Modeling Molecular Traffic: Theory, Simulations, and Potential Applications

Emma C. Aldrich

A thesis presented in partial fulfillment of
the requirements for the degree of
Bachelor of Science

Department of Physics and Engineering
Washington and Lee University
Lexington, VA, USA
April 13, 2022

Abstract

I present a traffic model inspired by the motion of molecular motors along microtubules, represented by particles moving along a one-dimensional track of variable length. As the particles move unidirectionally along the track, several processes can occur: additional particles can attach at an unoccupied site, particles already on the track can move to the next open site, or particles on the track can detach. I study the model using mean-field theory and Monte Carlo simulations, with a focus on the steady-state properties and the time evolution of the particle density and particle currents. I then expand the model to include two-dimensional side-stepping along a cylindrical microtubule, as well as bidirectional movement of two different species of motors along tracks of fixed and variable length. Though each model expansion adds complexity, I keep the model general to allow for versatile applications throughout non-equilibrium statistical physics and biology.

Acknowledgements

I would like to extend my sincerest gratitute to my thesis advisor, Dr. Irina Mazilu. This project began back in 2019 without any idea that it would transform into a three-year project, including survival through a pandemic summer and collaborations across seven hours of time zone differences. Dr. Mazilu's unwavering support not only helped me grow as a researcher and person, but inspired me to continue my career in research. Not enough can be said to her influence as mentor. I cannot wait to come back from CU Boulder, Ph.D. in hand, and tell the next generation of Washington and Lee physics majors how much she helped me back in the day.

I would also like to thank the rest of my thesis committee, Dr. Dan Mazilu (Washington and Lee University) and Dr. Laurențiu Stoleriu (Cuza University, Iasi, Romania) for their invaluable mentorship and expertise. Dr. Stoleriu contibuted significantly to the Monte Carlo simulations, and this project would not have gotten off the ground without his help. Similarly, Dr. Mazilu's eye for formatting and attention to detail has greatly improved the quality of this thesis.

The Washington and Lee University Physics and Engineering Department has been an instrumental part of the success of this project. The community of faculty and students is second to none, and I will carry its values with me for years to come. Specifically, I would like to thank the Class of 2022 physics majors for their invaluable help in and out of class, and Beth Reed '22 for her help in the first year of this project. They have made Howe Hall a home that will be hard to leave.

Contents

Chapter 1

Introduction and Background

1.1 Applications of Non-Equilibrium Systems

Traffic models have been of interest to physicists and mathematicians for a very long time for their versatile interdisciplinary applications. For biophysicists, it is interesting to study how traffic jams at the intracellular level may result in severe disruptions in the well-functioning of the human body. Stochastic processes driving traffic jam formation appear in many physical systems and have been thoroughly studied in physics, chemistry, biology and social sciences using a variety of stochastic models. One category of models that lead to useful quantitative results for different traffic-type models is the *totally asymmetric exclusion process* (TASEP), studied by itself to isolate edge effects in smaller systems [1] or coupled with Langmuir kinetics to model bulk dynamics [2, 3]. Most traffic models assume tracks with fixed length over time, but there are processes in nature for which these tracks have variable length. The most common example in biology is the traffic of molecular motors [4, 5] on cellular tracks called *microtubules* [6, 7].

Interest in molecular motor dynamics increased greatly with the realization that the proper functioning of cells depends on the active, directed transport of macromolecules at the intracellular level. Analogous to vehicles, molecular motors experience traffic jams on cytoskeletal tracks, observable as a rapid increase in density and decrease in speed, leading to a blockage that prevents additional motor proteins from traveling. From the perspective of a physicist, the traffic of molecular motors represents an interesting non-equilibrium system directly amenable to analysis using methods of non-equilibrium statistical physics. In the past twenty years, the non-equilibrium physics community demonstrated an increased interest in modeling molecular motors using the driven lattice gas approach, with many studies using the TASEP model as a paradigm [8, 9, 10]. These models were studied using both computer simulation techniques and analytical means [11, 12]. A special interest was shown in describing the movements of molecular motors on cytoskeletal filaments as random walks on a track [13].

Not only are the motors by themselves intricate biological machines, but also the system increases in complexity through dynamic interactions between motors and their cellular tracks. Before proceeding into an explanation of the stochastic model, it is necessary that I provide a brief background on microtubule dynamics, motor proteins, and the biochemistry that relates them. I will then explain how the biology applies to the TASEP model.

1.2 Biological Background

1.2.1 Microtubule Dynamics and Motor Molecules

In eukaryotic cells, microtubules are linear, polarized polymers that contribute to intracellular transport, cell motility, division and differentiation [14, 15]. Microtubules are formed from the polymerization of alpha-beta tubulin dimers (Fig. 1), which combine linearly to form a *protofilament*. An average of thirteen protofilaments interact longitudinally to form a cylindrical shape characteristic of microtubules [15].

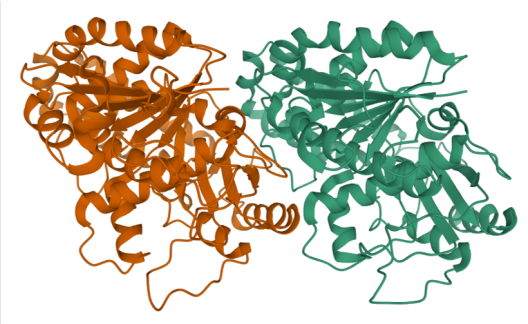


Figure 1.1: Electron diffraction of alpha-beta tubulin dimer. Alpha tubulin depicted in orange, beta tubulin depicted in green [16].

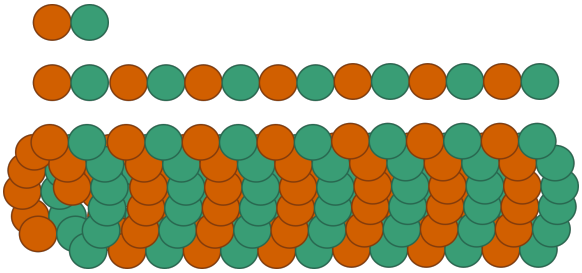


Figure 1.2: Basic structure of a microtubule. Alpha tubulin (orange) and beta tubulin (green) dimerize (top) to form the base subunit of a protofilament (middle). Protofilaments interact to form a cylindrical microtubule (bottom).

The polarity of a microtubule arises from the consistent alignment of dimers, with the beta-tubulin forming the more dynamic positively charged end [17]. The dynamic end experiences more rapid attachment of guanosine triphosphate (GTP), forming a GTP cap that regulates the polymerization of additional dimers [18]. Conversely, GTP hydrolysis weakens the adjacent protofilaments' affinity for new tubulin dimers, thus increasing the rate of depolymerization and shrinking the microtubule [19]. Lateral interactions between protofilament tips coupled with variable stability from GTP caps causes each protofilament to polymerize at slightly different rates [20, 21]. The consequences of these variable polymerization rates are discussed in further in Section 3.2. Frequent

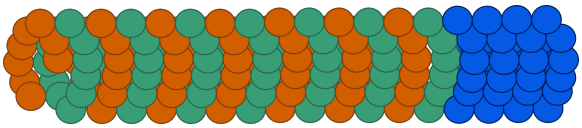


Figure 1.3: Structure of microtubule in growth phase. As above, orange circles are alpha-tubulin and green circles are beta-tubulin. Blue circles represent GTP molecules, creating a cap that provides microtubule stability and permits growth. Addition of tubulin dimers occurs on the positive end (right side).

transitions between polymerization and depolymerization, with periodic growth and shrinkage, are referred to as *dynamic instability* [22, 23, 24, 25]. At an extreme state of instability, the removal of the stabilizing GTP cap results in rapid dissociation known as *microtubule catastrophe* [18, 19, 22, 23]. Understanding microtubule dynamics could help explain the role of malfunction-

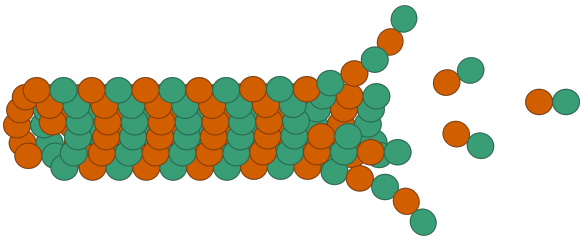


Figure 1.4: Microtubule shrinking rapidly, as in catastrophe.

ing microtubules in the development of diseases, providing new potential treatments for cancer or neurodegenerative conditions [26, 27, 28].

Aside from their role in cell division, I draw attention to the contributions of microtubules in the transport of organelles, protein complexes, and other macromolecules across a cell, made possible through interactions between microtubules and motor molecules [6, 29, 30]. Motor molecules, like kinesin, dynein and myosin, utilize adenosine triphosphate (ATP) hydrolysis to "walk" in a hand-over-hand fashion along actin filaments or microtubule tracks, carrying with them cargo vesicles [31]. Biological data have now reached a stage of quantitative and reproducible detail, which allows for theoretical modeling of motor behavior, including step size, frequency, and direction that aligns with experimental evidence [28, 32, 33]. Interest in molecular motor dynamics increased greatly with the realization that the behavior of the cell and its structure depends on the active, directed transport of macromolecules at the intracellular level, including, but not limited to, the delivery messenger RNAs during RNA localization, axonal transport of proteins and membranes to synaptic terminals, or the dispersion of large membrane organelles [6, 25, 30, 34].

Just as a disruption of traffic hurts the functioning of a city, ineffective molecular transport can result in a variety of diseases. For example, defective motor molecules reduce efficiency of axonal transport, leading to the neurodegenerative disease amyotrophic lateral sclerosis [35]. Analogous to a traffic jam of vehicles, motor molecules experience traffic jams on cytoskeletal tracks, observable as a rapid increase in density and decrease in speed, forming a blockage that prevents additional motor proteins from traveling [27, 36]. These traffic jams of motor molecules are thought to be

either density-induced, in which the density of motor molecules exceeds a threshold, or bottleneck-induced, in which low detachment rates at microtubule ends cause pileups [36]. Processivity of motor molecules as well as conditions that affect attachment and detachment rates, such as temperature or ATP concentration, contribute to the frequency of traffic jams. Research supports that motor proteins may have adapted to reduce jams [36]. Consequently, modeling density and current of motor molecules on a dynamic microtubule is of interest to biologists and physicists alike and could help shed light on the development of traffic jams in neurodegenerative patients.

Microtubule stability, interactions with motor molecules, and subsequent intramolecular transport are heavily influenced by microtubule associated proteins (MAPs). Table 1.1 summarizes MAPs with critical functions in human health.

Name	Functions							
	Neural development	Microtubule stabilization in axons	Microtubule stabilization in dendrites	Signal transduction	Axonal transport	Brain enlargement	Cortical neuron migration	Microtubule growth regulation
MAP1A	■	■	■					
MAP1B	■	■	■					
MAP2A			■	■				
MAP2B			■	■				
MAP2C	■							
MAP- Tau	■	■			■			
ASPM						■		
DCX	■						■	
LIS1	■						■	
CLIP-115								■

Table 1.1: Functions of critical MAPs. Functions that each MAP exhibits are highlighted in black. ASPM: abnormal spindle-like protein, microencephaly-associated. DCX: doublecortin. LIS1: lissencephaly-1. CLIP-115: cytoplasmic linker protein-115. Adapted from Craddock 2011 [37]

Though MAPs are not a focus of this model, they are an emerging topic of biological study. A future direction of this project may be to introduce the effect of variable concentrations of MAPs on dynamic instability and neurodegenerative disease.

Further still, interactions between motor molecules and microtubules change not only the microtubule's affinity for the attachment of more motor molecules, but also the polymerization rates associated with dynamic instability [29, 38]. There is evidence that when kinesin-1, a member of the kinesin superfamily, is in its strong binding state, it changes the structure of its microtubule track to inhibit shrinking and increase stability [38]. Additionally, attractive interactions between kinesin-1 neighbors influences dynamic instability. The binding of kinesin-1 has been demonstrated to change a microtubule lattice into a high-affinity state that promotes additional binding of kinesin [29]. These attractions between kinesin-1 motors increase their duration of attachment to the microtubule, resulting in clusters [39]. Clustering at the end of a microtubule then alters dynamic instability by decreasing the rate of depolymerization [23, 57].

This model focuses on the movement of kinesin along a dynamic microtubule. Kinesins are a superfamily of proteins which step hand-over-hand toward the positive end of a microtubule. The next section will describe the kinesin stepping mechanism along cytoskeletal filaments, as driven by hydrolysis of ATP.

1.2.2 Biochemistry of Stepping Mechanism: Kinesin Superfamily of Proteins

Before further investigating motor molecule dynamics, it is important to understand the biochemistry underlying the kinesin stepping mechanisms. The kinesin superfamily proteins (KIFs) fulfill a variety of roles in the cell, including intracellular transport, morphogenesis (cell development), and basic functioning, and more recent research reveals their roles in cognitive functions like memory and learning, asymmetrical developmental processes, and tumorigenesis suppression [40].

The speed of ATP hydrolysis determines the speed of kinesin translocation. KIFs are divided into subfamilies, whose variations in structure alter the efficiency of the ATP cycle. Though there is 50% sequence identity conserved betweeen KIFs, high diversity between subfamilies results in a variety of properties and subsequent functions. As a useful reference, I provide a table of kinesin subfamilies and associated properties in Table 1.2.

Subfamily	Translocase activity		Processivity	Sliding	Regulation of microtubule dynamics				Example
					Depolymerization		Polymerization		
	+	-			+	-	+	-	
Kinesin-1	■		■						Kif5
Kinesin-2	■		■						Fla10
Kinesin-3	■								Kif1
Kinesin-4	■		■	■	■				Kif4
Kinesin-5	■			■	■				Eg5
Kinesin-7	■		■				■	■	CENP-E
Kinesin-8	■		■		■				Kip3
Kinesin-10							■		NOD
Kinesin-13					■	■			MCAK
Kinesin-14		■		■		■			NCD

Table 1.2: Description of KIF subfamilies and associated properties. Properties that kinesins demonstrate are highlighted in blue [41].

All kinesins express a highly conserved motor domain (Fig. 1.5), serving as a switch triggered by the bound nucleotide: ATP, ADP in complex with inorganic phosphate, ADP, or no nucleotide bound (empty). Each nucleotide conformation regulates binding affinity to the microtubule and transitions from low to high affinity. In a chemically and mechanically coupled cycle, the binding of a nucleotide triggers its hydrolysis as the binding of the kinesin motor domain causes a change in motor domain kinetics, reciprocally triggering a conformational change. In its simplest form, this process is known throughout molecular biology as the *ATP turnover cycle* (Fig. 1.6).

Differences in structures of subfamilies within KIFs change the dynamics of the cycle. Kinesin motor domains (or heads) are attached to a neck linker and neck, connected to a stalk and tail, which can carry molecular cargo. Variations in neck effects between subfamilies alter processivity, how many steps a motor takes before dissociating from the microtubule. Further, some proteins have longer necks that increase probability for side stepping in the two dimensional model, explained in Chapter 3.

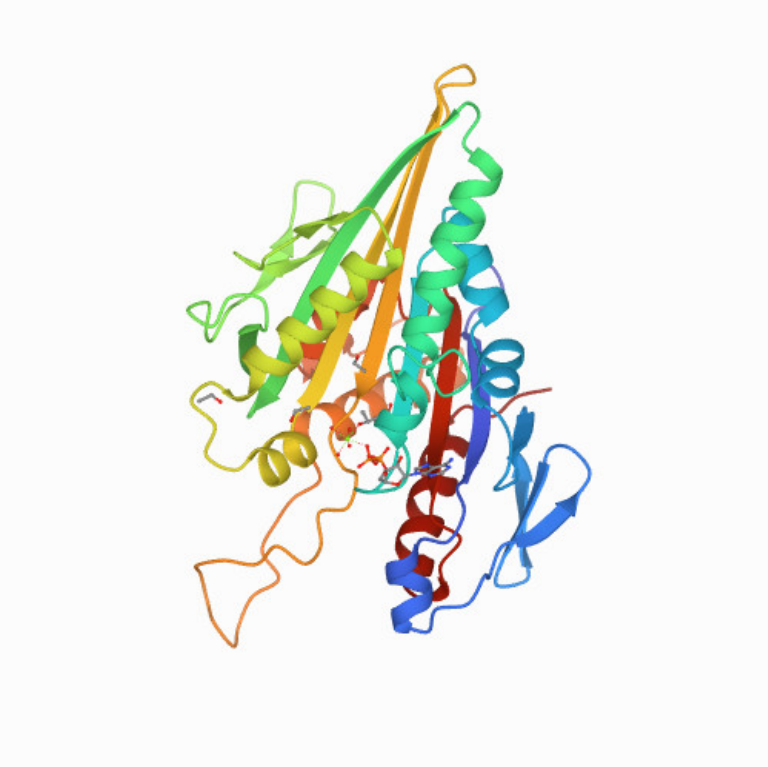


Figure 1.5: Highly conserved KIFs motor domain [42].

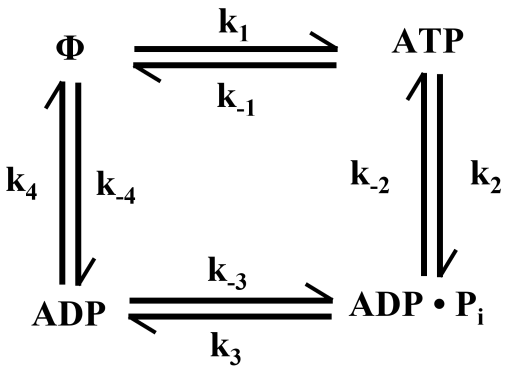


Figure 1.6: Simplest version of the ATP turnover cycle. There are four nucleotide states: ATP, ADP in complex with inorganic phosphate, P_i, ADP, and empty, Φ. Transition rates are represented by k_{on} values with positive subscripts and k_{off} values with negative subscripts.

1.2.3 A note on ATP, Equilibrium, and Phase Transitions

At chemical equilibrium, the ratios of local binding and detachment rates remain fixed, resulting in no net movement of a motor molecule. In the cell, however, the ATP hydrolysis reaction is maintained out of equilibrium. The net motion of the molecular motors along the filaments happens above a critical concentration of ATP, for which the rate of stimulated detachment is

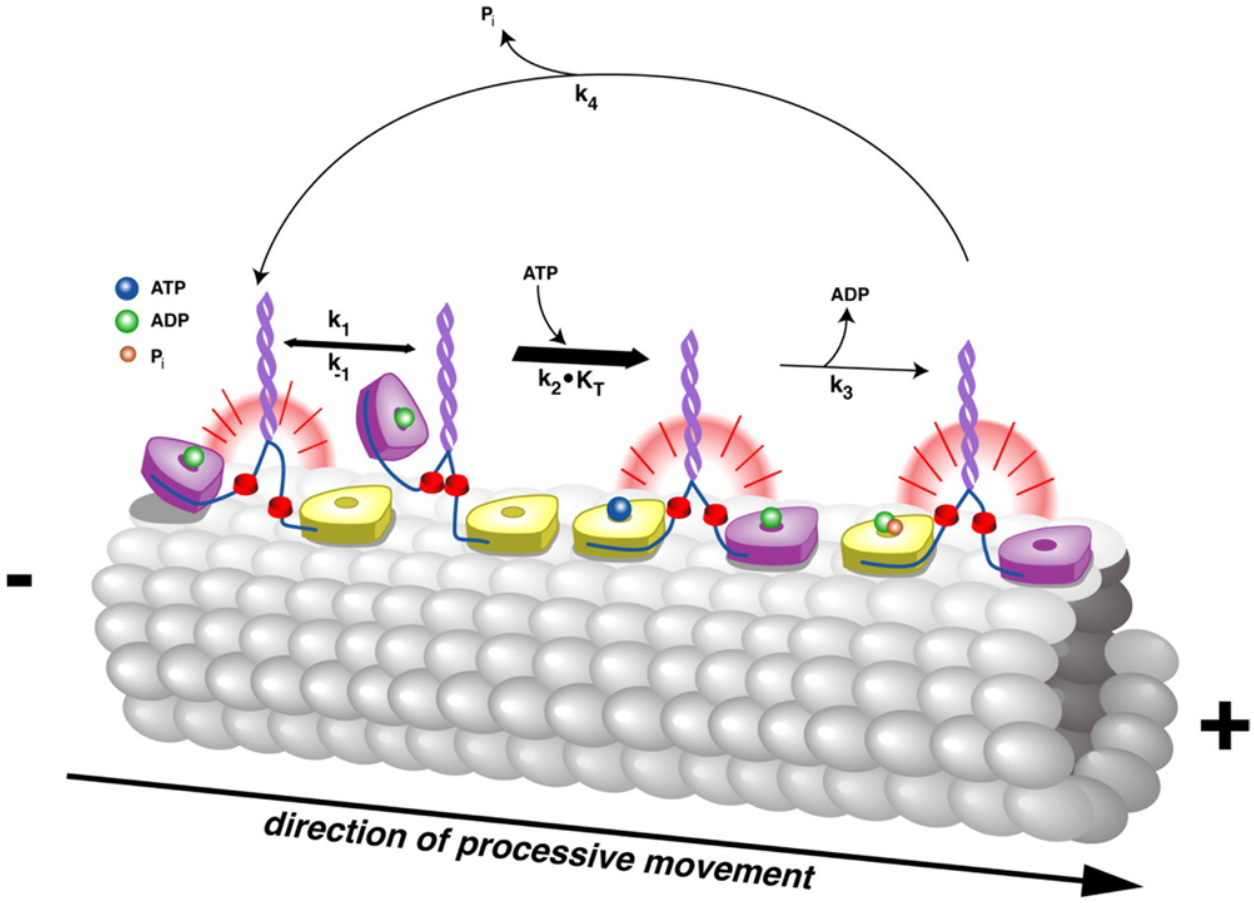


Figure 1.7: Kinesin stepping mechanism using the ATP turnover cycle. Kinesin heads are depicted in purple and yellow. The purple coil represents the kinesin stalk and blue lines represent the neck linkers. Red discs represent tunnel magnetoresistance probes, a tool for measurement used by Toprak *et al.* 2009 and not a natural part of kinesin structure [43]. In their experiment, Toprak *et al.* measured changes in kinesin confirmation as it moved along the microtubule (gray) by rays emitted from the probes when they the neck linkers for each head were associated (red glow). ATP is represented by the blue sphere, ADP is represented by the green sphere, and inorganic phosphate is represented by the orange sphere. Steps 1, 2, and 3 of the ATP turnover cycle help the kinesin move toward the positive end of the microtubule. Step four returns kinesin to its original confirmation, ready to take another step. Image created by Ahmet Yildiz [43].

.

sufficiently high. This situation is analogous to phase transitions in condensed matter physics [3]. In para– to ferromagnetic transitions, for example, cooperative behavior can align spins, even in the absence of external magnetic fields [45]. Indeed, the general mathematical properties at the critical point are closely related in both systems. However, there are significant differences: the team of motors is a non-equilibrium system that is controlled by chemical kinetics, as opposed to an equilibrium system that is controlled by temperature [45].

1.3 TASEP models with Langmuir Dynamics

In the one-dimensional non-equilibrium system, movement of particles is driven by two stochastic processes. At the boundaries, entrance and exit of particles is driven by the totally asymmetric exclusion process (TASEP). In the bulk resevoir of the one-dimensional lattice, the attachment and detachment of particles is driven by Langmuir dynamics (also referred to in literature as Langmuir kinetics).

In its most basic terms, TASEP is a stochastic model beginning with a one-dimensional lattice of integer N sites. Each site can can be either occupied or unoccupied by a particle, modeled by a 1 or 0 [44]. Particles enter and exit the lattice exclusively on the boundaries, entering the lattice at rate α and exiting at the opposite boundary at rate β. To move along the lattice, a particle at site n hops right to the next unoccupied site, $n+1$ with probability p. If the site adjacent to the particle is occupied, the particle becomes immobilized due to exclusion [45]. In the *totally* asymmetric exclusion process, particle motion is constrained to one direction only, such that the probability of a particle moving to the left, q, is 0. The asymmetric condition, by comparison, results in two-dimensional motion and particle drift, such that $q > p$ results in drift to the left and $q < p$ results in drift to the right.

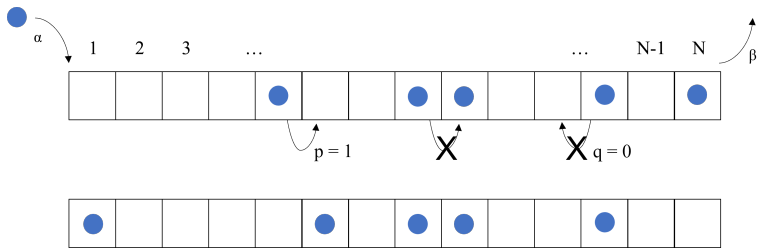


Figure 1.8: Schematic of TASEP on a one-dimensional lattice. Particles are introduced to the system at rate α and exit at rate β . Particles transition to the next unoccupied site at rate $p = 1$. For TASEP, particles can only hop in one direction, such that $q = 0$. In this example, one of the particles in the bulk of the lattice is held immobile by the particle occupying the adjacent site.

Classical TASEP displays a complex phase diagram for the total particle current that has three phases: low-density, high-density, and a maximal current. These results are well established and widely applicable throughout non-equilibrium physics research and are explained for reference in Appendix A.

In comparison to TASEP where attachment only occurs at boundaries, Langmuir dynamics allows for stochastic attachment and detachment in the bulk of the one-dimensional lattice at rates ω_A and ω_D, representing adsorption and desorption, respectively. By microscopic reversibility, the kinetic rates of attachment and detachment will cause the system to evolve into equilibrium [3].

For systems in the thermodynamic limit, with both TASEP and Langmuir dynamics, the latter dominates particle behavior. As a result, statistical physics requires that Langmuir-driven systems evolve into a steady state. By contrast, TASEP driven systems evolve into stationary, non-equilibrium states [3]. The limit case occurs in large, yet finite systems in which TASEP and Langmuir kinetics compete. The study of such systems resuls in the observation of particle behavior

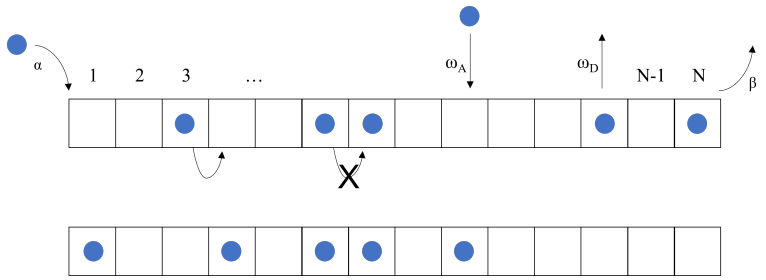


Figure 1.9: Schematic visualization of TASEP and Langmuir kinetics along a one-dimensional lattice. ω_A represents bulk attachment and ω_D represents bulk detachment.

distinct from either paradigm, resulting in phase separation [3]. While phase diagrams related to the movement of motor molecules along a microtubule lattice are beyond the scope of this thesis, they are a potential future expansion of the stochastic unidirectional model.

1.3.1 TASEP Applications to Biology

TASEP is a highly versatile model with many biological applications. An early application of TASEP was in modeling translation, the process in which transfer RNAs read messenger RNA transcripts and convert sequences of three-base codons into an amino acid chain [44, 46, 47, 48]. The mRNA serves as a one-dimensional lattice, and ribosomes, complexes of ribosomal RNA that provide a site for translation, move unidirectionally 5' to 3' along the lattice [44]. Each site on the lattice is represented by a codon. A ribosome attaches to one site at a time, n, but blocks sites $n-1$ and $n+1$ from attachment by another ribosome. In a reading frame of N codons, a ribosome begins at the start codon and elongates a polypeptide as it moves toward the 3' end, adding a new amino acid at each codon (that is, at each site of the lattice). The addition of the last amino acid in the polypeptide occurs at site $N-1$, as the N^th site is designated by the stop codon, which does not code for an amino acid. TASEP can be used to model two scenarios, including a uniform-density state, where $\rho(\mathrm{t}) = \rho$, as well as a steady state, where $\mathrm{d}\rho(\mathrm{t})/\mathrm{dt} = 0$ and there is an upper limit to the range of rates at which new amino acids can be added to the chain (Fig 1.10) [49].

A similar application can be made to the central dogma, through the movement of RNA polymerase (RNAP) and the displacement of histones during transcription [51]. RNAP is responsible for the initiation of transcription by binding to the promoter at the beginning of a gene, as well as the elongation of the pre-mRNA transcript. Using DNA as its one-dimensional lattice, RNAP moves down a gene in the 5' to 3' direction by hopping along and elongating the transcript with the complement of each base in the sequence. RNAP does not move unobstructed, but rather competes for binding sites with histones [51]. Histones are protein complexes that DNA wraps around, changing what sequences of DNA are available for RNAP to transcribe, thus playing a critical role in gene expression. At the end of a gene, RNAP detaches from the lattice, known as termination. TASEP can be used to model both RNAP's stochastic initiation of transcription as well as the displacement of histones from the lattice during elongation [51].

Yet another application of TASEP can be found in gel electrophoresis, a technique in molecular genetics for separating DNA fragments by length using electric current [47, 52, 53, 54]. The

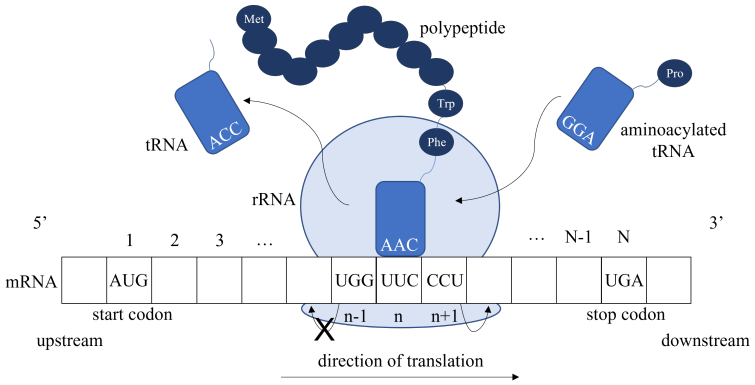


Figure 1.10: Example of TASEP in protein translation. Each box on the mRNA transcript lattice represents a three-base codon. A ribosome, rRNA in light blue, attaches to the mRNA transcript and scans for the start codon to begin translation. Aminoacylated tRNAs (blue boxes with navy circles) align their anti-codon to the codon, and attach their amino acid to the growing chain to form a polypeptide. The ribosome moves 5' $\rightarrow$ 3' direction, demonstrating a probability of movement in only one direction. Further, the ribosome occupies three codons at a time, preventing other ribosomes or proteins from attaching to the mRNA transcript at that time. Adapted from [50].

gel through which DNA fragments diffuse can be represented by a square lattice, where each site is a pore in the gel. Each DNA fragments represents a traditional TASEP particle, with only one fragment occupying one pore at a time [52]. In an idealized, pure system, all particles would respond the same to the applied electric field, with identical bond lengths and equal transition rates as they move through the gel asymmetrically [52]. In the presence of impure segments, the TASEP model can be adapted, as different segments associate with different charges, leading to different bond lengths and non-uniform transition rates [52]. Essentially, impurities in gel electrophoresis that lead DNA to behave differently to the electric field corresponds to a TASEP system in which each particle follows different attachment rates at each site in the lattice [52].

Animal behavior can also be modeled with TASEP. Using ant trails as a one-dimensional lattice, ants move down the trail and lay pheromones, neurochemical signals used for communication, as they go [55]. These pheromones increase the probability of an ant hopping to the next unoccupied site on the lattice [55]. The system is dependent on both the probability of ant motion and the evaporation of pheromones, and their dynamics can be modeled either in parallel or randomly (Fig 1.11) [55].

The extent of biological applications of TASEP speaks to its relevance and demonstrates the versatility of the present model of kinesin on a dynamic microtubule.

1.4 Thesis Structure

Building on previous models, I begin by presenting a general, versatile model in which the motors travel along a one-dimensional track of variable length. The change in length captures microtubule instability and rescue, the transition between polymerization and depolymerization.

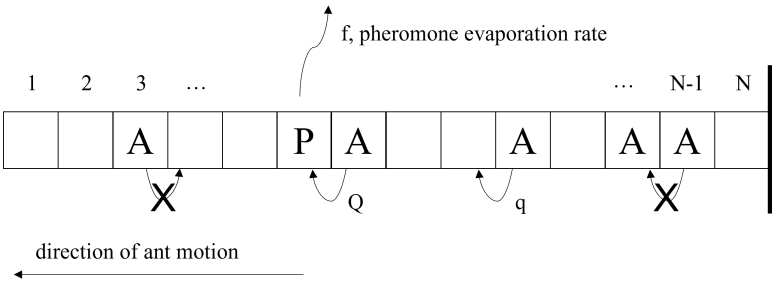


Figure 1.11: Example of TASEP to the unidirectional motion of ants as they move along a trail, where probabilities of stepping forward change with pheromones. Ants, A, with unoccupied sites in front of them move forward with probability q. Ants with pheromones, P, in front of them move forward with probability Q, and pheromones evaporate at rate f. Ants with another ant in front of them do not move forward. While simplified, this is highly analogous to the motion of kinesin along a microtubule introduced in Chapter 2. Adapted from [56].

Usually this process is caused by the loss of the GTP cap, but it is occasionally triggered by depolymerizing motors at the end of the microtubules, particularly kinesins 4, 5, 8, and 13 (Table 1.2) [23, 57]. For example, when Kinesin-13 reaches the end of the microtubule, the motor stalls in a state prior to ATP hydrolysis, during which the tubulin subunit bends. The bent conformation of tubulin is more prone to catastrophe, increasing the frequency of depolymerization [57, 58]. The model can be adapted for members of the kinesin-8 subfamily, which has a higher processivity than kinesin-13 and depolymerize multiple microtubule subunits before detaching [59, 60]. The model becomes more complex as kinesins can also attach and detach as they travel along the track. I employ the mean-field method to study the model analytically and complement it with Monte Carlo simulations.

In the first chapter, I introduce the general model in the context of the traffic of kinesin motors on microtubules. This simplified system includes unidirectional movement and bulk attachement and detachment, without regard to concentration of ATP, temperature, impacts of other microtubule associated proteins, interactions between motors, or lateral interactions between protofilaments. It also assumes rate of polymerization for each protofilament in the microtubule to be the same, which is again an oversimplification necessary for the introduction of basic model rules. I highlight some interesting results for special cases, and discuss the general case. I then acknowledge the limitations of the mathematical techniques employed in the development of the general model.

In the following chapter, I introduce layers of complexity by allowing for side-stepping motion in two dimensions between protofilaments. As an aside, I present the lateral interactions between protofilaments as an interesting application of a coupled harmonic oscillator. Returning to the unidimensional track with time and length dependency, I then consider the bidirectional motion, anterograde and retrograde toward either tip of the microtubule, representative of different types of motor molecules.

Given the number of parameters in the fully expanded model, it would be nearly impossible

to run by hand every combination of parameters, understand their physical meaning, and interpret their biological applications. Instead, I suggest machine learning algorithms as a potential method for TASEP modeling, which can be employed in a sensitivity analysis in future iterations of the project.

Though any model struggles to capture the complex (and yet to be fully understood) biological phenomena, the general model, special cases, and subsequent expansions attempt to capture the dynamic conditions, emulate empirical results from experimental literature, and provide a foundation for advanced model development.

Chapter 2

Unidirectional Stochastic Model in One-Dimension

2.1 Model Rules

Inspired by coupled dynamics of the motors motion and the microtubules, I define the model on a one-dimensional lattice of variable length (the microtubule) with N sites. N is not constant. Each site can be empty or occupied by a particle (the representation of our molecular motor) and is described by an occupation number $n_i = 0$ for empty and $n_i = 1$ for occupied. The right end is fixed (site N), and the left end (site 1) can extend, remain the same, or contract. Following the same method as in [61, 62], I choose a reference frame attached to the growing tip of the filament, with site 1 defined as the leftmost site. Any site "i" measures the distance of that site from the fluctuating tip. When the lattice grows or shrinks due to the attachment or detachment of a new site at the tip, all the site labels are updated $i \to i \pm 1$.

The traffic of particles happens from right to left. I assumed unidirectionality as a first approximation. Almost all kinesins undergo unidirectional motion, as interactions of polar structures between kinesins and microtubule subunits energetically favor the microtubule plus-end [63]. This phenomenon is not consistent through all molecular motors, evident by the bidirectionality of some fungal members of the kinesin-5 subfamily and the dynein family of motors [64]. Retrograde motion (toward the negatively charged end of the microtubule) characteristic of the dynein family of motors is introduced in Chapter 4, but the current model focuses strictly on the anterograde (positive-end) motion of most kinesins.

The system evolves according to the following rules (depicted in Fig. 2.1):

At sites 1 and 2:

- **10** $\to$ **00** with rate $\boldsymbol{\beta}$: particles leave the track with exit rate $\boldsymbol{\beta}$;
- **01** $\to$ **10** with rate **1**: particles move from right to left if the neighboring site is empty;
- **00** $\to$ **000** with rate $\boldsymbol{\gamma}$: the length of the track increases by one unit when the two first sites are empty; this is equivalent to spontaneous polymerization of a microtubule;

- **11 → 0** with rate $\boldsymbol{\delta}$: the length of the track decreases by one unit when two particles (occupying site 1 and 2) are present; at the same time, the particles leaves the track; this is equivalent to motor-induced depolymerization for a microtubule;

Two terminal tubulin dimers are depicted as unoccupied to demonstrate that spontaneous polymerization is a motor-independent process made possible by stability provided by the GTP cap. The mechanism of the motors' steps is electrostatically dependent on the tubulin dimers [60]. Consequently, during spontaneous polymerization, the presence of the GTP cap prevents motors from occupying terminal sites of the microtubule.

Depolymerization of the microtubule can occur due to the loss of the GTP cap (a motor-independent process) or from stimulation by depolymerizing motors (a motor-dependent process). In Fig. 1(c), I depict motor-induced depolymerization with the terminal sites of the microtubule occupied by depolymerizing motors. In the case of spontaneous depolymerization, (often referred to as *catastrophe*) the loss of tubulin subunits is attributed to instability from the loss of the GTP cap, not the presence of kinesins, and thus would occur with terminal sites vacant.

In bulk (sites $\boldsymbol{n_i = 3...N-1}$):

- **01 → 10** with rate **1**: particles move to the left with rate 1 as long as the neighbor to the left is empty;

- **1 → 0** with rate $\boldsymbol{\omega_D}$: if site is occupied, remove particle with rate $\boldsymbol{\omega_D}$;

- **0 → 1** with rate $\boldsymbol{\omega_A}$: if site is empty, add particle with rate $\boldsymbol{\omega_A}$;

At site N:

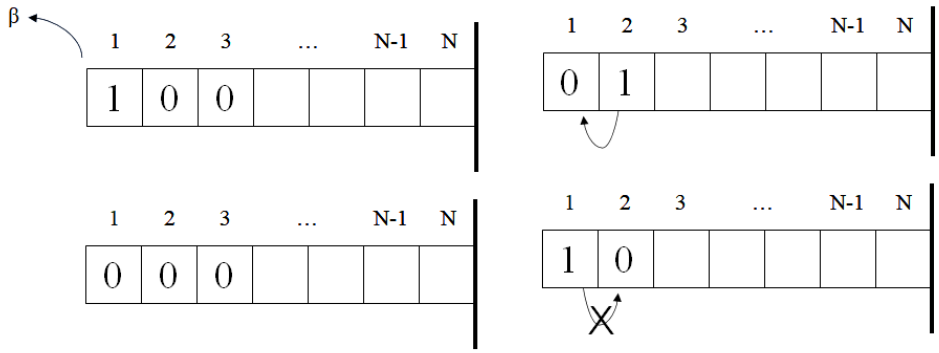
- **00 → 01** with rate $\boldsymbol{\alpha}$: particles enter the track with entrance rate $\boldsymbol{\alpha}$;

- **01 → 10** with rate **1**: diffusion to the left with rate 1;

2.2 Mean Field Method

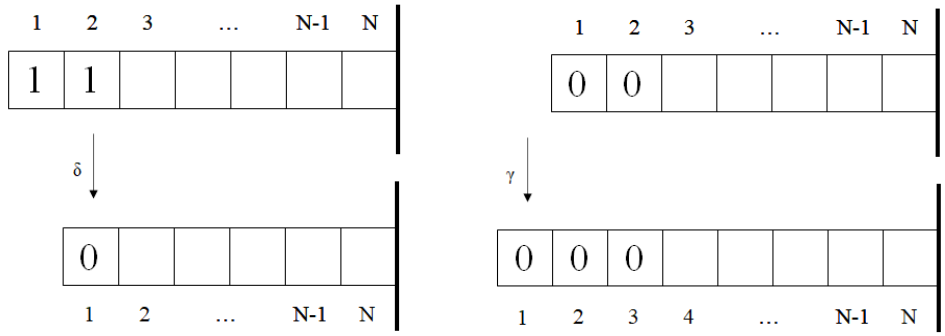
A common strategy in statistical physics for the analysis of a stochastic model is to study a simpler model. Consequently, I apply mean field theory and approximate the stochastic model by averaging over the degrees of freedom. This reduces a multi-body problem to a single-body problem, allowing for analytically solvable solutions.

I present below the evolution equations for the site occupation numbers. For boundary sites 1, 2, 3 and $\boldsymbol{N}$ the equations are:



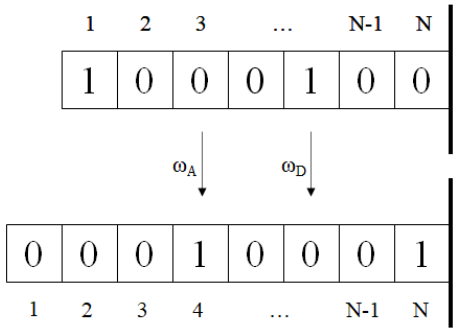
(a) Motors leave track with rate $\boldsymbol{\beta}$

(b) Unidirectional diffusion of motors



(c) Motor-induced depolymerization with rate $\boldsymbol{\delta}$

(d) Spontaneous polymerization with rate $\boldsymbol{\gamma}$



(e) Langmuir Kinetics: motor attachment with rate $\boldsymbol{\omega_A}$ and motor detachment with rate $\boldsymbol{\omega_D}$

Figure 2.1: Model Rules

$$
\begin{aligned}
\frac{dn_1}{dt} &= -\beta n_1(1-n_2) + n_2(1-n_1) - \delta n_1 n_2 \qquad (2.1)\\
\frac{dn_2}{dt} &= n_3(1-n_2) - n_2(1-n_1) - \delta n_1 n_2 + \delta n_1 n_2 n_3\\
\frac{dn_3}{dt} &= n_4(1-n_3) - n_3(1-n_2) - \omega_D n_3 + \omega_A(1-n_3) - \gamma(1-n_1)(1-n_2)n_3 + \delta n_1 n_2(n_4-n_3)\\
\frac{dn_N}{dt} &= \alpha(1-n_N) - n_N(1-n_{N-1})
\end{aligned}
$$

The evolution of occupation of sites "1" and "2" in time is dictated by the exit of particles with rate $\boldsymbol{\beta}$, the diffusion of particles from site "2" into site "1" with rate 1, the spontaneous polymerization of the microtubule with rate $\boldsymbol{\gamma}$ and the motor-induced depolymerization of the tubule with rate $\boldsymbol{\delta}$. For example, if both sites (1 and 2) are occupied, the microtubule may shrink due to depolymerization, with the loss of both sites reflected in the "–" sign of the $\boldsymbol{\delta}$ terms. Similarly, the $\boldsymbol{\gamma}$ term in the evolution equation for site **3** captures the spontaneous polymerization of the microtubule if the first two sites are empty. In this case, site "3" becomes site "4" due to the lengthening of the track and this is counted with a negative sign in the equation.

And for the bulk sites, $\boldsymbol{n_i = 4...N-1}$:

$$\frac{\boldsymbol{dn_i}}{\boldsymbol{dt}} = \boldsymbol{n_{i+1}(1-n_i)-n_i(1-n_{i-1})+\gamma(1-n_1)(1-n_2)(n_{i-1}-n_i)+\delta n_1 n_2(n_{i+1}-n_i)-\omega_D n_i+\omega_A(1-n_i)} \tag{2.2}$$

In the bulk, the additional processes considered are the attachment and detachment of particles with rates $\boldsymbol{\omega_A}$ and $\boldsymbol{\omega_D}$, according to Langmuir dynamics.

In analyzing these equations, I use the mean-field method presented in [2, 3, 61, 62]. I start by defining the mean local densities as the mean ensemble values of the site occupation numbers $\boldsymbol{\rho_i} =< \boldsymbol{n_i} >$ and I replace the mean correlations between site occupation numbers with the product of their averages: $< \boldsymbol{n_i n_j} >= \boldsymbol{\rho_i \rho_j}$. With these assumptions in mind, the equations above become:

$$\begin{aligned}
\frac{\boldsymbol{d\rho_1}}{\boldsymbol{dt}} &= \boldsymbol{-\beta\rho_1(1-\rho_2)+\rho_2(1-\rho_1)-\delta\rho_1\rho_2} && (2.3)\\
\frac{\boldsymbol{d\rho_2}}{\boldsymbol{dt}} &= \boldsymbol{\rho_3(1-\rho_2)-\rho_2(1-\rho_1)-\delta\rho_1\rho_2+\delta\rho_1\rho_2\rho_3} &&\\
\frac{\boldsymbol{d\rho_3}}{\boldsymbol{dt}} &= \boldsymbol{\rho_4(1-\rho_3)-\rho_3(1-\rho_2)-\omega_D\rho_3+\omega_A(1-\rho_3)-\gamma(1-\rho_1)(1-\rho_2)\rho_3} &&\\
&+ \boldsymbol{\delta\rho_1\rho_2(\rho_4-\rho_3)} && (2.4)
\end{aligned}$$

The evolution equation for site N is:

$$\frac{\boldsymbol{d\rho_N}}{\boldsymbol{dt}} = \boldsymbol{\alpha(1-\rho_N)-\rho_N(1-\rho_{N-1})} \tag{2.5}$$

And for the bulk sites, $\boldsymbol{n_i = 4...N-1}$:

$$\frac{\boldsymbol{d\rho_i}}{\boldsymbol{dt}} = \boldsymbol{\rho_{i+1}(1-\rho_i)-\rho_i(1-\rho_{i-1})+\gamma(1-\rho_1)(1-\rho_2)(\rho_{i-1}-\rho_i)+\delta\rho_1\rho_2(\rho_{i+1}-\rho_i)-\omega_D\rho_i+\omega_A(1-\rho_i)} \tag{2.6}$$

In order to extract information about the system's behavior, I study the system in the thermodynamic limit, $N \to \infty$. Based on the Monte Carlo simulation results that I will discuss later in the paper, it is worth noting that the system is mostly in a growth phase, unless the shrink rate $\delta > 0.1$. Sites 1, 2 and 3 are the sites affected by the particle dynamics at the fluctuating tip. The approximation $\rho_4 = \rho_3$ serves two purposes: it permits an analytical solution for the four densities (ρ_1, ρ_2 and $\rho_3 = \rho_4$) by closing the system of equations and it also captures the behavior of the system in the thermodynamic limit of $N \to \infty$, when the bulk phase adjoining the fluctuating tip is a high-density phase. The analytical expressions for ρ_1, ρ_2 and $\rho_3 = \rho_4$ are too complicated to report here, so I will continue our analysis using these symbols as placeholders for densities obtained numerically for specific sets of parameters.

To obtain a continuous equation for the bulk density ($n_i = 4...N-1$), I follow the standard method presented in [10, 2]. I define the following variables: $\epsilon = \frac{L}{N}$, the spacing of the lattice given the length of the lattice L and the number of sites N; and $x = \frac{i\epsilon}{L}$, which measures the relative position of the particle in the variable lattice (measured with respect to the left end). The lattice length L and the number of lattice sites N both increase or decrease due the polymerization/depolymerization processes which occur at tip of the filament (the left end) in such a way that their ratio ϵ remains constant. I also introduce the reduced rates, $\Omega_A = N\omega_A$ and $\Omega_D = N\omega_D$ as the total bulk attachment and detachment rates of particles for the entire system. I use the series approximation: $\rho(x \pm \epsilon) = \rho(x) \pm \epsilon\partial_x\rho(x) \pm \frac{1}{2}\epsilon^2\partial_x^2\rho(x) + O(\epsilon^3)$ and will keep just the two first terms in the series.

The continuous expression of Eq. 5 for the steady state becomes:

$$(C - 2\rho(x)) \left(\frac{d\rho(x)}{dx}\right) - (\Omega_A + \Omega_D)\rho(x) + \Omega_A = 0 \tag{2.7}$$

where $C = \gamma(1 - \rho_2)(\rho_1 - 1) + \delta\rho_1\rho_2 + 1$. This constant will have a numerical value for every set of parameters.

This equation can be rewritten as a continuity equation:

$$\frac{dJ(x)}{dx} - (\Omega_A + \Omega_D)\rho(x) + \Omega_A = 0 \tag{2.8}$$

where the particle current is defined as $J(x) = \rho(C - \rho)$.

The continuous version of the evolution equation for the fixed tip (re-scaled as $x = 1$ with the approximation $N - 1 \approx N$ in the thermodynamic limit) is:

$$\frac{d\rho(1)}{dt} = \alpha(1 - \rho(1)) - \rho(1)(1 - \rho(1)) \tag{2.9}$$

I now study in more detail the general equation for the bulk density in the steady-state case, Eq. 6. With the notations: $u(x) = C - 2\rho(x)$ and $\Omega = \Omega_A + \Omega_D$, the equation becomes:

$$u(x)\left(\frac{du(x)}{dx} - \Omega\right) = 2\Omega_A - \Omega C \tag{2.10}$$

For clarity, I will first discuss special cases of this equation, which lead to other interesting applications of this model. I study these special cases using the mean field method in conjunction with the Monte Carlo simulations. Before I discuss these cases, I will briefly present in the following section the Monte Carlo method implemented in Python.

2.3 Monte Carlo Methods

2.3.1 Introduction to Monte Carlo Simulations for Non-Equilibrium Systems

Monte Carlo methods include a variety of computational algorithms that implement repeated random sampling to produce numerical solutions. Monte Carlo uses randomness to solve deterministic problems, which is especially useful in optimization, numerical integration, and probability distribution contexts when master equations are not otherwise analytically solvable.

In probability, a Markov process is a distribution of random states. In Monte Carlo simulations, the transition probabilities of switching from one state to the next is dependent on the distribution of random states [65]. This is in direct contrast with mean-field theory, in which transition probabilities depend on sequential interacting particles [65].

Markov Chain Monte Carlo (MCMC) is a class of algorithms for sampling a probability distribution and is highly versatile in non-equilibrium statistical physics. A common application is in the discussion of phase transitions. At critical points, the particles in a system slow to the point where they do not relax to equilibrium in the thermodynamic limit [66]. Analysis of these systems in non-equilibrium evolution can include correlation functions, scaling functions and amplitude ratios, which are all well suited to Monte Carlo methods [66].

Though molecular dynamics simulations are common for sampling complex multidimensional systems [67], these simulations are not perfect. Local changes in configuration result in highly correlated samples, which slows the convergence of estimated expectations and requires the use of small time steps to provide numerical stability [67]. This problem, known as *critical slowing down*, is combated by non-local, or cluster, methods. Non-local methods make changes over large subdomains of the configuration, making cluster methods especially effective at reducing critical slowing down in large systems near their critical points [65, 68]. Consequently, MCMC is a powerful tool in the study of non-equilibrium systems, and will be an asset in the analysis of motor molecule dynamics.

2.3.2 Methods for Monte Carlo Analysis

To begin the Monte Carlo analysis, I first defined the parameters for track control using parameters for site length (the size of the tubulin binding site), average microtubule length, step size, and duration of time step as guided by empirical evidence from biological literature [28, 32, 33].

Track parameters were held constant for all simulations (Table 2.1).

Track control parameter	Value	Description
initial length	1000	starting length of the microtubule, defines the initial number of sites
site length	4.00E-04	site length in µm (initial length*site length = initial length in µm)
maximum length	2.0	maximum length in µm
maximum sites	10000	maximum length in number of sites (maximum length / site length)
minimum sites	30	
real duration	4000	duration in number of minutes
real Tau	0.0002	time step in minutes
duration	int(real duration / real Tau)	duration in number of time steps
total length	11*initial length	
left	10*initial length	left starts at 10*initial length to leave room for growth towards 0
right	left + initial length - 1	right fixed at the end of the array

Table 2.1: Track control parameters for the unidirectional, one-dimensional model. Biological parameters were introduced from experimental data [28, 32, 33] and remained constant while parameters for motors, such as attachment and detachment rates, were manipulated.

Motor parameters for $\boldsymbol{\Omega_A}$ and $\boldsymbol{\Omega_D}$ were defined relative to site length. Injection and ejection rates for motors, as well as growth and shrinkage rates at the left-most boundary varied depending on the desired simulation.

To model the system, an array was initially populated with zeros, representative of all empty sites. Then, particles were either injected at rates defined by the parameters, or assigned to random sites in the array. A random number generated between 0 and 1 was then multiplied by the number of sites in the array. This determined on which site the first particle attempted to attach. For example, in a lattice of 100 sites, if the random number generated is 0.2, the first site a particle would attempt new attachment would be site 20. Given this site, probability of attachment (dictated by the given parameters, occupation of the site, and occupation of its neighbors) was compared to another random number generated between 0 and 1. If the random number generated was higher than the probability of attachment, then the particle attached to the track. After updating the population of the lattice, these runs were repeated for at least as many sites existed in the initial

lattice. Often, the number of runs were double the number of sites in the lattice to ensure that no sites in the lattice were skipped by random chance.

Simulations for the density and current of motors along the lattice were generated in a method similar to [2], with the notable exception that in this simulation, motor dynamics alter track length (representative of phenomena observed with microtubule instability from the biological literature). To account for a dynamic track with time and length dependency, the structure of the simulation relied on arrays to store indices of first and last active sites of the track, as well as the total length of the track at each time step.

Data was processed in Microsoft Excel to extract trends. Code was run by varying at first one, then multiple parameters at a time, and plots were analyzed side by side in Excel to determine the emergence of patterns by parameter. Here, I present a collection of special and interesting cases from the Monte Carlo simulation. In the interest of reproducibility, I provide screenshots of the Python code employed in Appendix D.1.

2.4 Special cases

The first special case that I discuss is the equivalent of a TASEP model defined on a track of variable length. The attachment and detachment rates are set to zero $\boldsymbol{\Omega_D = \Omega_A = 0}$, so there are no coupled Langmuir dynamics. One end is fixed, while the other one can undergo growth or shrinkage. The growth and shrinkage of the tracks are triggered by the presence of pairs (*dimers*) of either empty of full sites, consistent with the biological literature on polymerization and depolymerization of microtubules [6].

Under the assumption that there are no attachment and detachment rates along the track, $\boldsymbol{\Omega_A = \Omega_D = 0}$, the steady-state equation (Eq. 9) simplifies to:

$$\boldsymbol{(C - 2\rho(x))\left(\frac{d\rho(x)}{dx}\right) = 0} \tag{2.11}$$

with $\boldsymbol{C = \gamma(1-\rho_2)(\rho_1 - 1) + \delta\rho_1\rho_2 + 1}$.

The solution for the density is not dependent on $\boldsymbol{x}$; it is a constant that incorporates the densities of boundary sites 1 and 2. Specifically, the bulk solutions for the steady state particle density and current are equal to:

$$\begin{aligned} \boldsymbol{\rho(x)} &= \boldsymbol{\frac{C}{2} = \frac{\gamma(1-\rho_2)(\rho_1-1) + \delta\rho_1\rho_2 + 1}{2}} \\ \boldsymbol{J(x)} &= \boldsymbol{\rho(C-\rho) = \frac{C^2}{4} = \frac{\left(\gamma(1-\rho_2)(\rho_1-1) + \delta\rho_1\rho_2 + 1\right)^2}{4}} \end{aligned} \tag{2.12}$$

If the length is constant ($\boldsymbol{\gamma = 0}$ and $\boldsymbol{\delta = 0}$), I recover the well-known TASEP result of a bulk density equal to 0.5 and a constant current of 0.25 for the maximum current phase, as reported in [10].

For the right boundary, the constant of integration is equal to $\boldsymbol{\alpha}$, the entrance rate, as derived from Eq. 8 for the steady state. For the left boundary, the density is equal to $\boldsymbol{\rho_3}$, considered as the boundary site. Given the high correlation between sites 1 and 2, they can be seen, intuitively, as one unit, a *dimer*.

To summarize, the density profile is a piece-wise function as follows:

$$\boldsymbol{\rho(x)} = \begin{cases} \boldsymbol{\rho_3} & \text{at the left boundary (position of the moving tip)} \\ \frac{\boldsymbol{\gamma(1-\rho_2)(\rho_1-1)+\delta\rho_1\rho_2+1}}{\boldsymbol{2}} & \text{in the bulk} \\ \boldsymbol{\alpha} & \text{at the fixed right boundary} \end{cases} \tag{2.13}$$

And for the steady-state current:

$$\boldsymbol{J(x)} = \begin{cases} \boldsymbol{\rho_3(C-\rho_3)} & \text{at the left boundary (position of the moving tip)} \\ \frac{\boldsymbol{(\gamma(1-\rho_2)(\rho_1-1)+\delta\rho_1\rho_2+1)^2}}{\boldsymbol{4}} & \text{in the bulk} \\ \boldsymbol{\alpha(C-\alpha)} & \text{at the fixed right boundary} \end{cases} \tag{2.14}$$

The discussion of density and current profiles in the parameter space becomes very complicated with so many parameters in play. For specific sets of parameters, I can find the numerical values for densities $\boldsymbol{\rho_1}$, $\boldsymbol{\rho_2}$ and $\boldsymbol{\rho_3}$, which in return lead to numerical values for the steady-state bulk densities and currents . I will not discuss here the whole phase diagram for the model and the identification of domain walls. Instead, I opt to present some special cases in the hope that they will lead to a more profound and intuitive understanding of the complexities of this model and may be used as templates for practical applications in modeling traffic at the cellular level.

2.4.1 $\boldsymbol{\delta = 0}$

The first case that I discuss is $\boldsymbol{\delta = 0}$ (no shrinkage allowed when the first two sites are occupied). This describes a microtubule in the growth (polymerization) phase for which a cap of motors at the moving tip does not trigger a shortening of the microtubule. In this situation, the mean-field solutions for the first three site densities have more manageable expressions obtained by solving Eq. 3 for the case $\boldsymbol{\delta = 0}$ and no Langmuir dynamics ($\boldsymbol{\Omega_A = \Omega_D = 0}$):

$$\begin{aligned} \boldsymbol{\rho_1} &= \frac{\boldsymbol{\gamma - \sqrt{\beta\gamma(1-\beta)}}}{\boldsymbol{\beta^2 - \beta + \gamma}} \\ \boldsymbol{\rho_2} &= \frac{\boldsymbol{\beta\rho_1}}{\boldsymbol{1 - \rho_1(1-\beta)}} \\ \boldsymbol{\rho_3} &= \boldsymbol{\beta\rho_1} \end{aligned} \tag{2.15}$$

Interestingly, if $\boldsymbol{\gamma = 0}$ as well (no polymerization allowed), I recover again the well-known TASEP result of a bulk density equal to 0.5 and a constant current of 0.25 for the classical TASEP maximum current phase [10], regardless of the values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. However, from the equations above it can be seen that if $\boldsymbol{\gamma = 0}$, $\boldsymbol{\rho_1 = \rho_2 = \rho_3 = 0}$. So, there is a cap of three empty sites

followed by a jump in density to 0.5, and the fixed end will be at a density dictated by the entrance rate $\boldsymbol{\alpha}$.

The Monte Carlo simulations tell a different story, as presented in Fig. 2.2. Due to the strong correlations between the first two sites at the moving tip, particles leave the track only when the first site is occupied and the second site is empty, and they are forbidden to leave (due to $\boldsymbol{\delta = 0}$) when both sites are occupied. Therefore, the steady-state density is close to 1 even in the case of $\boldsymbol{\beta > \alpha}$, which is different from the classic TASEP model result.

This strong correlation is not captured in the mean-field equations, and it leads to drastically different results. The bulk density is constant for $\boldsymbol{\alpha \geq \beta}$, but it settles at almost double the mean-field value. The track is almost full, which leads to a small current along the track. For $\boldsymbol{\alpha < \beta}$, the average site density has a linear dependence on the position, with a corresponding decreasing current. In the time-dependent graphs, there is an interesting feature of a maximum in current, with the peak shifted to the right for $\boldsymbol{\alpha < \beta}$.

Another interesting case worth mentioning for the growing regime ($\boldsymbol{\delta = 0}$, $\boldsymbol{\gamma \neq 0}$) is the case of $\boldsymbol{\beta = \gamma}$, for which the first three densities simplify to:

$$\begin{aligned} \boldsymbol{\rho_1} &= \frac{\mathbf{1} - \sqrt{\mathbf{1} - \boldsymbol{\beta}}}{\boldsymbol{\beta}} \\ \boldsymbol{\rho_2} &= \frac{\boldsymbol{\beta}\left(\sqrt{\mathbf{1} - \boldsymbol{\beta}} - \mathbf{1}\right)}{(\boldsymbol{\beta} - \mathbf{1})\sqrt{\mathbf{1} - \boldsymbol{\beta}} - \mathbf{2}\boldsymbol{\beta} + \mathbf{1}} \\ \boldsymbol{\rho_3} &= \mathbf{1} - \sqrt{\mathbf{1} - \boldsymbol{\beta}} \end{aligned} \tag{2.16}$$

When $\boldsymbol{\beta = \gamma \to 1}$, the tip has a cap of densities $\boldsymbol{\rho_1 = \rho_2 = \rho_3 = 1}$ and the bulk density settles at 0.5. When $\boldsymbol{\beta = \gamma \to 0}$, $\boldsymbol{\rho_2 = \rho_3 = 0}$ but $\boldsymbol{\rho_1 = 0.5}$, leading to a bulk density of again 0.5. The dependence of the bulk density and the bulk current on $\boldsymbol{\beta}$ in comparison with the Monte Carlo simulations are presented in Fig. 2.3. As can be seen from the graphs, contrasted with the simplicity of the mean-field solutions, the Monte Carlo simulations show a more complex behavior of the system that merits a more in-depth analysis. The mean-field solutions are somewhat close to the simulations results for a very small interval around $\boldsymbol{\beta = \gamma = 0.3}$.

In Fig. 2.4 I present sample Monte Carlo data for the polymerization case ($\boldsymbol{\delta = 0}$ and $\boldsymbol{\beta \neq \gamma}$). Again, I observe a discrepancy between the mean-field theory and the simulations. The qualitative solution of constant bulk density is captured by MC simulations for values of $\boldsymbol{\beta > \gamma}$, where the density is constant between $\boldsymbol{x \in (0, 0.15)}$ and then it drops to 0. Although for the small interval of $\boldsymbol{x}$ the density is constant in MC simulations, the value is twice the mean-field value of 0.42. The corresponding MC current has a peak of 0.125, close to the mean-field value of 0.17. For $\boldsymbol{\beta \leq \gamma}$ the MC bulk density displays a linear decrease along the track. The mean-field solution for the density does not capture this decrease, as it is a constant equal to 0.48. The MC current is close to the mean-field solution of 0.23 for $\boldsymbol{x \in (0, 0.1)}$.

Given the mean-field solutions (Eq. 13 and Eq. 14), by placing the condition $\boldsymbol{\rho_{left} = \rho_{middle}}$, $\boldsymbol{\rho_{right} = \rho_{middle}}$, one can easily find the boundary curves $\boldsymbol{\beta = f(\gamma)}$ and $\boldsymbol{\alpha = f(\gamma)}$ along which

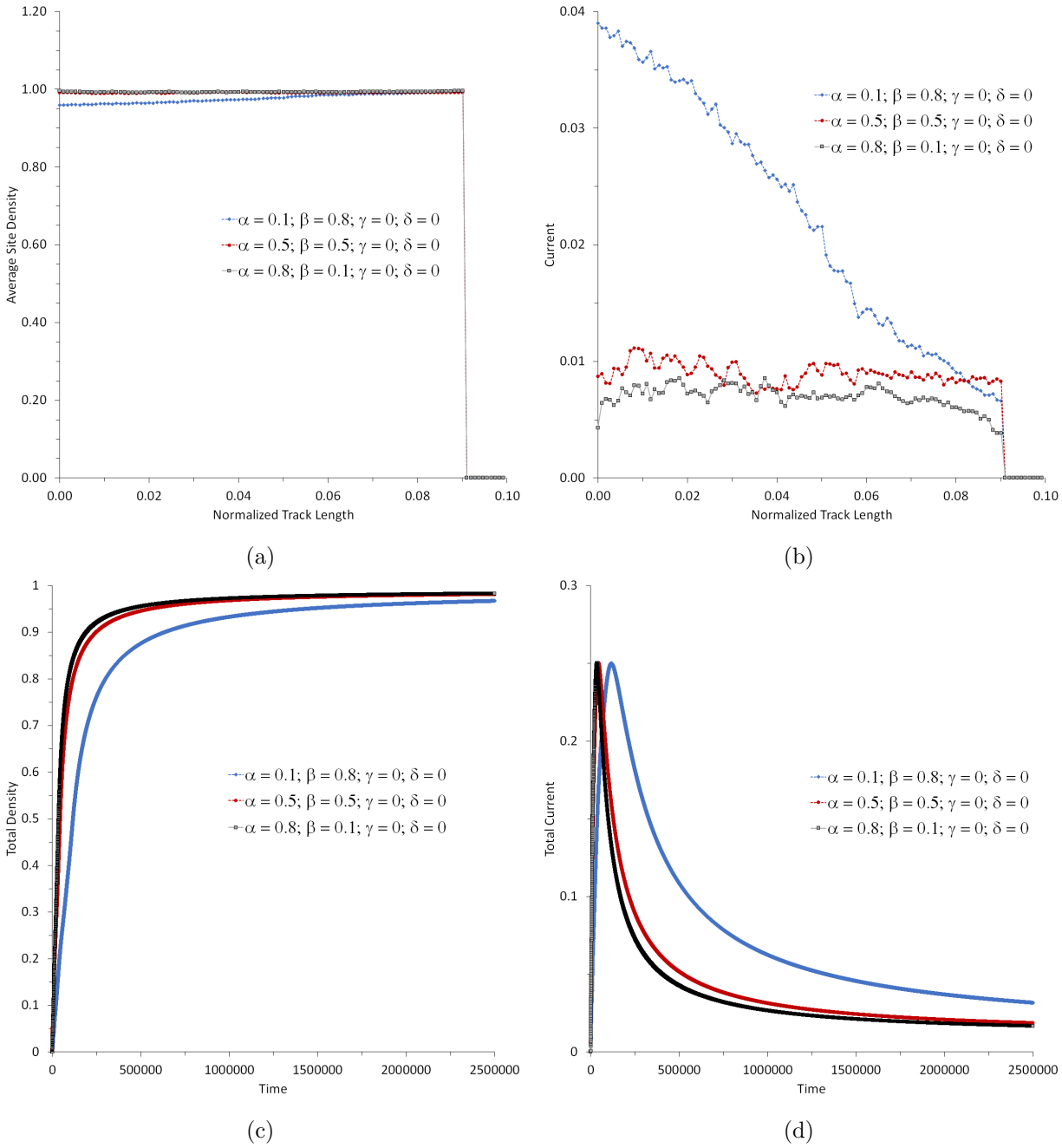
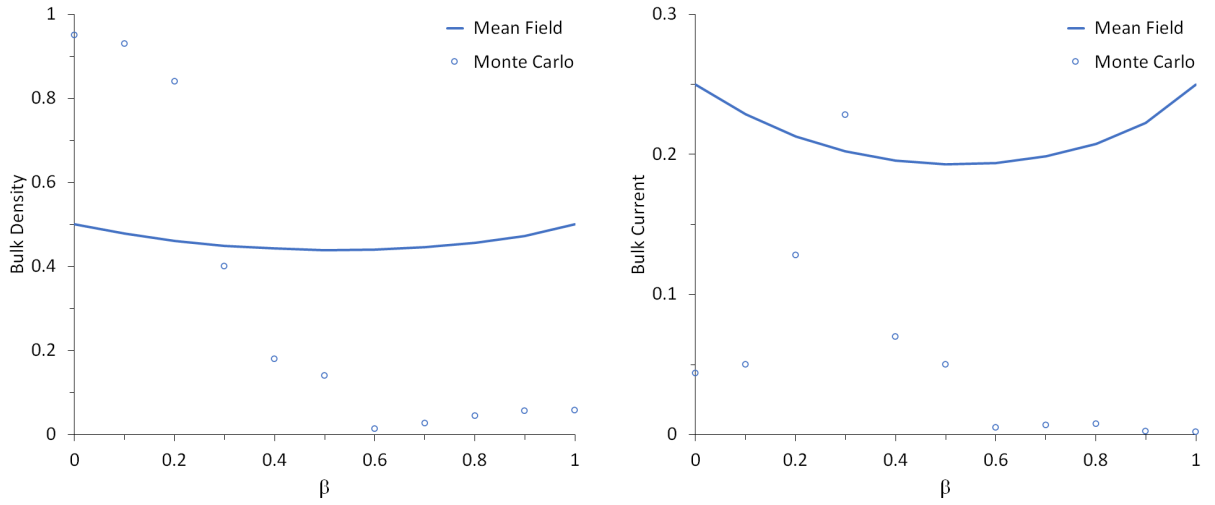


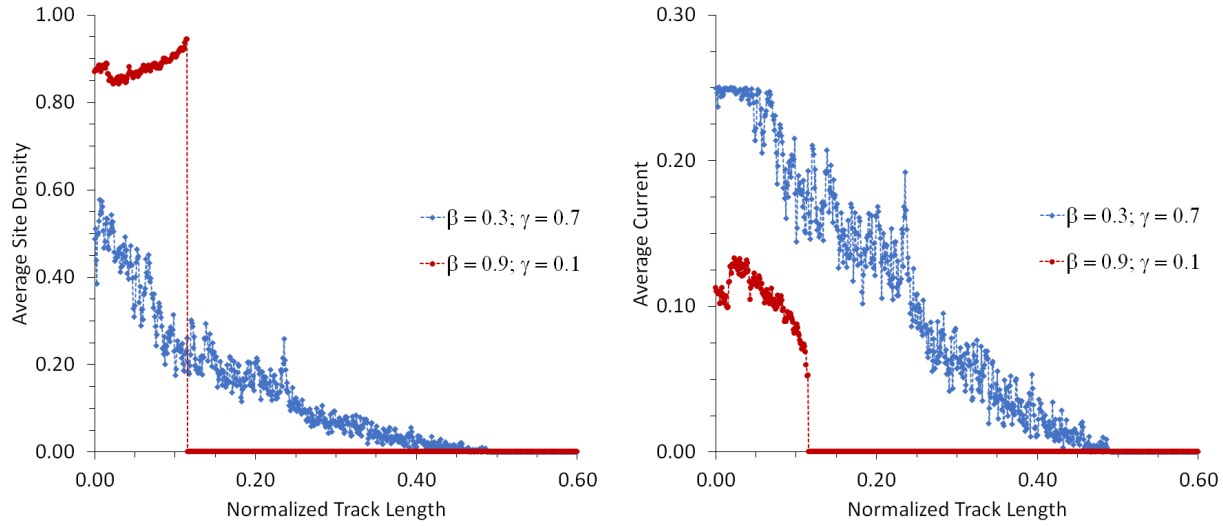
Figure 2.2: The case of constant length tracks: (a) average steady-state site density along the track; (b) average steady-state current along the track; (c) the time evolution of the total density; (d) the time evolution of the total current.

the solutions are equal to each other. The analysis can be extended much further toward finding a full phase diagram for this special case. However, given the effect of the strong correlations that are not captured in the mean-field solutions, such a diagram will not reflect correctly the parameter domains for this model. I propose that a better approach would be a more in-depth MC analysis



(a) (b)

Figure 2.3: Special case of polymerization regime: Comparison between mean-field and the MC results for bulk densities (a) and currents (b) as functions of $\boldsymbol{\beta = \gamma}$, with $\boldsymbol{\alpha = 0.5}$.



(a) (b)

Figure 2.4: Monte Carlo results for the polymerization regime $\boldsymbol{\delta = 0}$, $\boldsymbol{\alpha = 0.5}$, and initial length $\boldsymbol{N = 100}$:(a) average density along the track; (b) average current along the track.

in conjunction with a modified mean-field approach with some correlations built-in.

2.4.2 $\gamma = 0$

The other limit case is $\boldsymbol{\gamma = 0}$, for which the track can only shrink when the two sites at the tip are occupied. This describes the motor-induced depolymerization for a microtubule [11]. In

this case, the expressions for $\boldsymbol{\rho_1}$, $\boldsymbol{\rho_2}$ and $\boldsymbol{\rho_3}$ are:

$$\boldsymbol{\rho_1} = \frac{\boldsymbol{\beta + \delta - 1}}{\boldsymbol{\delta^2 + \beta - 1}} \tag{2.17}$$

$$\boldsymbol{\rho_2 = \rho_3} = \frac{\boldsymbol{\beta + \delta - 1}}{\boldsymbol{\beta - 1}} \tag{2.18}$$

and they are confined to $\boldsymbol{\beta + \delta < 1}$ to maintain their values between zero and 1.

It is interesting to see that if I set $\boldsymbol{\delta = 0}$, the first three sites have a density equal to one. In the previous case, which started with the solution for $\boldsymbol{\delta = 0}$ and then turned off $\boldsymbol{\gamma}$ as well, the cap was all empty sites. Overall, when the track is not allowed to shrink or expand, $\boldsymbol{\rho_1}$, $\boldsymbol{\rho_2}$, $\boldsymbol{\rho_3}$ are all 0 or 1, forming a cap of empty sites that will tend to polymerize or a cap of full sites that will trigger depolymerization.

Fig. 2.5 shows the average site density and the current density along the track, as well as the total density and current for two cases of $\boldsymbol{\beta < \delta}$ and $\boldsymbol{\beta > \delta}$. The mean-field theory predicts a constant bulk density and current for a given set of parameters. A qualitative agreement holds for the current in the case of $\boldsymbol{\beta = 0.8}$, $\boldsymbol{\delta = 0.1}$ for a small portion of the track $\boldsymbol{x \in (0, 0.04)}$, with $\boldsymbol{J_{MC} = 0.25}$ and $\boldsymbol{J_{MF} = 0.267}$ (a **6.8%** difference). The densities compare well for the beginning of the track: **3.4%** difference for $\boldsymbol{\beta = 0.8}$, $\boldsymbol{\delta = 0.1}$, but the trend shown in Monte Carlo simulations is not the one predicted by the mean-field theory. For the $\boldsymbol{\beta = 0.1}$, $\boldsymbol{\delta = 0.8}$ case, the system doesn't reach a steady-state. It is interesting to see the unusual behavior of the total density and total current as functions of time. From the plots, I can identify the position of the domain walls where the current suddenly drops to a lower value, then continues to decrease linearly, signaling the presence of traffic jams. Because there is no steady-state, Fig. 2.5 (a) and Fig. 2.5 (b) represent the average site density and average site current along the track after the system underwent a set number of Monte Carlo steps. It is a snapshot of the system, but not its steady-state. These unusual features deserve a further systematic investigation in future studies.

I finish the section with a sample set of data for the case of $\boldsymbol{\gamma \neq \delta \neq 0}$: $\boldsymbol{\alpha = 0.8}$, $\boldsymbol{\beta = 0.1}$, $\boldsymbol{\gamma = 0.07}$ and $\boldsymbol{\delta = 0.95}$ and an initial number of 100 sites and a maximum number of 500 sites (Fig. 2.6). This sample case captures the switch between growth and shrinkage for the microtubules, known in literature as the *microtubule instability*. For the Monte Carlo simulations I ran a whole range of track sizes, from 100 to 5000 initial number of sites. The main results remain qualitatively the same.

The mean field predicts in this case a bulk density of $\boldsymbol{\rho = \frac{1-\gamma}{2}}$, the Monte Carlo simulations give us a value of approximately 0.46, and the mean-field current has a value of 0.25. Although the mean-field results for the bulk density and current seem to agree with the simulations, this comparison doesn't take into account the time scale of the track length fluctuations and the time it takes for the system to reach steady state. In our model they are on the same order. If the time scale of the track length fluctuations were much smaller than the time needed to reach steady state, one may consider the track to be of essentially constant length.

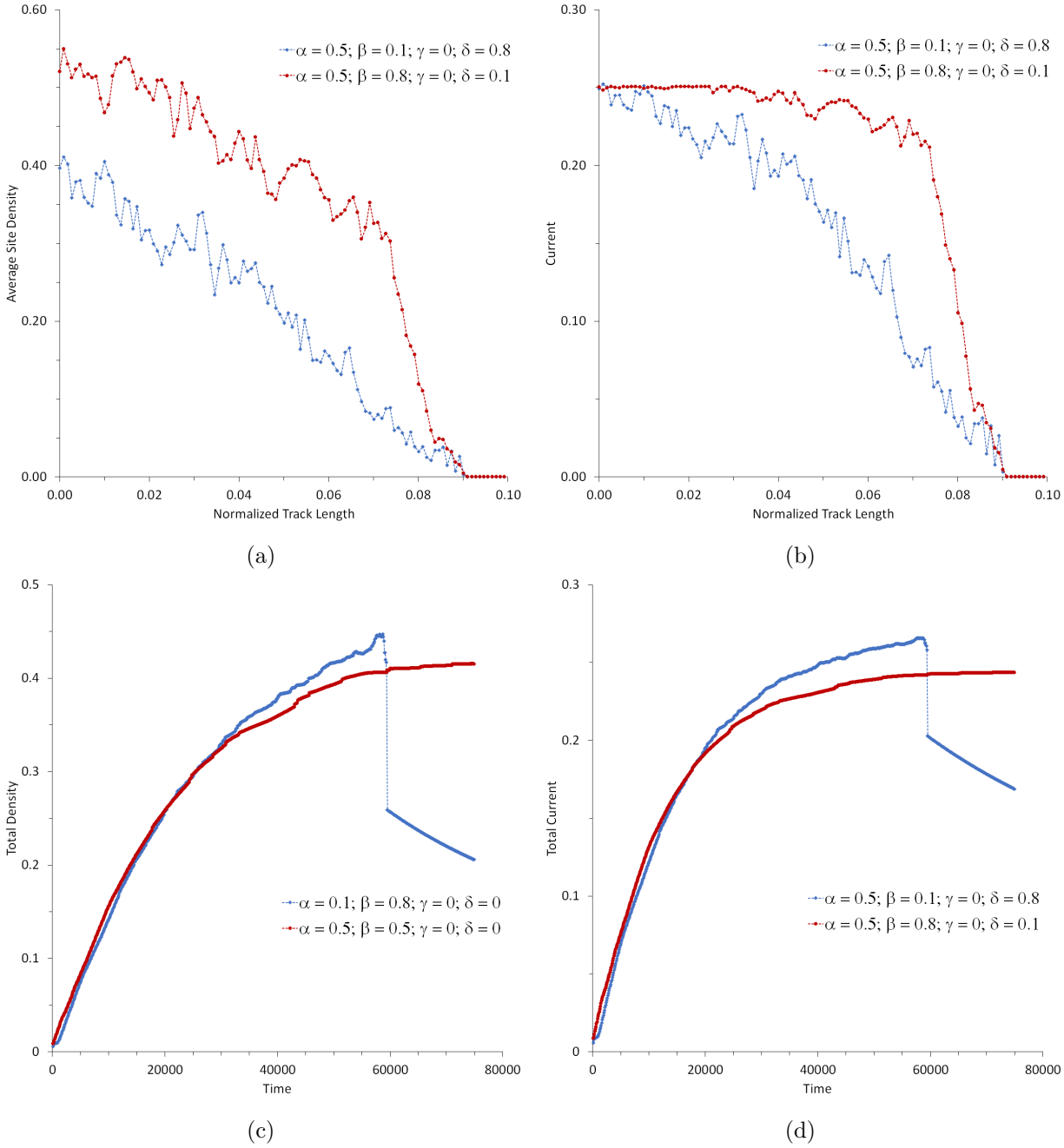


Figure 2.5: Monte Carlo results for the depolymerization regime $\boldsymbol{\gamma = 0}$ and initial length $\boldsymbol{N = 100}$: (a) average site density along the track; (b) average site current along the track; (c) the time evolution of the total density; (d) the time evolution of the total current.

2.4.3 Time-Dependent Solutions

The time-dependent solution of the dynamic TASEP model is very similar to the one presented, for example, in [10]. The time-dependent continuous equation can be written as:

$$\frac{\partial \rho(x,t)}{\partial t} + (C - 2\rho(x,t)) \left(\frac{\partial \rho(x,t)}{\partial x} \right) = 0 \tag{2.19}$$

The transformation $u = C - 2\rho$ recasts the equation into the inviscid Burgers equation [10]:

$$\frac{\partial u(x,t)}{\partial t} + u(x,t) \frac{\partial u(x,t)}{\partial x} = 0 \tag{2.20}$$

The characteristic for this first order nonlinear partial differential equation is a straight line in the the $x - t$ plane with a slope given by:

$$\frac{dx}{dt} = C - 2\rho \tag{2.21}$$

Compared to the TASEP model defined on a fixed length for which the slope is $1 - 2\rho$, here the slope is $C - 2\rho$. I invite the reader to revisit the time-dependent solution of the TASEP model defined on a constant-length lattice as presented in [10], as the theory applies here as well, with the appropriate adjustments for γ and δ. The change in the slope due to the increasing or decreasing rates γ and δ is incorporated in the factor C.

2.5 General case

I now return to the original case of the coupled TASEP-dynamics with Langmuir kinetics defined on the one-dimensional lattice with variable length. The governing equation for the steady-state in the hydrodynamic approximation is:

$$u(x) \left(\frac{du(x)}{dx} - \Omega \right) = 2\Omega_A - \Omega C \tag{2.22}$$

with $u(x) = C - 2\rho(x)$ and $\Omega = \Omega_A + \Omega_D$.

If I introduce the extra constraint $2\Omega_A - \Omega C = 0$, the equation above leads to solutions similar to the ones presented in [2]. The correction due to variation in length represented by δ and γ affects the bulk density, which is still a constant for a given set of parameters.

$$\rho(x) = \begin{cases} \Omega x + \rho_3 & \text{if } 0 \leq x \leq x_{left} \\ \frac{\gamma(1-\rho_2)(\rho_1 - 1) + \delta \rho_1 \rho_2 + 1}{2} & \text{if } x_{left} \leq x \leq x_{right} \\ \Omega(x-1) + \alpha & \text{if } x_{right} \leq x \leq 1 \end{cases} \tag{2.23}$$

And for the steady-state current:

$$J(x) = \begin{cases} (\Omega x + \rho_3)(C - \Omega x - \rho_3) & \text{if } 0 \leq x \leq x_{left} \\ \frac{(\gamma(1-\rho_2)(\rho_1 - 1) + \delta \rho_1 \rho_2 + 1)^2}{4} & \text{if } x_{left} \leq x \leq x_{right} \\ (\Omega(x-1) + \alpha)(C - \Omega(x-1) - \alpha) & \text{if } x_{right} \leq x \leq 1 \end{cases} \tag{2.24}$$

A specific set of parameters for which the extra constraint $\mathbf{2\Omega_A - \Omega C = 0}$ is obeyed is: $\boldsymbol{\beta = 0.5}$, $\boldsymbol{\delta = 0.5}$, $\boldsymbol{\alpha = 0.5}$, $\boldsymbol{\Omega_A = \Omega_D = 0.5}$, $\boldsymbol{\gamma = 0.25}$. The mean-field predicts the following numerical values: $\boldsymbol{\rho_1 = 0.439}$, $\boldsymbol{\rho_2 = 0.392}$, $\boldsymbol{\rho_3 = 0.440}$ and $\boldsymbol{\rho_{bulk} = 0.5}$. The $\boldsymbol{x_{left}}$ and $\boldsymbol{x_{right}}$ boundaries are found by equating the left and the bulk solutions and the right and the bulk solutions. These boundaries will change with the change of the parameters. For this special case, $\boldsymbol{x_{left} = 0.06}$ and $\boldsymbol{x_{right} = 1}$, which means that the bulk solution for the density is a good approximation for the whole system. Because of the multitude of parameters, building a phase diagram for the general case becomes a messy endeavor. [2] is a good resource that outlines the steps one needs to follow in order to identify the regions of high and low density, the maximum current phase and the position of the domain walls (found by matching $\boldsymbol{J_{left} = J_{right}}$). I would like to emphasize, however, that due to the effect of correlations between sites at the tip of the microtubule, the mean-field solutions are able to qualitatively match the Monte Carlo solutions only for specific cases, as seen in some examples earlier in the paper.

The Monte Carlo simulations for densities and currents are presented in Fig. 2.7 for the two cases of systems with and without parameters obeying the constraint. It can be observed from the plots that the behavior of the system is very different for the two situations. When the constraint is obeyed, the system is in a growing regime, as opposed to a switch between growth and shrinkage as seen in Fig. 2.7(e).

If I don't introduce the extra constraint $\mathbf{2\Omega_A - \Omega C = 0}$, I solve the general equation using separation of variables methods. The integration of this equation leads to:

$$\frac{\boldsymbol{K}\ln(\boldsymbol{K} - \boldsymbol{\Omega u(x)}) + \boldsymbol{\Omega u(x)}}{\boldsymbol{\Omega^2}} = (\boldsymbol{x} - \boldsymbol{x_b}) \tag{2.25}$$

with the new constant $\boldsymbol{K = \Omega C - 2\Omega_A}$.

In a more compact form, the solution can be reported using LambertW special function:

$$\boldsymbol{u}(\boldsymbol{x}) = -\frac{\boldsymbol{K}}{\boldsymbol{\omega}}\left(\mathbf{LambertW}\left(-\frac{\mathbf{1}}{\boldsymbol{K}}\mathbf{e}^{-\mathbf{1}-\frac{K_1\omega^2}{K}-\frac{x\omega^2}{K}}\right)+\mathbf{1}\right) \tag{2.26}$$

where $\boldsymbol{K_1}$ is a new constant of integration found by matching the boundary conditions. This leads to a solution for $\boldsymbol{\rho(x)}$:

$$\boldsymbol{\rho}(\boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{2}}\left(\boldsymbol{C} + \frac{\boldsymbol{K}}{\boldsymbol{\omega}}\left(\mathbf{LambertW}\left(-\frac{\mathbf{1}}{\boldsymbol{K}}\mathbf{e}^{-\mathbf{1}-\frac{K_1\omega^2}{K}-\frac{x\omega^2}{K}}\right)+\mathbf{1}\right)\right) \tag{2.27}$$

Although there is a somewhat qualitative agreement between simulations and mean-field theory for the general case with the constraint in place, the mean-field theory fails to capture the intricacies of the model dynamics when the constraint is lifted, as seen in Fig. 2.7.

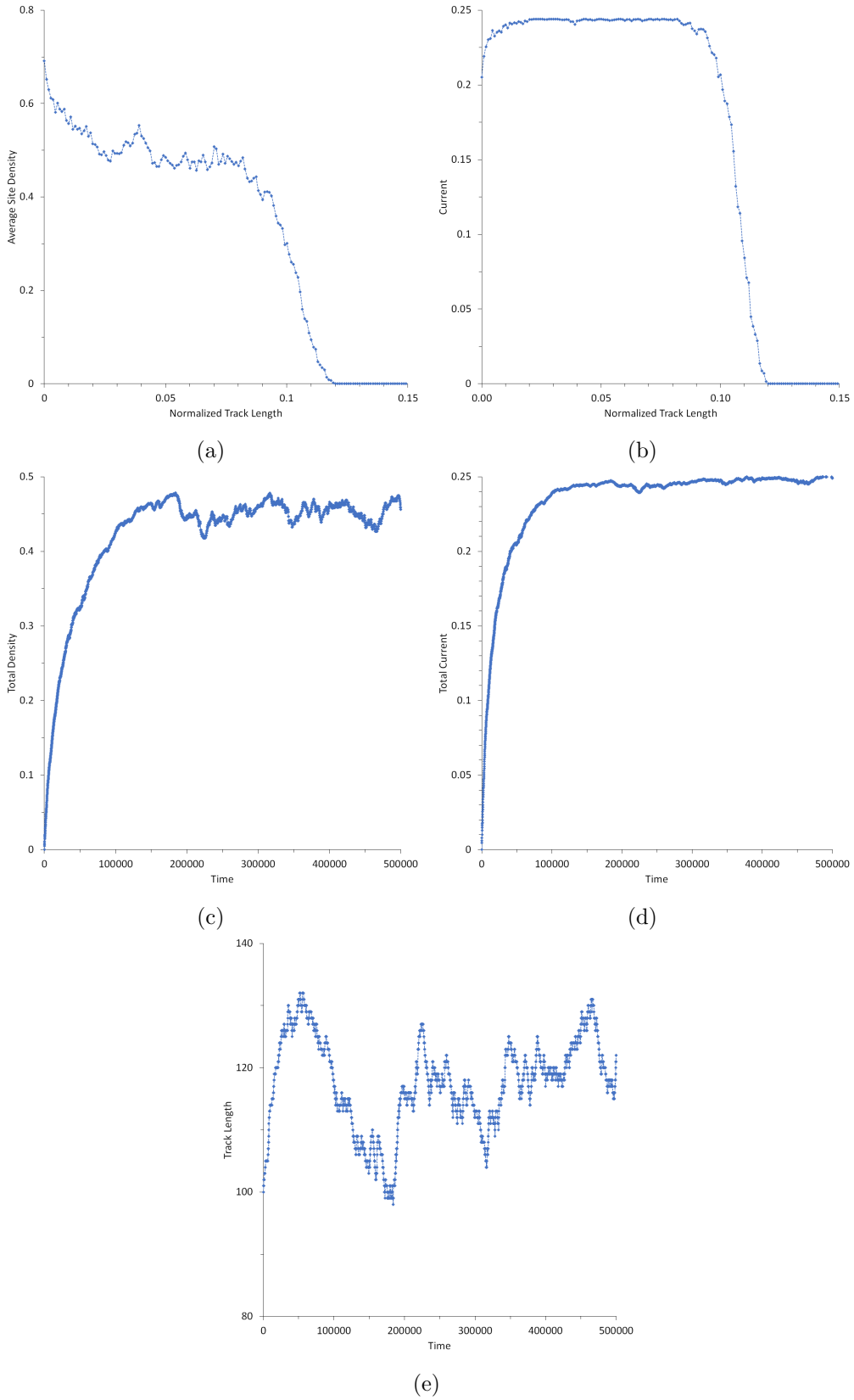
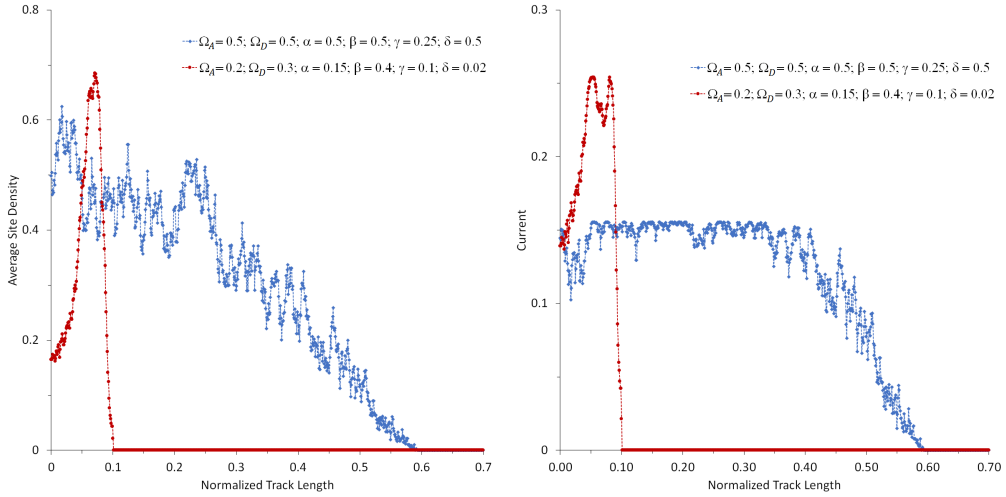
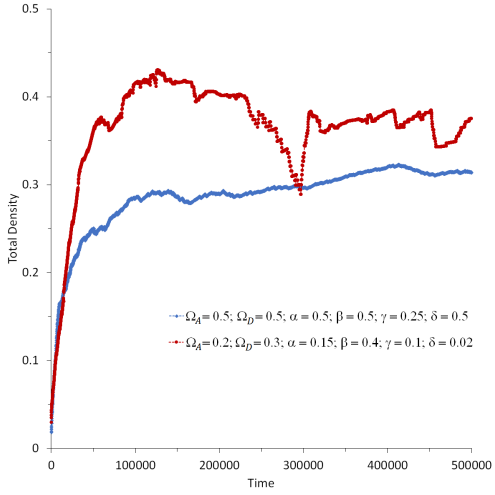


Figure 2.6: Microtubule instability. Monte Carlo results for the case of $\boldsymbol{\gamma \neq \delta \neq 0}$ and initial length $\boldsymbol{N = 100}$ and maximum number of 500 sites, $\boldsymbol{\alpha = 0.8}$, $\boldsymbol{\beta = 0.1}$, $\boldsymbol{\gamma = 0.07}$ and $\boldsymbol{\delta = 0.95}$: (a) average density along the track; (b) average current along the track; (c) the time evolution of the total density; (d) the time evolution of the total current; (e) The average length of the microtubule as a function of time.

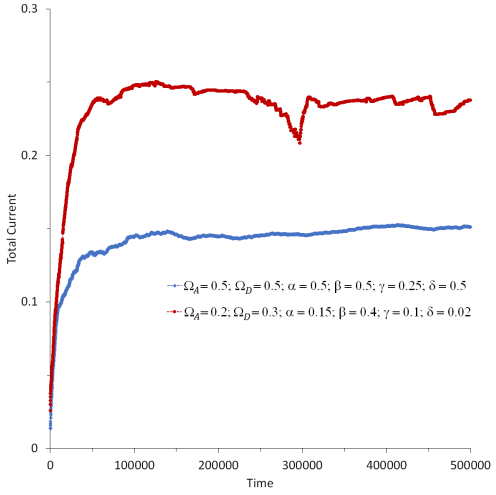


(a)

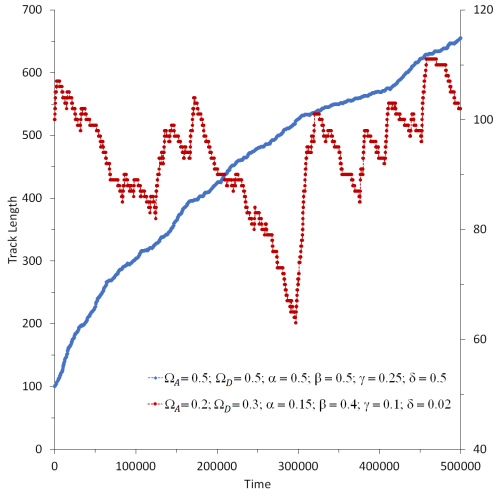
(b)



(c)



(d)



(e)

Figure 2.7: General case. Sample Monte Carlo results for the case of $\boldsymbol{\gamma \neq \delta \neq 0}$ and $\boldsymbol{\omega_D \neq 0}$, $\boldsymbol{\omega_A \neq 0}$, initial length $\boldsymbol{N = 100}$ and maximum number of 700 sites. (a) average density along the track; (b) average current along the track; (c) the time evolution of the total density; (d) the time evolution of the total current.(e) the time evolution of the length.

Chapter 3

Model Expansion: Introduction of Side-Stepping

3.1 Side Stepping Without Tip Dynamics

In a model that better reflects the reality of biological systems, the motion of motor molecules along their microtubule tracks is not strictly one-dimensional. Depending on the processivity and properties of the member of the kinesin family in question, the motors have the ability to step laterally to an adjacent protofilament. Side-stepping occurs to avoid other microtubule associated proteins (including other kinesins), cross-linking factors, passenger proteins, or microtubule end-tracking proteins [69]. In the yeast kinesin-8 family (known in the biological literature as Kip3), motors have been empirically observed to demonstrate a left side-stepping directional bias, traveling helically around the microtubule as it moves forward. Interestingly, this left-hand bias is independent of forward velocity, but strongly depends on the average time required for the motor to take a step, and subsequently increases under limited ATP concentrations. This could be caused by a bifurcation in the stepping cycle of Kip3, in which the two-head bound conformation transitions to a one-head bound conformation, weakening interactions with the microtubule and allowing a greater range of lateral motion [69].

The helical motion of Kip3 as it moves toward the positive end of a microtubule can be visualized with a model produced from experimental results collected from [69], who labeled Kip3 motors with quantum dots and tracked their motion with highly inclined thin illumination microscopy (Fig. 3.1).

Side-stepping of motor molecules to adjacent protofilaments at unoccupied sites can be modeled with periodic boundary conditions. With a cylinder of 13 protofilaments, stepping from the thirteenth to what would be the fourteenth protofilament is the same as returning to the first protofilament. Consequently, I can adapt the one dimensional code to account for two-dimensional motion around the microtubule, assign periodic boundary conditions, and introduce left-hand bias in side-stepping probability that matches the biological literature.

A complication is introduced, however, by the fact that not all protofilaments in the microtubule polymerize or depolymerize at the same rate [20]. Rather, dynamic interactions between

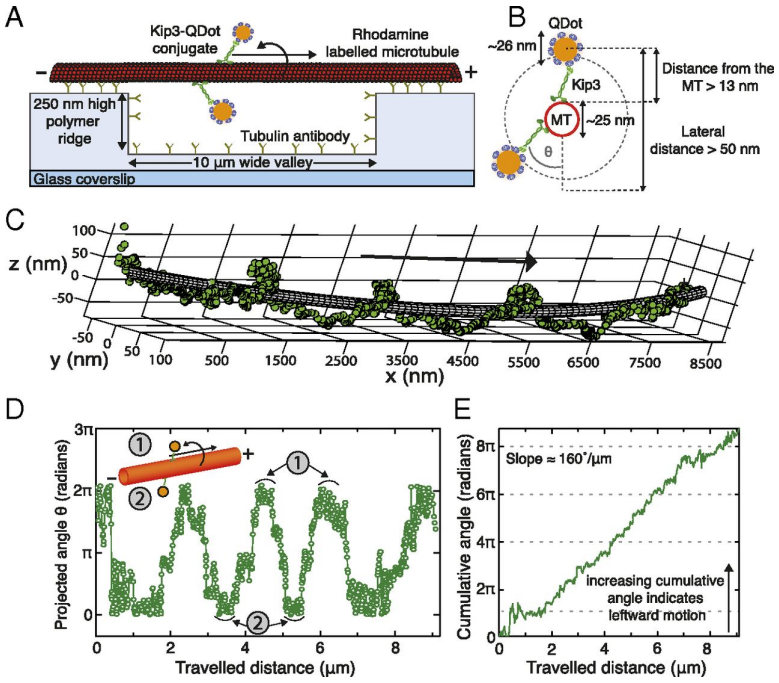


Figure 3.1: Helical trajectory of Kip3 exhibits a lefthanded bias. [69] provides a A) schematic representation of their experimental setup for monitoring motor dynamics B) annotated schematic detailing geometries of Kip3, quantum dots, and the microtubule. C) They report the 3D tracking result of the motion of quantum dot labeled Kip3 and D) project the angle between the motor and the microtubule axis, represented as theta in panel B. E) They also provide the cumulative change in angle as the motor travels toward the positive end of the microtubule. The increase in angle demonstrates an effective lefthand bias.

protofilament tips result in different rates of growth, shrinkage, catastrophe, and rescue. The modeling of side-stepping across thirteen protofilaments, each with unique tip dynamics, grows increasingly complicated. Consequently, I first address the side-stepping of motors in a simplified two-dimensional model that neglects lateral interactions between protofilament tips, allowing polymerization to be constant around the microtubule.

3.2 Tip Dynamics and Lateral Interactions

It is known that the combination of multiple protofilaments create different configurations that alter rates of polymerization and depolymerization at their tips, quantified by their association and dissociation constants, $\boldsymbol{k_{on}}$ and $\boldsymbol{k_{off}}$ [20].

Historically, it has been difficult to distinguish kinetic rates of each protofilament in a microtubule due to limited imaging technology and reliance on model approximations whose assumptions vary throughout the literature [20]. The standard technique, video-enhanced differential interference contrast microscopy, made possible observations for ($\boldsymbol{k_{on}}$ and $\boldsymbol{k_{off}}$) for the microtubule as a whole, but did not provide high enough granularity to specify differences in association and disso-

ciation constants for each protofilament. However, the emergence of optical tracking techniques, especially in combination with back focal plane interferometry, allows for the more precise measurements necessary to observe tip fluctuations and dynamics between protofilaments.

The new technique implemented by [21] utilized interferometric scattering micrscopy to measure incorporation of gold-labeled tubulin into the microtubule. This allowed for the direct measurment of $\boldsymbol{k_{on}}$ and $\boldsymbol{k_{off}}$ for different protofilaments. Interestingly, when measuring $\boldsymbol{k_{off}}$, the researchers recorded dwell times, or the average time required for the dissociation of a tubulin dimer. They observed two distinct groups: those with short dwell times, which the authors attribute to tubulins with one longitudinal bond, and those with longer dwell times, which they attribute to the combination of a longitudinal bond and a lateral bond with an adjacent protofilament. This theory is supported by an additional experiment in which a mutation to destabilize the lateral interface between tubulins was introduced, leading to a higher frequency of short dwell times. As a result, it is understood that interprotofilament interactions change the kinetic rates for polymerization and depolymerization.

Though the current model remains simplified and assumes that each protofilament polymerizes and depolymerizes at the same rate, a further expansion of the model could consider the effects of lateral interactions between protofilaments tips.

3.3 Two-Dimensional Model Code Structure

The two-dimensional track is defined by its transversal section, where each number represents the index of a track perpendicular to the lattice and the direction of motor movement.

By definition, this allows the total number of sites to be the initial length of the microtubule multiplied by the number of protofilament tracks, which is usually 13. Further, motors may only advance to adjacent sites either on its own track or a directly neighboring track. The track control parameters are the same constants as introduced in Table 2.1. Parameters for attachment, detachment, spontaneous polymerization, and motor-induced polymerization are the same as for the one dimensional model, with an important modification: we introduce a new parameter, $\boldsymbol{\Omega_J}$, as a probability of jumping laterally to a neighboring protofilament. For more details, the interested reader should refer to the source code in Appendix D.2.

3.4 Analysis of Side-Stepping Model Results

3.4.1 $\boldsymbol{\Omega_J = 0}$

I begin by presenting the two-dimensional case with a zero probability of side stepping, $\boldsymbol{\Omega_J = 0}$, such that particles walk only along the filament that they are initially injected on. I expect this to recover behavior from the one-dimensional case. Importantly, results for total density and current are summations over each of the individual 13 filaments as a function of time, while occupancy is reported as a snapshot for an individual filament as a function of position.

The relationship between density and current is modeled as $\boldsymbol{J = \rho(1-\rho)}$, which is reflected

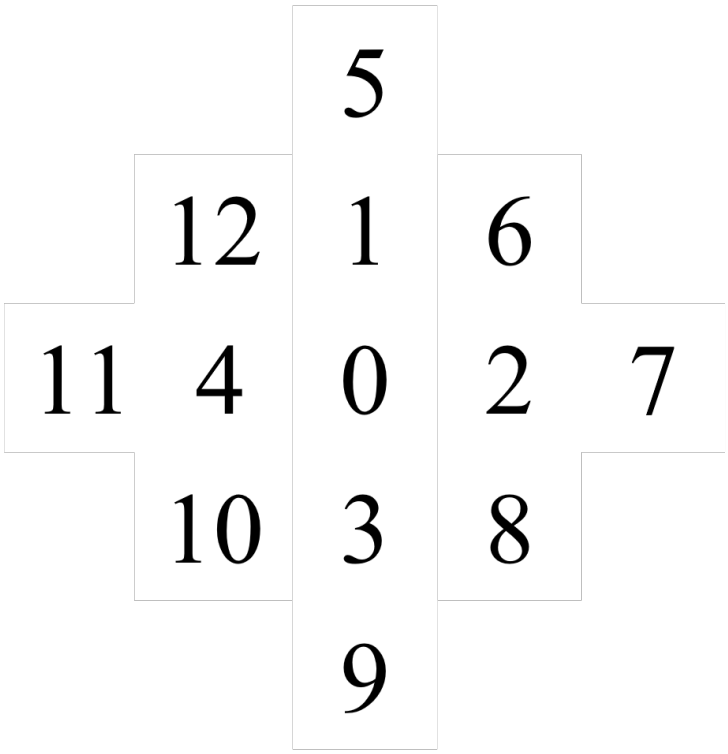


Figure 3.2: A transversal of the indices of each protofilament, as seen perpendicular to the axis of motor motion.

in the code after it reaches steady state. The model does not start recording measurements at the beginning of the process, explaining the initial transient behavior. Once traffic on the microtubule builds up to full density, the pile-up results, analogous to traffic jams on a highway. This is followed by detachment from the microtubule, after which steady state dynamics can be observed (following the expected behavior of $\boldsymbol{\rho(1-\rho)}$). I observe a current that extends below zero. This is due to the mean-field correction in the equation for current. In the one dimensional case, current was modeled as $\boldsymbol{J = \rho(C - \rho)}$, where C was the constant, $\boldsymbol{C = \gamma(1 - \rho_2)(\rho_1 - 1) + \delta\rho_1\rho_2 + 1}$ defined in Eq. 2.11. The negative current from the mean-field correction proves that the mean-field is not physically meaningful and fails in the side-stepping model, supporting the use of Monte Carlo simulations in further adaptations of the model.

In this iteration of the model, each protofilament has the same properties, including the same rate of polymerization, an absence of lateral dynamics between filaments, and the same parameters for motor dynamics (biologically, this is an oversimplification that may be addressed in a refined version of the model). Since occupancy is recorded individually for each filament, rather than as a sum across all 13 filaments of the cylinder, dynamics for individual filaments may be compared to ensure that their behavior is in fact similar. Consequently, I compare individual filament occupancy for three different filaments, each run with the same parameters. I conduct two tests, first comparing filaments for a system with no side-stepping (Fig 3.5), and a second with a non-zero $\boldsymbol{\Omega_J}$

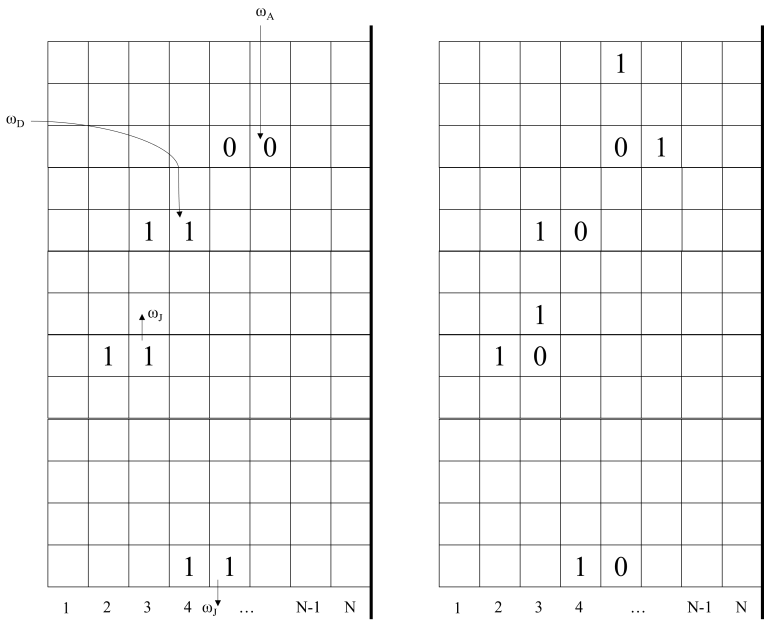
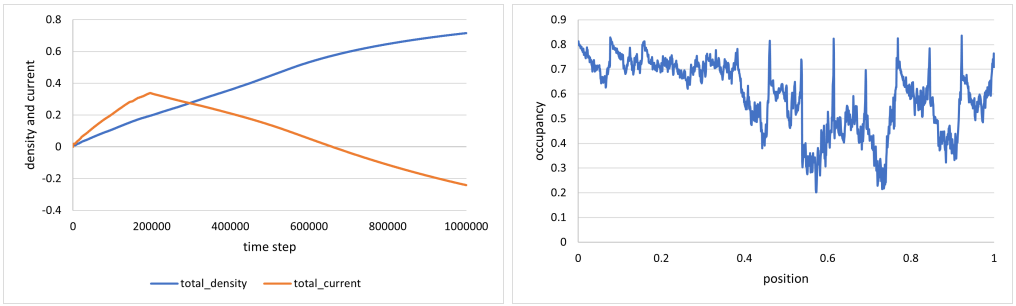


Figure 3.3: Schematic of Langmuir kinetic parameters for the two-dimensional system. As in the one dimensional model, $\boldsymbol{\omega_A}$ and $\boldsymbol{\omega_D}$ show the attachment and detachment of particles in bulk. New to the two-dimensional model, $\boldsymbol{\omega_J}$ shows the side stepping of particles that encounter an obstacle. It should be noted that in the two-dimensional lattice, the bottom lattice is periodic with the top lattice, allowing a particle to side step from what appears to be the bottom lattice back to the top.



(a) (b)

Figure 3.4: Two-dimensional model on a fixed length track with a zero probability of side stepping. a) Total density and current. b) Occupancy. Parameters for a) and b) : $\boldsymbol{\Omega_A = \Omega_D = \Omega_J = 0, \alpha = \beta = 0.5, \gamma = \delta = 0.1}$.

(Fig. 3.6).

I observe that for both the systems with and without side-stepping, occupancy on each filament follow the same patterns. This recovers expected results, as measuring a different filament should not change dynamics. Aside from the sanity check of comparing filaments, I also observe that without the capability of side-stepping, each filament maintains a higher average steady-state occupancy, 0.58, compared to the side-stepping counterpart, 0.39. This behavior occurs because

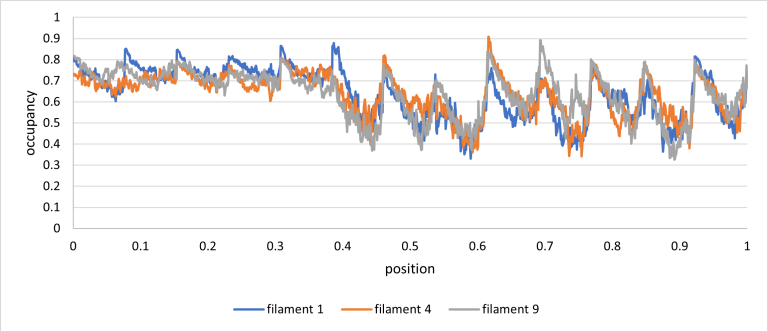


Figure 3.5: Comparison of occupancy of three different filaments (1, 4 and 9) for on a fixed-length track with an equal probability of Langmuir attachment and detachment, but a *zero* probability of side stepping. Parameters: $\boldsymbol{\Omega_A = \Omega_D = 0.5}$, $\boldsymbol{\Omega_J = 0}$, $\boldsymbol{\alpha = \beta = 0.5}$, $\boldsymbol{\gamma = \delta = 0.1}$.

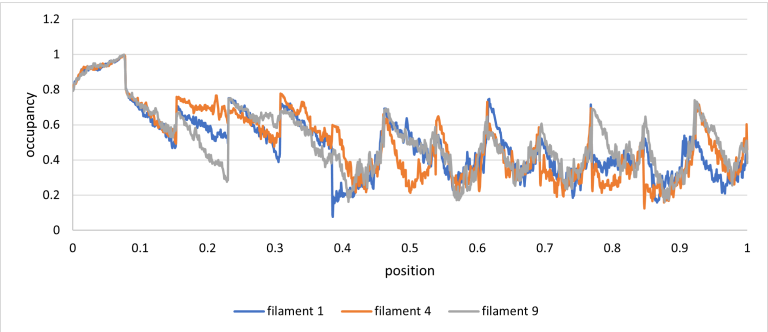


Figure 3.6: Comparison of occupancy of three different filaments (1, 4 and 9) for on a fixed-length track with an equal probability of Langmuir attachment and detachment, but a *non-zero* probability of side stepping. Parameters: $\boldsymbol{\Omega_A = 0.3}$, $\boldsymbol{\Omega_D = 0.7}$, $\boldsymbol{\Omega_J = 0.8}$, $\boldsymbol{\alpha = 0.3}$, $\boldsymbol{\beta = 0.7}$, $\boldsymbol{\gamma = \delta = 0.1}$.

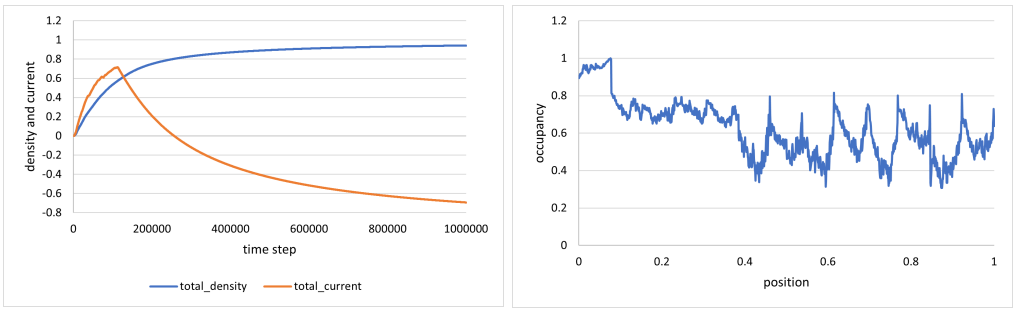
the inability of motors to side step causes obstructions and pile-ups when they encounter other motors. In contrast, the side-stepping system allows motors to move laterally and continue travel down the track, thus lowering the track's occupancy in steady state.

3.4.2 $\Omega_J \neq 0$

Next, I examine the case in which the rate of attachment, detachment, and side stepping are all equal and nonzero (Fig. 3.7).

As expected, I observe the same $\boldsymbol{J = \rho(1-\rho)}$ relationship for density and current, and a higher average steady state occupancy than for the non-zero case.

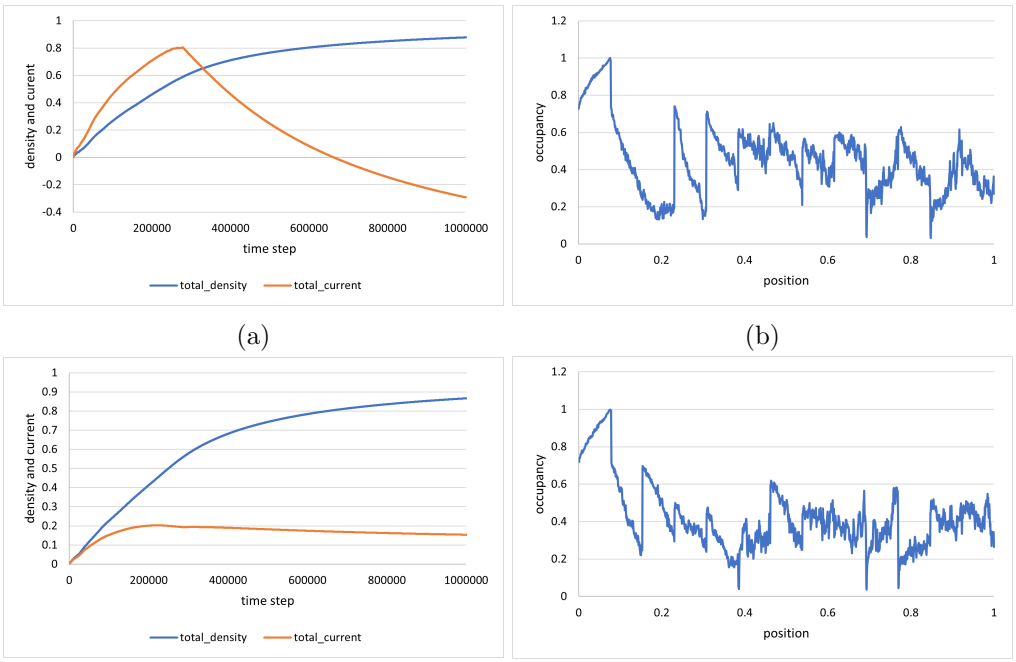
Now, I analyze the two-dimensional model with at two extreme cases. The first extreme case is a high rate of side stepping, $\boldsymbol{\Omega_J = 0.9}$, with low attachment and detachment rates, $\boldsymbol{\Omega_A = \Omega_D = 0.1}$, analyzed for both a fixed and variable length track (Fig. 3.8). The second extreme case is a low (but non-zero) rate of side stepping, $\boldsymbol{\Omega_J = 0.1}$, with high attachment and



(a) (b)

Figure 3.7: Two-dimensional model on a fixed length track with an equal probability of Langmuir attachment, detachment, and side stepping. a) Total density and current. b) Occupancy along the first filament. Parameters: $\boldsymbol{\Omega_A = \Omega_D = \Omega_J = 0.5, \alpha = \beta = 0.5, \gamma = \delta = 0.1}$.

detachment rates, $\boldsymbol{\Omega_A = \Omega_D = 0.9}$ (Fig. 3.9).



(a) (b)

(c) (d)

Figure 3.8: Two-dimensional model extreme case with a *high* rate of side stepping and *low* rates of Langmuir attachment and detachment, on tracks of fixed and variable lengths. a) Total density and current, fixed length. b) Occupancy, fixed length. c) Total density and current, variable length. d) Occupancy, variable length. Parameters: $\boldsymbol{\Omega_A = \Omega_D = 0.1, \Omega_J = 0.9, \alpha = 0.2, \beta = 0.8, \gamma = \delta = 0.1}$

Similar to the individual filament comparisons without and with side-stepping in Fig. 3.5 and 3.6, I observe that the presence of side-stepping reduces average occupancy in steady state.

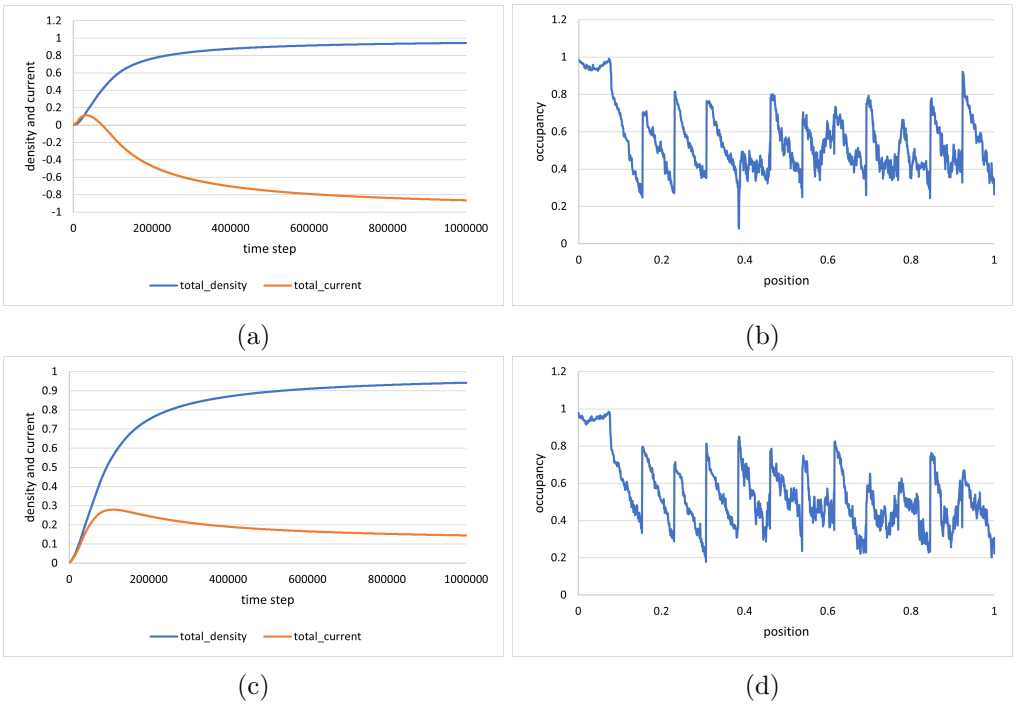


Figure 3.9: Second extreme case with a *low* rate of side stepping and *high* rates of Langmuir attachment and detachment, on tracks of fixed and variable lengths. a) Total density and current, fixed length. b) Occupancy, fixed length. c) Total density and current, variable length. d) Occupancy, variable length. Parameters: $\boldsymbol{\Omega_A = \Omega_D = 0.9, \Omega_J = 0.1, \alpha = 0.2, \beta = 0.8, \gamma = \delta = 0.1}$

This is analogous to cars switching lanes to spread their occupancy along a highway.

Though these examples represent only a small number of many possible combinations of parameters. It remains over-simplified compared to the complex biological reality of the non-equilibrium system, as it neglects interprotofilament dynamics, bidirectional motors, and other biochemical parameters that affect motor processivity. Despite its simplifications, the two-dimensional model is more nuanced than the original unidirectional, one-dimensional model and demonstrates the dynamic nature of motor molecule side-stepping. The iterative process of model refinement allows for the introduction of new complexities, accompanied by the rise of new theoretical and experimental questions. The high number of parameters necessitates the arbitrary nature of the selected cases and may overlook interesting model behavior in parameter combinations not analyzed. This supports the introduction of an algorithm for strategically analyzing all combinations of parameters, which will be discussed in Chapter 5.

This side-stepping model will provide statistical physicists and computational biophysicists alike the opportunity to explore new horizons in applied two-dimensional non-equilibrium systems, as well as guide experimental research specific to motor molecule and microtubule dynamics.

3.5 A Biological Application of the Coupled Harmonic Oscillator

I now take the opportunity to consider a unique application of a familiar model that accounts for some complexities overlooked in our side-stepping model. By considering the lateral, inter-protofilament interactions and the longitudinal, intraprotofilament interactions between tubulin dimers as springs, microtubule dynamics can be modeled as a coupled simple harmonic oscillator. In the two-dimensional model (and in the biological systems it represents), microtubule shrinkage following hydrolysis of the GTP cap is more complicated than simple one-dimensional depolymerization. Tubulin dimers with associated GDP exhibit a naturally bent conformation, storing potential energy in the form of mechanical strain when assembly into a microtubule forces each protofilament into a straight conformation. Therefore, during shrinkage, lateral interactions dissociate, releasing both chemical and mechanical energy then utilized to exert cellular forces, namely on kinetochores during mitosis [70]. To describe this phenomena, multiple models have been introduced. Most research has been conducted under the assumptions of the allosteric model, which states that GTP hydrolysis increases the mechanical strain in each protofilament. More recently developed based on observations from cryo-EM, the lattice model states that both GTP and GDP tubulin dimers form bent protofilaments, and hydrolysis affects the latitudinal and longitudinal bonds between dimers. In the lattice model, hydrolysis does not alter mechanical strain, but rather decreases the amount of strain the microtubule can tolerate before shrinkage occurs [70].

There is strong potential to adapt the model developed by [70] in which they used the allosteric model as a foundation, and incorporated the lattice model by introducing the dissociation of interprotofilament bonds as stochastic events with corresponding force-dependent rates. Alpha-beta tubulin dimers would represent the smallest subunit, depicted in the model as cylinders. Springs would connect each cylinder at their longitudinal and lateral points of interaction.

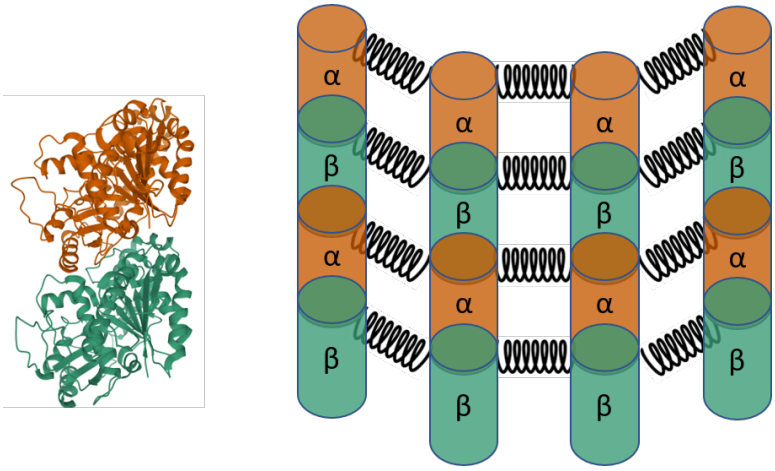


Figure 3.10: A diagram of the coupled harmonic oscillator model for lateral dynamics between alpha-beta tubulin dimers (adapted from [70]).

Integrating the current model with [70] could result in an effective two-dimensional model that rigorously accounts for interactions between protofilaments.

Chapter 4

A Three-State ASEP Model

In Section 1.3, it was explained that TASEP models particle motion on a one-dimensional lattice in one direction. This applies to unidirectional motor molecules, but is an incomplete model for bidirectional motor molecules with both anterograde (toward the beta-tubulin positively charged end) and retrograde (toward the alpha-tubulin negatively charged end) motion. Consequently, we turn to the *asymmetric exclusion principle (ASEP) model*, where the probability of a step toward the negative end of the microtubule is non-zero. Before discussing how ASEP driven dynamics alter the stochastic model presented in Chapter 2, I first introduce the molecular mechanics of bidirectional motors.

4.1 Bidirectionality: Introduction to Dynein

Similar to the kinesin superfamily, dynein is a family of motor proteins. The major difference between the two families is the direction of motion: dynein carries intracellular cargo from the periphery of the cell to the center, moving toward the negatively charged end, while kinesin carries its cargo to the cell periphery, generally moving toward the positively charged end.

Dynein's structure is different than that of kinesin. It consists of one long chain folded into many domains [71]. Of a core ring of six AAA+ domains, the AAA1 domain connects to a linker domain and a long tail, and the tail connects two to three dyneins together. Domain AAA4 is a long, non-rigid, coiled, anti-parallel stalk with a microtubule binding domain, and AAA5 is a strut domain that stabilizes the stalk. There are four binding sites for ATP, one of which is utilized in a chemo-mechanical cycle for motility, and the remaining three are believed to be regulatory [71].

The mechanism for dynein motility is similar but distinct from that of kinesin, and consists of a primary and power stroke. In the primary stroke, ATP binding to AAA1 causes release of the microtubule binding domain from the microtubule and changes the linker into a bent confirmation [72]. Following ATP hydrolysis, the microtubule binding domain rebinds to the microtubule and the linker unbends, creating a force-generating power stroke [72].

There are two models proposing the mechanism driving dynein's stepping direction toward the negative end. In the linker swing vector model, the motor domain pivots about the linker and the direction of stepping is the same as the direction of the linker swing [72]. In the asymmetric

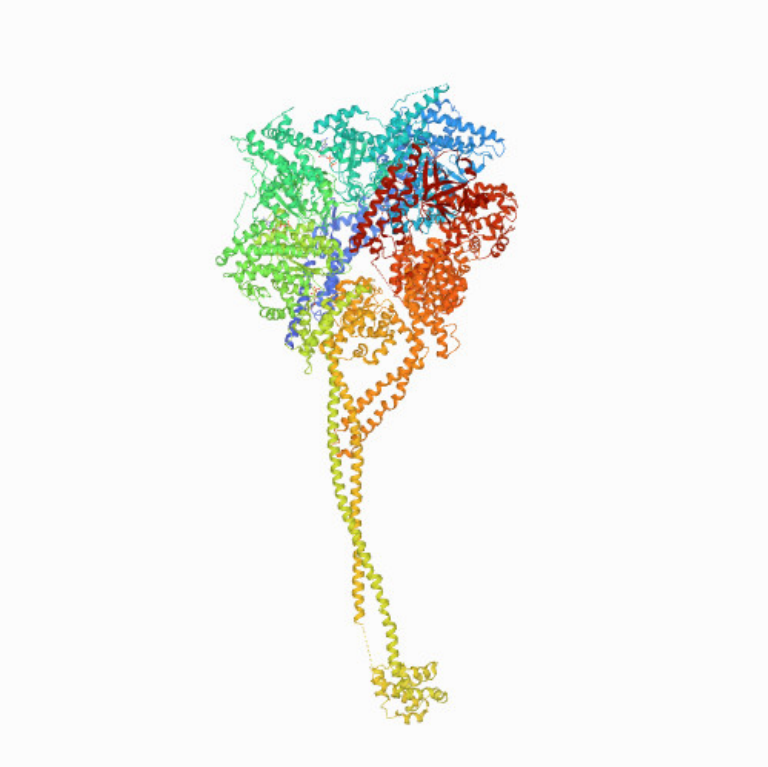


Figure 4.1: X-ray diffraction for a full-length dynein. The tail domain is not included in the crystal structure but would attach to the linker. Gray ligands are ATP [71].

release model, the faster release of dynein toward the negative end creates a net stepping bias [72]. The length and angle of the stalk are also critical in controlling the direction of motion toward the negative end and are conserved through all members of the dynein family [72].

4.2 Adaptation of the Model

Adapting the model to have a non-zero probability of retrograde motion deeply complicates the one-dimensional Monte Carlo code, and especially complicates the thirteen protofilament, side-stepping model. A one-dimensional model that included dynein would need to have different attachment and detachment rates at boundary conditions. If the motor has anterograde movement, it would attach at site $\boldsymbol{N}$ and move toward site $\boldsymbol{N_1}$ (as the lattice is labeled in Figure 2.1). Conversely, a motor with retrograde movement would attach at site $\boldsymbol{N_1}$ and move toward site $\boldsymbol{N}$. It is necessary to simplify the dynamics in the one-dimensional, bidirectional model when anterograde and retrograde motors confront one another. Like motorists avoiding a head-on collision on a road, one of the motors would necessarily detach prematurely from the lattice at rate $\boldsymbol{\omega_D}$, as governed by Langmuir dynamics. Which motor desorbs is a matter of probability in the oversimplified one-dimensional model.

In the thirteen protofilament adaptation of the model, instead of decreasing their processivity, a collision could be avoided through side stepping. Empirical evidence could be used to determine how motors interact with one another, and experimentally determined probabilities of biased left or right-handed side stepping in the face of an obstacle could be applied. Though the dynamics anterograde and retrograde motors becomes more biologically sensible in the two-dimensional model than its one-dimensional counterpart, the code becomes rapidly more complex. Consequently, I adapt the one-dimensional model to include bidirectionality, but defer expansion of bidirectionality in the two-dimensional version.

Empirical observations show that vesicles and other intracellular cargo have a small number of motor proteins tightly bound, including between one to five dynein and one to four kinesin [73]. Transport is driven by force-dependent kinetics, where in general, kinesins exert force toward the positive end of the microtubule and dynein toward the negative [73]. The bidirectional motion is disrupted with directional switching as different motor proteins attach to the microtubule track. It is common throughout biological literature to refer to this process as a tug-of-war [73, 74]. Other evidence supports a "paradox of codependence," in which inhibition of motors of either polarity decreases motitility in both directions [75].

Though the net motion of cargo is coordinated by regulatory proteins and is beyond the scope of this thesis, we now adapt the model rules presented in Section 2.1 to account for movement of both kinesin and dynein. Then, we present the results of the bidirectional model, and conclude the chapter with a loose analogy to the three-spin-state Potts model for equilibrium systems. Interested readers may refer to Appendix D.3 for the source code for both *pile-up* and *drive-by* interpretations, and Appendix C.2 for a mathematical analysis of the Potts model analogy.

4.3 Bidirectional Model Rules

As in the unidirectional model presented in Chapter 2, the negatively charged right end of the lattice is fixed (site $\boldsymbol{N}$) and the positively charged left end (site 1) can polymerize to extend, remain constant, or depolymerize to contract. Consistent with the previous model, lattice growth and shrinkage due to attachment or detachment at the leftmost site updates all site labels $\boldsymbol{i \rightarrow i \pm 1}$.

As before, kinesin traffic occurs from right to left, towards the positive end of the microtubule. Conversely, dynein traffic occurs from left to right, toward the negative end of the microtubule. To account for the two types of particles, I adapt the model to a three-state system in which lattice sites can be occupied by either kinesin, 1, dynein, 2, or unoccupied, 0. If two motors are neighbors, they block one another.

The system evolves according to the following rules (depicted in Fig 4.2) :
At sites 1 and 2:

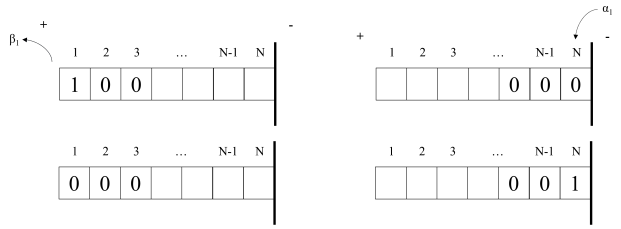
- $\mathbf{10 \rightarrow 00}$ with rate $\boldsymbol{\beta_1}$: kinesin particles leave the track with exit rate $\boldsymbol{\beta_1}$;
- $\mathbf{00 \rightarrow 20}$ with rate $\boldsymbol{\alpha_2}$: dynein particles enter the track with entrance rate $\boldsymbol{\alpha_2}$;
- $\mathbf{01 \rightarrow 10}$ with rate $\mathbf{1}$: kinesin particles move from right to left if the neighboring site is empty;
- $\mathbf{20 \rightarrow 02}$ with rate $\mathbf{1}$: dynein particles move from left to right if the neighboring site is empty;
- $\mathbf{00 \rightarrow 000}$ with rate $\boldsymbol{\gamma}$: the length of the track increases by one unit when the two first sites are empty; this is equivalent to spontaneous polymerization of a microtubule;
- $\mathbf{11 \rightarrow 0}$ or $\mathbf{12 \rightarrow 0}$ or $\mathbf{21 \rightarrow 0}$ or $\mathbf{22 \rightarrow 0}$ with rate $\boldsymbol{\delta}$: the length of the track decreases by one unit when two particles (occupying site 1 and 2) are present; at the same time, the particles leaves the track; this is equivalent to motor-induced depolymerization for a microtubule;

In bulk (sites $\boldsymbol{n_i = 3 ... N-1}$):

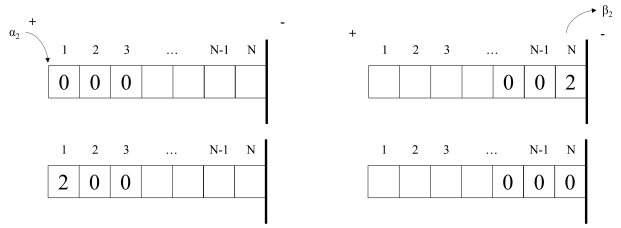
- $\mathbf{01 \rightarrow 10}$ with rate $\mathbf{1}$: kinesin particles move to the left with rate 1 as long as the neighbor to the left is empty;
- $\mathbf{20 \rightarrow 02}$ with rate $\mathbf{1}$: dynein particles move to the right with rate 1 as long as the neighbor to the left is empty;
- $\mathbf{1 \rightarrow 0}$ with rate $\boldsymbol{\omega_{D,1}}$: if site is occupied, remove kinesin particle with rate $\boldsymbol{\omega_{D,1}}$;
- $\mathbf{2 \rightarrow 0}$ with rate $\boldsymbol{\omega_{D,2}}$: if site is occupied, remove dynein particle with rate $\boldsymbol{\omega_{D,2}}$;
- $\mathbf{0 \rightarrow 1}$ with rate $\boldsymbol{\omega_{A,1}}$: if site is empty, add kinesin particle with rate $\boldsymbol{\omega_{A,1}}$;
- $\mathbf{0 \rightarrow 2}$ with rate $\boldsymbol{\omega_{A,2}}$: if site is empty, add kinesin particle with rate $\boldsymbol{\omega_{A,2}}$;

At site N:

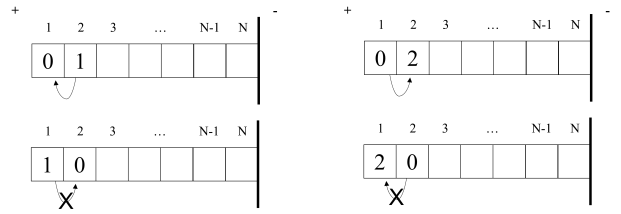
- **00 → 01** with rate $\boldsymbol{\alpha_1}$: kinesin particles enter the track with entrance rate $\boldsymbol{\alpha_1}$;
- **02 → 00** with rate $\boldsymbol{\beta_1}$: dynein particles leave the track with exit rate $\boldsymbol{\beta_2}$;
- **01 → 10** with rate **1**: diffusion of kinesin to the left with rate 1;
- **20 → 02** with rate **1**: diffusion of dynein to the right with rate 1;



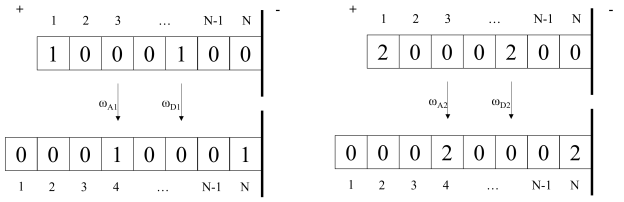
(a) Kinesin enters track with rate $\boldsymbol{\alpha_1}$ at site $\boldsymbol{N}$ and exits with rate $\boldsymbol{\beta_1}$ at positive end.



(b) Dynein enters track with rate $\boldsymbol{\alpha 2}$ at the positive end and exits with rate $\boldsymbol{\beta_2}$ at site $\boldsymbol{N}$.



(c) Kinesin diffuses to the next unoccupied site toward the positive end with rate 1, and dynein diffuses toward th negative end.



(d) Langmuir Kinetics: kinesin attachment with rate $\boldsymbol{\omega_{A1}}$, kinesin detachment with rate $\boldsymbol{\omega_{D1}}$, dynein attachment with rate $\boldsymbol{\omega_{A2}}$, and dynein detachment with rate $\boldsymbol{\omega_{D2}}$.

Figure 4.2: Three-state system model rules for kinesin and dynein

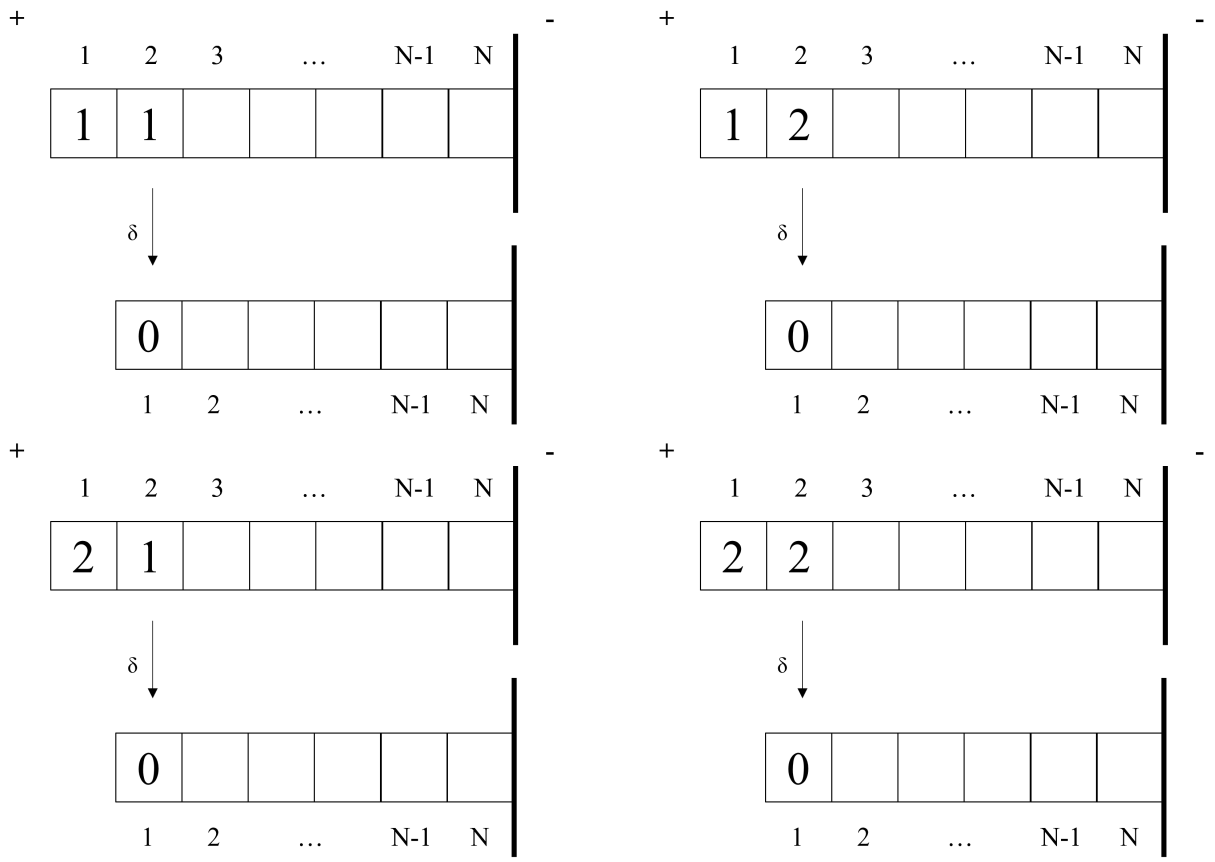


Figure 4.3: Motor-induced depolymerization for different combinations of kinesin and dynein at the positive end.

4.4 Bidirectional Model Code Structure

For each parameter introduced, the mean field method introduces more error into the model. For the bidirectional model, the necessity of averaging across so many parameters prevents the mean field approximation from accurately representing the system. I introduce the mean field method for two types of motors in Appendix C.1, but move forward with the Monte Carlo approach.

To model directional dynamics along a one-dimensional lattice. we define one current for each species, for both kinesin and dynein:

$$\langle \boldsymbol{n_i}(\mathbf{1} - \boldsymbol{n_{i+1}}) \rangle \tag{4.1}$$

where $\boldsymbol{n_i}$ represents the occupation site on the lattice.

I present two versions of the bidirectional code, distinguished by what happens when kinesin and dynein collide, or occupy adjacent sites on the lattice. In the *pile-up* interpretation, if the motor's neighboring site, $\boldsymbol{n_{i+1}}$ is occupied by either the same or opposite type of motor, then the motors block one another and a pile of motors builds. This will lead to a traffic jam and motor-

induced depolymerization. As described in Section 4.3 for model rules, this leads to the following detachments as a result of collisions:

- **11 → 0**
- **12 → 0**
- **12 → 0**
- **22 → 0**

The other version of the bidirectional will be referred to as the *drive-by* interpretation. As before, when motors of the *same* type occupy adjacent sites, the motor in $\boldsymbol{n_i}$ detaches. However, when motors of the *opposite* type occupy adjacent sites, I observe a *drive by* or hopping scenario analogous to lane switching in the two-dimensional model: if a kinesin occupies site $\boldsymbol{n_i}$ and a dynein occupies $\boldsymbol{n_{i+1}}$, then the motors will side step one another, where in the next time step the kinesin will occupy $\boldsymbol{n_{i+1}}$ and the dynein will occupy $\boldsymbol{n_i}$. This allows for the transition

- **12 ↔ 21**

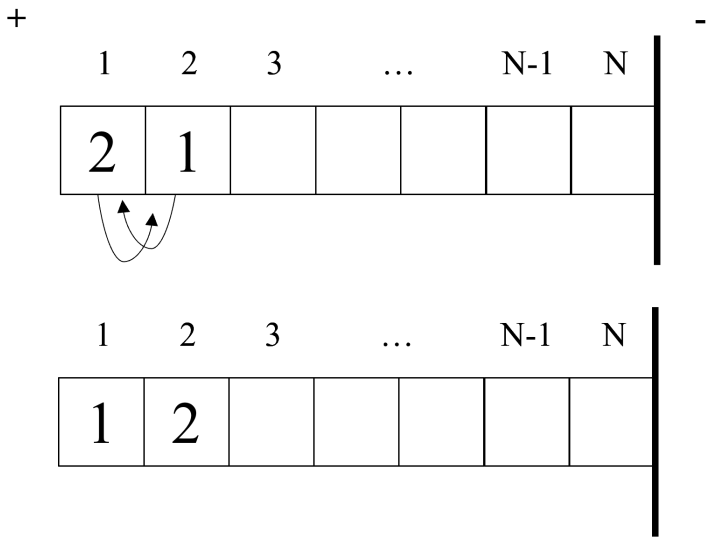


Figure 4.4: Schematic of the bidirectional model with the "drive by" interpretation, allowing a dynein and kinesin in adjacent sites to exchange positions. This is a different version of kinesin-dynein collisions than the "pile up" conditions presented in Fig.4.3.

The pile-up scenario presented in Fig. 4.3 is less biologically relevant, but leads to mathematical results of interest to the statistical physics community. The latter interpretation with the drive-by option, Fig 4.4, is more biologically relevant and will be easily adapted to the two-dimensional side stepping model presented in Chapter 3.

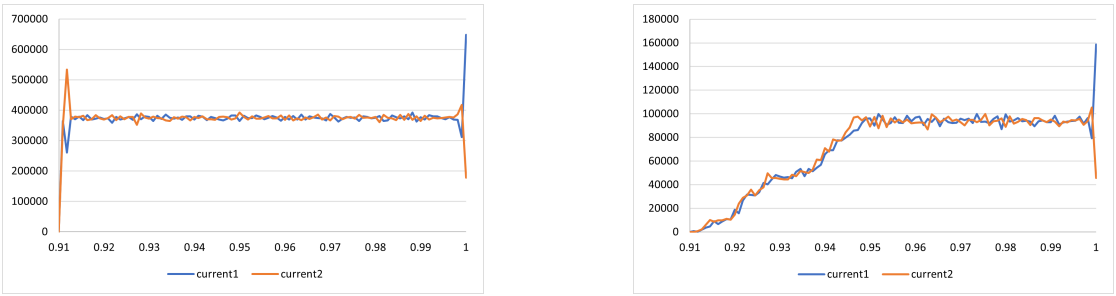
For each species, the measurement of the currents is observed separately, though interactions between species influence their dynamics. Notably, the total current is the sum of the two currents

in absolute value to account for the total motion of motors in either direction, not measuring the net motion if positive-end directed motion of kinesin wins the tug-of-war over dynein or vice versa.

I manipulate changes in initial length and attachment/detachment parameters of the track to observe changes in the development of steady state, as well as changes in the current, density and occupancy of the track. I analyze both two species interpretations, pile-up and drive-by, on tracks of fixed and variable lengths.

4.5 Analysis of Bidirectional Model Results

First, I demonstrate as that the Monte Carlo simulations can lead to a steady state dynamic equilibrium, following initial transient behavior. Steady state can be reached for both fixed (Fig. 4.5.a) and variable (Fig. 4.5.b) length tracks.



(a) (b)

Figure 4.5: Current of each type of particle and the total current (as an absolute value) on a track of a) fixed and b) variable length. c) The length over time for the steady state current. Parameters: $\boldsymbol{\Omega_{A1} = \Omega_{A2} = \Omega_{D1} = \Omega_{D2} = 0.5}$, $\boldsymbol{\alpha_1 = \beta_1 = \alpha_2 = \beta_2 = 0.5}$, $\boldsymbol{\gamma = \delta = 0.1}$

Next, I introduce a special case in the drive-by scenario in which the manipulation of $\boldsymbol{\gamma}$, the rate of spontaneous polymerization, can lead to the study microtubule catastrophe and phase transitions. As expected, a large $\boldsymbol{\gamma}$ leads to microtubule growth. A small $\boldsymbol{\gamma}$, by comparison, demonstrates rapid depolymerization. Using the Monte Carlo simulations, I can observe the point at which the phase transition occurs (Fig. 4.6). At lower and higher rates of spontaneous depolymerization, the track unravels or continues growth, but directly at the phase transition, the microtubule demonstrates treadmilling, also known as dynamic instability (Fig. 4.7).

Phase transitions are a subject of intense study in non-equilibrium statistical physics. Similarly, understanding the causes and disruptions to dynamic instability is of strong interest to biophysicists. Consequently, this case provides one of many examples of how this model is a useful foundation for future research. The parameters selected for this phase transition are an arbitrary combination that led to a familiar physical phenomena, and many other cases not examined could also lead to meaningful results. However, the high quantity of parameters makes in unfeasible to run all combinations of parameters manually. This calls for a methodical system to conduct a sensititivty analysis, the introduction of a training and test set, and implementation of machine learning algorithms.

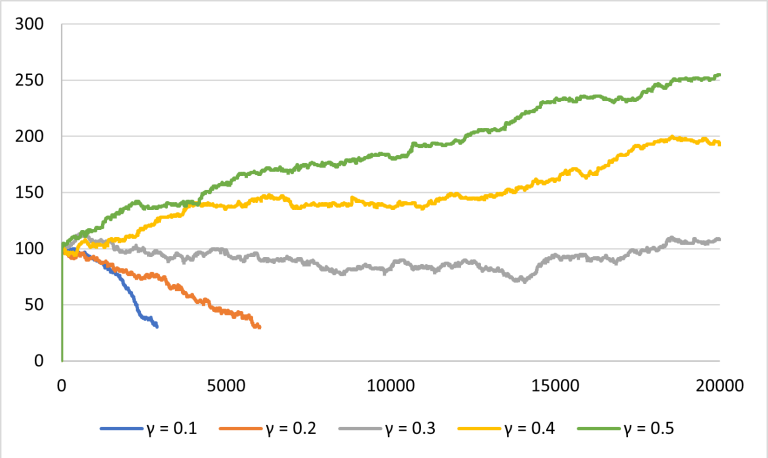


Figure 4.6: For the drive-by bidirectional scenario, I present Monte Carlo results for microtubule lengths. With rates of spontaneous polymerization below 0.3, the microtubule experiences catastrophe, while rates of spontaneous polymerization above 0.3 lead to growth. At exactly 0.3, the microtubule length stays roughly constant. For all manipulations of γ, all other parameters were held constant: $\boldsymbol{\Omega_{A1} = \Omega_{A2} = \Omega_{D1} = \Omega_{D2} = 1, \alpha_1 = 0.3, \beta_1 = 0.7, \alpha_2 = 0.7, \beta_2 = 0.3, \delta = 0.1}$.

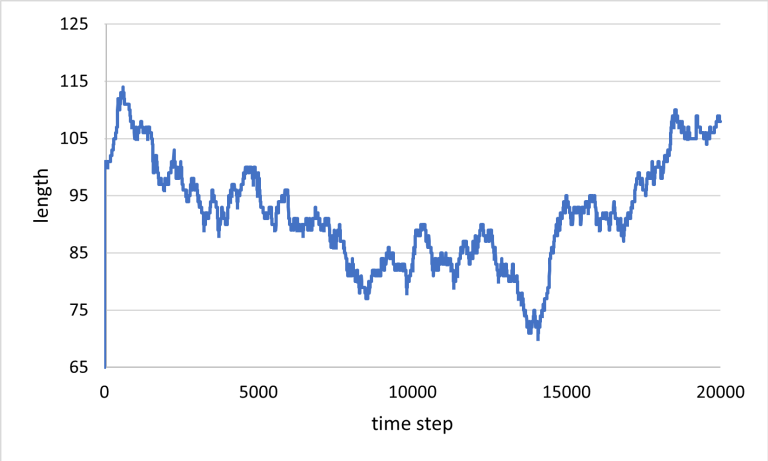


Figure 4.7: A close up of the gray line in Fig.4.6 At the phase transition, the microtubule demonstrates dynamic instability, depolymerizing and undergoing rescue to polymerize again.

4.6 An Analogy in Equilibrium Statistical Physics: The Potts model

In statistical physics, the Potts model represents spins on a crystalline lattice, in which spins take one of $\boldsymbol{q}$ possible values, distributed uniformly around a circle [76]. Interaction between spins occurs in a non-Abelian (non-commutative) way. Applications of the Potts model depend on the designated $\boldsymbol{q}$ value. In statistical physics, the Potts model is generally used to study phase transitions. When $\boldsymbol{q \leq 4}$, the model represents continuous transitions; such is the case in our biophysical application, as our three-state system can be represented as $\boldsymbol{q = 3}$.

It should be emphasized that the movement of kinesin and dynein with diffusion across the microtubule track is a non-equilibrium system, while the Potts model is strictly in equilibrium. While the analogy between the two is limited, the Potts model three-state system has relevance in the study of phase transitions in biophysics. For the interested reader, I detail the statistical physics behind the Potts Model analogy in Appendix C.2.

Chapter 5

New horizons: Machine Learning Applications for TASEP

5.1 Introduction to Machine Learning

Machine learning is a rapidly expanding computational tool, and emerging biophysical applications traverse many contexts. Recently, machine learning algorithms have been essential in formulating predictions for drug delivery systems, including nanocrystals, solid dispersion, and self-emulsifying systems [77]. Similarly, machine learning algorithms have been applied to optimize physiochemistry of lipid nanoparticle formulation, used as delivery systems for the mRNA vaccines that became prevalent after the COVID-19 pandemic [77]. Other applications include development of molecular dynamics simulations in protein engineering to predict functional fitness given molecular interactions, and modeling transcription elongation through DNA polymerase II dynamics [78, 79].

Given the essential role of machine learning in the future of computational biology, I present a brief review of machine learning approaches to totally asymmetric exclusion process and traffic modeling. In hopes of continued expansion of this project, I suggest how machine learning algorithms could be applied to modeling motor molecule dynamics, but leave it to the motivated reader to execute the algorithms.

Broadly, machine learning is the process through which computers develop independent pattern recognition to make predictions from data sets [80]. Without being explicitly programmed, machine learning allows computers to make predictions and adjustments from data sets, with applications throughout industry, academia, and beyond. There are four general categories of machine learning, depending on how much guidance the algorithm is given. In supervised, semi-supervised, and unsupervised learning, the algorithms extracts patterns from a labeled, partially labeled, or an unlabeled data set, respectively [80]. Within unsupervised learning, there are many strategies an algorithm can employ including (but not limited to) clustering, which identifies and groups similar data points, density estimation, which analyzes distributions, anomaly detection, which identifies outliers, and principal component analysis, which summarizes the set and makes predictions accordingly [80].

A prominent and emerging branch of machine learning is deep learning, in which an algorithm develops a neural network to determine the success of an output and reiterate to improve accuracy of results [80]. Deep learning consists of two steps: training, in which an algorithm interprets data and adapts based on feedback, and inference (testing), in which the neural networks make predictions using an unfamiliar data set.

A successful neural network is predicated on the best choice of algorithm based on the data set, as well as the complexity of the training data set. The accuracy of a neural net is determined by minimization of the risk function, $\boldsymbol{R(\Theta)}$, where $\boldsymbol{\Theta}$ represents the model ($\boldsymbol{\Theta^*}$ represents the optimal model), the set $\boldsymbol{z = z_1 ... z_l}$ represents the data set, $\boldsymbol{F(z)}$ represents the distribution, and $\boldsymbol{Q(z, \Theta)}$ represents the cost function:

$$\boldsymbol{R(\Theta) = \int Q(z, \Theta) dF(z) \Theta \in A} \tag{5.1}$$

A fundamental understanding of the mechanics of deep learning allows for the best choice of network architecture, effective models, and interdisciplinary applications.

5.2 Applications of Machine Learning to Traffic, TASEP and Motor Molecule Dynamics

Statistical physics literature refers to hydrodynamical equations as those that represent the flow of traffic:

$$\boldsymbol{q(x, t) = \rho(x, t) V(x, t)} \tag{5.2}$$

where $\boldsymbol{q}$ represents flow, $\boldsymbol{\rho}$ represents density, and $\boldsymbol{V}$ represents speed.

Widespread through the statistical physics community, hydrodynamical models often assume the fundamental hypothesis, also known as the fundamental diagram relationship, that flow is exclusively dependent on density [81]. This assumption is the foundation of the Lighthill-Whitham-Richards (LWR) equation, a first-order model of traffic dynamics [81]:

$$\boldsymbol{\frac{\partial \rho(x, t)}{\partial t} + \frac{\partial q[\rho(x, t)]}{\partial x} = 0} \tag{5.3}$$

Sacrificing accuracy for simplicity, the LWR equation falls short in modeling many of the empirical phenomena of traffic flow. As a result, many models often incorporate a second order equation to govern dynamics:

$$\boldsymbol{\frac{\partial V(x, t)}{\partial t} + V(x, t) \frac{\partial V(x, t)}{\partial t} = -\nu \frac{1}{\rho(x, t)} \frac{\partial \rho(x, t)}{\partial x} + \frac{1}{\tau} (V_e[\rho(x, t)] - V(x, t))} \tag{5.4}$$

with $\boldsymbol{\nu}$ serving as a viscosity coefficient and $\boldsymbol{\tau}$ representing a relaxation time to settle into equilibrium speed, $\boldsymbol{V_e[\rho]}$ [81].

At the microscopic scale in the continuous limit, hydronamic equations lead to "car following" models, in which the motion of individual particles (in the case of highways, vehicles, in the case of microtubules, motor molecules) abide by a kinematic rule that is a function of distance and speed

relative to an adjacent particle [81].

This leads to the introduction of the Nagel Schreckenberg (NaSch) model for traffic, with strong applications to totally asymmetric exclusion principle. Coined in 1992 by Nagel and Schreckenberg as they used Monte Carlo simulations to simulate freeway traffic, the NaSch model executes four rules for updating the position of each particle. These rules lead to a density-dependent transition from laminaar flow to "start-stop waves," as observed empirically [82]. The four rules, executed in parallel to update the position of each particle, are as follows:

- acceleration: $\boldsymbol{v_n \rightarrow min(v_n + 1, v_{max})}$
- deceleration to avoid collision: $\boldsymbol{v_n \rightarrow min(v_n, d_n)}$
- randomization: $\boldsymbol{v_n \rightarrow max(v_n - 1, 0)}$ at probability P
- particle motion: $\boldsymbol{x_n(t + 1) \rightarrow xn + vn}$

  where

$$\boldsymbol{d_n(t) = x_{n+1} - x_n - 1} \tag{5.5}$$

represents the current empty sites in front of the $\boldsymbol{n}$th particle and $\boldsymbol{x_n}$ represents the position of the particle at time $\boldsymbol{t}$ [82]. Some literature refers to the "start-stop waves" as "phantom jams," defined as spontaneous and symmetrical breaking among identical particles [81]. As in the original context of vehicles on the highway, the NaSch model rules can be applied to the TASEP of motor molecules traveling along a microtubule to recreate intracellular traffic patterns of density-dependent and phantom jams.

Utilizing these hydrodynamical equations as the cost function in a machine learning algorithm opens the door to complex biophysical models able to be trained by empirical data. Though the execution of such algorithms is outside the scope of this thesis, machine learning is arguably the future of computational biophysical modeling.

Chapter 6

Discussion, Conclusions, and Future Direction

6.1 Conclusions

I presented a versatile traffic model on dynamic tracks motivated by the motion of molecular motors on microtubules of variable length. The model captures the interplay between polymerization and depolymerization of the microtubule, consistent with the microtubule instability observed in lab experiments [83, 84]. To model motor molecule dynamics, we utilized TASEP and Langmuir Kinetics as the foundation for a mean-field analysis, and compared analytical results to Monte Carlo simulations when applicable. I began with a unidirectional, one-dimensional model and then expanded to increasing levels of complexity, including motion of motors in two-dimensions and bidirectional toward the positive and negative ends of the microtubule.

As with any model, there are benefits and weaknesses to its simplicity. Though many complexities from both a biologists' and a physicists' perspective are not accounted for, the simple model structure lends to its generality and versatility.

The first iteration of the model with unidirectional motion on a one-dimensional lattice can be adapted to a particular motor with motor-induced depolymerization capabilities (such as kinesin-8 or kinesin-13), but it is also general enough to be applied to other physical systems. The general nature of this model also contributes to widespread applicability and purpose in the context of interdisciplinary research. There is value in studying this model as an abstract non-equilibrium statistical physics system that can shed light on the new features of a traffic model defined on dynamic tracks.

As introduced in Section 1.3.1, many such applications can be made in biophysics. For example, TASEP-type models are useful in modeling translation, the process in which transfer RNAs convert messenger RNA transcripts into a polypeptide chain. Ribosomes, complexes of ribosomal RNA, move unidirectionally from N-terminus to C-terminus (5' to 3') along the one-dimensional mRNA lattice [44]. Each site on the lattice is represented by a three-base codon, which codes for an amino acid. In eukaryotes, transcription initiates when transcription factors recruit a ribosome to bind to a promoter, a sequence upstream of a gene, and the ribosome begins moving one codon

at a time downstream (toward the 3' end) in search of a start codon. This triggers the beginning of elongation, in which a new amino acid is added to the polypeptide chain at each codon, that is, at each site of the lattice. As it moves, the ribosome attaches to one site at a time but blocks the adjacent sites from attachment by another ribosome. In a reading frame of N codons, the addition of the last amino acid in the polypeptide occurs at site $\boldsymbol{N-1}$, as the $\boldsymbol{N^{th}}$ site is designated by the stop codon, which does not code for an amino acid and causes termination. Notably, the ribosome does not attach and detach freely as it moves down the lattice, indicating that translation is an application of TASEP without Langmuir kinetics. A similar application can be made to the movement of RNA polymerase (RNAP) and the displacement of histones during transcription [51].

From a theoretical point of view, the model is rich in its dynamics and can lead to more in-depth studies of its special cases or extensions. I used the mean field and hydrodynamics approach to study it analytically, and compared our results to the Monte Carlo simulations with mixed success. Specifically, in the unidirectional, one-dimensional model, the high correlation effects between the sites at the tip of the microtubule that trigger the change in length caused the mean-field approach worked for only a few cases, when the microtubule can be easily approximated as being constant in length. For other sets of parameters, the mean-field theory fails, and other analysis methods need to be explored. The first step in addressing the shortcomings of the mean-field theory is to include particle correlations into the equations, starting with two and three point correlations for the boundary sites. Another avenue of study may be a kinetic approach to small-size systems by solving numerically the governing master equation.

Though remaining highly simplified, the Monte Carlo simulations of the two-dimensional, unidirectional model increased complexity by incorporating motor side-stepping between protofilaments of a microtubule. These results recreated traffic dynamics familiar to a motorist on a highway: occupancy in steady state was lower in systems with higher rates of side-stepping, analogous to a higher rate of changing lanes on a higway. The next step in addressing the shortcomings of the two-dimensional model is to account for interprotofilament tip dynamics, where each protofilament in the mirotubule polymerizes and depolymerizes at slightly different rates and affects neighboring protofilament dynamics. I expect track dynamics to have an effect on the build-up, and likely the detachment, of molecular motors.

The final expansion of the model introduced bidirectionality. The high number of parameters caused too many approximations to make the mean-field approach meaningful, leading us to rely exclusively on Monte Carlo simulations. I analyzed two interpretations of bidirectional systems, the *pile-up* and *drive-by* scenarios, of which the latter led to more biologically relevant results. By varying rates of spontaneous polymerization of a track of variable length, I isolated a phase transition, the point at which the track experienced catastrophe and rapidly dissociated. Further study of the parameters that result in phase transitions in the bidirectional model are of interest to non-equilibrium physicists as well as biophysicists whose work applies to medicine, as understanding the mechanisms and mathematical modeling behind the malfunctions of intracellular transport is a critical step in treating neurodegenerative and other diseases.

As is the direction of biophysical computational modeling, I discussed the potential for application of TASEP to machine learning. Though I leave it to the motivated reader to use our

model to ask questions and continue research, I predict that the use of machine learning in traffic modeling will be an effective computational approach in the future.

With a few exceptions (for example [61], [62], and others), traditional TASEP studies for motion of molecular motors on microtubules consider the track as fixed in length. The value of this thesis is in considering three competing processes that occur at the biological level: (1) the track variable in length (incorporates both growth and shrinkage); (2) TASEP dynamics; (3) Langmuir dynamics. I hope this study can be useful for biologists and biophysicists working in the molecular motors field. I also hope that this is an interesting enough non-equilibrium statistical physics model worthy of further studies and extensions to other physical systems.

Appendix A

TASEP Master Equation

In section 1.3, I introduce TASEP as a stochasitc model for the one-dimensional motion of particles on a one-dimensional lattice. Here, I present the master equation for the probability of of finding the system in configuration $\{n_i\}$ after t time steps:

$$P(\{n_i\}, t+1) - P(\{n_i\}, t) = \sum_{\{n_i\}} \mathcal{L}(\{n_i^{'}\}, \{n_i\}) P(\{n_i\}, t) \tag{A.1}$$

where the Liovillian (like the quantum Hamiltonian) can be written as

$$\mathcal{L}(\{n_i^i\}, \{n_i\}) = W(\{n_i^i\}, \{n_i\}) - \delta(\{n_i^i\}, \{n_i\}) \sum_{\{n_i^{''}\}} W(\{n_i^i\}, \{n_i^{''}\}) [44] \tag{A.2}$$

Here, δ is the Kroncecker delta and $W(\{n_i^{'}\}, \{n_i\})$ represents the transition probability between configurations:

$$\begin{aligned} \frac{1}{L+1}[\alpha(1-n_1)\delta(n_1^{'}, n_1+1) \prod_{j>1} \delta(n_j^{'}, n_j) + \\ \sum_{k=1}^{L-1} \gamma n_k (1-n_{k+1}) \delta(n_k^{'}, n_k - 1) \delta(n_{k+1}^{'}, n_{k+1}+1) \prod_{j \neq k, k+1} \delta(n_j^{'}, n_j) \\ + \beta n_L \delta(n_L^{'}, n_L - 1) \prod_{j<L} \delta(n_j^{'}, n_j)] \end{aligned} \tag{A.3}$$

Appendix B

Phase Transitions

Also in section 1.3, I begin to introduce the role of TASEP and ASEP as an approach to phase diagrams. Non-equilibrium physicists define density and current for three phases: low-density, high-density, and a maximal current.

For total current, $\boldsymbol{J}$:

$$\boldsymbol{J} = \begin{cases} \frac{\mathbf{1}}{\mathbf{4}} & \boldsymbol{\alpha} \geq \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\beta} \geq \frac{\mathbf{1}}{\mathbf{2}} \text{ (MC phase)} \\ \boldsymbol{\alpha}(\mathbf{1}-\boldsymbol{\alpha}) & \boldsymbol{\alpha} < \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\beta} > \boldsymbol{\alpha} \text{ (LD phase)} \\ \boldsymbol{\beta}(\mathbf{1}-\boldsymbol{\beta}) & \boldsymbol{\beta} < \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\alpha} > \boldsymbol{\beta} \text{ (HD phase)} \end{cases} \tag{B.1}$$

For total density, $\boldsymbol{\rho}$:

$$\boldsymbol{\rho} = \begin{cases} \frac{\mathbf{1}}{\mathbf{2}} & \boldsymbol{\alpha} \geq \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\beta} \geq \frac{\mathbf{1}}{\mathbf{2}} \text{ (MC phase)} \\ \boldsymbol{\alpha} & \boldsymbol{\alpha} < \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\beta} > \boldsymbol{\alpha} \text{ (LD phase)} \\ \mathbf{1}-\boldsymbol{\beta} & \boldsymbol{\beta} < \frac{\mathbf{1}}{\mathbf{2}}\boldsymbol{\alpha} > \boldsymbol{\beta} \text{ (HD phase)} \\ \boldsymbol{\alpha} + (\mathbf{1}-\mathbf{2}\boldsymbol{\alpha})\boldsymbol{x} & \boldsymbol{\alpha} = \boldsymbol{\beta} < \frac{\mathbf{1}}{\mathbf{2}} \end{cases} \tag{B.2}$$

where the position is denoted as

$$\boldsymbol{x} = \frac{\boldsymbol{i}}{\boldsymbol{N}} \tag{B.3}$$

and $\boldsymbol{i}$ labels the site.

On the one-dimensional lattice $\boldsymbol{S}$, the state $\boldsymbol{n}$ of the system is 0 when the site $\boldsymbol{x}$ in $\boldsymbol{S}$ is unoccupied, and 1 when site $\boldsymbol{x}$ is occupied. Growth processes can then be described using a height function, $\boldsymbol{h_t}$, (time dependent based on the current of the particles. The height function is constructed in pieces: if $\boldsymbol{n_t(x) = 1}$, representing an occupied site, then $\boldsymbol{h_t(x) \in [x, x+1]}$ increases with a slope of 1. If $\boldsymbol{n_t(x) = 0}$, representing a vacant site, then $\boldsymbol{h_t(x) \in [x, x+1]}$ has a slope of -1.

In the continuous limit, exclusion processes are described by the Kadar-Parisi-Zhang equation:

$$\frac{\boldsymbol{\partial h}}{\boldsymbol{\partial t}} = \boldsymbol{\nu}\frac{\boldsymbol{\partial^2 h}}{\boldsymbol{\partial x^2}} + \boldsymbol{\Lambda}\left(\frac{\boldsymbol{\partial h}}{\boldsymbol{\partial x}}\right)^{\mathbf{2}} + \boldsymbol{\omega} \tag{B.4}$$

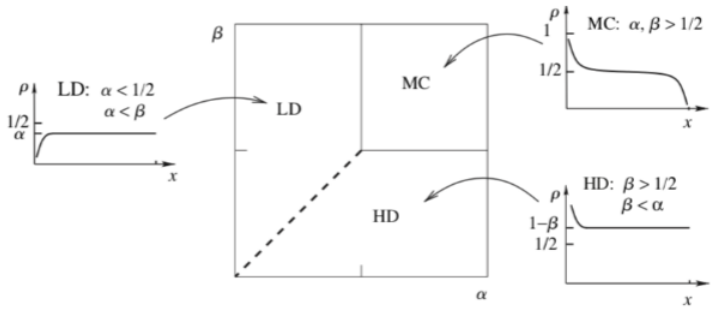


Figure B.1: Central: a schematic phase diagram of ASEP, where LD represents low-density phase, HD represents the high-density phase, and MC represents the maximal-current. Subplots depict changes in density by coordinate [45]

where the second order partial derivative accounts for diffusion, the first order partial derivative accounts for growth, and $\boldsymbol{\omega}$ accounts for noise [85, 86].

In the hydrodynamic limit, TASEP is governed by the noisy Burgers equation:

$$\boldsymbol{u(x,t)} = \frac{\boldsymbol{\partial h}}{\boldsymbol{\partial x}} \tag{B.5}$$

[3, 86].

Appendix C

Mean Field Method for Bidirectional Motors

C.1 Three-State Mean Field Equations

In order to be able to write the differential mean field equations, I will introduce two occupation numbers for the two types of particles, $\boldsymbol{n_i = 0}$ for empty and $\boldsymbol{n_i = 1}$ for occupied (kinesin) and $\boldsymbol{p_i = 0}$ for empty and $\boldsymbol{p_i = 1}$ for occupied (dynein). Though there are two versions of the bidirectional model, one where colliding kinesin and dynein *pile up* and one where they *drive by* one another, here I present equations for the pile-up version only.

I present below the evolution equations for the site occupation numbers. Even though all the equations are coupled, for clarity, I will write separate sets of equations for the two types of models. I first start with the boundary sites 1, 2, 3 and $\boldsymbol{N}$ the equations are:

$$\frac{\boldsymbol{dn_1}}{\boldsymbol{dt}} = -\boldsymbol{\beta_1 n_1(1-p_1)(1-n_2)(1-p_2) + n_2(1-p_2)(1-n_1)(1-p_1) - \delta(n_1 n_2 + p_1 p_2 + n_1 p_2 + p_1 n_2)} \tag{C.1}$$

$$\frac{\boldsymbol{dn_2}}{\boldsymbol{dt}} = \boldsymbol{n_3(1-n_2)(1-p_3)(1-p_2) - n_2(1-n_1)(1-p_1)(1-p_2) - \delta(n_1 n_2 + p_1 p_2 + n_1 p_2 + p_1 n_2) + \delta n_1 n_2 n_3 p_1 p_2 p_3)} \tag{C.2}$$

$$\frac{\boldsymbol{dn_3}}{\boldsymbol{dt}} = \boldsymbol{n_4(1-p_4)(1-n_3)(1-p_3) - n_3(1-p_3)(1-n_2)(1-p_2) - \omega_{D,1} n_3(1-p_3) + \omega_{A,1}(1-n_3)(1-p_3) - \gamma(1-n_1)(1-p_1)(1-n_2)(1-p_3) + \delta(n_1 p_1 + n_2 p_2 + n_1 p_2 + p_1 n_2)(n_4 p_4 - n_3 p_3)} \tag{C.3}$$

$$\frac{dn_N}{dt} = \alpha_1(1 - n_N)(1 - p_N) - n_N(1 - p_N)(1 - n_{N-1})(1 - p_{N-1}) \quad \text{(C.4)}$$

For the dynein motors, the equations for the boundary sites are:

$$\frac{dp_1}{dt} = +\alpha_2(1-n_1)(1-p_1)(1-n_2)(1-p_2) - p_1(1-p_2)(1-n_1)(1-p_1) - \delta(n_1n_2 + p_1p_2 + \\ + n_1p_2 + p_1n_2) \quad \text{(C.5)}$$

$$\frac{dp_2}{dt} = p_1(1-n_1)(1-n_2)(1-p_2) - p_2(1-n_1)(1-p_1)(1-n_2) - \delta(n_1n_2 + p_1p_2 + n_1p_2 + p_1n_2) + \\ \delta n_1n_2n_3p_1p_2p_3) \quad \text{(C.6)}$$

$$\frac{dp_3}{dt} = p_2(1 - n_2)(1 - n_3)(1 - p_3) - p_3(1 - n_3)(1 - n_4)(1 - p_4) - \omega_{D,2}p_3(1 - n_3) \\ +\omega_{A,2}(1-p_3)(1-n_3) - \gamma(1-n_1)(1-p_1)(1-n_2)(1-p_3) + \delta(n_1p_1 + n_2p_2 + n_1p_2 + p_1n_2)(n_4p_4 - n_3p_3)) \quad \text{(C.7)}$$

$$\frac{dp_N}{dt} = -\beta_2(1 - n_N)(1 - p_N) - p_{N-1}(1 - n_N)(1 - p_N)(1 - n_{N-1}) \quad \text{(C.8)}$$

And for the bulk sites, $n_i = 4...N-1$ and $p_i = 4...N-1$:

$$\frac{dn_i}{dt} = n_{i+1}(1 - n_i)(1 - p_{i+1})(1 - p_i) - n_i(1 - p_i)(1 - n_{i-1})(1 - p_{i-1}) + \\ \gamma(1 - n_1)(1 - n_2)(1 - p_1)(1 - p_2)(n_{i-1}p_{i-1} - n_ip_i) + \\ \delta n_1n_2p_1p_2((n_{i+1}p_{i+1} - n_ip_i) - \omega_{D,1}n_i(1 - p_i) + \omega_{A,1}(1 - n_i)(1 - p_i) \quad \text{(C.9)}$$

$$\frac{dp_i}{dt} = p_{i-1}(1 - p_i)(1 - n_{i-1})(1 - n_i) - p_i(1 - n_i)(1 - n_{i+1})(1 - p_{i+1}) + \\ \gamma(1 - p_1)(1 - p_2)(1 - n_1)(1 - n_2)(n_{i-1}p_{i-1} - n_ip_i) + \\ \delta n_1n_2p_1p_2((n_{i+1}p_{i+1} - n_ip_i) - \omega_{D,2}p_i(1 - n_i) + \omega_{A,2}(1 - p_i)(1 - n_i) \quad \text{(C.10)}$$

C.2 Potts Model Analogy

The Potts model is an equilibrium system used to investigate phase transitions. Distribution of q spins is modeled as

$$\Theta_n = \frac{2\Pi n}{q} n = 0, 1, ..., q-1 \tag{C.11}$$

In the vector Potts model, we introduce the interaction Hamiltonian:

$$H_c = J_c \sum_{i,j} \cos(\Theta_{s_i} - \Theta_{s_j}) \tag{C.12}$$

for nearest neighbor pairs, i and j over all lattice sites, and site "colors", s_i assume values in $1, ...q$. J_c represents the coupling constant, indicative of the interaction strength between spins. In further iterations of this applications of the Potts model, J_c could be represented with empirical data quantifying the extent of interactions between kinesin and dynein, analogous to the strength of the interaction between spin states 1 and 2.

In the (simpler) standard Potts model, the Hamiltonian is represented by

$$H_p = -J_p \sum i, j\delta(s_i, sj) \tag{C.13}$$

where $\delta(s_i, sj)$ is the Kronecker delta, collapsing to one when $s_i = s_j$ and zero everywhere else. Notably, the standard and vector Potts models are made equilvalent when

$$J_p = \frac{3}{2} J_c \tag{C.14}$$

The Potts model is generalized by the Fortuin-Kasteleyn representation, also referred to as the random cluster model, which unifies the Potts model with Bernoulli percolation ($q = 1$) and the Ising model ($q = 2$). The random cluster model is used to study random combinatorial structures, phase transitions, ferromagnetism, and (as we demonstrate here) biophysical systems. Importantly, the generalizations made by the random cluster model has aided in the development of Markov chain Monte Carlo approaches to model the Potts model for small q, applicable to our three-state system. Markov chain Monte Carlo approaches are detailed in Chapter 5.

Appendix D

Code Screenshots

For the sake of reproducibility, I provide screenshots of the python code utilized to make each model.

D.1 Unidirectional Stochastic Model in One-Dimension

D.2 Side-Stepping Model

D.3 Three-State ASEP Model

D.3.1 Interpretation One: *pile-up*

D.3.2 Interpretation Two: *drive-by*

```python
S# -*- coding: utf-8 -*-
"""
Created on Sun Jul  5 09:14:11 2020
@author: Laurențiu Stoleriu

reverse walk R-to-L with new rules
"""

import random
import numpy as np
import matplotlib.pyplot as plt
import time
start_time = time.time()

"""////////////////////////////////////////////////////////////////////////
// PARAMETERS  FOR  TRACK  CONTROL                                        """
len_ini = 1000                                  #starting length of microtubule (no.of sites)
site_length = 4.0e-4                            #site length in µm (len_ini*site_length = lenght_ini in µm)
max_length = 2.0                                #max length in µm
max_sites = 100000 #int(max_length / site_length)       #max length in no. of sites
min_sites = 30

NO_LENGTH_VARIATION = False                      # just set NO_LENGTH_VARIATION = True for fixed size

real_Duration =4000                          #duration in minutes
real_Tau = 0.0002                              #time step in minutes
Duration = int(real_Duration / real_Tau)     #duration in no, of time steps

len_total = 11*len_ini
left = 10*len_ini           #(former first) left starts at 10*len_ini to leave room for growth towards 0
right = left + len_ini - 1  #(former last) right fixed at the end of the array
"""////////////////////////////////////////////////////////////////////////
// END OF PARAMS FOR TRACK
////////////////////////////////////////////////////////////////////////
// PARAMETERS  FOR  MOTORS                                                """
omega_D_noN = 0.0                       # omega_D = omega_D_noN / length  [PRL90(8)2003]
omega_D = omega_D_noN / site_length
omega_A_noN = 1.0 * omega_D_noN         # K = omega_A / omega_D = 3  [PRL90(8)2003]
omega_A = omega_A_noN / site_length
alpha = 0.5                              # injection rate at right-hand boundary  [PRL90(8)2003]
beta  = 0.3                              # ejection rate at left-hand boundary  [PRL90(8)2003]
gamma = 0.7                             # growth rate at left-hand boundary
delta = 0.0                            # shrinkage rate at left-hand boundary

countInjRight = 0        # to count Injected at each timeStep
countEjLeft = 0        # to count Ejected  at each timeStep
countAttach = 0         # to count Attached at each timeStep
countDetach = 0         # to count Detached at each timeStep
countLimbo = 0          # to count those lost by track shortening
"""////////////////////////////////////////////////////////////////////////
// END OF PARAMS FOR MOTORS
////////////////////////////////////////////////////////////////////////"""

print(f"Initial microtubule of length {(right-left)*site_length} (µm) created")

# arrays to store indexes of the first/last active site and total length at each time step
list_of_firsts  = np.zeros(Duration, dtype=int)      #obsolete - first is fixed on right
list_of_lasts   = np.zeros(Duration, dtype=int)
list_of_lengths = np.zeros(Duration, dtype=int)

list_of_firsts[0] = right
list_of_lasts[0] = left
list_of_lengths[0] = list_of_firsts[0] - list_of_lasts[0] + 1
```

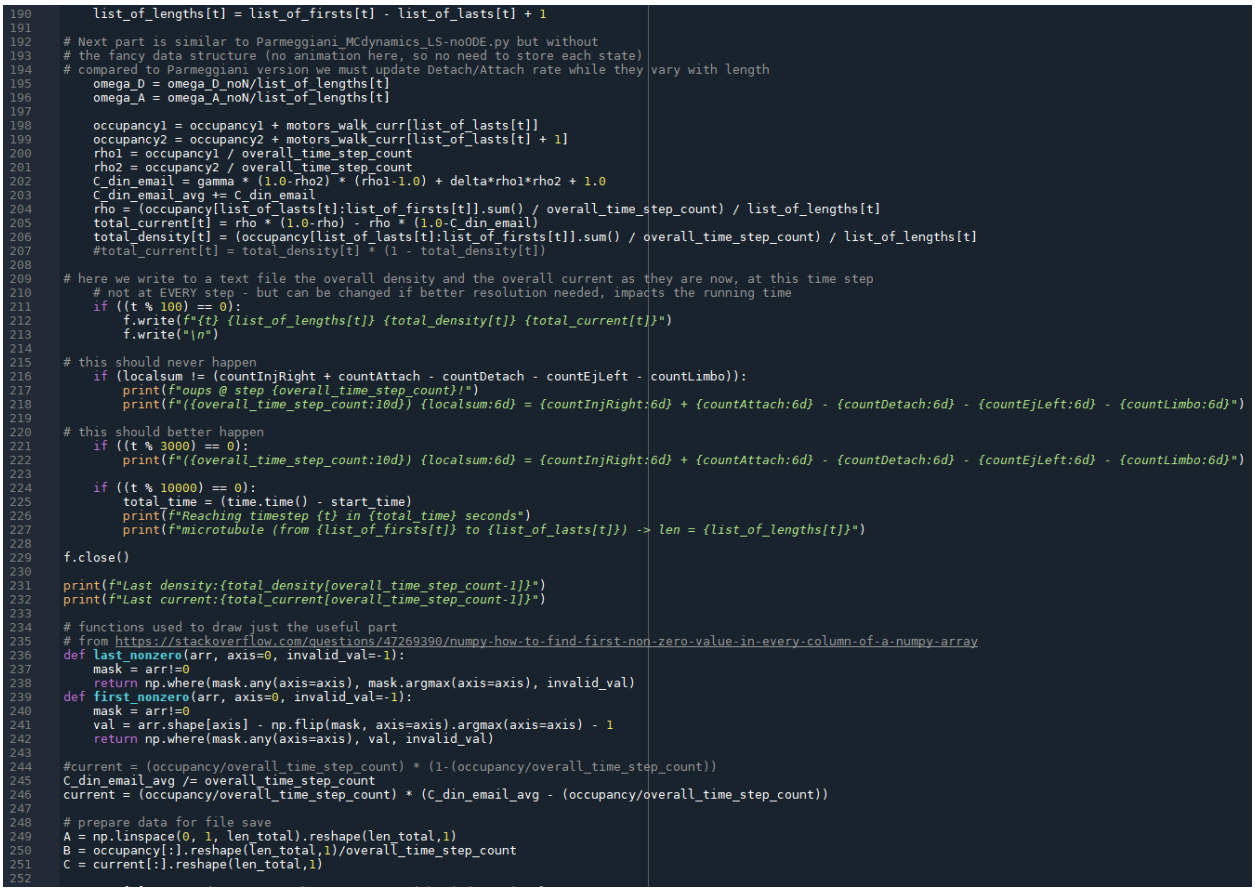
Figure D.1

```
    motors_walk_ante = np.zeros(len_total, dtype=int)
    motors_walk_curr = np.zeros(len_total, dtype=int)
    occupancy = np.zeros(len_total)

    total_density = np.zeros(Duration)
    current = np.zeros(len_total)
    total_current = np.zeros(Duration)

    """
    So, we've created two arrays several times larger than the initial size called motors_walk_ante
    and motors_walk_curr (large enough to give the microtubule enough space to grow on the left and
    on the right)
    Initially just the middle part is active  (first active site is at index 10*len_ini,
    last active site is at index=first+len_ini-1)
    We can keep track of the total length just using "right" and "left" variables
    The active part of motors_walk_ante [between right and left] will have ones and zeros for
    free/occupied sites
    """

    # and then we inject one motor at first active site on the right
    motor = 1
    # each motor can have a different index to keep track of them
    motors_walk_ante[list_of_firsts[0]] = motor
    motors_walk_curr[list_of_firsts[0]] = motor
    occupancy[list_of_firsts[0]] = motors_walk_curr[list_of_firsts[0]]
    countInjRight = 1

    overall_time_step_count = 1
    occupancy1 = 0
    occupancy2 = 0

    total_density[0] = (occupancy.sum()/overall_time_step_count) / list_of_lengths[0]
    total_current[0] = 0 #total_density[0] * (1 - total_density[0])
    C_din_email_avg  = 1.0 - gamma # initial value of C_din_email for occup1=0 and occup2=0

    if(NO_LENGTH_VARIATION):
        f = open(f"RtoL_length_and_current_fix_{len_ini}_sites_during_{real_Duration}_minutes.txt", "w")
    else:
        f = open(f"RtoL_length_and_current_var_{len_ini}_sites_during_{real_Duration}_minutes.txt", "w")

    f.write("time_step length total_density total_current")
    f.write("\n")

    if(NO_LENGTH_VARIATION):        # just set NO_LENGTH_VARIATION = True for fixed size
        gamma = -1
        delta = -1

    for t in range(1, Duration):                    # this is going to be the main loop

        overall_time_step_count = overall_time_step_count + 1 # now not really needed (= t)

        localsum = 0

        for i in range(list_of_lasts[t], (list_of_firsts[t]+1)):
            motors_walk_curr[i] = motors_walk_ante[i]           #initialize with values from previous step

        list_of_firsts[t] = list_of_firsts[t - 1]               #initialize with values from previous step
        list_of_lasts[t] = list_of_lasts[t - 1]
        list_of_lengths[t] = list_of_lengths[t - 1]

        particle = list_of_lasts[t] + int(random.random() * list_of_lengths[t])    #a random position between first and last

```

Figure D.2

```
    # BEGINNING THE BIG IF
    if ( (particle == list_of_lasts[t]) or (particle == list_of_lasts[t]+1)):
        # are we dealing with the last two sites (no. 1&2, left)?
        # then we can have:  (a) eject (10)->(00)  (b) jump (01)->(10)  (c) grow (00)->(000)  (d) shrink (11)->(0)
        if (0 != motors_walk_curr[list_of_lasts[t]]):   # if the first site is not empty
            if (0 == motors_walk_curr[list_of_lasts[t]+1]):
                if (beta > random.random()):
                    motors_walk_curr[list_of_lasts[t]] = 0    # (a) eject with beta probability
                    countEjLeft = countEjLeft + 1
            else:
                if (delta > random.random()):
                    if (list_of_lengths[t] <= min_sites):       #just a safety net for shrinkage
                        print(" *** too short - shrinkage denied *** ")
                        break
                    else:
                        countLimbo = countLimbo + 2
                        motors_walk_curr[list_of_lasts[t]] = 0     # (d) shrink with delta probability
                        motors_walk_curr[list_of_lasts[t] + 1] = 0
                        list_of_lasts[t] = list_of_lasts[t] + 1
        else:
            if (0 == motors_walk_curr[list_of_lasts[t]+1]):
                if (gamma > random.random()):
                    if (list_of_lengths[t] >= max_sites):       #just a safety net for growing
                        print(" *** too large - growth denied *** ")
                        break
                    else:
                        list_of_lasts[t] = list_of_lasts[t] - 1#   (c) grow with gamma probability
                        motors_walk_curr[list_of_lasts[t]] = 0     # add a new one to the left and initialize it with 0
                else:
                    motors_walk_curr[list_of_lasts[t]]   = 1       # (b) JUMP
                    motors_walk_curr[list_of_lasts[t]+1] = 0

    elif (particle == list_of_firsts[t]):
        # are we dealing with the first site (no. N, right)?
        if (0 == motors_walk_curr[list_of_firsts[t]]):       # if the first site is empty
            if (alpha > random.random()):
                motors_walk_curr[list_of_firsts[t]] = 1      # inject with alpha probability
                countInjRight = countInjRight + 1
        elif (0 == motors_walk_curr[list_of_firsts[t]-1]):   # if next is empty JUMP
            motors_walk_curr[list_of_firsts[t]-1] = motors_walk_curr[list_of_firsts[t]]
            motors_walk_curr[list_of_firsts[t]] = 0

    else:
        # now we know that we're dealing with a bulk site - no first or last two
        if (0 != motors_walk_curr[particle]):               # if the site is full
            if (omega_D > random.random()):                 # maybe we can detach it
                motors_walk_curr[particle] = 0
                countDetach = countDetach + 1
            else:
                if (0 == (motors_walk_curr[particle - 1])): # if we didn't detach it and if the next site is empty - JUMP (no ifs)!
                    motors_walk_curr[particle - 1] = 1
                    motors_walk_curr[particle] = 0
        else:                                               # if the site is empty
            if (omega_A > random.random()):                 # maybe we can attach one
                motors_walk_curr[particle] = 1
                countAttach = countAttach + 1
    # END OF THE BIG IF
    # save the state
    for m in range(list_of_lasts[t], (list_of_firsts[t]+1)):
        occupancy[m] = occupancy[m] + motors_walk_curr[m]
        localsum = localsum + motors_walk_curr[m]
        motors_walk_ante[m] = motors_walk_curr[m]
```

Figure D.3

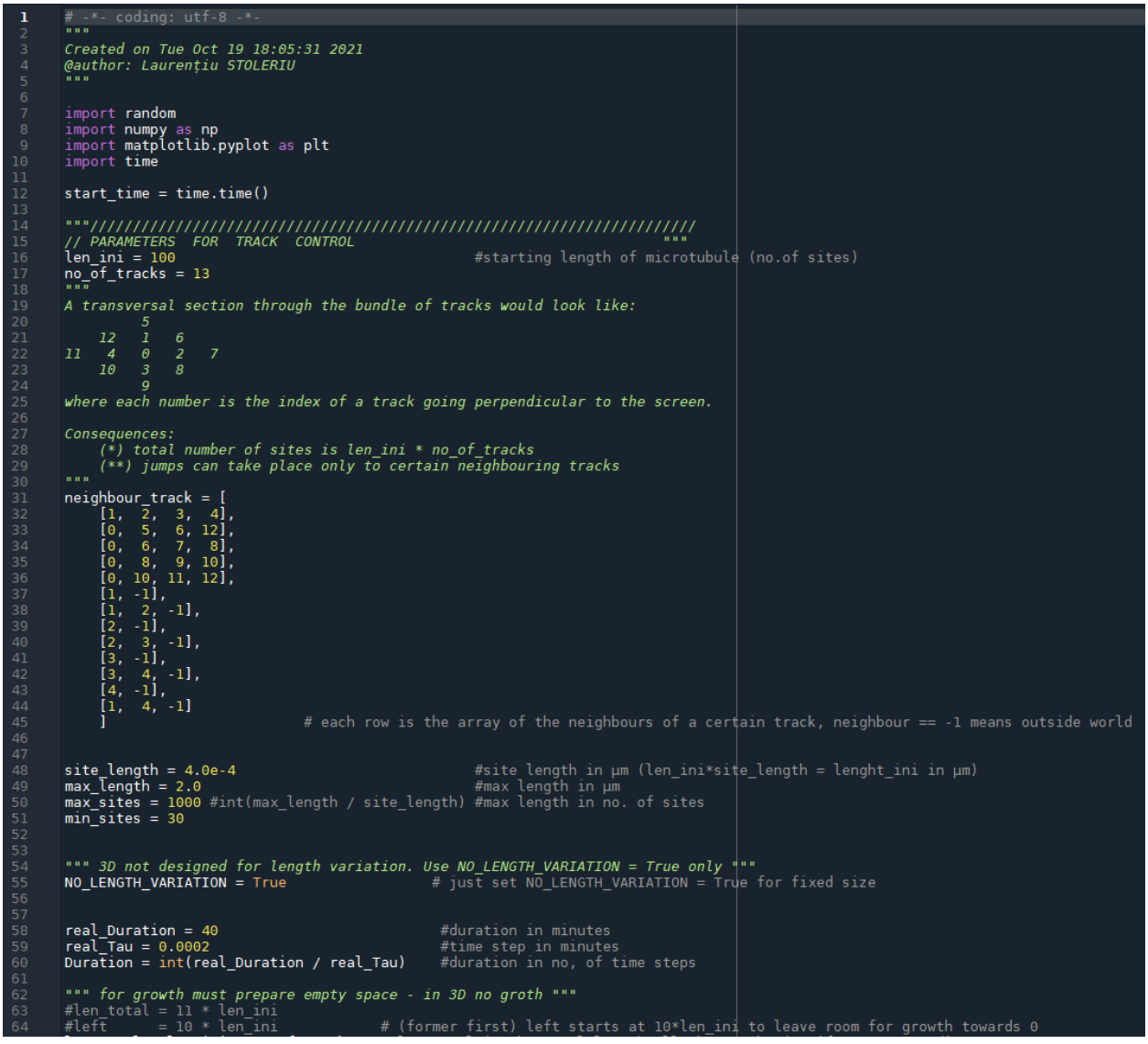


```
    list_of_lengths[t] = list_of_firsts[t] - list_of_lasts[t] + 1

# Next part is similar to Parmeggiani_MCdynamics_LS-noODE.py but without
# the fancy data structure (no animation here, so no need to store each state)
# compared to Parmeggiani version we must update Detach/Attach rate while they vary with length
    omega_D = omega_D_noN/list_of_lengths[t]
    omega_A = omega_A_noN/list_of_lengths[t]

    occupancy1 = occupancy1 + motors_walk_curr[list_of_lasts[t]]
    occupancy2 = occupancy2 + motors_walk_curr[list_of_lasts[t] + 1]
    rho1 = occupancy1 / overall_time_step_count
    rho2 = occupancy2 / overall_time_step_count
    C_din_email = gamma * (1.0-rho2) * (rho1-1.0) + delta*rho1*rho2 + 1.0
    C_din_email_avg += C_din_email
    rho = (occupancy[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]
    total_current[t] = rho * (1.0-rho) - rho * (1.0-C_din_email)
    total_density[t] = (occupancy[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]
    #total_current[t] = total_density[t] * (1 - total_density[t])

# here we write to a text file the overall density and the overall current as they are now, at this time step
    # not at EVERY step - but can be changed if better resolution needed, impacts the running time
    if ((t % 100) == 0):
        f.write(f"{t} {list_of_lengths[t]} {total_density[t]} {total_current[t]}")
        f.write("\n")

# this should never happen
    if (localsum != (countInjRight + countAttach - countDetach - countEjLeft - countLimbo)):
        print(f"oups @ step {overall_time_step_count}!")
        print(f"({overall_time_step_count:10d}) {localsum:6d} = {countInjRight:6d} + {countAttach:6d} - {countDetach:6d} - {countEjLeft:6d} - {countLimbo:6d}")

# this should better happen
    if ((t % 3000) == 0):
        print(f"({overall_time_step_count:10d}) {localsum:6d} = {countInjRight:6d} + {countAttach:6d} - {countDetach:6d} - {countEjLeft:6d} - {countLimbo:6d}")

    if ((t % 10000) == 0):
        total_time = (time.time() - start_time)
        print(f"Reaching timestep {t} in {total_time} seconds")
        print(f"microtubule (from {list_of_firsts[t]} to {list_of_lasts[t]}) -> len = {list_of_lengths[t]}")

f.close()

print(f"Last density:{total_density[overall_time_step_count-1]}")
print(f"Last current:{total_current[overall_time_step_count-1]}")

# functions used to draw just the useful part
# from https://stackoverflow.com/questions/47269390/numpy-how-to-find-first-non-zero-value-in-every-column-of-a-numpy-array
def last_nonzero(arr, axis=0, invalid_val=-1):
    mask = arr!=0
    return np.where(mask.any(axis=axis), mask.argmax(axis=axis), invalid_val)
def first_nonzero(arr, axis=0, invalid_val=-1):
    mask = arr!=0
    val = arr.shape[axis] - np.flip(mask, axis=axis).argmax(axis=axis) - 1
    return np.where(mask.any(axis=axis), val, invalid_val)

#current = (occupancy/overall_time_step_count) * (1-(occupancy/overall_time_step_count))
C_din_email_avg /= overall_time_step_count
current = (occupancy/overall_time_step_count) * (C_din_email_avg - (occupancy/overall_time_step_count))

# prepare data for file save
A = np.linspace(0, 1, len_total).reshape(len_total,1)
B = occupancy[:].reshape(len_total,1)/overall_time_step_count
C = current[:].reshape(len_total,1)

```

Figure D.4

```
    Arev = A[:] #no need to reverse here - we use right indexes in plot
    Brev = B[::-1]
    Crev = C[::-1]
    ready_to_save = np.hstack((A, B, C))
    if(NO_LENGTH_VARIATION):
        np.savetxt(f"RtoL_occup_of_fix_{len_ini}_sites_after_{real_Duration}_minutes.txt", ready_to_save, header="position  occupancy  current", comments='', delimiter=" ")
    else:
        np.savetxt(f"RtoL_occup_of_var_{len_ini}_sites_after_{real_Duration}_minutes.txt", ready_to_save, header="position  occupancy  current", comments='', delimiter=" ")

    if True:
        fig, [[ax1, ax3], [ax2, ax4], [ax5, ax6]] = plt.subplots(nrows=3, ncols=2)
        first_to_plot = right   #was first_nonzero(occupancy)-2
        #if first_to_plot < 0:
        #    first_to_plot = 0
        last_to_plot = last_nonzero(occupancy)-2
        if last_to_plot > len_total:
            last_to_plot = len_total
        #ax1.plot(A[last_to_plot:first_to_plot], B[last_to_plot:first_to_plot], color = "indianred")
        ax1.plot(Arev[0:len_total-last_to_plot], Brev[0:len_total-last_to_plot], color = "indianred")
        ax1.set_title("occupancy along the track")
        #ax1.set_xlabel("site position (normalized)")
        ax1.set_xticks([])
        ax1.set_ylabel("occupancy")

        #ax2.plot(A[last_to_plot:first_to_plot], current[last_to_plot:first_to_plot], color = "indianred")
        ax2.plot(Arev[0:len_total-last_to_plot], Crev[0:len_total-last_to_plot], color = "indianred")
        ax2.set_title("current along the track")
        ax2.set_xlabel("site position (normalized)")
        ax2.set_ylabel("current")

        ax3.plot(total_density)
        ax3.set_title("total density in time")
        #ax3.set_xlabel("site position (normalized)")
        ax3.set_xticks([])
        ax3.set_ylabel("total density")

        ax4.plot(total_current)
        ax4.set_title("total current in time")
        ax4.set_xlabel("time")
        ax4.set_ylabel("total current")

        ax6.plot(list_of_lengths)
        ax6.set_title("length in time")
        ax6.set_xlabel("time")
        ax6.set_ylabel("length")

        plt.show()
```

Figure D.5



```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 19 18:05:31 2021
@author: Laurențiu STOLERIU
"""

import random
import numpy as np
import matplotlib.pyplot as plt
import time

start_time = time.time()

"""//////////////////////////////////////////////////////////////////////
// PARAMETERS  FOR  TRACK  CONTROL                                    """
len_ini = 100                                   #starting length of microtubule (no.of sites)
no_of_tracks = 13
"""
A transversal section through the bundle of tracks would look like:
         5
     12   1   6
11    4   0   2   7
     10   3   8
         9
where each number is the index of a track going perpendicular to the screen.

Consequences:
    (*) total number of sites is len_ini * no_of_tracks
    (**) jumps can take place only to certain neighbouring tracks
"""
neighbour_track = [
    [1,  2,  3,  4],
    [0,  5,  6, 12],
    [0,  6,  7,  8],
    [0,  8,  9, 10],
    [0, 10, 11, 12],
    [1, -1],
    [1,  2, -1],
    [2, -1],
    [2,  3, -1],
    [3, -1],
    [3,  4, -1],
    [4, -1],
    [1,  4, -1]
    ]                       # each row is the array of the neighbours of a certain track, neighbour == -1 means outside world


site_length = 4.0e-4                            #site length in µm (len_ini*site_length = lenght_ini in µm)
max_length = 2.0                                #max length in µm
max_sites = 1000 #int(max_length / site_length) #max length in no. of sites
min_sites = 30


""" 3D not designed for length variation. Use NO_LENGTH_VARIATION = True only """
NO_LENGTH_VARIATION = True                # just set NO_LENGTH_VARIATION = True for fixed size


real_Duration = 40                          #duration in minutes
real_Tau = 0.0002                           #time step in minutes
Duration = int(real_Duration / real_Tau)    #duration in no, of time steps

""" for growth must prepare empty space - in 3D no groth """
#len_total = 11 * len_ini
#left      = 10 * len_ini          # (former first) left starts at 10*len_ini to leave room for growth towards 0
```

Figure D.6

```
65  len_total = len_ini * no_of_tracks   # len_total is the total length all the tracks (as if concatenated)
66  left      = 0
67  right     = left + len_ini - 1       # (former last) right fixed at the end of the array
68  """////////////////////////////////////////////////////////////////////////
69  // END OF PARAMS FOR TRACK
70  ////////////////////////////////////////////////////////////////////////
71  // PARAMETERS  FOR  MOTORS                                              """
72  omega_D_noN = 0.0                    # omega_D = omega_D_noN / length  [PRL90(8)2003]
73  omega_D = omega_D_noN / site_length
74  omega_A_noN = 1.0 * omega_D_noN      # K = omega_A / omega_D = 3  [PRL90(8)2003]
75  omega_A = omega_A_noN / site_length
76  omega_J = 0.5                        # NEW FOR 3D - some probability to jump from a track to a neighbouring track (not outside)
77
78  alpha = 0.5                          # injection rate at right-hand boundary  [PRL90(8)2003]
79  beta  = 0.3                          # ejection rate at left-hand boundary  [PRL90(8)2003]
80  gamma = 0.7                          # growth rate at left-hand boundary
81  delta = 0.0                          # shrinkage rate at left-hand boundary
82
83  countInjRight = 0       # to count Injected at each timeStep
84  countEjLeft = 0         # to count Ejected  at each timeStep
85  countAttach = 0         # to count Attached at each timeStep
86  countDetach = 0         # to count Detached at each timeStep
87  countLimbo = 0          # to count those lost by track shortening
88  """////////////////////////////////////////////////////////////////////////
89  // END OF PARAMS FOR MOTORS
90  ////////////////////////////////////////////////////////////////////////"""
91
92  print(f"Initial microtubule of length {(right-left)*site_length} (μm) created")
93
94  # arrays to store indexes of the first/last active site and total length at each time step
95  list_of_firsts  = np.zeros(Duration, dtype=int)      #obsolete - first is fixed on right
96  list_of_lasts   = np.zeros(Duration, dtype=int)
97  list_of_lengths = np.zeros(Duration, dtype=int)
98
99  list_of_firsts[0]  = right
100 list_of_lasts[0]   = left
101 list_of_lengths[0] = list_of_firsts[0] - list_of_lasts[0] + 1
102
103 motors_walk_ante = np.zeros(len_total, dtype=int)
104 motors_walk_curr = np.zeros(len_total, dtype=int)
105 occupancy = np.zeros(len_total)
106
107 total_density = np.zeros(Duration)
108 current = np.zeros(len_total)
109 total_current = np.zeros(Duration)
110
111 """
112 So, we've created two arrays several times larger than the initial size (of a single track) called
113 motors_walk_ante and motors_walk_curr (large enough to give the microtubule enough space to grow
114 on the left and on the right)
115 Initially just the middle part is active (first active site is at index 10*len_ini,
116 last active site is at index=first+len_ini-1)
117 We can keep track of the total length just using "right" and "left" variables
118 The active part of motors_walk_ante [between right and left] will have ones and zeros for
119 free/occupied sites
120 """
121
122 """
123 In 3D there are 13 tracks & no length variation.
124 Kept the 2D structure (i.e. the len_ini length of each track) but must add jumps between tracks
125 """
126
127 # and then we inject one motor at first active site on the right of the track no. 1
128 motor = 1
```

Figure D.7

```
# each motor can have a different index to keep track of them
motors_walk_ante[list_of_firsts[0]] = motor
motors_walk_curr[list_of_firsts[0]] = motor
occupancy[list_of_firsts[0]] = motors_walk_curr[list_of_firsts[0]]
countInjRight = 1

overall_time_step_count = 1
occupancy1 = 0
occupancy2 = 0

total_density[0] = (occupancy.sum()/overall_time_step_count) / list_of_lengths[0]
total_current[0] = 0                # total_density[0] * (1 - total_density[0])
C_din_email_avg  = 1.0 - gamma      # initial value of C_din_email for occup1=0 and occup2=0

if(NO_LENGTH_VARIATION):
    f = open(f"RtoL_length_and_current_fix_{len_ini}_sites_during_{real_Duration}_minutes.txt", "w")
else:
    f = open(f"RtoL_length_and_current_var_{len_ini}_sites_during_{real_Duration}_minutes.txt", "w")

f.write("time_step length total_density total_current")
f.write("\n")

if(NO_LENGTH_VARIATION):        # just set NO_LENGTH_VARIATION = True for fixed size
    gamma = -1
    delta = -1

for t in range(1, Duration):                    # this is going to be the main loop (time evolution)

    overall_time_step_count = overall_time_step_count + 1   # now not really needed (= t)

    localsum = 0

    for k in range(0, no_of_tracks):
        for i in range(list_of_lasts[t], (list_of_firsts[t]+1)):
            motors_walk_curr[i + len_ini*k] = motors_walk_ante[i + len_ini*k]    # initialize with values from previous step

    list_of_firsts[t] = list_of_firsts[t - 1]               # initialize with values from previous step
    list_of_lasts[t] = list_of_lasts[t - 1]
    list_of_lengths[t] = list_of_lengths[t - 1]

    particle = list_of_lasts[t] + int(random.random() * list_of_lengths[t])    # a random position between first and last
    track_idx = int(random.random() * no_of_tracks)
    track = len_ini * track_idx                                                # a random track
    """
    track variable is in fact the offset of the position in motors_walk_curr
    so motors_walk_curr[particle + track] is a random position on a random track
    being an offset it comes pre-multiplied with len_ini to save some time below
    """

    # BEGINNING THE BIG IF
    if ((particle == list_of_lasts[t]) or (particle == list_of_lasts[t]+1)):
        # are we dealing with the last two sites (no. 1&2, left)?
        # then we can have:  (a) eject (10)->(00)  (b) jump (01)->(10)  (c) grow (00)->(000)  (d) shrink (11)->(0)
        """ in 3D we keep only cases (a) eject and (b) jump while we comment out the (c) and (d) """
        if (0 != motors_walk_curr[list_of_lasts[t] + track]):   # if the first site of the chosen track is not empty
            if (0 == motors_walk_curr[list_of_lasts[t] + track + 1]):
                if (beta > random.random()):
                    motors_walk_curr[list_of_lasts[t] + track] = 0    # (a) eject on left with beta probability
                    countEjLeft = countEjLeft + 1
            """ shrinkage code - NOT FOR 3D """
            #else:
            #    if (delta > random.random()):
            #        if (list_of_lengths[t] <= min_sites):       #just a safety net for shrinkage
            #            print(" *** too short - shrinkage denied *** ")
```

Figure D.8

```
            #           break
            #       else:
            #           countLimbo = countLimbo + 2
            #           motors_walk_curr[list_of_lasts[t]] = 0     # (d) shrink with delta probability
            #           motors_walk_curr[list_of_lasts[t] + 1] = 0
            #           list_of_lasts[t] = list_of_lasts[t] + 1
        else:
            """ growth code - NOT FOR 3D """
            #if (0 == motors_walk_curr[list_of_lasts[t]+1]):
            #    if (gamma > random.random()):
            #        if (list_of_lengths[t] >= max_sites):       #just a safety net for growing
            #            print(" *** too large - growth denied *** ")
            #            break
            #        else:
            #            list_of_lasts[t] = list_of_lasts[t] - 1#   (c) grow with gamma probability
            #            motors_walk_curr[list_of_lasts[t]] = 0     # add a new one to the left and initialize it with 0
            #else:
            #    motors_walk_curr[list_of_lasts[t]]   = 1       # (b) JUMP
            #    motors_walk_curr[list_of_lasts[t]+1] = 0
            if (0 != motors_walk_curr[list_of_lasts[t] + track + 1]):
                motors_walk_curr[list_of_lasts[t] + track]   = 1       # (b) JUMP
                motors_walk_curr[list_of_lasts[t] + track + 1] = 0

    elif (particle == list_of_firsts[t]):
        # are we dealing with the first site (no, N, right)?
        if (0 == motors_walk_curr[list_of_firsts[t] + track]):       # if the first site is empty
            if (alpha > random.random()):
                motors_walk_curr[list_of_firsts[t] + track] = 1      # inject with alpha probability
                countInjRight = countInjRight + 1
        elif (0 == motors_walk_curr[list_of_firsts[t] + track - 1]):   # if next is empty JUMP
            motors_walk_curr[list_of_firsts[t] + track - 1] = motors_walk_curr[list_of_firsts[t] + track]
            motors_walk_curr[list_of_firsts[t] + track] = 0

    else:
        # now we know that we're dealing with a bulk site - no first or last two
        target_track = int( random.random() * len(neighbour_track[track_idx]) )     # chosing a random neighbour track of the current track
        if (0 != motors_walk_curr[particle + track]):                               # if the site is full

            if (neighbour_track[track_idx][target_track] < 0):                      # is -1, so exterior
                if (omega_D > random.random()):                                     # maybe we can detach it
                    motors_walk_curr[particle + track] = 0
                    countDetach = countDetach + 1
            else:                                                                   # is a regular neighbouring track
                if ( (0 == motors_walk_curr[particle + target_track]) and (omega_J > random.random()) ):    # maybe it can jump to other track
                    motors_walk_curr[particle + track] = 0
                    motors_walk_curr[particle + target_track] = 1
                    #countJump = countJump + 1                  # not really useful, total number of motors unchanged
                else:
                    if (0 == (motors_walk_curr[particle + track - 1])): # if we didn't send it to another track it and if the next site is empty - JUMP (no ifs)!
                        motors_walk_curr[particle + track - 1] = 1
                        motors_walk_curr[particle + track] = 0
        else:                                   # if the site is empty
            if ((neighbour_track[track_idx][target_track] < 0) and (omega_A > random.random()) ):  # maybe we can attach one from outside
                motors_walk_curr[particle + track] = 1
                countAttach = countAttach + 1
    # END OF THE BIG IF
    # save the state
    for k in range(0, no_of_tracks):
        for m in range(list_of_lasts[t], (list_of_firsts[t]+1)):
            occupancy[m + len_ini*k] = occupancy[m + len_ini*k] + motors_walk_curr[m + len_ini*k]
            localsum = localsum + motors_walk_curr[m + len_ini*k]
            motors_walk_ante[m + len_ini*k] = motors_walk_curr[m + len_ini*k]

    list_of_lengths[t] = list_of_firsts[t] - list_of_lasts[t] + 1
```

Figure D.9

```
257
258     # Next part is similar to Parmeggiani_MCdynamics_LS-noODE.py but without
259     # the fancy data structure (no animation here, so no need to store each state)
260     # compared to Parmeggiani version we must update Detach/Attach rate while they vary with length
261         omega_D = omega_D_noN/list_of_lengths[t]
262         omega_A = omega_A_noN/list_of_lengths[t]
263
264         occupancy1 = occupancy1 + motors_walk_curr[list_of_lasts[t]]
265         occupancy2 = occupancy2 + motors_walk_curr[list_of_lasts[t] + 1]
266         rho1 = occupancy1 / overall_time_step_count
267         rho2 = occupancy2 / overall_time_step_count
268         C_din_email = gamma * (1.0-rho2) * (rho1-1.0) + delta*rho1*rho2 + 1.0
269         C_din_email_avg += C_din_email
270         rho = (occupancy[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]
271         total_current[t] = rho * (1.0-rho) - rho * (1.0-C_din_email)
272         total_density[t] = (occupancy[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]
273         #total_current[t] = total_density[t] * (1 - total_density[t])
274
275     # here we write to a text file the overall density and the overall current as they are now, at this time step
276         # not at EVERY step - but can be changed if better resolution needed, impacts the running time
277         if ((t % 100) == 0):
278             f.write(f"{t} {list_of_lengths[t]} {total_density[t]} {total_current[t]}")
279             f.write("\n")
280
281     # this should never happen
282         if (localsum != (countInjRight + countAttach - countDetach - countEjLeft - countLimbo)):
283             print(f"oups @ step {overall_time_step_count}!")
284             print(f"({overall_time_step_count:10d}) {localsum:6d} = {countInjRight:6d} + {countAttach:6d} - {countDetach:6d} - {countEjLeft:6d} - {countLimbo:6d}")
285
286     # this should better happen
287         if ((t % 3000) == 0):
288             print(f"({overall_time_step_count:10d}) {localsum:6d} = {countInjRight:6d} + {countAttach:6d} - {countDetach:6d} - {countEjLeft:6d} - {countLimbo:6d}")
289
290         if ((t % 10000) == 0):
291             total_time = (time.time() - start_time)
292             print(f"Reaching timestep {t} in {total_time} seconds")
293             print(f"microtubule (from {list_of_firsts[t]} to {list_of_lasts[t]}) -> len = {list_of_lengths[t]}")
294
295     f.close()
296
297     print(f"Last density:{total_density[overall_time_step_count-1]}")
298     print(f"Last current:{total_current[overall_time_step_count-1]}")
299
300     # functions used to draw just the useful part
301     # from https://stackoverflow.com/questions/47269390/numpy-how-to-find-first-non-zero-value-in-every-column-of-a-numpy-array
302     def last_nonzero(arr, axis=0, invalid_val=-1):
303         mask = arr!=0
304         return np.where(mask.any(axis=axis), mask.argmax(axis=axis), invalid_val)
305     def first_nonzero(arr, axis=0, invalid_val=-1):
306         mask = arr!=0
307         val = arr.shape[axis] - np.flip(mask, axis=axis).argmax(axis=axis) - 1
308         return np.where(mask.any(axis=axis), val, invalid_val)
309
310     #current = (occupancy/overall_time_step_count) * (1-(occupancy/overall_time_step_count))
311     C_din_email_avg /= overall_time_step_count
312     current = (occupancy/overall_time_step_count) * (C_din_email_avg - (occupancy/overall_time_step_count))
313
314     # prepare data for file save
315     A = np.linspace(0, 1, len_total).reshape(len_total,1)
316     B = occupancy[:].reshape(len_total,1)/overall_time_step_count
317     C = current[:].reshape(len_total,1)
318
319     Arev = A[:] #no need to reverse here - we use right indexes in plot
320     Brev = B[::-1]
```

Figure D.10

```
321     Crev = C[::-1]
322     ready_to_save = np.hstack((A, B, C))
323     if(NO_LENGTH_VARIATION):
324         np.savetxt(f"RtoL_occup_of_fix_{len_ini}_sites_after_{real_Duration}_minutes.txt", ready_to_save, header="position  occupancy  current", comments='', delimiter=" ")
325     else:
326         np.savetxt(f"RtoL_occup_of_var_{len_ini}_sites_after_{real_Duration}_minutes.txt", ready_to_save, header="position  occupancy  current", comments='', delimiter=" ")
327
328     if True:
329         fig, [[ax1, ax3], [ax2, ax4], [ax5, ax6]] = plt.subplots(nrows=3, ncols=2)
330         first_to_plot = right   #was first_nonzero(occupancy)-2
331         #if first_to_plot < 0:
332         #    first_to_plot = 0
333         last_to_plot = last_nonzero(occupancy)-2
334         if last_to_plot > len_total:
335             last_to_plot = len_total
336         #ax1.plot(A[last_to_plot:first_to_plot], B[last_to_plot:first_to_plot], color = "indianred")
337         ax1.plot(Arev[0:len_total-last_to_plot], Brev[0:len_total-last_to_plot], color = "indianred")
338         ax1.set_title("occupancy along the track")
339         #ax1.set_xlabel("site position (normalized)")
340         ax1.set_xticks([])
341         ax1.set_ylabel("occupancy")
342
343         #ax2.plot(A[last_to_plot:first_to_plot], current[last_to_plot:first_to_plot], color = "indianred")
344         ax2.plot(Arev[0:len_total-last_to_plot], Crev[0:len_total-last_to_plot], color = "indianred")
345         ax2.set_title("current along the track")
346         ax2.set_xlabel("site position (normalized)")
347         ax2.set_ylabel("current")
348
349         ax3.plot(total_density)
350         ax3.set_title("total density in time")
351         #ax3.set_xlabel("site position (normalized)")
352         ax3.set_xticks([])
353         ax3.set_ylabel("total density")
354
355         ax4.plot(total_current)
356         ax4.set_title("total current in time")
357         ax4.set_xlabel("time")
358         ax4.set_ylabel("total current")
359
360         ax6.plot(list_of_lengths)
361         ax6.set_title("length in time")
362         ax6.set_xlabel("time")
363         ax6.set_ylabel("length")
364
365         plt.show()
```

Figure D.11

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 26 22:42:15 2022
@author: Laurențiu STOLERIU

two species with new rules

"""

import random
import numpy as np
import matplotlib.pyplot as plt
import time
start_time = time.time()

"""////////////////////////////////////////////////////////////////////////
// PARAMETERS  FOR  TRACK  CONTROL                                      """
len_ini = 100                  # starting length of microtubule (no.of sites)
site_length = 4.0e-4            # site length in µm (len_ini*site_length = lenght_ini in µm)
max_length = 2.0                # max length in µm
max_sites = 100000 #int(max_length / site_length)       #max length in no. of sites
min_sites = 30

NO_LENGTH_VARIATION = True      # just set NO_LENGTH_VARIATION = True for fixed size

real_Duration = 35              # duration in minutes
real_Tau = 0.0002                       # time step in minutes
Duration = int(real_Duration / real_Tau)     # duration in no, of time steps

len_total = 11*len_ini       # enough space in both left and right
left = 10*len_ini            # (former first) left starts at 10*len_ini to leave room for growth towards 0
right = left + len_ini - 1   # (former last) right fixed at the end of the array
"""////////////////////////////////////////////////////////////////////////
// END OF PARAMS FOR TRACK
////////////////////////////////////////////////////////////////////////
// PARAMETERS  FOR  MOTORS                                              """
omegaD1_noN = 0.0                       # omega_D = omega_D_noN / length  [PRL90(8)2003]
omegaD1 = omegaD1_noN / site_length
omegaA1_noN = 1.0 * omegaD1_noN         # K = omega_A / omega_D = 3  [PRL90(8)2003]
omegaA1 = omegaA1_noN / site_length

omegaD2_noN = 0.0                       # omega_D = omega_D_noN / length  [PRL90(8)2003]
omegaD2 = omegaD2_noN / site_length
omegaA2_noN = 1.0 * omegaD2_noN         # K = omega_A / omega_D = 3  [PRL90(8)2003]
omegaA2 = omegaA2_noN / site_length

alpha1 = 0.2                            # injection rate for kinesin at right-hand boundary  [PRL90(8)2003]
beta1  = 0.0                            # ejection rate for kinesin at left-hand boundary  [PRL90(8)2003]

alpha2 = 0.3                            # injection rate for dynein at left-hand boundary
beta2  = 0.0                            # ejection rate for dynein at right-hand boundary

gamma = 0.1                             # growth rate at left-hand boundary
delta = 0.1                             # shrinkage rate at left-hand boundary

count1InjRight = 0        # to count Injected kinesin at each timeStep
count1EjLeft = 0          # to count Ejected kinesin at each timeStep
count1Attach = 0          # to count Attached kinesin at each timeStep
count1Detach = 0          # to count Detached kinesin at each timeStep
count1Limbo = 0           # to count lost kinesin by track shortening

count2InjLeft = 0         # to count Injected dynein at each timeStep
count2EjRight = 0         # to count Ejected dynein at each timeStep
count2Attach = 0          # to count Attached dynein at each timeStep
```

Figure D.12

```
65    count2Detach = 0          # to count Detached dynein at each timeStep
66    count2Limbo = 0           # to count lost dynein by track shortening
67
68    """////////////////////////////////////////////////////////////////////////
69    // END OF PARAMS FOR MOTORS
70    ////////////////////////////////////////////////////////////////////////"""
71
72    print(f"Initial microtubule of length {(right-left)*site_length} (μm) created")
73    print("(time  step):")
74    print(" totSum = <Right1 + Attach1 - Detach1 -  <Left1 -  Limbo1")
75    print("          >Right2 + Attach2 - Detach2 -  >Left2 -  Limbo2")
76
77
78    # arrays to store indexes of the first/last active site and total length at each time step
79    list_of_firsts  = np.zeros(Duration, dtype=int)      #obsolete - first is fixed on right
80    list_of_lasts   = np.zeros(Duration, dtype=int)
81    list_of_lengths = np.zeros(Duration, dtype=int)
82
83    list_of_firsts[0]  = right
84    list_of_lasts[0]   = left
85    list_of_lengths[0] = list_of_firsts[0] - list_of_lasts[0] + 1
86
87    motors_walk_ante = np.zeros(len_total, dtype=int)
88    motors_walk_curr = np.zeros(len_total, dtype=int)
89
90    occupancy = np.zeros(len_total)
91    occupancy1 = np.zeros(len_total)
92    occupancy2 = np.zeros(len_total)
93
94    total_density = np.zeros(Duration)
95    total_density1 = np.zeros(Duration)
96    total_density2 = np.zeros(Duration)
97
98    current = np.zeros(len_total)
99    current1 = np.zeros(len_total)
100   current2 = np.zeros(len_total)
101
102   total_current = np.zeros(Duration)
103
104   """
105   So, we've created two arrays several times larger than the initial size called motors_walk_ante
106   and motors_walk_curr (large enough to give the microtubule enough space to grow on the left and
107   on the right)
108   Initially just the middle part is active  (first active site is at index 10*len_ini,
109   last active site is at index=first+len_ini-1)
110   We can keep track of the total length just using "right" and "left" variables
111   The active part of motors_walk_ante [between right and left] will have zeros, ones and twos for
112   free/occupied sites
113   """
114
115   # and then we inject one motor at first active site on the right
116   kinesin = 1
117   dynein = 2
118   #
119   # some initialization - not needed now
120   #motors_walk_ante[list_of_firsts[0]] = kinesin
121   #motors_walk_curr[list_of_firsts[0]] = kinesin
122   #occupancy[list_of_firsts[0]] = 1 #motors_walk_curr[list_of_firsts[0]]
123   #count1InjRight = 1
124
125   overall_time_step_count = 1
126
127
128   total_density[0] = (occupancy.sum()/overall_time_step_count) / list_of_lengths[0]
```

Figure D.13

```
129     total_density1[0] = (occupancy1.sum()/overall_time_step_count) / list_of_lengths[0]
130     total_density2[0] = (occupancy2.sum()/overall_time_step_count) / list_of_lengths[0]
131
132     total_current[0] = 0 #total_density[0] * (1 - total_density[0])
133     #C_din_email_avg  = 1.0 - gamma # initial value of C_din_email for occup1=0 and occup2=0
134
135     if(NO_LENGTH_VARIATION):
136         f = open(f"RtoL_length_and_current_fix_{len_ini}_sites_during_{real_Duration}_minutes.txt", "w")
137     else:
138         f = open(f"RtoL_length_and_current_var_{len_ini}_sites_during_{real_Duration}_minutes.txt", "w")
139
140     f.write("time_step length total_density total_current")
141     f.write("\n")
142
143     if(NO_LENGTH_VARIATION):        # just set NO_LENGTH_VARIATION = True for fixed size
144         gamma = -1
145         delta = -1
146
147     for t in range(1, Duration):                    # this is going to be the main loop
148
149         overall_time_step_count = overall_time_step_count + 1   # now not really needed (= t)
150
151         if False:
152             ### DEBUG save microtubule @ every step
153             with open("Microtubule_fix.txt", "a") as fis:
154                 np.savetxt(fis, motors_walk_curr[list_of_lasts[t-1]:list_of_firsts[t-1]], fmt='%1d', newline=' ')
155                 np.savetxt(fis, [8], fmt='%1d', newline='\n')
156
157         if False:
158             ### DEBUG print microtubule @ every step
159             for m in range(list_of_lasts[t-1], (list_of_lasts[t-1]+21)):
160                 print(f"{motors_walk_curr[m]}", end ="")
161             print(" ... ", end =" ")
162             for m in range(list_of_firsts[t-1]-20, (list_of_firsts[t-1]+1)):
163                 print(f"{motors_walk_curr[m]}", end ="")
164             print(" ")
165             ### END DEBUG print microtubule
166
167         localsum = 0
168
169         for i in range(list_of_lasts[t], (list_of_firsts[t]+1)):
170             motors_walk_curr[i] = motors_walk_ante[i]           #initialize with values from previous step
171
172         list_of_firsts[t] = list_of_firsts[t - 1]               #initialize with values from previous step
173         list_of_lasts[t] = list_of_lasts[t - 1]
174         list_of_lengths[t] = list_of_lengths[t - 1]
175
176         particle = list_of_lasts[t] + int(random.random() * list_of_lengths[t])    #a random position between first and last
177
178         # BEGINNING THE BIG IF
179         if ( (particle == list_of_lasts[t])):
180             # are we dealing with the last two sites (no. 1&2, left)?
181             # then we can have:  (a) eject (10)->(00)  (b) jump (01)->(10)  (c) grow (00)->(000)  (d) shrink (11)->(0), (12)->(0), (21)->0, (22)->(0)
182             #                    (e) insert (00)->(20) (f) jump (20)->(02)
183             firstTwo = str(motors_walk_curr[list_of_lasts[t]]) + str(motors_walk_curr[list_of_lasts[t]+1])
184             if ("10" == firstTwo):
185                 if (beta1 > random.random()):
186                     motors_walk_curr[list_of_lasts[t]] = 0       # (a) eject kinesin with beta1 probability
187                     count1EjLeft = count1EjLeft + 1
188             elif ("11" == firstTwo):
189                 if (delta > random.random()):
190                     if (list_of_lengths[t] <= min_sites):         #just a safety net for shrinkage
191                         print(" *** too short - shrinkage denied *** ")
192                         break
```

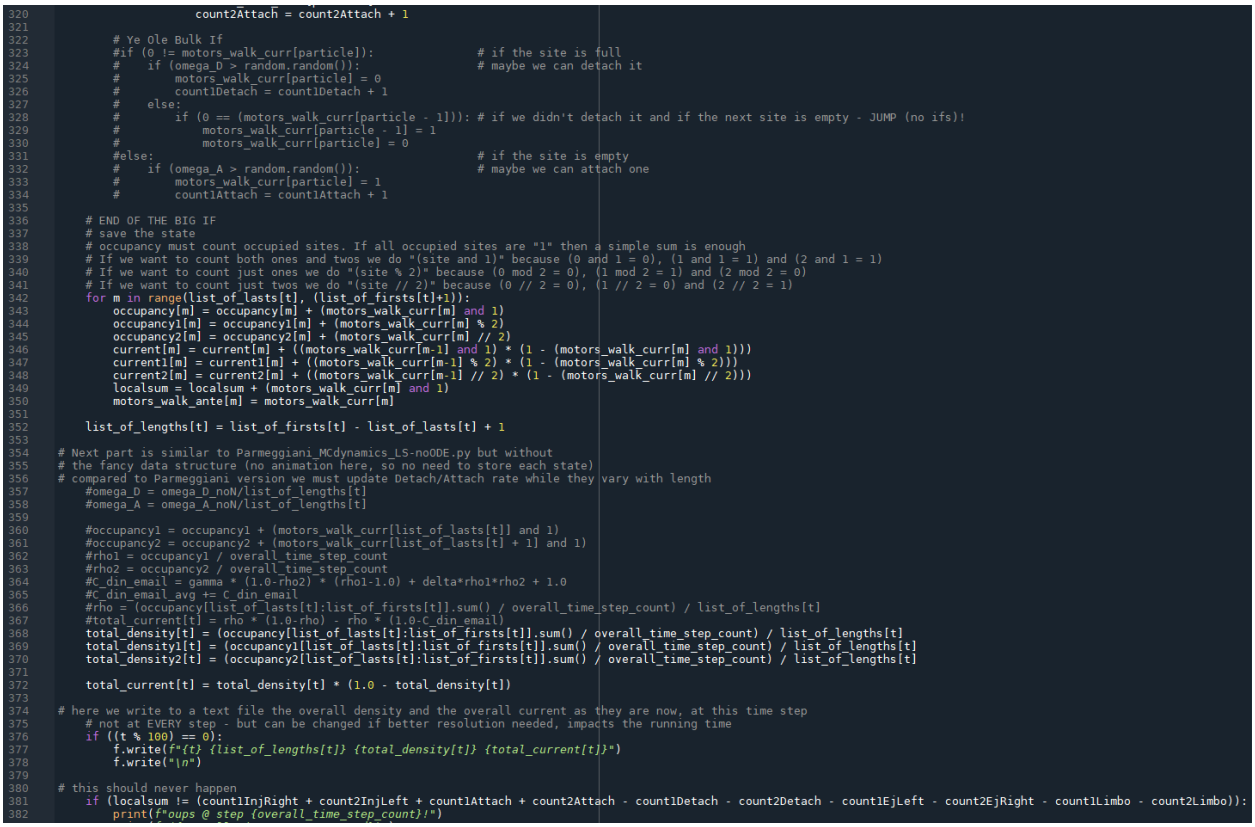
Figure D.14

```
                else:
                    count1Limbo = count1Limbo + 2
                    motors_walk_curr[list_of_lasts[t]] = 0     # (d) shrink with delta probability
                    motors_walk_curr[list_of_lasts[t] + 1] = 0
                    list_of_lasts[t] = list_of_lasts[t] + 1
            elif ("12" == firstTwo) or ("21" == firstTwo):
                if (delta > random.random()):
                    if (list_of_lengths[t] <= min_sites):          # just a safety net for shrinkage
                        print(" *** too short - shrinkage denied *** ")
                        break
                    else:
                        count1Limbo = count1Limbo + 1
                        count2Limbo = count2Limbo + 1
                        motors_walk_curr[list_of_lasts[t]] = 0     # (d) shrink with delta probability
                        motors_walk_curr[list_of_lasts[t] + 1] = 0
                        list_of_lasts[t] = list_of_lasts[t] + 1
            elif ("22" == firstTwo):
                if (delta > random.random()):
                    if (list_of_lengths[t] <= min_sites):          # just a safety net for shrinkage
                        print(" *** too short - shrinkage denied *** ")
                        break
                    else:
                        count2Limbo = count2Limbo + 2
                        motors_walk_curr[list_of_lasts[t]] = 0     # (d) shrink with delta probability
                        motors_walk_curr[list_of_lasts[t] + 1] = 0
                        list_of_lasts[t] = list_of_lasts[t] + 1
            elif ("20" == firstTwo):
                motors_walk_curr[list_of_lasts[t]]   = 0       # (f) JUMP dynein
                motors_walk_curr[list_of_lasts[t]+1] = 2
            elif ("01" == firstTwo):
                motors_walk_curr[list_of_lasts[t]]   = 1       # (b) JUMP kinesin
                motors_walk_curr[list_of_lasts[t]+1] = 0
            elif ("00" == firstTwo):
                if (alpha2 > random.random()):
                    motors_walk_curr[list_of_lasts[t]] = 2         # (e) inject dynein with alpha2 probability
                    count2InjLeft = count2InjLeft + 1
                else:
                    if (gamma > random.random()):
                        if (list_of_lengths[t] >= max_sites):       # just a safety net for growing
                            print(" *** too large - growth denied *** ")
                            break
                        else:
                            list_of_lasts[t] = list_of_lasts[t] - 1     # (c) grow with gamma probability
                            motors_walk_curr[list_of_lasts[t]] = 0

            # Ye Ole Left End If
            #if (1 == motors_walk_curr[list_of_lasts[t]]):         # if the first site is kinesin
            #    if (0 == motors_walk_curr[list_of_lasts[t]+1]):   # and the next is free
            #        if (beta1 > random.random()):
            #            motors_walk_curr[list_of_lasts[t]] = 0    # (a) eject with beta1 probability
            #            count1EjLeft = count1EjLeft + 1
            #    else:
            #        if (delta > random.random()):
            #            if (list_of_lengths[t] <= min_sites):       #just a safety net for shrinkage
            #                print(" *** too short - shrinkage denied *** ")
            #                break
            #            else:
            #                count1Limbo = count1Limbo + 2
            #                motors_walk_curr[list_of_lasts[t]] = 0     # (d) shrink with delta probability
            #                motors_walk_curr[list_of_lasts[t] + 1] = 0
            #                list_of_lasts[t] = list_of_lasts[t] + 1
            #else:
            #    if (0 == motors_walk_curr[list_of_lasts[t]+1]):
```

Figure D.15

```
256         #        if (gamma > random.random()):
257         #            if (list_of_lengths[t] >= max_sites):       #just a safety net for growing
258         #                print(" *** too large - growth denied *** ")
259         #                break
260         #            else:
261         #                list_of_lasts[t] = list_of_lasts[t] - 1#   (c) grow with gamma probability
262         #                motors_walk_curr[list_of_lasts[t]] = 0     # add a new one to the left and initialize it with 0
263         #    else:
264         #        motors_walk_curr[list_of_lasts[t]]   = 1       # (b) JUMP
265         #        motors_walk_curr[list_of_lasts[t]+1] = 0
266 
267     elif (particle == list_of_firsts[t]):
268         # are we dealing with the first site (no. N, right)?
269         lastTwo = str(motors_walk_curr[list_of_firsts[t]-1]) + str(motors_walk_curr[list_of_firsts[t]])
270         if ("00" == lastTwo):
271             if (alpha1 > random.random()):
272                 motors_walk_curr[list_of_firsts[t]] = 1         # inject kinesin with alpha1 probability
273                 count1InjRight = count1InjRight + 1
274         elif ("01" == lastTwo):                                 # JUMP kinesin
275             motors_walk_curr[list_of_firsts[t]-1] = 1
276             motors_walk_curr[list_of_firsts[t]] = 0
277         elif ("02" == lastTwo):
278             if (beta2 > random.random()):
279                 motors_walk_curr[list_of_firsts[t]] = 0         # (a) eject dynein with beta1 probability
280                 count2EjRight = count2EjRight + 1
281         elif ("20" == lastTwo):                                 # JUMP dynein (last jump - reaching end)
282             motors_walk_curr[list_of_firsts[t]-1] = 0
283             motors_walk_curr[list_of_firsts[t]] = 2
284 
285         # Ye Ole Right End If
286         #if (0 == motors_walk_curr[list_of_firsts[t]]):       # if the first site is empty
287         #    if (alpha1 > random.random()):
288         #        motors_walk_curr[list_of_firsts[t]] = 1      # inject with alpha1 probability
289         #        count1InjRight = count1InjRight + 1
290         #elif (0 == motors_walk_curr[list_of_firsts[t]-1]):   # if next is empty JUMP
291         #    motors_walk_curr[list_of_firsts[t]-1] = motors_walk_curr[list_of_firsts[t]]
292         #    motors_walk_curr[list_of_firsts[t]] = 0
293 
294     else:
295         # now we know that we're dealing with a bulk site - no first or last two
296         if (1 == motors_walk_curr[particle]):
297             if (omegaD1 > random.random()):                 # maybe we can detach kinesin
298                 motors_walk_curr[particle] = 0
299                 count1Detach = count1Detach + 1
300             else:
301                 if (0 == (motors_walk_curr[particle - 1])): # if we didn't detach it and if the next site is empty - JUMP (no ifs)!
302                     motors_walk_curr[particle - 1] = 1
303                     motors_walk_curr[particle] = 0
304         elif (2 == motors_walk_curr[particle]):
305             if (omegaD2 > random.random()):                 # maybe we can detach dynein
306                 motors_walk_curr[particle] = 0
307                 count2Detach = count2Detach + 1
308             else:
309                 if (0 == (motors_walk_curr[particle + 1])): # if we didn't detach it and if the next site is empty - JUMP (no ifs)!
310                     motors_walk_curr[particle + 1] = 2
311                     motors_walk_curr[particle] = 0
312         else:
313             if (0.5 > random.random()):                     # select kinesin OR dyenin to test attach
314                 if (omegaA1 > random.random()):             # if the site is empty maybe we can attach kinesin
315                     motors_walk_curr[particle] = 1
316                     count1Attach = count1Attach + 1
317             else:
318                 if (omegaA2 > random.random()):             # if the site is empty maybe we can attach kinesin
319                     motors_walk_curr[particle] = 2
```

Figure D.16



```
                    count2Attach = count2Attach + 1

            # Ye Ole Bulk If
            #if (0 != motors_walk_curr[particle]):              # if the site is full
            #    if (omega_D > random.random()):                # maybe we can detach it
            #        motors_walk_curr[particle] = 0
            #        count1Detach = count1Detach + 1
            #    else:
            #        if (0 == (motors_walk_curr[particle - 1])): # if we didn't detach it and if the next site is empty - JUMP (no ifs)!
            #            motors_walk_curr[particle - 1] = 1
            #            motors_walk_curr[particle] = 0
            #else:                                              # if the site is empty
            #    if (omega_A > random.random()):                # maybe we can attach one
            #        motors_walk_curr[particle] = 1
            #        count1Attach = count1Attach + 1

        # END OF THE BIG IF
        # save the state
        # occupancy must count occupied sites. If all occupied sites are "1" then a simple sum is enough
        # If we want to count both ones and twos we do "(site and 1)" because (0 and 1 = 0), (1 and 1 = 1) and (2 and 1 = 1)
        # If we want to count just ones we do "(site % 2)" because (0 mod 2 = 0), (1 mod 2 = 1) and (2 mod 2 = 0)
        # If we want to count just twos we do "(site // 2)" because (0 // 2 = 0), (1 // 2 = 0) and (2 // 2 = 1)
        for m in range(list_of_lasts[t], (list_of_firsts[t]+1)):
            occupancy[m] = occupancy[m] + (motors_walk_curr[m] and 1)
            occupancy1[m] = occupancy1[m] + (motors_walk_curr[m] % 2)
            occupancy2[m] = occupancy2[m] + (motors_walk_curr[m] // 2)
            current[m] = current[m] + ((motors_walk_curr[m-1] and 1) * (1 - (motors_walk_curr[m] and 1)))
            current1[m] = current1[m] + ((motors_walk_curr[m-1] % 2) * (1 - (motors_walk_curr[m] % 2)))
            current2[m] = current2[m] + ((motors_walk_curr[m-1] // 2) * (1 - (motors_walk_curr[m] // 2)))
            localsum = localsum + (motors_walk_curr[m] and 1)
            motors_walk_ante[m] = motors_walk_curr[m]

        list_of_lengths[t] = list_of_firsts[t] - list_of_lasts[t] + 1

    # Next part is similar to Parmeggiani_MCdynamics_LS-noODE.py but without
    # the fancy data structure (no animation here, so no need to store each state)
    # compared to Parmeggiani version we must update Detach/Attach rate while they vary with length
        #omega_D = omega_D_noN/list_of_lengths[t]
        #omega_A = omega_A_noN/list_of_lengths[t]

        #occupancy1 = occupancy1 + (motors_walk_curr[list_of_lasts[t]] and 1)
        #occupancy2 = occupancy2 + (motors_walk_curr[list_of_lasts[t] + 1] and 1)
        #rho1 = occupancy1 / overall_time_step_count
        #rho2 = occupancy2 / overall_time_step_count
        #C_din_email = gamma * (1.0-rho2) * (rho1-1.0) + delta*rho1*rho2 + 1.0
        #C_din_email_avg += C_din_email
        #rho = (occupancy[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]
        #total_current[t] = rho * (1.0-rho) - rho * (1.0-C_din_email)
        total_density[t] = (occupancy[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]
        total_density1[t] = (occupancy1[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]
        total_density2[t] = (occupancy2[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]

        total_current[t] = total_density[t] * (1.0 - total_density[t])

    # here we write to a text file the overall density and the overall current as they are now, at this time step
        # not at EVERY step - but can be changed if better resolution needed, impacts the running time
        if ((t % 100) == 0):
            f.write(f"{t} {list_of_lengths[t]} {total_density[t]} {total_current[t]}")
            f.write("\n")

    # this should never happen
        if (localsum != (count1InjRight + count2InjLeft + count1Attach + count2Attach - count1Detach - count2Detach - count1EjLeft - count2EjRight - count1Limbo - count2Limbo)):
            print(f"oups @ step {overall_time_step_count}!")
```

Figure D.17

```
        print(f"oups @ step {overall_time_step_count}!")
        print(f"({overall_time_step_count:10d})")
        print(f"{localsum:6d} = {count1InjRight:6d} + {count1Attach:6d} - {count1Detach:6d} - {count1EjLeft:6d} - {count1Limbo:6d}")
        #print("             +       +       -       -       -  ")
        print(f"         {count2InjLeft:6d}   {count2Attach:6d}   {count2Detach:6d}   {count2EjRight:6d}   {count2Limbo:6d}")

# this should better happen
    if ((t % 3000) == 0):
        print(f"({overall_time_step_count:10d}): ", end="")
        for m in range(list_of_lasts[t], (list_of_lasts[t]+21)):
            print(f"{motors_walk_curr[m]}", end ="")
        print(" ... ", end =" ")
        for m in range(list_of_firsts[t]-20, (list_of_firsts[t]+1)):
            print(f"{motors_walk_curr[m]}", end ="")
        print(" ")
        print(f"{localsum:6d} = {count1InjRight:6d} + {count1Attach:6d} - {count1Detach:6d} - {count1EjLeft:6d} - {count1Limbo:6d}")
        #print("             +       +       -       -       -  ")
        print(f"         {count2InjLeft:6d}   {count2Attach:6d}   {count2Detach:6d}   {count2EjRight:6d}   {count2Limbo:6d}")


    if ((t % 10000) == 0):
        total_time = (time.time() - start_time)
        print(f"Reaching timestep {t} in {total_time} seconds")
        print(f"microtubule (from {list_of_lasts[t]} to {list_of_firsts[t]}) -> len = {list_of_lengths[t]}")
        print("(time  step):")
        print(" totSum = <Right1 + Attach1 - Detach1 -  <Left1 -  Limbo1")
        print("          >Right2 + Attach2 - Detach2 -  >Left2 -  Limbo2")

f.close()

print(f"Last density:{total_density[overall_time_step_count-1]}")
print(f"Last current:{total_current[overall_time_step_count-1]}")

# functions used to draw just the useful part
# from https://stackoverflow.com/questions/47269390/numpy-how-to-find-first-non-zero-value-in-every-column-of-a-numpy-array
def last_nonzero(arr, axis=0, invalid_val=-1):
    mask = arr!=0
    return np.where(mask.any(axis=axis), mask.argmax(axis=axis), invalid_val)
def first_nonzero(arr, axis=0, invalid_val=-1):
    mask = arr!=0
    val = arr.shape[axis] - np.flip(mask, axis=axis).argmax(axis=axis) - 1
    return np.where(mask.any(axis=axis), val, invalid_val)

#current = (occupancy/overall_time_step_count) * (1-(occupancy/overall_time_step_count))
#C_din_email_avg /= overall_time_step_count
#current = (occupancy/overall_time_step_count) * (C_din_email_avg - (occupancy/overall_time_step_count))
#current1 = (occupancy1/overall_time_step_count) * (C_din_email_avg - (occupancy1/overall_time_step_count))
#current2 = (occupancy2/overall_time_step_count) * (C_din_email_avg - (occupancy2/overall_time_step_count))

# prepare data for file save
A = np.linspace(0, 1, len_total).reshape(len_total,1)
B = occupancy[:].reshape(len_total,1)/overall_time_step_count
B1 = occupancy1[:].reshape(len_total,1)/overall_time_step_count
B2 = occupancy2[:].reshape(len_total,1)/overall_time_step_count
C = current[:].reshape(len_total,1)
C1 = current1[:].reshape(len_total,1)
C2 = current2[:].reshape(len_total,1)

Arev = A[:] #no need to reverse here - we use right indexes in plot
Brev = B[::-1]
Brev1 = B1[::-1]
Brev2 = B2[::-1]
Crev = C[::-1]
Crev1 = C1[::-1]
```

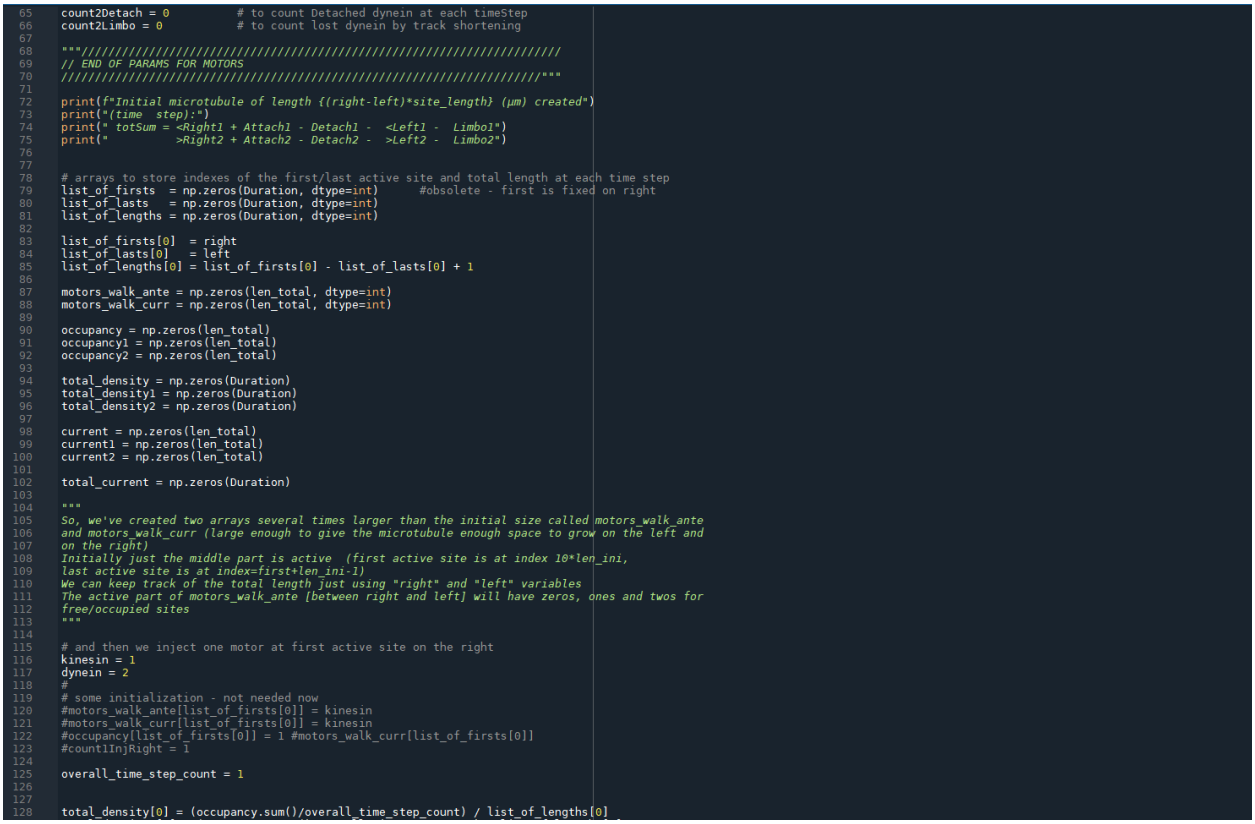
Figure D.18

```
Crev2 = C2[::-1]

ready_to_save = np.hstack((A, B, B1, B2, C, C1, C2))

if(NO_LENGTH_VARIATION):
    np.savetxt(f"RtoL_occup_of_fix_{len_ini}_sites_after_{real_Duration}_minutes.txt", ready_to_save, header="position  occupancy  occupancy1 occupancy2  current  current1
else:
    np.savetxt(f"RtoL_occup_of_var_{len_ini}_sites_after_{real_Duration}_minutes.txt", ready_to_save, header="position  occupancy  occupancy1 occupancy2  current  current1

if True:
    fig, [[ax1, ax3], [ax2, ax4], [ax5, ax6]] = plt.subplots(nrows=3, ncols=2)
    first_to_plot = first_nonzero(occupancy)-2   # replace with right for full range
    if first_to_plot < 0:
        first_to_plot = 0
    last_to_plot = last_nonzero(occupancy)-2    # replace with 0 for full range
    if last_to_plot > len_total:
        last_to_plot = len_total
    ax1.plot(A[last_to_plot:first_to_plot], B[last_to_plot:first_to_plot], color = "royalblue")
    ax1.plot(A[last_to_plot:first_to_plot], B1[last_to_plot:first_to_plot], color = "indianred")
    ax1.plot(A[last_to_plot:first_to_plot], B2[last_to_plot:first_to_plot], color = "forestgreen")
    #ax1.plot(Arev[0:len_total-last_to_plot], Brev[0:len_total-last_to_plot], color = "indianred")
    ax1.set_title("occupancy along the track")
    #ax1.set_xlabel("site position (normalized)")
    ax1.set_xticks([])
    ax1.set_ylabel("occupancy")

    ax2.plot(A[last_to_plot:first_to_plot], current[last_to_plot:first_to_plot], color = "royalblue")
    ax2.plot(A[last_to_plot:first_to_plot], current1[last_to_plot:first_to_plot], color = "indianred")
    ax2.plot(A[last_to_plot:first_to_plot], current2[last_to_plot:first_to_plot], color = "forestgreen")
    #ax2.plot(Arev[0:len_total-last_to_plot], Crev[0:len_total-last_to_plot], color = "indianred")
    ax2.set_title("current along the track")
    ax2.set_xlabel("site position (normalized)")
    ax2.set_ylabel("current")

    ax3.plot(total_density)
    ax3.set_title("total density in time")
    #ax3.set_xlabel("site position (normalized)")
    ax3.set_xticks([])
    ax3.set_ylabel("total density")

    ax4.plot(total_current)
    ax4.set_title("total current in time")
    ax4.set_xlabel("time")
    ax4.set_ylabel("total current")

    ax6.plot(list_of_lengths)
    ax6.set_title("length in time")
    ax6.set_xlabel("time")
    ax6.set_ylabel("length")

    plt.show()
```

Figure D.19

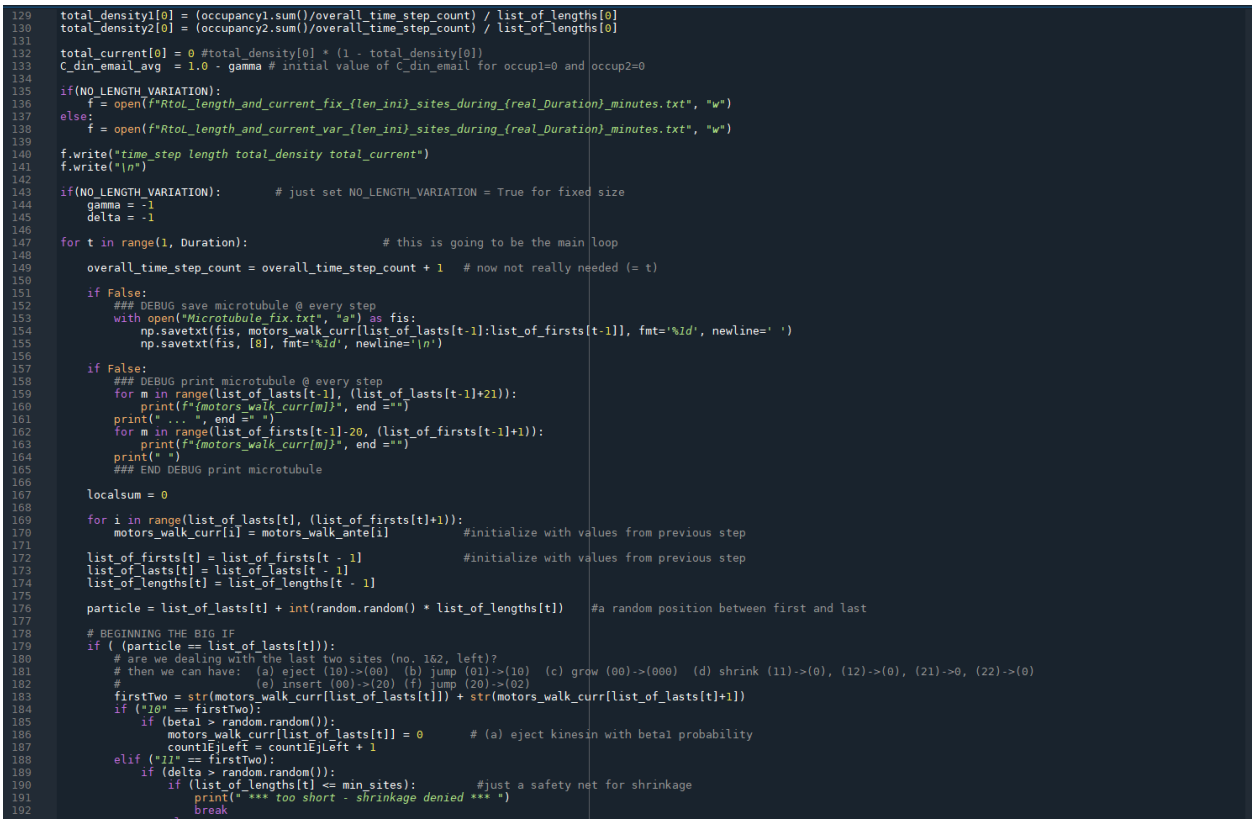
```
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 26 22:42:15 2022
@author: Laurențiu STOLERIU

two species with new rules

"""

import random
import numpy as np
import matplotlib.pyplot as plt
import time
start_time = time.time()

"""//////////////////////////////////////////////////////////////////////////
// PARAMETERS  FOR  TRACK  CONTROL                                        """
len_ini = 100                   # starting length of microtubule (no.of sites)
site_length = 4.0e-4             # site length in µm (len_ini*site_length = lenght_ini in µm)
max_length = 2.0                 # max length in µm
max_sites = 100000 #int(max_length / site_length)       #max length in no. of sites
min_sites = 30

NO_LENGTH_VARIATION = True      # just set NO_LENGTH_VARIATION = True for fixed size

real_Duration = 10              # duration in minutes
real_Tau = 0.0002                   # time step in minutes
Duration = int(real_Duration / real_Tau)     # duration in no, of time steps

len_total = 11*len_ini       # enough space in both left and right
left = 10*len_ini            # (former first) left starts at 10*len_ini to leave room for growth towards 0
right = left + len_ini - 1   # (former last) right fixed at the end of the array
"""//////////////////////////////////////////////////////////////////////////
// END OF PARAMS FOR TRACK
//////////////////////////////////////////////////////////////////////////
// PARAMETERS  FOR  MOTORS                                                """
omegaD1_noN = 0.0                       # omega_D = omega_D_noN / length  [PRL90(8)2003]
omegaD1 = omegaD1_noN / site_length
omegaA1_noN = 1.0 * omegaD1_noN         # K = omega_A / omega_D = 3  [PRL90(8)2003]
omegaA1 = omegaA1_noN / site_length

omegaD2_noN = 0.0                       # omega_D = omega_D_noN / length  [PRL90(8)2003]
omegaD2 = omegaD2_noN / site_length
omegaA2_noN = 1.0 * omegaD2_noN         # K = omega_A / omega_D = 3  [PRL90(8)2003]
omegaA2 = omegaA2_noN / site_length

alpha1 = 0.0                            # injection rate for kinesin at right-hand boundary  [PRL90(8)2003]
beta1  = 0.3                            # ejection rate for kinesin at left-hand boundary  [PRL90(8)2003]

alpha2 = 0.2                            # injection rate for dynein at left-hand boundary
beta2  = 0.0                            # ejection rate for dynein at right-hand boundary

gamma = 0.1                             # growth rate at left-hand boundary
delta = 0.1                             # shrinkage rate at left-hand boundary

count1InjRight = 0        # to count Injected kinesin at each timeStep
count1EjLeft = 0          # to count Ejected kinesin at each timeStep
count1Attach = 0          # to count Attached kinesin at each timeStep
count1Detach = 0          # to count Detached kinesin at each timeStep
count1Limbo = 0           # to count lost kinesin by track shortening

count2InjLeft = 0         # to count Injected dynein at each timeStep
count2EjRight = 0         # to count Ejected dynein at each timeStep
count2Attach = 0          # to count Attached dynein at each timeStep
```

Figure D.20



```
count2Detach = 0          # to count Detached dynein at each timeStep
count2Limbo = 0           # to count lost dynein by track shortening

"""//////////////////////////////////////////////////////////////////////////
// END OF PARAMS FOR MOTORS
//////////////////////////////////////////////////////////////////////////"""

print(f"Initial microtubule of length {(right-left)*site_length} (μm) created")
print("(time  step):")
print(" totSum = <Right1 + Attach1 - Detach1 -  <Left1 -  Limbo1")
print("          >Right2 + Attach2 - Detach2 -  >Left2 -  Limbo2")


# arrays to store indexes of the first/last active site and total length at each time step
list_of_firsts  = np.zeros(Duration, dtype=int)      #obsolete - first is fixed on right
list_of_lasts   = np.zeros(Duration, dtype=int)
list_of_lengths = np.zeros(Duration, dtype=int)

list_of_firsts[0]  = right
list_of_lasts[0]   = left
list_of_lengths[0] = list_of_firsts[0] - list_of_lasts[0] + 1

motors_walk_ante = np.zeros(len_total, dtype=int)
motors_walk_curr = np.zeros(len_total, dtype=int)

occupancy = np.zeros(len_total)
occupancy1 = np.zeros(len_total)
occupancy2 = np.zeros(len_total)

total_density = np.zeros(Duration)
total_density1 = np.zeros(Duration)
total_density2 = np.zeros(Duration)

current = np.zeros(len_total)
current1 = np.zeros(len_total)
current2 = np.zeros(len_total)

total_current = np.zeros(Duration)

"""
So, we've created two arrays several times larger than the initial size called motors_walk_ante
and motors_walk_curr (large enough to give the microtubule enough space to grow on the left and
on the right)
Initially just the middle part is active  (first active site is at index 10*len_ini,
last active site is at index=first+len_ini-1)
We can keep track of the total length just using "right" and "left" variables
The active part of motors_walk_ante [between right and left] will have zeros, ones and twos for
free/occupied sites
"""

# and then we inject one motor at first active site on the right
kinesin = 1
dynein = 2
#
# some initialization - not needed now
#motors_walk_ante[list_of_firsts[0]] = kinesin
#motors_walk_curr[list_of_firsts[0]] = kinesin
#occupancy[list_of_firsts[0]] = 1 #motors_walk_curr[list_of_firsts[0]]
#count1InjRight = 1

overall_time_step_count = 1


total_density[0] = (occupancy.sum()/overall_time_step_count) / list_of_lengths[0]
```

Figure D.21



```
    total_density1[0] = (occupancy1.sum()/overall_time_step_count) / list_of_lengths[0]
    total_density2[0] = (occupancy2.sum()/overall_time_step_count) / list_of_lengths[0]

    total_current[0] = 0 #total_density[0] * (1 - total_density[0])
    C_din_email_avg  = 1.0 - gamma # initial value of C_din_email for occup1=0 and occup2=0

    if(NO_LENGTH_VARIATION):
        f = open(f"RtoL_length_and_current_fix_{len_ini}_sites_during_{real_Duration}_minutes.txt", "w")
    else:
        f = open(f"RtoL_length_and_current_var_{len_ini}_sites_during_{real_Duration}_minutes.txt", "w")

    f.write("time_step length total_density total_current")
    f.write("\n")

    if(NO_LENGTH_VARIATION):        # just set NO_LENGTH_VARIATION = True for fixed size
        gamma = -1
        delta = -1

    for t in range(1, Duration):                    # this is going to be the main loop

        overall_time_step_count = overall_time_step_count + 1   # now not really needed (= t)

        if False:
            ### DEBUG save microtubule @ every step
            with open("Microtubule_fix.txt", "a") as fis:
                np.savetxt(fis, motors_walk_curr[list_of_lasts[t-1]:list_of_firsts[t-1]], fmt='%1d', newline=' ')
                np.savetxt(fis, [8], fmt='%1d', newline='\n')

        if False:
            ### DEBUG print microtubule @ every step
            for m in range(list_of_lasts[t-1], (list_of_lasts[t-1]+21)):
                print(f"{motors_walk_curr[m]}", end ="")
            print(" ... ", end =" ")
            for m in range(list_of_firsts[t-1]-20, (list_of_firsts[t-1]+1)):
                print(f"{motors_walk_curr[m]}", end ="")
            print(" ")
            ### END DEBUG print microtubule

        localsum = 0

        for i in range(list_of_lasts[t], (list_of_firsts[t]+1)):
            motors_walk_curr[i] = motors_walk_ante[i]           #initialize with values from previous step

        list_of_firsts[t] = list_of_firsts[t - 1]               #initialize with values from previous step
        list_of_lasts[t] = list_of_lasts[t - 1]
        list_of_lengths[t] = list_of_lengths[t - 1]

        particle = list_of_lasts[t] + int(random.random() * list_of_lengths[t])    #a random position between first and last

        # BEGINNING THE BIG IF
        if ( (particle == list_of_lasts[t])):
            # are we dealing with the last two sites (no. 1&2, left)?
            # then we can have:  (a) eject (10)->(00)  (b) jump (01)->(10)  (c) grow (00)->(000)  (d) shrink (11)->(0), (12)->(0), (21)->0, (22)->(0)
            #                    (e) insert (00)->(20) (f) jump (20)->(02)
            firstTwo = str(motors_walk_curr[list_of_lasts[t]]) + str(motors_walk_curr[list_of_lasts[t]+1])
            if ("10" == firstTwo):
                if (beta1 > random.random()):
                    motors_walk_curr[list_of_lasts[t]] = 0       # (a) eject kinesin with beta1 probability
                    countlEjLeft = countlEjLeft + 1
            elif ("11" == firstTwo):
                if (delta > random.random()):
                    if (list_of_lengths[t] <= min_sites):         #just a safety net for shrinkage
                        print(" *** too short - shrinkage denied *** ")
                        break
```

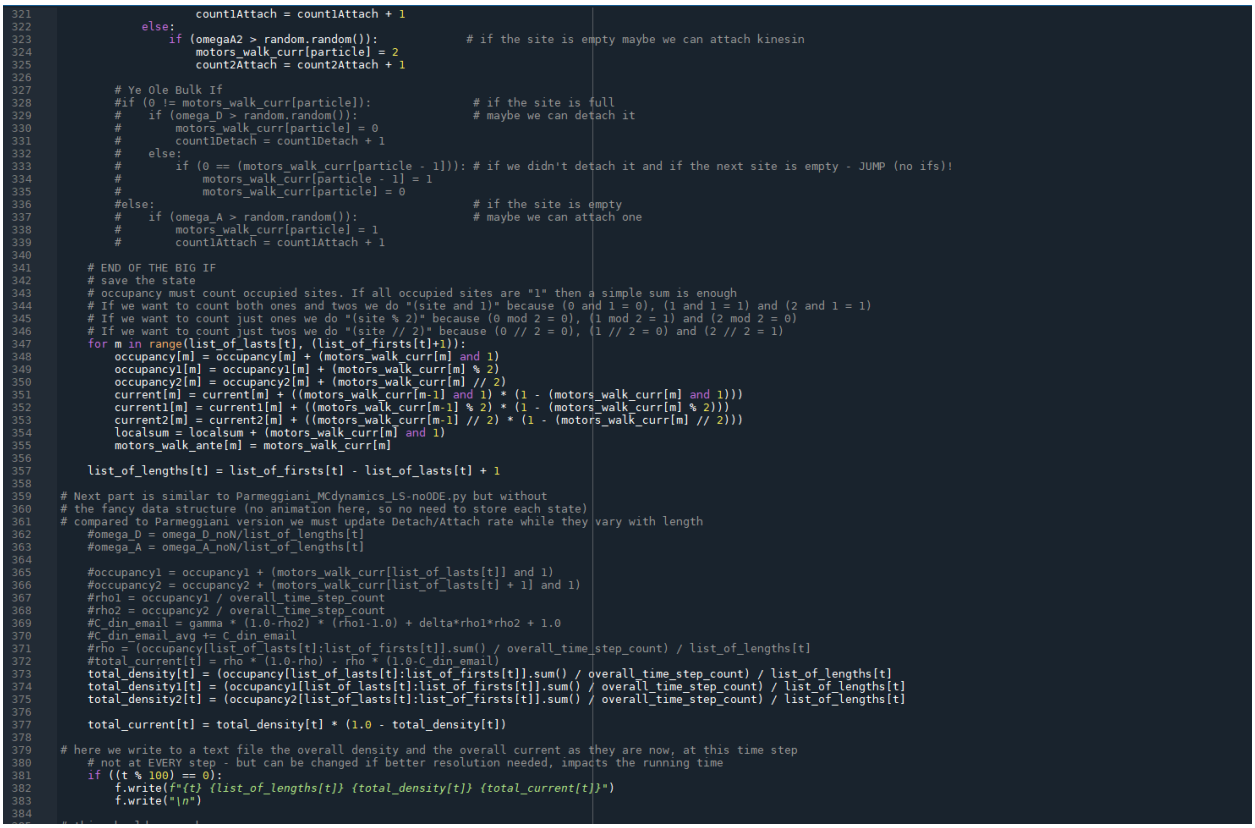
Figure D.22

```
                else:
                    count1Limbo = count1Limbo + 2
                    motors_walk_curr[list_of_lasts[t]] = 0     # (d) shrink with delta probability
                    motors_walk_curr[list_of_lasts[t] + 1] = 0
                    list_of_lasts[t] = list_of_lasts[t] + 1
        elif ("12" == firstTwo) or ("21" == firstTwo):
            if (delta > random.random()):
                if (list_of_lengths[t] <= min_sites):          # just a safety net for shrinkage
                    print(" *** too short - shrinkage denied *** ")
                    break
                else:
                    count1Limbo = count1Limbo + 1
                    count2Limbo = count2Limbo + 1
                    motors_walk_curr[list_of_lasts[t]] = 0     # (d) shrink with delta probability
                    motors_walk_curr[list_of_lasts[t] + 1] = 0
                    list_of_lasts[t] = list_of_lasts[t] + 1
        elif ("22" == firstTwo):
            if (delta > random.random()):
                if (list_of_lengths[t] <= min_sites):          # just a safety net for shrinkage
                    print(" *** too short - shrinkage denied *** ")
                    break
                else:
                    count2Limbo = count2Limbo + 2
                    motors_walk_curr[list_of_lasts[t]] = 0     # (d) shrink with delta probability
                    motors_walk_curr[list_of_lasts[t] + 1] = 0
                    list_of_lasts[t] = list_of_lasts[t] + 1
        elif ("20" == firstTwo):
            motors_walk_curr[list_of_lasts[t]]   = 0       # (f) JUMP dynein
            motors_walk_curr[list_of_lasts[t]+1] = 2
        elif ("01" == firstTwo):
            motors_walk_curr[list_of_lasts[t]]   = 1       # (b) JUMP kinesin
            motors_walk_curr[list_of_lasts[t]+1] = 0
        elif ("00" == firstTwo):
            if (alpha2 > random.random()):
                motors_walk_curr[list_of_lasts[t]] = 2         # (e) inject dynein with alpha2 probability
                count2InjLeft = count2InjLeft + 1
            else:
                if (gamma > random.random()):
                    if (list_of_lengths[t] >= max_sites):       # just a safety net for growing
                        print(" *** too large - growth denied *** ")
                        break
                    else:
                        list_of_lasts[t] = list_of_lasts[t] - 1     # (c) grow with gamma probability
                        motors_walk_curr[list_of_lasts[t]] = 0

        # Ye Ole Left End If
        #if (1 == motors_walk_curr[list_of_lasts[t]]):         # if the first site is kinesin
        #    if (0 == motors_walk_curr[list_of_lasts[t]+1]):   # and the next is free
        #        if (beta1 > random.random()):
        #            motors_walk_curr[list_of_lasts[t]] = 0    # (a) eject with beta1 probability
        #            count1EjLeft = count1EjLeft + 1
        #    else:
        #        if (delta > random.random()):
        #            if (list_of_lengths[t] <= min_sites):       #just a safety net for shrinkage
        #                print(" *** too short - shrinkage denied *** ")
        #                break
        #            else:
        #                count1Limbo = count1Limbo + 2
        #                motors_walk_curr[list_of_lasts[t]] = 0     # (d) shrink with delta probability
        #                motors_walk_curr[list_of_lasts[t] + 1] = 0
        #                list_of_lasts[t] = list_of_lasts[t] + 1
        #else:
        #    if (0 == motors_walk_curr[list_of_lasts[t]+1]):
        #        if (gamma > random.random()):
```

Figure D.23

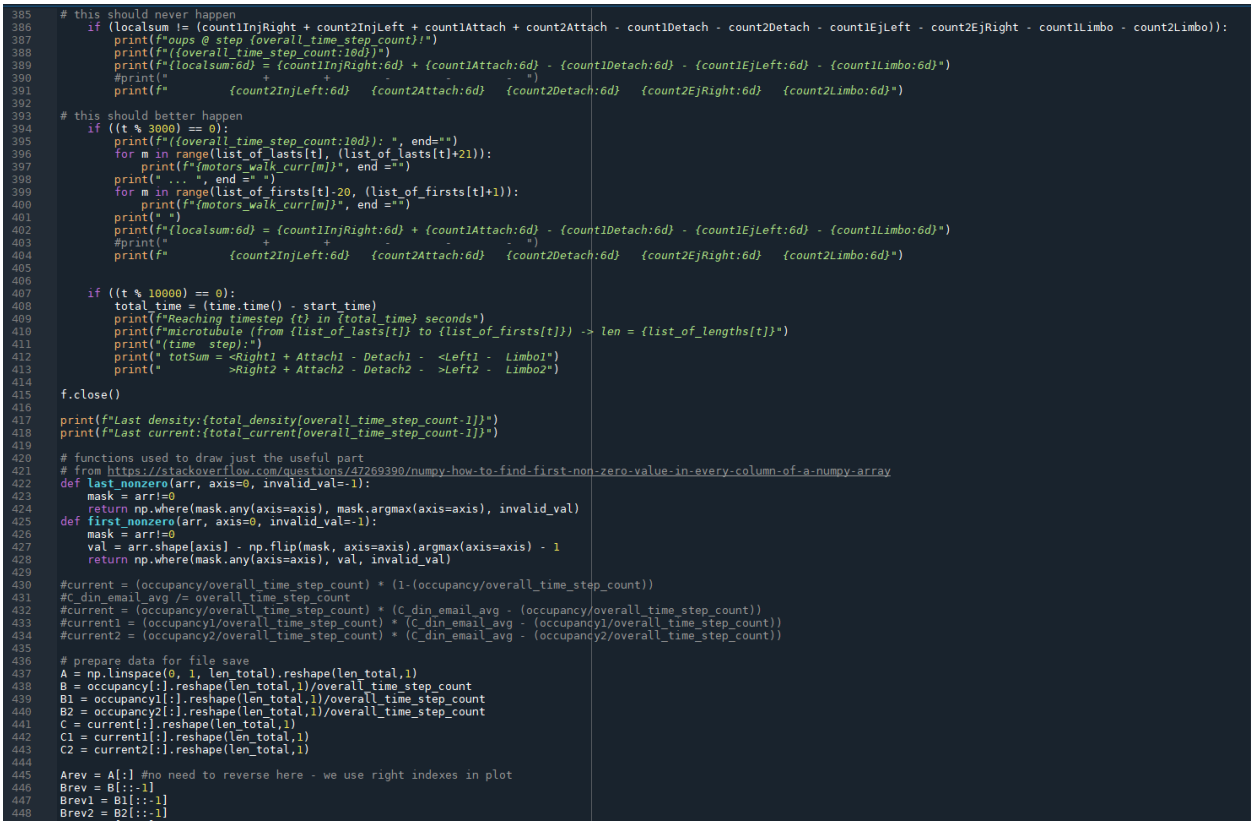
```
            #            if (list_of_lengths[t] >= max_sites):       #just a safety net for growing
            #                print(" *** too large - growth denied *** ")
            #                break
            #            else:
            #                list_of_lasts[t] = list_of_lasts[t] - 1#   (c) grow with gamma probability
            #                motors_walk_curr[list_of_lasts[t]] = 0     # add a new one to the left and initialize it with 0
            #    else:
            #        motors_walk_curr[list_of_lasts[t]]   = 1       # (b) JUMP
            #        motors_walk_curr[list_of_lasts[t]+1] = 0

        elif (particle == list_of_firsts[t]):
            # are we dealing with the first site (no. N, right)?
            lastTwo = str(motors_walk_curr[list_of_firsts[t]-1]) + str(motors_walk_curr[list_of_firsts[t]])
            if ("00" == lastTwo):
                if (alpha1 > random.random()):
                    motors_walk_curr[list_of_firsts[t]] = 1         # inject kinesin with alpha1 probability
                    count1InjRight = count1InjRight + 1
            elif ("01" == lastTwo):                                 # JUMP kinesin
                motors_walk_curr[list_of_firsts[t]-1] = 1
                motors_walk_curr[list_of_firsts[t]] = 0
            elif ("02" == lastTwo) or ("12" == lastTwo):
                if (beta2 > random.random()):
                    motors_walk_curr[list_of_firsts[t]] = 0         # (a) eject dynein with beta1 probability
                    count2EjRight = count2EjRight + 1
            elif ("20" == lastTwo):                                 # JUMP dynein (last jump - reaching end)
                motors_walk_curr[list_of_firsts[t]-1] = 0
                motors_walk_curr[list_of_firsts[t]] = 2
            elif ("21" == lastTwo):                                 # this could be much better implemented
                motors_walk_curr[list_of_firsts[t]-1] = 1
                motors_walk_curr[list_of_firsts[t]] = 2

            # Ye Ole Right End If
            #if (0 == motors_walk_curr[list_of_firsts[t]]):       # if the first site is empty
            #    if (alpha1 > random.random()):
            #        motors_walk_curr[list_of_firsts[t]] = 1      # inject with alpha1 probability
            #        count1InjRight = count1InjRight + 1
            #elif (0 == motors_walk_curr[list_of_firsts[t]-1]):   # if next is empty JUMP
            #    motors_walk_curr[list_of_firsts[t]-1] = motors_walk_curr[list_of_firsts[t]]
            #    motors_walk_curr[list_of_firsts[t]] = 0

        else:
            # now we know that we're dealing with a bulk site - no first or last two
            if (1 == motors_walk_curr[particle]):
                if (omegaD1 > random.random()):                     # maybe we can detach kinesin
                    motors_walk_curr[particle] = 0
                    count1Detach = count1Detach + 1
                else:
                    if (1 != (motors_walk_curr[particle - 1])): # if we didn't detach it and if the next site is somethingelse - JUMP (no ifs)!
                        motors_walk_curr[particle] = motors_walk_curr[particle - 1]
                        motors_walk_curr[particle - 1] = 1

            elif (2 == motors_walk_curr[particle]):
                if (omegaD2 > random.random()):                     # maybe we can detach dynein
                    motors_walk_curr[particle] = 0
                    count2Detach = count2Detach + 1
                else:
                    if (2 != (motors_walk_curr[particle + 1])): # if we didn't detach it and if the next site is empty - JUMP (no ifs)!
                        motors_walk_curr[particle] = motors_walk_curr[particle + 1]
                        motors_walk_curr[particle + 1] = 2

            else:                                                   # i.e. when is zero
                if (0.5 > random.random()):                         # select kinesin OR dyenin to test attach
                    if (omegaA1 > random.random()):                 # if the site is empty maybe we can attach kinesin
                        motors_walk_curr[particle] = 1
```

Figure D.24



```
                    count1Attach = count1Attach + 1
            else:
                if (omegaA2 > random.random()):             # if the site is empty maybe we can attach kinesin
                    motors_walk_curr[particle] = 2
                    count2Attach = count2Attach + 1

        # Ye Ole Bulk If
        #if (0 != motors_walk_curr[particle]):               # if the site is full
        #    if (omega_D > random.random()):                 # maybe we can detach it
        #        motors_walk_curr[particle] = 0
        #        count1Detach = count1Detach + 1
        #    else:
        #        if (0 == (motors_walk_curr[particle - 1])): # if we didn't detach it and if the next site is empty - JUMP (no ifs)!
        #            motors_walk_curr[particle - 1] = 1
        #            motors_walk_curr[particle] = 0
        #else:                                               # if the site is empty
        #    if (omega_A > random.random()):                 # maybe we can attach one
        #        motors_walk_curr[particle] = 1
        #        count1Attach = count1Attach + 1

    # END OF THE BIG IF
    # save the state
    # occupancy must count occupied sites. If all occupied sites are "1" then a simple sum is enough
    # If we want to count both ones and twos we do "(site and 1)" because (0 and 1 = 0), (1 and 1 = 1) and (2 and 1 = 1)
    # If we want to count just ones we do "(site % 2)" because (0 mod 2 = 0), (1 mod 2 = 1) and (2 mod 2 = 0)
    # If we want to count just twos we do "(site // 2)" because (0 // 2 = 0), (1 // 2 = 0) and (2 // 2 = 1)
    for m in range(list_of_lasts[t], (list_of_firsts[t]+1)):
        occupancy[m] = occupancy[m] + (motors_walk_curr[m] and 1)
        occupancy1[m] = occupancy1[m] + (motors_walk_curr[m] % 2)
        occupancy2[m] = occupancy2[m] + (motors_walk_curr[m] // 2)
        current[m] = current[m] + ((motors_walk_curr[m-1] and 1) * (1 - (motors_walk_curr[m] and 1)))
        current1[m] = current1[m] + ((motors_walk_curr[m-1] % 2) * (1 - (motors_walk_curr[m] % 2)))
        current2[m] = current2[m] + ((motors_walk_curr[m-1] // 2) * (1 - (motors_walk_curr[m] // 2)))
        localsum = localsum + (motors_walk_curr[m] and 1)
        motors_walk_ante[m] = motors_walk_curr[m]

    list_of_lengths[t] = list_of_firsts[t] - list_of_lasts[t] + 1

# Next part is similar to Parmeggiani_MCdynamics_LS-noODE.py but without
# the fancy data structure (no animation here, so no need to store each state)
# compared to Parmeggiani version we must update Detach/Attach rate while they vary with length
    #omega_D = omega_D_noN/list_of_lengths[t]
    #omega_A = omega_A_noN/list_of_lengths[t]

    #occupancy1 = occupancy1 + (motors_walk_curr[list_of_lasts[t]] and 1)
    #occupancy2 = occupancy2 + (motors_walk_curr[list_of_lasts[t] + 1] and 1)
    #rho1 = occupancy1 / overall_time_step_count
    #rho2 = occupancy2 / overall_time_step_count
    #C_din_email = gamma * (1.0-rho2) * (rho1-1.0) + delta*rho1*rho2 + 1.0
    #C_din_email_avg += C_din_email
    #rho = (occupancy[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]
    #total_current[t] = rho * (1.0-rho) - rho * (1.0-C_din_email)
    total_density[t] = (occupancy[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]
    total_density1[t] = (occupancy1[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]
    total_density2[t] = (occupancy2[list_of_lasts[t]:list_of_firsts[t]].sum() / overall_time_step_count) / list_of_lengths[t]

    total_current[t] = total_density[t] * (1.0 - total_density[t])

# here we write to a text file the overall density and the overall current as they are now, at this time step
    # not at EVERY step - but can be changed if better resolution needed, impacts the running time
    if ((t % 100) == 0):
        f.write(f"{t} {list_of_lengths[t]} {total_density[t]} {total_current[t]}")
        f.write("\n")
```

Figure D.25



```
    # this should never happen
        if (localsum != (count1InjRight + count2InjLeft + count1Attach + count2Attach - count1Detach - count2Detach - count1EjLeft - count2EjRight - count1Limbo - count2Limbo)):
            print(f"oups @ step {overall_time_step_count}!")
            print(f"({overall_time_step_count:10d})")
            print(f"{localsum:6d} = {count1InjRight:6d} + {count1Attach:6d} - {count1Detach:6d} - {count1EjLeft:6d} - {count1Limbo:6d}")
            #print("          +         +         -         -         -  ")
            print(f"         {count2InjLeft:6d}   {count2Attach:6d}   {count2Detach:6d}   {count2EjRight:6d}   {count2Limbo:6d}")

    # this should better happen
        if ((t % 3000) == 0):
            print(f"({overall_time_step_count:10d}): ", end="")
            for m in range(list_of_lasts[t], (list_of_lasts[t]+21)):
                print(f"{motors_walk_curr[m]}", end ="")
            print(" ... ", end =" ")
            for m in range(list_of_firsts[t]-20, (list_of_firsts[t]+1)):
                print(f"{motors_walk_curr[m]}", end ="")
            print(" ")
            print(f"{localsum:6d} = {count1InjRight:6d} + {count1Attach:6d} - {count1Detach:6d} - {count1EjLeft:6d} - {count1Limbo:6d}")
            #print("          +         +         -         -         -  ")
            print(f"         {count2InjLeft:6d}   {count2Attach:6d}   {count2Detach:6d}   {count2EjRight:6d}   {count2Limbo:6d}")


        if ((t % 10000) == 0):
            total_time = (time.time() - start_time)
            print(f"Reaching timestep {t} in {total_time} seconds")
            print(f"microtubule (from {list_of_lasts[t]} to {list_of_firsts[t]}) -> len = {list_of_lengths[t]}")
            print("(time  step):")
            print(" totSum = <Right1 + Attach1 - Detach1 -  <Left1 -  Limbo1")
            print("          >Right2 + Attach2 - Detach2 -  >Left2 -  Limbo2")

    f.close()

    print(f"Last density:{total_density[overall_time_step_count-1]}")
    print(f"Last current:{total_current[overall_time_step_count-1]}")

    # functions used to draw just the useful part
    # from https://stackoverflow.com/questions/47269390/numpy-how-to-find-first-non-zero-value-in-every-column-of-a-numpy-array
    def last_nonzero(arr, axis=0, invalid_val=-1):
        mask = arr!=0
        return np.where(mask.any(axis=axis), mask.argmax(axis=axis), invalid_val)
    def first_nonzero(arr, axis=0, invalid_val=-1):
        mask = arr!=0
        val = arr.shape[axis] - np.flip(mask, axis=axis).argmax(axis=axis) - 1
        return np.where(mask.any(axis=axis), val, invalid_val)

    #current = (occupancy/overall_time_step_count) * (1-(occupancy/overall_time_step_count))
    #C_din_email_avg /= overall_time_step_count
    #current = (occupancy/overall_time_step_count) * (C_din_email_avg - (occupancy/overall_time_step_count))
    #current1 = (occupancy1/overall_time_step_count) * (C_din_email_avg - (occupancy1/overall_time_step_count))
    #current2 = (occupancy2/overall_time_step_count) * (C_din_email_avg - (occupancy2/overall_time_step_count))

    # prepare data for file save
    A = np.linspace(0, 1, len_total).reshape(len_total,1)
    B = occupancy[:].reshape(len_total,1)/overall_time_step_count
    B1 = occupancy1[:].reshape(len_total,1)/overall_time_step_count
    B2 = occupancy2[:].reshape(len_total,1)/overall_time_step_count
    C = current[:].reshape(len_total,1)
    C1 = current1[:].reshape(len_total,1)
    C2 = current2[:].reshape(len_total,1)

    Arev = A[:] #no need to reverse here - we use right indexes in plot
    Brev = B[::-1]
    Brev1 = B1[::-1]
    Brev2 = B2[::-1]
```

Figure D.26

```
    Brev2 = B2[::-1]
    Crev = C[::-1]
    Crev1 = C1[::-1]
    Crev2 = C2[::-1]

    ready_to_save = np.hstack((A, B, B1, B2, C, C1, C2))

    if(NO_LENGTH_VARIATION):
        np.savetxt(f"RtoL_occup_of_fix_{len_ini}_sites_after_{real_Duration}_minutes.txt", ready_to_save, header="position  occupancy  occupancy1 occupancy2  current  current1  cur
    else:
        np.savetxt(f"RtoL_occup_of_var_{len_ini}_sites_after_{real_Duration}_minutes.txt", ready_to_save, header="position  occupancy  occupancy1 occupancy2  current  current1  cur

    if True:
        fig, [[ax1, ax3], [ax2, ax4], [ax5, ax6]] = plt.subplots(nrows=3, ncols=2)
        first_to_plot = first_nonzero(occupancy)-2   # replace with right for full range
        if first_to_plot < 0:
            first_to_plot = 0
        last_to_plot = last_nonzero(occupancy)-2    # replace with 0 for full range
        if last_to_plot > len_total:
            last_to_plot = len_total
        ax1.plot(A[last_to_plot:first_to_plot], B[last_to_plot:first_to_plot], color = "royalblue")
        ax1.plot(A[last_to_plot:first_to_plot], B1[last_to_plot:first_to_plot], color = "indianred")
        ax1.plot(A[last_to_plot:first_to_plot], B2[last_to_plot:first_to_plot], color = "forestgreen")
        #ax1.plot(Arev[0:len_total-last_to_plot], Brev[0:len_total-last_to_plot], color = "indianred")
        ax1.set_title("occupancy along the track")
        #ax1.set_xlabel("site position (normalized)")
        ax1.set_xticks([])
        ax1.set_ylabel("occupancy")

        ax2.plot(A[last_to_plot:first_to_plot], current[last_to_plot:first_to_plot], color = "royalblue")
        ax2.plot(A[last_to_plot:first_to_plot], current1[last_to_plot:first_to_plot], color = "indianred")
        ax2.plot(A[last_to_plot:first_to_plot], current2[last_to_plot:first_to_plot], color = "forestgreen")
        #ax2.plot(Arev[0:len_total-last_to_plot], Crev[0:len_total-last_to_plot], color = "indianred")
        ax2.set_title("current along the track")
        ax2.set_xlabel("site position (normalized)")
        ax2.set_ylabel("current")

        ax3.plot(total_density)
        ax3.set_title("total density in time")
        #ax3.set_xlabel("site position (normalized)")
        ax3.set_xticks([])
        ax3.set_ylabel("total density")

        ax4.plot(total_current)
        ax4.set_title("total current in time")
        ax4.set_xlabel("time")
        ax4.set_ylabel("total current")

        ax6.plot(list_of_lengths)
        ax6.set_title("length in time")
        ax6.set_xlabel("time")
        ax6.set_ylabel("length")

        plt.show()
```

Figure D.27

Bibliography

[1] G. M. Schutz *Exactly Solvable Models in Many-Body Systems*, eds. C. Domb and J. L. Lebowitz, in Phase Transitions and Critical Phenomena Vol. 19, Academic Press, London, (2001).

[2] A. Parmeggiani, T. Franosch, and E. Frey Totally asymmetric simple exclusion process with Langmuir kinetics *Phys. Rev. E* **70** 046101 (2004).

[3] A. Parmeggiani, T. Franosch, and E. Frey Totally asymmetric simple exclusion process with Langmuir kinetics *Phys. Rev. Lett.* **90** 8 (2003).

[4] A. Kolomeisky, *Motor Proteins and Molecular Motors* , CRC Press, Taylor and Francis Group (2020).

[5] D. Chowdhury, Stochastic mechano-chemical kinetics of molecular motors: A multidisciplinary enterprise from a physicist's perspective, *Physics Reports* textbf529, 1 (2013).

[6] H. Lodish, A. Berk, C. A. Kaiser, M. Krieger, A. Bretscher, H. Ploegh, A. Amon and K. C. Martin *Molecular Cell Biology* 8th edn., W. H. Freeman and Company, N.Y. (2016).

[7] J. Howard, *Mechanics of Motor Proteins and the Cytoskeleton*, Sinauer Associates, Sunderland, (2001).

[8] H. Spohn, *Large Scale Dynamics of Interacting Particles*, Springer Verlag, New York, (1991).

[9] B. Derrida and M. Evans, in *Nonequilibrium Statistical Mechanics in One Dimension*, edited by V. Privman, Cambridge University Press, Cambridge, England, (1997) Chap. 14, pp. 277–304.

[10] P. L. Krapivsky, S. Redner and E. Ben-Naim E *A Kinetic View of Statistical Physics*, Cambridge University Press, (2010).

[11] H. Flyvbjerg, T. E. Holy and S. Leibler Stochastic Dynamics of Microtubules: A Model for Caps and Catastrophes *Phys. Rev. Lett.* **73** 2372, (1994).

[12] T. Antal, P. L. Krapivsky, S. Redner, M. Mailman and B. Chakraborty Dynamics of an Idealized Model of Microtubule Growth and Catastrophe *Phys. Rev.* E **76** 41907, (2007).

[13] T. M. Nieuwenhuizen, S. Klumpp, and R. Lipowsky, Random walks of molecular motors arising from diffusional encounters with immobilized filaments, *Phys. Rev.* E **69** 061911, (2004).

[14] S. Honoré, F. Hubert, M. Tournus, D. White, A Growth-Fragmentation Approach for Modeling Microtubule Dynamic Instability, Bull. Math. Biol. 81 (2019) 722–758. https://doi.org/10.1007/s11538-018-0531-2.

[15] H.R. Saeidi, A. Lohrasebi, K. Mahnam, External electric field effects on the mechanical properties of the

[16] RCSB Protein Data, RCSB PDB - 1TUB: TUBULIN ALPHA-BETA DIMER, ELECTRON DIFFRACTION, (n.d.). https://www.rcsb.org/structure/1TUB.

[17] J. van Haren, T. Wittmann, Microtubule Plus End Dynamics - Do We Know How Microtubules Grow?, BioEssays. 41 (2019) 1800194.

[18] I. Mazilu, G. Zamora, J. Gonzalez, A stochastic model for microtubule length dynamics, Phys. Stat. Mech. Its Appl. 389 (2010) 419–427. https://doi.org/10.1016/j.physa.2009.10.017.

[19] G.M. Cooper, G.M. Cooper, The Cell, 2nd ed., Sinauer Associates, 2000.

[20] N. Gudimchuk, A. Roll-Mecak, Watching microtubules grow one tubulin at a time, PNAS. (2019).

[21] K.J. Mickolajczyk, E.A. Geyer, T. Kim, L.M. Rice, W.O. Hancock, Direct observation of individual tubulin dimers binding to growing microtubules, Proc Natl Acad Sci U S A. 116 (2019) 7314–7322. https://doi.org/10.1073/pnas.1815823116.

[22] H. Bowne-Anderson, A. Hibbel, J. Howard, Regulation of Microtubule Growth and Catastrophe: Unifying Theory and Experiment, Trends Cell Biol. 25 (2015) 769–779. https://doi.org/10.1016/j.tcb.2015.08.009.

[23] T. Mitchison and M. Kirschner Dynamic Instability of Microtubule Growth *Nature (London)* **312**, (1984).

[24] A. Hibbel, A. Bogdanova, M. Mahamdeh, A. Jannasch, M. Storch, E. Schäffer, D. Liakopoulos, J. Howard, Kinesin Kip2 enhances microtubule growth in vitro through length-dependent feedback on polymerization and catastrophe, ELife. 4 (2015) e10542. https://doi.org/10.7554/eLife.10542.

[25] P. Ranjith, D. Lacoste, K. Mallick, J.-F. Joanny, Nonequilibrium Self-Assembly of a Filament Coupled to ATP/GTP Hydrolysis, Biophys. J. 96 (2009) 2146–2159. https://doi.org/10.1016/j.bpj.2008.12.3920.

[26] Y. Zheng, R. Sethi, L.S. Mangala, C. Taylor, J. Goldsmith, M. Wang, K. Masuda, M. Karaminejadranjbar, D. Mannion, F. Miranda, S. Herrero-Gonzalez, K. Hellner, F. Chen, A. Alsaadi, A. Albukhari, D.C. Fotso, C. Yau, D. Jiang, S. Pradeep, C. Rodriguez-Aguayo, G. Lopez-Berestein, S. Knapp, N.S. Gray, L. Campo, K.A. Myers, S. Dhar, D. Ferguson, R.C. Bast, A.K. Sood, F. von Delft, A.A. Ahmed, Tuning microtubule dynamics to enhance cancer therapy by modulating FER-mediated CRMP2 phosphorylation, Nat. Commun. 9 (2018) 476. https://doi.org/10.1038/s41467-017-02811-7.

[27] J. Dubey, N. Ratnakaran, S.P. Koushika, Neurodegeneration and microtubule dynamics: death by a thousand cuts, Front. Cell. Neurosci. 9 (2015). https://doi.org/10.3389/fncel.2015.00343.

[28] Z. Abraham, E. Hawley, D. Hayosh, V.A. Webster-Wood, O. Akkus, Kinesin and Dynein Mechanics: Measurement Methods and Research Applications, J. Biomech. Eng. 140 (2018) 0208051–02080511. https://doi.org/10.1115/1.4037886.

[29] T. Shima, M. Morikawa, J. Kaneshiro, T. Kambara, S. Kamimura, T. Yagi, H. Iwamoto, S. Uemura, H. Shigematsu, M. Shirouzu, T. Ichimura, T.M. Watanabe, R. Nitta, Y. Okada, N. Hirokawa, Kinesin-binding–triggered conformation switching of microtubules contributes to polarized transport, J. Cell Biol. 217 (2018) 4164–4183. https://doi.org/10.1083/jcb.201711178.

[30] R.D. Vale, The Molecular Motor Toolbox for Intracellular Transport, Cell. 112 (2003) 467–480. https://doi.org/10.1016/S0092-8674(03)00111-9.

[31] Q. Shao, Y.Q. Gao, On the hand-over-hand mechanism of kinesin, Proc. Natl. Acad. Sci. 103 (2006) 8072–8077. https://doi.org/10.1073/pnas.0602828103.

[32] Y. Kinoshita, T. Kambara, K. Nishikawa, M. Kaya, H. Higuchi, Step Sizes and Rate Constants of Single-headed Cytoplasmic Dynein Measured with Optical Tweezers, Sci. Rep. 8 (2018) 16333. https://doi.org/10.1038/s41598-018-34549-7.

[33] G. Bhabha, G.T. Johnson, C.M. Schroeder, R.D. Vale, How dynein moves along microtubules, Trends Biochem. Sci. 41 (2016) 94–105. https://doi.org/10.1016/j.tibs.2015.11.004.

[34] K.L. Engel, A. Arora, R. Goering, H.-Y.G. Lo, J.M. Taliaferro, Mechanisms and consequences of subcellular RNA localization across diverse cell types, Traffic. 21 (2020) 404–418. https://doi.org/10.1111/tra.12730.

[35] K.Y. Soo, M. Farg, J.D. Atkin, Molecular Motor Proteins and Amyotrophic Lateral Sclerosis, Int. J. Mol. Sci. 12 (2011) 9057–9082. https://doi.org/10.3390/ijms12129057.

[36] Cécile Leduc, Kathrin Padberg-Gehle, Vladimír Varga, Dirk Helbing, Stefan Diez, Jonathon Howard, Molecular crowding creates traffic jams of kinesin motors on microtubules — PNAS, Proc. Natl. Acad. Sci. 109 (2012) 6100–6105. https://doi.org/10.1073/pnas.1107281109.

[37] T. Craddock, D. Chopra, N. Casey, L. Goldstein, S. Hameroff, R. Tanzi, The Zinc Dyshomeostasis Hypothesis of Alzheimer's Disease, PloS One. 7 (2012) e33552.

[38] D. Peet, N. Burroughs, R.A. Cross, Kinesin Binding Expands and Stabilises the GDP-Microtubule Lattice, Biophys. J. 114 (2018) 507a. https://doi.org/10.1016/j.bpj.2017.11.2769.

[39] W.H. Roos, O. Campàs, F. Montel, G. Woehlke, J.P. Spatz, P. Bassereau, G. Cappello, Dynamic kinesin-1 clustering on microtubules due to mutually attractive interactions, Phys. Biol. 5 (2008) 046004. https://doi.org/10.1088/1478-3975/5/4/046004.

[40] N. Hirokawa, Y. Noda, Intracellular transport and kinesin superfamily proteins, KIFs: structure, function, and dynamics, Physiol Rev. 88 (2008) 1089–1118.

[41] C. Friel, J. Howard, Coupling of kinesin ATP turnover to translocation and microtubule regulation: One engine, many machines, Journal of Muscle Research and Cell Motility. 33 (2012).

[42] RCSB Protein Data Bank, RCSB PDB - 1BG2: HUMAN UBIQUITOUS KINESIN MOTOR DOMAIN, (n.d.). https://www.rcsb.org/structure/1bg2 (accessed December 27, 2021).

[43] E. Toprak, A. Yildiz, M.T. Hoffman, S.S. Rosenfeld, P.R. Selvin, Why kinesin is so processive, PNAS. 106 (2009) 12717–12722. https://doi.org/10.1073/pnas.0808396106.

[44] R. K. P. Zia, J. J. Dong, B. Schmittmann, Modeling Translation in Protein Synthesis with TASEP: A Tutorial and Recent Developments, *J Stat Phys* **144** 405, (2011).

[45] P.L. Krapivsky, S. Render, E. Ben-Naim, A Kinetic View of Statistical Physics, Cambridge University Press, 2010. https://cnls.lanl.gov/ ebn/kvsp/ (accessed February 19, 2022).

[46] K.D. Duc, Z.H. Saleem, Y.S. Song, Theoretical analysis of the distribution of isolated particles in the TASEP: Application to mRNA translation rate estimation, BioRxiv. (2017) 147017. https://doi.org/10.1101/147017.

[47] M. Liu, X. Tuo, R. Wang, R. Jiang, Recent developments in totally asymmetric simple exclusion processes with local inhomogeneity, Chin. Sci. Bull. 56 (2011) 1527. https://doi.org/10.1007/s11434-011-4449-4.

[48] M. Lavrentovich, The Totally Asymmetric Simple Exclusion Process: Applications to Molecular Motor Modeling, (n.d.) 6.

[49] C.T. MacDonald, J.H. Gibbs, A.C. Pipkin, Kinetics of biopolymerization on nucleic acid templates, Biopolymers. 6 (1968) 1–25. https://doi.org/10.1002/bip.1968.360060102.

[50] tRNAs and ribosomes (article) — Translation, Khan Academy. (n.d.). https://www.khanacademy.org/science/biology/gene-expression-central-dogma/translation-polypeptides/a/trna-and-ribosomes (accessed February 21, 2022).

[51] A.A. Van Den Berg, The interplay between polymerase organization and nucleosome occupancy on genes: How dynamic roadblocks on the DNA induce the formation of RNA polymerase pelotons, Delft University of Technology, 2017.

[52] Z. Toroczkai, R.K.P. Zia, A model for electrophoresis of polymers with impurities: Exact distribution for a steady state, Phys. Lett. A. 217 (1996) 97–103. https://doi.org/10.1016/0375-9601(96)00311-8.

[53] B. Widom, J.L. Viovy, A.D. Defontaines, Repton model of gel electrophoresis and diffusion, J. Phys. I. 1 (1991) 1759–1784. https://doi.org/10.1051/jp1:1991239.

[54] A.K. Gupta, Collective dynamics on a two-lane asymmetrically coupled TASEP with mutually interactive Langmuir Kinetics, J. Stat. Phys. 162 (2016) 1571–1586. https://doi.org/10.1007/s10955-016-1463-6.

[55] D. Chowdhury, Traffic flow of interacting self-driven particles: rails and trails, vehicles and vesicles, Phys. Scr. T106 (2003) 13. https://doi.org/10.1238/Physica.Topical.106a00013.

[56] S. Pradhan, S. Patra, Y.E. Dai, A. Schadschneider, D. Chowdhury, Flux-density relation for traffic of army ants in a 3-lane bi-directional trail, Physica A: Statistical Mechanics and Its Applications. 567 (2021) 125664. https://doi.org/10.1016/j.physa.2020.125664.

[57] C. A. Moores, R. A. Milligan, Lucky 13 - microtubule depolymerisation by kinesin-13 motors, *Journal of Cell Science.* **119**, 3905 (2006).

[58] A. Desai, S. Verma, T. J. Mitchison, C. E. Walczak, Kin I Kinesins Are Microtubule-Destabilizing Enzymes, *Cell.* **96** 69 (1999).

[59] M. K. Gardner, M. Zanic, J. Howard, Microtubule Catastrophe and Rescue, *Curr. Opin. Cell Biol.* **25**, 14 (2013).

[60] G. Woehlke, A.K. Ruby, C.L. Hart, B. Ly, N. Hom-Booher, R.D. Vale Microtubule Interaction Site of the Kinesin Motor, *Cell***90** 207, (1997).

[61] K. Sugden, M. Evans, A dynamically extending exclusion process, *J. Stat. Mech. Theory Exp.* (2007).

[62] S. Muhuri, Scale-invariant density profiles of a dynamically extending TASEP, *E. Phys. Lett.*, **101** (2013).

[63] Y. Taniguchi, M. Nishiyama, Y. Ishii, T. Yanagida, Entropy rectifies the Brownian steps of kinesin, *Nature Chemical Biology.* **1** 342 (2005).

[64] S. K. Singh, H. Pandey, J. Al-Bassam, L. Gheber, Bidirectional motility of kinesin-5 motor proteins: structural determinants, cumulative functions and physiological roles, *Cell Mol Life Sci.* textbf75 1757 (2018).

[65] Random cluster model, Wikipedia. (2021).

[66] P.V. Prudnikov, V.V. Prudnikov, E.A. Pospelov, P.N. Malyarenko, A.N. Vakilov, Aging and non-equilibrium critical phenomena in Monte Carlo simulations of 3D pure and diluted Ising models, Progress of Theoretical and Experimental Physics. 2015 (2015) 053A01.

[67] J.P. Nilmeier, G.E. Crooks, D.D.L. Minh, J.D. Chodera, Nonequilibrium candidate Monte Carlo is an efficient tool for equilibrium simulation, PNAS. 108 (2011) E1009–E1018.

[68] B. Borchers, What is local Monte Carlo simulation? - Computational Science Stack Exchange, (2013). https://scicomp.stackexchange.com/questions/8321/what-is-local-monte-carlo-simulation (accessed January 16, 2022).

[69] A. Mitra, F. Ruhnow, S. Girardo, S. Diez, Directionally biased sidestepping of Kip3/kinesin-8 is regulated by ATP waiting time and motor–microtubule interaction strength, PNAS. 115 (2018) E7950–E7959.

[70] M. Schmidt, J. Kierfeld, Chemomechanical simulation of microtubule dynamics with explicit lateral bond dynamics, Front. Phys. 9 (2021) 673875.

[71] RCSB Protein Data Bank, RCSB PDB - 3VKH: X-ray structure of a functional full-length dynein motor domain, (n.d.).

[72] S. Can, S. Lacey, M. Gur, A.P. Carter, A. Yildiz, Directionality of dynein is controlled by the angle and length of its stalk, Nature. 566 (2019) 407–410.

[73] A.G. Hendricks, J.E. Lazarus, E. Perlson, M.K. Gardner, D.J. Odde, Y.E. Goldman, E.L.F. Holzbaur, Dynein tethers and stabilizes dynamic microtubule plus ends, Curr Biol. 22 (2012) 632–637.

[74] University of Texas at Austin Physics Department,

[75] W.O. Hancock, Bidirectional cargo transport: moving beyond tug of war, Nat Rev Mol Cell Biol. 15 (2014) 615–628.

[76] Potts model, Wikipedia. (2021).

[77] W. Wang, S. Feng, Z. Ye, H. Gao, J. Lin, D. Ouyang, Prediction of lipid nanoparticles for mRNA vaccines by the machine learning algorithm, Acta Pharmaceutica Sinica B. (2021).

[78] Y. Luo, G. Jiang, T. Yu, Y. Liu, L. Vo, H. Ding, Y. Su, W.W. Qian, H. Zhao, J. Peng, ECNet is an evolutionary context-integrated deep learning framework for protein engineering, Nat Commun. 12 (2021) 5743.

[79] P. Feng, A. Xiao, M. Fang, F. Wan, S. Li, P. Lang, D. Zhao, J. Zeng, A machine learning-based framework for modeling transcription elongation, PNAS. 118 (2021).

[80] Hewlett Packard Enterprise Development LP, What is Machine Learning? – Enterprise IT Definitions, (2022).

[81] C. Furtlehner, Statistical physics methods for machine learning and traffic forecasting, thesis, Université Paris Saclay, 2020.

[82] K. Nagel, M. Schreckenberg, A cellular automaton model for freeway traffic, J. Phys. I France. 2 (1992) 2221–2229.

[83] A. Melbinger, L. Reese and E. Frey Microtubule Length Regulation by Molecular Motors *Phys. Rev. Lett.* **108** 8 (2012).

[84] V. VanBuren, L. Cassimeris, D. J. Odde, Mechanochemical Model of Microtubule Structure and Self-Assembly Kinetics, *Biophysics J* **89** (5), (2005).

[85] M. Kardar, G. Parisi, Y.-C. Zhang, Dynamic Scaling of Growing Interfaces, Phys. Rev. Lett. 56 (1986) 889–892. https://doi.org/10.1103/PhysRevLett.56.889.

[86] C.A. Tracy, The Asymmetric Simple Exclusion Process: Integrable Structure & Limit Theorems, (n.d.) 41.

[87] M. Aridor, L. A. Hannan, Traffic Jam: A compendium of human diseases that affect intracellular transport processes, *Traffic*, **1**, (2000).

[88] T. Midha, A. K. Gupta, Role of interactions and correlations on collective dynamics of molecular motors along parallel filaments, *J. Stat. Phys.* **169** (2017).

[89] A. K. Verma, N. Sharma, A. K. Gupta, On the Role of Interacting Particles and Limited Resources in the Regulation of Lattice Length Dynamics, *J. Stat. Phys.* **179** (2020).

[90] D. Panda, H. P. Miller and L. Wilson, Rapid Treadmilling of Brain Microtubules Free of Microtubule-Associated Proteins In Vitro and its Suppression by Tau *Proc. Natl Acad. Sci. USA* **96** 12459, (1999).