Drift and Trapping of Particles under Biased Motion on Disordered Lattices

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Hunnervir Singh

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2024

Supervisor: Prof. Deepak Dhar

Certificate

This is to certify that this dissertation entitled Drift and Trapping of Particles under Biased Motion on Disordered Lattices towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Hunnervir Singh at Indian Institute of Science Education and Research, Pune under my supervision during the academic year 2023-2024.

Deepak Dhar

Prof. Deepak Dhar

Committee:

Prof. Deepak Dhar

Prof. Sreejith Ganesh Jaya

This thesis is dedicated to the love of learning.

Declaration

I hereby declare that the matter embodied in the report entitled Drift and Trapping of Particles under Biased Motion on Disordered Lattices are the results of the work carried out by me at the Department of Physics, Indian Institute of Science Education and Research, Pune, under the supervision of Prof. Deepak Dhar and the same has not been submitted elsewhere for any other degree.

Hunner

Hunnervir Singh

Acknowledgments

Firstly, I would like to thank my supervisor, Prof. Deepak Dhar, whose expert guidance enabled me to complete my work while learning a lot. I would also like to thank Prof. Sreejith G.J. for being available even when he was on a sabbatical. Finally, I would like to thank my parents, friends, and family for their continued support.

Abstract

This project deals with the properties of systems of biased random walkers on disordered lattices. We use a percolation cluster to model the disordered lattice and study how the motion of the walkers is affected by the disorder in the lattice. I developed an algorithm to find the backbone and branches of a cluster. We find the steady state for a system of non-interacting biased walkers on a percolation cluster theoretically. Using this, we plot the average velocity of the walkers as a function of the bias. We simulate interacting biased random walkers with hard-core interactions on a percolation cluster to understand the current of the walkers as a function of bias and walker density. The long-time velocity-velocity autocorrelation function is a slowly varying function of the bias for interacting random walkers on a regular comb. This slow decay is due to the dynamic heterogeneity in the random walkers' motion, which means there are different regions where the walkers' motion differs. In the region inside the branches, the average velocity of the walkers is low as most are trapped beneath other walkers. Meanwhile, closer to the backbone, the walkers are free to move and have a larger average velocity. The velocity-velocity autocorrelation function also shows bumps corresponding to the walkers trapped at different depths. We find the occupation probabilities in a regular comb using the partition function and compare it to the occupation probabilities in our simulations and the theoretical occupation probability for an infinite comb. A walker deep inside the trap stays there for a long time, which can be observed by plotting the probability that the trapping time is greater than τ vs τ, which is a slowly decaying function of τ and shows steps corresponding to the walkers being trapped at different depths.

Contents

Introduction

The motion of fluids has been a fundamental problem in physics for a long time. We get an idea of the importance of this problem by the fact that one of the Millennium Prize Problems in mathematics is proving the existence of smooth solutions of the Navier-Stokes equation governing fluid flow. Difficulties in finding exact solutions have led to physicists trying simpler models to model fluid flow and transport phenomena. Random walkers have been used to model Brownian motion, diffusion processes and fluid flow for a long time[1]. Broadbent and Hammersley studied the percolation problem in 1957[2], and percolation clusters have been used as the prototypical model of disordered media ever since. The motion of random walkers on a disordered medium was studied by de Gennes in 1976 using the percolation cluster as the disordered medium[3].

Biased diffusion in a disordered medium is when particles are subjected to an external field when passing through a porous medium. Biased diffusion happens during the seepage of pollutants into the groundwater systems under gravity or during the displacement of molecules in gel electrophoresis under the influence of an electric field. We use biased random walkers moving on the percolation cluster with a bias g to model biased diffusion in a disordered medium. Barma and Dhar argued that for biased random walkers with no mutual interactions, the asymptotic velocity of the particles becomes zero at a finite value of the biasing field g_c after increasing linearly with g initially[4]. This happens due to the dead-end branches in the direction of the electric field. It is difficult for the particles to escape from such branches at high field values, so they stay in these branches for a long time. Thus, they are also known as *trapping branches/ traps*.

We model the short-range interactions between the particles using hard-core interactions. A simple process for studying transport phenomena is the *asymmetric simple exclusion process* (ASEP) defined by Frank Spitzer in 1970[5]. This process has become the paradigm for studying transport phenomena and non-equilibrium statistical physics in general[6]. We simulate walkers following this process. R. Ramaswamy and M. Barma previously studied the transport of interacting biased random walkers on a percolation cluster in 1987[7]. They studied the behaviour of the average velocity of the particles as a function of the density and the bias.

A recent paper by C. Iyer, Barma and Dhar has claimed that the fractional number of particles that have been in the same side branch for a time interval greater than T_w varies as $\sim \exp\left(-c\sqrt{\log(T_w)}\right)$ for large T_w, where the constant c depends only on the bias[8]. This slow decay is a general feature of systems showing dynamic heterogeneity, meaning that different regions show different particle motion behaviours. In the region close to the deep-end traps, particles stay stuck for a long time, whereas in the region close to the backbone, the particles are free to move.

I verified the non-linear velocity of non-interacting biased random walkers on a percolation cluster as a function of bias. We used simulations and the steady-state probability distribution for this system of biased noninteracting walkers on a cluster. I simulated interacting random walkers with hard-core interactions to study the velocity of the walkers as a function of bias and the density of walkers. To understand the trapping times in a single branch, we used a conceptually simpler lattice: the regular comb. I measured the long-time velocity-velocity autocorrelation function. The slow decay in this function indicates that the velocity of a walker remains similar for a long time. This function also shows bumps corresponding to the trapping at different depths. I also found how different depths contribute to the different trapping times of the particle by plotting the probability that the trapping time is greater than τ vs τ given different maximum depths reached by the walker.

In Chapter 1, I recall some basic features of the percolating systems in Section 1.1 and an introduction to Markov processes is given in Section 1.2, a familiarity with these topics is needed to understand the rest of my thesis work. Chapter 2 deals with the percolation clusters. Section 2.1 outlines the algorithm for finding the percolation clusters. I did most of my simulations of the random walkers' motion on spanning percolation clusters. In Section 2.2, I discuss the algorithm for finding a cluster's backbone and branch sites. The knowledge of these sites is essential because the time-averaged current of the walkers through a branch site must be zero for $t \to \infty$.

In Chapter 3, we simulate a single biased random walker on a percolation cluster. For random walkers with no mutual interactions, it is known that the asymptotic velocity of the particles becomes zero at a finite value of the bias E_c after increasing linearly with E initially[4]. This happens due to the presence of branches of random lengths in the direction of the biasing field. As it is difficult for the particles to escape from such branches at high values of biases, particles stay in them for a long time. Thus, they are also known as *trapping branches/ traps.*

Previous studies have shown that the time taken by the walker current to reach its steady-state value is very high and can be hard to compute in a reasonable CPU time using simulations only. The time needed to see the asymptotic behaviour diverges roughly as $exp(c/|E_c - E|)$, as E approaches E_c from below, as $L \to \infty$[9]. So, we study the Markovian evolution using simulations to study how the probability of the walker being at different sites evolves with time in Chapter 4. We observe that the system tends to a steady state. This steady state is found exactly in Section 4.3 by solving the eigenvalue equation of the Markov time evolution matrix.

In Chapter 5, we simulate interacting biased walkers and discuss how the system's behaviour changes upon adding interactions to our system. Chapter 6 discusses the results of our simulation of interacting biased random walkers on the regular comb. Here, we try to understand the dynamic heterogeneity in the system by using the velocity-velocity autocorrelation function. Using the partition function, we find the occupation probabilities at different depths for a finite lattice. We also find the distribution of trapping times for this lattice. Chapter 7 will outline my thesis work's conclusions and directions for some future work.

Chapter 1

Preliminaries

1.1 Percolation Theory

Consider a lattice, for instance, a square lattice consisting of occupied and unoccupied sites, and let p be the probability of a vertex v_i being occupied (occupation probability). Two occupied sites are said to be *connected* if we can go from one site to the other using a set of occupied neighbouring sites. A *cluster* is a maximally connected lattice component. Different clusters correspond to different maximally connected lattice components, as shown in Fig. 1.1. Broadbent and Hamersley have studied such clusters and their properties in 1957[2]. There has been a lot more subsequent work in this field known as *percolation theory*[10].

An important property of the percolation process is the existence of two phases: with and without long-range connectivity. A second-order phase transition occurs at $p = p_c$. At $p > p_c$, the largest cluster spans the length of the entire lattice (meaning that we get an infinite length cluster in the $L \to \infty$ limit), which is not the case for $p < p_c$. This value p_c is the *percolation threshold.* We define the backbone and the branches on this spanning cluster. Sites in this cluster with two disjoint paths leading to infinity belong to the *backbone.* The rest of the sites belong to the *branches.* There is another threshold at which there is a change in the properties of the cluster at $p = p_d > p_c$, where p_d is the *directed percolation threshold.* At $p > p_d$, a directed path exists through the lattice along the occupied sites. A directed path is a path that does not 'turn back'; that is, along that path, the component of displacement is always non-negative along a specific direction.[10]

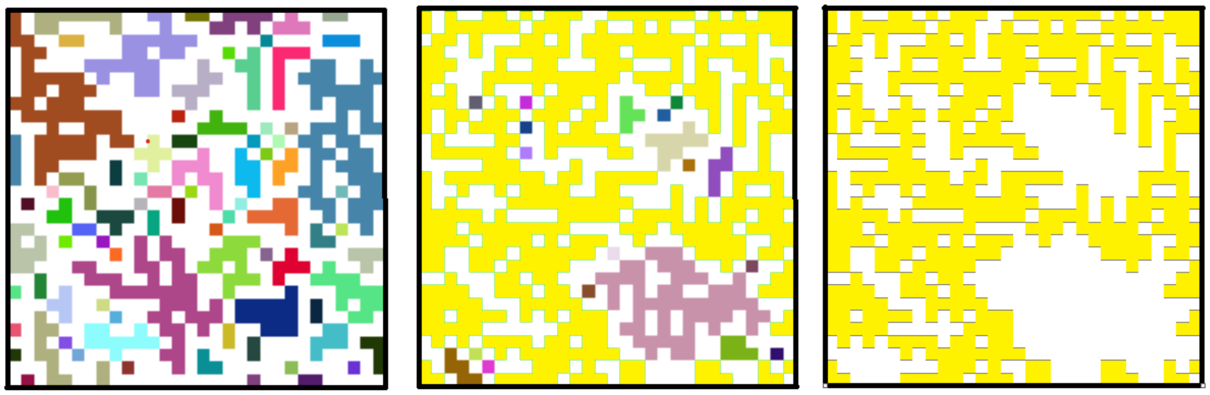


Figure 1.1: Examples of different percolation clusters on a square lattice with $l = 30$ are shown in different colours. All clusters at $p = 0.5$ are shown (left), all clusters at $p = 0.6$ (centre), largest percolating cluster at $p = 0.6$(right). Site percolation threshold for a square lattice, $p_c = 0.59274605$

1.2 Markov Process

A system is said to undergo a continuous time *Markovian evolution* if the state of the system at time $t + dt$ depends only upon the state of the system at time t, which means:

$$\frac{dP_n(t)}{dt} = \sum_m (W_{nm}P_m(t) - W_{mn}P_n(t)) \tag{1.1}$$

$P_n(t)$ is the probability that the system is in configuration n at time t. W_{mn} is the Markov matrix element, and the probability that the system goes from configuration n to configuration m is $W_{mn}dt$ in an infinitesimal time dt. Where, $W_{nn} = 1 - \sum_{m \neq n} W_{nm}$.

In the discrete-time case, we can rewrite the equation for the Markovian evolution(Eq. 1.1) using a vector $|P(t)\rangle = \{P_n(t)\}$ containing the probabilities of the system's different configurations at time t, and the Markov matrix W where W_{ij} is the probability of going from configuration j to configuration i in one timestep, the evolution equation becomes:

$$|P(t+1)\rangle = W|P(t)\rangle \tag{1.2}$$

If our Markov process is such that there is a path from every state to every other state, the chain is called *irreducible*. The process always has a unique steady-state probability distribution if our chain is irreducible. The steady state corresponds to the state vector

$|P(t)\rangle$ being independent of time. So, it corresponds to the eigenvector $|P^*\rangle$ with eigenvalue 1 of the above matrix W:

$$|P^*\rangle = W|P^*\rangle \tag{1.3}$$

Chapter 2

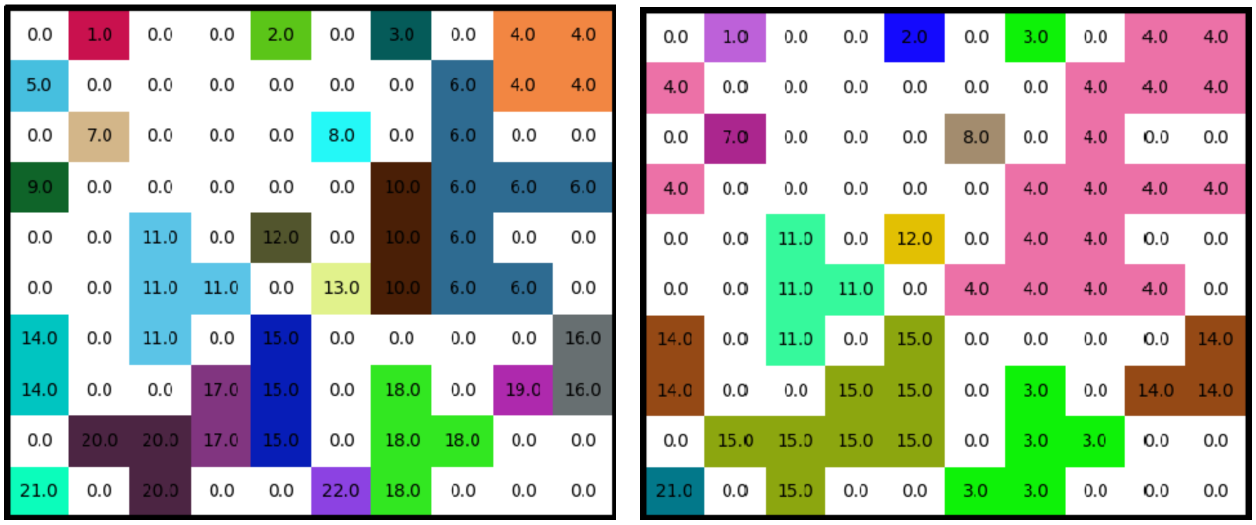
Modelling the Percolation Clusters

We use a 2-D matrix containing all zeroes to represent a square lattice containing all unoccupied sites. We randomly assign 1's at each site with probability p to generate a realisation of the percolation process on a square lattice with occupation probability p. This lattice with occupied and unoccupied sites contains several clusters. To identify these clusters, we use the Hoshen-Kopelman algorithm.[11]

2.1 Hoshen-Kopelman Algorithm to Identify Different Clusters

In the Hoshen-Kopelman Algorithm, we scan the lattice from left to right row by row, starting from the first row, and check the given site and its top and left neighbours. We make a 2-D matrix containing all zeroes. This matrix will store the cluster numbers of the occupied sites on the lattice. In the first scan, we assign cluster no. 1 to the first occupied site. For successive sites, we follow the following rule:

- If the scanned site has an already scanned site neighbour to the top or left, we give the site the same cluster number as the minimum cluster number of its neighbours.
- If the scanned site does not have an already scanned neighbour to the top or left, we give the site a cluster number that is one larger than the highest cluster number we



(a) initial Hoshen-Kopelman clusters

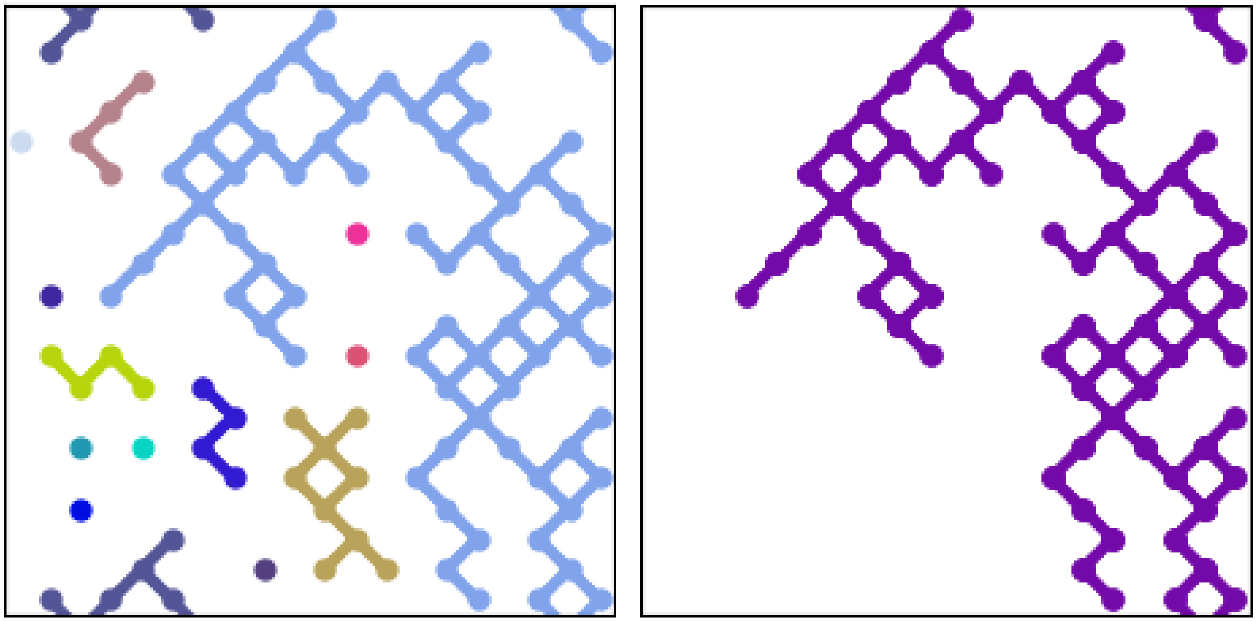
(b) final Hoshen-Kopelman clusters

Figure 2.1: Numbering in the initial step of the Hoshen-Kopelman algorithm, and the final numbering for an $l = 10$ lattice at $p = 0.4$, identified clusters are shown in different colours. The initial cluster identification is incomplete.

have assigned.

One scan is complete after doing this for all sites on the lattice. If the coloured sites represent the occupied sites, an example of the cluster numbers we get at this step's end is shown in Fig. 2.1a. So, we have generated the initial clusters without using the periodic boundary condition. Now, we redefine the neighbours using the periodic boundary conditions. Using these initial clusters, we scan the lattice repeatedly until there is no discrepancy between the neighbouring cluster numbers. The rules for this algorithm are as follows:

- During a scan, we search and compare each site with the neighbours to the top and left and give it the minimum cluster number among its neighbours(including the site itself).
- For this comparison, we use a list of connectivities such that $list[i]$ is the smallest cluster number connected to the ith cluster.
- We start with $list[i] = i$ for all i. Whenever we find a discrepancy in cluster numbers between sites i, its top neighbour j or its left neighbour k, we change the cluster numbers $list[i], list[j], list[k]$ to $min(list[i], list[j], list[k])$ so that now they are a part of the same cluster numbered $min(list[i], list[j], list[k])$.



(a) Different tilted clusters (b) Largest cluster among the tilted clusters

Figure 2.2: Different clusters found using the Hoshen-Kopelman algorithm on a tilted lattice using Periodic Boundary conditions at $p = 0.6$, in a 20×20 lattice.

- When there is no discrepancy left, we change the list to make $list[i] = list[list[i]]$ by making sure that every cluster number that we have stored for the connecting cluster is the minimum cluster number found connected to that cluster in this iteration.
- We again scan the lattice and repeat this algorithm until $list[i] = list[list[i]]$ without changing the list after checking the discrepancies.
- Now, we enumerate the clusters with the final cluster numbers by changing i to $list[i]$ in the original cluster numbers matrix.

We save the cluster with the largest number of sites in a file. This cluster corresponds to the infinite cluster for $p > p_c$, and we use it to study the motion of biased random walkers on the infinite percolation cluster. For a tilted square lattice, we have to slightly modify this algorithm to find the infinite cluster on a tilted lattice because of the difference in the definition of the periodic boundary conditions.

A tilted percolation cluster differs from a percolation cluster on a regular square lattice in the boundary conditions. Figure 2.2 shows the boundary conditions for tilted clusters.

We use these boundary conditions to define each site's top-left, top-right, bottom-left and bottom-right neighbours and find the largest cluster using the Hoshen-Kopelman algorithm. Using the Hoshen-Kopelman algorithm, in this case, will require comparing the cluster number on our given site to its top-left and top-right neighbours instead of the top and left neighbours during each scan.

2.2 Identifying the Backbone and Branches on a Cluster using DFS

We defined the backbone for infinite clusters as the set of sites with two disjointed paths leading to infinity. The clusters in our simulations are finite in size. So, we define the backbone as the cluster's largest biconnected component. A *biconnected component* is a subcluster that cannot be broken into two disconnected parts by removing a single site. The remaining sites on the cluster are called *branch sites*. We use the DFS (Depth First Search) algorithm to identify the backbones and branches on the cluster.[12] Initially, we define the backbone list as the list of all sites on the cluster. Then, we remove all the sites in a simple dead end. Figure 2.3a shows the removed simple dead ends. These are the dead ends whose last site is attached to the backbone only through sites with two neighbours. Removing one neighbour from any of those sites will make that site a new dead-end site. To remove such dead ends:

- We remove all sites with only one neighbour from the backbone list, as one neighbour means the dead-end site which does not belong to the backbone.
- We repeat this with the new backbone list obtained until no site with one neighbour is left.

We must go beyond this simple algorithm to remove the more complicated branches. In this, we use the DFS algorithm to partition the cluster into different biconnected components and remove all the smaller ones until the biconnected components belonging to the backbone are the only ones left.

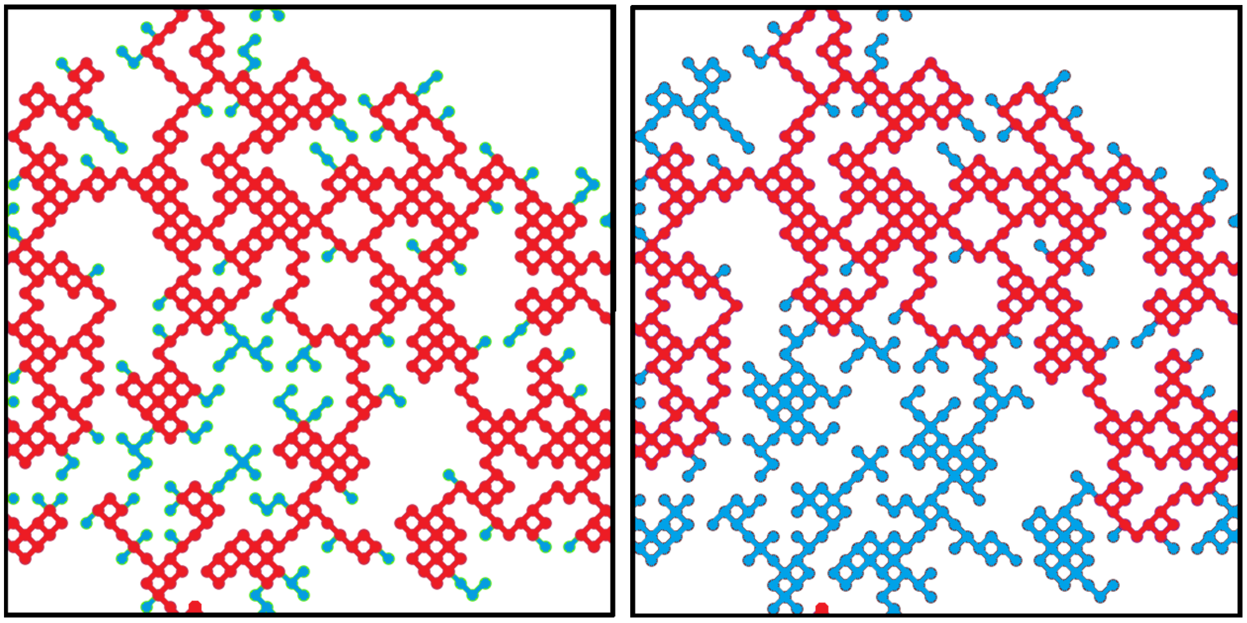

(a) Simple dead ends shown in blue are removed (b) Final backbone and branches

Figure 2.3: The largest cluster on a 50×50 lattice, at $p = 0.6$ is shown. The backbone sites are shown in red, and the branch sites are shown in blue.

- We start a loop over all the sites remaining in the backbone list. We label the chosen site *start*.
- We remove the *start* site from the backbone array if it has one or zero neighbours and only progress with the next step if it has two neighbours.
- Now, we will decide if the *start* site with degree two belongs to the backbone. We use another loop under the previous loop that ends only when it decides whether this site belongs to the backbone.
- We traverse along one *start*'s neighbours. During this traversal, we visit only the new sites we have not visited previously and save the path as P_1. We keep going until we reach a site where no new neighbouring sites are available.
- When we reach such a site, we go back along the path P_1 until we return to a previously visited site with unvisited neighbours.
- From this site, we go to the unvisited neighbours and save all the visited sites into a list $visited_1$.

- We repeat this algorithm till we are back at the starting site. If we have visited all the sites in the backbone array, the site *start* is biconnected. Otherwise, we start traversing toward the second neighbour of *start*.
- We use a similar traversal algorithm(DFS) for traversal in the direction of the other neighbour, and the visited sites are saved into the list $visited_2$. We will continue until we are back at the *start* site.
- If we have now visited all the sites in the backbone array, this means that site *start* is biconnected.
- If both $visited_1$ and $visited_2$ lists are smaller than the backbone list, we cannot visit all the sites through any one path. So, the site *start* is not biconnected (is an *articulation point*).
- We now see which list is smaller $visited_1$ or $visited_2$, and remove all the sites belonging to the smaller list, which means that they are the sites in the direction of the dead end of the backbone starting from one of the neighbours of *start*.

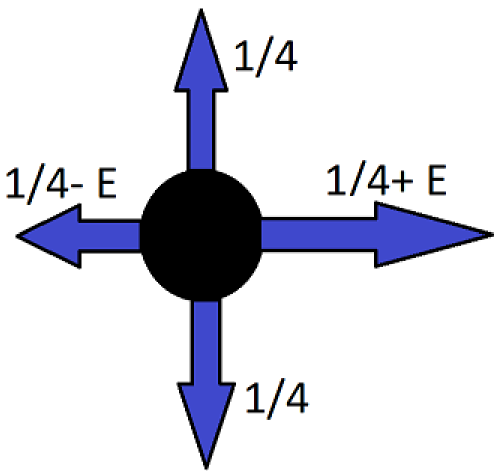
Figure 2.3b shows the backbone and branches on a tilted infinite cluster found using our algorithm. This figure shows that our algorithm works well for finding the backbone and branch sites on a percolation cluster.

Chapter 3

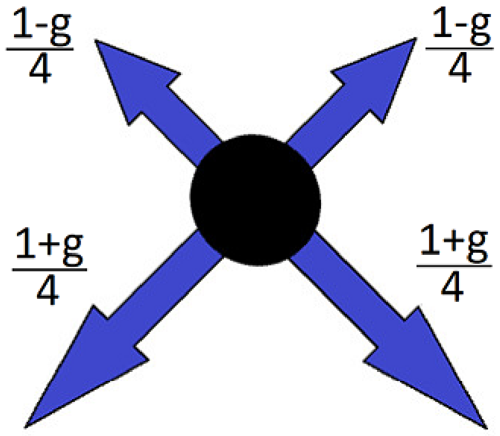
Simulation of Non-interacting Biased Random Walkers on the Percolation Cluster

3.1 Definition of the Model

In a system containing many non-interacting random walkers, the motion of one walker is not affected by the presence/ absence of other walkers. So, we use a single walker to simulate multiple non-interacting random walkers. To simulate the biased motion of a single walker on the percolation cluster found above, we use what is known in the literature as a *blind random walker*. By 'blind', we mean that at each time step, the particle chooses a neighbouring site randomly with $p(\text{up}) = p(\text{down}) = 1/4$ and $p(\text{left}) = 1/4 - E$, $p(\text{right}) = 1/4 + E$ where $0 \leq E \leq 1/4$ without considering its occupation as shown in Fig. 3.1a. The walker jumps to the chosen site only if that site is occupied. Otherwise, it remains stationary for the time step. So, the probability of going up and down is independent of the biasing field(E), which is towards the right, thus biasing the motion of the random walker towards the right (higher probability). We measure the displacement of the walker at each timestep, using which we calculate the average velocity. Because of the periodic boundary conditions, we cannot find the velocity simply by using the difference between the final and initial positions.



(a) Biases for walkers on a square lattice, where $0 \leq E \leq 1/4$



(b) Biases for walkers on a tilted square lattice, where $0 \leq g \leq 1$

Figure 3.1: Biases for the walkers on different types of lattices used in our simulations

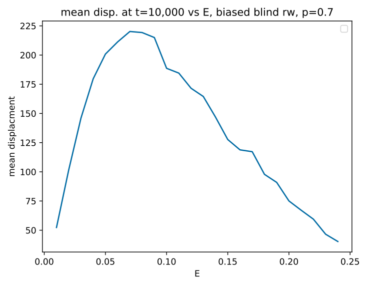
3.2 Simulation Details

We simulated the motion of a single biased random walker for $10,000$ timesteps on the largest percolation cluster on a 1000×1000 square lattice at occupation probability $p = 0.8$. We averaged our simulation results over 500 runs of the walker starting from random initial positions on the cluster.

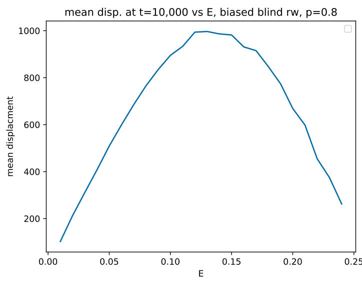
3.3 Results and Discussions

For random walkers with no mutual interactions, the asymptotic velocity of the particles becomes zero at a finite value of the bias E_c after increasing linearly with E initially.[4] This happens due to the presence of trapping branches. Using our simulation, we checked the non-linearity of velocity as a function of the biasing field by plotting the mean displacement vs bias(E) graph, as shown in Figure 3.2.

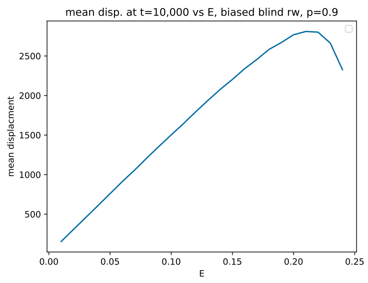
The density of traps of length r is an exponentially decreasing function of r, and $Prob(r) \propto e^{-Ar}$, where A is a constant for a particular value of the occupation probability p. Whereas



(a) v vs E at $p = 0.7$



(b) v vs E at $p = 0.8$



(c) v vs E at $p = 0.9$

Figure 3.2: Result of Mean displacement vs E for Monte Carlo random walks of 500 non-interacting walkers on a percolation cluster at $p = 0.8$ on a 1000×1000 square lattice after $10,000$ timesteps. (for a square lattice, $p_d = 0.70548$)[13]

the trapping time in a trap of depth r varies as λ^r, where $\lambda = (1+4E)/(1-4E)$ is the ratio of the probability of the walker moving along and opposite to the bias. So the average time a walker stays in branches is given by:

$$< \tau >= \sum_{r=0}^{\infty} \lambda^r e^{-Ar} \tag{3.1}$$

The current goes to zero when the average trapping time $\to \infty$. This sum goes to infinity at $E = E_c$. As the sum of a geometric progression, it goes to ∞ only when $\lambda e^{-A} > 1$. In the right-hand side of equation 3.1, there is a more significant contribution to the trapping time due to the larger branches when $\lambda e^{-A} > 1$.[14] But our simulations are on a finite-size lattice and did not have very large traps. Thus, we cannot observe the zero current regime in Figure 3.2. We try to keep $p > p_d$ for the walkers because we want at least one path through which the walkers can move without an opposing bias, such that it is always possible for a walker to keep on moving constantly through the lattice even at very high bias values.

If we define the quantity *bias-induced length* ($L(E)$) as the length of the trap in which the occupation probability of a walker changes by a factor of e. Then the probability that a walker gets out of a trap of length d is given by:

$$Prob(d) = e^{-d/L(E)}, \text{ where } L(E) = \left(log\left(\frac{1+E/4}{1-E/4}\right)\right)^{-1}$$

Walkers spend most of the time inside the bias-induced length of a trap. So, the walkers can leak out of the traps to give a finite current when the average length of the traps is lesser

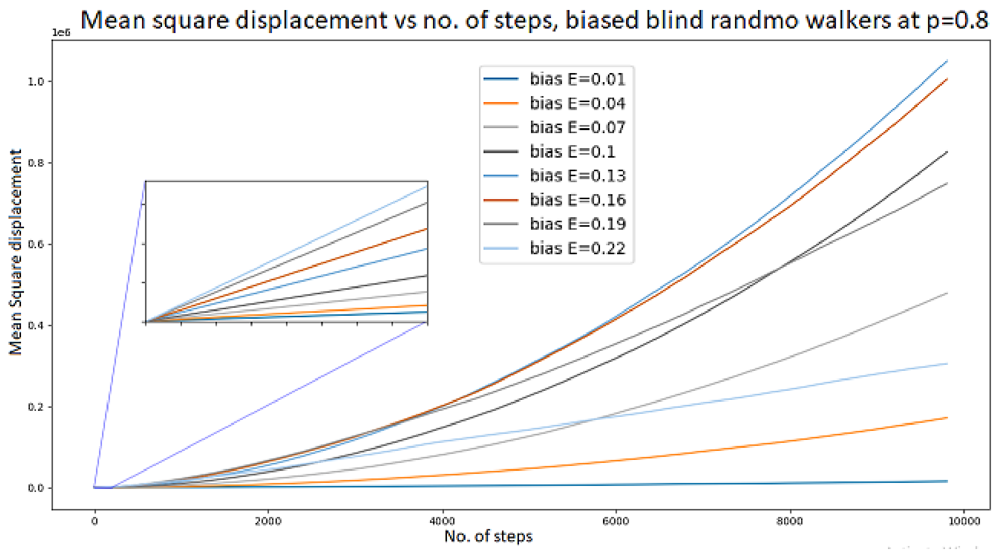


Figure 3.3: Result of Mean Displacement square vs no. of steps for Monte Carlo random walks of 500 non-interacting walkers in a 1000×1000 square lattice with $p = 0.8$ for different values of E.

than the bias-induced length. This condition is derived from the previous condition for the the zero current phase found from equation 3.1.

Previous studies have shown that the regime where the current of the particles goes to zero takes a significant amount of time to observe using only simulations as the time needed to see the asymptotic behaviour diverges roughly as $exp(c/|E_c - E|)$, as E approaches E_c from below, as $L \to \infty$.[9] We can also observe from Figure 3.3 that the steady state is not reached instantaneously. Initially, the velocity is faster for higher values of E, but as the walkers approach larger traps, the velocity becomes slower for the higher values of E because of the more strongly trapped walkers.

Chapter 4

Using Markov Matrix to Study a Single Walker

4.1 Defining the Markov Matrix

The system containing a single random walker on the percolation cluster follows a Markovian evolution in discrete time, as the system's state at time $t+1$ depends only upon the state of the system at time t. Let $P_n(t)$ be the probability that the walker is on the site n at time t. Let W_{mn} be the mn^{th} element of the transition matrix, which is the probability that the walker goes from site n to site m in one timestep. So:

$$W_{mn} = \begin{cases} 1/4 & \text{if } m \text{ is occupied and above/below } n \\ 1/4 + E & \text{if } m \text{ is occupied and to the right of } n \\ 1/4 - E & \text{if } m \text{ is occupied and to the left of } n \\ 0 & \text{if } |\vec{m} - \vec{n}| > 1 \text{ or m or n is unoccupied} \\ 1 - \sum_{m \neq n} W_{mn} & \text{if } m \text{ is occupied and } m = n \end{cases} \tag{4.1}$$

Where W_{nn} is the probability that the random walker stays at the same site n for one time step, and its value depends upon the occupancy of the neighbouring sites of n. Now, we can rewrite the equation for the Markovian evolution of this system using the discrete-time Markovian evolution equation(Eq. 1.2). Here, the vector $|P(t)\rangle = \{P_n(t)\}$ contains the probabilities of the particle being at different sites at time t.

Starting from $|P(0)\rangle = e_i$, where e_i is the i'th standard basis, we can use the Markov Matrix to see how the system evolves with time as:

$$|P(t)\rangle = W^t |P(0)\rangle \tag{4.2}$$

The resulting vector $|P(t)\rangle$ tells us the probability of the walker being at each site at time t after starting from the i'th site. The results of this evolution are shown in Figure 4.1.

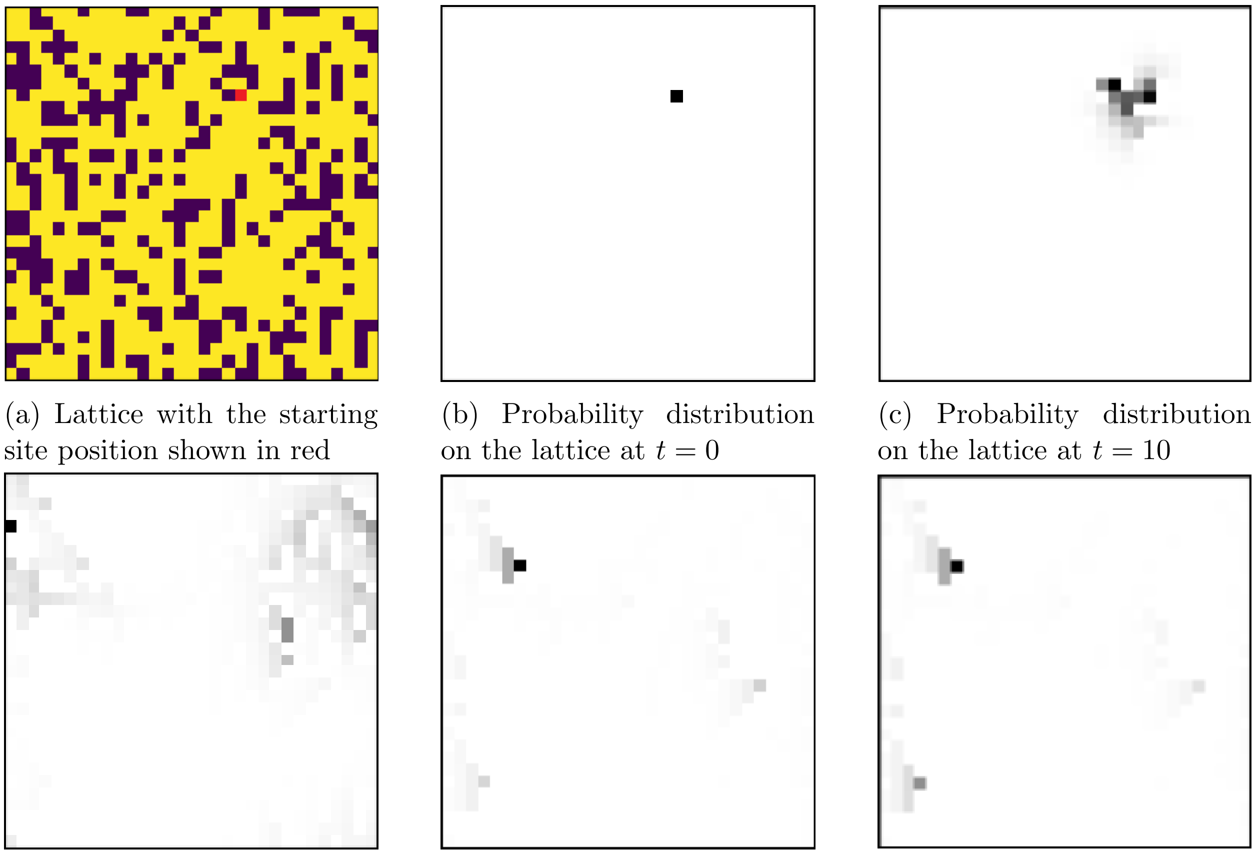
4.2 Time Evolution using the Markov Matrix

Using the Markov matrix, we observe how the probability distribution evolves with time in Figure 4.1. We can see in Figure 4.1c that the probability distribution evolves to the right side of the starting point as the bias E is towards the right direction. Also, Figure 4.1e and 4.1f show that the probability distribution of a walker becomes stationary after some time, independent of the initial conditions. We reached this state starting from a random initial position of the walker; this implies that this system reaches a steady state after a long time independent of the initial position.

The movement of a single-biased random walker on a connected cluster with a finite bias is a Markov process where every site is reachable. If our specific Markov process is such that there is a path from every state to every other state, then there is always a steady state. This steady state is reached as $t \to \infty$, giving us the probability of random walkers being at different sites in the steady state. In the next section, we will discuss the method to find this steady state and use the steady state distribution to calculate the average velocities of the walkers.

4.3 Steady-State Using Gauss-Seidel Method

The system is at a steady state, which means that the state vector $|P(t)\rangle$ does not change with time. So, it corresponds to the eigenvector $|P^*\rangle$ with eigenvalue 1 of the above matrix W. We use the Gauss-Seidel method to solve the eigenvalue equation 1.3. Gauss-Seidel



(a) Lattice with the starting site position shown in red

(b) Probability distribution on the lattice at $t = 0$

(c) Probability distribution on the lattice at $t = 10$

(d) Probability distribution on the lattice at $t = 100$

(e) Probability distribution on the lattice at $t = 1000$

(f) Probability distribution on the lattice at $t = 10000$

Figure 4.1: Time evolution of the probability distribution on the lattice shown in 4.1a from the initial $p = 1$ at a single point shown in 4.1b. Darker means a higher probability of being at that place after t timesteps starting from 4.1b. The lattice is of size 31×31, and the occupation probability is $p = 0.7$.

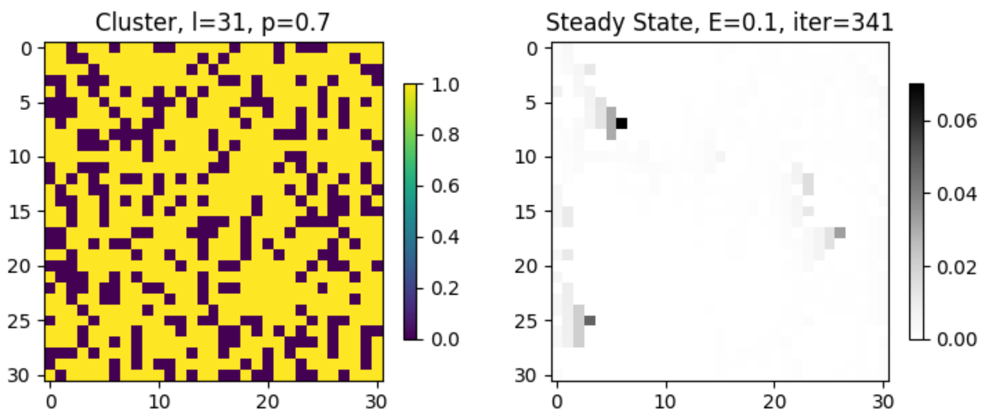


Figure 4.2: An example of exact steady-state found using the Gauss-Seidel method on a square lattice of size 31×31, at $p = 0.7$, and $E = 0.1$.

method is a basic iterative method for solving linear equations[15]. We use this method instead of Gaussian elimination as numerical methods are better for a large set of equations computationally than elimination methods because of the large round-off errors during the elimination process. We can also specify the maximum error allowed in our answer when finding the solution using the Gauss-Seidel method.

In the Gauss-Seidel Method, we start with a guess solution to the eigenvector equation and keep on improving this guess solution at each iteration till it converges. Let the guess eigenvector at the 0^{th} iteration be $|P^*\rangle_{G0} = 1/n$ to satisfy the normalisation condition. Where $n =$ no. of occupied sites. We use Equation 1.3 to write $|P^*\rangle_{Gti}$(guess steady state probability at the t^{th} iteration of the Gauss-Seidel method at site i) in terms of values $|P^*\rangle_{Gtj}$'s for $j > i$ and $|P^*\rangle_{G(t-1)j}$'s for $j < i$. At each iteration we start from $i = 1$ and update the value of $|P^*\rangle_{Gt1}$ in terms of $|P^*\rangle_{G(t-1)j}$'s for $j \neq 1$. Using this new value of $|P^*\rangle_{Gt1}$ and the previous values for all $|P^*\rangle_{G(t-1)j}(j = 1, n$ and $j \neq 1, 2)$ we calculate $|P^*\rangle_{Gt2}$ and so on using the expression:

$$P_{Gti} = \frac{\sum_{j=1}^{i-1} W_{ij} P_{Gtj} + \sum_{j=i+1}^{n} W_{ij} P_{G(t-1)j}}{1 - W_{ii}} \tag{4.3}$$

One iteration of the Gauss-Seidel method is complete when we have found $|P^*\rangle_{Gtj}$ for all the sites j using the values of $|P^*\rangle_{Gti}$'s for $1 < i < j$ and the values of $|P^*\rangle_{G(t-1)i}$'s for $j < i < n$ using equation 4.3. After each iteration, we renormalise $|P^*\rangle_{Gt}$ to make the total probability 1. We stop this algorithm when the absolute difference in the eigenvector at the $t+1$'th iteration and the t'th iteration becomes smaller than a previously specified tolerance value. This tolerance value specifies the round-off error in the solution of our eigenvalue equations, i.e. $|P^*\rangle$. Thus, we find the steady-state eigenvectors $|P^*\rangle$ containing the steady-state probabilities at different sites $\{p_i^*\}$. The steady-state probability distribution on the lattice studied in Figure 4.1a is shown along with the lattice in Figure 4.2.

Using the steady-state probability distribution $\{p_i^*\}$, we can calculate the mean velocity of a random walker in the steady state of the lattice. Where $p_i^* W_{ji}$ is the probability that a walker is present at site i in the steady state and that the walker moves to the site j at the next time step. So, we can write the mean velocity at a timestep as:

$$\vec{v}_{walk} = \sum_{i,j} (\vec{j} - \vec{i}) p_i^* W_{ji} \tag{4.4}$$

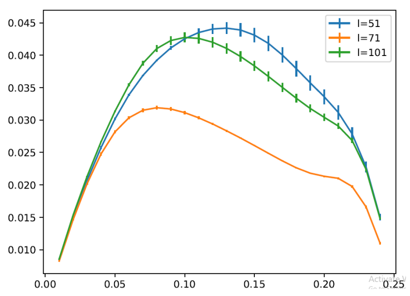
Where $\vec{j}$ represents the position of site j.

4.4 Simulation Details

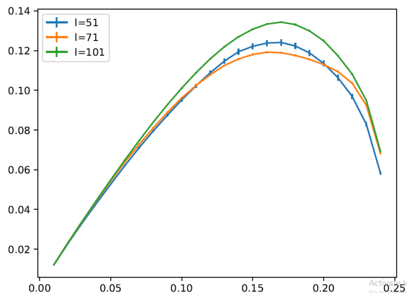
We find the largest percolation clusters for six lattices, each of lengths $L = 51$, $L = 71$, and $L = 101$ at $p = 0.7$, $p = 0.8$ and $p = 0.9$ on a regular square lattice. This gives us 54 different clusters, six for each L and p. We find the steady-state probability distributions for the Markov matrix defined in Equation 4.1 on these clusters at different values of E from $E = 0.01$ to $E = 0.24$ with 0.01 increment. We use the Gauss-Seidel method to find the steady-state distributions. Our algorithm ends when tolerance=0.01. Using these steady-state probabilities, we find the steady-state velocity on a lattice at a particular value of E using Equation 4.4. Averaging the velocity for different lattices of the same length, for the same value of p and E, we get Figure 4.3.

4.5 Results and Discussion

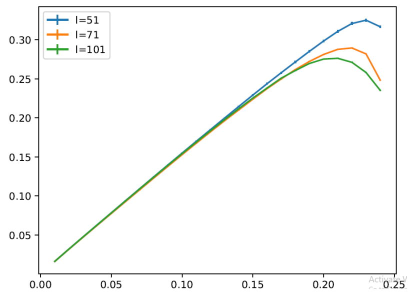
We can see that the steady-state probability distribution is similar to the long-time probability distribution of the position of a random walker starting from any initial state, as



(a) v vs E at $p = 0.7$



(b) v vs E at $p = 0.8$



(c) v vs E at $p = 0.9$

Figure 4.3: Graphs of mean velocity vs bias obtained using the steady-state of non-interacting walkers over six different lattices at lattice sizes of 51 shown in blue, 71 shown in yellow and 101 shown in green with tolerance for the convergence of Gauss-Seidel algorithm= 0.01.

shown in 4.1f. This similarity in the long-time state and the exact steady-state implies that the system eventually reaches this steady-state probability distribution. We can also observe that the probability of the walker being in a large trap is significantly higher than at other places in Figure 4.2. So, we can see that at significant values of bias E, a walker is more likely to be in a trap.

We observe that the peak of the random walker current shifts towards higher biases E for higher occupation probability p. Two factors contribute to the nonmonotonic behaviour of the v vs E graph. The increase in current happens due to the increase in the probability of hopping in the direction of the bias. Meanwhile, the decrease in current happens because of the increase in trapping. At higher values of p, the density of large traps decreases, decreasing the effect of trapping. So the current keeps on increasing till higher values of E until finally, the effect of trapping, even in these small number of branches, is higher than the effect of the increase in hopping.

We can also observe that the graphs shown in Figure 4.3 are smoother than those in Figure 3.2 because here we are using the exact steady state of the system. Whereas, in Chapter 3, the walkers were not at the exact steady state because the time required to reach the steady state is very large. However, in our simulations, we could not see the zero velocity phase because traps of large sizes are absent in our simulations. Although the current at a given p and E is expected to be a decreasing function of length L of the lattice, our graphs do not show any significant L dependence for L from 51 to 101, which might be due to the small lattice sizes used in our simulations.

Chapter 5

Interacting Biased Random Walkers

5.1 Definition of the Model

We use hard-core interactions between the particles to simulate *interacting biased random walkers*. Hard-core interaction means that the walkers follow an additional constraint on their motion, meaning that multiple walkers cannot simultaneously be at the same site. We can no longer study the walkers' motion using a single random walker as now the interaction between the walkers, which depends upon the walker density (ρ), also affects the motion of the walkers. We start with a specified density of walkers ρ and place the walkers randomly to a fraction ρ of the occupied sites. Let the no. of random walkers placed on the lattice be n. This system is shown in Figure 5.1

For the time evolution, we randomly select a walker. The selected walker chooses one neighbouring site randomly with probabilities $p(\text{up}) = p(\text{down}) = 1/4$ and $p(\text{left}) = 1/4 - E$, $p(\text{right}) = 1/4 + E$ where $0 \leq E \leq 1/4$, but the walker jumps to that site only if that site is occupied and no other walker is present at that site. Otherwise, it remains stationary. After each attempted move, the clock time increases by $1/n$ MCS(Monte-Carlo Steps). We say that one Monte Carlo Step is complete after attempting n moves for n randomly chosen walkers, where one walker can move more than once during one MCS. We store the displacement of each walker at each step and use this displacement to find the velocity of the walker. The results of this simulation are shown in Figure 5.2.

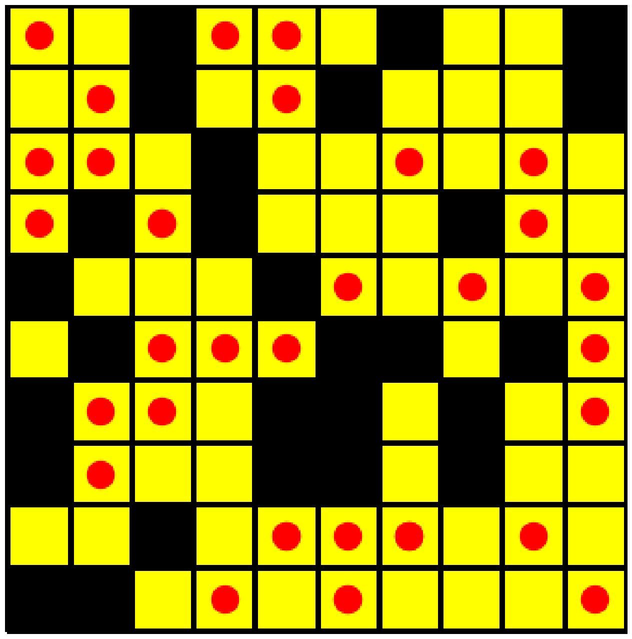
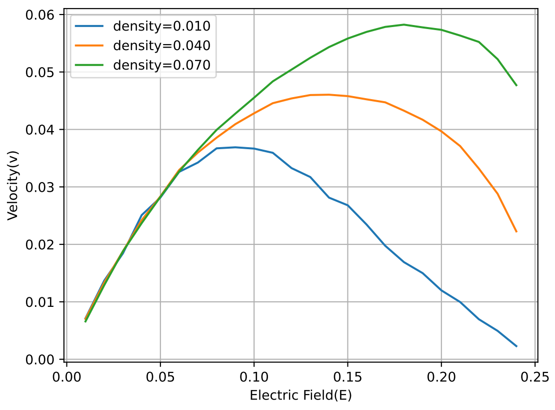


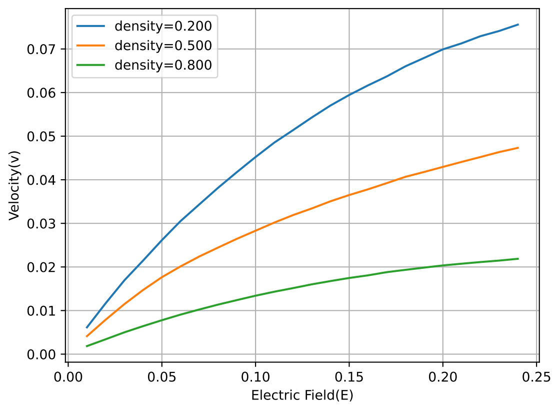
Figure 5.1: Figure showing a 10×10 slice of a large percolation cluster at $p = 0.72$ where density of walkers $\rho = 0.5$. Here, the red dots are the random walkers, and the yellow squares represent the sites belonging to the cluster. Black sites are the sites on the lattice that do not belong to the infinite cluster.

5.2 Simulation Details

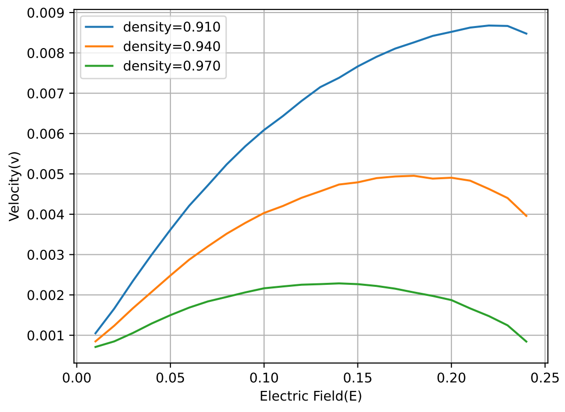
We simulate the motion of walkers on the spanning percolation cluster on a 100×100 lattice at $p = 0.72$. We measure the net displacement of all the walkers after 1000 MCS. We find the average velocity of the walkers for five walks on a cluster starting from different initial conditions on five clusters at the $p = 0.72$. We do these simulations for different values of ρ from $\rho = 0.01$ to $\rho = 0.1$ and $\rho = 0.9$ to $\rho = 0.99$ with 0.01 increment and for $\rho = 0.1$ to $\rho = 0.9$ with 0.1 increment. We do this for E values ranging from $E = 0.01$ to $E = 0.24$ with 0.01 increment. We have shown the results of our simulations for v vs E for different walker densities in Figures 5.2a,5.2b and 5.2c. We have shown the results for $v \times \rho$ vs ρ for different values of E in Figure 5.2d.



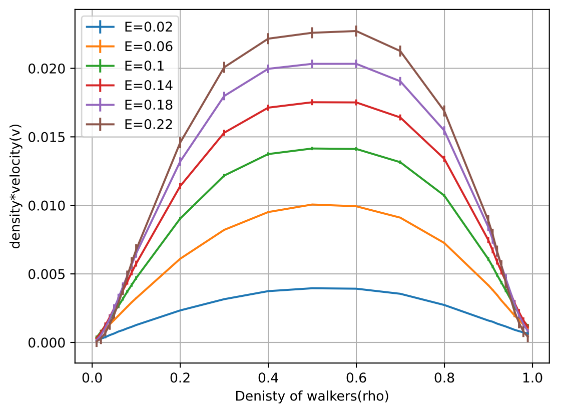
(a) v vs E at low density



(b) v vs E at intermediate density



(c) v vs E at high density



(d) $v \times \rho$ vs ρ at different E

Figure 5.2: Results of our simulation on a 100×100 lattice, averaged over five different clusters at $p = 0.72$. On each cluster, we calculate the velocity after 1000 steps averaged over five different iterations of the random walk

5.3 Results and Discussions

At low densities of walkers, the random walkers rarely interact. So, the velocity of the walkers as a function of the bias is similar to that of non-interacting walkers and shows nonmonotonic behaviour, as shown in Figure 5.2a. As we increase the density of the walkers, more and more traps get filled by the walkers, and in the steady state, some walkers remain trapped for a long time. However, most of the walkers are free to move without getting trapped due to the lowering of the effective trap length, which leads to an increase in velocity on increasing the density at low-density values, as seen in Figure 5.2a. The increase in velocity with density happens till all the dead-end branches get filled by the walkers.

After filling the dead-end branches to a certain extent such that the effective trap length now becomes lesser than the bias-induced length, the velocity increases monotonically with the increase in the biasing field, as shown in Figure 5.2b as the non-trapped walkers behave like the walkers in the nonzero current phase. However, a further increase in the density of walkers only leads to greater traffic through the backbone, decreasing the velocity of the walkers. Even though velocity remains monotonically increasing because of the already filled trapping branches, as seen in Figure 5.2b.

On increasing the density beyond $\rho = 0.9$, we get back the nonmonotonic velocity behaviour, as seen in Figure 5.2c. This non-monotonicity is because, at such high densities, the particles have already occupied all the traps in the direction of the biasing field and all the paths through which the particles flow could have been possible. So, the particles cannot move freely. The only way for a particle in the backbone to move now is when a backbone site is vacant, which happens when another particle moves into a branch opposite to the bias, creating a vacancy for the first particle to move to. As the probability that a walker moves into such a branch (in the opposite direction of the bias) decreases with the increase in bias, the velocity of walkers also decreases. This velocity keeps decreasing with increasing density, as at the limiting case of $\rho = 1$, the net current must be zero.

Alternatively, we can think of the walker moving to the right in terms of a vacancy moving to the left, as every time a walker moves to a vacant site, a vacancy moves in the opposite direction. So, the total current of the vacancy would be the same as the total current of the walkers but in the opposite direction. So, the current for the vacancy plus the walkers should be zero. As, current= density of the walkers $\times$ average velocity of the walkers:

$$\rho \times v_{walk}(\rho) + (1-\rho) \times v_{vac}(1-\rho) = 0 \tag{5.1}$$

Where, $v_{walk}(\rho)$ = mean velocity of a walker at the density of walkers ρ, and $v_{vac}(1-\rho)$ = the mean velocity of a vacancy when the density of vacant sites is $(1-\rho)$ (same as the case when the density of walkers is ρ).

A walker moving to the right with a probability of $1/4 + E$ to a vacant site leads to a vacancy moving to the left with a probability of $1/4 + E$ to a site with a walker. We can make a similar statement for the motion of the walkers and corresponding vacancies in every orientation. Adding the convention that a vacancy does not move to the site where another vacancy is already present, the evolution of vacancies is the same as that of the walkers but in the opposite direction. So, putting the condition $v_{walk}(\rho) = -v_{vac}(\rho)$ into equation 5.1 we get:

$$\rho \times v_{walk}(\rho) = (1-\rho) \times v_{walk}(1-\rho)$$

We have verified this relation by plotting the graph $v \times \rho$ vs ρ as shown in Figure 5.2d. This relation is true as the graph is symmetric and has the same values at ρ and $1-\rho$. At high ρ, $(1-\rho)$ is small and thus $v_{walk}(1-\rho)$ shows non-monotonic behaviour. At a particular value of ρ, this relation implies that $v_{walk}(\rho)$ also shows nonmonotonic behaviour at high densities.

Chapter 6

Interacting Biased Random Walkers on a Regular Comb

6.1 Definition of the model

To study the dynamical heterogeneity of interacting walkers under a bias, we study the motion of these walkers on a conceptually simpler regular comb. A *regular comb* is a lattice consisting of a backbone and side branches of constant length. The use of this lattice is motivated by the fact that it makes it easier for us to get exact results, as will be discussed in section 6.2. Also, the trapping mechanism of the particles on this lattice is the same as that of the particles in the dead ends of percolation clusters, which we are interested in understanding. A regular comb is shown in Figure 6.1. We assume periodic boundary conditions for the backbone of the comb.

We randomly place the walkers at fraction ρ of the sites on the comb. The bias is at 45 deg to the backbone, as shown in Figure 6.1. So, the walker chooses a neighbouring site randomly with $p(down, left) = p(down, right) = (1 + g)/4$, and $p(up, left) = p(up, right) = (1 - g)/4$ as shown in Figure 3.1b, where $0 \leq g \leq 1$. However, the walker goes to this site only if the site belongs to the comb and no walker is present. We use MCS time in our simulations to study the motion of the walkers.

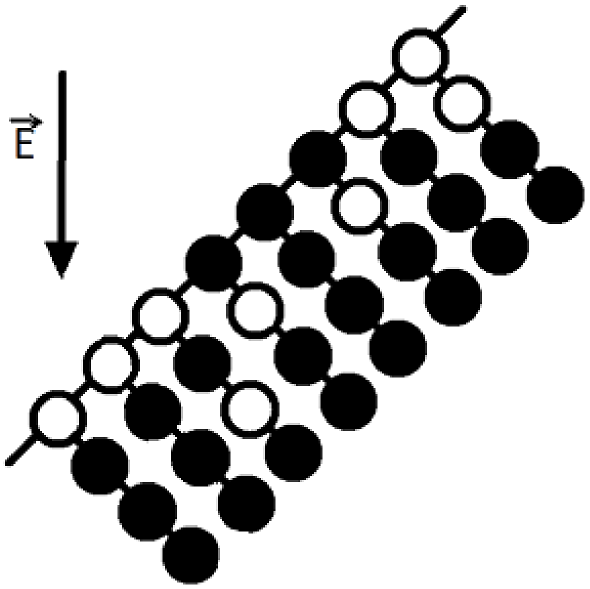


Figure 6.1: A tilted regular comb is shown with a backbone size=8 and branch length=3 on which the particles move downwards under Biasing Field $\vec{E}$ at $\rho = 0.7$.

6.2 Finding the Theoretical steady-state for an Infinite Regular Comb

Due to the periodic boundary conditions, all the branches are equivalent. Let ρ_i be the occupation probability in a branch at depth i. All the branches must have the same occupation probability at depth i because of their equivalence. So, knowing the occupation probability at the backbone ρ_0, we can find the occupation probability at any depth in the steady state using the Master's equation. We use the fact that in the steady state, the rate at which a walker goes from a site at depth i to a site at depth $i+1$ is equal to the rate at which the walker goes from a site at depth $i+1$ to a site at depth i:

$$\frac{\rho_i(1+g)(1-\rho_{i+1})}{4} = \frac{\rho_{i+1}(1-g)(1-\rho_i)}{4}$$

$$\implies Z_{i+1} = \left(\frac{1+g}{1-g}\right) Z_i \tag{6.1}$$

Where, $Z_i = \rho_i/(1+\rho_i)$ is the activity at depth i. We can find the activity Z_i at any depth from Z_0 using the recursion relation in equation 6.1, this implies:

$$Z_i = \left(\frac{1+g}{1-g}\right)^i Z_0$$

We can get ρ_i in terms of ρ_0 from this equation. To find ρ_0, we use the condition that the total occupation probability of the backbone site plus the corresponding branch sites is the same as the average number of walkers in a backbone site plus the corresponding branch sites:

$$\sum_{i=0}^{br} \rho_i = \rho(br+1) \tag{6.2}$$

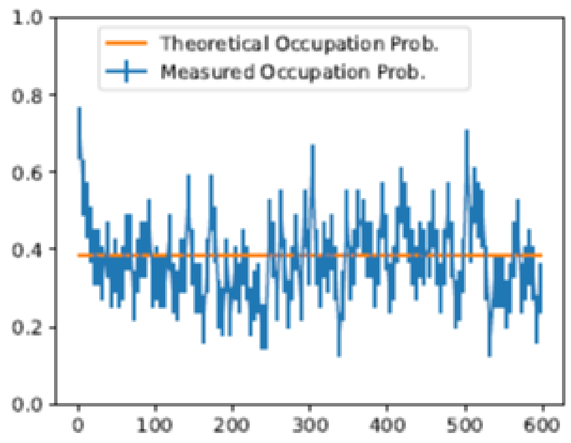
Where br = length of a branch and $(br+1)$ is the total number of sites (the backbone site plus the corresponding branch sites).

6.3 Simulation details for approaching the steady-state

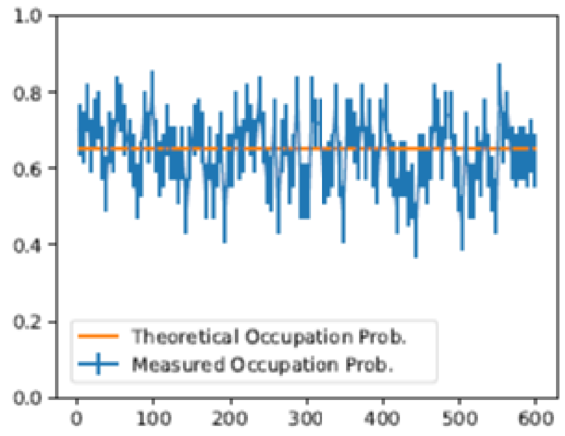
We solve equation 6.2 numerically using the command *fsolve* from the *scipy.optimize* module in Python. The solution to this gives us the steady state occupation probability at each depth. We measure the fraction of sites at a particular depth occupied at some time and average this fraction over five timesteps to find the occupation fraction in a time interval. We do this simulation on a comb with a backbone of length $bb = 10$ and a branch of length $br = 5$ with $\rho = 0.8$. We compare the occupation probabilities calculated for an infinite cluster with the occupation fraction in our simulations to see the approach to the steady state. We have shown the approach to the steady state in figure 6.2.

6.4 Discussion

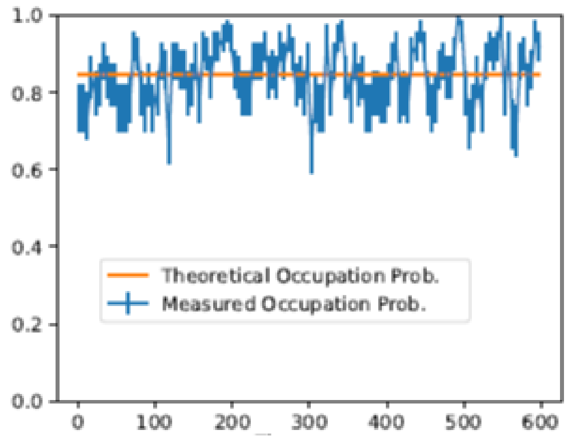
Figure 6.2 shows the variation of the occupation fraction at different depths with time. We can see that the initial occupation fractions are close to ρ at time $t = 0$ because the initial distribution of the walkers is random. The occupation fractions start changing rapidly with time and fluctuate about the steady-state value. For deeper trap depths, at depths four



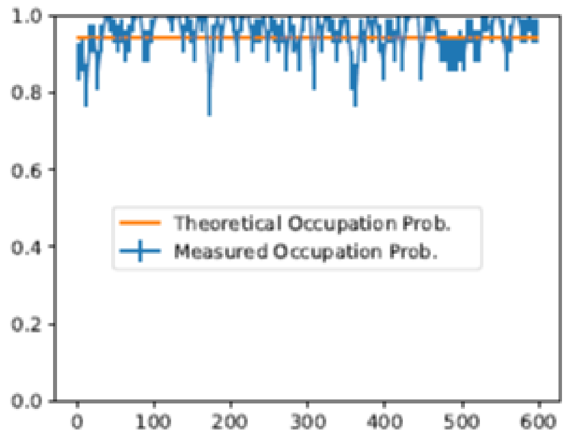
(a) ρ_0 vs t



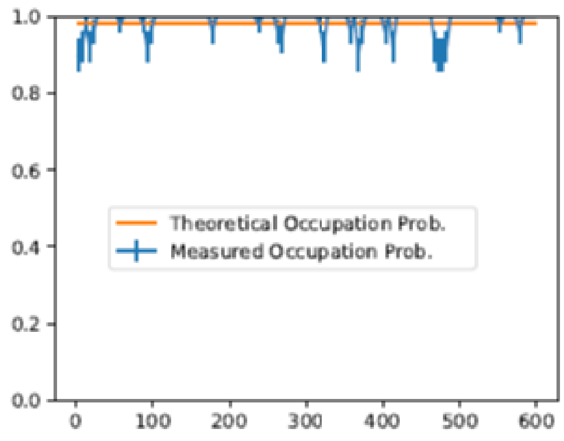
(b) ρ_1 vs t



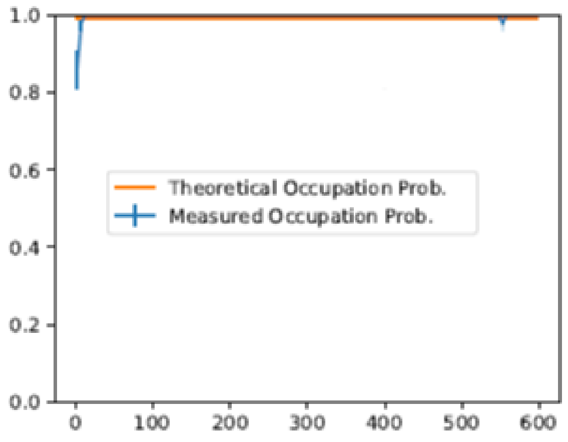
(c) ρ_2 vs t



(d) ρ_3 vs t



(e) ρ_4 vs t



(f) ρ_5 vs t

Figure 6.2: Graphs of measured occupation fraction(ρ) vs time(t) are shown in blue. The theoretical steady-state densities for an infinite regular comb are shown in yellow.

and five, we can see that the occupation fraction remains one most of the time and rarely fluctuates from this value due to the strong trapping at these depths, which makes it tough for the walker to move.

A more detailed analysis of the occupation probabilities in our simulation shows that the occupation probability tends to a different value than the one we calculated. The mistake in calculating the steady-state occupation probability is the assumption that the probability at a site at depth d depends only on the same branch's probability at depth $d-1$. This assumption works well for the limiting case of an infinite lattice. However, for a finite lattice, due to the conservation of the number of particles, the occupation probability at a particular site depends not only on its neighbours but also on the occupation probability of the non-neighbour sites. So, to find the theoretical probabilities for a finite lattice, we have to use a different method. In section 6.5, we use the partition function for calculating the occupation probabilities for a finite lattice.

6.5 Finding the Steady-state Using the Partition Function

We define a partition function that gives us the probabilities for different configurations on a regular comb for a given branch size br, a backbone size bb, and a given bias g. We define $\alpha = (1+g)/(1-g)$ as the ratio of activities at different depths. The site at depth i has more probability of being occupied than the site at depth zero by a factor α^i, the partition function for a configuration $\{z\}$ is given by:

$$\Omega(\{z\}) = (\prod_{i=0}^{br}(1+z\alpha^i))^{bb} \tag{6.3}$$

In our simulations, we have a particular no. of walkers in the comb(let N). So, the reduced partition function for N walkers is given by:

$$\Omega_N = \text{ Coefficient of } z^N \text{ in } \left(\prod_{i=0}^{br}(1+z\alpha^i)\right)^{bb} \tag{6.4}$$

Ω_N gives us the measure of the probability of N walkers on the lattice. We also use a modified Partition function:

$$\Omega(\{z\},\{b_j\}) = \left((1+b_j z\alpha^j) \prod_{i\in[0,bs]-\{j\}} (1+z\alpha^i)\right)^{bb} \tag{6.5}$$

In this partition function, the coefficient of b_j^k gives us the measure of the probability of the configuration where k sites are occupied at the depth j. So, the measure of k sites being occupied at depth j when the total no. of walkers is N is:

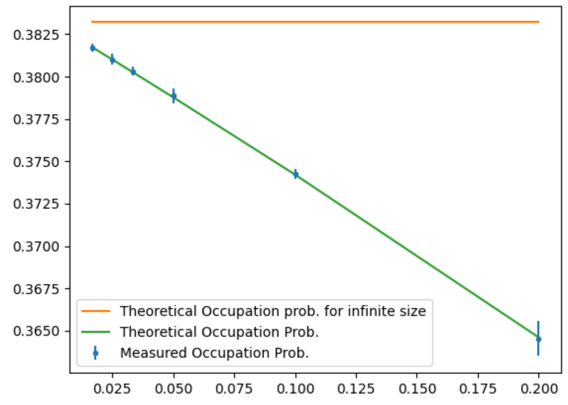
$$\Omega_{Njk} = \text{ Coefficient of } z^N b_j^k \text{ in } \left((1+b_j z\alpha^j) \prod_{i\in[0,bs]-\{j\}} (1+z\alpha^i)\right)^{bb} \tag{6.6}$$

So, the occupation probability at depth j is given by:

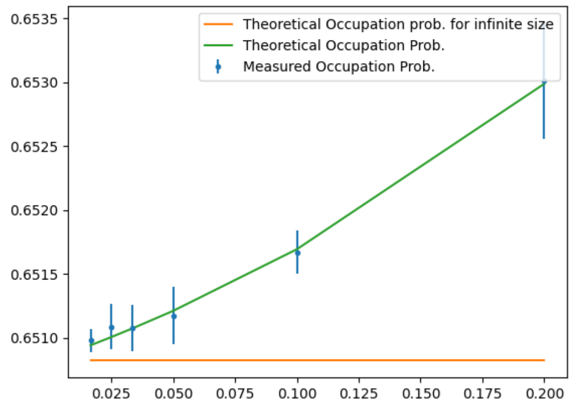
$$\rho_j = \sum_{k=1}^{bb} \left(\frac{\Omega_{Njk}}{\Omega_N}\right)\left(\frac{k}{bb}\right) \tag{6.7}$$

6.6 Simulation Details

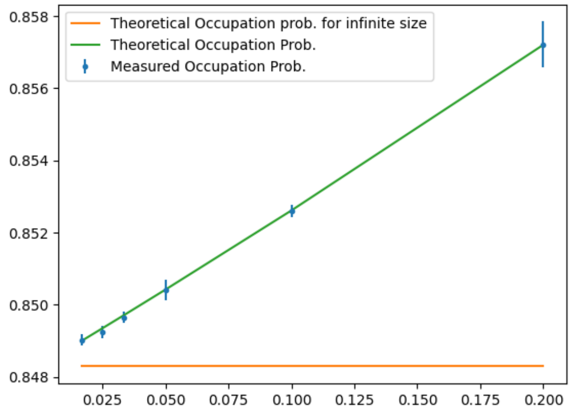
We use *Mathematica* to find the Coefficients of the different terms in the partition function and calculate the occupation probabilities for finite lattices. We find the occupation probabilities for infinite lattices by solving equation 6.2 numerically, which gives us correct results up to 8 decimal places. We measure the occupation probability in our simulations on six lattices with backbone sizes $bb = 5, 10, 20, 30, 40$ and 60 with branch size $br = 5$ at $\rho = 0.8$. We do this simulation for 10^7 MCS on lattices $bb = 5$ and $bb = 10$ and for 10^6 MCS on lattices with $bb = 20, 30, 40, 60$. After each MCS, we find the fraction of occupied sites at each depth and average it over the number of timesteps to find the mean occupation probability. We have plotted the results of our calculations and simulations for occupation probabilities at different depths vs $1/bb$ in figure 6.3.



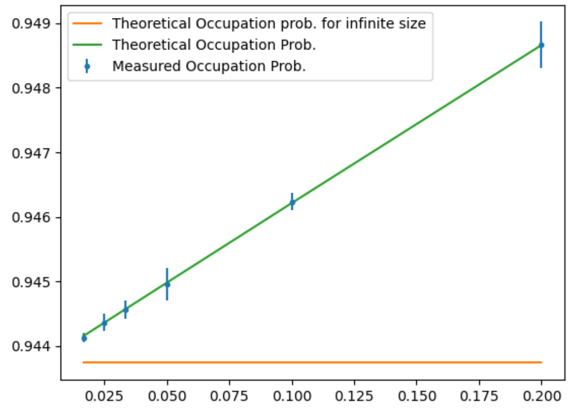
(a) ρ_0 vs $1/L$



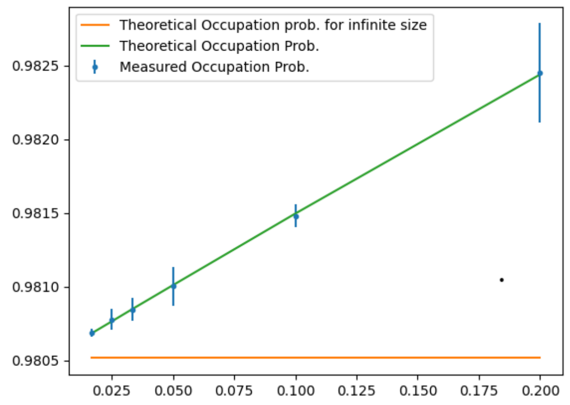
(b) ρ_1 vs $1/L$



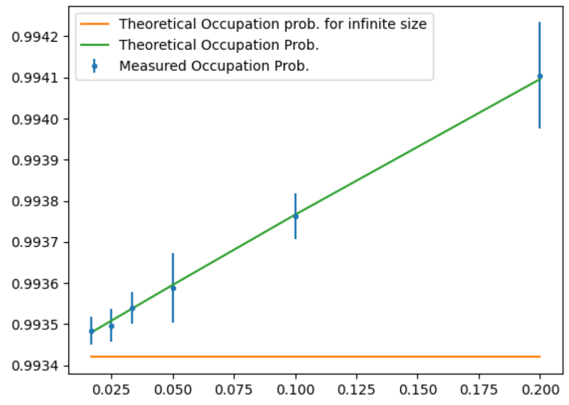
(c) ρ_2 vs $1/L$



(d) ρ_3 vs $1/L$



(e) ρ_4 vs $1/L$



(f) ρ_5 vs $1/L$

Figure 6.3: Graphs of occupation probability(ρ) vs $1/L$. Occupation probability measured using our simulations is shown in blue. The theoretical steady-state densities for an infinite regular comb are shown in yellow. The theoretical steady-state densities for a given size of regular comb found using the partition function are shown in red.

6.7 Results and Discussions

Our simulations show that the steady-state occupation probabilities depend on the backbone size; this is why the occupation probability in our simulations does not match the infinite comb occupation probability calculated in Section 6.2. We can see that the measured occupation probabilities for our simulations match the theoretical values calculated using the partition function for a given backbone size in Figure 6.3. The occupation probabilities' limiting value is the value of the occupation probability for an infinite regular comb which is reached with an increasing backbone length as $1/L$.

The occupation probability at the backbone decreases with the increase in backbone length in Figure 6.3a. Meanwhile, the occupation probability at any other depth decreases with increasing backbone length; this means that the walkers are trapped more strongly on a finite lattice than on an infinite one. The increased trapping on a finite lattice is due to finite-size effects. For instance, when a walker deep inside a branch tries to escape, the decrease in the density of the branch leads to an increase in the density everywhere else on the comb for a finite comb. However, this is not the case for an infinite comb where the change in occupation probabilities is only local.

6.8 Finding the velocity-velocity Autocorrelation Function

We find the velocity-velocity autocorrelation function using our simulations of biased interacting random walkers on the regular comb lattice with $bb = 100$ and $br = 5$ for 5×10^6 MCS. The simulation rules are tweaked slightly to make the simulation numerically faster, meaning that more walkers move at each timestep, even though the probability distribution remains unchanged. We save the displacement of each walker at each timestep. These displacements are zero for most walkers, so we must average the autocorrelation function over many timesteps to measure any significant correlation, especially for walkers trapped deep inside the branches, which is not computationally feasible. We solve this problem by finding the total displacement of all the walkers in a particular interval instead of displacement at a single timestep. We now use the average velocity of the walkers in that interval to find the

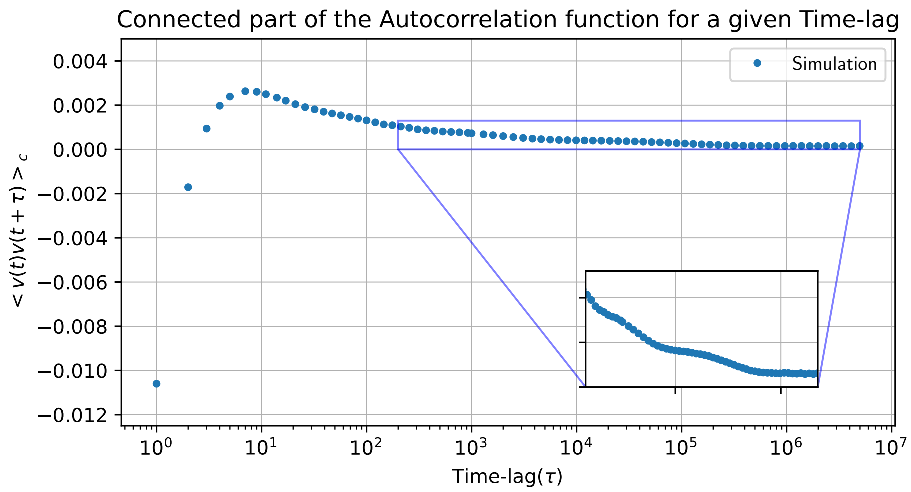


Figure 6.4: Connected part of the autocorrelation function($< v(t)v(t+\tau) >_c$) vs lag(τ) with a logarithmic x-scale. The autocorrelation function is found for a single simulation run on a regular comb with a backbone of size $bb = 100$ and branches of size $br = 5$.

autocorrelation function, as it is less likely to be zero. The length of this interval is called the *bin size*. If the bin size $<< \tau$, then the connected part of the autocorrelation function is independent of the bin size. In our simulations, we find the total displacement of the walkers for bin size 100 MCS to find the velocities, which are then used to calculate the velocity-velocity autocorrelation function. The not-normalised connected part of the autocorrelation after time τ is given by:

$$C(\tau) =< v(t)v(t+\tau) > - < v(t) >^2 \tag{6.8}$$

We measure the autocorrelation function for different τ on a regular comb with $bb = 100$, $br = 5$, at $\rho = 0.8$ for a single run of 5×10^6 MCS and average it over time and all the walkers. For $\tau < 1000$, we use the velocity at each timestep to measure the velocity-velocity autocorrelation function averaged over 5×10^5 timesteps. For higher values of $\tau > 1000$, we use the velocities in bin size 100 to calculate the autocorrelation function over all the available timesteps (which means the 5×10^4 available timesteps for bin size 100). The graph showing the connected part of the autocorrelation function is shown in figure 6.4 along with a zoomed-in inset at large times.

6.9 Results and Discussions

In figure 6.4, we can see that the connected part of the velocity-velocity autocorrelation function is negative at small values of τ. This happens because the largest contribution to the velocity-velocity autocorrelation at short times is due to the motion of the walkers in the backbone, which is anti-correlated at small τ due to crowding in the backbone, giving negative autocorrelation. At considerable time lags, the dominant part in the velocity-velocity autocorrelation is due to the trapped walkers. As the fraction of the trapped walkers decreases on increasing τ, the autocorrelation function also decreases and shows bumps corresponding to the trapping at different depths. These bumps are visible in the zoomed-in inset in figure 6.4. These bumps are present because the velocities of particles at depth i are no longer correlated after the trapping time at depth i (τ_i) as these particles are no longer trapped with zero velocity. So, at $\tau > \tau_i$, the autocorrelation is only due to the particles at depths $> i+1$, which leads to bumps at these time scales. Some walkers remain trapped for a long time, so the connected part of the autocorrelation function remains significant even at large times. It is about one-tenth of its maximum value even at 5×10^6 MCS.

6.10 Trapping Time Distribution

Using our simulations, we find the probability that the trapping time of a walker is t ($P(t)$) by finding the time a walker takes to exit the branch it enters. We save the maximum depth the walker reaches along with this time, as the trapping time is a function of the maximum depth reached. Let the probability that a randomly selected walker has been in the branch for time t be $\tilde{P}(t)$. As each walker with trapping time t stays in the branch for t steps, we have:

$$\tilde{P}(t) = tP(t)$$

Then, the probability that a randomly selected walker has been in a branch for a time greater than τ is given by:

$$P(>\tau) = \int_{\tau}^{\infty} \tilde{P}(t)\, dt$$

We can also find the probability that a randomly selected walker has been in the branch for a time greater than τ, and the maximum depth it reached is d. In our simulation, we used

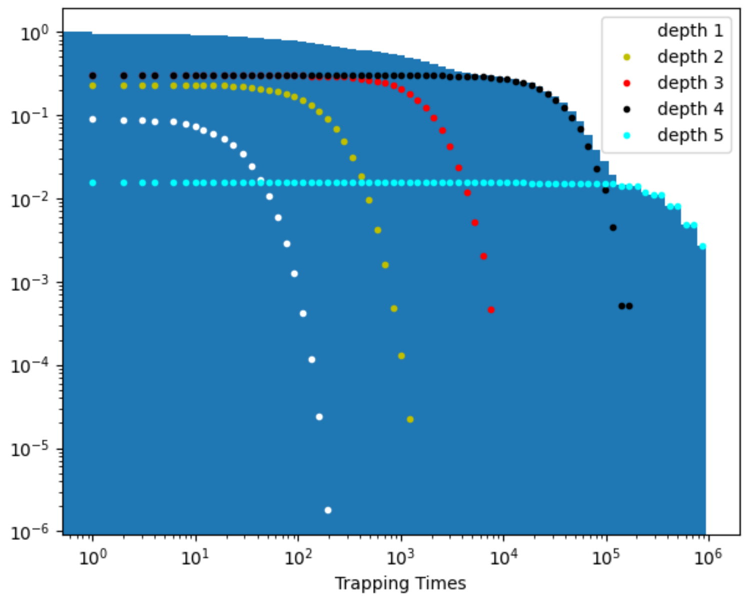


Figure 6.5: Probability that the trapping time is greater than τ vs τ. The graph for all the trapping times is shown in blue as the histogram. The points give the probability that the trapping time is greater than τ given the maximum depth the walker reaches. Axes are plotted on the log scale.

interacting biased random walkers moving on a comb with backbone length 100 and branch length 5 at $\rho = 0.7$ density of walkers for 10^6 timesteps. We used the histogram data to find $P(t)$ for logarithmic bins. We find $P(> \tau)$ by numerically integrating the histogram for $tP(t)$. Using logarithmic bins, we plot the histogram for $P(> \tau)$ on logarithmic axes. We also plotted the $P(> \tau)$ graphs given the maximum depth reached for different depths on the same graph.

6.11 Results and Discussions

Figure 6.5 shows the probability that the trapping time is greater than τ vs τ. This figure also shows the probabilities that the trapping time is greater than τ given different maximum depths reached. We can see that the net histogram shows step-like behaviour when the contribution to the trapping times becomes significant for a new depth. Namely, all the contributions for high trapping times are due to the walkers reaching a maximum depth of 5. On decreasing τ, the contribution to the trapping times for a walker that reaches a maximum depth 4 becomes significant, and then we can see a step-like behaviour in the net histogram. The same happens when the contribution to the trapping times for a walker that reaches a maximum depth of 3 becomes significant, although this step is less prominent. These results match the results obtained by C. Iyer, M. Barma and D. Dhar using a different method[8].

Chapter 7

Conclusion

In our work, we first simulated noninteracting and interacting biased random walkers on a percolation cluster. We were not able to find the zero velocity limit for these walkers. It is not easy to find the zero velocity limit for the noninteracting walkers numerically, which has been a topic of debate in the literature for a long time. However, we could observe the nonmonotonic behaviour of the noninteracting walkers using our simulations and the steady state distribution on a finite percolation cluster. We also saw the nonmonotonic v vs E graph become monotonic when the density of walkers was increased beyond a certain value for interacting walkers and how it again became nonmonotonic at very high densities of walkers.

We used the Hoshen-Kopelman and DFS algorithms to find the different clusters and backbones of clusters, respectively. Using these algorithms on a percolation cluster with periodic boundary conditions is non-trivial, and it took a lot of trial and error. We saw the approach to the steady-state distribution and the steady-state distribution's dependence on the lattice's length for the simulations of interacting biased random walkers on a regular comb. We could also find the exact steady state for a comb of finite size and match it with our measured simulation data. Using numerical simulations, we confirmed the findings of C. Iyer, M. Barma and D. Dhar[8] for the trapping time distribution. We also found the relation between the trapping time distribution and the maximum depth reached by a walker. We also found the connected part of the velocity-velocity autocorrelation function. This graph shows that the autocorrelation in the walker velocities is significant even at large times. This graph also shows bumps corresponding to particles being trapped at different depths.

Further work can be done based on the work in my thesis. Understanding the trapping time distribution and the velocity-velocity autocorrelation function on a percolation cluster is an interesting question; it was the motivation for much of my thesis work, but it is still not understood well. There are some arguments based on the distribution of side branches on a random comb, which imply that the Probability that the trapping time is greater than T_w is $\sim \exp\left(-c\sqrt{\log(T_w)}\right)$ on a percolation cluster [8], but these results need to be studied more rigorously. Further work can be done to understand the results on the percolation cluster based on the results found in this thesis for a single branch.

Bibliography

[1] Karl Pearson. "The problem of the random walk". In: *Nature* 72.1865 (1905), pp. 294–294.

[2] Simon R Broadbent and John M Hammersley. "Percolation processes: I. Crystals and mazes". In: *Mathematical proceedings of the Cambridge philosophical society*. Vol. 53. 3. Cambridge University Press. 1957, pp. 629–641.

[3] Pierre Gilles de Gennes et al. "La percolation: un concept unificateur". In: *La recherche* 7.72 (1976), pp. 919–927.

[4] Mustansir Barma and Deepak Dhar. "Directed diffusion in a percolation network". In: *Journal of Physics C: Solid State Physics* 16.8 (1983), p. 1451.

[5] Frank Spitzer. "Interaction of Markov processes". In: *Advances in Mathematics* 5.2 (1970), pp. 246–290. ISSN: 0001-8708. DOI: https://doi.org/10.1016/0001-8708(70)90034-4. URL: https://www.sciencedirect.com/science/article/pii/0001870870900344.

[6] Kirone Mallick. "The exclusion process: A paradigm for non-equilibrium behaviour". In: *Physica A: Statistical Mechanics and its Applications* 418 (2015), pp. 17–48.

[7] Ramakrishna Ramaswamy and Mustansir Barma. "Transport in random networks in a field: interacting particles". In: *Journal of Physics A: Mathematical and General* 20.10 (1987), p. 2973.

[8] Chandrashekar Iyer, Mustansir Barma, and Deepak Dhar. "Asymmetric simple exclusion process on the percolation cluster: Waiting time distribution in side-branches". In: *arXiv preprint arXiv:2312.15508* (2023).

[9] Deepak Dhar and Dietrich Stauffer. "Drift and trapping in biased diffusion on disordered lattices". In: *International Journal of Modern Physics C* 9.02 (1998), pp. 349–355.

[10] Dietrich Stauffer and Ammon Aharony. *Introduction to percolation theory.* CRC press, 2018.

[11] Joseph Hoshen and Raoul Kopelman. “Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm”. In: *Physical Review B* 14.8 (1976), p. 3438.

[12] Thomas H Cormen et al. *Introduction to algorithms.* MIT press, 2022.

[13] Iwan Jensen. “Low-density series expansions for directed percolation: I. A new efficient algorithm with applications to the square lattice”. In: *Journal of Physics A: Mathematical and General* 32.28 (1999), p. 5233.

[14] Steven R White and Mustansir Barma. “Field-induced drift and trapping in percolation networks”. In: *Journal of Physics A: Mathematical and General* 17.15 (1984), p. 2995.

[15] William Ford. *Numerical linear algebra with applications: Using MATLAB.* Academic Press, 2014.