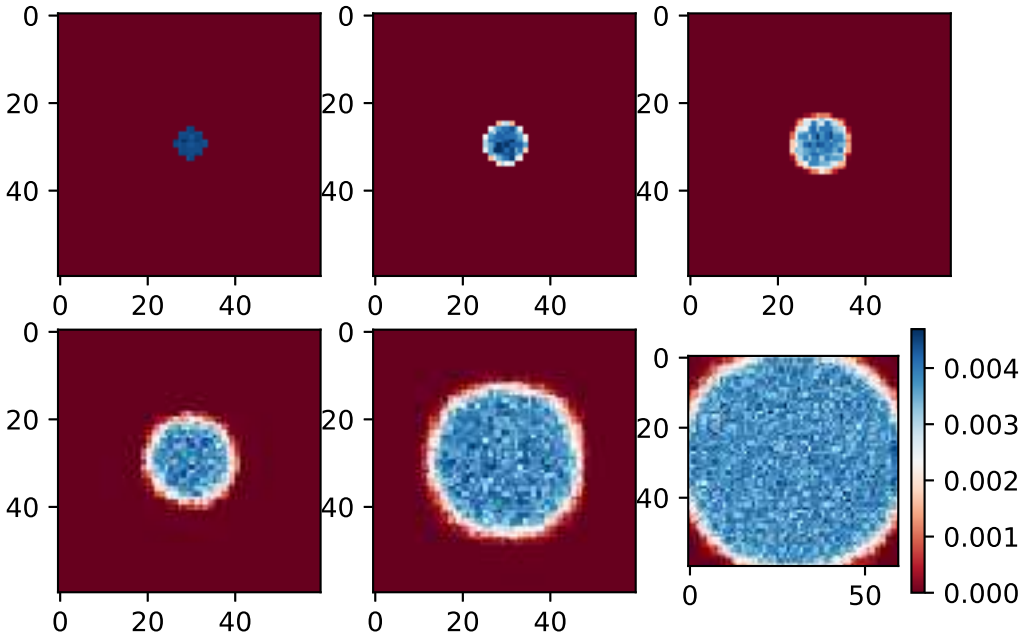

Figure 4.20: Average state of automaton, from 10,000 realizations, with neighborWeight = 0.6.

Figure 4.21 displays the marginal density profile of active cells for the above automaton.

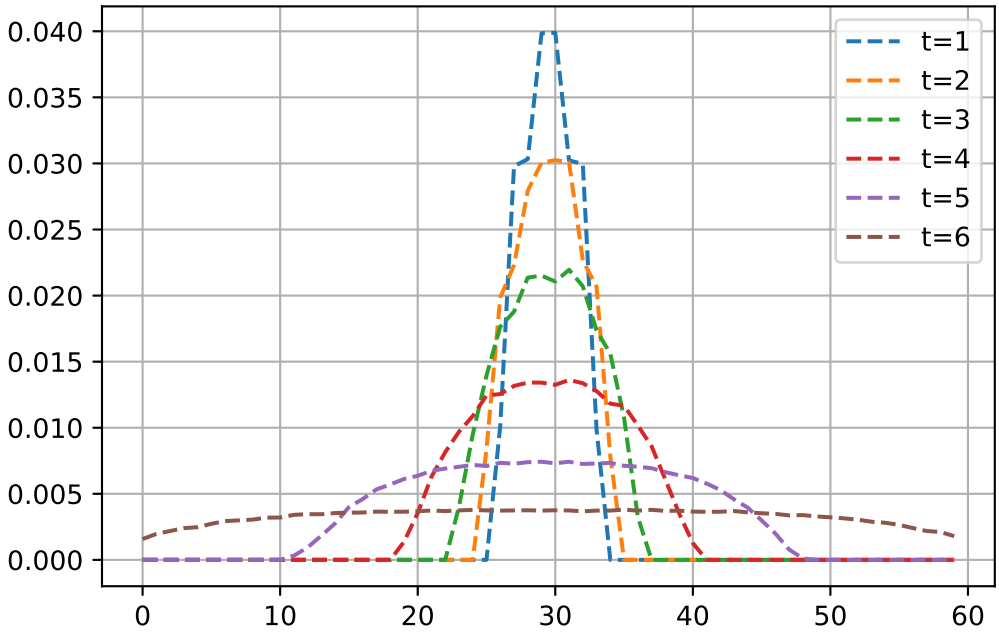

Figure 4.21: Average state of automaton, from 10,000 realizations, with neighborWeight = 0.6.

It is easy to see by comparison that as m *increases* (neighborWeight) the tendency for a cell to maintain its infection increases, causing a change in the dynamics of dispersion. While *the rate* at which neighboring cells become infected *increases*, we also see the tendency for a cell to maintain state *increases*. This causes a *sharper* **interfacial boundary** on a cluster of active cells. The steady state patterns changes, as well as the rates at which they are reached.

L^2 Convergence with Expected State

We also investigate how an individual process diverges from its expectation. Does an individual realization behave like -or have similar statistical properties to- its expected value?

Using the average densities derived above, we implement the following MCMC conditions: For a given m, as defined in Eq.3.5, the PCA is initialized with a single, circular cluster of active cells: each $\xi_0(x_n) = 1$, within a central, circular region of the automaton. The stationary state quantities are then measured after relaxing the system through $100L$ steps of the Monte Carlo chain, described in Alg. 3. Each Monte Carlo step is considered a synchronous update of all L cells of the automaton. Periodic boundary conditions are observed.

We take the L^2 distance of each individual realization of the process with this average density. The following plot illustrates the dynamics of these individual process realizations:

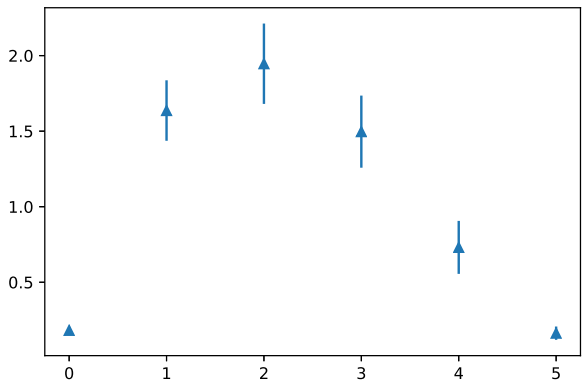

Figure 4.22: L^2 distance of an individual realization from the average state of automaton, across time, from 10,000 realizations, with neighborWeight = 0.6.

We see that the individual realization diverges from the average at first, but after further relaxation, the process enters a steady state and -once again- “looks like" its average, in terms of statistical properties (in space and fractional density).

We can see that individual processes vary greatly from realization to realization, even under the same initial conditions. The following plot illustrates this same L^2 distance from the average, for many realizations, at one time point.

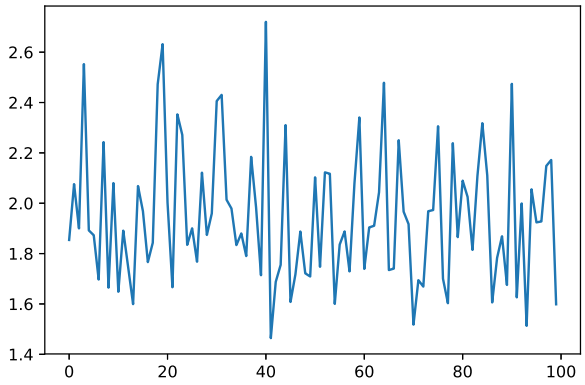

Figure 4.23: L^2 distance of individual realizations from the average state of automaton from one time point, from 10,000 realizations, with neighborWeight = 0.6.

Chapter 5

Complete Statistical Upscaling of Stochastic Forcing in Multiscale, Multiphysics Modeling

As has been shown, modeling nuclear radiation damage necessarily implies multiple scales in both time and space, where molecular-level models have drastically different assumptions and phenomena than continuum-level models. In this chapter, we propose to explicitly couple these multiple scales of the microstructure damage to engineering scales through a statistical upscaling of models using stochastic processes that take atomic collision cascade simulations, from physical first principles of vacancy generation in the micro scale, upscaled into a *Generalized Probabilistic Cellular Automata* model at the meso scale, through to phase field models of void nucleation and growth in irradiated materials, in both space and time. The effect of radiation source, through these stochastic processes, on the dynamics of void nucleation and growth is illustrated.

5.1 Upscaling Micro to Meso

We build upon the construction in Chapter 2, specifically referring to Section 2.3, to begin our modeling from the physical first principles at the molecular scale, to explicitly drive our *Cellular Automata* models at the mesoscale. The result is a multiscale model combining the very fast, molecular scale collision cascades generating atomic vacancies with the complex, stochastic dynamics of much larger damage regions evolving over longer time scales. In concrete terms, the spatial and temporal scales of a reasonably accurate meso model are much larger than the spatial and temporal scales of individual molecules; ie. collision cascades. In this section, we describe a statistical approach to upscaling the molecular scale events to be included as this driving term for the mesoscale, stochastic process model.

5.1.1 Simulating Collision Cascades on the Molecular Scale

Following the simulations of Binary Collision Cascade Approximations, detailed in Section 2.3, we again use the TRIM [5] codes to generate binary collision cascades, which have become the defacto manner of modeling the radiation damage introduced by individual collision cascades [2–4]. The reader may refer to a typical simulation of a collision cascade as presented in Fig. 2.1.

Computing the source term involves simulating many, e.g. 100,000, collision cascades. For cascade simulation j, we record the number of atoms displaced from the lattice d_j, and the 3-dimensional range projections (X_j, Y_j, Z_j), measured in angstroms, that coarsely quantify the size of the cascade. As in the previous simulations, these are all the specific characteristics of simulated collision cascades used in the statistical upscaling.

Recall that a collision cascade results in a molecular scale region in the material consisting of atoms that are displaced from their lattice positions as well as interstitial molecules isolated in vacancies, see Fig. 2.1. The Cahn-Hilliard model is not derived at a scale that represents the dynamics of individual molecules or the geometric complexities of collision cascades. Rather, the descriptive variables are densities of atom types, e.g., vacancy, interstitial, or matrix, over much larger nominal cells of a material. The first step in the model is to build density functions associated with each collision cascade that quantify the number of displaced and interstitial atoms respectively in the region of the cascade.

Representing a Collision Cascade as a Density

We represent the displaced vacancy and interstitial molecules involved in each collision cascade as densities.

As before, we define a volume for cascade j as,

$$v_j = \frac{4}{3}\pi\, X_j Y_j Z_j \text{ ang}^3,$$

where recall that X_j, Y_j, and Z_j are the range projections of cascade j. To characterize the number of molecules affected by a collision cascade, we define the non-dimensional **Cascade Fractional**

Density (cfd) for each cascade j:

$$\bar{c}_j = (\text{cfd})_j = \frac{d_j}{v_j \rho},$$

where ρ is the atom density of the material in units of atoms/ang^3. For pure copper,

$$\rho = \frac{4}{a^3} = \frac{4}{3.615^3} \approx \frac{4}{47.24163} \frac{\text{atoms}}{\text{ang}^3}.$$

The quantities d_j, v_j, and $\bar{c}_j$ are the basic building blocks for the model we construct computed from SRIM simulations. Since the collision cascades computed using SRIM are stochastic, the quantities d_j, v_j, and $\bar{c}_j$ are random variables.

5.1.2 Molecular Damage Introduction Over Space

We incorporate the ongoing production of vacancies and interstitial molecules due to radiation into our concentration fields and models using the stochastic source terms $P_v^{casc}(\mathbf{x}, \mathbf{t})$ and $P_i^{casc}(\mathbf{x}, \mathbf{t})$, of Equation 2.2, which randomly introduce localized increases in point defects in our concentration fields which correspond to our spatially-resolved ensembles of vacancies and interstitials resulting from displacement cascades.

In contrast to the modeling in Chapter 2, we now define $P_v^{casc}(\mathbf{x}, \mathbf{t})$ and $P_i^{casc}(\mathbf{x}, \mathbf{t})$ as below,

$$P_v^{casc}(\mathbf{x}, \mathbf{t}) = (\text{cfd})_{\mathbf{j}} \delta(\mathbf{t} - \mathbf{t_c}) \mathbb{1}_{\mathbf{x_c}} \tag{5.1}$$

$$P_i^{casc}(\mathbf{x}, \mathbf{t}) = (\text{cfd})_{\mathbf{j}} \xi_{\mathbf{b}}(\mathbf{x_c}) \delta(\mathbf{t} - \mathbf{t_c}) \mathbb{1}_{\mathbf{x_c}} \tag{5.2}$$

where x_c is the center of a cascade occurring at time t_c. Further, ξ_b, in equation 2.10, is a factor introduced to account for production bias in the number of interstitials produced during the cascade damage.

The binary collision cascades assume perfect material for each event so that, at it's most fundamental, we break up our mesoscale grid, described in detail below, into a grid of cells of sufficiently

small size such that we may ensure that this assumption holds and that these point defects occur in a single cell and independently of one another. For the simulations shown in the rest of this chapter, we chose to use cell sizes of 500 angstrom^2. This was a material-specific choice, as a binary collision cascade simulation almost surely is contained within this chosen size, for the material Cu with the initial energies that we simulated. Grid size may be appropriately chosen to satisfy the above assumptions according to the specific material and specific ion/energy simulated.

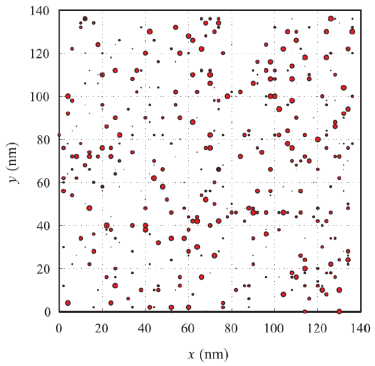

Figure 5.1: Example of random cascades in a unit area over period T.

We then model the displacement collision cascades as discrete point process events and distribute their occurrence uniformly over space, Fig. 2.9. At each cascade occurrence event, we uniformly draw spatial coordinates to distribute the cascade event. We also assume that no cascade can occur within an already-damaged cell, and thus do not distribute over already damaged cells.

5.1.3 Molecular Damage Introduction over Time

We will define a "waiting time" to be the time between an arrival event of a collision cascade occurrence and the immediate subsequent cascade arrival. We assume that each introduction of irradiation damage to our domain is independent of one another. Therefore, the arrival time of each cascade event is independent of one another. Since we assume that each cascade arrival time is independent, continuous, and memory-less, we have a natural model to impose on the waiting times: exponential. We can then generate a new collision cascade at each time step, where the

length of each time step is random and is distributed according to this prescribed exponential distribution. See Section 2.3 for more details.

Here, we specify the **arrival rate**, λ, as $\lambda = \bar{R_c}^{-1}$, $\forall\ i = 1, 2, ...$ where $\bar{R_c}$ is the average rate of collision occurrence prescribed by the physics of the material. Given the exponential model for waiting times, we can use it to simulate the process by generating new collisions at arrival times drawn from the exponential distribution.

We define the following in our model:

$$t_i = \text{the time of occurrence of the } i^{th} \text{ cascade}$$

and thus have that

$$t_{i+1} - t_i \sim exp(\lambda) \tag{5.3}$$

where $\lambda = \bar{R_c}^{-1}$, $\forall\ i = 1, 2, ...$ and $\bar{R_c}$ is the average rate of collision occurrence prescribed by the physics of the material.

5.1.4 Simulating over PDE timestep

Since we impose exponential wait times between individual, independent cascade occurrences, we can exploit the fact that the number of cascade events over a larger time scale follows a Poisson Process.

We thusly have that if $N(T) =$ the number of occurrences until time T, then

$$N(t + \tau) - N(t) \sim Poisson(\tau\lambda), \quad \tau \geq 0 \tag{5.4}$$

where $N(t + \tau) - N(t)$ is independent of $N(r)$ for $0 \leq r \leq \tau$, and λ is defined in Equation 5.3.

This enables us to take advantage of the fact that the mesoscale evolution of material defects takes place at a much slower speed than the occurrence of individual cascades, which happen within fractions of nanoseconds. We do this through aggregation: drawing multiple cascade parameters from their distributions at each of the larger PDE solver time steps, aggregating the dam-

age in space from the micro to meso spatial scales. The Poisson Process ensures that the number of events within an interval is proportional to the length of the interval. The parameter of this Poisson Process is also directly determined by the parameter of the exponential arrival times.

5.2 Implementation of Mesoscale Model

5.2.1 Definition of Computational Domain at Mesoscale

Our first step in implementing this stochastic process model, at the meso scale, is to carefully, and explicitly define the computational domain on which to distribute the stochastic cascade occurrences. It is important to emphasize that this grid is explicitly defined firstly -and independently- of the PDE discretization. This mesoscale allows us to distribute the molecular-level cascade events into a statistically upscaled representation, density, that the engineering-scale PDE can then understand. This is now the scale at which the movement of damages will occur using the *Generalized Probabilistic Cellular Automata*.

We aggregate these independent cascade occurrences over both space and time, explained further below. This mesoscale grid has units expressed by the individual cascade cells defined in Sec. 2.3.5. Fig. 5.2 shows individual molecular cascades distributed across this mesoscale grid, as well as as the upscaled grid introduced as the forcing terms to the concentration fields, Equations 5.1 and 5.2.

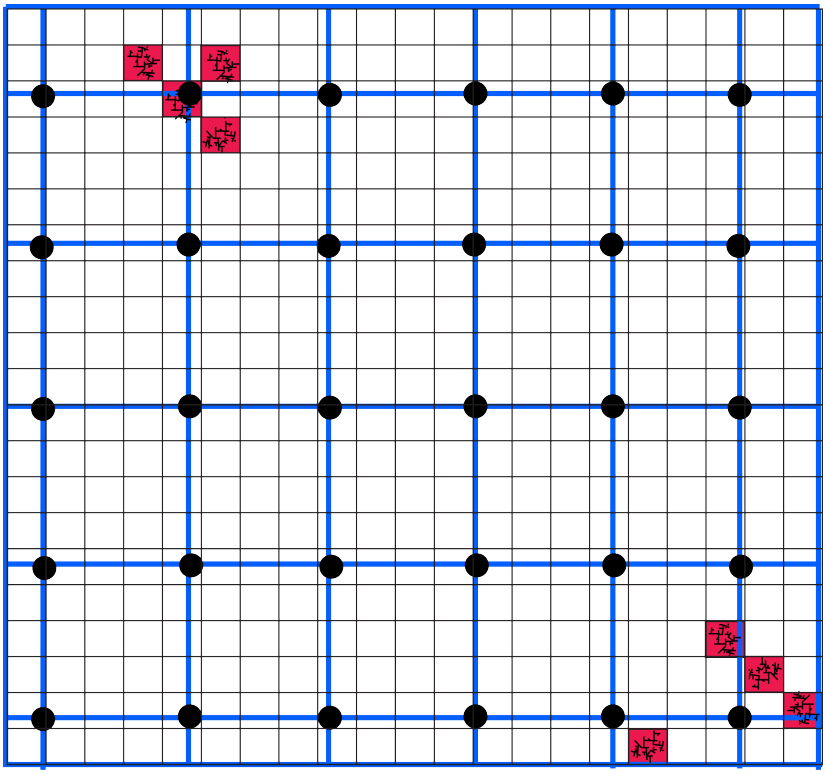

Figure 5.2: Mesoscale Computational Domain: *Cellular Automata*.

5.2.2 PDE Discretization

The second step in implementing the stochastic process model, at the meso scale, involves explicitly defining the ratio between the number of cascade cells in each cell of the PDE discretization grid. Since the PDE is a model of physical phenomena at the engineering scale, or continuum scale, the units are expressly defined. Likewise, the simulations performed at the micro scale are also expressly defined. This reconciliation at the computational domain of the *cellular automata* makes explicit the scales at which each model is being simulated.

We illustrate the ratio between these two grid sizes by changing Fig. 5.2 to include vastly more cascade cells per PDE discretization cell in Fig. 5.3.

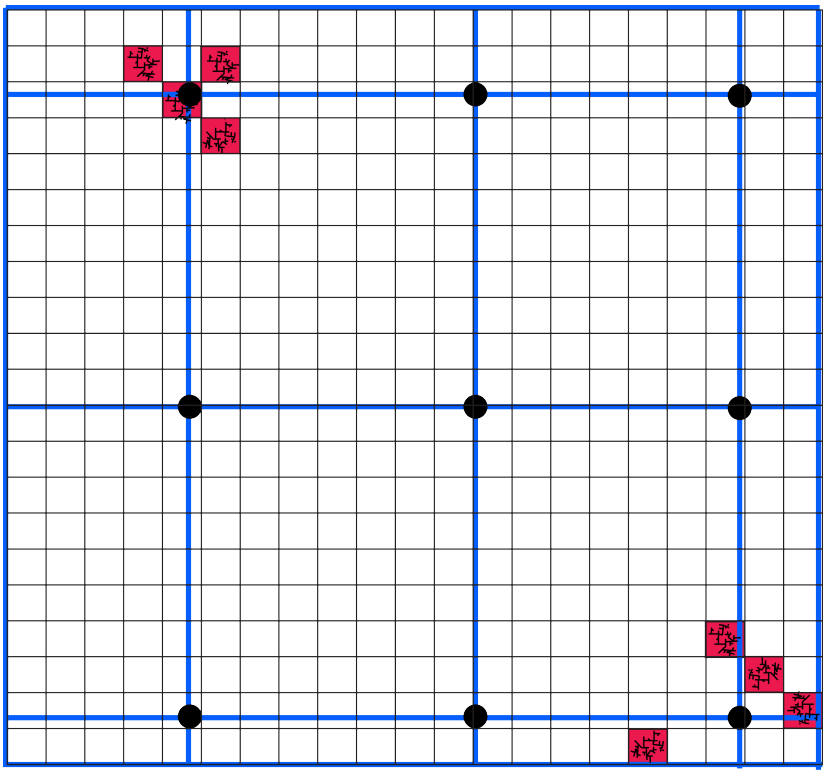

Figure 5.3: Mesoscale Computational Domain: Micro to Meso to Macro.

5.2.3 Simulation of Mesoscale

Following the methodology outlined in Chapter 4, we detail the complete algorithmic approach to simulating the *Generalized Probabilistic Cellular Automata* as our meso model. This algorithm will be implemented as one part of the complete PDE simulation, detailed in the following Section.

Algorithm to Couple the Micro and Meso Models:

We now present an algorithmic view of the meso model, below:

Algorithm 6 *Probabilistic Cellular Automata* at the Meso Scale

initialize D_T = n $\times$ n zero matrix ▷ n is the number of rows/columns in our window

draw $N(T) \sim$ Poisson($T\lambda$) ▷ create an initial state of damage
for i in N(T) **do**
  $\bar{c}_i =$ draw from cascade fractional density distribution, Eq. 2.3.2
  $x_i =$ draw from discrete unif(0,n)
  $y_i =$ draw from discrete unif(0,n)
  $D_{T_{x_i,y_i}} = D_{T_{x_i,y_i}} + \bar{c}_i$
end for
while not steady state **do**

  create list of all $clusters$

  for each $cluster$ **do**
    create $extendedCluster$
  end for

  for each $extendedCluster$ **do**
    for each cell in $extendedCluster_j$ **do**
      draw from Bernoulli(p_i) ▷ p_i is defined in Eq. 3.4
    end for
  end for
end while

Introduce Meso Grid into PDE Source Term as Concentration Field

Simulating Meso Models to Steady State:

The algorithm detailed above relies on the “While" loop to accomplish the evolution per the movement rules. We follow the results of Section 4.5 to determine the adequate runtime for each full PDE simulation. These steady states, as previously shown, depend on the various parameters of the model: namely the neighbor weights, or the transition probabilities.

The analytical properties derived in Section 4.5 show that initial damage states (percentages) and domain size, are independent of the rates of convergence to steady states. This allows us to more efficiently simulate the *cellular automata* models, as we do not need to dynamically change the simulation times per the damage and domain during the PDE simulation. Instead, we simply need to run the simulation according to the transition probabilities used and the critical state phase transition values.

5.3 Practical Implementation of Stochastic Source in PDE

Using Algorithm 6, we can now fully implement our meso scale model in our engineering-scale PDE simulation.

Algorithm 7 Stochastic Cascade Generation, and Evolution, as Defect Production Source

for each time step of the PDE solve **do**

$C_{T_{x_i,y_i}}$: current Concentration Field at Meso Scale

initialize D_T = n $\times$ n zero matrix ▷ n is the number of rows/columns in our window

draw $N(T) \sim$ Poisson($T\lambda$) ▷ create an initial state of damage
for i in N(T) **do**
$\bar{c}_i =$ draw from cascade fractional density distribution, Eq. 2.3.2
$x_i =$ draw from discrete unif(0,n)
$y_i =$ draw from discrete unif(0,n)
$D_{T_{x_i,y_i}} = D_{T_{x_i,y_i}} + \bar{c}_i$
end for
$C_{T+1_{x_i,y_i}} = C_{T_{x_i,y_i}} + D_{T_{x_i,y_i}}$

while not steady state **do**

create list of all $clusters$

for each $cluster$ **do**
create $extendedCluster$
end for

for each $extendedCluster$ **do**
for each cell in $extendedCluster_j$ **do**
draw from Bernoulli(p_i) ▷ p_i is defined in Eq. 3.4
end for
end for
end while

Return Meso Grid into PDE Source Term as updated Concentration Field
end for

5.4 Numerical Simulations

We investigate the properties of our methodology through two proposed experiments. The physical behaviors that are being modeled at each scale interact with the various model parameters through two driving forces: the transition probabilities of the stochastic process at the meso scale, and the ratio between the meso scale grid and the engineering scale discretization.

For this very reason, we perform simulations of the Phase Field models using our stochastic source terms with the goal of investigating the changes in PDE solution from changes in the stochastic input parameters.

Throughout time, there are two main integral quantities measured in each of the simulations to observe time behaviors of the model: the void volume fraction and void volume average across the domain. See definitions, and explanations, in Section 2.3

$$\pi_{\Omega} = \frac{< c_v >^{casc}_{total}}{\Omega} \tag{5.5}$$

and

$$\Pi_{\Omega} = \frac{\mathcal{N}_{\ell}}{\Omega} \tag{5.6}$$

where $\mathcal{N}_{\ell}$ is the number of damaged lattice sites on the PDE domain mesh.

ht

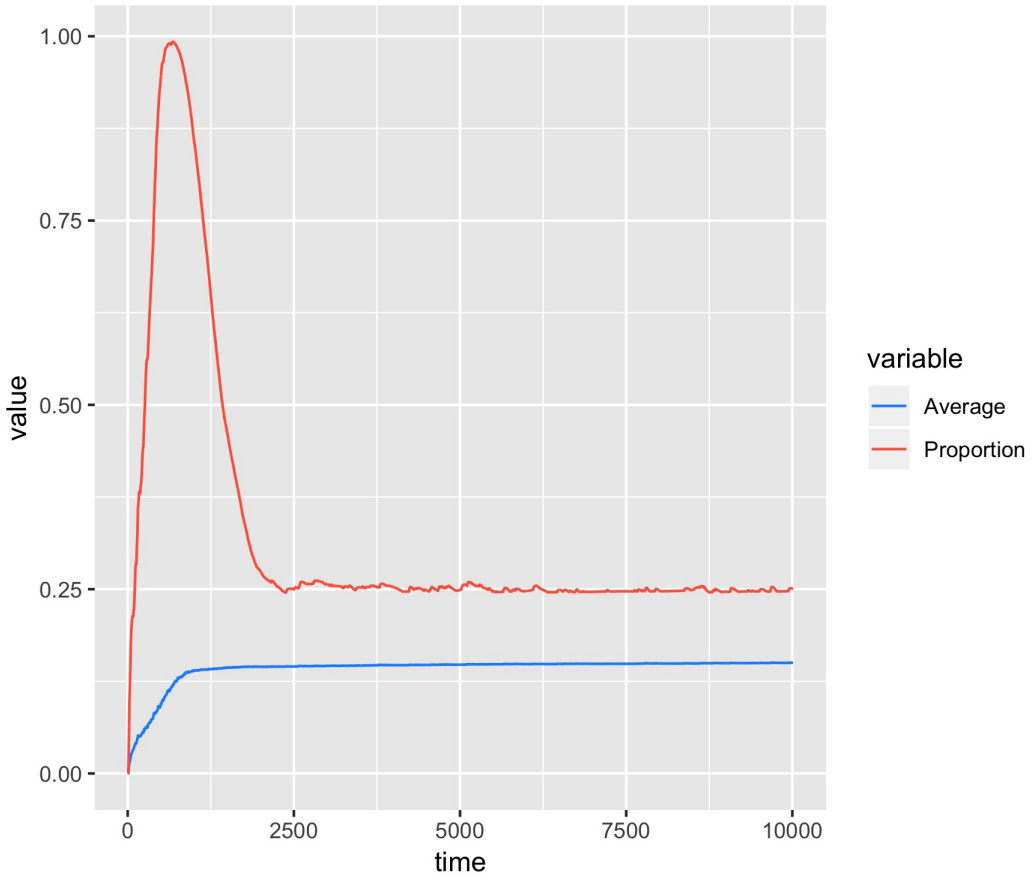

Figure 5.4: Integral Quantities against Time

Figure (5.4) illustrates the evolution of these integral quantities over time, highlighting the varying stages of void nucleation from incubation through nucleation, growth and stability.

ht

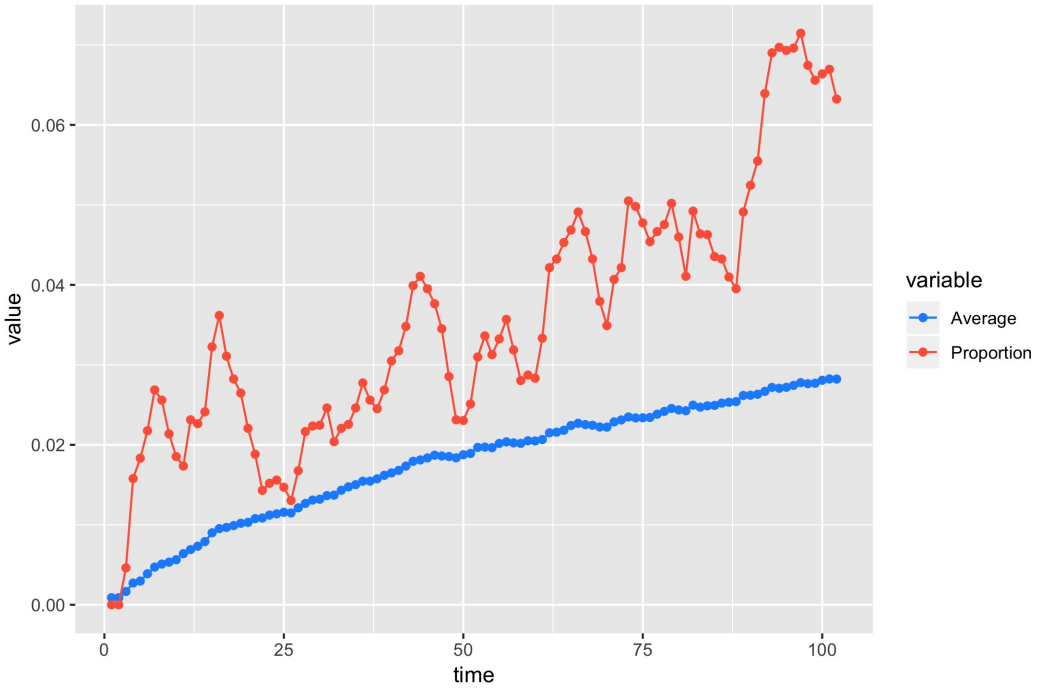

Figure 5.5: Stochastic Short-Term Variations Over Time

We also highlight the stochastic nature of the forcing on the concentration fields over time, as seen in the evolution of the integral quantities in Figure (5.5). Although the Phase Field Models act as an averaging smoother, the stochastic fluctuations have an evident effect on forcing the PDEs.

5.4.1 Experiment 1: Changes in Transition Probabilities

The first experiments that we perform investigate the interactions between the meso model and the PDE solver when the transition probabilities, or neighbor weights, defined in Eq. 3.5, vary. As was detailed in Chapter 4, by varying these transition probabilities, we greatly affect the long-term properties of the stochastic process, both in terms of the steady states reached, as well as the rates of diffusion.

In the following experiments all PDE and micro scale parameters are kept equal while we only vary the transition probabilities of the *cellular automata* model; the neighbor weights. We find

ht

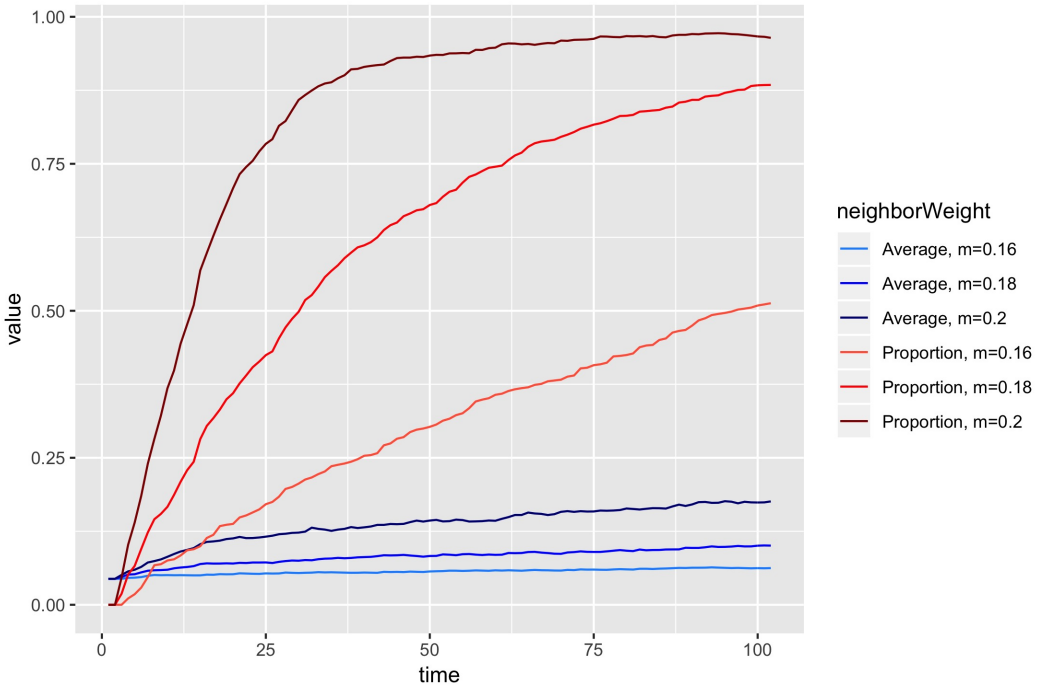

Figure 5.6: Integral Quantities with varying neighborWeights

that the transition probabilities have an enormous effect on the rates at which the PDE evolves the system.

The effects of varying the transition probabilities are most evident in the stage of void incubation, through to nucleation. Thusly, there is a rather large effect on the physics being modeled by the Phase Field equations.

5.4.2 Experiment 2: Changes in the Ratio between Meso and Macro Grids

The second experiment that we perform investigates the interactions between the meso model and the PDE solver when the ratio between the meso scale and engineering scales vary. This is where the scales are explicitly defined. Since the micro scale cells capture the quantities of damage in terms of densities, these cells have a size that is predetermined. Likewise, the meso scale is explicitly defined.

ht

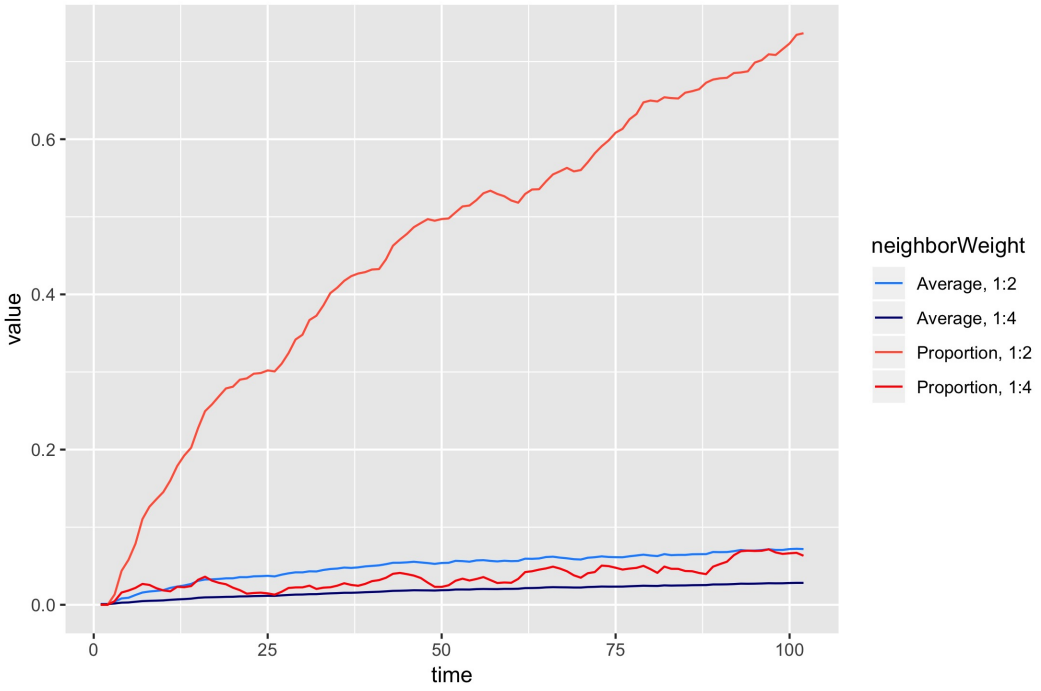

Figure 5.7: Integral Quantities with varying ratios of Meso:Macro

We follow the definitions in Section 5.2 regarding the construction of the computational domain. Since we are explicitly defining two (or more) scales simultaneously, we find an interaction effect between the two.

In the following experiments all micro scale parameters and PDE initial conditions are kept equal while we only vary the ratio between the grid at which the *cellular automata* model is simulated and the discretization of the PDE solver. As were the results in the first experiment, we find that this ratio has an enormous effect on the rates at which the PDE evolves the system and the states to which it reaches.

The effects of varying the transition probabilities are most evident in the stage of void incubation, through to nucleation. Thusly, there is a rather large effect on the physics being modeled by the Phase Field equations.

ht

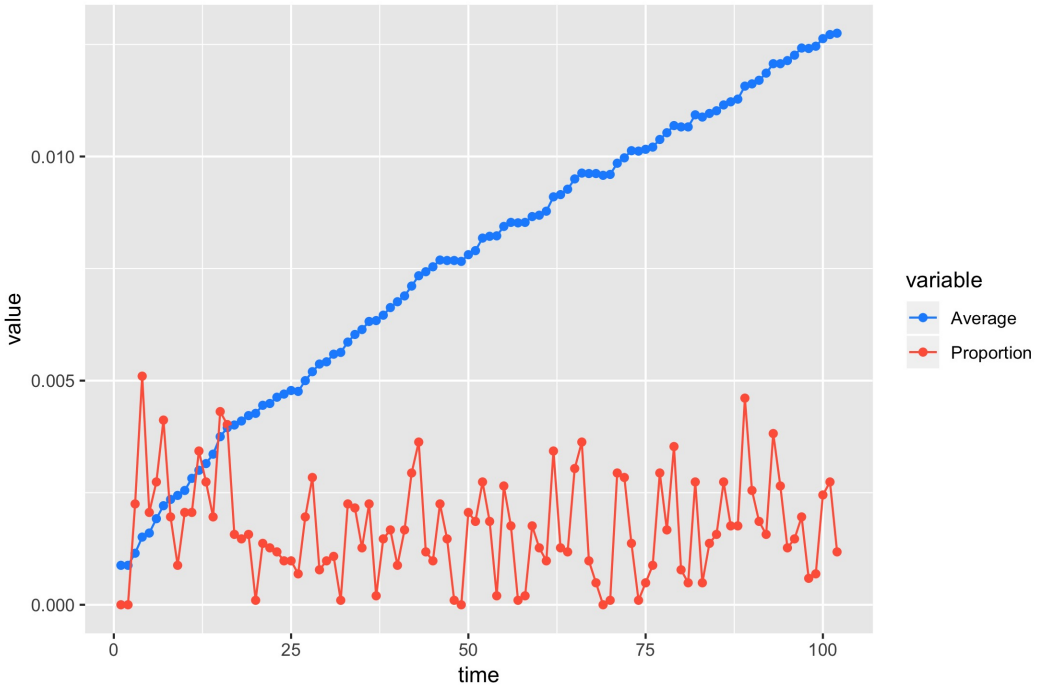

Figure 5.8: Integral Quantities with high ratio of Meso:Macro... 1:10

Dislocated Scales

There is a point where the ratio between the meso grid and the PDE discretization no longer functions to communicate well. Figure 5.8 illustrates this event, where the meso scale grid is too small to communicate the information to the PDE discretization.

We can see in this plot, Figure 5.8, that the PDE “smoothes" out any stochastic fluctuation of introduced damage. Although micro scale events are occurring, the Phase Field Solver tries to minimize the interfacial free energy and the damage never spreads nor accumulates. This is partly because the domain of the meso scale has increased dramatically with regards to the PDE discretization (ie. the movement at the meso scale needs to be much greater to be “seen" by the PDE).

5.5 Discussion of Results

As hypothesized, the properties of the forcing term, 2.9, in our Phase Field Model, 2.2, had a large effect on the material properties being modeled. Under our proposed modeling scheme for the forcing term of the differential equation, we are able to explicitly specify the physical properties and behavior of our desired material and radiation conditions. Small changes in the parameters of the proposed stochastic process affected the void concentration field behavior.

Our method allows us to specify the model from first principles at the micro, atomic scale from the original physical source of defect and damage processes driven by irradiation in the form of atomic collision cascades. Explicitly defining the meso scale also allows a flexibility in modeling, as we are permitted to specify differing behaviors at each time and space scale. We find that small changes at each scale affects the subsequent behaviors of void nucleation and growth in supersaturated material.

We also find that our stochastic process model at the meso scale permits us to spatially resolve the coupling between our atomic and meso scales in a manner that maintains consistency with the PDE solver. Since our upscaling from the molecular level is non-dimensional, we are able to give the PDE concentration field densities, on which both the stochastic process and the PDE can act on.

Chapter 6

Concluding Remarks and Future Development

Our approach is a novel deconstruction of the additive forcing term of an adapted Cahn-Hilliard equation for Phase Field Models.

Our deconstruction permits an explicit coupling from first-principal, physical processes at the micro scale, all the way to the engineering scale PDE models, utilizing stochastic processes and statistical upscaling at the meso scale to bridge these greatly-differing scales. We propose coupling well-founded simulations of binary collision cascades to represent the atomic displacement cascades that naturally occur in materials under irradiation. We then take non-dimensional approximations of the damage events to create a spatially resolved coupling to a domain mesh at the meso scale that permit consistent solutions. Through stochastic processes, we resolve the time and space scales of the physics from the micro level to the PDE engineering scales that maintains statistical fidelity throughout the physical processes being modeled.

Our method and simulations bring a statistical and mathematical rigor to multi-scale modeling unseen in the current literature with far reaching implications beyond irradiated materials.

Future Work

While this work is rather large in breadth, there is vast depth yet to be explored. The analytical, and even numerical, properties of the models at each stage are hardly uncovered. The couplings between each scale, from *micro* to *meso* and *meso* to *macro*, have profound interaction effects that need to be explored. And the flexibility of the meso scale stochastic process models open the doors to vast opportunities in both modeling work and pure probabilistic theory.

The chief impetus for this modeling work comes from the pragmatic need to characterize materials properties undergoing outside sources of forcing that change the dynamics of the material's physics. These physical processes are stochastic in nature and remain to be robustly investigated.

These methodologies could be slightly altered, or even unchanged, and immediately applied to other contexts of physical models even outside the realm of materials.

Lastly, these investigations may have profound impacts on various fields, all while the academic work is immensely rewarding.

Bibliography

[1] R Bullough and R. S. Nelson. Voids in irradiated metals. *Phys. Technol.*, 5:29–67, 1974.

[2] R. Simth. *Atomic Ion Collisions in Solids and at Surfaces: Theory, Simulation and Applications*. Cambridge, UK. Cambride University Press, 1997.

[3] M. Robinson. Computer simulation studies of high-energy collision cascades. *Nuclear Instruments and Methods in Physics Research Section B*, 67((1-4)):396–400, 1992.

[4] M. Robinson and I. Torrens. Computer simulation of atomic displacement cascades in solids in the binary-collision approximation. *Physical Review B*, 9((12)):5008–50024, 1974.

[5] J.F. Ziegler. Srim-2003. *Nuclear Instruments and Methods in Physics Research Section B*, 1027:219–220, 2004.

[6] *The Stopping and Range of Ions in Solids*. Pergamon Press, New York, 1985.

[7] K. C. Russell. Nucleation of voids in irradiated metals. *Acta Metall.*, 19:753–758, 1971.

[8] R.M. Mayer and Brown L.M. Nucleation and growth of voids by radiation: Ii. differential equations. *Journal of Nuclear Materials*, 95:58–63, 1980.

[9] S. Rokkam, El-Azab A., P. Millet, and Wolf D. Phase field modeling of void nucleation and growth in irradiated metals. *Modeling and Simulation in Materials Science and Engineering*, 6(17):64, 2009.

[10] Martin Gardner. Mathematical games - the fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223:120–123, 1970.

Appendix A

Generalized Probabilistic Cellular Automata Code

```
1
2  def markovChain(numRow=15, numCol=15, sample_size=1000, scheme=1,
3       neighbor_weight=1):
4      """
5      Generates a Markov Chain Monte Carlo sample of grids of void
6       damage.
7
8      This code will generate an initial state of the grid and apply
9       the cell-interaction behavior specified in the argument of
10   the function call, along with new void production.
11
12     Parameters:
13
14         *numRow is the number of rows desired in your grid
15           (default=15)
16         *numCol is the number of desired columns
17           (default=15)
18         *sample_size is simply the length of the chain
19           (default=1000)
20         *scheme refers to the cell-interaction behavior scheme
21           (default is Scheme 1)
22
23     Returns: A NumPy array containing the sample path.
24     """
25     # === Set up an array to store the output === #
```

```
    # using dtype=object allows storing of arbitrary Python objects
    X_chain = np.empty(sample_size, dtype=object)

    # === Initialize an initial state for our grid === #
    nrow = numRow
    ncol = numCol
    m = neighbor_weight
    r = 3  # this is the rate for my poisson...
      # this dictates how fast our states evolve
    X=np.zeros((nrow,ncol))

    # === Damage our original grid === #
    for k in range(5):
        D=np.zeros((nrow,ncol))
        N=np.random.poisson(r)
         """r is the parameter given
           by the physics of the problem
(the inverse of the cascade occurence rate)

        """
        for i in range(N):
            p=np.random.choice(data)
            x=np.random.randint(0,nrow)
            y=np.random.randint(0,ncol)
            D[y][x]=p

        X=np.add(X,D)        # this becomes our initial state
    X_chain[0] = X

```

```
55
56      # === Define a few functions that we will be needing === #
57      def checkNeighbors(t):
58          """
59          checkNeighbors() will take a cell index (ie (i,j) )
60            and create and return a list of indices of
61    its four neighbors (to the immediate right
62    and all three adjecent below) that also have
63    voids
64
65          we will use this when we construct the mini clusters
66          """
67          # if the damaged cell is the last column,
68            #can't check to its right
69          neighbors = []
70          if t[1] != ncol-1:
71              if X[t[0]][t[1]+1] > 0:
72                  neighbors.append( (t[0], t[1]+1) )
73                  # as above, if last row then nothing below it
74              if t[0] != nrow-1:
75                  for i in range(-1,2):
76                      if X[ t[0] + 1 ][ t[1] + i] > 0:
77                          neighbors.append( (t[0]+1, t[1] + i) )
78          else:     # those in the last column are sent down here
79              if t[0] != nrow-1:
80                  for i in range(-1,1):
81                      if X[ t[0] + 1 ][ t[1] + i] > 0:
82                          neighbors.append( (t[0]+1, t[1] + i) )
83          return neighbors
```

```
84
85      def extend(cluster):
86          """
87          extend() will take a cluster (a list of tuples) as input
88            and determine all of its neighbors, placing
89    them in the list neighbors[]
90
91          extend() will return a list of all of the cells in the
92            extended cluster
93          """
94          neighbors = []
95          for k in range(len(cluster)):  #for each cell in cluster
96              if cluster[k][0] == 0:          #cell is on top border
97                  for i in range(0,2):
98                  #cell on top and left border
99                      if cluster[k][1] == 0:
100                         for j in range(0,2):
101                             neighbors.append(
102                              (cluster[k][0]+i,cluster[k][1]+j) )
103                     #cell on top and right border
104                     elif cluster[k][1] == ncol-1:
105                         for j in range(-1,1):
106                             neighbors.append(
107                              (cluster[k][0]+i,cluster[k][1]+j) )
108   #cell only on top and no side border
109                     else:
110                         for j in range(-1,2):
111                             neighbors.append(
112                              (cluster[k][0]+i,cluster[k][1]+j) )
```

```
#cell is on bottom border
            elif cluster[k][0] == nrow-1:
                for i in range(-1,1):
                #cell on bottom and left border
                    if cluster[k][1] == 0:
                        for j in range(0,2):
                            neighbors.append(
                             (cluster[k][0]+i,cluster[k][1]+j) )
 #cell on bottom and right border
                    elif cluster[k][1] == ncol-1:
                        for j in range(-1,1):
                            neighbors.append(
                             (cluster[k][0]+i,cluster[k][1]+j) )
 #cell on bottom and no side border
                    else:
                        for j in range(-1,2):
                            neighbors.append(
                             (cluster[k][0]+i,cluster[k][1]+j) )
#cell is only on left border
            elif cluster[k][1] == 0:
                for i in range(-1,2):
                    for j in range(0,2):
                        neighbors.append(
                          (cluster[k][0]+i,cluster[k][1]+j) )
 #cell is only on right border
            elif cluster[k][1] == ncol-1:
                for i in range(-1,2):
                    for j in range(-1,1):
                        neighbors.append(
```

```
142                             (cluster[k][0]+i,cluster[k][1]+j) )
143             else:       #cell is not on ANY border
144                 for i in range(-1,2):
145                     for j in range(-1,2):
146                         neighbors.append(
147                             (cluster[k][0]+i,cluster[k][1]+j) )
148     #these will be the 9-cell block
149         return neighbors
150
151
152     def getTotalVoid(cluster, newList):
153         """
154         This function takes a list of clusters
155           (which, in turn, are a list of tuples)
156         and will fill the newList with
157           the total amount of void in each cluster.
158
159         Parameters:
160             *cluster is actually a LIST of clusters
161             *newList is generally an empty list to be filled
162         """
163         for i in range(len(cluster)):
164             count = 0.0
165             for k in range(len(cluster[i])):
166             #this takes the void in the cell and adds it to count
167                 count += X[cluster[i][k][0]][cluster[i][k][1]]
168             newList.append(count)
169
170
```

```
171     def numNeighbors(tup, cluster):
172         """
173         numNeighbors() takes an index of a cell and the
174           extended cluster that it is in, and
175   returns a count of the number of neighbors
176           that have any positive amount of void.
177 
178         numNeighbors() also counts each cell itself,
179           as well, IF it has void.
180 
181         we are also adding m to the count if the cell has void,
182           this allows us to control to what extent we want
183   existing cells with void to retain that void
184         """
185         count = 0
186         for i in range(len(cluster)):
187             if X[cluster[i][0]][cluster[i][1]] > 0 and
188               tup[0]-1 <= cluster[i][0] and \
189                 cluster[i][0] <= tup[0]+1 and tup[1]-1 \
190                   <= cluster[i][1] and
191                   cluster[i][1] <= tup[1]+1:
192                 count += 1
193         # === here, we add 'm' to each cell that, itself,
194          # === has void  === #
195         if X[tup[0]][tup[1]] > 0:
196             count += m
197         return count
198 
199     # === This is the "second edition" of numNeighbors(),
```

```
200     # ===  in which we count each neighbor as m === #
201     # === this is used in scheme 2 and allows more flexibility
202     # ===   insofar as cluster growth/movement === #
203     def numNeighbors2(tup, cluster):
204         """
205         numNeighbors() takes an index of a cell and
206           the extended cluster that it is in, and
207   returns a count of the number of neighbors
208           that have any positive amount of void.
209
210         numNeighbors2() also counts each cell itself,
211           as well, IF it has void.
212
213         we are also subtracting m from the count if
214           the cell has void, thus ONLY counting neighbors
215         """
216         count = 0
217         for i in range(len(cluster)):
218             if X[cluster[i][0]][cluster[i][1]] > 0 and
219               tup[0]-1 <= cluster[i][0] and \
220                 cluster[i][0] <= tup[0]+1 and tup[1]-1 \
221                   <= cluster[i][1] and cluster[i][1] <= tup[1]+1:
222                 count += m
223         # === here, we subtract 'm' from each cell that, itself,
224         # === has void, since we already counted it above  === #
225         if X[tup[0]][tup[1]] > 0:
226             count -= m
227         return count
228
```

```
229
230
231
232
233     # === Now, we do the cell-interaction behavior and damage
234     # === for each step of the chain=== #
235
236     for t in range(sample_size - 1):
237
238         # === Introduce damage to the grid === #
239         for k in range(5):
240             D=np.zeros((nrow,ncol))
241             N=np.random.poisson(r)
242             # === r is the parameter given by the physics of the
243             # === problem
244             # === (the inverse of the cascade occurence rate)
245
246             """ Clearly this is NOT optimized,
247                 yet we proceed for *** sake.
248                 """
249             for i in range(N):
250                 p=np.random.choice(data)
251                 x=np.random.randint(0,nrow)
252                 y=np.random.randint(0,ncol)
253                 D[y][x]=p
254
255             X=np.add(X,D)        # this becomes our initial state
256
257
```

```
258         # === Identify the cells with void === #
259         voidCells=[]
260         for i in range(nrow):
261             for k in range(ncol):
262                 if X[i][k] > 0:
263                     voidCells.append((i,k))
264 
265         # === Make void cells into miniClusters === #
266         miniClusters = []
267         for i in range(len(voidCells)):
268             temp = checkNeighbors(voidCells[i])
269             temp.append(voidCells[i])
270             miniClusters.append(temp)
271         # === Convert the miniClusters list into a tuple
272         # === to create the network graph === #
273         for i in range(len(miniClusters)):
274             miniClusters[i] = tuple(miniClusters[i])
275 
276 
277         # === Use the networkX library to find the strongly
278         # === connected components of our graph === #
279         G=nx.Graph()
280         for i in range(len(miniClusters)):
281             if len(miniClusters[i]) > 1:
282                 for k in range(len(miniClusters[i])):
283                     if k < len(miniClusters[i])-1:
284                         G.add_edge(miniClusters[i][k],
285                           miniClusters[i][k+1])
286             else:
```

```
287                 G.add_node(miniClusters[i][0])
288
289         # === take the clusters out of the graph === #
290         bigClusters=nx.connected_components(G)
291
292         # === Extend bigClusters to include all of the
293         # === immediate neighbors === #
294         extClusters = []
295         for i in range(len(bigClusters)):
296             temp = extend(bigClusters[i])
297             temp=set(temp)
298             # === gets rid of duplicates along with
299             # === the following list generator
300             extClusters.append([i for i in temp if i in temp])
301
302             ## ++++++++++++++++++++++++++++++++++++++++ ##
303         ## ++++++++++++++++++++++++++++++++++++++++++++++++ ##
304         # === We now perform the cell-interaction behavior === #
305
306
307         ### ++++++++++++++ ###
308         ## === if Scheme 1 === ##
309
310         if (scheme == 1):
311             """
312             We are now using Scheme 1
313             """
314             # === Set up a vector containing
315             # === the total void in each cluster === #
```

```
316             totalClusterVoid = []
317             getTotalVoid(bigClusters, totalClusterVoid)
318 
319             # === Now create a list of only the extended
320             # === cells in the extClusters === #
321             for i in range(len(extClusters)):
322                 extClusters[i] = set(tuple(extClusters[i]))
323             for i in range(len(bigClusters)):
324                 bigClusters[i] = set(tuple(bigClusters[i]))
325             extendedIndices = []
326             for i in range(len(extClusters)):
327                 extendedIndices.append(
328                   list(extClusters[i]-bigClusters[i]))
329 # === accomplished using set operations "-"
330             # === and switch our sets back into lists === #
331             for i in range(len(extClusters)):
332                 extClusters[i] = list(extClusters[i])
333             for i in range(len(bigClusters)):
334                 bigClusters[i] = list(bigClusters[i])
335 
336 
337             # === Create a new list of cells that will have void
338             # === after distributing totalClusterVoid === #
339             newbigClusters = []
340             newbigClusters = bigClusters
341 
342             # === Roll the dice to see who is included in the
343             # === cluster for the next time step === #
344             alpha = 1.0
```

```
345             for i in range(len(extClusters)):
346             # === loop over every cluster
347                 totalNeighbors = 0
348                 for j in range(len(extClusters[i])):
349                 # === get total neighbor count for cluster
350                 # === (loop over every cell in cluster)
351                     totalNeighbors += numNeighbors(
352                         extClusters[i][j], extClusters[i])**alpha
353                 for k in range(len(extendedIndices[i])):
354                 # ===  now, roll the dice for every
355                 # ===  extended-neighbor cell in the cluster
356                 p = float(numNeighbors(
357                      extendedIndices[i][k],extClusters[i])**alpha )\
358     /totalNeighbors
359                 if (np.random.binomial(1,p)):
360                     newbigClusters[i].append( \
361                          extendedIndices[i][k])
362
363             # === Now simply distribute the totalClusterVoid
364             # === over the new cluster ===#
365             for i in range(len(newbigClusters)):
366             # === for each cluster
367                 for k in range(len(newbigClusters[i])):
368                 # === take each cell in that cluster
369                     X[newbigClusters[i][k][0]][\
370                       newbigClusters[i][k][1]] \
371                       = totalClusterVoid[i]/len(newbigClusters[i])
372             """
373             # === Introduce healing to the grid === #
```

```
            for k in range(5):
                N=np.random.poisson(r)
                # === we heal the cells at the same rate
                # === that we are introducing void

                for i in range(N):
                    x=np.random.randint(0,nrow)
                    y=np.random.randint(0,ncol)
                    X[x][y] = 0
            """









        ### ++++++++++++++ ###
        ## === if Scheme 2 === ##

        elif (scheme == 2):
            """
            We are now using Scheme 2
            """
            # === Set up a vector containing the
            # === total void in each cluster === #
            totalClusterVoid = []
            getTotalVoid(bigClusters, totalClusterVoid)

            # === Create a new vector quasibigClusters === #
```

```
403             quasibigClusters = []
404             for i in range(len(extClusters)):
405                 quasibigClusters.append([])
406 
407             # === We now roll the dice on each cell in extClusters
408             # === for inclusion in the next iteration === #
409             # === This allows the possibility of healing AND a
410             # === cluster breaking apart === #
411             # === Thus, the receiving list is quasi since they may
412             # === not actually be clusters === #
413             alpha = 1.0
414             for i in range(len(extClusters)):
415             # === loop over every cluster
416                 """
417                 totalNeighbors = 0
418                 for j in range(len(extClusters[i])):
419                 # === get total neighbor count for cluster
420                 # === (loop over every cell in cluster)
421                     totalNeighbors += numNeighbors(
422                       extClusters[i][j], extClusters[i])**alpha
423                 """
424                 for k in range(len(extClusters[i])):    #
425                 # === now, roll dice for every extended-neighbor
426                 # === cell in the cluster
427                     neighbors_w_void = float(
428                      numNeighbors2(extClusters[i][k],
429    extClusters[i]))
430 # ===  here, we are using numNeighbors2()
431                     p = neighbors_w_void / (neighbors_w_void + 1)
```

```
                    if (np.random.binomial(1,p)):
                        quasibigClusters[i].append(
                          extClusters[i][k])

            # === Now distribute the void from whence
            # === they came (even if no longer a cluster) === #
            X_new=np.zeros((nrow,ncol))

            for i in range(len(quasibigClusters)):
            # === for each cluster
                if (quasibigClusters[i]):
                # === some are empty, skip them
                    for k in range(len(quasibigClusters[i])):
                    # === take each cell in that cluster
                        if (totalClusterVoid[i] > \
                          len(quasibigClusters[i]) ):
                        # ===can't recieve more than 100% void
                            X_new[quasibigClusters[i][k][0]] \
                              [quasibigClusters[i][k][1]] = 1.0
                        else:
                            X_new[quasibigClusters[i][k][0]] \
                             [quasibigClusters[i][k][1]] \
    = totalClusterVoid[i]/len(quasibigClusters[i])
            X = X_new

        X_chain[t+1] = X

    return X_chain
```