An investigation into jamming percolation using renormalization group methods.

Samuel Schoenholz

Under the supervision of Professor Amy Bug and Professor Andrea Liu

March 22, 2010
1 Overview

Many systems in nature fail to satisfy the ergodic hypothesis and feature strong nonequilibrium behavior. These systems have proven remarkably difficult to study. In recent years theorists have studied a phase transition known as the jamming phase transition. The jamming phase transition unifies several far from equilibrium phase transitions including: the glass transition, the solidification of granular media in response to shearing forces, and the stiffening of colloids in response to increases in pressure. It has recently been shown that certain correlated percolation models can exhibit behavior analogous to the jamming phase transition. In particular this has been shown for two models, coined the spiral model and the force-balance model.

The primary goal of this study was to determine whether or not the spiral model and the force-balance model reside in the same universality class. To investigate this issue we used the renormalization group to perform a numerical investigation of the spiral model and force balance model. There is evidence to suggest that both models feature exponential, as opposed to power-law scaling of the correlation function. It was therefore difficult to investigate, in a computationally feasible way, large enough systems so as to reduce the finite size effects. To do this we came up with a variety of algorithms including: a Monte-Carlo method, a binary search, and a linear time culling algorithm.

The combination of these three strategies allowed us to investigate significantly larger systems than had been investigated in the past. Specifically the two preceding studies on which we based our work had investigated systems up to sizes 1600 and 3000. We managed to investigate systems up to 15000 in size. This gave us promising results that agree with exact results found in work by Fisher et al. and Schwarz et al. Although we do not yet have conclusive results, we believe that determining whether these models lie in the same universality class should now be within reach.
## Contents

1 Overview 2

2 Introduction 5

2.1 The Jamming phase transition 7

2.2 Jamming and percolating systems 10

3 Theory 13

3.1 Notions from statistical mechanics 13

3.1.1 Phase Transitions 16

3.1.2 The Ising model 20

3.1.3 Scaling behavior and universality 24

3.2 Percolation theory 26

3.2.1 Spiral percolation 29

3.2.2 Force-balance percolation 34

3.3 Renormalization group theory 35

3.3.1 1D Ising model 36

3.3.2 2D Ising model 39

3.3.3 Renormalization in percolation theory 42

4 Methods 49

4.1 Java simulations 49

4.2 C simulations 51

4.2.1 Exact renormalization 53

4.2.2 Monte-Carlo scheme 56

4.2.3 Binary searching and finite clusters 60

4.2.4 A better culling algorithm? 62

5 Results 67

5.1 Exact renormalization 67
5.2 Agreement between exact and Monte Carlo renormalization. ............... 68
5.3 Agreement for standard percolation. ........................................... 69
5.4 Lack of finite-clusters ............................................................. 72
5.5 Investigating error bounds. ...................................................... 72
5.6 Agreement with the binary search and novel culling algorithms. ......... 75
5.7 Results for the spiral model. .................................................... 76
5.8 Results for 24NN model. ......................................................... 78

6 Conclusion ................................................. 81

7 Acknowledgements ................................................. 83

8 Appendix ......................................................... 88

8.1 Java Code ................................................................. 88
8.2 C code ................................................................. 97
8.3 Merzenne-Twister algorithm ............................................... 116
2 Introduction

Despite the success of physical theories in describing vast portions of nature, there are many basic phenomena that reject our advances. Systems that are in states far from equilibrium have proven particularly evasive. Simultaneously, there are many important, and commonplace, systems that do not lie in a regime where equilibrium statistical mechanics may be applied; some examples of such systems include granular materials and emulsions that flow in response to induced stresses, the cooling of a liquid to form a glass, and the stiffening of colloidal suspensions in response to increased density or pressure[1, 2]. The importance of these topics from a practical perspective is evident in our ubiquitous interactions with both glassy and granular materials. However, these problems are also interesting from a theoretical perspective. As is quoted in [1], the glass transition has been called “the deepest and most interesting unsolved problem in solid state theory[3].”

The glass transition differs from the standard liquid to solid phase transition in several important respects. The first difference between the two phase transitions is that the liquid-glass phase transition is characterized by a second order phase transition during which the stress relaxation time\(^1\) increases continuously by several orders of magnitude over a small range in temperature [4, 2]. This difference may be seen in the contrasting phase diagrams in figure 1. The second manner in which the dynamics of a system undergoing a glass transition differs from those of a system freezing into a crystal is that in the former particles remain disordered while in the latter long range order develops [6, 7]. In fact, a snapshot of such system in the glassy phase is indistinguishable from a snapshot of the same system as a supercooled liquid before the transition. As the molecules in a glass become thermally constrained in a disordered state, many regions of phase space become inaccessible. Thus, glass systems fail the conditions of the ergodic hypothesis and are, thus, not equilibrium systems.

Granular materials experience a similar solidification when an external strain is reduced. For example, shaken sand flows to fill a container but unshaken sand will not [8, 9, 10]. While the mechanism for this transition is, again, an open problem, we do know that the motion of the

\(^1\)The stress relaxation time of a material is the amount of time that it takes the material to relieve stress under constant strain [?]. A system with with a stress relaxation time less than infinity exhibits fluid like behavior.
Figure 1: A comparison between a glass transition and a standard liquid-solid phase transition. This diagram shows how the enthalpy of a material changes with temperature as the material traverses a phase transition. We note that the liquid-solid transition exhibits a discontinuous drop in enthalpy, and incidentally also in volume, while the enthalpy of the glass varies continuously with a characteristic kink at $T_g$. Adapted from Ref. [5].

The granular medium is athermal. The molecules in granular media are, almost by definition, too large to be affected by thermal fluctuations. Hence, random motion within the material must be induced by some external forces for example, shaking the sand in a container. In this way, a frozen granular medium in the absence of shear forces appears to be in a disordered state. Furthermore, because the size of the grains are on the order of the inter-particular spacing, frozen granular media are also unable to explore all of phase space. Thus, they might too be considered far-from-equilibrium systems.

Colloidal suspensions are mixtures in which molecules of one type are suspended in molecules of another. In general the suspended molecules need to be significantly larger than the molecules in which they are suspended for such a state to occur. In disperse colloidal suspensions the larger particles are free to diffuse in the medium and explore phase space. However, as the pressure or density increases and the molecules are packed together more tightly they become constrained. Again in such situations the colloidal suspension develops a yield stress[11, 1, 9]. Furthermore, the colloid is unable to explore phase space and might, thus, be considered a far from equilibrium system. This phenomenon is familiar to anyone who has tried to drink a milkshake but found that
it clogged the straw, but flowed quickly when that person stopped sucking.

These phenomena are certainly disparate. The various transitions result from fundamentally different processes. In the case of the glass transition, the development of yield stress\(^2\) is inherently thermal in origin; by contrast the sand transition is fundamentally kinetic while the colloidal transformation appears to have its origins in entropic, as opposed to energetic, considerations\(^1\). However, recent evidence suggests that all three of these transitions may be described by the same universal theory. In particular: no significant structural deformation has been observed for any of these transitions\(^3\) \([1, 13, 12]\), the stress relaxation time in all of these systems appears to be super-Arrhenius\(^4\) as a function of the control parameter \([1, 14, 15]\), and all of these systems feature dynamical heterogeneities near their respective transitions\(^5\) \([16, 17, 18, 19, 20]\). As a result of these apparent similarities it is attractive to try to search for a unifying theory encompassing all of these phase transitions. The result of these efforts has been coined the “jamming phase transition”.

### 2.1 The Jamming phase transition.

The first goal in constructing a theory that would unify the phase transitions of glasses, colloidal suspensions, and granular materials would be to relate the origins of the respective phase transitions. In 1998 it was proposed in \([21]\) that jammed granular media belong to a new class of materials which they termed ’fragile matter’. Shortly after, Liu and Nagel argued in \([22]\) that this notion of fragile matter could be extended to foams and emulsions as well as glasses and colloidal suspensions. To accomplish this Liu \textit{et al.} noted the similarities between the various phase transitions mentioned above and introduced an unconventional phase diagram that they coined the ’jamming phase diagram’ \([22, 1]\). The jamming phase diagram is pictured qualitatively in figure 2. The jamming diagram allows for three control parameters and hence features three axes. These three axes are the temperature \(T\), the packing fraction \(\phi\), and the shear stress \(\Sigma\). A surface, coined

---

\(^2\)The yield stress is defined as the stress that needs to be applied before plastic deformation of a material.

\(^3\)Structural deformation can be viewed as large scale, with respect to interparticulate spacing, correlations in the structure. In \([12]\) this has been seen directly using diffraction measurements.

\(^4\)In Arrhenius scaling the measured quantity varies as \(\gamma \sim e^{A/\varphi} \) with respect to the control parameter \(\varphi\). In super-Arrhenius scaling the quantity scales faster. An example of super-Arrhenius scaling might be \(\gamma \sim e^{A/(\varphi-\varphi_0)} \) for some \(\varphi_0\).

\(^5\)Dynamical heterogeneities occur when a dynamical property, for example kinetic or rotational motion, as opposed to a structural property, as opposed to density, is heterogeneous throughout a material.
Figure 2: The 'jamming phase diagram'. The axes correspond to the different control parameters: temperature, shear stress, and packing fraction. A 'jamming surface' separates the jammed region from the unjammed region. Finally, point \( J \) is the point on the jamming surface at zero temperature and zero applied shear stress. Adapted from Ref. [1].

the 'jamming surface' separates the jammed region from the unjammed region. The shape of the jamming surface is dictated by the system in question.

We note that each of the systems discussed above can be fit into the jamming phase diagram. The classical glass transition may be thought of as residing on the \( T-1/\phi \) plane of the diagram while the phase transition for a granular system might reside in the \( 1/\phi-\Sigma \) plane [1, 9]. Implicit in the construction of the jamming phase diagram is the assumption that all three control parameters feature in each of the systems, but the range over which these parameters are relevant changes. The utility of the jamming phase diagram implies that shear stresses can force a particles in a thermal system to explore phase space and help the system to flow. Equally, introducing temperature into an otherwise athermal system can also help the system to unjam.

Producing a definition for the jamming phase transition that is well defined is a difficult problem. As is illuminated by systems of glasses, and as can be seen in figure 1, the glassy phase transition is far less abrupt than the standard first-order liquid-solid phase transition. The first-order liquid-solid phase transition features sharp discontinuities in many observable quantities such as the free energy
which allow for a well defined critical temperature $T_c$; by contrast the glass transition features no such discontinuity in an observable parameter must therefore be defined by kink in the free energy. Consequently, and as is apparent in the same figure, the critical temperature $T_g$ is less well defined than the critical temperature of a first-order transition $T_c$.

Ideally the jamming phase transitions would be defined by the point at which the system in question develops a yield stress and behaves like a solid, in practice this definition is impractical. It is often impossible to ascertain whether a system has developed a yield stress and adopted solid character or whether its stress relaxation time has simply exceeded the time-scale of the experiment. This is exemplified by the determination of the position of the kink in the free energy of a glass transition. Thus, a necessarily practical definition for jamming is to determine a system resides in the jammed state if its stress relaxation time exceeds some experimental time-scale.

While the definition of the transition between jammed and unjammed regimes is a practical one it is also theoretically challenging. Because the jamming transition is defined arbitrarily in terms of an experimental time-scale, perturbations of the defining time-scale can correspondingly induce perturbations in the jamming surface [1]. It is quite difficult to generally study the jamming surface for a particular system due to this inherent uncertainty in the surface. However, there is a point, labeled $J$ on figure 2 at zero temperature and applied stress, that has been demonstrated [1, 23, 24] to be well defined for systems exhibiting repulsive finite-ranged potentials. In particular, in the infinite system limit the packing fraction at which systems jam is found in [1] to be independent of both the details of the potential and the procedure used to prepare the system.

In addition to being a well defined point on the jamming surface, recent studies [1, 9, 25, 23, 24] of point J have found that it features numerous important characteristics that make its study worthwhile. It was found that the packing fraction $\phi_c$ at point $J$ occurs at the random closed packing fraction\textsuperscript{6}. An additionally notable aspect of point J is that it is an isostatic point where the number of contacts between the various particles is equal to the number of force-balance equations used to describe them. This both implies that point J is a geometric condition, independent of the

\textsuperscript{6}The random closed packing fraction is the natural packing fraction that an ensemble of hard sphere systems will take on upon being shaken. It was argued [26] that due to its definition as an ensemble quantity and dependence on the protocol by which the ensembles are created, random closed packing is not a well defined quantity despite its importance [27, 28, 29, 30].
details of the potential in question and that point J is a minimal stable configuration. However, the most directly interesting feature of point J is evidence that the jamming phase transition features many of the properties making up the glassy transition.

2.2 Jamming and percolating systems.

Percolation theory\(^7\) has been used to study nature since World War II when Flory and Stockmayer used it to describe how large macromolecules are composed of small branching molecules\(^3\). Percolation theory steadily grew in popularity due to its utility as an effective method of making complex systems more manageable, while still retaining much of the fundamental physics. Thus, percolating systems can often yield interesting qualitative and quantitative results about systems that would otherwise defy analysis. Percolating systems have been studied extensively in statistical mechanics since they are simple systems that exhibit phase transitions. In the case of standard percolation the phase transition is similar to a liquid-gas phase transition.

As will be discussed in section 3.3, standard percolation theory, in which sites on a lattice are randomly occupied, may be extended by preparing the lattice so that the various sites are correlated. The procedure of correlating the lattice is often accomplished by establish some criteria and culling those sites of a random lattice that fail to satisfy the established rules. It was recently noted in [25] that a certain class of correlated percolation systems, known as k-core systems\(^8\), exhibit many of the same characteristics as the jamming transition displays at point J. In particular the same study argues that both feature mixed phase transitions.

In general statistical mechanical systems may exhibit two distinct types of phase transitions: first order phase transitions and continuous phase transitions. First-order phase transitions are those changes in phase that feature a discontinuity in the derivative of the thermodynamic potentials. By contrast continuous phase transitions are those changes of phase that feature no such discontinuities\(^3\). In general continuous phase transitions are characterized by power law scal-

\(^7\)Percolating systems consist of a, usually infinite, graph \((G, E)\) whose vertices can either be occupied or unoccupied. Generally graphs are occupied randomly with some probability \(p\).

\(^8\)In k-core percolation an occupied site is called stable if it has \(k\) adjacent occupied neighbors. A site that is not stable is called unstable. After creating a random lattice all unstable sites are recursively deleted until every occupied site is stable.
ing in thermodynamic quantities near the critical point. It is possible for certain phase transitions to feature the power-law scaling characteristic of continuous phase transitions and a discontinuity characteristic of first-order phase transitions. The jamming phase change at point J appears genuinely feature both continuous and first-order characteristic\[25\]. It has been argued that the number of contacts features a discontinuity characteristic of a first order phase transition while the diverging length scales are characteristic of second-order phase transitions.

It was further proposed in [34, 35] that a new class of correlated percolation systems known as jamming percolation systems might give insight into the glass transition. The premise behind the various jamming percolation schemes is that they feature a notion of mechanical stability in the determination of what constitutes an stable\[10\] site. Specifically it was shown that both the glass transition and the transition experienced by the spiral model feature ergodicity breaking above the critical density, $\rho_c$ and that the fraction of jammed particles is discontinuous at the transition. In the unjammed phase they found that both feature faster than power-law divergence of correlation length and time-scale. Due to these similarities, an in depth knowledge of the various jamming percolation models might illuminate the real jamming phase transition at point J.

In this thesis we continue the work by Jeng and Schwarz in [36, 37] and attempt to study various jamming percolation models. We focus specifically on three models known as the spiral model, the 24NN force-balance model, and the 16NN force-balance model. Our overall goal is to settle the question of whether these three correlated percolation models fall under the same universality class. This would, in turn, extend the directed percolation\[11\] universality exponent to include these three additional models.

---

\[9\] In power-law scaling near a critical point $\phi_c$, thermodynamic quantities often behave as $f(\phi) = (\phi - \phi_c)^\beta$.[32, 5, 33]

\[10\] A site in a correlated system is stable if it satisfies the constraints of the culling algorithm. So, for example, a jamming percolation model might require that some spatial distribution about an occupied site be likewise occupied for the site to be considered stable.

\[11\] The model of directed percolation is rather different from the various models considered in this thesis and will only be mentioned in passing though the interested reader is directed to [33]. In directed percolation correlation is established by ensuring that lattice sites may only be linked in a certain direction. Consequently, in directed percolation the critical exponents must be considered somewhat differently from critical exponents in standard percolation and so we take universality in a somewhat different light.
3 Theory

3.1 Notions from statistical mechanics.

Before approaching the subjects at the core of this study we provide a review of some familiar topics from statistical mechanics and introduce rigorously the concepts of phase transitions and universality. Although some the concepts that we subsequently introduce are true only for equilibrium systems, many of the concepts will carry over to non-equilibrium systems. We begin by discussing the different kinds of systems studied in equilibrium statistical mechanics. The three most prevalent classification of systems are: the microcanonical ensemble, the canonical ensemble, and the grand canonical ensemble. The microcanonical ensemble is that of a totally isolated system and as such each microstate\(^{12}\) has the same energy, volume, and number of particles. The canonical ensemble describes a system where the number of particles and the volume are constant but the system may exchange energy with a much larger heat reservoir. Finally, the grand canonical ensemble describes a system that is allowed to exchange both energy and particles with a connected reservoir. For the theory needed in this thesis we can restrict ourselves to working out details the canonical ensemble.

Fundamental to thermodynamics and, in particular, the physics of phase transitions are the thermodynamic potentials. The thermodynamic potentials are functions that provide different metrics of the energy and entropy of a system. The most intuitive of the potentials is the internal energy of the system, \(U\). In fact, all of the various thermodynamic potentials are intimately related to the internal energy of the system \(U\) via Legendre transformations\(^{13}\). The utility of the internal energy, and more generally the collection of thermodynamic potentials, is related to the first law of thermodynamics. The first law of thermodynamics is a statement of conservation and can be

\(^{12}\)In statistical mechanics a microstate represents an exact (as opposed to statistical) description of a system. It is impossible to know the exact microstate of a system and one must resort to looking at statistical ensembles of microstates to derive the global properties of the system.

\(^{13}\)The Legendre transformation is a change of variables whereby a functions dependence is shifted from a set of variables \(\{x_i\}\) to a set of conjugate variables \(\{p_i\}\) defined by \(p_i = \partial f / \partial x_i\). The Legendre transformation is defined by the equation,

\[
    f(p_1, \cdots, p_n) = f(x_1, \cdots, x_n) - \sum_i p_i x_i.  \tag{1}
\]

Note, for example, that the Hamiltonian is the Legendre transform of the Lagrangian.
written as
\[ dU = \delta Q - \delta W \] (2)
for any process. Here \( \delta Q \) is the heat exchanged between the system and its surroundings and \( \delta W \) is the work done by the system. Crucial to this definition is that neither \( \delta Q \) nor \( \delta W \) are in general exact differentials. This implies that it is, in general, neither true that
\[ \int_C \delta Q = 0 \quad \text{nor that} \quad \int_C \delta W = 0. \] (3)
By contrast \( dU \) is an exact differential and so irrespective of the process \( \int_C dU = 0 \). This property is extremely useful in many thermodynamic calculations.

Although in general it is impossible write down either \( \delta Q \) or \( \delta W \) exactly, in cases where the actions on the system are quasistatic the two quantities can be written as \( TS \) and \( pdV \) respectively. In this case the first law of thermodynamics becomes,
\[ dU = TdS - pdV. \] (4)
Despite the utility of the internal energy as a conceptual tool the requirement inherent in the definition that \( U \) is a function of \( S \) and \( V \) is inconvenient. Instead it is common to consider the Helmholtz free energy given by the Legendre transformation,
\[ F = U - TS \] (5)
whose total derivative becomes
\[ dF = -SdT - PdV \]
and thus becomes a function of \( T \) and \( V \) instead.

Recall that the canonical ensemble represents a system that is weakly coupled to a much larger reservoir. We note that the combination of the system and the reservoir act as a microcanonical ensemble. In an equilibrium system the probability of the system as a whole being in a particular
microstate will be $1/\Omega$ by the ergodic hypothesis\textsuperscript{14}. Here $\Omega$ represents the total number of possible microstates. We now consider the probability $P(E)$ that the system has energy $E$. We note that this probability will be proportional to the multiplicity of the microstates in the reservoir consistent with such an energetic configuration, $P(E_R) \propto \Omega(E_R)$ where $E_R = E_T - E$ and $E_T$ is the total energy of the system. Recalling the definition of entropy in terms of the multiplicity as,

$$S(E_R) = k_B \log(\Omega(E_R))$$

we find the relationship,

$$P(E) \propto e^{S(E_T - E)/k_B}.$$  

(7)

Finally, noting that the total amount of energy is much larger than the energy in the system we may taylor expand to find,

$$P(E) \propto e^{(S(E_T - E)/k_B) \propto e^{-E/k_BT}}$$

(8)

where we have used the relationship $T = \partial E/\partial S$. Often the variable $\beta = 1/k_BT$ is used for convenience and the above relationship may be recast into the form $P(E) \propto e^{-\beta E}$.

The constant of proportionality in the preceding relationship is an exceedingly important quantity in its own right and is known as the inverse of the partition function, $Z$. In many respects deriving the properties of statistical mechanical systems is tantamount to determining and then analyzing the partition function for the system in question. To determine the partition function we consider the normalization condition $\sum_i P(E_i) = 1$. Plugging in we find that,

$$Z = \sum_i g_i e^{-\beta E_i}$$

(9)

where $g_i$ is the degeneracy of states with energy $E_i$. Most of the principal thermodynamic quantities

\textsuperscript{14}The ergodic hypothesis represents a crucial hypothesis differentiating equilibrium statistical mechanics from non-equilibrium analysis. The ergodic hypothesis states that all microstates lying on an equipotential surface will be visited by the system with equal probability.
can be derived from the partition function. For example, the internal energy $U$ will be given by,

$$U = \langle E \rangle = \sum_i E_i P(E_i)$$

$$= \frac{\sum_i E_i g_i e^{-\beta E_i}}{\sum_i g_i e^{-\beta E_i}}$$

$$= -Z^{-1} \frac{\partial Z}{\partial \beta} = - \frac{\partial \log Z}{\partial \beta}.$$  

Similarly we may find the entropy to be given by,

$$S = \sum_i S(E_i) P(E_i) = -k_B \sum_i P(E_i) \log P(E_i)$$

$$= -k_B Z^{-1} \sum_i e^{-\beta E_i} \log Z^{-1} e^{-\beta E_i}$$

$$= k_B Z^{-1} \left[ \sum_i e^{-\beta E_i} (\log Z + \beta E_i) \right]$$

$$= k_B (\log Z + \beta \langle E \rangle) = -\beta^2 \frac{\partial}{\partial \beta} (\beta^{-1} \log Z).$$

Finally we note that the Helmholtz free energy, defined above, can be written in terms of the partition function as,

$$F = \langle E \rangle - TS = k_B \log Z. \quad (10)$$

### 3.1.1 Phase Transitions

We present a rudimentary discussion of phase transitions from [32, 5] from the paradigm of thermodynamics. Phenomenology and more involved topics such as ergodicity breaking are not discussed, but can be pursued by the interested reader in the sources listed. We begin by considering a general system in $d$ dimensions. Recalling that for a finite system $\omega$ with linear dimension $L$ the free energy $F_\omega$ will be given by equation (10). From our knowledge of the various forms of free energy, we know that the free energy of a system will be an extensive quantity. For a large system we will, thus, have the relationship that $F_\omega \propto V(\omega)$ where $V(\omega)$ is the volume of the system $\omega$. We proceed with

---

\[15\] In thermodynamics there are two principal types of quantities extensive and intensive. Extensive quantities vary with system size whereas intensive quantities do not. For example, energy is extensive while temperature is intensive.
the assumption that for a large system $F_\omega$ may be written as,

$$F_\omega = f_b V(\omega) + f_s S(\omega) + O(L^{n-2})$$

where $f_b$ is called the bulk free energy density and $f_s$ is the surface free energy density. Here $S(\omega)$ is the surface area and we note that $V(\omega) \propto L^n$ and $S(\omega) \propto L^{n-1}$.

We can find a definition of $f_b$ and $f_s$ to be given by,

$$f_b = \lim_{V(\omega) \to \infty} \left[ \frac{F_\omega}{V(\omega)} \right]$$

and

$$f_s = \lim_{S(\omega) \to \infty} \left\{ \frac{F_\omega - V(\omega)f_b}{S(\omega)} \right\}$$

where we have neglected the $S(\omega)f_s$ term in the former limit since

$$\lim_{V(\omega) \to \infty} \frac{S(\omega)}{V(\omega)} = 0.$$  

We note that $f_b$ contains the bulk thermodynamic information of the infinite system while $f_s$ contains the surface information and leading-order finite-size effects. When these limits exist then we say that the thermodynamic limit of the system exists. In general, one may not assume the existence of the thermodynamic limit. For systems defined over a lattice, as in the case of percolation, a more practical definition of equation (11) is,

$$f_b = \lim_{N(\omega) \to \infty} \left[ \frac{F_\omega}{N(\omega)} \right]$$

where $N(\omega)$ is the number of lattice sites.

Before we proceed we make the notational note that we will henceforth let $\{K_n\}$ represent the possible coupling constants for the system in question and $\{\Theta_n\}$ represent the possible combinations of the dynamical degrees of freedom of the system. The dynamical degrees of freedom represent the different dimensions in phase space; for example, in a ferromagnetic system the $\{\Theta_n\}$ might
be the spins, or in a mechanical system the \( \{ \Theta_n \} \) might be the various positions and momenta. In this notation the Hamiltonian for the system may be written as

\[
H_\omega = -k_B T \sum_n K_n \Theta_n. \tag{15}
\]

Note, that in general many of the possible couplings will be zero. In general we denote the set of coupling constants \( K \).

In the current theory, the definition and classification of phase transitions is accomplished with respect to \( f_b \). Hence, the notion of phase transition is only sharply defined in the infinite-system limit and some care must be taken in applying the results of the theory to finite-size systems. The thermodynamic will be directly relevant to our research since it is only possible to simulate a finite system. Therefore, as is further treated in section 3.3.3, we must perform our simulations for finite systems and then extrapolate as the system size tends towards the thermodynamic limit. With this caveat, the definition of a phase boundary involves regions on which \( f_b \) fails to be analytic\(^{16}\) as a function of the coupling constants, \( K \). The points at which \( f_b \) fails to be analytic will be surfaces such as points, lines, planes, and hyperplanes in the space of coupling constants \( K \). Supposing that there are \( D \) coupling constants, then two regions of space may be split by a surface of dimension \( D - 1 \). Thus, we define phase boundaries as surfaces of dimension \( D - 1 \) on which \( f_b \) fails to be analytic. We further define phases of a system regions of space separated by a phase boundary. The prototypical phase diagram may be seen in figure 3. We note that in the case of the liquid-gas phase transition it is possible to transition from the liquid state to the gaseous state without passing over a phase boundary. However, the same cannot be said about the liquid-solid phase transition. It turns out that this is a statement that the liquid and gas phases share common continuous symmetries that are broken in the solid phase.

Although the preceding definition of a phase transition in accurate, it is also somewhat broad. There are a variety of ways in which a function may fail to be analytic. In particular any order of derivatives may fail to be continuous. The fundamental behavior of phase transition are grouped into two major categories: first-order phase transitions and continuous phase transitions.

\(^{16}\)A function \( f : \mathbb{R}^n \to \mathbb{R}^m \) is analytic if \( f \) is has continuous derivatives of all orders.\[^{38}\].
Figure 3: The standard solid-liquid-gas phase transition. Note that there is a path through the phase diagram without encountering a point of non-analyticity.

1. First-order transitions: In a first-order phase transition at least one of the partials, \( \partial f_b / \partial K_i \) is discontinuous across the phase boundary. There are often discontinuities in the entropy and pressure since

\[
S = -\frac{\partial f_b}{\partial T} \quad \text{and} \quad P = -\frac{\partial f_b}{\partial V}. \tag{16}
\]

2. In continuous phase transitions at least one of the partials, \( \partial^n f_b / \partial K_i^n \) with \( n > 1 \), must be discontinuous across the phase boundary. The most common continuous phase transitions feature this discontinuity in the second derivative. In this case it is often the specific heat capacities that feature discontinuities since,

\[
C_p = \frac{\partial S}{\partial T} = -\frac{\partial^2 f_b}{\partial T^2}. \tag{17}
\]

Although there are more intricate ways to classify phase transitions, this classification will suffice.

As discussed above first-order phase transitions are defined by a discontinuity in a thermody-
namic quantity at the critical point. For example, if $f$ is such a quantity then,

$$f(\phi) = \begin{cases} 
0 & \phi < \phi_c \\
\phi_0 & \phi > \phi_c
\end{cases}$$

(18)

where $\phi_c$ is the critical point. Continuous phase transitions, on the other hand, are characterized by power law scaling of thermodynamic quantities near the critical point. This often takes the form,

$$f(\phi) \approx A(\phi - \phi_c)^\beta$$

(19)

where $\beta$ is known as the critical exponent. It is possible, however, for systems to feature phase transitions with both first-order and continuous characteristics. In many of these cases the physical quantities in question behave as,

$$f(\phi) = \begin{cases} 
0 & \phi < \phi_c \\
f_c + f_0(\phi - \phi_c)^\beta & \phi > \phi_c
\end{cases}$$

(20)

and so features a discontinuity along with power law scaling[25]. In general it is not an easy problem to determine the type of phase transition and the critical surface along which the discontinuity appears. In section 3.2 we will make plausibility arguments for these various kinds of phase transitions using our numerical results.

### 3.1.2 The Ising model.

We now introduce an instructive model known as the Ising model. In general this simplest of all models of a ferromagnetic system has provided both a pedagogically sound manner in which many thermodynamic quantities may be introduced and has provided a simple model on which new techniques can be tested. A general model of a ferromagnetic system consists of a collection of spins $\{S_i\}$ which may take on values in $\mathbb{R}^3$ and are constrained to lie on the vertices of a graph $G$. Normally we consider the graph to be $\mathbb{Z}^d$ for a system in $d$ dimensions. Furthermore, in many prevalent models the spins are constrained to ‘up’ and ‘down’ spins and they are demoted to scalar
quantities \( \{S_i\} \) taking on values of 1 and -1 respectively. We can see an example of such a spin system in figure 4. The general model of magnetic spins generally allows for arbitrary coupling between spins with a Hamiltonian given by,

\[
-H_\omega = \sum_{i \in \omega} H_i S_i + \sum_{i,j \in \omega} J_{ij} S_i S_j + \sum_{i,j,k \in \omega} K_{ijk} S_i S_j S_k + \cdots
\]  

where \( H_i, J_{ij}, \) and \( K_{ijk} \) are the various couplings between the spins. We further let \( \omega \) denote the collection of spins composing the system. The Ising model consists of 'up' and 'down' spins with the assumption that only coupling to the external field, \( H \), and nearest neighbor coupling is considered. Furthermore, the assumption is made that the field is constant and further that the nearest neighbor couplings are the same for all spins. Thus, the Hamiltonian for the Ising model will be,

\[
-H_\omega = H \sum_{i \in \omega} S_i + J \sum_{\{i,j\}} S_i S_j
\]
where \((i,j)\) denotes \((i,j)\) pairs that are nearest neighbors.

Ferromagnetic systems and, in particular, Ising systems feature phase transitions at a critical temperature, \(T_c\), in the absence of an external magnetic field when the dimension of the system is greater than one. A sense of this transition may be gleaned from figure 5. Above the critical temperature, the system is statistically symmetric under the exchange of up and down spins, \(S_i \leftrightarrow -S_i\); this symmetry implies that the system must experience no net magnetization. At criticality, as the entropy of the system decreases, the spins become increasingly correlated and clusters form. Finally, below criticality the symmetry is broken and the system acquires a net magnetization.

To characterize the phase transition we can define the average magnetization of the system by,

\[
m(T, h) = \frac{1}{N} \sum_{i \in \omega} \langle S_i \rangle
\]  

(23)

where \(N\) denotes the number of spins in the system. Furthermore, we may write the partition function for the Ising system as,

\[
Z = \sum_{\{S\}} \left( \prod_{i \in \omega} \exp(h \beta S_i) \right) \left( \prod_{(i,j)} \exp(J \beta S_i S_j) \right)
\]  

(24)

where the sum is over all possible spin configurations. It follows that the mean magnetization may
be alternatively expressed as

\[
m(T, h) = \frac{1}{N} \sum_{\{S\}} \left( \sum_{i \in \omega} S_i \right) P(\{S\})
\]

(25)

\[
= \frac{1}{N \sum_{\{S\}} \left( \sum_{i \in \omega} S_i \right) \left( \prod_{i \in \omega} \exp(h\beta S_i) \right) \left( \prod_{\{i,j\}} \exp(J\beta S_i S_j) \right)}
\]

(26)

\[
= \frac{1}{N \beta Z} \frac{\partial Z}{\partial h}
\]

(27)

\[
= -\frac{1}{N} \frac{\partial G}{\partial h}
\]

(28)

where \( P(\{S\}) \) is the probability of the system being in the state \( \{S\} \) and \( G \) is a free energy known as the Gibbs free energy and is given by \( G = k_B T \log Z \). The behavior of the magnetization of the system and the critical surface may be seen in Figure 6. We can see that when \( h = 0 \) as we approach

![Figure 6](image_url)

Figure 6: On the left, the critical surface in \((h, T)\) space. On the right \( m(T, h) \) is shown as the critical point \( T_c \) is crossed by varying the temperature at \( h = 0 \) in bold and for nonzero values of \( h \) in light shading. Adapted from [33].

the critical point \( T_c \) by varying temperature, the magnetization will feature a discontinuity in its derivative. From the definition of magnetization derived above it follows that this is a discontinuity in the second derivative of a free energy, and so this phase transition will be continuous. We will not discuss the phase transition for varying external field, however this phase transition is first order. Though we will not discuss it mathematically this first order transition is very relevant to our jamming work which follows.
3.1.3 Scaling behavior and universality.

We proceed to discuss a crucial phenomenon in a similar manner to [33], focussing on the Ising model. Figure 4 shows that nearby spins tend to align for sufficiently low temperatures. This important characteristic may be described by the degree to which two spins in the lattice are correlated. This relationship is defined as a correlation function in the usual statistical manner,

\[ G(i, j) = \langle S_i S_j \rangle - \langle S_i \rangle \langle S_j \rangle = \langle S_i S_j \rangle \]  

as the mean spin will be zero by construction when \( h = 0 \). It is shown in [39] that,

\[ G(i, j) \approx r_{ij}^{-d+2-\eta} \exp(-r_{ij}/\xi) \]  

near criticality in the absence of an external magnetic field. In this case \( r_{ij} \) is the distance between the \( i \)'th and \( j \)'th lattice sites, \( \xi \) is called the correlation length, \( d \) is the spacial dimension, and \( \eta \) is a critical exponent. The correlation length is a measure of the characteristic distance between sites that must be established before their spins become uncorrelated. If two spins are located a distance greater than \( \xi \) apart one may in general say that they will behave in a relatively uncorrelated manner. Conversely if two spins are located a distance less than \( \xi \) apart one may, in general, claim that their behaviors will be relatively correlated\(^{17}\).

For systems that exhibit continuous phase transitions, many thermodynamic quantities are characterized by power-law scaling sufficiently close to the critical point. In particular for the Ising model the quantities, \( \xi \) and \( m \), scale as,

\[ \xi \sim |\tau|^{-\nu} \quad \text{and} \quad m(T, 0) \sim \tau^\beta \]  

in the case of zero external field. Here \( \tau \) is given by,

\[ \tau = \frac{T_c - T}{T_c} \]  

\(^{17}\)Two sites are relatively correlated if given the knowledge of one spin predictions can be made about the state of the other spin.
where $T_c$ is the critical temperature at which the phase transition occurs. As will be subsequently discussed $\xi$ features a divergence as criticality is approached. Thus, close to the critical point it follows that $\exp(-r_{ij}/\xi) \approx 1$ for finite $r_{ij}$. Therefore, in systems that obey (30) it follows that near $T_c$ the correlation function will also feature power-law scaling as,

$$G(i,j) \approx r_{ij}^{-d+2-\eta}. \quad (33)$$

In cases where power-law scaling behavior is found, the thermodynamics of the material about the critical point are characterized by these exponents. Mean field theory\footnote{Mean field theory is a common first order approximation in Statistical Mechanics. Given a system $\omega$ featuring interactions, mean field theory is an approximation that smooths the details of these interactions to an overall interaction with a field. In the case of the Ising model we neglect the various spin-spin interactions and replace them with the interactions between the spins and an internal ‘mean’ field given by $B \propto n\langle S \rangle$. This allows for an analytic theory in many cases. \cite{40, 32}} predicts that $\nu = 1/2$ and $\beta = 1/2$. The different scaling characteristics may be seen in figure 7. Consider the correlation length, $\xi$, and note that it diverges as $T \to T_c$. This divergent length scale is primarily responsible for the creation of ferromagnetic domains at low temperatures as increasingly large portions of the lattice begin to affect one another. These increasingly large ferromagnetic domains spontaneously break the $S_i \leftrightarrow -S_i$ symmetry of the system causing a net magnetization. The net magnetization features no such divergence but instead spontaneously increases from 0 at $T = T_c$. The emergence of this net magnetization is reflected in the critical behavior of $m(T, 0)$.

![Figure 7: Scaling schemes for $\xi$ and $m(T, 0)$ near the critical point $T_c$.](image)

It was shown by Kadanoff \cite{41} that in many systems although the critical point specifically
depends on the details of the interaction and the type of lattice, the critical exponents are “uni-
versal” in that they depend on more corse-grained fundamental properties of the system like the 
spatial dimension, $d$. In this way Kadanoff reduced the large variety of phase transitions to a few 
equivalence classes\(^{19}\) known as universality classes. Systems in the same universality class exhibit 
the much of the same physics and have numerically identical critical exponents; for example many 
different materials go about liquid-solid phase transitions and, despite the fact that the various 
transitions occur at different points the behavior of all of these transitions is the same. Although 
this is an intuitively satisfying example there are many more striking examples of universality 
classes appearing in nature. For instance, the Ising universality class includes the the liquid-gas 
transition at the end points of the coexistence line of first-order transitions separating the liquid 
from the gaseous state \([33]\), binary mixtures of liquids, and order-disorder phase transitions.

The utility of universality is clear. If the universality class of a given continuous phase transition 
can be determined then all of its characteristic features may be referred from solved systems in the 
same universality class. Thus, universality classes hold a position in modern statistical mechanics 
that is similar to the position of isomorphisms in mathematics: they allow a vast array of phenomena 
to be reduced to the study of a few classes of phenomena. Although there is evidence that the glass 
transition exhibits features indicative of a true mixed phase transition, the notion of universality 
classes may still be applied.

### 3.2 Percolation theory.

At the core of this research is percolation theory. Percolation theory has been used to model 
forest fires, the distribution of oil and gas through a porous media, and diffusion in a disordered 
media. In each of these cases, percolation theory has found success because, like the Ising model, 
it is a simple enough model to admit elegant and, in many cases, analytic mathematical solutions 
while capturing enough physics to yield useful quantitative results. Another boon to percolation 
theory is that the various models easily admit many computational techniques. There are two areas 
of percolation theory: uncorrelated and correlated. As was discussed in the introduction, recent

\(^{19}\)An equivalence class is a subset that is induced by an equivalence relation $\sim$. The crucial aspect of equivalence 
classes is the fact that two equivalence classes are either disjoint or equal.
studies [25, 34, 35] have suggested that systems in correlated percolation theory share characteristics found in systems that exhibit jamming behavior. As percolation theory is generally more accessible both analytically and computationally than analogous physical systems such similarities are often leveraged to great effect. We will discuss percolation theory, beginning with uncorrelated systems as a stepping stone to the correlated systems in question. In addition to being a natural segway to jamming percolation, we used uncorrelated percolation in this research as a familiar environment on which to test our code, since results in this field are well known.

We begin by introducing uncorrelated percolation theory following the paradigm in [31]. In general there are two distinct types of percolation namely bond percolation and site percolation. In this study we focus exclusively on the latter but note that in a few cases it is isomorphic to the former. Percolating systems consist of a graph $G$ on which certain vertices are specified as being either occupied or unoccupied. Although the there are a variety of ways in which these sites can be occupied, ordinary percolation consists of doing it at random with probability $p$. The parameter $p$ is the sole degree of freedom in percolation theory. We define clusters as groups of occupied sites that may be reached by traveling on bonds that traverse only occupied vertices. In this thesis our computations are restricted to the square lattice, $\mathbb{Z}^2$. This situation can be seen in figure 8.

Figure 8: Percolation over $\mathbb{Z}^2$. The circles represent the vertices of the graph. The circles colored either red or white represent occupied sites. The circles colored red identify part of a cluster.
The study of these percolating systems is largely the study of the connected clusters. The cluster number, \( n_s(p) \), is defined as the number of clusters, per lattice site, containing exactly \( s \) sites. Thus, the probability of finding a cluster with \( s \) sites is \( n_s s \). It has been conclusively shown that percolation systems exhibit a discontinuity in \( n_\infty \) at some critical point \( p_c^{20} \). Even more generally we find that \( n_s \) exhibits scaling of the form,

\[
n_s \sim s^{-\gamma} f((p - p_c)^{\sigma})
\]  

(34)
sufficiently close to \( p_c \). Here \( f \) is a scaling function and \( \gamma \) and \( \sigma \) are critical exponents. It can also be shown that this scaling form is general and that the function \( f \) along with the parameters \( \gamma \) and \( \sigma \) are universal, depending only on dimension and not on the specific structure of the lattice. Finally, it is noted in [31] that it is nearly always the case that \( f(z) \approx \) constant for \( |z| \ll 1 \) while \( f(z) \to 0 \) for \( |z| \gg 1 \).

As with the Ising model an important quantity in percolation theory will be the correlation function \( g(r) \). We define the correlation function to be the probability that a site a distance \( r \) from an occupied site is also occupied and resides in the same cluster. The average number of sites to which a site at the origin is connected is then \( \sum g(r) \). The correlation length or connectivity length is then defined to be a measure of the average distance between two sites belonging to the same cluster. It is given by,

\[
\xi^2 = \frac{\sum_{r \in \omega} r^2 g(r)}{\sum_{r \in \omega} g(r)}.
\]  

(35)

It can be shown that \( \xi^2 \) may be alternatively written as,

\[
\xi^2 = \frac{2 \sum_{s} R_s^2 s^2 n_s}{\sum_{s} s^2 n_s}
\]  

(36)

\(^{20}\)This discontinuity in \( n_\infty(p) \) implies that the system spontaneously acquires an infinite cluster. It can be shown that the probability of an infinite cluster forming in uncorrelated percolation occurs with probability 0 below \( p_c \) and with probability 1 above \( p_c \).
where $R_s$ is the radius of gyration and is defined by

$$2R_s^2 = \sum_{ij} \frac{|r_i - r_j|^2}{s^2}.$$ \hspace{1cm} (37)

It turns out that generally we expect the length scale to feature power law divergence similar to the Ising model with temperature replaced by probability, $\xi \propto |p - p_c|^{-\nu}$.

In particular, on $\mathbb{Z}^2$, standard percolation features a continuous phase transition in cluster size with respect to occupation probability $p$. At the critical probability $p_c$ the probability of attaining an infinite cluster changes discontinuously from 0 to 1 in the infinite system limit. This may be seen in the divergent length scale, $\xi$, above. In figure 9 we see a demonstration of this fact on a periodic finite size system.

![Figure 9: Two randomly generated finite lattices with periodic boundary conditions and (a) $p < p_c$ and (b) $p > p_c$. Occupied sites are in color while unoccupied sites are black. Red sites denote sites that have not been identified with a cluster. Colored sites that are not red denote identified clusters. The marked difference in cluster size is readily apparent.](image)

### 3.2.1 Spiral percolation

In correlated percolation the probability of a given site being occupied is coupled with the probability that the sites in some neighborhood are occupied. This coupling can be achieved in a number
of ways. In this work we focus on achieving coupling by generating an uncorrelated lattice and then eliminating sites that fail to satisfy some constraints based on the conditions of their neighbors. This process is performed iteratively until either no sites on the lattice remain occupied or every remaining lattice site satisfies this constraint.

The canonical example of a system studied in correlated percolation is known as k-core, or bootstrap, percolation. First proposed in 1979 by Chalupa et al. in [42], k-core percolation introduces correlation by requiring that occupied sites have at least \( k \) occupied neighbors for some integer \( k \). In the 1979 work, the notion of bootstrap percolation was introduced in the context of the Blume-Capel model\(^{21}\). It was found that the k-core model produced results that agreed with those predicted by the Blume-Capel model. In 2006 it was shown by Schwarz, Liu, and Chayes in [25] that k-core percolation features a mixed phase transition on the Bethe lattice\(^{22}\). Perhaps equally important, it was shown that also on this lattice, k-core percolation featured critical exponents that agreed to a high degree with those found in the jamming phase transition at point \( J \).

In this work we turned to study a more contemporary model coined the spiral model. The spiral model was first introduced in 2007 in [34] as a model with “trivial interactions but constrained dynamics”. The spiral percolation model is used in modeling granular media. As such its culling strategy attempts to incorporate a notion of static equilibrium, culling active sites that are not supported. To accomplish this the spiral cull step considers neighbors and next nearest neighbors. These sites are grouped into North West (NW), South West (SW), South East (SE), and North East (NE) as shown in figure 10. Active sites are then culled if neither NW and SE, nor, NE and SE contain active sites. This forces active sites to have a spatial distribution of active neighbors in order to stay alive.

It was proven in [34] that the spiral model features several interesting properties. First and foremost it was shown, in finite dimensions, that the spiral model features a mixed phase transition

\(^{21}\)The Blume-Capel model, introduced in 1966 [43, 44], is the simplest representation of crystal field effects. It takes the form of an Ising model where spins may take on values in the triplet \( \{-1, 0, 1\} \) as opposed to the doublet \( \{-1, 1\} \).

\(^{22}\)The Bethe lattice in percolation theory is a graph with no loops on which many quantities may be computed exactly. Because the lattice has no loops, it is often thought of as the infinite dimensional limit. Therefore, results computed for the Bethe lattice in percolation theory are often thought of by analogy to those computed by mean field theory for the Ising model.
with the fraction of stable sites featuring a discontinuity at $p = p_c$. Although we will not prove that the fraction of occupied sites takes the form of (20) we do present a plausibility argument using data collected in the course of conducting this research. We let $p$ denote the probability of occupancy of the lattice and $P$ denote the fraction of stable sites after all unstable sites have been removed. We can see how the fraction of stable sites depends on the occupancy in figure 11. As may be seen in the figure, the fraction of stable sites increases in two steps at all system sizes. It
can be seen that the initial ascent grows increasingly sharp with respect to system size while the second increase is similar to the power law scaling seen for magnetization in figure 7.

The second feature of the spiral model that was proven in [34] is that the connectivity length, $\xi$, features exponential as opposed to power law scaling near the critical probability. Specifically it was found that the length scale diverges exponentially as

$$\xi \sim \exp(|p - p_c^\infty|^{-\mu}).$$  \hspace{1cm} (38)

Hence, any pursuit of universality must attempt to discern the superexponent $\mu$ as opposed to $\nu$. It must be stressed that this super power law scaling is highly unusual. It follows that we have no reason to assume that many of the techniques used to study standard percolation, in particular the renormalization group, will be applicable to this situation. However, as will be seen in section 5 there is evidence that the renormalization group does apply to this situation. Furthermore, the super power-law scaling form implies that $\xi$ will diverge much more quickly than for power law scaling as the system approaches criticality.

The final result from [?] was made by analogy to directed site percolation\(^{23}\). By relating the spiral model to the directed site percolation universality class, Fisher et al. were able to analytically determine values for the critical exponent $\mu$ as well as the critical probability $p_c$. In particular they found that,

$$p_c = p_c^{DP} \approx 0.705$$  \hspace{1cm} (39)

where $p_c^{DP}$ is the critical probability for directed percolation. They also found that

$$\mu = (1 - \zeta)\nu_\parallel \approx 0.64$$  \hspace{1cm} (40)

where $\zeta$ and $\nu_\parallel$ are critical exponents of directed percolation.

We can see a sample spiral percolation system as well as several of the culling steps needed to

---

\(^{23}\)The definition of directed percolation is similar to the definition of standard percolation except that directionality is associated with the bonds[33]. Thus, directed percolation is no longer an isotropic process. Directed percolation represents another method of introducing correlation into a percolation problem. The directed percolation universality class is one of the most well known of the universality classes.
realize this configuration in figure 12. We would direct the interested reader to the accompanying

Figure 12: The creation of a system satisfying the constraints of spiral percolation. The spacial heterogeneities are especially evident. Generated with auxiliary code to produce graphics.

video to more readily see the culling process. We can see in the figure that the spiral percolation model features distinct spacial heterogeneities as a result of the culling strategy having been applied. We further note that the entire system is connected.
3.2.2 Force-balance percolation

The second class of models that we studied, known as the force balance models, were introduced in [25]. The force balance models are a class of models constructed in the same paradigm as the spiral model. Each of the force-balance models incorporates the notion of mechanical stability into the definition and, likewise, fails to support finite clusters. In this thesis we studied a particular force-balance model known as the 24 Nearest Neighbor (24NN) model. The rule for this model, depicted in figure 13, considers the 24 nearest neighbors. In particular we require that one site be occupied in each of the regions A, B, C, and D for a total of three occupied sites amongst the 24 nearest-neighbors.

Unlike for the spiral model, there have been few analytic results proven for the 24NN model. In [37] it is proven that the 24NN model does feature a critical point \( p_c < 1 \). Furthermore, in the same source it is shown that the 24NN model shares many qualitative similarities with the spiral model. In particular there is evidence in [37, 36] that the 24NN model also features exponential

![Figure 13: Pairing of nearest neighbors in the 24NN model.](image)

...
scaling and a discontinuity in the fraction of occupied sites. Distinguishing the 24NN model from the spiral model, we note that the former features a critical point at a much lower probability of about 0.425 as seen in [37]. This makes sense qualitatively since the 24NN model features a larger neighborhood than the spiral model. We can see an image of the 24NN model in figure 14.

Figure 14: Pairing of nearest neighbors in the 24NN model.

3.3 Renormalization group theory.

We begin with a general overview of the renormalization group to frame the more technical discussion to come. Before discussing the method itself it is useful to clarify some misconceptions that may stem from the terminology. The first thing to be addressed is the use of the word “group”; despite its use in the title, the mathematical machinery used with the renormalization group has nothing to do with group theory. Furthermore, renormalization, in the quantum field theory sense, is not directly connected with the more general methods of the renormalization group as they stand today. However, the first uses of the theory were to the renormalization of electrodynamics to remove infinities and it took some time before the methods were found, by K. Wilson, to generalize to many other areas of physics. Finally, the use of the definite article ‘the’ is also misleading, as
the renormalization group is really a framework for solving physical problems. In short, this could be the most inappropriately named theory in all of physics.

The basic idea of the renormalization group, in its most general form, is to express the parameters of a problem in terms of new set of parameters that are favorable while preserving the physics of interest. In the case of statistical mechanics one usually generates a set of recursive equations called the renormalization group flow equations. These equations can be thought to describe the dynamics of the system in parameter space under repeated application of the renormalization group flow equations. These dynamics, as the number of applications of the renormalization group flow equations becomes infinite, are said to correspond to the renormalization group flows of the problem in question. These renormalization group flows are important in that singularities (with respect to final destination) in the renormalization group flows correspond to phase transitions in the system. One important consideration when solving problems with the renormalization group method is that there is not a unique way to apply the methods, and hence not a unique set of renormalization group flow equations.

This presentation will be organized as follows: we will first consider the examples of the ising model in one and two dimensions to illustrate the method on a familiar problem and then we will consider the renormalization group method in general.\textsuperscript{24}

3.3.1 1D Ising model

We begin our study with the one-dimensional Ising model for several reasons: because the model is soluble analytically, because we are familiar with the result, and finally because if this theory is to represent an improvement on mean field theory it should at least succeed in this case. The partition function for the one dimensional Ising model is given by,

\begin{equation}
Z = \sum_{s_1, s_2, \ldots, s_N = \pm 1} \exp[K(s_1 s_2 + s_2 s_3 + s_3 s_4 + \cdots)]
\end{equation}

\textsuperscript{24}There is a huge amount of material on the renormalization group and it would be infeasible to hope to cover any substantial portion of the theory here. For more information, the interested reader should refer to the books mentioned in the bibliography.
where \( K = J/k_B T \) is the nearest neighbor spin coupling as a function of temperature. We are going to simplify this system by removing degrees of freedom by explicitly evaluating the sum over some subset of the \( s_i \)'s. We note that the partition function may be expressed as,

\[
Z = \sum_{s_1, \cdots, s_N = \pm 1} \cdots \exp[K(s_1 s_2 + s_2 s_3)] \exp[K(s_3 s_4 + s_4 s_5)] \cdots .
\]  

(42)

As each even \( s_i \) appears in only one exponent we may compute the sum explicitly to find,

\[
Z = \sum_{s_1, s_3, \cdots} \{\exp[K(s_1 + s_3)] + \exp[-K(s_1 + s_3)]\} \times \{\exp[K(s_3 + s_5)] + \exp[-K(s_3 + s_5)]\} \times \cdots .
\]  

(43)

Following the renormalization group paradigm, we would really like to view this as a transformation in the parameter space \( \{K\} \). Hence, we would like to find some new set of coupling constants \( \{K'\} \) (potentially between spins of arbitrary separation) such that we may write the above as a partition function. In this case we may do this with just one coupling constant \( K' \) coupling nearest neighbors in this transformed scheme; however, as we will see, in general any set of couplings could be nonzero.

We therefore seek a solution to the equation,

\[
e^{K(s+s')} + e^{-K(s+s')} = f(K) e^{K's's'}
\]  

(44)

for all \( s, s' = \pm 1 \). If we can find the solution to this set of equations then we will have successfully halved the number of degrees of freedom and transformed the partition function,

\[
Z = \sum_{s_1, s_3, \cdots} f(K) \exp(K's_1 s_3) f(K) \exp(K's_3 s_5) \cdots = [f(K)]^{N/2} Z'.
\]  

(45)

This transformation is called a Kadanoff transformation. To find \( K' \) and \( f(K) \) we must consider the solutions to (44) for the different possible \( s \) and \( s' \). We note that if \( s = s' = \pm 1 \) then,

\[
e^{2K} + e^{-2K} = \cosh(2K) = f(K) e^{K'}
\]  

(46)
and if \( s = -s' = \pm 1 \) then we will have,

\[
f(K) = 2e^{K'}.
\]  

(47)

We may solve these equations by plugging in for \( f(K) \) to find that,

\[
K' = \frac{1}{2} \ln \cosh(2K) \quad \text{and} \quad f(K) = 2 \cosh^{1/2}(2K).
\]  

(48)

Now consider the equation \( \ln Z = Ng(K) \), which is proportional to the Gibbs free energy of the system. Since free energies are extensive, it follows that \( g(K) \) must be intensive. Applying the recurrence relation to \( Z \) we have that,

\[
\ln Z = \frac{N}{2} \ln f(K) + \ln Z'
\]  

(49)

and hence,

\[
g(K) = \frac{1}{2} \ln f(K) + \frac{1}{2} g(K'),
\]  

(50)

or finally, plugging in for \( f(K) \),

\[
g(K') = 2g(K') - \ln[2 \sqrt{\cosh(2K)}].
\]  

(51)

Together, equations (48) and (51) constitute the renormalization group flow equations for this system.

To analyze the 1-dimensional Ising model renormalization group flow equations we begin by noting that if \( K \rightarrow 0 \) then \( K' \rightarrow 0 \) and if \( K \rightarrow \infty \) then \( K' \rightarrow \infty \). We may identify two fixed points, one at \( K = \infty \) and one at \( K = 0 \); the fixed point at \( K = \infty \), or \( T = 0 \) is called an unstable fixed point since any perturbation of \( K \) will cause it to be renormalized down to \( K = 0 \). Furthermore, \( K = 0 \) corresponds to a stable fixed point as a state with \( K < \infty \) will be renormalized down to \( K = 0 \). We say that the renormalization group flows go from the unstable fixed point to the stable one. This is seen in figure 15.
In summary, the only phase transition given by the renormalization group, which is confirmed by exact calculation, is one at $T = 0$. All higher temperature states are paramagnetic. Only at $T = 0$ is there true long range order and a finite magnetization at arbitrarily small fields. We further note that the renormalization group flow is away from the nontrivial fixed point. This will be the same in the two dimensional case, as we will see in the next section.

### 3.3.2 2D Ising model

We now investigate the more complicated case of the two-dimensional Ising model. In this case we will no longer be able to solve the problem analytically and will have to make approximations. However, this case is also more interesting because the renormalization group flow is over a two dimensional parameter space and yields our first example of a discontinuity. We proceed in a similar manner to the one-dimensional Ising model, however in this case we remove half of the spins on the square lattice. This process can be seen in figure 16. In this case we notice that after removing the spins the geometry has changed and a rotation by $\pi/4$ has been introduced along with the
rescaling of the lattice. This gives the first hint that the situation may be more complex than in the one-dimensional case. In a similar manner to the one dimensional Ising model we seek a set of transformations that will rescale the partition function while transforming it from one set of couplings \( \{K\} \) to another set, \( \{K'\} \). However, it can be shown that in this case more than one parameter will need to be introduced. We can therefore find a set of renormalization group flow equations,

\[
\begin{align*}
K_1 &= \frac{1}{4} \ln \cosh(4K) \\
K_2 &= \frac{1}{8} \ln \cosh(4K) \\
K_3 &= \frac{1}{8} \ln \cosh(4K) - \frac{1}{2} \ln \cosh(2K) \\
f(K) &= 2[\cosh(2K)]^{1/2}[\cosh(4K)]^{1/8}.
\end{align*}
\]

We may now find the renormalized partition function to be,

\[
Z = \sum_{N\text{ spins}} \exp \left[ K \sum_{ij} s_is_j \right]
= [f(K)]^{N/2} \sum_{N/2\text{ spins}} \exp \left[ K_1 \sum_{ij} s_is_j + K_2 \sum_{lm} s_ls_m + K_3 \sum_{pqrt} s_ps_qs_rs_t \right]
\]

where the doubly primed sum is over next nearest neighbors and the triple primed sum is over all sets of neighboring four spins around the square.

The renormalization group calculations may not be performed exactly in this case and to proceed we must make some approximations. We note that if we neglect both \( K_2 \) and \( K_3 \) then the situation reduces to the situation of the one dimensional Ising model. So, we will neglect \( K_3 \) and assume that the coupling of sets of four spins is negligible. We will make the further approximation that the coupling between next nearest neighbors can be regarded as a perturbation of the coupling constant for nearest neighbor interactions. Thus we make the approximation,

\[
K_1 \sum_{ij} s_is_j + K_2 \sum_{lm} s_ls_m + K_3 \sum_{pqrt} s_ps_qs_rs_t \approx K'(K_1, K_2) \sum_{ij} s_is_j.
\]
This approximation gives the renormalization relation,

$$\begin{align*}
Z(K, N) &= [f(K)]^{N/2} Z(K'(K_1, K_2), N/2).
\end{align*}$$

(58)

Letting $$g(K) = N^{-1} \ln Z(K, N)$$ be the free energy per spin we have the familiar equations,

$$g(K) = \frac{1}{2} \ln f(K) + \frac{1}{2} g(K')$$

(59)

or in this case, plugging in from equation (55),

$$g(K') = 2g(K) - \ln\{2[\cosh(2K)]^{1/2}[\cosh(4K)]^{1/8}\}. \quad (60)$$

We can proceed to estimate $$K'$$ by considering the energy of the system when all the spins are aligned. Since there are $$N$$ nearest-neighbor bonds in a two-dimensional cubic lattice with $$N/2$$ spins, and there are $$N$$ next-nearest neighbor bonds, too,

$$\begin{align*}
K_1 \sum_{ij} s_i s_j &= NK_1, \\
K_2 \sum_{lm} s_l s_m &= NK_2
\end{align*}$$

(61)

when all the spins are aligned. It is then claimed that we may therefore estimate $$K' \approx K_1 + K_2$$ and so $$K' = 3/8 \ln \cosh(4K)$$. The important point about this renormalization group flow equation is that it has a nontrivial fixed point, or a finite $$K_c$$ such that,

$$K_c = \frac{3}{8} \ln \cosh(4K_c).$$

(62)

We can find using numerical methods that $$K_c = 0.50698$$. In this case the renormalization flow is broken into two paths diverging from $$K_c$$. This can be seen in figure 17. We now note that the heat capacity will be given by,

$$C = \frac{d^2}{dK^2} g(K).$$

(63)

Taking a taylor expansion of $$g(K)$$ about $$K_c$$ we can find that $$C$$ diverges as $$K \to K_c$$ according to
the power law,
\[ C \propto |T - T_c|^{-\alpha} \]  
(64)

where \( T = (J/k_B K) \) and \( \alpha = 0.131 \). Thus we have associated the fixed point \( K_c \) with a phase transition. The critical temperature is defined by,
\[ \frac{J}{k_B T_C} = 0.50698 \text{ or } \frac{k_B T_C}{J} = 1.97246 \]  
(65)

which is close to the exact value of,
\[ \frac{J}{k_B T_C} = 0.44069 \text{ or } \frac{k_B T_C}{J} = 2.26917 \]  
(66)

as we saw previously.

### 3.3.3 Renormalization in percolation theory.

We have demonstrated the power of the renormalization group on the Ising model. In applying the renormalization group methodology to the Ising model it was crucial to be able to use the hamiltonian since this is where the rescaling took place. However, we note that in the case of percolation theory we have no such hamiltonian to rescale. Instead, we must find another way to use rescaling to extract properties of the system. In the forthcoming discussion we follow the derivations in [31, 46]. The crucial property in percolation theory that is used in all renormalization group calculations are the scaling relationships of the different quantities under consideration. In the forthcoming derivations we consider standard percolation. The exponential, as opposed to power-law, scaling relationships greatly complicate the derivations. We instead make an anzats about the exponential scaling form and show in the results section that this anzats is well fit by
In section 3.2 we discussed scaling of quantities in the infinite system limit. In the same section it was found that a host of different quantities feature power-law scaling as, \(|p - p_c|^{-x}\). In particular we recall that the correlation length features scaling of the same form with \(\xi = |p - p_c|^{-\nu}\). However, such scaling relationships were derived only for systems that are infinite in extent. It can be shown that if a quantity \(X\) features power-law scaling in the infinite system limit then for finite systems it will feature scaling of the form,

\[
X(L, \xi) \propto \begin{cases} 
\xi^{x/\nu} & L \gg \xi \\
L^{x/\nu} & L \ll \xi
\end{cases}
\]  

(67)

where \(L\) is the size of the system. Recalling that \(\xi\) features power-law scaling this relationship implies that for systems whose length-scale greatly exceeds the correlation length, the scaling converges to power-law scaling. However, when the system size is smaller than the correlation length, the scaling is controlled by \(\xi\). It is this dependence on system size that we will exploit to implement a renormalization group scheme for percolation systems.

In a similar vein to the size-dependent scaling relation given in 67 we consider how the notion of an “infinite cluster” gets adapted to a finite system. In particular we replace the notion of infinite cluster by the notion of a spanning cluster, or a cluster that reaches from the top of the lattice to the bottom of the lattice. Let \(R(p)\) denote the probability that a spanning cluster exists on a lattice occupied with probability \(p\). In the infinite system limit we note that \(R(p) = 1\) for \(p > p_c\) and \(R(p) = 0\) for \(p < p_c\). Thus,

\[
R(p) = \Theta(p - p_c)
\]  

(68)

where \(\Theta(x)\) is the Heaviside function. As with the above scaling relationships, this relationship will change for a finite system of size \(L\) since there will, in general, be a finite probability of achieving a spanning cluster at any \(p\). In the case of a very large, but sub-infinite system it is given in [31] that we write,

\[
R(p) = \Phi[(p - p_c)L^{1/\nu}]
\]  

(69)
where $\Phi$ is a monotonically increasing smooth function such that $\Phi(-\infty) = 0$ and $\Phi(\infty) = 1$.

We now construct the scaling relationship that is crucial to applying renormalization group calculations to percolation systems. Consider the derivative,

$$\frac{\partial R}{\partial p} = L^{1/\nu} \Phi'(p - p_c) L^{1/\nu}. \quad (70)$$

We may define the average probability, $p_{av}(L)$, at which spanning clusters begin to form by the equation,

$$p_{av}(L) = \int p \left( \frac{\partial R}{\partial p} \right) dp. \quad (71)$$

Expanding this equation we find that,

$$p_{av}(L) = \int p \left( \frac{\partial R}{\partial p} \right) dp = \int L^{1/\nu} p \Phi'[(p - p_c)L^{1/\nu}] dp$$

and making the substitution $z = (p - p_c)L^{1/\nu}$ we find that

$$p_{av}(L) = L^{-1/\nu} \int (z + p_c L^{1/\nu}) \Phi'(z) dz$$

$$= L^{-1/\nu} \int z \Phi'(z) dz + p_c \int \Phi'(z) dz$$

$$= L^{-1/\nu} \int z \Phi'(z) dz + p_c$$

where we have used the fact that $\int \Phi'(z) dz = \Phi(1) - \Phi(0) = 1$. It follows that

$$p_{av}(L) - p_c \propto L^{-1/\nu}. \quad (72)$$

This is the equation from which a renormalization group computation on percolation systems begins: if one can find $R(p)$ for systems of a variety of sizes then it is possible to determine the critical exponent $\nu$ and the critical probability $p_c$.

We derived equation (72) for the quantity $p_{av}(L)$, defined by equation (71). However, it is given
in [31] that the scaling form (72) holds for many quantities that are reasonably acquired from \( R(p) \).

In particular we may define the quantities \( p_c(L) \) by \( p_c(L) = R(p_c(L)) \) and \( p_{1/2} \) by \( R(p_{1/2}) = 1/2 \).

We now apply the paradigm of the renormalization group to percolation theory. Take a very large system, \( \omega \), of size \( \Lambda \times \Lambda \) and occupy the system with probability \( p \). We split the system into regions of size \( L \times L \). The system may then be rescaled to form a system \( \omega' \) by letting each region of length \( L \) in \( \omega \) become a single site in \( \omega' \). This new site is said to be occupied if the region from which it was derived percolated\(^{25}\). The new system \( \omega' \) will have length \( \Lambda/L \). We may see this process in figure 18. We note that the occupancy of the rescaled system, \( \omega' \) will be equal to the fraction of \( L \times L \) regions that percolate. By definition this is just given by \( R(p) \) as constructed above. However, we are now able to give some interpretation to \( R(p) \). In particular, we note that \( R(p) \) is the renormalization group flow. Furthermore, we note that \( p_c = R(p_c) \) will be a fixed point for the system and, since \( R(p) \) is monotonically increasing, a lattice occupied with \( p < p_c \) will be rescaled to an empty lattice while a lattice occupied with probability \( p > p_c \) will be rescaled to a fully occupied lattice. The renormalization group flow for an infinite and sub-infinite system can be seen in Figure 19. We may therefore apply the tools of renormalization group theory to determine the critical exponents and critical points. In particular we do so by finding \( p_c(L) \) (or later, as will be discussed, \( p_{1/2}(L) \)) for various \( L \) and then fitting to equation (72).

\[
p_c(L) = R(p_c(L)). \tag{73}
\]

In this particular case it is possible to find a explicit form for \( R(p) \); we present this argument subsequently. Let \( n \) denote the number of occupied sites in an \( L \times L \) lattice. Trivially, \( n \) must reside between 0 and \( L^2 \). Further let \( S_n \) denote the fraction of \( L \times L \) lattices with \( n \) occupied sites that percolate. If the large system, \( \omega \), is occupied with probability \( p \) it follows that each region will similarly be occupied with probability \( p \). So we pose the question: given a region occupied with probability \( p \), what is the probability that it will have \( n \) occupied sites? This is equivalent

\(^{25}\)In this study a site was occupied only if the region used to generate it percolated vertically. Although one could look for horizontal percolation or horizontal and vertical percolation all three will yield the same results in the infinite system limit since percolation is an isotropic process.
Figure 18: An example of the renormalization group flow for standard percolation. In the subinfinite regime the fraction of lattices that percolate varies smoothly from 0 to 1. However, in the infinite system limit there is a discontinuous jump in $\Pi$ at the critical probability $p_c$.

to asking for the probability of obtaining $n$ heads out of $L^2$ coin-tosses with a weighted coin. The answer to the latter problem is well known and is given by,

$$P_n(p) = \binom{L^2}{n} p^n (1-p)^{L^2-n}$$  \hspace{1cm} (74)$$

since $p^n(1-p)^{L^2-n}$ is the probability of constructing a particular sequence with $n$ heads and $\binom{L^2}{n}$ is the number of such sequences. Thus, the expected fraction of $L \times L$ lattices occupied with
Figure 19: An example of the renormalization group flow for standard percolation. In the subinfinite regime the fraction of lattices that percolate varies smoothly from 0 to 1. However, in the infinite system limit there is a discontinuous jump in $\Pi$ at the critical probability $p_c$.

The probability $p$ to have a spanning cluster will be,

$$R(p) = \sum_{n=0}^{L^2} S_n P_n(p) = \sum_{n=0}^{L^2} \left( \frac{L^2}{n} \right) S_n p^n (1-p)^{L^2-n}. \quad (75)$$

This defers our ignorance from the function $R(p)$ to the factors of $S_n$. In particular we note that the $S_n$ solely determine the renormalization group flow. This functional form also provides a natural method to compute $R(p)$ by enumerating all of the $L \times L$ lattices with $n$ occupied sites and then setting

$$S_n = \frac{\# \text{ Percolated}}{\text{Total}}. \quad (76)$$

In this thesis, a prevalent concern was the study of the correlated percolation models developed in sections 3.2.1 and 3.2.2 as opposed to uncorrelated percolation. Recall that both of these models have been demonstrated to feature exponential, as opposed to power-law, scaling of the connectivity length, $\xi$. As such the developed scaling relationship (72) is no longer valid. Furthermore, the notion of renormalization must be changed slightly although the general form found in (75) remains valid. We address the requisite changes to the formalism presently. First, instead of the asymptotic form
for $R(p)$ in (69) we present a similar form used in [37] and based on exponential scaling of $L$,

$$R(p) = \Phi[(p - p_c)(\log L)^{1/\mu}].$$

(77)

We do not rigorously show that this scaling holds, but we demonstrate that our results, in sections 5.7 and 5.8, are consistent with such a form. Given the validity of equation (77) a derivation, directly analogous to that beginning with equation (71), shows that $p_{av}(L)$ obeys the scaling,

$$p_{av}(L) - p_c \sim (\log L)^{-1/\mu}.$$  

(78)

We now bring the notion of renormalization into the context of correlated percolation. We can think about the renormalization step in the following way. We first constructing a large system $\omega$ of occupancy $p$ and then culling this system by the method outlined in section 3.2.1 or 3.2.2 until every site satisfies the stability condition. We then split this culled lattice into $L \times L$ pieces and renormalize to form the system $\omega'$. Thus, in this context the $S_n$ represent the fraction of $L \times L$ lattices with $n$ occupied sites that percolate after culling.

The alternate form for $R(p)$ in (69) is valid only in the limit of very large $L$. Thus, the scaling form (72) is valid only in the asymptotic limit where $L \gg \xi$. This may be explained by noting that during a rescaling connectivity is not preserved between the various $L \times L$ regions. Therefore, only in the limit where sites in one region are uncorrelated from sites in another region will such a rescaling preserve the connectivity of the lattice. This asymptotic limit also implies that, computationally, to extract critical quantities we must perform the rescaling of the lattice with large enough regions. Unfortunately, the exponential scaling of the spiral and 24-nearest neighbor models means that $\xi$ grows very quickly and so we must consider extremely large systems before residing in the asymptotic limit.
4 Methods

In this section we discuss the methodology and process through which we have conducted our investigation of jamming percolation. We present our work in chronological order, and attempt to show how each result lead to the subsequent work. We will often refer to code segments to complement our descriptions. It may therefore be useful to consider the code found in the appendix while reading this section.

4.1 Java simulations

We began our study of percolating systems using the Open Source Physics (OSP) libraries of Java as a testbed[47]. These libraries allowed us to quickly and visually look at different percolation models and subsequently improve our intuition about these models. Specifically, we began with a code class file PercolationApp ([47] pp. 455-457) and modified it to simulate correlated percolation. In particular we implemented k-core percolation, spiral percolation, and force-balance percolation.

Correlated percolation models were implemented using a culling function

\texttt{CullLattice()}

that can be found in section one of the appendix. The culling function reads in the culling type from the input control and then applies it to a copy of the original lattice. The culling function further decides whether it should perform a single culling step or cull until the stability condition is satisfied based on the status of “Full Cull” in the menu. The occupancy after each cull-step is then plotted in the “Occupation Probability vs Iteration” window. We were able perform culling on variable lattice sizes and then interact with the systems either by renormalizing or viewing clusters in real time. The screenshot of the application shown in Figure 20 shows panels with two lattices, output data, and a calculation control. Looking at the control, one notes that the input parameters may be specified on the fly. In addition to visualizing lattices our codes contained added functionality to follow the renormalization group flow for a given lattice. The renormalization of the lattice was accomplished using the function

\texttt{renormalize()}

49
which may likewise be found in the appendix. The renormalize function broke the larger lattice into chunks at a scale specified by “Renormalization Cell”. The lattice was then rescaled in the manner discussed in section 3.3.3. This is either performed once or until the lattice is below the size of the renormalization cell based on the option specified in “Renormalization Cascade”. At each step of the renormalization process a new lattice is created.

It quickly became apparent that using Java based OSP as the framework to arrive at meaningful quantitative results for $p_c$ and critical exponents for correlated percolation was impossible. As was discussed previously, in order to get accurate quantitative predictions we eventually wanted to be able to perform large-cell renormalization on lattices larger than $10,000 \times 10,000$. Both the speed of the supporting libraries and the fact that OSP carries a substantial front-end that further decreases its speed are responsible for this inadequacy. Despite its failings, this initial code has proven invaluable in subsequent work.
4.2 C simulations

While the OSP libraries lacked speed they provided an excellent beginning, allowing us to debug a great deal of code in a very visual medium. However, in order to increase the speed of our algorithms and decrease the memory consumption we quickly constructed a library to perform the entire simulation in C. Before getting into the specifics of our forays into C we begin with some overall comments about the paradigm with which we constructed the code and about general aspects of the program.

In general as with much scientific programming we began by constructing code that we could check against a known result. We then extended the code in some manner to include needed, new functionality. Then, we conducted tests to ensure that both the new and old code agreed with one another in regime where the old code was applicable. In this way we were able to come up with working algorithms whose output was novel, while maintaining a degree of confidence in the validity of our results. Furthermore, it was of paramount importance for the code that we developed to scale as efficiently as possible with $L$ both with respect to runtime and memory. An important theme in development was producing a library that could produce results with relative ease. Early on we decided to have output written in a format which could be read directly into Mathematica as a notebook. An example of some output generated by our program may be seen in figure 21. We also wanted to produce a library that did not need to be recompiled in order to change the parameters of a run, so input parameters were read from a separate file. Using this dynamic allocation we allowed our code to load an options file specifying the range of system sizes to test, the seed for the random number generator, the number of iterations for the Monte Carlo simulation, the allowable error in the result, and the output file. An example of such a file may be seen in 22. This decision proved to be especially important when we ran on a remote cluster since recompiling the code took some effort.

Minimizing the runtime of the code was one of the paramount challenges during this project; with this in mind, most changes to the code were in pursuit of faster runtimes and, in particular, better scaling with system size. Improving how the runtime scaled with system size became increasingly important, and difficult, as we wished to probe increasingly large systems. To minimize
memory consumption the lattices are stored are $L^2$ arrays of booleans. Furthermore, to reduce wasted memory and to improve the usability of our code we ensured that all large portions of memory were allocated dynamically. The ability to specify the seed of the random number generator was particularly important since it allowed us to produce reproducible results.

In any algorithms whose underlying mechanism is stochastic, the manner in which one generates random numbers is crucial. Since pseudo-random number generation is a generally difficult process and using a poor random number generator can create artifacts in the data, care must be taken. For this purpose we chose to use the Mersenne Twister [48] which has many favorable properties.
including a very large period of $2^{19937} - 1$ making it a good approximation to a true random number generator.

To ensure that our culling algorithms were functioning properly en-lieux of a visual environment we began by considering small-cell renormalization. In these cases we were able to compute the renormalization group flows exactly and compare them with the literature.

4.2.1 Exact renormalization

We began in the known regime of small cell renormalization. Constructing the renormalization group flows for small lattices allowed us to perform the calculations exactly. In order to compute the renormalization group flows exactly we explicitly generated every possible $L \times L$ lattice and determined whether it percolated. As can be seen by an elementary combinatorial argument the number of $L \times L$ lattices is $2^{L^2}$; thus, this method quickly became prohibitively slow. We were only able to compute exact renormalization group flows for lattices of size less than $6 \times 6$. To perform the renormalization group calculation for a $6 \times 6$ lattice would have required the consideration of $2^{36}$, or $68,719,476,736$, lattices. As usual we began with standard percolation, and then moved on to spiral and force-balance.

In order to successfully perform the computations, two primary functions were used. The first, auxiliary, function is titled

```cpp
bool LatticePercolates(Lattice *lat)
```

as seen in section 2 appendix. This function determines whether or not a path of occupied cells exists going from the bottom of the lattice to the top. Here a path is composed of a collection of occupied sites that may be traversed by moving between nearest neighbors only. This process is accomplished by brute force, with sites checked iteratively until either all sites have been checked or a path has been found. If a path is found the function returns true, with false returned otherwise.

The second function was

```cpp
int CountPercolation(int n, int &total)
```

which may be seen in section 2 of the appendix. This function constructs every $L \times L$ lattice with $n$
occupied sites and then determines whether or not it percolates using the function LatticePercolates. This function returns the number of $L \times L$ lattices that percolated as well as accumulates the variable total which counts the total number of $L \times L$ lattices that have percolated.

In order to reconstruct the renormalization group flows we recall that $R(p)$ has the functional form of equation (75). Then we note that for a given lattice, given the occupation number, $n$ in our “microcanonical ensemble”, $p$, will be given by $n$ divided by the total number of sites on the lattice, $L^2$. In this way we are able to compute the coefficients $S_n$ in the renormalization group flow equation and in so doing determine the flow itself.

To give an example of the procedure we consider the set of $2 \times 2$ lattices, enumerate them explicitly, then count the total number of lattices that percolate. As can be noted there are 16 $2 \times 2$ lattices which may be seen in figure 23. In this case we can tabulate the number of lattices that percolated vertically for a given value of $0 \leq n \leq 4$. Finally, we may construct this renormalization group flow using equation (75).

$$R(p) = p^4 + 4(1 - p)p^3 + 3(1 - p)^2p^2$$  \hspace{1cm} (79)
The plot of the flow may be seen in Figure 24. We note that, qualitatively, this renormalization group flow is similar to the flow that we would expect in 19.

After ensuring the validity of standard percolation we added support for spiral percolation. To do this we added a culling step according to the rules outlined in section 3.2.1. This culling steps were accomplished using the functions

```c
bool CullLatticeWired(Lattice *lat)
```

and alternatively,

```c
bool CullLatticePeriodic(Lattice* lat)
```

both of which can be seen in section 2 of the appendix. In each case the lattice is traversed and those cells that fail to satisfy the stability condition of the spiral model are removed. This traversal is performed iteratively until no unstable cells are present on the lattice. However, when simulating finite systems and extrapolating to the infinite system limit there are two general strategies referred to by “wired” and “periodic” in routine names. First, one can add a few rows to each side of the lattice and assume that those cells on the boundary are fixed. The second strategy, and the strategy that we generally pursued, was to assume that the edges of the lattice wrap around to the other side to form periodic boundary conditions. In general these two strategies will yield different results.
for finite systems. The first function outlined above performs the former, “wired” culling strategy while the latter performs the periodic culling. At this stage culling is performed by iteratively looping through each cell on the lattice and removing it if it fails to satisfy the stability condition.

Beginning with small cell renormalization was a valuable exercise since the renormalization group flows are available for standard percolation. This allowed us to ensure that the algorithms discussed above functioned properly for a known model before exploring spiral percolation and 24NN percolation. Furthermore, as we implemented more complicated techniques to allow us to explore larger systems we could consistently check that they still worked in the regime of small cell renormalization.

4.2.2 Monte-Carlo scheme

As was noted in the above section the number of $L \times L$ lattices is $2^{L^2}$ and the number of lattices with $n$ occupied sites is given by $\binom{L^2}{n}$. It was impossible for us to compute renormalization group flows for systems larger than $6 \times 6$. Therefore a new method was needed to probe larger systems. To accomplish this task we turned to a powerful technique in computational physics known as Monte-Carlo. There is no single Monte-Carlo algorithm, the words Monte-Carlo refer to algorithms which employ random numbers to generate results; in general the quantities measured by a Monte-Carlo algorithm may be perfectly deterministic. Monte-Carlo simulations attempt to determine aspects of a complex sample space by leveraging randomness. This is usually accomplished by randomly sampling a representative subset of the sample space and using this subset to estimate desired properties of the entire space.

Here is a standard example of a Monte-Carlo algorithm that is fundamentally similar to the use of the technique in this work. We use Monte-Carlo methods to estimate $\pi$. Consider the unit circle inscribed in a square of side 2 as can be seen in figure 25. We note that the area of the square will be 4 while the area of the circle will be $\pi$. We then choose points $(x, y) \in [-1, 1]^2$ uniformly and determine whether or not the point resides in the circle by considering whether $\sqrt{x^2 + y^2} \leq 1$. We let $N$ be the number of points that we have chosen that reside in the circle and let $T$ be the total number of points. We can see in figure 26 that as we increase $T$ the circle is increasingly well
approximated by the random points. In particular, as the number of points tends to infinity the number of points that fall into the circle should be proportional to the area of the circle. Thus, we can make the approximation that the ratio $N/T$ will be roughly equal to the ratio $\pi/4$. To demonstrate this fact being used to compute an estimate for $\pi$ we ran this Monte-Carlo algorithm over 20,000,000 iterations and plotted its convergence. The plot of the convergence towards $\pi$ may be seen in figure 27. It is evident that Monte-Carlo algorithms may be used to great effect in sampling complex spaces. However, in this “area” of a circle example, convergence is fairly slow. In specific cases it is important to be able to estimate the error as a function of the number of Monte-Carlo steps.

In our case we wished to apply a Monte-Carlo style algorithm to compute renormalization group flows for large lattices. Our desired result was the number of lattices that percolated for a given
Figure 27: The convergence of a simple Monte-Carlo algorithm towards $\pi$ as the number of iterations of the system is increased from 0 to 20,000,000.

occupancy, $n$. Thus, for $L > 6$ we used a Monte-Carlo algorithm to estimate the fraction of lattices, $S_n$, with occupation probability $p$ that percolated, by randomly sampling the space of lattices with a given $n$ as opposed to enumerating them directly. This problem is analogous to the example of the circle above and our solution is likewise similar. We attempt to estimate the “volume” of some partition of a space by randomly choosing points in the space and determining what fraction reside in the “volume”. In the case at hand the points in our space are lattices and the “volume” are those lattices that percolate. Most practically useful Monte-Carlo algorithms attempt to choose points in sample space not uniformly at random, but according to some heuristic. However, since our space of lattices is relatively homogeneous we do not use such a criterion and simply sample the space at random.

The advantages of using a Monte-Carlo approach are clear as others have found [31, 47]. Instead of sampling the entire $\left(\begin{array}{l}2L^2 \\ n \end{array}\right)$ lattices of the space it suffices to sample, at random, a very small subset of these lattices. As we seek only the fraction of lattices of occupancy $n$ that percolate and the size of the sample space may be regarded as effectively infinite it follows that the accuracy of our results depend only on the number of trials and not on the size of the lattice. In other words, we postulate that we do not need a given fraction of all $L \times L$ lattices, for each $L$, but rather we need a certain, sufficiently large, number of lattices for any given $L$ and $n$. This is extremely important since it
implies that although the size of our sample space increases superalgebraically with system size, we need not increase the number of samples. This in turn implies that our Monte-Carlo algorithm is unaffected by the size of the sample space, as opposed to our exact enumeration which clearly scaled as $2^{L^2}$. However, other aspects such as culling and determining percolation will, in general depend on the size of the lattices.

While we do not use any heuristic for a given $n$, we do employ a heuristic in deciding how many realizations must be performed for a given occupancy. To motivate this note that if $n/L^2 \gg p_c$ then every lattice will percolate and likewise if $n/L^2 \ll p_c$ then no lattices will percolate for sufficiently large systems. Therefore, for such $n$ it suffices to check very few lattices, it is only about the critical point that more lattices must be tested. Therefore, we chose to distribute the number of lattices to test as a normal distribution\(^{26}\) whose mean was $n_c \approx p_c L^2$. This ensured that we took the most samples about $p_c$. In this case we knew from [34, 37] the rough value of $p_c$ a priori. However, in the absence of this knowledge we could, in principle, determine roughly where it was by constructing renormalization group flows with uniform sampling at first. We typically used 100,000 iterations spread out over the entire gaussian.

The majority of the new computational work is performed using the function,

```
int *MCCountPercolation(int L, int *iterations, int seed)
```

which may be seen in section 2 of the appendix. This function takes as input the size of the lattice, $L$, an array specifying the number of configurations be tested for each $n$ occupied sites, iterations, and a seed used to generate random numbers (all random numbers are generated using an open source Mersenne twister algorithm). The output of MCCountPercolation is an array specifying, for each occupancy, $n$, the number of tested lattices to survive percolation. The variable iterations is an array that specifies the number of iterations to test.

\(^{26}\)A normal, or gaussian, distribution has a probability density function $p(x) \propto \exp(- (x - \mu)^2 / 2\sigma^2)$ where $\mu$ is the mean and $\sigma^2$ is the variance.
4.2.3 Binary searching and finite clusters

The implementation of a Monte-Carlo approach allowed for the exploration of systems whose size was drastically larger than those systems probed by exact methods. Although we were able to probe systems whose size was on the order of $L = 100$ we were unable to consider systems that were much larger. However, it became readily apparent that in order to accurately compute either the critical point, $p_c$, or the critical exponent $\mu$ we would need to access still larger systems. It became necessary to evaluate our work and determine which aspects of the code were slowing the project down and which parts could be made significantly faster. Through the course of our investigation we found that there were two primary ways in which the code could be made faster. First we noted that to compute $p_c^\infty$ it suffices to construct only a small fraction of the renormalization group flow. The second realization was that we need not check for percolation when considering either the spiral model or the 24NN model because, as was noted in section 3.2 and will be demonstrated in section 5.2.2, neither model admit finite clusters.

We begin by giving some motivation and methodology for the determination of $p_c(L)$ without the necessity of computing the renormalization group flow. We first consider the flow equation (75), $R(p)$. We note that the binomial coefficients will grow quickly with $L$ and will peak very strongly at $n = L^2/2$. In particular, the binomial coefficients will change much faster than the factors of $p^n(1 - p)^{L^2 - n}$ and so the latter may be regarded as constant with $n$ for large enough $L$. It follows that since the factors of $\binom{L^2}{n}S_n$ are the number of lattices with $n$ occupied sites that percolate and the $p^n(1 - p)^{L^2 - n}$ acts as a constant normalization factor then the function $\pi(p) = S_{pL^2}$ will well approximate $R(p)$. The approximation of $R(p)$ for two system sizes can be seen in Figure 28.

We now examine the notion of the critical point $p_c(L)$ and its extrapolation to $p_c^\infty$, as discussed in section 3.2.1. Recall that the critical point $p_c(L)$ for a system of size $L$ is defined as a fixed point for $R(p)$, satisfying equation (73),

$$p_c = R(p_c).$$

(80)

In section 3.3.5 and, in particular, in figure 19 we note that while for finite systems $R(p)$ is a smoothly varying function in the infinite system limit $R(p)$ features a discontinuity at $p_c^\infty$. We
Figure 28: Approximation of the renormalization group flow, $R(p)$ with the $S_n$ computed by the Monte Carlo algorithm for $10 \times 10$ and $50 \times 50$ uncorrelated lattices. In grey are the $S_n$ and in black in the renormalization group flow. We can see that the $S_n$ better approximate $R(p)$ as the system size increases. Generated using Mathematica from data acquired using our algorithm.

thus define $p'_c(L)$ alternatively by $\frac{1}{2} = R(p'_c(L))$, noting that any sensible, decently large constant would work in the place of $1/2$. Since $R(p)$ discontinuously increases from 0 to 1 in the infinite system limit it follows that

$$\lim_{{L \to \infty}} p'_c(L) = \lim_{{L \to \infty}} p_c(L) = p_c^\infty. \quad (81)$$

This equivalence of the various definitions for $p_c(L)$ may be seen in figure 28. Since any notion of criticality is only defined in the infinite system limit, it follows that these various definitions for $p_c(L)$ are equivalent. The new definition of $p_c(L)$ offers computational advantages. A similar

Figure 29: Example illustrating equivalence of the definitions of $p_c(L)$ as $c = R(p_c(L))$ and $p_c(L) = R(p_c(L))$ respectively.
The results of the last two paragraphs may be used to come up with a more efficient algorithm to compute \( p_c(L) \) without reproducing the entire renormalization group flow. The new algorithm was written as the function

\[
\text{void MCCountPercolationRootFind(int L, int *\&bounds, int lowerbound, int total, int error, int seed)}
\]

which can be seen in section 2 of the appendix. To implement this algorithm we use the above definition for \( p_c(L) \) as \( 1/2 = R(p_c(L)) \). We begin by describing a base case for the algorithm and note that it may be applied iteratively to determine \( p_c(L) \). Suppose that \( p_c(L) \in [a, b] \) for \( 0 < a < b < 1 \) and \( a = (n_1 + n_2)/2L^2 \) and \( b = n_2/L^2 \). We then use the Monte-Carlo algorithm discussed above to compute \( S_{n_3} \) where \( n_3 = [(n_1 + n_2)/2] \). If \( S_{n_3} > 1/2 \) then consider the interval in the next step \([n_1/L^2, n_3/L^2]\) otherwise use the interval \([n_3/L^2, n_2/L^2]\). Applying this procedure iteratively we note that the interval containing \( p_c(L) \) halves at each step. This process will terminate when either the interval in which \( p_c(L) \) resides is of the form \([n/L^2, (n+1)/L^2]\) or when some acceptable precision has been reached. The former imposes a limit on the possible precision with which we may find \( p_c(L) \); while this precision limit may seem like a shortcoming of this method in practice it is irrelevant because the limit on the error is \( 1/L^2 \) which is quickly overtaken by the error inherent in our Monte-Carlo algorithm. The improved efficiency resulting from this new algorithm may be seen in figure 30. This figure clearly illustrates the advantages of using a partitioning algorithm as opposed to constructing the entire renormalization group flow. In the former case far fewer runs are needed while the accuracy in computing \( p_c(L) \) is maintained and increases for larger systems. In the case of a 100 \times 100 system 10,000 values of \( S_n \) were reduced to 13. We also note that since the binary search is a multiplicative property it also does not depend on the size of the system and so is independent of \( L \).

### 4.2.4 A better culling algorithm?

As before, the efficiency gained from directly searching for \( p_c(L) \), as opposed to constructing the renormalization group flow and removing checks for percolation, vastly improved the size of systems that we were able to consider. We were thus able to extract \( p_c(L) \) from systems whose sizes
Figure 30: The number of points needed to construct the entire renormalization group flow for a 100 × 100 lattice is shown in grey on the left. Conversely, the number of points needed to find $p_c(L)$ using the partitioning algorithm appears in grey on the right. Using the renormalization group flow we find $p_c(L) = 0.5927$ while using the partitioning algorithm we find $p_c(L) = 0.5923$. Figure produced with Mathematica using data from our algorithms.

were on the order of 1000. However, we were again unable to increase the size of systems under considerations to the order of 10,000 in any reasonable time-scale. To make matters worse we were experiencing scaling of our culling algorithm that went as $O(L^\tau)$ with $\tau \approx 3$. This scaling is attributed to two effects: first, our culling algorithm traversed the entire lattice for each culling iteration, introducing a scaling of $O(L^2)$, second it was shown in [37] that the number of culling steps went as $O(L^\tau)$ with $\tau \approx 1.22$. Thus, overall our algorithm scaled as given. This was especially problematic, that e.g. increasing the length scale by an order of magnitude increased the computational time by three orders of magnitude. This indicated to us that there should be a more efficient method for culling the lattice since $O(L^2)$ of the computation time came from simply traversing the lattice. Clearly many sites were either inactive or stable at each step and did not need to be considered repeatedly Happily, we were able to devise an algorithm which didn’t require repeated traversals of the entire lattice. To our knowledge this is the first use of such an algorithm for correlated percolation requiring culling [49].

To motivate our new algorithm consider an occupied site that satisfies the stability condition. Depending on the model in use this site’s stability depends only on the 8, 16, or 24 nearest neighbors. The only condition under which an initially stable site may become unstable is if one of its neighbors was culled in the previous iteration. Our algorithm leveraged this property by initially determining
an active perimeter of sites that are unstable, in the form of a list. These sites are then culled and their nearest neighbors that have subsequently become unstable become the new active perimeter and are appended to the list. In this way we avoid the necessity of traversing the entire lattice at each culling step and focus only on those cells that will be culled in the next step. We can see a figure of this process in figure 31 and video in the appendix. Likewise the function

```c
bool CullLargeLatticePeriodicFast(LargeLattice *lat)
```

...can also be seen in the appendix.

The efficiency of this new algorithm was exceptionally good. Using the old method of traversal it took more than 24 hours to complete one Monte-Carlo step on a lattice of size 10,000. With the new implementation this time was contracted to approximately 100 seconds. Furthermore, we found evidence that this new algorithm scales as $O(L^{1.3})$ which represents a significant improvement over an $L^3$ algorithm.
Figure 31: The incremental culling of a $400 \times 400$ lattice under the spiral model with $p = 0.67$. Red marks denote sites on the active perimeter. Proceeding from (a) to (d) we see steps in the incremental culling of the lattice. Quickly the active perimeter consists of only very few points on the lattice.
5 Results

Our results will be presented chronologically, beginning with exact-renormalization before moving to the regime of Monte-Carlo renormalization. One of the primary goals of this section is to demonstrate the validity of the methods discussed above. To this end we discuss the agreement of the Monte Carlo results with the exact enumeration results, of the results for standard percolation with those present in the literature, of the critical probabilities derived by binary search with those derived from the group flows, of tests performed on the new culling algorithm, and finally of agreement between the results found for the spiral model and the results in [34]. In each of these cases the results are discussed exclusively for standard percolation and for the spiral model.

Discussion of the 24NN model is deferred until section 5.7 although many of the same comparisons were made as for the spiral model. The reason why we discuss comparisons with the spiral model as opposed to the 24NN model is that all of the algorithms are generic and do not depend on the details of the culling step. Therefore, we felt that including additional discussions of the agreement with the 24NN model at each step would only convolute the discussion. Furthermore, since precise results have not been determined for the 24NN model it is less satisfying to compare such results to those found in the literature.

5.1 Exact renormalization

We begin by giving our numerical results obtained by direct enumeration for systems of sizes $3 \times 3$, $4 \times 4$, and $5 \times 5$ for standard percolation and for the spiral model. These results may be seen to be tabulated in figures 32 and 33 for $L = 3$ and $L = 4$ respectively. We also plot the exact renormalization group flows for $L = 3, 4, 5$ for both standard percolation and spiral percolation in figure 34. We can see that qualitatively these plots are similar to what we might expect. We find $p_c = 0.619355, 0.61926, 0.618095$ for $L = 3, 4, 5$ respectively. We note both that these values for $p_c$ are in the same region as the expected value of 0.5927 and also that they are decreasing with $L$. Given the extremely small lattices under consideration it is unsurprising that the $p_c$ extracted from these lattices is not correct. Furthermore, the renormalization group flows agree with those found in [?].
<table>
<thead>
<tr>
<th># Occupied Sites</th>
<th>Fraction Percolate (Standard)</th>
<th>Fraction Percolate (Spiral)</th>
<th>#Lattices</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>$1/28$ (0.0357)</td>
<td>$1/28$ (0.0357)</td>
<td>84</td>
</tr>
<tr>
<td>4</td>
<td>$11/63$ (0.175)</td>
<td>$1/7$ (0.143)</td>
<td>126</td>
</tr>
<tr>
<td>5</td>
<td>$59/126$ (0.468)</td>
<td>$59/126$ (0.468)</td>
<td>126</td>
</tr>
<tr>
<td>6</td>
<td>$67/84$ (0.798)</td>
<td>67</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>36</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 32: Exact enumeration of $3 \times 3$ lattices. The number of occupied sites, $n$, is displayed on the left and the number of lattices with $n$ occupied sites is displayed on the right. In the middle are the number of lattices with $n$ occupied sites that percolate with, and without, culling.

<table>
<thead>
<tr>
<th># Occupied Sites</th>
<th>Fraction Percolate (Standard)</th>
<th>Fraction Percolate (Spiral)</th>
<th>#Lattices</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>560</td>
</tr>
<tr>
<td>4</td>
<td>$1/455$ (0.00220)</td>
<td>$1/455$ (0.00220)</td>
<td>1820</td>
</tr>
<tr>
<td>5</td>
<td>$5/364$ (0.0137)</td>
<td>$1/91$ (0.0110)</td>
<td>4368</td>
</tr>
<tr>
<td>6</td>
<td>$15/308$ (0.0487)</td>
<td>$3/91$ (0.0330)</td>
<td>8008</td>
</tr>
<tr>
<td>7</td>
<td>$33/260$ (0.127)</td>
<td>$233/2860$ (0.0815)</td>
<td>11440</td>
</tr>
<tr>
<td>8</td>
<td>$1708/6435$ (0.265)</td>
<td>$163/858$ (0.190)</td>
<td>12870</td>
</tr>
<tr>
<td>9</td>
<td>$659/1430$ (0.461)</td>
<td>$577/1430$ (0.403)</td>
<td>11440</td>
</tr>
<tr>
<td>10</td>
<td>$2707/4004$ (0.676)</td>
<td>$662/1001$ (0.661)</td>
<td>8008</td>
</tr>
<tr>
<td>11</td>
<td>$467/546$ (0.855)</td>
<td>$467/546$ (0.855)</td>
<td>4368</td>
</tr>
<tr>
<td>12</td>
<td>$438/455$ (0.963)</td>
<td>$438/455$ (0.963)</td>
<td>1820</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>560</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>120</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 33: Exact enumeration of $4 \times 4$ lattices. The number of occupied sites, $n$, is displayed on the left and the number of lattices with $n$ occupied sites is displayed on the right. In the middle are the number of lattices with $n$ occupied sites that percolate with, and without, culling.

5.2 Agreement between exact and Monte Carlo renormalization.

We reference the above results to argue that the Monte Carlo computation converges to the same renormalization group flows as the exact computation. To do this we consider the three flows above and reconstruct them using our Monte-Carlo methods. We tabulate the results of a Monte Carlo
Figure 34: Exact renormalization group flows for $L = 3, 4, 5$ for standard percolation and the spiral model. On the left the flows for standard percolation are in bold while the flows for the spiral model are dashed to provide contrast. Conversely, on the right the flows for spiral percolation are in bold while the flows for standard percolation are light. In each case the steeper flows represent higher $L$.

search for $L = 3$ and $L = 4$ in figures 35 and 36. This provides strong evidence for the convergence

<table>
<thead>
<tr>
<th># Occupied Sites</th>
<th>Fraction Percolate (Standard)</th>
<th>Fraction Percolate (Spiral)</th>
<th># Realizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>23574</td>
</tr>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>76098</td>
</tr>
<tr>
<td>2</td>
<td>0.000</td>
<td>0.000</td>
<td>124590</td>
</tr>
<tr>
<td>3</td>
<td>0.034</td>
<td>0.034</td>
<td>165566</td>
</tr>
<tr>
<td>4</td>
<td>0.174</td>
<td>0.142</td>
<td>181802</td>
</tr>
<tr>
<td>5</td>
<td>0.468</td>
<td>0.468</td>
<td>165873</td>
</tr>
<tr>
<td>6</td>
<td>0.797</td>
<td>0.797</td>
<td>123769</td>
</tr>
<tr>
<td>7</td>
<td>1.000</td>
<td>1.000</td>
<td>76427</td>
</tr>
<tr>
<td>8</td>
<td>1.000</td>
<td>1.000</td>
<td>38577</td>
</tr>
<tr>
<td>9</td>
<td>1.000</td>
<td>1.000</td>
<td>23724</td>
</tr>
</tbody>
</table>

Figure 35: Monte Carlo enumeration of $3 \times 3$ lattices. The number of occupied sites, $n$, is displayed on the left and the number of lattices with $n$ occupied sites that were tested is displayed on the right. In the middle are the fraction of lattices with $n$ occupied sites that percolate with, and without, a culling step.

of the Monte Carlo routines to the values produced by exact enumeration.

5.3 Agreement for standard percolation.

We next wished to use our Monte Carlo renormalization group methods to determine $p_c^\infty$ of standard percolation. We further wished to verify the scaling relationship given in (72). To do this we
<table>
<thead>
<tr>
<th># Occupied Sites</th>
<th>Fraction Percolate (Standard)</th>
<th>Fraction Percolate (Spiral)</th>
<th># Realizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>22925</td>
</tr>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>27476</td>
</tr>
<tr>
<td>2</td>
<td>0.000</td>
<td>0.000</td>
<td>39145</td>
</tr>
<tr>
<td>3</td>
<td>0.000</td>
<td>0.000</td>
<td>54284</td>
</tr>
<tr>
<td>4</td>
<td>0.002</td>
<td>0.002</td>
<td>69680</td>
</tr>
<tr>
<td>5</td>
<td>0.014</td>
<td>0.011</td>
<td>83091</td>
</tr>
<tr>
<td>6</td>
<td>0.048</td>
<td>0.033</td>
<td>94652</td>
</tr>
<tr>
<td>7</td>
<td>0.126</td>
<td>0.081</td>
<td>100035</td>
</tr>
<tr>
<td>8</td>
<td>0.266</td>
<td>0.192</td>
<td>10704</td>
</tr>
<tr>
<td>9</td>
<td>0.461</td>
<td>0.405</td>
<td>94306</td>
</tr>
<tr>
<td>10</td>
<td>0.678</td>
<td>0.663</td>
<td>83403</td>
</tr>
<tr>
<td>11</td>
<td>0.855</td>
<td>0.855</td>
<td>69207</td>
</tr>
<tr>
<td>12</td>
<td>0.961</td>
<td>0.961</td>
<td>53966</td>
</tr>
<tr>
<td>13</td>
<td>1.000</td>
<td>1.000</td>
<td>39428</td>
</tr>
<tr>
<td>14</td>
<td>1.000</td>
<td>1.000</td>
<td>26987</td>
</tr>
<tr>
<td>15</td>
<td>1.000</td>
<td>1.000</td>
<td>17525</td>
</tr>
<tr>
<td>16</td>
<td>1.000</td>
<td>1.000</td>
<td>23186</td>
</tr>
</tbody>
</table>

Figure 36: Monte Carlo enumeration of 4 × 4 lattices. The number of occupied sites, \( n \), is displayed on the left and the number of lattices with \( n \) occupied sites that were tested is displayed on the right. In the middle are the fraction of lattices with \( n \) occupied sites that percolate with, and without, a culling step.

We computed the renormalization group flows for lattices up to \( L = 100 \). We can see these flow equations in figure 37. From the renormalization group flow equations we may find the various estimates for the critical point by looking a fixed point of the renormalization group flow equations.

The quantity that we are really interested in is the critical point \( p_c^\infty \) in the infinite system limit. Hence, we use the scaling form in (72) and we plot the various \( p_c(L) \) against \( 1/L^{1/\nu} \). By fitting these points to a line, we may estimate \( p_c^\infty \) as the y-intercept of the line. Recall that the data will only obey the proper scaling relationships in the asymptotic limit. Thus, we need to be confident that we are approaching the asymptotic limit before performing the estimation. In the case of standard percolation \( \nu = 4/3 \) is known. We then construct the plot as defined above for the \( p_c(L) \) given to us by the renormalization group flows for \( L = 6, 10, 30, 50, \) and 100 we find the result in figure 38. This figure shows the manner in which the points approach their asymptotic limit. We find that the three points corresponding to the largest systems are somewhat more linear than the other points. We therefore regress our fits against a wider selection of data and against
Figure 37: Renormalization group flows for lattices between \( L = 6 \) and \( L = 100 \) as well as the line \( p \).

Figure 38: Scaling of data for standard percolation.

the last three points. The convergence of the \( y \)-intercept shows the approach to the asymptotic limit. Furthermore, we note that our regressed values for \( p_c^\infty \) of 0.5911 and 0.5929 are quite close to the value in the literature of 0.5927. Hence, we propose that the renormalization group methods developed above are applicable to the case of standard percolation.
5.4 Lack of finite-clusters

We recall that for uncorrelated percolation the renormalization group flow equation is given by (75). However, in the case of correlated percolation the notion of $S_n$ is not so clear. In principle two things must happen for a lattice to contribute to $S_n$. The lattice must first survive culling and then it must independently percolate. Thus, we may decompose $S_n$ into the product of two numbers $A_n$ and $B_n$ where $A_n$ is the probability that a lattice will survive culling and $B_n$ is the probability that a lattice that survives culling will percolate. The total probability is then given by $S_n = A_n B_n$ seeing as these probabilities are independent.

We wished to investigate how $A_n$ and $B_n$ as the lattice size increased. To do this we constructed plots of the total number of lattices with $n$ filled sites that were attempted, the number of these that survived culling, and the number of these that percolated. For varying $L$ these plots can be seen in figure 39. From these plots we can see that for large $L$, surviving culling is synonymous with percolating in the spiral model. This reflects a known fact that the spiral model features no supported finite clusters. Thus, any lattice that survives culling will necessarily feature a spanning cluster and will therefore percolate. This allowed us to remove the step testing for percolation and in doing so improve the speed of the algorithm.

5.5 Investigating error bounds.

These realizations about the methods that we have employed in sampling configuration space lead us to evaluate our algorithms in more detail. We constructed a method to give an indication of how many $L \times L$ lattices with $n$ occupied sites need to be tested to achieve an accuracy of $\epsilon$ or better. The algorithm will be described subsequently but we must first clarify some terminology. First, fix $L$, $n$, and $\epsilon$. We define a “bin” to be a variable that keeps track of the fraction of lattices generated for the bin that have survived culling. In general, for the $i$th bin we define the number of lattices that have survived culling to be $b_i$. We define the bin size, $B$, to be the maximum number of lattices that may be generated for each bin. Finally we fix the number of bins to be $N$. Our algorithm is then:

1. Construct $N$ different bins.
Figure 39: Plots of the number of lattices tested, the number of lattices that survived culling, and the number of lattices that percolated for $L = 10, 20, 40$, and $80$ respectively. The envelope represents the total number of lattices tested. The number of lattices that survived culling and the number that percolated after that are below the envelope, however they are mostly indistinguishable.
2. Let $1 \rightarrow i$.

3. While the number of tested lattices is less than $B$
   
   (a) Construct a lattice, if it survives culling then increment $b_i$.

4. Let $i + 1 \rightarrow i$.

5. Repeat from 3 until each of the $N$ bins has been processed.

6. Compute the mean fraction of lattices that survived percolation,
   
   $$\bar{b} = \frac{1}{BN} \sum_{i=1}^{N} b_i.$$  \hfill (82)

7. Compute the sum of squares error,
   
   $$\epsilon' = \sqrt{\sum_{i=1}^{N} \left( \frac{b_i}{B} - \bar{b} \right)^2}.$$  \hfill (83)

8. If $\epsilon < \epsilon'$ then let $2B \rightarrow B$, repeat from 2.

In this way we produce new lattices for testing, to see if they percolate, until the error has dropped below some threshold. Running this code over several lattice sizes we produced the results of figure 40.

There are several worthwhile qualitative features that may be inferred from these plots. First and foremost is that the form of the distribution has gaussian-like features. As one might expect each distribution is peaked about $L^2 p_c$ and falls off quickly to zero. Some features that we did not expect that are encouraging. First, it does not appear that, for a given fraction of occupied sites, significantly more lattices are required to get at accurate view of phase space for increasing $L$. Also, the absolute width of the distribution stays roughly constant. It follows that the fraction of configuration space that must be sampled stays roughly constant with $L$. This is good for our theory since the problem might have otherwise become intractable. Second, it appears that the
distribution falls off much more quickly than the gaussian that we are predicting. Hence, it might be possible to use a more intelligent metric that samples a smaller fraction of phase space.

5.6 Agreement with the binary search and novel culling algorithms.

The agreement between the values of $p_c(L)$ given by the method of binary search outlined in section 4.2.3 is compared with the standard Monte Carlo method. To perform the comparison we considered the values of $p_c(L)$ produced by binary search with those values produced by the reconstruction of the renormalization group flow in the regime where both methods could be performed. In particular we considered a selection of lattices of size $L = 5, 10, 20, 40$ and 80. These renormalization group flows may be seen in figure 41. As $L$ increases from 5 to 80 it is evident that the predictions of the root finding algorithm improve. In particular we can see that in the case of $L = 5$ the sparse number of $p$ that may be chosen by the search algorithm has reduced the quality of the fit. This provides evidence that the result found by the root finding algorithm converges to the value of $p_c(L)$ given by the condition that $R(p_c(L)) = 1/2$ for large $L$. Since we apply this algorithm only
to systems where \( L > 100 \) it follows that the results of this algorithm will acceptably produce \( p_c(L) \).

The deviations of the root finding algorithm from the root determined from the flow itself may be seen in figure 42.

In order to test the improved culling algorithm we constructed two identical copies of a lattice and then culled one lattice using the standard technique and culled one with the new algorithm. We then compared each site of the lattice searching for deviations. We then repeated this experiment hundreds of times finding no deviations in any case. This satisfactorily demonstrated the success of our culling algorithm in producing identically culled lattices with significantly improved runtime.

### 5.7 Results for the spiral model.

Using the host of new algorithms we wished to demonstrate agreement between our numerical methods and the rigorous results determined in [34]. We first wished to verify the exponentially diverging length scale; recall that given an exponentially diverging length scale we expect

\[
\log L = \log A + |p_c(L) - p_c^\infty|^\mu. \tag{84}
\]
To test the scaling form we fit the functional form in equation (84) to our data with $p_c^\infty$, $A$, and $\mu$ as fitting parameters. The result may be seen in figure 43. We first see that our data is fit well by the scaling form in (84), this provides evidence for the proposed exponential scaling form. Further
encouragement is found in the regressed values for $\mu$ and $p_c^\infty$ of 0.668 and 0.699 respectively. These results represent good agreement with the exact values by Fisher et al. of $\mu \approx 0.64$ and $p_c^\infty \approx 0.705$. Our results appear to be in excellent agreement with the literature values and we expect this agreement to only improve as we probe larger systems and reduce finite-size effects.

5.8 Results for 24NN model.

The 24NN model was investigated in a similar manner to the spiral model, although at the time of writing this thesis much larger systems have been investigated for the latter. As with the spiral model we began by attempting to provide evidence for an exponentially diverging length scale. In a similar manner to the spiral model we would expect the length to diverge as given by (84). We used the same nonlinear fitting method as was used in our investigation of the spiral model to simultaneously find $p_c^\infty$, $\mu$, and $A$ in this case. Specifically, we found $p_c^\infty = 0.433$ and $\mu = 0.684$. We note that, again, both of these results agree with the numerical results in [37] of $p_c^\infty = 0.425 \pm 0.005$ and $\mu = 0.76 \pm 0.2$. In order to test the scaling ansatz we plotted our results in a similar manner to the spiral model. The result can be seen in figure 44. This plot demonstrates that, once again,

![Figure 44: A plot of log(L) against pc(L). In this data we use lattices whose sizes range from L = 100 and L = 6000.](image)

the 24NN model appears to feature super power-law scaling. Despite the fact that our results for
the 24NN model do not show universal behavior conclusively, it is encouraging that the $\mu$ regressed in both cases lay near one another. Once again, we expect as we probe increasingly large systems these results will only improve.
6 Conclusion

We introduced percolation theory in the context of jamming. We had as our goal to demonstrate that two correlated percolation models, the spiral model and the 24 nearest-neighbor model, lay in the same universality class. This goal could be restated as a goal to show that the super-exponent, $\mu$, was the same for both models. The main source of difficulty in this experiment pertained to the exponential divergence of the connectivity length, $\xi$. This exponential scaling forced us to come up with ways to explore extremely large systems before nearing the asymptotic limit.

To this end we successfully developed several techniques to investigate correlated percolation models. We began with small systems where results could be computed exactly and then proceeded to investigate larger systems with new methods where our knowledge was less certain. Through comparison with the exact results we were able to gain confidence in our methods. Each time we implemented new methods to explore larger systems we continued in this paradigm.

The first technique that we implemented was a Monte-Carlo renormalization group approach. The use of random sampling allowed us to explore extremely large configuration spaces in constant time with respect to the system size. We then implemented a binary search of the lattice occupancy, which was again constant in time with regard to system size. Finally, we constructed a nearly linear-time culling algorithm. Thus, our final method is nearly linear in time and allows us to explore systems so long as memory restrictions will allow. To our knowledge this was the first time such a culling algorithm had been used to explore correlated percolation.

These techniques allowed us to explore systems that were larger than previous studies had considered. In particular Fisher et al. in [34] considered systems up to $1600 \times 1600$ and Schwarz et al in [37] considered systems up to $3000 \times 3000$. By contrast we were able to explore systems up to $15,000 \times 15,000$. This allowed us to get much closer to the asymptotic limit than previous studies.

We were able to get quantitative results for both models. For the spiral model, we were able to support the exponential scaling proposed by Fisher et al. [34]. We found $p_c^\infty = 0.699$ which represents a deviation of less than 1% from the exact results of Fisher et al. We further found $\mu = 0.668$ which is a deviation of less than 5% from the theoretical results. This represents an improvement over the results of Jeng and Schwarz of $p_c = 0.701 \pm 0.011$ and $\mu = 0.81 \pm 0.16$. 

81
For the force-balance model we found $p_c = 0.433$ which is close to the Jeng and Schwarz value of $0.425 \pm 0.005$ and $\mu = 0.684$ which is both within the error of the Jeng and Schwarz value and relatively near the known result for the spiral model. Overall, we expect to continue looking at larger systems and believe that our results will improve. Furthermore, this agreement supports our methods and, in particular, supports our use of the renormalization group on systems that feature exponential, as opposed to power-law divergences.

In closing, we believe that the goal of determining whether the spiral model and 24 nearest-neighbor model lie in the same universality class is now within reach.
7 Acknowledgements

I would like to thank, first and foremost, Amy Bug for her constant assistance and many suggestions and revisions. I would also like to thank Andrea Liu for her guidance and support throughout. I would additionally like to thank Carl Grossman for being the second reader of the thesis and for his contributions. I would also like to acknowledge the Liu group for helpful discussions. Finally, I would like to thank Swarthmore College, the LRSM REU program, and the HHMI grant for funding this work.
References


[49] J. M. Schwarz. Based on correspondence with Schwarz and access to their code.
8 Appendix

8.1 Java Code

We first have the culling routine added to the OSP libraries.

Listing 1: Java culling routine

```
public void CullLattice() {
    boolean first = true;
    boolean done = false;

    if (control.getBoolean("Full Cull")) {
        probPlot.clearData();
        probPlot.append(0, 0, control.getDouble("Site occupation probability");
        iteration = 0;
    }

    while ((first || control.getBoolean("Full Cull") && !done)) {
        done = true;
        first = false;
        //control.println("culling method invoked");
        int tmpdata[] = new int[L*L]; //culledlattice.getAll();
        int p = 0;
        for(int x=0;x<L;x++) {
            for(int y=0;y<L;y++) {
                int numberOfNbrs = 0;
                //control.println("siteToTest: " + siteToTest + " has state " + lattice.getIndex(siteToTest));
            }
        }
    }
}
```
if (culledlattice.getValue(x, y) != -2) {
    if (control.getString("Cull Mode").compareToIgnoreCase("k-core") == 0) {
        // Perform k-core percolation culling
        for (int j = 0; j < 4; j++) {
            int tx = (int)Math.round(Math.cos(j * 2 * Math.PI / 4));
            int ty = (int)Math.round(Math.sin(j * 2 * Math.PI / 4));
            int nbr = 0;
            if (!control.getBoolean("Periodic"))
                if (x + tx >= L || x + tx < 0 || y + ty < 0 || y + ty >= L)
                    nbr = -1;
            if (nbr != -1) {
                if (culledlattice.getValue(pmod((x + tx), L),
                                       pmod((y + ty), L)) != -2)
                    numberOfNbrs++;
            }
            if (numberOfNbrs < k) {
                tmpdata[y * L + x] = -2;
            done = false;
            } else {
                tmpdata[y * L + x] = -1;
            p++;
            }
        } else {
            if (control.getString("Cull Mode").compareToIgnoreCase("spiral") == 0) {
                // Perform spiral model culling
            }
        }
    }
}
boolean northwest;
boolean southwest;
boolean southeast;
boolean northeast;

if (!control.getBoolean("Periodic"))
{
    northwest = x>0&&((y+1<L&&culledlattice.getValue(x-1,y+1)!=-2)
                        ||(culledlattice.getValue(x-1,y)!=-2));

    southwest = y>0&&((x>0&&culledlattice.getValue(x-1,y-1)!=-2)
                        ||culledlattice.getValue(x,y-1)!=-2);

    southeast = x+1<L&&((y>0&&culledlattice.getValue(x+1,y-1)!=-2)
                        ||culledlattice.getValue(x+1,y)!=-2);

    northeast = y+1<L&&((x+1<L&&culledlattice.getValue(x+1,y+1)
                        !=-2)||culledlattice.getValue(x,y+1)!=-2);
}
else{
    northwest = culledlattice.getValue(pmod(x-1,L),pmod(y+1,L))
                        !=-2||culledlattice.getValue(pmod(x-1,L),pmod(y,L))!=-2;

    southwest = culledlattice.getValue(pmod(x-1,L),pmod(y-1,L))
                        !=-2||culledlattice.getValue(pmod(x,L),pmod(y-1,L))!=-2;

    southeast = culledlattice.getValue(pmod(x+1,L),pmod(y-1,L))
                        !=-2||culledlattice.getValue(pmod(x+1,L),pmod(y,L))!=-2;

    northeast = culledlattice.getValue(pmod(x+1,L),pmod(y+1,L))
                        !=-2||culledlattice.getValue(pmod(x,L),pmod(y+1,L))!=-2;
67    }
68    if (!((northeast=true&&southwest=true) || (southeast=true&&
69        northwest=true)))
70    {
    71        tmpdata[y*L+x]=-2;
72        done = false;
73    } else {
74        tmpdata[y*L+x]=-1;
75        p++;
76    }
77} else {
78    //define the different regions
79    boolean A,B,C,D;
80    A = B = C = D = false;
81    for (int tx = -2; tx <= 2; tx++)
82    {
83        for (int ty = -2; ty <= 2; ty++)
84        {
85            if (! (tx===GSz&ty==0))
86            {
87                int cx=x+tx;
88                int cy=y+ty;
89
90                    if (control.getBoolean("Periodic") || (cx<L&&cx>=0&&cy<L&&cy
91                          >=0))
92                    {
93                        cx = pmod(cx,L);
94                        cy = pmod(cy,L);
```java
int value = culledlattice.getValue(cx, cy);
if (value != -2)
{
    numberOfNbrs++;
    if (tx < 0)
        C = true;
    if (ty < 0)
        B = true;
    if (tx > 0)
        D = true;
    if (ty > 0)
        A = true;
}

if (numberOfNbrs >= 3 && ((M & B) && (C & D)))
{
    tmpdata[y * L + x] = -1;
    p++;
} else {
    tmpdata[y * L + x] = -2;
    done = false;
}
```

tmpdata[y*L+x]= -2;

//control.println("site " + siteToTest + " has " +
   numberOfNbrs + " neighbors");
numberOfNbrs = 0;
}
}
iteration++;
probPlot.append(0, iteration, ((float)p)/(L*L));
control.println("lattice culled. New: " + ((float)p)/(L*L));
culledlattice.setAll(tmpdata);
}
culledlattice.repaint();
probPlot.repaint();
}

The Java renormalization group code.

Listing 2: Java renormalization routine

```java
public void renormalize()
{
   boolean first = true;
   LatticeFrame newLattice = culledlattice;

   if(control.getBoolean("Renormalize Cascade")){
      probPlot.clearData();
      iteration = 1;
      cullLattice(newLattice);
   }
}
while ((first || control.getBoolean("Renormalize\_Cascade")) && L > 2)
{
  first = false;
  iteration++;
  if (L % 2 == 0)
  {
    int data[] = new int[L*L / 4];
    int p = 0;
    for (int y = 0; y < L / 2; y++)
    {
      for (int x = 0; x < L / 2; x++)
      {
        boolean left = false;
        boolean right = false;
        if (!control.getBoolean("Periodic"))
        {
          left = y * 2 + 1 < L && newLattice.getValue(x * 2, y * 2 + 1) != -2 &&
            newLattice.getValue(x * 2, y * 2) != -2;
          right = x * 2 + 1 < L && y * 2 + 1 < L &&
            newLattice.getValue(pmod(x * 2 + 1, L),
            pmod(y * 2 + 1, L)) != -2 &&
            newLattice.getValue(pmod(x * 2 + 1, L), y * 2)
            != -2;
        }
        else
        {
          left = (newLattice.getValue(x * 2, pmod(y * 2 + 1, L)) != -2) &&
            (newLattice.getValue(x * 2, y * 2) != -2);
          right = newLattice.getValue(pmod(x * 2 + 1, L),
            pmod(y * 2 + 1, L)) != -2 &&
            newLattice.getValue(pmod(x * 2 + 1, L), y * 2) != -2;
        }
      }
    }
  }
data[y*L/2+x] = -1;
p++;
} else
data[y*L/2+x] = -2;
}

L = L/2;

if (control.getBoolean("Renormalize Cascade"))
{
    newLattice = new LatticeFrame("renormalized lattice");
    // unoccupied sites are black and have value -2
    newLattice.setIndexedColor(-2, Color.BLACK);
    // occupied sites that are not part of an identified cluster
    // are red and have value -1
    newLattice.setIndexedColor(-1, Color.RED);
    newLattice.resizeLattice(L, L);
    newLattice.setAll(data);
    cullLattice(newLattice);
    newLattice.setVisible(true);
    renormalizedFrames.add(newLattice);
} else{
    newLattice.resizeLattice(L, L);
    newLattice.setAll(data);
    newLattice.repaint();
}

control.println("lattice renormalized. New p: " + (float)p/(L*L));
60 } else {
61   return;
62 }
63 }
64 probPlot.repaint();
65 }
66
67 public void reset() {
68   control.setValue("Lattice size", 32);
69   control.setValue("Site occupation probability", 0.5927);
70   control.setValue("Random seed", 100);
71   control.setValue("k value", 2);
72   control.setValue("Cull Mode", "k-core");
73   control.setValue("Renormalize Mode", "Vertical");
74   control.setValue("Periodic", true);
75   control.setValue("Renormalize Cascade", false);
76   control.setValue("Full Cull", false);
77   calculate();
78 }
79
80 public static void main(String args[]) {
81   CalculationControl control = CalculationControl.createApp(new
82     kcor9PercolationApp());
83
84   control.addButton("cull", "Cull sites", "Culls sites with less than
85     k neighbors.");
86   control.addButton("renormalize", "Renormalize", "Performs a
87     renormalization of the lattice.");
8.2 C code

We now present the Monte-Carlo renormalization routine.

Listing 3: C Monte-Carlo renormalization group routine

```c
int *MCCountPercolation(int L, int *iterations, int seed, int *\& survived)
{
    // instantiate a new Mersene Twister with seed (seed).
    MTRand *rnd = new MTRand(seed);
```
//create arrays to hold the total number of lattices attempted, the
total number of lattices that
//survived culling, and the total number of lattices that have
percolated with
//n filled sites.
int *percolates = (int*)malloc(sizeof(int)*L*L+1);
survived = (int*)malloc(sizeof(int)*L*L+1);
memset(percolates, 0, sizeof(int)*L*L+1);
mempset(survived, 0, sizeof(int)*L*L+1);

//variables to aid in the construction of a random lattice
int indices[L*L+1];
int lastindex = L*L;

//create a blank lattice
Lattice *lat = CreateLattice(L);

//variable to denote whether the culled lattice has any filled cells
left.
bool empty = true;

//Create data used by CullLattice & LatticePercolates
//holds a copy of the original lattice for use during the cull step
//bool *cullData = (bool*)malloc(sizeof(bool)*L*L);

//holds a copy of the points visited for the percolation step
list<Point*> *percList = new list<Point*>();

// go through all possible numbers of filled sites
for (int it = 0; it <= L*L; it++)
{
    for (int j = 0; j < iterations[it]; j++)
    {
        // Create a random lattice with (filled) filled sites.
        // go through and set the lattice to be empty and set the array
        // indices to contain a reference
        // to each site on the lattice
        for (int i = 0; i < L*L; i++)
        {
            indices[i] = i;
            lat->data[i] = 0;
        }

        // go through until (filled) sites are filled
        for (int i = 0; i < it; i++)
        {
            // choose a random index in the array of indices
            int index = rnd->randInt(lastindex);
            // set that position of the lattice to true
            lat->data[indices[index]] = true;
            // set the value at index to the value of the index at the end of
            // the array
            indices[index] = indices[lastindex];
            // "move" the end of the array inwards.
lastindex--;  
}  
// reset last index  
lastindex = L*L-1;  

// fully cull the lattice using periodic boundary conditions  
while (CullLatticePeriodic(lat, empty)){}  

if (empty==false)  
  survived[it]++;

// check whether the lattice percolates. If so then increment the number of lattices with (filled) full sites that have percolated  
if (LatticePercolates(lat, percList))  
  percolates[it]++;

for(list<Point*>::iterator it = percList->begin(); it!=percList->end(); it++)  
  free(*it);  
percList->clear();

} }

// free any assigned data
FreeLattice(lat);

// delete percList;
//return the total number of percolated lattices.
return percolates;
}

The original wired and periodic culling algorithms.

Listing 4: C original culling algorithms

//Culls a lattice with spiral percolation using a padded boundary.
bool CullLatticeWired(Lattice *lat)
{
//holds a copy of the original lattice for use during the cull step
static bool *data = (bool*)malloc(sizeof(bool)*LATTICE_SIZE*
    LATTICE_SIZE);

//variables used to store the states of the different spiral regions
bool northeast = false;
bool northwest = false;
bool southeast = false;
bool southwest = false;

//variable to denote whether the lattice changed during the culling
//step
bool changed = false;

//copy the data from the lattice to the temporary variable
memcpy(data,lat->data,sizeof(bool)*lat->size*lat->size);

//iterate through each point on the lattice
for(int y=1;y<lat->size-1;y++)
for (int x = 1; x < lat->size - 1; x++)
{
    // if the site is occupied
    if (lat->data[y * lat->size + x] == true)
    {
        // if neither the northwest nor the southeast quadrant are occupied
        northwest = lat->data[(y + 1) * lat->size + x - 1] || lat->data[y * lat->size + x - 1];
        southeast = lat->data[(y - 1) * lat->size + x + 1] || lat->data[y * lat->size + x + 1];
        if (! (northwest && southeast))
        {
            // and neither the southwest nor the northeast are occupied
            southwest = lat->data[(y - 1) * lat->size + x - 1] || lat->data[(y - 1) * lat->size + x];
            northeast = lat->data[(y + 1) * lat->size + x + 1] || lat->data[(y + 1) * lat->size + x];
            // then the site is unsupported so set it to be unoccupied, and
            // note that the lattice has changed.
            if (! (southwest && northeast))
            {
                data[y * lat->size + x] = false;
                changed = true;
            }
        }
    }
}
//return whether or not the lattice changed.
return changed;

//Culls a lattice with spiral percolation using periodic boundary conditions
bool CullLatticePeriodic(Lattice *lat, bool &empty)
{
    //variables used to store the states of the different spiral regions
    bool northeast = false;
    bool northwest = false;
    bool southeast = false;
    bool southwest = false;

    //variable to denote whether the lattice changed during this cull step.
    bool changed = false;

    empty = true;

    //copy the data from the lattice to the temporary variable
    //memcpy(data, lat->data, sizeof(bool)*lat->size*lat->size);

    //loop through each site on the lattice
    for(int y=0; y<lat->size; y++)
    {
        for(int x=0; x<lat->size; x++)
        {
        }
// if the site is occupied
if (lat->data[y*lat->size+x] == true) {
    // check the northwest and southeast quadrants
    // Note: at each step we use fmod to impose periodic boundary conditions.
    northwest = lat->data[fmod(y-1, lat->size)*lat->size+fmod(x-1, lat->size)];
    southeast = lat->data[fmod(y+1, lat->size)*lat->size+fmod(x+1, lat->size)];
    // if neither is occupied
    if (!(northwest&&southeast)) {
        // then check the southwest and northeast
        southwest = lat->data[fmod(y+1, lat->size)*lat->size+fmod(x-1, lat->size)];
        northeast = lat->data[fmod(y-1, lat->size)*lat->size+fmod(x+1, lat->size)];
        // if neither is occupied then the cell is unsupported so set it to be empty and notice that the lattice has changed
        if (!(southwest&&northeast)) {  
            lat->data[y*lat->size+x] = false;
            changed = true;
        }
    }
    empty = false;
}
The Monte-Carlo renormalization root finding algorithm

Listing 5: C Monte-Carlo renormalization group root finding routine

```c
void MCCountPercolationRootFind(int L, int *bounds, int lowerbound,
                                 int error, MTRand *rnd)
{
    // instantiate a new Mersene Twister with seed (seed).
#ifdef SPIRAL
    int upperbound = 0.72*L*L;
    lowerbound = 0.4 * L*L;
#else
    int upperbound = 0.5*L*L;
    lowerbound = 0.3*L*L;
#endif
double critical = 0.0;
int percolates = 0;

    // create a blank lattice
Lattice *lat = CreateLattice(L);

    // variable to denote whether the culled lattice has any filled cells left.
```
bool empty = true;

//variables to aid in the construction of a random lattice
int *indices = (int*)malloc(sizeof(int)*L*L);
indices[0] = 0;
int lastindex = L*L-1;
int total = 50;
//holds a copy of the original lattice for use during the cull step
//bool *cullData = (bool*)malloc(sizeof(bool)*L*L);

//go through all possible numbers of filled sites
while(true)
{
    percolates = 0;
    total = 50;
    int occupied = (upperbound-lowerbound)/2+lowerbound;
    for(int j = 0;j<total;j++)
    {
        printf("Beginning\n");
        //Create a random lattice with (filled) filled sites.
        //go through and set the lattice to be empty and set the array
        //indices to contain a reference
        //to each site on the lattice
        printf("%d\n",L*L);
        for(int i = 0;i<L*L;i++)
        {
            indices[i] = i;
            lat->data[i]= 0;
for (int i = 0; i < occupied; i++)
{
    // choose a random index in the array of indices
    int index = rand->randInt(lastindex);
    // int index = rand()%(lastindex+1);
    // printf("%d\n",index);
    // set that position of the lattice to true
    lat->data[indices[index]] = true;
    // set the value at index to the value of the index at the end of the array
    indices[index] = indices[lastindex];
    // "move" the end of the array inwards.
    lastindex--;
}
// reset last index
lastindex = L*L-1;
// printf("Blahblahblah\n");
// fully cull the lattice using periodic boundary conditions
#ifdef SPIRAL
    //while(CullLatticePeriodic(lat, empty)){}
    empty = CullLatticePeriodicFast(lat);
#else
    // Lattice *tmplattice = CopyLattice(lat);
    empty = CullLatticePeriodicFast24NN(lat);
#endif
if (empty==false)
    percolates++;

if (j==9&&((percolates==10||percolates==0))
{
    total = 10;
    break;
}

printf("Finished\nLattice\nCull\n");

critical = ((double)percolates/(double)total);
if (critical < 0.5)
    lowerbound = occupied;
else
    upperbound = occupied;
if (upperbound-lowerbound<=error || upperbound-lowerbound <=1)
    break;

//free any assigned data
FreeLattice(lat);
//free(cullData);
free(indices);
delete rnd;
bounds[0] = lowerbound;
bounds[1] = upperbound;

// delete percList;
Finally, we present the fast culling algorithm.

Listing 6: C Monte-Carlo renormalization group root finding routine

```c
//Culls a lattice with spiral percolation using periodic boundary
//conditions

bool CullLargeLatticePeriodicFast(LargeLattice *lat) {

//variables used to store the states of the different spiral regions
bool northeast = false;
bool northwest = false;
bool southeast = false;
bool southwest = false;

bool **ltpy = new bool*[lat->size];
unsigned int **xcoords = new unsigned int*[lat->size];
unsigned int **ycoords = new unsigned int*[lat->size];

for(int i =0;i<lat->size;i++){
  ltpy[i] = new bool[lat->size];
  xcoords[i] = new unsigned int[lat->size];
  ycoords[i] = new unsigned int[lat->size];
}

int index = 0;
int sites = 0;
```
//copy the data from the lattice to the temporary variable
memcpy(data, lat->data, sizeof(bool)*lat->size*lat->size);
int x = 0;
int y = 0;
//loop through each site on the lattice
for (y = 0; y < lat->size; y++)
{
    for (x = 0; x < lat->size; x++)
    {
        lcopy[y][x] = 0;
        //if the site is occupied
        if (lat->data[y][x] == true)
            {
                sites++;
                //check the northwest and southeast quadrants
                //Note: at each step we use fmod to impose periodic boundary conditions.
                northwest = lat->data[fmod(y-1, lat->size)][fmod(x-1, lat->size)] ||
                             lat->data[y][fmod(x-1, lat->size)];
                southeast = lat->data[fmod(y+1, lat->size)][fmod(x+1, lat->size)] ||
                             lat->data[y][fmod(x+1, lat->size)];
                //if neither is occupied
                if (!(northwest \&\& southeast))
                    {
                        //then check the southwest and north east
                        southwest = lat->data[fmod(y+1, lat->size)][fmod(x-1, lat->size)] ||
                                     lat->data[fmod(y+1, lat->size)][x];
                        //...
northeast = lat->data[fmod(y-1, lat->size)][fmod(x+1, lat->size)] ||
        lat->data[fmod(y-1, lat->size)][x];

    // if neither is occupied then the cell is unsupported so set it to be empty and notice that the lattice has changed
    if (! (southwest & & northeast)) {
        //indizes[index] = y*lat->size + x;
        int yp = index \% lat->size;
        xcoords[index/lat->size][yp] = x;
        ycoords[index/lat->size][yp] = y;
        ltcp[y][x] = 1;
        index++;
    }
}
}
}
}
}
index--;  
x = 0; y = 0;
while (index>=0)
{
    //printf("%d\n", index);
    int yp = index \% lat->size;
    x = xcoords[index/lat->size][yp]; //indizes[index]\%lat->size;
    y = ycoords[index/lat->size][yp]; // (indizes[index]-x)/lat->size;
    // if the site is occupied
    if (lat->data[y][x]==true) 
    
}
lcpy[y][x] = 0;

//check the northwest and southeast quadrants
//Note: at each step we use fmod to impose periodic boundary conditions.
northwest = lat->data[fmod(y-1,lat->size)][fmod(x-1,lat->size)];
    lat->data[y][fmod(x-1,lat->size)];
southeast = lat->data[fmod(y+1, lat->size)][fmod(x+1, lat->size)];
    lat->data[y][fmod(x+1, lat->size)];

//if neither is occupied
if (! (northwest \&\& southeast)) {
    //then check the southwest and northeast
    southwest = lat->data[fmod(y+1, lat->size)][fmod(x-1, lat->size)];
        lat->data[y][fmod(x-1, lat->size)];
    northeast = lat->data[fmod(y-1, lat->size)][fmod(x+1, lat->size)];
        lat->data[y][fmod(x+1, lat->size)];

    //if neither is occupied then the cell is unsupported so set it to be empty and notice that the lattice has changed
    if (! (southwest \&\& northeast)) {
        sites --;
        lat->data[y][x] = false;
        if (lat->data[fmod(y-1, lat->size)][fmod(x-1, lat->size)]\&\&lcpy [fmod(y-1, lat->size)][fmod(x-1, lat->size)]==0) {
            //indices[index] = fmod(y-1, lat->size)*lat->size+fmod(x-1, lat->size);
            xcoords[index/lat->size][yp] = fmod(x-1, lat->size);
            ycoords[index/lat->size][yp] = fmod(y-1, lat->size);
ltcpy[fmod(y-1,lat->size)][fmod(x-1,lat->size)] = 1;
index++;
yp = index \% lat->size;
}
if(lat->data[y][fmod(x-1,lat->size)]&&ltcpy[y][fmod(x-1,lat->
size)]==0){
    //indices[index] = y*lat->size+fmod(x-1,lat->size);
xcoords[index/lat->size][yp] = fmod(x-1,lat->size);
ycoords[index/lat->size][yp] = y;
ltcp[y][fmod(x-1,lat->size)] = 1;
index++;
yp = index \% lat->size;
}
if(lat->data[fmod(y+1,lat->size)][fmod(x+1,lat->size)]&&ltcpy[
    fmod(y+1,lat->size)][fmod(x+1,lat->size)]==0){
    //indices[index] = lat->data[fmod(y+1,lat->size)*lat->size+fmod(
        x+1,lat->size)];
xcoords[index/lat->size][yp] = fmod(x+1,lat->size);
ycoords[index/lat->size][yp] = fmod(y+1,lat->size);
ltcp[y][fmod(x+1,lat->size)][fmod(x+1,lat->size)] = 1;
index++;
yp = index \% lat->size;
}
if(lat->data[y][fmod(x+1,lat->size)]&&ltcpy[y][fmod(x+1,lat->
size)]==0){
    //indices[index] = y*lat->size+fmod(x+1,lat->size);
xcoords[index/lat->size][yp] = fmod(x+1,lat->size);
ycoords[index/lat->size][yp] = y;
117  \texttt{ltcpy[y][fmod(x+1, lat->size)]=1;}
118  index++;\texttt{ yp = index \% lat->size;}
119  }
120  \textbf{if} (lat->data[fmod(y+1, lat->size)][fmod(x-1, lat->size)]\&\&\texttt{ltcpy[fmod(y+1, lat->size)][fmod(x-1, lat->size)]=0)}\{
121  \textit{// indices[index] = fmod(y+1, lat->size)*lat->size+fmod(x-1, lat->size);}
122  xcoords[index/lat->size][yp] = fmod(x-1, lat->size);
123  ycoords[index/lat->size][yp] = fmod(y+1, lat->size);
124  \texttt{ltcpy[fmod(y+1, lat->size)][fmod(x-1, lat->size)]=1;}
125  index++;\texttt{ yp = index \% lat->size;}
126  \}
127  \textbf{if} (lat->data[fmod(y+1, lat->size)][x]\&\&\texttt{ltcpy[fmod(y+1, lat->size)][x]==0)}\{
128  \textit{// indices[index] = fmod(y+1, lat->size)*lat->size+x;}
129  xcoords[index/lat->size][yp] = x;\textit{/fmod(x-1, lat->size);}
130  ycoords[index/lat->size][yp] = fmod(y+1, lat->size);
131  \texttt{ltcpy[fmod(y+1, lat->size)][x] = 1;}
132  index++;\texttt{ yp = index \% lat->size;}
133  \}
134  \textbf{if} (lat->data[fmod(y-1, lat->size)][fmod(x+1, lat->size)]\&\&\texttt{ltcpy[fmod(y-1, lat->size)][fmod(x+1, lat->size)]=0)}\{
135  \textit{// indices[index] = fmod(y-1, lat->size)*lat->size+fmod(x+1, lat->size);}
136  xcoords[index/lat->size][yp] = fmod(x+1, lat->size);
140 \text{ycoords[index/lat->size][yp] = fmod(y-1, lat->size);} \\
141 \text{ltcpy[fmod(y-1, lat->size)][fmod(x+1, lat->size)] = 1; } \\
142 \text{index++;} \\
143 \text{yp = index \% lat->size; } \\
144 \text{}} \\
145 \text{if(lat->data[fmod(y-1, lat->size)][x]\&\&ltcpy[fmod(y-1, lat->size)] = 0) \{} \\
146 \text{//indices[index] = fmod(y-1, lat->size)*lat->size+x; } \\
147 \text{xcoords[index/lat->size][yp] = x; //fmod(x-1, lat->size);} \\
148 \text{ycoords[index/lat->size][yp] = fmod(y-1, lat->size);} \\
149 \text{ltcpy[fmod(y-1, lat->size)][x] = 1; } \\
150 \text{index++;} \\
151 \text{yp = index \% lat->size; } \\
152 \text{}} \\
153 \text{\textbf{else} \{} \\
154 \text{ltcpy[y][x] = 0; } \\
155 \text{\}} \\
156 \text{\textbf{else} \{} \\
157 \text{ltcpy[y][x] = 0; } \\
158 \text{\}} \\
159 \text{\textbf{else} \{} \\
160 \text{//printf("\%d,\%d\n", x, y); } \\
161 \text{\} } \\
162 \text{index--; } \\
163 \text{\}} \\
164 \text{\textbf{for}(int i = 0; i<lat->size; i++) \{} \\
165 \text{\} } \\
166 \text{delete ltcpy[i];}
167    delete xcoords[i];
168    delete ycoords[i];
169  }
170    delete ltcpy;
171    delete xcoords;
172    delete ycoords;
173  //return whether the lattice changed during the iteration.
174  return sites==0;
175  }

8.3 Merzenne-Twister algorithm

The Merzenne-Twister algorithm used to generate random numbers. Was not written by me.

Listing 7: C Monte-Carlo renormalization group root finding routine

1   // MersenneTwister.h
2   // Mersenne Twister random number generator — a C++ class MTRand
3   // Based on code by Makoto Matsumoto, Takuji Nishimura, and Shawn Cokus
4   // Richard J. Wagner v1.0 15 May 2003 rjwagner@writeme.com
5
6   // The Mersenne Twister is an algorithm for generating random numbers.
7   // It
8   // was designed with consideration of the flaws in various other
9   // generators.
10  // The period, $2^{19937} - 1$, and the order of equidistribution, 623
11    // dimensions,
12  // are far greater. The generator is also fast; it avoids
13    // multiplication and
14  // division, and it benefits from caches and pipelines. For more
// see the inventors’ web page at http://www.math.keio.ac.jp/~matumoto/mt.html

// Reference
// M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-
// Dimensionally
// Equidistributed Uniform Pseudo-Random Number Generator", ACM
// Transactions on

// Copyright (C) 1997 – 2002, Makoto Matsumoto and Takuji Nishimura,
// Copyright (C) 2000 – 2003, Richard J. Wagner
// All rights reserved.

// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:

// 1. Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
// 2. Redistributions in binary form must reproduce the above
// copyright
// notice, this list of conditions and the following disclaimer in
// the
documentation and/or other materials provided with the
distribution.

//
// 3. The names of its contributors may not be used to endorse or promote
// products derived from this software without specific prior written
// permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
// FOR
// A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
// COPYRIGHT OWNER OR
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL
// ,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
// OF
// LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
// NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
// SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
// The original code included the following notice:
//
// When you use this, send an email to: matumoto@math.keio.ac.jp
// with an appropriate reference to your work.
It would be nice to CC: rjwagner@writeme.com and Cokus@math.washington.edu
when you write.

#ifndef MERSENNETWISTER_H
#define MERSENNETWISTER_H

// Not thread safe (unless auto-initialization is avoided and each
thread has
// its own MTRand object)

#include <iostream>
#include <limits.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

class MTRand {
  // Data
  public:
    typedef unsigned long uint32; // unsigned integer type, at least 32
    bits

    enum { N = 624 }; // length of state vector
    enum { SAVE = N + 1 }; // length of array for save()
enum { M = 397 }; // period parameter

uint32 state[N]; // internal state
uint32 *pNext; // next value to get from state
int left; // number of values left before reload needed

// Methods
public:
MTRand( const uint32& oneSeed ); // initialize with a simple uint32
MTRand( uint32 *const bigSeed, uint32 const seedLength = N ); // or an array
MTRand(); // auto-initialize with /dev/urandom or time() and clock()

// Do NOT use for CRYPTOGRAPHY without securely hashing several returned
// values together, otherwise the generator state can be learned after
// reading 624 consecutive values.

// Access to 32-bit random numbers
double rand(); // real number in [0,1]
double rand( const double& n ); // real number in [0,n]
double randExc(); // real number in [0,1)
double randExc( const double& n ); // real number in [0,n)
double randDblExc(); // real number in (0,1)
double randDblExc( const double& n ); // real number in (0,n)
uint32 randInt(); // integer in [0,2^32-1]
uint32 randInt( const uint32& n ); // integer in [0,n] for n <
double operator()() { return rand(); } // same as rand()

// Access to 53-bit random numbers (capacity of IEEE double precision)
double rand53(); // real number in [0,1)

// Access to nonuniform random number distributions
double randNorm( const double& mean = 0.0, const double& variance = 0.0);

// Re-seeding functions with same behavior as initializers
void seed( const uint32 oneSeed );
void seed( uint32 *const bigSeed, const uint32 seedLength = N );
void seed();

// Saving and loading generator state
void save( uint32* saveArray ) const; // to array of size SAVE
void load( uint32 *const loadArray ); // from such array
friend std::ostream& operator<<( std::ostream& os, const MTRand& mtrand );
friend std::istream& operator>>( std::istream& is, MTRand& mtrand );

protected:
void initialize( const uint32 oneSeed );
void reload();

uint32 hiBit( const uint32& u ) const { return u & 0x80000000UL; }
uint32 loBit( const uint32& u ) const { return u & 0x00000001UL; }
uint32 loBits( const uint32& u ) const { return u & 0x7fffffffUL; }
uint32 mixBits(const uint32& u, const uint32& v) const
{
    return hiBit(u) | loBits(v);
}

uint32 twist(const uint32& m, const uint32& s0, const uint32& s1) const
{
    return m ^ (mixBits(s0, s1) >> 1) ^ (-loBit(s1) & 0x9908b0dfUL);
}

static uint32 hash(time_t t, clock_t c);

inline MTRand::MTRand(const uint32& oneSeed)
{
    seed(oneSeed);
}

inline MTRand::MTRand(uint32 *const bigSeed, const uint32 seedLength)
{
    seed(bigSeed, seedLength);
}

inline MTRand::MTRand()
{
    seed();
}

inline double MTRand::rand()
{
    return double(randInt()) * (1.0/4294967295.0);
}

inline double MTRand::rand(const double& n)
{
    return rand() * n;
}

inline double MTRand::randExc()
{
    return double(randInt()) * (1.0/4294967296.0);
}

inline double MTRand::randExc(const double& n)
156  { return randExc() * n; }
157
inline double MTRand::randDblExc()
159  { return ( double(randInt()) + 0.5 ) * (1.0/4294967296.0); }
160
inline double MTRand::randDblExc( const double& n )
162  { return randDblExc() * n; }
163
inline double MTRand::randInt()
166  
167  { return a * 67108864.0 + b ) * (1.0/9007199254740992.0); // by Isaku Wada
168  }
170
inline double MTRand::randNorm( const double& mean, const double& variance )
172  { // Return a real number from a normal (Gaussian) distribution with given
173    // mean and variance by Box–Muller method
174    double r = sqrt( -2.0 * log( 1.0-randDblExc() ) ) * variance;
175    double phi = 2.0 * 3.14159265358979323846264338328 * randExc();
176    return mean + r * cos(phi);
177  }
179
inline MTRand::uint32 MTRand::randInt()
// Pull a 32-bit integer from the generator state
// Every other access function simply transforms the numbers extracted here

if ( left == 0 ) reload();
—left;

register uint32 sl;
s1 = *pNext++;
s1 ^= (s1 >> 11);
s1 ^= (s1 << 7) & 0x9d2c5680UL;
s1 ^= (s1 << 15) & 0x600000UL;
return ( s1 ^ (s1 >> 18) );

inline MTRand::uint32 MTRand::randInt( const uint32& n )
{
// Find which bits are used in n
// Optimized by Magnus Jonsson (magnus@smartelectronix.com)
uint32 used = n;
used |= used >> 1;
used |= used >> 2;
used |= used >> 4;
used |= used >> 8;
used |= used >> 16;

// Draw numbers until one is found in [0,n]
uint32 i;
do
doi = randint() & used;  //toss unused bits to shorten search
while(i > n);
return i;
}

inline void MTRand::seed(const uint32 oneSeed)
{
   //Seed the generator with a simple uint32
   initialize(oneSeed);
   reload();
}

inline void MTRand::seed(uint32 *const bigSeed, const uint32 seedLength)
{
   //Seed the generator with an array of uint32’s
   //There are 2^19937–1 possible initial states. This function allows
   //all of those to be accessed by providing at least 19937 bits (with
   //default seed length of N = 624 uint32’s). Any bits above the lower
   //32
   //in each element are discarded.
   //Just call seed() if you want to get array from /dev/urandom
   initialize(19650218UL);
   register int i = 1;
register uint32 j = 0;
register int k = ( N > seedLength ? N : seedLength );
for( ; k; --k )
{
    state[i] =
    state[i] ^ ( (state[i-1] ^ (state[i-1] >> 30)) * 1664525UL );
    state[i] += ( bigSeed[j] & 0xffffffffUL ) + j;
    state[i] &= 0xffffffffUL;
    ++i; ++j;
    if( i >= N ) { state[0] = state[N-1]; i = 1; }
    if( j >= seedLength ) j = 0;
}
for( k = N - 1; k; --k )
{
    state[i] =
    state[i] ^ ( (state[i-1] ^ (state[i-1] >> 30)) * 1566083941UL );
    state[i] -= i;
    state[i] &= 0xffffffffUL;
    ++i;
    if( i >= N ) { state[0] = state[N-1]; i = 1; }
}
state[0] = 0x80000000UL; // MSB is 1, assuring non-zero initial array
reload();
// Seed the generator with an array from /dev/urandom if available
// Otherwise use a hash of time() and clock() values

// First try getting an array from /dev/urandom
FILE* urandom = fopen( "/dev/urandom", "rb" );
if ( urandom )
{
    uint32 bigSeed[N];
    register uint32 *s = bigSeed;
    register int i = N;
    register bool success = true;
    while( success && i-- )
        success = fread( s++, sizeof(uint32), 1, urandom );
    fclose(urandom);
    if( success ) { seed( bigSeed, N ); return; }
}

// Was not successful, so use time() and clock() instead
seed( hash( time(NULL), clock() ) );

}  

inline void MTRand::initialize( const uint32 seed )
{
    // Initialize generator state with seed
    // See Knuth TAOCP Vol 2, 3rd Ed, p.106 for multiplier.
    // In previous versions, most significant bits (MSBs) of the seed affect
// only MSBs of the state array. Modified 9 Jan 2002 by Makoto Matsumoto.

register uint32 *s = state;
register uint32 *r = state;
register int i = 1;
*s++ = seed & 0xffffffffUL;
for( ; i < N; ++i )
{
    *s++ = ( 1812433253UL * ( *r ^ (*r >> 30) ) + i ) & 0xffffffffUL;
    r++;
}

inline void MTRand::reload()
{
    // Generate N new values in state
    // Made clearer and faster by Matthew Bellew (matthew.bellew@home.com)
    register uint32 *p = state;
    register int i;
    for( i = N - M; i--; ++p )
    *p = twist( p[M], p[0], p[1] );
    for( i = M; --i; ++p )
    *p = twist( p[M-N], p[0], p[1] );
    *p = twist( p[M-N], p[0], state[0] );
    left = N, pNext = state;
}
inline MTRand::uint32 MTRand::hash( time_t t, clock_t c )
{
    // Get a uint32 from t and c
    // Better than uint32(x) in case x is floating point in [0,1]
    // Based on code by Lawrence Kirby (fred@genesis.demon.co.uk)

    static uint32 differ = 0; // guarantee time-based seeds will change

    uint32 h1 = 0;
    unsigned char *p = (unsigned char *) &t;
    for( size_t i = 0; i < sizeof(t); ++i )
    {
        h1 *= UCHAR_MAX + 2U;
        h1 += p[i];
    }
    uint32 h2 = 0;
    p = (unsigned char *) &c;
    for( size_t j = 0; j < sizeof(c); ++j )
    {
        h2 *= UCHAR_MAX + 2U;
        h2 += p[j];
    }
    return (h1 + differ++) ^ h2;
}
inline void MTRand::save( uint32* saveArray ) const
{
    register uint32 *sa = saveArray;
    register const uint32 *s = state;
    register int i = N;
    for( ; i--; *sa++ = *s++ ) {} *sa = left;
}

inline void MTRand::load( uint32 *const loadArray )
{
    register uint32 *s = state;
    register uint32 *la = loadArray;
    register int i = N;
    for( ; i--; *s++ = *la++ ) {} left = *la;
PNext = &state[N-left];
}

inline std::ostream& operator<<( std::ostream& os, const MTRand& mtrand )
{
    register const MTRand::uint32 *s = mtrand.state;
    register int i = mtrand.N;
    for( ; i--; os << *s++ << "\t" ) {}
    return os << mtrand.left;
inline std::istream& operator>>( std::istream& is, MTRand& mtrand )
{
    register MTRand::uint32 *s = mtrand.state;
    register int i = mtrand.N;
    for( ; i--; is >> *s++ ) {}
    is >> mtrand.left;
    mtrand.pNext = &mtrand.state[mtrand.N-mtrand.left];
    return is;
}

#endif // MERSENNETWISTER.H