An Integrative Modelling Framework for Multicellular Systems

Jonathan Naylor

School of Computing Science

University of Newcastle

A thesis submitted for the degree of

Doctor of Philosophy

September 2019

Statement of authorship

I, Jonathan Naylor, declare that this thesis titled, 'An Integrative Modelling Framework for Multicellular Systems' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Newcastle, October 15, 2019 .....

Abstract

Multicellular systems exhibit complex population scale behaviour that emerge from the interactions between constituent cells. Integrative modelling (IM) techniques are a valuable tool for studying these systems capturing processes that occur at many temporal and spatial scales. The application of IM to multicellular systems is challenging as it is knowledge and resource intensive, additionally there do not exist effective frameworks or tools, inhibiting its wider application in Systems and Synthetic biology.

This thesis presents Simbiotics, a novel IM framework for the modelling of mixed species bacterial consortia. Simbiotics is a spatially explicit multi-scale modelling platform for the design, simulation and analysis of bacterial populations. A library of modules simulating features such as cell geometries, physical force dynamics, genetic circuits, metabolic pathways, chemical diffusion and cell interactions is implemented, that the modeller may compose into their own custom models. Common modelling methods such as Boolean networks, differential equations, Gillespie models and SBML are implemented. With the platform *in-silico* experiments can be conducted with programmed experiment interactions, data collection and analysis. The framework is extendable and modular, allowing for the library to be updated as knowledge progresses. A novel file format for the reuse and communication of multicellular models and simulation methods is also implemented. Additionally an intuitive graphical user interface, Easybiotics, has been developed allowing for multicellular modelling with minimal programming experience.

Four novel case studies are pursued with Simbiotics studying the emergent behaviours of multicellular systems. The effect of physical cell

interactions are characterised in the first two studies. Investigation into how chemical signalling and intracellular dynamics influence population dynamics and patterns are studied in the final two case studies. These studies demonstrate how Simbotics can be integrated into a Systems/Synthetic biology workflow, facilitating the studying of natural systems and as a CAD tool for developing novel synthetic systems.

Acknowledgements

First I would like express my gratitude to Natalio Krasnogor for giving the opportunity to work under his guidance and leadership. I am extremely grateful for his support, belief in me and my work, and for the knowledge, wisdom and vision he has passed on to me. It has been an honour to be a part of the frontier science and integrated work environment he embodies. I am also extremely grateful to my second supervisor Harold Fellermann for his support, guidance, patience, belief and tutorship - having worked so closely with Harold his emotional stability, logical approach and rigour are qualities I hope to emulate. I also thank my collaborators and their associates for their hard work and determination. Specifically Joy Mukherjee and Waleed Mohammed for the long hours they put in conducting experiments, data collection and analysis. Also to Francisco Romero-Campero for his hospitality during my visits to Sevilla. Insights into their work and discpiline, and briding the language barriers often faced in interdisciplinary contexts, taught me valuable lessons about communication and opened my eyes to wider more detailed world.

My thanks extend to Newcastle University and the School of Computing Science, for the scholarship they awarded me with to do this PhD, and for granting access to the high performance computing cluster (HPC) on which the simulations were executed. Also to the EPSRC who supported the projects, specifically with grants EP/I031642/2, EP/J004111/2, EP/L001489/2, EP/N031962/1 and the MRC (MR/N005872/1).

A special thank you goes out to Pawel Widera for his wisdom and support, the expertise he brings to the group are invaluable, and his investment in helping others achieve a high quality of work is ad-

mirable. I would also like to thank Jaume Bacardit for his help with the HPC.

Another special thank goes to the people that used the software I developed during my PhD - namely Wim Hordijk and Bradley Brown, who both provided great feedback, were a pleasure to work with, and helped me understand how to make a *tool* to help people conduct their work with.

I would like to thank those who I have seen almost weekly during my PhD - for their support and motivation, positive energies, and for helping me to keep a balanced life during this research: Srinivas Ganti and Imran Qureshi, Jurek Kozyra and Charles Winterhalter, Nunzia Lopicollo and Benjamin Shirt-Ediss, Fedor Shmarov and Yuchun Ding.

To my friends in Bradford n' that, your friendships are cherished - you are brothers and sisters to me, each with your own beauty. For his intellectual spark and influence on my scientific pursuits, I thank Joseph Walkland, without him I may never have trained the critical thinking so crucial for this work. For his wisdom, pragmatism, positive outlook and sense of humour, I thank Tom Sadeghi - "Lyyyynn!".

Last but certainly not least, I would like to thank my family - each of you has taught me valuable life lessons, shown me unwavering love, and brought joy to my life. To my mother for teaching me to always keep one foot on the ground and a happy stomach, my father for teaching me to reach for my creative dreams and showing me the intricate beauty of the universe, and my brother for teaching me how to integrate these into a balanced personality, and to take responsibility by embodying ethics into a way of life. I always saw you all smile at the world and the world seemed to smile back, I can not have asked for a better family of a more beautiful upbringing. To my grandparents, who's minds still live on within mine, your love and legacy lives on, and here I pay homage to all you did and what you were.

Contents

List of Figures

List of Tables

Part I

Chapter 1

Introduction and Motivation

This chapter introduces and motivates the research, outlining the scope, and establishing a set of aims and objectives. The contributions of this research are then described, followed by an overview of the thesis structure.

1.1 Studying complex systems

The natural world exhibits complex and vibrant behaviour that emerge from dynamics of parts existing at systems at many scales [58, 106, 173, 187]. The design of natural systems has allowed for the stacking of complexity and evolution of robust, versatile systems spanning many orders of magnitude [227]. Discernment of how nature arranges itself can be gained through the measured observation of phenomena and formalisation of our understanding into models - those models can then be used to make experimentally testable predictions, validating whether our understanding of a system is correct or not. Through this process we have decoded some of the design and regulation principles of nature, and continued investigation hopes to uncover vital strategies by which we can manage natural systems and expand our own engineering capabilities.

Many systems in nature fall under the umbrella discipline of complexity theory, which is the study of complex systems (CS). A CS is a network of many interacting parts, which establishes organisation through the ensemble of interactions between those parts [77]. Multicellular systems such as bacterial colonies are

an example of a CS, where each cell is a part that can interact with other cells thereby forming colonies and other multicellular organisations such as biofilms [53, 104]. Figure 1.1 overviews length and time scales at which processes relating to multicellular systems occur. CS can exhibit adaptive behaviour, emerging from feedback loops between large scale population organisation and small scale part

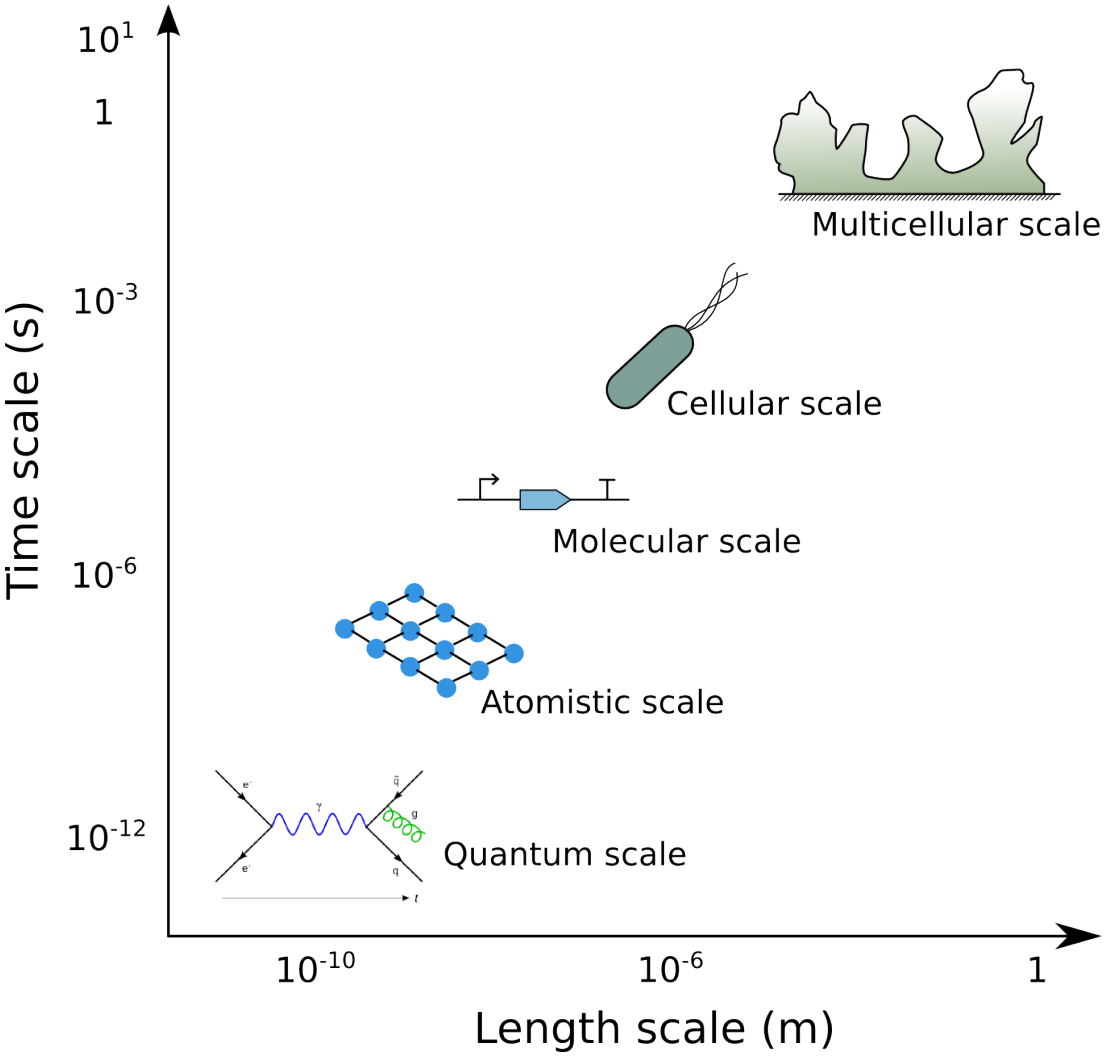


Figure 1.1: An overview of the different scales at play in multicellular systems. At the most fundamental level they are composed on quantum phemoena, which gives rise to the atomistic phenomena, and so on. Multiscale systems such as these exhibit forward and backward causality, depending feedback loops of features at many scales modulating each others behaviour.

dynamics, making it difficult to identify cause-and-effect relationships within a system [39].

The formal study of CS emerged in the 1970s [219], and has since been applied to domains such as biology, medicine, politics, and economics [83, 129, 133, 141]. Computational modelling techniques are used to facilitate in the study of CS across these domains [39, 70, 86, 119], often involving the development of multi-scale models in order to simulate the feedback between processes at small and large scales [165]. Despite the wide spread application of multi-scale methods, the development of novel models is still a technical task which requires programming expertise. Software frameworks have been developed to ease the construction of multi-scale models [25, 36], however these still require programming to develop new models in, restricting the capacity of domain experts who may not have programming expertise to engage in model building without assistance from an informatician.

1.2 Multicellular bacterial systems

Understanding how complex systems of bacteria behave is invaluable for developing solutions to a multitude of problems. Bacteria are involved in a phenom-

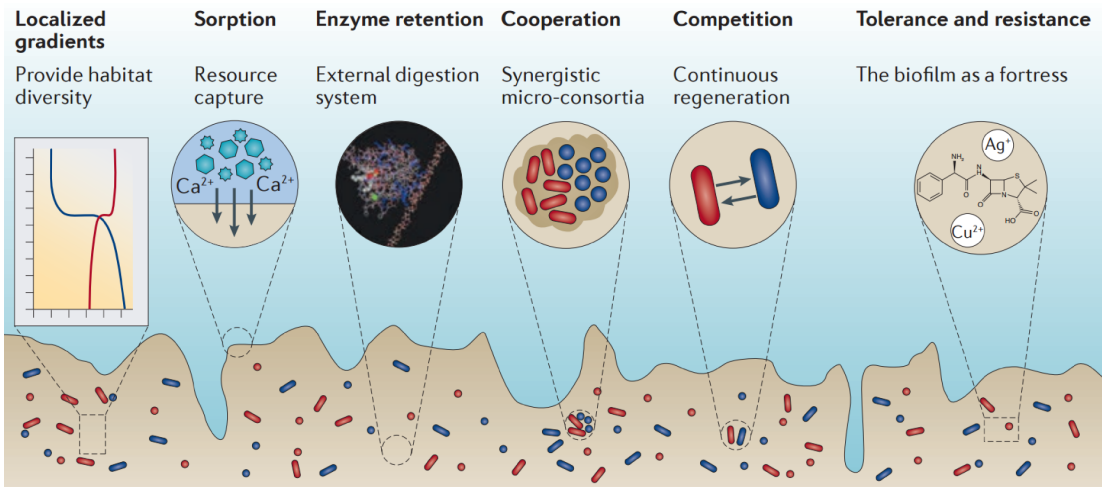


Figure 1.2: An example of some of the emergent behaviours and features of multicellular bacterial systems. Image taken from [64]

ena ranging from our digestion and immune systems, to ecological processes and industrial methods such as bioremediation [20, 43, 82, 97, 213]. Biofilms are a particularly prevalent complex community of bacteria, being one of the most widely distributed and successful modes of life on Earth [205]. They typically consist of multiple species of bacteria that form synergistic relationships, embedded in a self-secreted extracellular matrix which helps maintain homeostasis within the biofilm and regulates interactions with the environment [64]. They exhibit emergent and self-organising behaviours such as the formation of mushroom-like structures [40] and colony expansion through mechanisms such as coordinated twitching-motility [27, 190]. An overview of some biofilm features can be seen in Figure 1.2.

Research into biofilms has deduced many of processes involved in biofilm formation and development. The systems emerge from interplay between molecular interactions and the spatial organisation of cells [104, 145]. Short-range physical interactions such as membrane-mediated adhesion and cell-shoving as well as long-range chemical interactions such as quorum-sensing [153] help coordinate cells into a physical structure and cooperative gene regulatory behaviour.

1.3 Systems and Synthetic Biology

Systems Biology is the study of *life as it is*, and is concerned with the mathematical and computational modelling of biological systems [103, 121]. Systems Biology considers systems to be structured networks with dynamics between elements in the network, giving rise to functional behaviour. This approach has lead to identification of design principles of biological systems, such as the discovery of recurrent motifs in gene networks [5, 6].

Computational techniques in this domain have been successfully applied to develop a whole-cell model of *Mycoplasma genitalium* [41], allowing for phenotype prediction from genotype. Characterisation of natural systems through Systems Biology research has paved the way for understanding biological systems as mechanistic biochemical systems, and how we can isolate functional components of cells.

Synthetic Biology on the other hand is concerned with *life as it could be*, and

aims to repurpose biological components for novel applications, such as creating customised cells which carry out targeted functions [118]. Some such systems have been conceived; such as a genetic toggle switch [13, 67], synthetic gene circuits that allow either single cells or population of cells produce synchronous oscillations of gene regulation [45, 56, 66, 206], population control [235], event-triggered biofilm formation [123], and possibly most interesting - an *E. coli* mutant that can count to 3 [65].

Research in Synthetic Biology has shown that integration of large synthetic circuits into individual cells act unpredictably, and that to solve this problem the synthetic circuit should be decomposed into smaller more predictable modules, which are distributed across numerous cellular species that communicate through chemical signalling [193]. A major challenge in the development of these systems resides in their scalability and robustness, thus multi-scale models of distributed cellular devices are essential in their realisation.

1.4 Aims and Objectives

This thesis aims to advance multi-scale modelling techniques for studying multicellular systems. There exist a wide range of modelling techniques for simulating intracellular processes such as gene regulatory networks and metabolisms [99, 184]. Despite numerous multicellular modelling tools and frameworks emerging [127, 130], there is yet a flexible platform that can model a wide range of multicellular systems and be readily extended as our understanding of biological systems advances. With the emergence of bioengineering through disciplines such as Molecular, Systems, Synthetic Biology and Microbiology, the availability of easy to use multi-scale modelling tools is invaluable. Such tools can allow for low-cost *in-silico* testing of system feasibility and robustness prior to synthesis, and in the formal design and communication of bioengineered devices.

I aim to develop novel methods for modelling multicellular systems by integrating state-of-the-art techniques into a toolbox for building, simulating and analysing models. The toolbox should allow modellers to express a wide range of systems features in an intuitive manner with minimal programming experience. The toolbox framework should be implemented flexibly such that new methods

can be readily added as scientific knowledge progresses. The developed method should be applicable to both Systems and Synthetic Biology, allowing for the study of natural population dynamics, and for aiding in the design of novel distributed cellular devices.

The platform is then to be used for investigating the emergent dynamics of bacterial populations. Specifically, we will conduct an investigation into how physical and biochemical interactions between cells can effect population dynamics, and how we can design these interactions to produce directed synthetic population behaviour.

The goals of this thesis have been formalised into a two overarching aims and 2 specific objectives for each of those aims.

- **Aim 1:** Develop an easy to use, flexible and extendable workbench for integrative modelling of multicellular populations.
- **Aim 2:** Model and analyse multicellular populations patterned by physical and biochemical interactions.
- **Objective 1:** Development of an extendable modelling platform and data format which allows one to express models of interacting multi-species bacteria, simulate those systems and perform analysis.
- **Objective 2:** Development of an easy to use interface to enable those with minimal programming experience to build models of mixed consortia of bacteria.
- **Objective 3:** Study the effect of physical shoving and cell-cell adhesion on bacterial aggregation and biofilm formation.
- **Objective 4:** Study the effect of synthetic chemical signalling and gene-regulation on biophysical patterning in bacterial populations.

1.5 Structure of the thesis

The thesis is divide into two parts. Part I introduces the thesis and describes the background theory relating to multicellular bacterial systems and state of the

art modelling techniques, followed by the description of how these concepts have been integrated into a user friendly multi-scale modelling software platform. Part II includes four case study applications of the software, studying the population patterning emerging from physical and biochemical interactions. The chapter structure is described below.

Part I

Chapter 2 - Background

This chapter describes the concepts underpinning this work, and presents the relevant biological theory for modelling bacterial populations, as well as the computational techniques for modelling these. A literature review of existing modelling tools is also presented.

Chapter 3 - Simbiotics, an integrative framework for modelling multicellular populations

In this chapter the Simbiotics modelling framework is presented, describing its implementation and features. The mathematical implementation of simulation features are described here.

Chapter 4 - Validation of Simbiotics modelling features

This chapter presents the validation tests performed on Simboitics, ensuring the correctness of the implemented features. Simbiotics is also used to reproduce literature results from other simulators in this domain, ensuring the modelling features can generate previously published behave findings.

Chapter 5 - Biomodel and numerical methods representation

This chapter describes the representation of biomodels in Simbiotics, as well as how these models are mapped to numerical methods to approximate them. The file formats developed for communicating biomodels and simulation libraries is also presented.

Chapter 6 - Easybiotics, a graphical environment enabling rapid population modelling and analysis

In this chapter I introduce Easybiotics, which enables the use of Simbiotics from a graphical user interface. The modelling environment and features for facilitating in Systems and Synthetic biology contexts are described.

Part II

Chapter 7 - Study 1, Dental plaque: Bacterial coaggregation *(Published)*

In this chapter Ipresent a study conducted with Simbiotics regarding the coaggregation of bacterial species mediated by purely physical interactions.

Chapter 8 - Study 2, Synthetic E. coli biofilms

In this chapter I present a study conducted with Simbiotics regarding the influence of physical interactions on biofilm formation and structure.

Chapter 9 - Study 3, Population Dynamics of Autocatalytic Sets in a Compartmentalised Spatial World

In this chapter I present a study conducted with Simbiotics on the dynamics of populations of compartments of autocatalytic sets.

Chapter 10 - Study 4, Pattern formation via synthetic cell signalling

In this chapter I present a study conducted with Simbiotics on pattern formation based on synthetic biochemical interactions between cells.

Chapter 11 - Discussion and Conclusions

This chapter concludes the thesis, reviewing the contributions in relation to the aims and objectives. The limitations of the work, use of Simbiotics by other individuals, and the future outlook of the work are also presented.

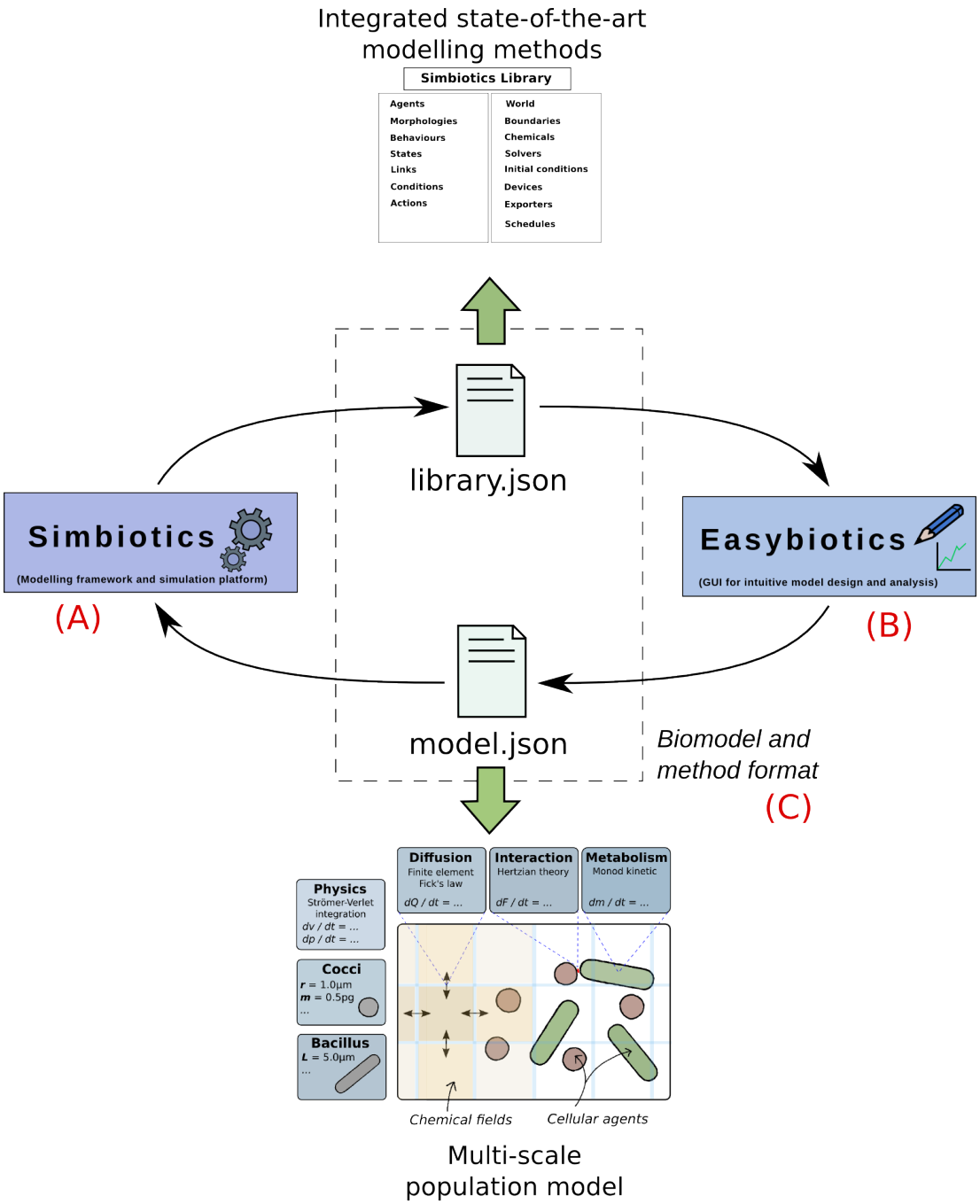


Figure 1.3: An overview of the software components contributed in my thesis, showing the relationships between the developed methods. (A), (B) and (C) are described in the Contributions - Section 1.6

1.6 Contributions

The key contributions of this work are highlighted here. The developed software, as well as the the models developed for case studies were the main deliverable contributions. Simbiotics (Chapter 3) was published with case study findings for the coaggregation and biofilm studies (Chapters 7 and 8) in *ACS Synthetic Biology* in 2017, and Chapter (9) was published in *Life* 2018 (Full citations are shown in Section 1.7).

1.6.1 Simbiotics

A primary deliverable from the research is Simbiotics, a modelling framework for multi-scale modelling of multicellular systems in a spatial domain (shown in 1.3 (A)). The platform integrates state-of-the-art methods for simulating bacteria with an spatially explicit agent-based model, enabling multi-scale modelling of physically realistic multicellular and multi-species systems. The platform implementation was motivated by a review of existing model techniques, frontier questions relating to the studying multicellular systems, and the requirements of biologists engaging in modelling in both Systems and Synthetic biology. The platform was therefore developed as a toolbox, providing a library of modelling techniques which can be compositional structured in a model to represent, simulate and analyse a desired multicellular system.

Simbiotics can be found at: https://bitbucket.org/simbiotics/simbiotics/wiki/Home

1.6.2 Easybiotics

Another deliverable of the project is Easybiotics, a graphical user interface (GUI) for engaging in multi-scale modelling of multicellular systems with minimal programming experience (shown in 1.3 (B)). The GUI provides a user-friendly layer of abstraction on top of the Simbiotics framework, providing full access to Simbiotics modelling and analysis features. Easybiotics allows for model development, simulation and analysis to be done via click and select commands.

Easybiotics is included as part of the Simbiotics package at:https://bitbucket.org/simbiotics/simbiotics/wiki/Home

1.6.3 Biomodel and method format

A file format for representing multi-scale population models and modelling methods was implemented for reuse, communication and formalisation of the developed methods (shown in 1.3 (C)). The format is general and non-prescriptive, with a flexible and extendable implementation to allow for it to be maintained as related knowledge and methods advance. This was developed with consideration to the growing standardisation of existing formats such as SBML, allowing for populations of interacting SBML models to be defined. The file format is presented in Chapter 5.

1.6.4 Case study models

The case studies involved developing models of various multicellular systems. These models all follow the population biomodel format described above. Each of the models are briefly described, including where those models can be found.

Coaggregation model

The model developed in the coaggregation study (Chapter 7) simulated how purely physical forces can influence the aggregation of bacteria in a fluid phase. The model simulates bacteria is a spheres free-floating a fluid phase which experience a passive mixing force due to Brownian motion, and can interact with other cells through specific membrane-mediated interactions and non-specific electrostatic interactions.

The Java version of the model can be found in the Simbiotics source code at: *simbiotics.examples.casestudies.coaggregation.Study_Coaggregation*

The JSON encoding of the file (which follows the format described in Chapter 5) can be found in the Simbiotics examples folder: *examples/models/casestudies/-coaggregation.json*

Biofilm model

A model of biofilm formation and development was developed (Chapter 8), simulating how physical forces affect biofilm structure. The model simulates bacteria as spheres which are free-floating in a fluid phase where the lower boundary of the domain was modelled as a solid substratum. Cells experience a passive mixing force due to Brownian motion, and can adhere to others cells and the substratum via membrane-mediated interactions. Cells growth is modelled assuming a constant nutrient supply. The model also includes that cells which have adhered to the substratum/biofilm experience a lower mixing force and a higher growth rate.

The Java version of the model can be found in the Simbiotics source code at: *simbiotics.examples.casestudies.biofilm.Study_ Biofilm*

The JSON encoding of the file (which follows the format described in Chapter 5) can be found in the Simbiotics examples folder: *examples/models/casestudies/biofilm.json*

Compartmentalized autocatalytic sets model

A model of compartmenatalized autocatalytics sets in a spatial domain was developed (Chapter 9), modelling the influence of inter-compartment diffusble molecules on molecular activity across the population of compartments. Compartments are modelled as immotile spheres on a flat plane, embedded in a 2D grid for simulating fluxes of extracellular chemical concentrations. The model includes a Gillespie model of a chemical reaction networks embedded in spatial compartments, where some of the chemicals in that network may diffuse out of a compartment, through the extracellular space, and into other compartments.

The Java version of the model can be found in the Simbiotics source code at: *simbiotics.examples.casestudies.raf.Study_ RAFs*

The JSON encoding of the file (which follows the format described in Chapter 5) can be found in the Simbiotics examples folder: *examples/models/casestudies/raf_ sets.json*

Pulse generator model

A model of a synthetic genetic circuit for generating a gene regulation pulse across a population was developed (Chapter 10). The model represents cells as static spheres on a flat plane. embedded in a 2D grid for simulating extracellular chemical diffusion. Two cellular species are modelled, composing a *sender* and *receiver* system, where the sender produces *AHL* and the receiver responds to *AHL* by producing *GFP* for a limited time. *AHL* is secreted through the senders membrane into the extracellular space, diffuses through the extracellular space, and is transported into the receiver cells via an active transport mechanism. The emergent dynamics of the system when a population of sender cells are placed in a population of receiver cells is a pulse of GFP being expressed radiating from the sender positions.

The Java version of the model can be found in the Simbiotics source code at: *simbiotics.examples.casestudies.pattern.Study_PulseGenerator*

The JSON encoding of the file (which follows the format described in Chapter 5) can be found in the Simbiotics examples folder: *examples/models/casestudies/pulse_generator.json*

Pattern formation model

A model of synthetic multicellular pattern formation was developed (Chapter 10). The model simulations cells and the spatial domain in the same was as *Pulse generator* model, but only simulates a single *Receiver* species. The receiver species has a synthetic genetic circuit to produce one of two fluorescent proteins, induced by one of two possible diffusable signal molecules, and also produces and secretes the signal molecule for the other fluorescent protein. This results in the formation of stripes (or bands) of gene regulation forming throughout the colony, radiating from where the system was initially induced.

The Java version of the model can be found in the Simbiotics source code at: *simbiotics.examples.casestudies.pattern.Study_PatternFormation*

The JSON encoding of the file (which follows the format described in Chapter 5) can be found in the Simbiotics examples folder: *examples/models/casestudies/-pattern_formation.json*

1.6.5 Use of Simbiotics by the community

Simbiotics has been used by the Newcastle 2017 iGEM team to model a distributed biosensor. (http://2017.igem.org/Team:Newcastle)

Simbiotics was used by David Nettleship in his Undergraduate dissertation "Investigating programmable pattern formation in synthetic bacterial colonies", 2017.

Simbiotics and Easybiotics are being used by the Newcastle 2018 iGEM team to model chemotaxis.

1.7 Publications and presented work

Journal publications

J. Naylor, H. Fellermann, Y. Ding, W.K. Mohammed, N.S. Jakubovics, F. Dafhnis-Calas , S. Heeb, M. Camara, J. Mukherjee, C.A. Biggs, P.C. Wright, N. Krasnogor **Simbiotics: A Multiscale Integrative Platform for 3D Modeling of Bacterial Populations** in *ACS Synthetic Biology*, 6(7):1194-1210, July 2017

W. Hordikj, J. Naylor, H. Fellermann, N. Krasnogor **Population Dynamics of Autocatalytic Sets in a Compartmentalized Spatial World** in *Life*, 8(3):33, August 2018

J. Naylor, H. Fellermann, N. Krasnogor **Easybiotics: a GUI for 3D physical modelling of multi-species bacterial populations** in *Bioinformatics*, btz131, February 2019

Unpublished

J. Naylor, F. Romero-Campero, H. Fellermann, N. Kransogor **Pattern formation via synthetic cell signalling** (2019)

Conferences/Workshops

ECAL in York, 2015
SSBSS in Volterra, 2016
ABC in Karlsruhe, 2016
SSBSS in Cambridge, 2017

Chapter 2

Background

This chapter presents an overview of the underpinning concepts of this research, followed by the background theory of relevant biological theory, and the modelling techniques used to model these phenomena. It alsos present a review of some existing software tools in the domain of multi-scale multicellular modelling. From this literature and software review, we establish a set of requirements the developed software should achieve in order to advance modelling techniques.

2.1 Overview

Development of a software tool for integrative modelling of multicellular system requires a thorough understanding of the processes governing these systems and the techniques used to model them. Multicellular systems cover a wide range of phenomena, ranging from low-density populations of interacting bacteria to tightly packed cells in biofilms and tissues, and though the fundamental aspects of cells remain the same, the most appropriate or efficient techniques for modelling them can differ. For this reason we must establish the level of model abstraction we are choosing to implement in the software in order to inform the direction of the literature review.

As we are interested in feedback between micro-scale biochemical reactions and macro-scale spatial organisation, it is crucial to represent the system as a compartmentalised spatial domain, such that reactions may occur at localised

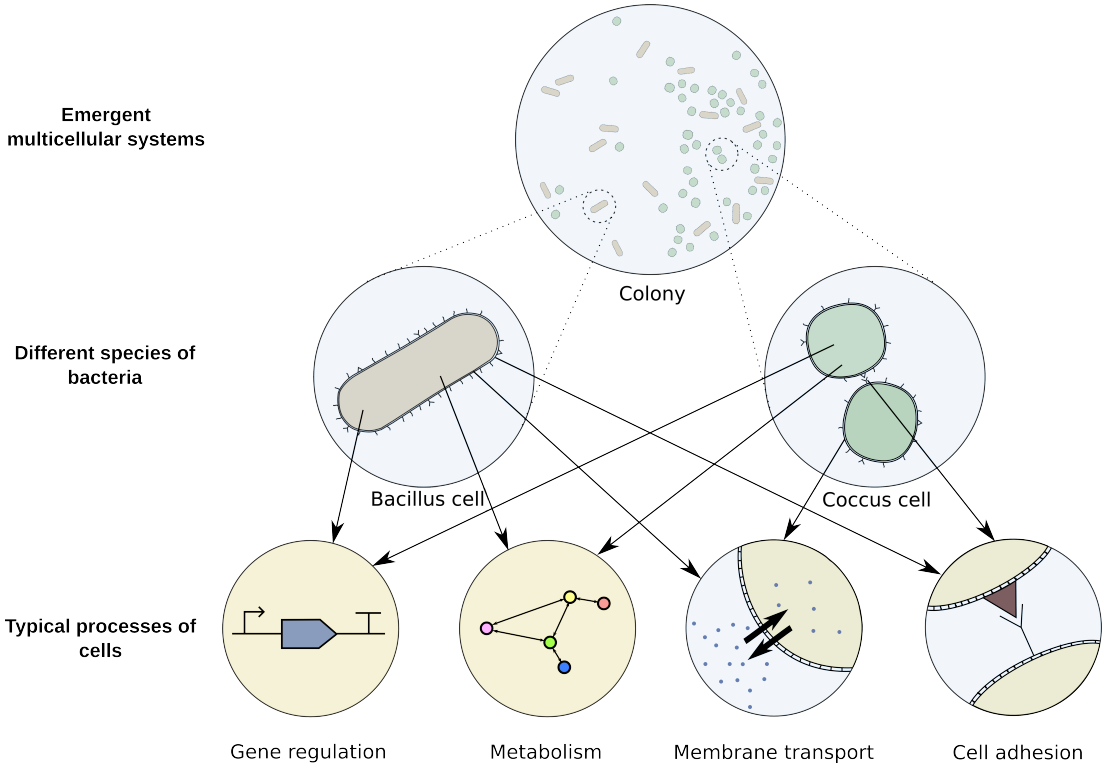


Figure 2.1: Schematic showing the different scales in bacterial multicellular systems and exemplifying some of cell processes. The bottom row shows some example processes of bacterial cells, illustrating how these mechanisms can be present across multiple species of bacteria (middle row), and how these lead to emergent colony organisations (top image).

positions. Cells are considered to be the individual entities in the model (acting as the *parts* in a complex system). Cells should be able to move around and collide with each other. Each cell should also have some internal behaviour describing chemical processes occurring within it. Cell internal behaviour should then be coupled to the spatial domain allowing the chemical signalling between cells.

With these constraints in mind, we can determine what the most relevant features of multicellular systems are for our modelling framework, and what the most appropriate modelling methods are to simulate them. As we consider cells to be the interacting entities in the model, we consider the theory underpinning the *single cell* (the processes that occur within a cell), and the theory behind the *social cell* (the processes by which cells interact).

This chapter is split into 3 sections. The first section presents a review of the biological theory underpinning the *single cell* and the *social cell*. The second section then reviews the methods applied to modelling the *single cell* and the *social cell*. The third section presents a software review of existing multicellular modelling tools, considering the current cutting-edge technologies in this area of research, and what the current obstacles to development in the field are.

2.2 Biological theory

This section presents the biological theory of multicellular systems with consideration the level of abstraction intended for the modelling framework. We take the perspective that: the cell is a physical individual that can move around, collide and adhere to things, it has a intracellular dynamics such as gene regulation and

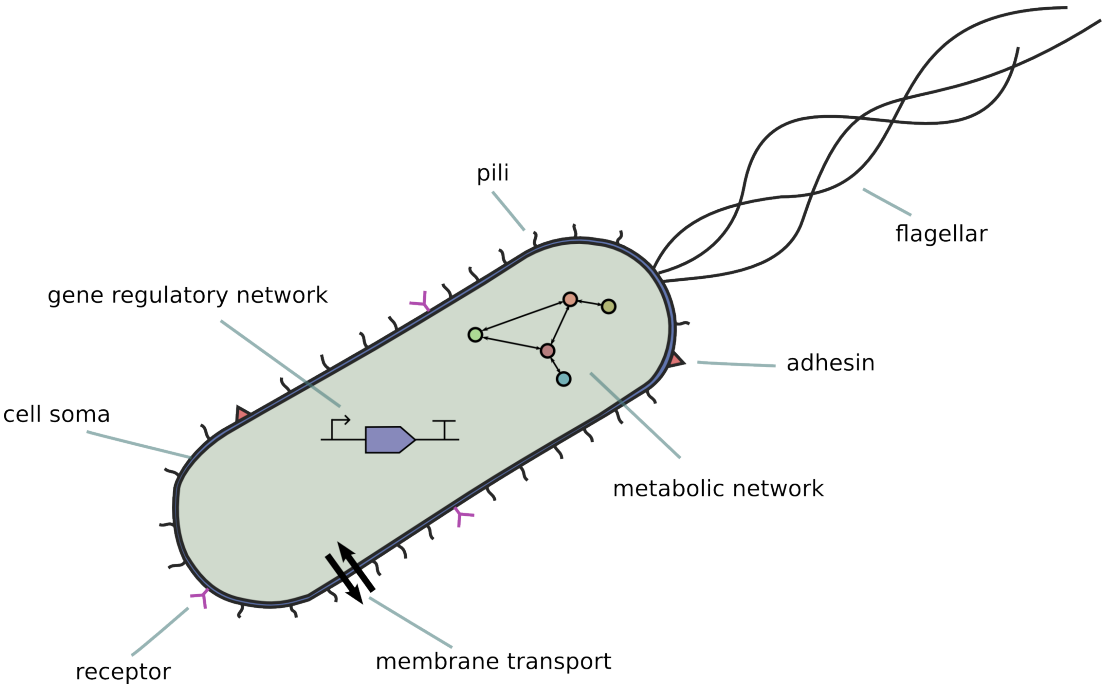


Figure 2.2: Schematic showing the key features of bacterial cell relevant to the level of model abstract we will consider in the software tool. The components and processes depicted are exemplar and not an exhaustive list of the phenomena that we consider.

metabolism, and it can communicate with others through physical and chemical interactions. A schematic depicting a population of cells with some exemplar processes relevant to the desired level of modelling can be seen in Figure 2.1.

2.2.1 The single cell

Bacteria are microscopic single celled organisms, many species have evolved over millenia, with estimates of the number of bacterial species between thirty-thousand and a few million [7, 55]. Species vary in both structure and function, however their underlying machinery follow the same principles and fundamental mechanisms. All species of bacteria are prokaryotes, meaning they do not have a nucleus and rather their DNA is stored either as chromosomes in the nucleoid region of the cell, or as plasmids in the cytoplasm [80]. Expression of this DNA is regulated by transcription and translation, which are the processes by which the DNA is interpreted in order to synthesize proteins. These proteins may then be used in the metabolism of the bacteria, which is essentially a complex chemical reaction network which allows the cell to maintain itself, grow and divide [1].

The theory presented here is a targeted set of key cellular features relevant to the level of model abstraction we have chosen. An overview of a single cell seen at this abstraction level can be seen in Figure 2.2, showing some exemplar cell features.

2.2.1.1 Gene regulation

The gene regulatory network (GRN) of a cell governs the expression of genes into proteins. The two major mechanisms involved in this process are *transcription* and *translation*. Transcription is the process of an mRNA polymerase binding to a coding site on the DNA, and building an mRNA molecule encoding the genetic information. Translation is the process of this mRNA molecule being used by a ribosome, instructing it on how to arrange amino acids into proteins.

Transcription factors such as the activity of the promotor region of a gene effect the rate at which the mRNA of a gene is synthesized, however gene regulation can also be effected during the translation phase. For example the degradation rate of the mRNA and concentration of ribosomes in the cytosol influence the

rate at which the mRNA is translated into a protein [9].

Gene activity may effect other genes activity, by their synthesized protein effecting the transcription or translation rates of other genes. Some genes may *activate* or *inhibit* the regulation of other genes, through mechanisms such as producing a protein that can bind to the promotor region of another gene, modifying its transcription rate. Genes may also be self-regulating, causing positive (activatory) or negative (inhibitory) feedback to their own synthesis rate.

Gene regulation influences the phenotype of the cell, such as a protein being used in the development of membrane fimbria such as adhesins and receptors. This changes how the cell interacts with the outside world. Similarly stresses and signals from the outside world can effect the gene regulation of a cell, resulting a dynamic feedback between gene regulation and the immediate environment of cell [122].

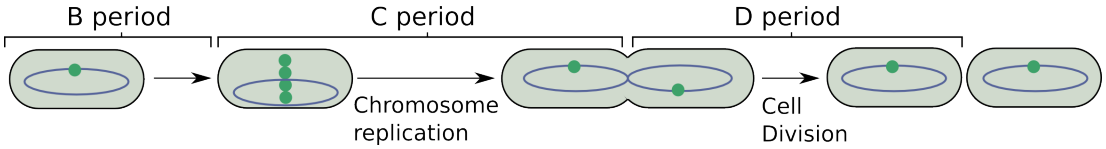


Figure 2.3: Schematic showing cell cycle stages in bacteria, taken from [225]

2.2.1.2 Metabolism, cell cycle and mitosis

Cells require energy and material for many processes, such as motility, building membranes, enzyme synthesis and mitosis (cell division) [1, 54]. The process by which a cell acquires and processes energy and materials is called its metabolism. The metabolism can be considered to be the set of all biochemical reactions which occur within a cell. Metabolic pathways typically convert one molecular species into another species, which are then involved in cellular processes. The metabolism and gene regulation are heavily linked, allowing for dynamic phenotypical changes when metabolic stresses occur [38].

The cell-cycle is the process by which a single cell grows and replicates its genetic information, to propagate to its daughter cells during mitosis (division into two daughter cells). It requires for the duplication of the DNA, building of a septum inside the cell to become the new cell wall, and distribution of intracellular

molecular components and chemicals between child cells. In bacteria the cell cycle is divided into three stages: the period between cell and the initiation of chromosome replication, known as the B period. The next stage is the time taken to complete replication of the genetic information, know as the C period. The final stage is the time taken between the end of replication and the end of division, referred to as the D period. A schematic illustrating the cell cycle in bacteria can be seen in Figure 2.3 [225].

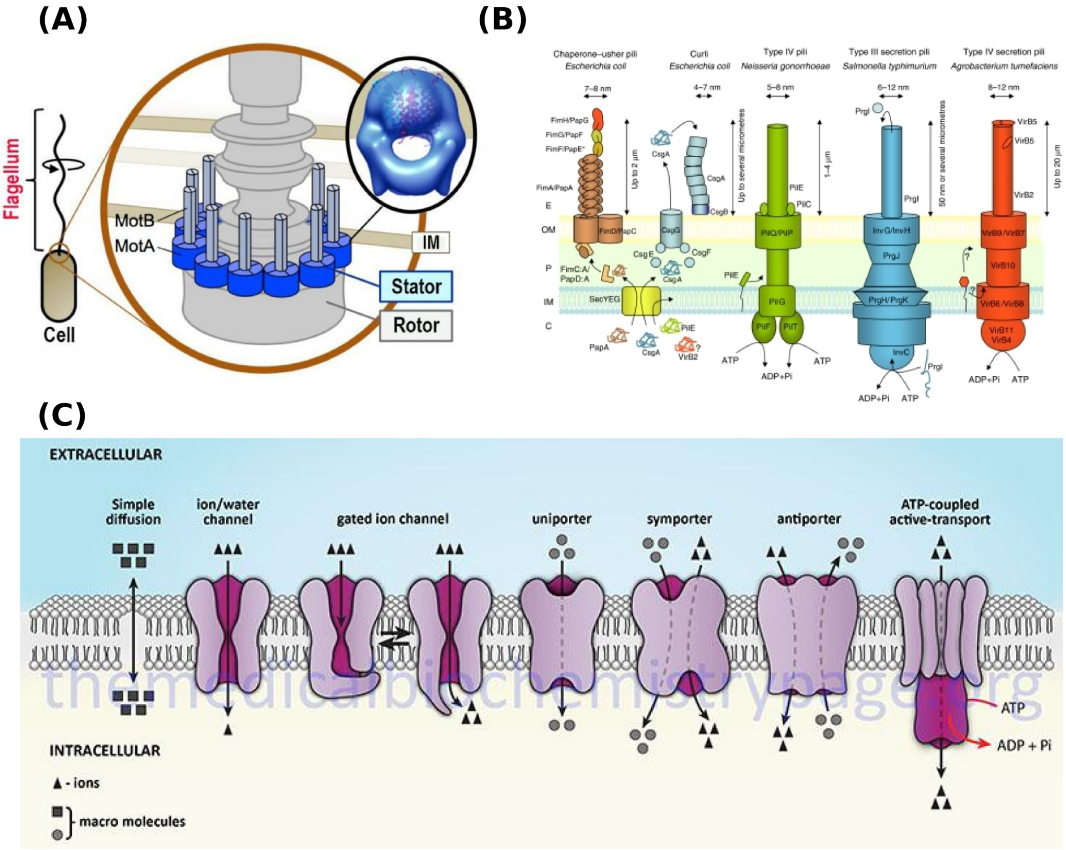


Figure 2.4: Schematic showing some of the different features present on bacterial membranes. **(A)** A flagellar - taken from https://openi.nlm.nih.gov/detailedresult.php?img=PMC2500206_emboj2008155f1&req=4. **(B)** Different types of *pili* - taken from https://phys.org/news/2016-09-japanese-team-elucidates-bacterial-flagellar.html . **(C)** Different types of membrane transport mechanisms - taken from https://themedicalbiochemistrypage.org/membranes.php (All websites accessed: 18-09-18)

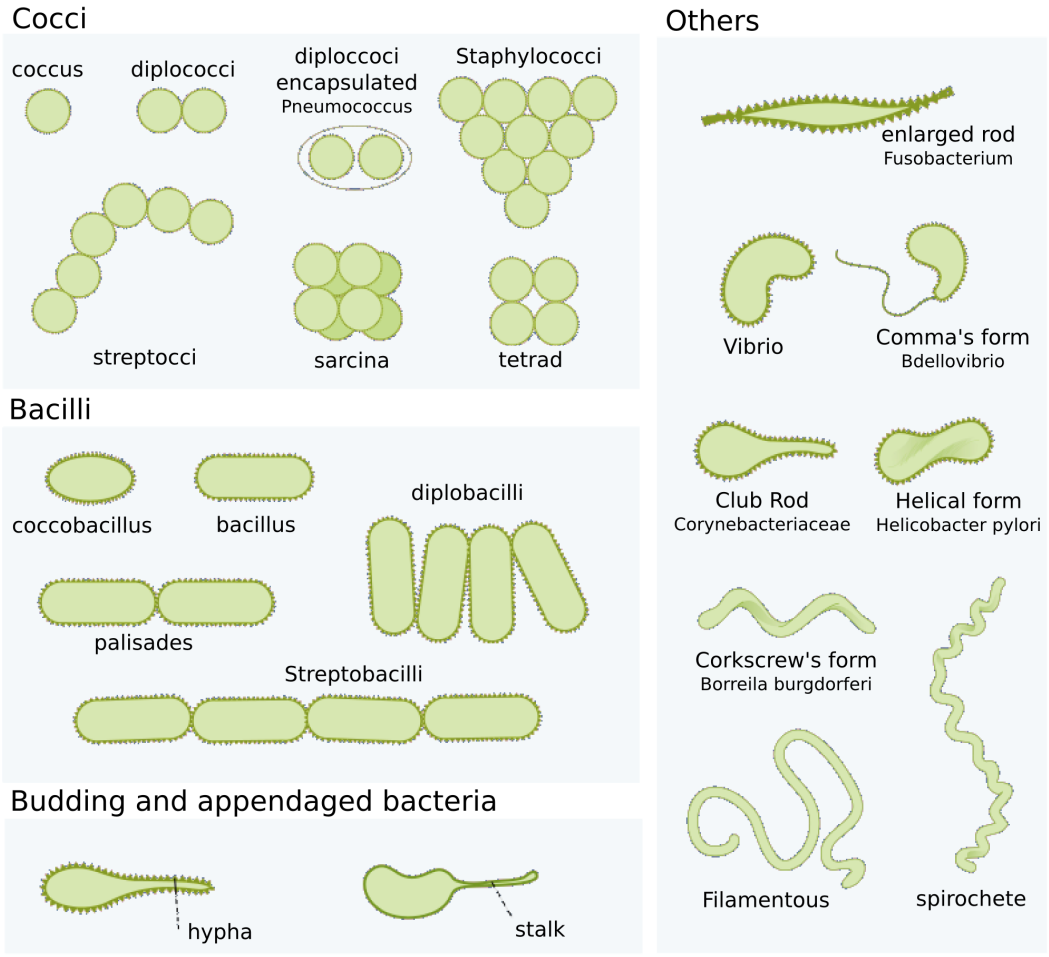


Figure 2.5: Examples of bacterial morphologies, taken and modified from https://en.wikipedia.org/wiki/Bacterial_cellular_morphologies (Accessed: 16-09-18)

2.2.1.3 Morphology, membranes and motility

Cells have a physical shape, referred to as its morphology. The most common are cocci (spherical) and bacilli (rod-shaped), however these may then go on to form complex structures. Various cell morphologies can be seen in Figure 2.5.

Bacteria are classified as either *Gram-positive* or *Gram-negative* depending on the structure of their membrane. Gram-negative bacteria have a thin peptidoglycan cell wall, surrounded by an outer membrane containing lipopolysaccharide,

where as Gram-positive bacteria only have a thick peptidoglycan layer [197]. Membrane-embedded structures such as *fimbriae*, *pili*, *flagellar*, *transporter proteins* and *adhesins (surface proteins)* may also be present [15, 102, 236]. Figure 2.4 illustrates some of these features.

Bacterial cells can be motile, employing different types of mechanisms to achieve different types of movement. Flagellar have been associated with bacteria swimming in fluid and swarming on surfaces, propelling them toward more favourable environments [212]. Type IV *pili* have been found to be involved in twitching motility [151].

These physical aspects of the cell are inherently connected with the gene regulation, metabolism and cell-cycle phase of the bacteria. Metabolic chemical signal pathways have been shown to be responsible for pili and flagellar regulation [161, 170, 234].

2.2.2 The social cell

Bacteria have many ways to communicate with each other, allowing cells to broadcast their state and intentions whilst also listening to others. This section reviews the types of cell communication, and how these play a part in the coordination of complex bacterial communities such as biofilms.

2.2.2.1 Physical and biochemical interactions

Bacterial physically interact with each other and the environment. Cells may shove/collide with each other (shown in Figure 2.6 (A)), they may adhere to surfaces and other cells via membrane-mediated interactions, governed by adhesin-receptor interactions on the cell surface as well as short range Van der Waals and electrostatic interactions [172, 185] (shown in Figure 2.6 (B)). Directly contacting cells may also undergo conjugation, which involves the genetic transfer from one cell to another [76] (shown in Figure 2.6 (C)).

Bacterial can also interact with each other through chemical signalling, where diffusable molecules are sensed and secreted by cells (shown in Figure 2.6 (D)). Quorum-sensing (QS) is an example of chemical signal circuits evolved in bacteria; QS is the regulation of gene expression in response to changes in cell population

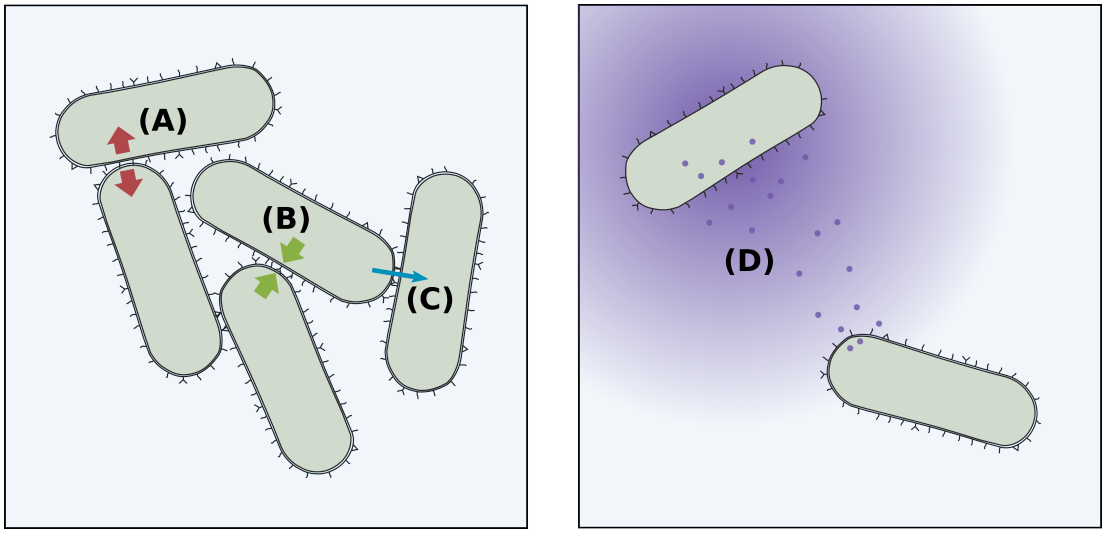


Figure 2.6: Schematic of some bacterial 'social' interactions. **(A)** Two cells physicallt shoving each other (colliding). **(B)** Two cells which have adhered to each other via membrane structures such as adhesins/receptors. **(C)** Two cells engaging in conjugation. **(D)** Two cells communicating via a chemical signal, such as in the case of quorum-sensing.

density [153]. It has been shown that QS can be involved in interactions between bacteria and a host they inhabit, as opposed to being exclusively for cell-cell communication [100].

2.2.2.2 Colonies and biofilms

Biofilms are complex, self-organised communities of bacteria. Exhibiting adaptive and diverse behaviours, biofilms occur in a multitude of situations, employing altruistic survival tactics ensuring the longevity of the community as a whole [126]. They may occur when the collective chemical signals released by bacteria accumulate to a point that triggers cells to differentiate and form a biofilm [42, 135].

The coordination of biofilm development results from local interactions, which propagate across the community leading to changes in gene expression [53]. For example, a change in the local environment may cause an individual bacterium to move, resulting in a reconfiguration of the surrounding bacteria. In turn this larger scale spatial reorganisation may cause changes in nutrient level distribu-

tions within the biofilm. This change in the local environment for bacteria may then result in them moving again or changing their gene expression. An interplay emerges between micro-scale and macro-scale processes in the biofilm, leading to its spatial patterning in structure and sometimes metabolic and gene regulatory behaviour [26, 81, 132].

A key feature which differentiates biofilms from typical bacterial colonies is the production of *extracellular polymeric substances (EPS)*. Consisting of mainly *exopolysaccharide*, *proteins* and *extracellular DNA*, *EPS* forms a sticky capsule around bacteria. In large populations this *EPS* develops into a slime layer, or *extracellular matrix (ECM)*, embedding the bacterial community. The function of this *ECM* is not fully established, however it appears to help protect the biofilm from changes in the local environment, along with giving structural support to the biofilm structure. The *ECM* may also act as a catalyst to intercellular signalling within the biofilm, potentially increasing metabolic efficiency [40, 63, 101].

The mechanisms through which bacteria coordinate biofilm development vary greatly between species, thus establishing a typical rule-set to describe their behaviour is difficult to determine. To frame biofilms at a level of abstraction which can be applied to all species (in general), a biofilm life-cycle has been conceptualised (shown in Figure 2.7). This life cycle considers the key stages in a biofilm's

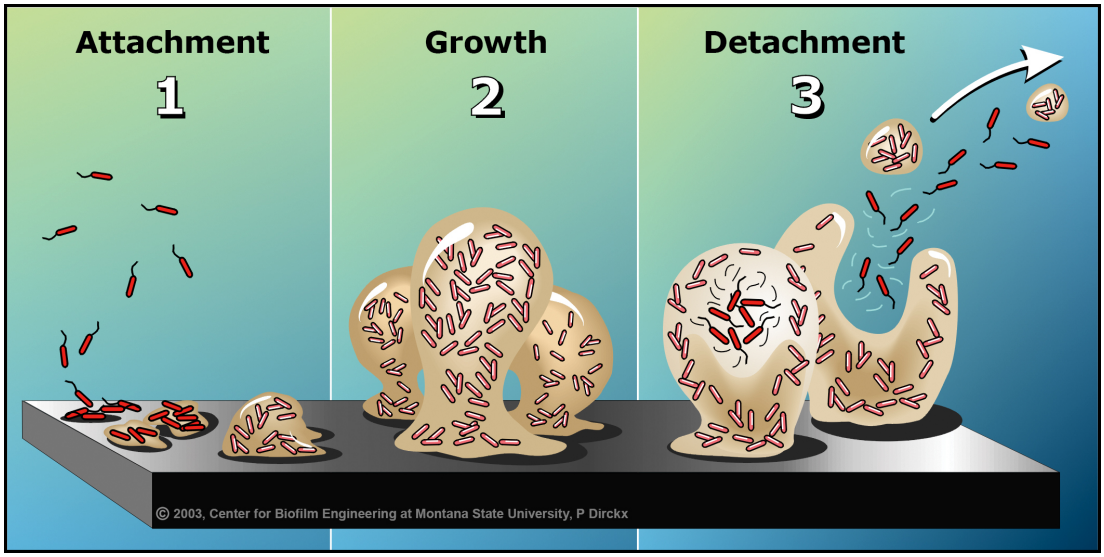


Figure 2.7: Schematic showing Test 8

development.

Biofilm life cycle

Attachment

Individual bacteria adhere to some substratum (surface). This adhesion is initially reversible, and bacteria can relocate if this is not an appropriate location for a biofilm to develop. This process is mediated by the flagella, cell-surface appendages and polysaccharides [98].

Adhesion then occurs irreversibly, as bacteria begin to secrete *EPS* to form a slime-layer, embedding the bacteria in an *extracellular matrix (ECM)* [68].

Growth

Early biofilm growth occurs through both the recruitment of planktonic bacteria which attach to the biofilm, and through mitosis of existing members of the biofilm.

Maturation often consists of cellular differentiation within the biofilm, along with spatial organisation which typically results in water channels forming through the biofilm and mushroom-like structures which extend into the substrate.

Detachment

The detachment of bacteria from the surface of the biofilm can be initiated both voluntarily and involuntarily. Through this mechanism planktonic bacteria are freed, which flow downstream and have the potential to develop a new biofilm colony [40, 110].

2.3 Modelling biological phenomena

There are numerous techniques for modelling biological phenomenom. Specific processes and systems may be simulated through continuous methods, where as others may require discrete methods to accurately predict them. The application of a specific method to modelling a phenomena often depends on the questions being posed to the model. Research into bacterial dynamics has conceived many

models of bacterial behaviour. These models can simulate phenomena such as bacterial motility, growth, division, gene regulation and aggregation. Hybrid models have also been conceived to couple the simulation of different processes. This section reviews the main techniques for simulating the *single* cell and the *social* cell.

2.3.1 The single cell

Single cell modelling primarily focusses on approximating gene expression and metabolic activity. Both of these processes involve the diffusion of molecules through the cytoplasm and interacting with other molecules. This process can be represented in numerous forms, each with their strengths and weaknesses.

2.3.1.1 Boolean networks

Boolean networks can be used to model gene regulation [48]. This is achieved by representing genes as boolean states, which are either *ON* or *OFF* (describing whether they are transcribed of not). The genes are then connected by boolean relations, which are either *activatory* or *inhibitory* interactions between genes. Gene states are updated simultaneously in discrete time steps, with the new state of a gene depending on the states of the genes with which it has a relation [115, 122].

This simplified model of a dynamical interacting system gives a basic approximation of how numbers of genes interact over time. Boolean networks may also enable the modelling of cell decision making, such as if certain environmental and intracellular criteria are met, then a differentiation process should occur.

Boolean networks can take on 2^N system-wide states, where N is the number of nodes. The state-space therefore increases exponentially with the number of nodes. They are however very inexpensive to solve, and tend towards a small number of attractors.

This method has been successfully applied to the modelling of attractors in gene networks [226], synthetic biological boolean gates [183], and in the prediction of the cell cycle sequence of fission yeast [47].

2.3.1.2 Ordinary differential equations

Ordinary differential equations are effective techniques for modelling how biological properties change over time, and have been applied to modelling a wide range of biochemical reactions [201]. They have been used to model gene expression [30], defining terms which describe how molecular concentrations change over time. For example, consider the general equation

$$\frac{dx_i}{dt} = f_i(x_1, ..., x_n) \quad i = 1, ..., n. \tag{2.1}$$

Where x_i is the concentration of some intracellular moleclular species, such as mRNA or protein. The function f_i is the rate equation as to how the concentration of x_i changes with respect to the other molecular species concentrations. This form allows for the representation and numerical approximation of many interacting molecular species [122].

A limitation of using ODEs to represent chemical systems is that they assume variables are continuous values, where as molecular numbers must be discrete. Another limitation of ODEs is that they are a deterministic method, and do not consider noise and random fluctuations within the system.

2.3.1.3 Gillespie stochastic simulation

The Gillespie method is a discrete probabilistic method originally developed for modelling the time evolution of chemical species in a reaction network. The method represents molecules in discrete integers rather than as continuous values, and also account for the probabilistic nature of these systems by accounting for varying concentrations and diffusion times of participants to react. Stochastic methods such as the Gillespie method are suited to modelling biological phenomena due to the inherent noise associated with these systems [207, 237].

Variants of the Gillespie method have been developed, such as tau-leaping, aiming to provide better performance, however this is not always the case. It has been shown that different models of biochemical reactions perform best with different versions of the Gillespie method, for which the most appropriate method for a given model of biochemical reactions can be determined through a topological analysis of the reaction networks [184].

Stochastic methods may also be used in the process of statistical model checking. They can be used to determine the probability an event will occur over a given amount of time, allowing for the verification that assumptions about certain system states hold true. Variations of statistical model checking using stochastic simulations have been shown to be able to estimate reaction rate parameters of relatively large system sizes [51, 239].

2.3.1.4 SBML

The Systems Biology Markup Language (SBML) [99] is a standard representation for describing metabolic and genetic activity within a cell. It is an XML dialect which supported by over 280 software packages. There exist software to easily develop SBML models, and numerous packages for simulating and analysing models [22, 188]. SBML can be simulated with deterministic or stochastic methods, and can be defined to have events and model interactions. There are large collections of SBML models from published work stored in databases such as the BioModels archive [131].

2.3.2 The social cell

There are numerous techniques which have been developed for modelling populations of interacting cells. These methods all involve representing some compartmentalised dynamics, where compartments may interact via some defined rules.

2.3.2.1 Cellular automata and P systems

Common methods used for modelling bacterial populations include P systems and cellular automata. These two methods represent compartmentalised system dynamics, and can be used to model how interactions between constituent parts lead to population dynamics.

Cellular automata (CA) are composed of a grid of cells, where each cell is in one of a finite number of defined states, and rules are applied to update cell states based on some local criteria to that cell (the states of its neighbouring cells)

[231]. This method has been applied to the study of multicellular systems such the growth patterns of bacterial colonies, and structure of biofilms [21, 194, 198, 222].

P systems are computational modelling technique that are inspired by the way in which biological systems process information [171]. They are commonly used in membrane computing, modelling the interactions of two spatially separated compartments via an interface [37]. Extensions of P systems have been conceived to allow for multiple interacting compartments with specific spatial interfaces [179]. P systems have been used to model quorum-sensing in P. aeruginosa [124] and photosynthesis processes [10].

2.3.2.2 Agent-based modelling

Agent-based models (ABMs) represent systems as interacting agents within an environment [3]. This bottom-up approach allows for the description of micro-parts and their interactions, and simulation of how a population of these parts behave. This technique allows for self-organisation and emergence to be modelled, observing how communities of independently acting agents result in population level behaviour which were not pre-programmed. This paradigm lends itself to modelling bacterial communities, as they are composed on individual cells which sense and react to their local environment.

ABMs have been used in the modelling of bacterial populations, typically coupled with a spatial simulation to represent cells as physical individuals with individual behaviour [74, 108, 127, 130, 181]. This technique of coupling an ABM with a spatial domain in which to represent chemical gradients has proven to be a valuable method for multi-scale modelling of multicelluar systems.

2.4 Multicellular modelling software review and current challenges

Several modelling tools have been developed to understand the dynamics of bacterial populations and the multicellular systems they form [24, 125, 183]. General cell population modelling tools include gro [105] and CellModeller4 [181], intended to simulate the biophysical patterning of multicellular systems in 2D,

	iDynoMiCS[130]	BSim[75]	gro[105]	CellModeller4[181]	DiSCUS[74]	Infobiotics Workbench[24]
Rod shaped cells			X	X	X	
Bacterial growth	X	X	X	X	X	
Rule based internal cell dynamics		X	X	X	X	X
Differential equation based internal cell dynamics		X		X	X	
SBML based internal cell dynamics						
Gillespie submodels						X
Bacterial motility	X	X				
Chemotaxis		X	X	X		
Cell surface interactions						
Membrane transport		X				X
Environmental forces		X			X	
Chemical environment	X	X	X	X		X
Extracellular polymeric substances	X					
2D	X		X	X	X	X
3D	X	X				
Fluid dynamics					X	
Conjugation					X	
Computational acceleration				X		
Programming language	Java	Java	Python	Python	Python	Java

Table 2.1: High-level feature comparison of existing agent-based modelling tools for bacterial populations. X marks a feature being present.

focusing on physical interactions and chemical signalling. BSim[75] is another general tool which is used to model cells in 3D, providing a general agent-based platform in which the user can define custom rules to describe cellular behaviour, as well as environmental structures via 3D meshes. More specific tools include iDynoMiCS[130], the successor of BacSim[127], which is a modelling tool for biofilm formation. It allows for the specification of cellular properties and simulation of a biofilm growing on a surface. Substrate dependence is represented in the iDynoMiCS model, simulating how the location of cells effects their growth. DiSCUS [74] is a specific 2D bacterial simulator modelling horizontal gene transfer between neighbouring cells. All of these tools simulate chemical diffusion by discretising the environment space into subvolumes and calculating the flux between neighbouring compartments via a finite-element method. An overview of the modelling features present in the different software can be seen in Table 2.1.

Each of the existing multicellular modelling tools takes a level of abstraction at which to represent the system, and focuses on simulating relevant features to the questions for which the models are developed. For example, *iDynoMiCS* has been driven by the research into biofilms the influence of individual cell metabolic behaviour, and thus its implementation and features focus on those aspects. On the other hand *CellModeller4* has been used to model gene regulation and chemical signalling, and does not simulate EPS or other features relevant to biofilms. The existing platforms have a common underlying theme of an agent-based mod-

elling framework, representing cells interacting in a spatial environment, coupled with a grid simulating chemical diffusion. The specific implementation of each platform however is at some level over-fit to the phenomena they intend to simulate, which is of course always the case with abstract modelling - however there is almost certainly a common level of abstraction at which all of the case studies conducted with these tools can be represented, though such a unification of these models has not yet been conceived.

An issue presented here is that in order to develop new models of multicellular systems programmers have to modify an existing framework or build a new one fit for purpose. None of the existing platforms provide a programming framework which can be used to build novels models in a dynamic manner, such as exist in software development domain where development frameworks have dramatically enhanced programmers capacities to build cross-platform applications without being concerned with the low lying implementation [137, 211, 233]. Similarly in traditional engineering disciplines, software frameworks have been used to formally represent knowledge and integrate it into CAD tools which allow for the relatively non-programmatic design and simulation of devices [46, 157, 221].

The endeavour of developing such frameworks and tools for multicellular modelling is young, and the first steps toward this goal have brought some formalisation of numerical methods and biological models into a programmatic environment, as seen in these existing multicellular modelling tools. A challenge in this area is how to find a common level at which to represent these systems, and how to accurately integrate the modelling techniques into a flexible platform that a modeller can build novel models with. Additionally, to conceive of a general tool for this type of modelling, some of the main aspects about the tool from the end-users perspective (the modeller) are: the modelling capacity of the platform, the usability of the platform, and the scalability of the platform. Modelling capacity being the ability of a platform to describe a given multicellular system, representing its constituent components and processes. Usability being the ease in which that description can be expressed in the platform, and scalability the potential for the platform to simulate large industrially relevant systems.

Considering these aspects and reviewing the existing tools show us that some of the current limitations in this domain are:

- Lack of unification in multicellular modelling techniques;
- Lack of software framework for programmers to readily compose new multicellular models without modifying an existing software;
- Lack of higher level tools that allow non-programmers to build and analyse multicellular models.

2.5 Software requirements

Reviewing literature of bacterial phenomena and state of the art modelling methods has informed the design of a general simulation platform. It has exposed the key features of bacteria and the ways in which they coordinate their behaviour to form complex communities, as well as informing which methods can be used to model these features. Additionally, challenges and limitations of the existing software tools for engaging in multicellular modelling have exposed some of the main aspects we must consider to move forward.

Iteractions with collaborators also set the software requirements, their systems of interest and the modelling questions they had were used to derive the features required to be a powerful modelling tool. The details of how each collaboration infuenced the development of Simbiotics can be found in the case study chapters (Chapters 7 - 10).

The developed modelling platform should be able to simulate mixed species populations, where each cell can have a different physical shape, intracellular dynamics and interactions. The platform should be able to simulate these in different spatial configurations, such as planktonic (free-floating) or sessile (adhered to a colony or biofilm). Extracellular aspects such as chemical diffusion and EPS should also be included in the platform. The simulation should be physically, chemically and biological realistic, integrating all of these processes into a multiscale model that can be tailored by the modeller. The techniques used to simulate specific processes should also be flexible, such as the modeller being able to choose whether they use continuous ODEs or discrete Gillespie submodels to represent biochemical processes. The specific features that modelling environment should be able to represent are:

- The shape of the bacteria;
- The metabolism of the bacteria;
- The gene regulatory network of the bacteria;
- Many species of bacteria;
- How molecules and chemicals cross the membrane of the bacteria;
- Extracellular chemicals and their diffusion through space;
- Physical interactions between bacteria have (collisions, adhesion);
- Motility of bacteria;
- Bacterial conjugation;
- The environmental physical forces acting on the bacteria;
- Extracellular polymeric substances.

The methods for simulating the biological phenomena should be able to be achieved via both discrete and continuous methods, thus the following methods should be integrated into the platform:

- Boolean networks, ODEs and Gillespie models.

Additionally, for the integration of these into a useful *tool* that empowers modellers to build and analyse these systems, the following items are added to the requirements:

- 2D and 3D models;
- A library of modelling methods the user can integrate to build a model;
- Customisable data collection;
- Virtualised lab equipment (such as pipettes, microsensors, chemostats, etc.);

- Statistical physics tools (such as measuring the mean-squared displacement of cells);
- Integration with common data formats in the domain (such as SBML);
- User friendly environment for building and running models with minimal programming;
- Computational acceleration for simulating large systems.

The platform should also be extendable such that new processes may be added to the library, usable such that the platform interface is intuitive and does not require extensive programming to achieve valid model outputs, and scalable such that modelling of populations relevant to real system is feasible. Implementation of some computational acceleration such as multi-threading, parallelization and GPU acceleration is crucial for this scaling up to large system sizes.

2.6 Summary

In this chapter we have reviewed the relevant biological and computational literature, forming an understanding of the problem we are trying to solve, and the existing techniques which are used to study these phenomena. From this literature review, we have established a set of requirements that our simulation platform should meet in order to provide adequate features for modelling multi-cellular systems, focusing bacterial populations dynamics.

Chapter 3

Simbiotics - an integrative framework for modelling multicellular populations

This chapter introduces Simbiotics, a modelling framework for 3D simulations of bacterial populations. An overview of the software features is presented, followed by the mathematical details of the modelling and analysis modules. Additional novel features such as SBML integration and microscopy image parsing are also presented. This chapter addressed Objective 1 - Development of an extendable modelling platform and data format which allows one to express models of interacting multi-species bacteria, simulate those systems and perform analysis. The work presented in this chapter is an updated version of the published work: J. Naylor, H. Fellermann, Y. Ding, W.K. Mohammed, N.S. Jakubovics, F. Dafhnis-Calas , S. Heeb, M. Camara, J. Mukherjee, C.A. Biggs, P.C. Wright, N. Krasnogor **Simbiotics: A Multiscale Integrative Platform for 3D Modelling of Bacterial Populations** in *ACS Synthetic Biology*, 6(7):1194-1210, July 2017

3.1 Overview

Simbiotics [159] allows for the design, simulation and analysis of multicellular population models, with current features focusing on bacterial populations. Sim-

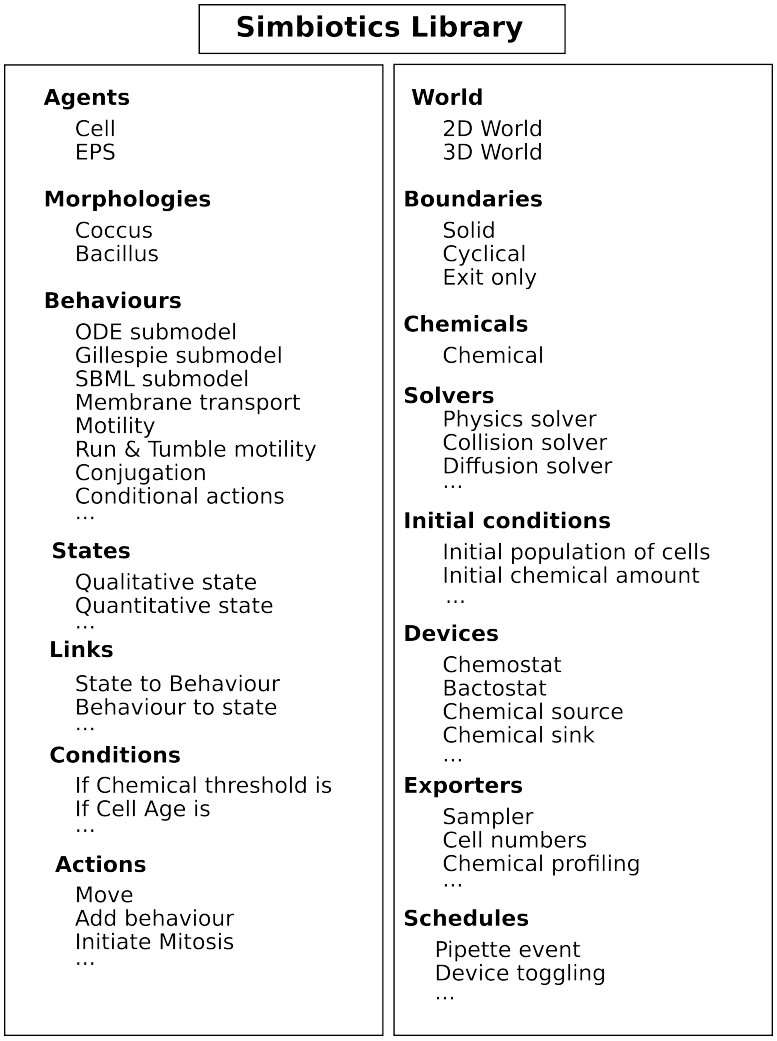


Figure 3.1: An overview of the Simbiotics library and structure. The **left panel** lists the modelling features used for representing individual agents and their dynamics. The sections (which are in bold) are the *domains* of the library, and contain *modules* which are individual Java classes implementing the feature. **Right panel:** the modelling features which are used to represent the spatial domain and its boundary conditions, as well as initial conditions and model events (schedules). An exhaustive list and more details on the library modules is provided in Section 8 of the Simbiotics user guide (In Appendix B).

biotics simulates hybrid-models, where an agent-based model which describes bacteria and other physically interacting agents is coupled to a continuous chemical environment. Through this coupling we can observe the interplay between individual cellular processes and population level organisation. Each agent can have many submodels describing its internal biological processes. Many unique species can be defined, and large mixed populations can be simulated.

Simbiotics provides a modelling library consisting of a wide range of processes, describing physical, chemical and biological processes of bacterial dynamics. This library contains a range of common modelling methods applied in bioinformatics, such as and boolean networks, Gillespie simulations and differential equation integration with a range of integrator modes (such as 3rd and 4th order Runge-Kutta integration). The library also contains analysis tools and virtual lab equipment for interacting with and collecting data from the simulation. An overview of the Simbiotics library can be seen in Figure 3.1.

The features provided in the library are motivated by the case study systems and other models presented in this thesis. The experimental systems and our collaborators set the requirements of Simbiotics, directing its development and determining which features should be implemented in the library in order to be a useful tool for modelling multicellular systems.

Simbiotics library modules are parameterised to allow for fine-tuning of user needs. New library modules may be developed and added to the library by implementing Simbiotics interface classes (full description is provided in the User Manual in Appendix B). The platform and library are readily extendable to add new features as necessary, ensuring the relevance of the software as computational methods and knowledge of biological processes develop.

Development of a model specification is achieved through a pattern somewhat similar to an entity-component system (ECS), where functional components are composed together to create complex objects. Model specifications are constructed by attaching library modules to a model specification, where they can be composed with other modules. Simbiotics then simulates and integrates only the processes defined in the model.

Novel functionality in Simbiotics include the processing of microscopic images to initialise the simulation of the spatial state, additionally the integration

of existing standards such as SBML ensure the accessibility and communication of models. We allow for the modelling of processes such as membrane transport through passive or active mechanisms, active cell motility due to flagellar or pili activity and chemotaxis. Simbiotics may also simulate cell surface adhesion through processes such as electrostatic and receptor-adhesin interactions. Extracellular-polymeric substances (EPS) may also be modelled, either through a particulate representation or via mass-spring kinetics which cause bacteria to adhere to the substratum and other cells. The integration of these multi-scale processes is a main contribution of Simbiotics, allowing for the modelling of large bacterial populations whilst capturing micro-processes in individual cells.

3.2 Modelling

We describe the main Simbiotics Library modules, elaborating on their functionality and the mathematics used in their calculations. These submodels are independent and can be attached to models to compose them into a full model specification. Description of how the library system and models are implemented programmatically is described in Chapter 5.

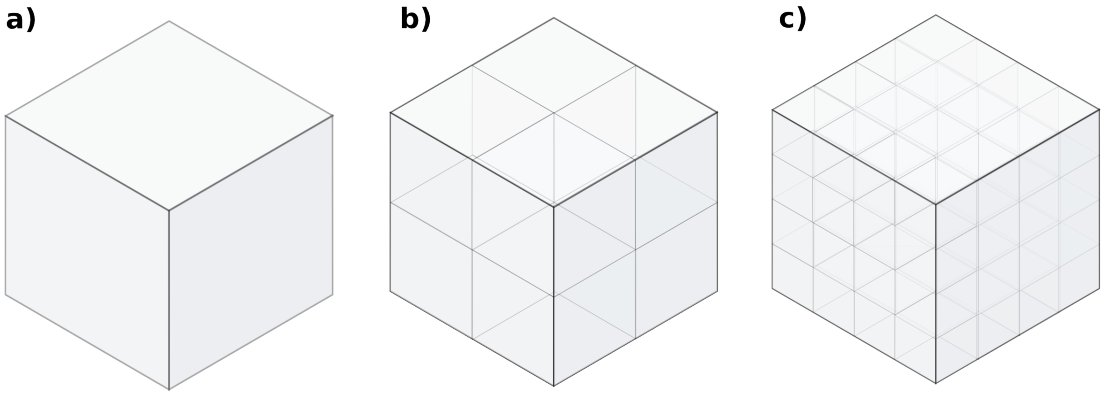


Figure 3.2: **a)** An unrasterised cuboidal simulation domain. **b)** Rasterisation of the domain where $G_d = 1$. **c)** Rasterisation of the domain where $G_d = 2$.

Environment

The spatial domain describes the spatial environment and boundaries of the simulation, it may be 3D or constrained to 2D. It is continuous space with a grid system to rasterise the domain into subvolumes for representing chemical distributions. The rasterisation is based on a binary division of the spatial domain into an octree structure, where the depth of the octree is defined by the grid depth G_d, which must be an integer. An example can be seen in Figure 3.2. Domain boundary conditions may describe a solid surface and its physicochemical characteristics, periodic boundaries may also be defined such that cellular and chemical entities enter the opposing side of the domain which they leave. An escape boundary can be defined such that entities are removed from the simulation when leaving the boundary. Additionally a boundary may describe a rate with which to introduce chemicals or bacteria into the environment, modelling a chemostat or stochastic bacterial world outside of the simulation domain.

Spherical cells

Cells may be *cocci*, represented as spheres. Each has a position vector $\boldsymbol{p}_i$ which represents its center as coordinates in 3D continuous space bounded within the simulation domain, a radius r_i and mass m_i. Additionally each cell has a velocity vector $\boldsymbol{v}_i$ which describes its current velocity as a 3D vector, and a 3D unit vector which describes the orientation of the body $\hat{\boldsymbol{\psi}}_i$.

Rod-shaped cells

Cells may be *bacilli*, represented as rods. Each rod cell is modelled by two points at positions $\boldsymbol{p}_i^a$ and $\boldsymbol{p}_i^b$, that are connected by a stiff spring. This representation allows for physically realistic force calculations, capturing rotation of the rod by distributing forces to the two points whilst maintaining a fixed rod length. This rod length can then be modified by cell growth calculations, that cause a symmetrical extension or contraction of the length, not affecting the radius of the rod. This representation produces a realistically growing bacillus cell, producing growth patterns as observed in experiments.

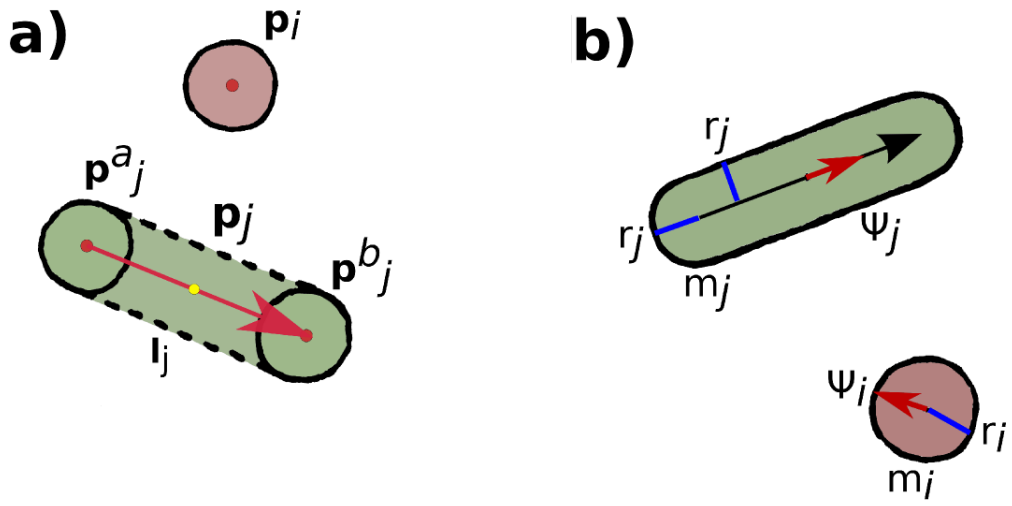


Figure 3.3: We consider two cells, a spherical cell (coccus) at position $\boldsymbol{p}_i$ and a rod-shaped cell (bacillus) at position $\boldsymbol{p}_j$. (a) Spheres have a position $\boldsymbol{p}_i$. Rods have start and end positions $\boldsymbol{p}_j^a$ and $\boldsymbol{p}_j^b$, a length $\boldsymbol{l}_j = \boldsymbol{p}_j^b - \boldsymbol{p}_j^a$, and a center position $\boldsymbol{p}_j$ which is equidistant between $\boldsymbol{p}_j^a$ and $\boldsymbol{p}_j^b$ along its length $\boldsymbol{l}_j$. (b) Both spheres and rods have an associated radius $\boldsymbol{r}_{i/j}$ and mass $\boldsymbol{m}_{i/j}$. Spheres have an orientation $\hat{\boldsymbol{\psi}}_i$, and rods have an orientation $\hat{\boldsymbol{\psi}}_j = \frac{\boldsymbol{l}_j}{l_j}$.

The two positions defining the length of the rod construct the line $\boldsymbol{l}_i = \boldsymbol{p}_i^b - \boldsymbol{p}_i^a$, which describes its length $l_i = |\boldsymbol{l}_i|$ and orientation $\hat{\boldsymbol{\psi}}_i = \frac{\boldsymbol{l}_i}{l_i}$. Rods are considered to be cylindrical along $\boldsymbol{l}_i$, with hemispherical caps. Each has a center of mass $\boldsymbol{p}_i$ which is the point along the rod axis that is equidistant from the two end points. Rods also have a radius r_i, mass m_i and each of its spheres has a velocity vector, $\boldsymbol{v}_i^a$ and $\boldsymbol{v}_i^b$.

A schematic for spherical and rod-shaped cells can be seen in Figure 3.3 (a) and (b).

Cell neighbourhood

A Verlet-list is implemented to store the nearest neighbours of a cell, for a cell i its nearest neighbour list is denoted as $\boldsymbol{M}_i$. A neighbouring cell j is included

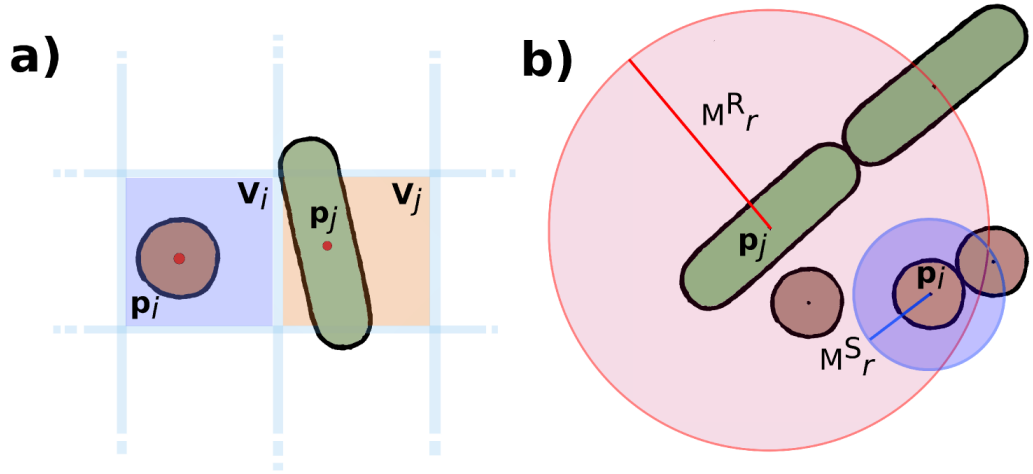


Figure 3.4: (a) A cell's center position determines which diffusion grid voxel $\boldsymbol{V}_{i/j}$ the cell is in. (b) An example of cell neighbourhoods. The red circle represents the neighbourhood range of the bacilli cell at $\boldsymbol{p}_j$, and the blue circle the range of the cocci cell at $\boldsymbol{p}_i$. For a given cell, other cells are considered in a neighbour if their center point exists within the range.

in this list if the absolute distance between cell's closest points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ is less than a given threshold M_r. For spherical cells, $M_r^S = r_i + r_{max}$, where r_{max} is the maximum cell radius in the system. For rod-shaped cells $M_r^R = 0.5l_i + 0.5l_{max}$, where l_{max} is the maximum rod length in the system. The total number of cells at any time t is denoted by $N(t)$. A schematic showing the representation of cellular agents can be seen in Figure 3.4 (b).

A cell's local environment also has chemical properties, its position $\boldsymbol{p}_i$ maps to a voxel V_i in the discretised grid space. This voxel contains a list of chemical species and corresponding concentrations present in that volume. The concentration at P_i may be an interpolation between V_i neighbouring voxel concentrations, this is calculated with Sheppard's method as implemented in the Cx3Dp component of the software [238]. Alternatively it may be assumed that each voxel has a uniform distribution within it. A schematic can be seen in Figure 3.4 (a).

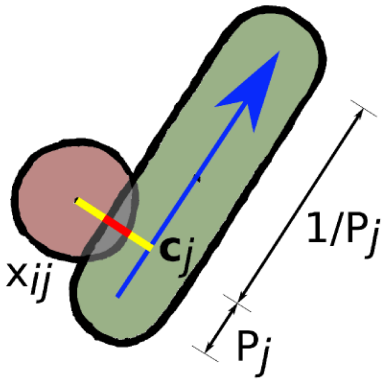


Figure 3.5: An example of a sphere and rod cell colliding, x_{ij} is the magnitude of the overlap between cells. For a bacilli the force of a collision is distributed to its start and end points according to P_j.

3.2.1 Physics

The motion of cells is determined by Newtonian dynamics, forces are translated into a change in velocity, and subsequently a change in velocity resulting in a change in position:

$$\frac{d\boldsymbol{p}_i(t)}{dt} = \boldsymbol{v}_i(t) \tag{3.1}$$

$$\frac{d\boldsymbol{v}_i(t)}{dt} = \frac{\boldsymbol{F}_i^{\mathrm{T}}(t)}{m_i}, \tag{3.2}$$

where $\boldsymbol{F}_i^{\mathrm{T}}$ is the total force experienced by a bacterial cell. The equation to calculate $\boldsymbol{F}_i^{\mathrm{T}}$ is user defined, and may have as many force components as desired. Here we present the default equation used to calculate $\boldsymbol{F}_i^{\mathrm{T}}$,

$$\boldsymbol{F}_i^{\mathrm{T}} = \sum_{j=1}^{M_i} (\boldsymbol{F}_{ij}^{\mathrm{C}} + \boldsymbol{F}_{ij}^{\mathrm{S}} + \boldsymbol{F}_{ij}^{\mathrm{E}}) + \boldsymbol{F}_i^{\mathrm{R}} + \boldsymbol{F}_i^{\mathrm{F}} + \boldsymbol{F}_i^{\mathrm{G}} + ..., \tag{3.3}$$

where i runs from 1 to $N(t)$, $\boldsymbol{F}_{ij}^{\mathrm{C}}$ is the force due to cell-cell collisions, $\boldsymbol{F}_{ij}^{\mathrm{S}}$ is the force due to specific adhesin receptor interactions, $\boldsymbol{F}_{ij}^{\mathrm{E}}$ is the force due to non-

specific electrostatic interactions, $\boldsymbol{F}_i^{\mathrm{R}}$ is the translational diffusion force, $\boldsymbol{F}_i^{\mathrm{F}}$ is the force of viscous drag on the cell and $\boldsymbol{F}_i^{\mathrm{G}}$ is the force of gravity. Together, $\boldsymbol{F}_i^{\mathrm{R}}$ and $\boldsymbol{F}_i^{\mathrm{F}}$ turn equations 3.1 through 3.3 into a Langevin Dynamics approach [162]. The Strömer-Verlet method is used for the numerical integration of positions and velocities due to forces. We calculate the force components individually, as follows.

3.2.1.1 Collisions

Cells experience forces due to collisions with other geometries in the 3D domain. For two cells at positions $\boldsymbol{p}_i$ and $\boldsymbol{p_j}$, $\hat{\boldsymbol{\theta}}_{ij}$ is the unit vector describing the line orientation between the cell centers from j to i, calculated as $\hat{\boldsymbol{\theta}}_{ij} = \frac{\boldsymbol{p}_i - \boldsymbol{p}_j}{|\boldsymbol{p}_i - \boldsymbol{p}_j|}$. Resolving collisions between spherical cell involves calculating response forces to apply to each cell. This is modelled as a strong spring which pushes cells apart, where $\boldsymbol{F}_{ij}^{\mathrm{C}}$ is the total force experienced by a cell due to its colliding neighbours, $\boldsymbol{K}_{\mathrm{C}}$ the spring constant for collisions and x_{ij} is the overlap distance of collision partners, $x_{ij} = r_i + r_j - |\boldsymbol{p}_j - \boldsymbol{p}_i|$,

$$\boldsymbol{F}_{ij}^{\mathrm{C}} = \begin{cases} K_{\mathrm{C}} x_{ij} \hat{\boldsymbol{\theta}}_{ij} & \text{if } x_{ij} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

The sphere at position $\boldsymbol{p}_i$ receives the force $\boldsymbol{F}_i^C = \boldsymbol{F}_{ij}^C$, and the sphere at position $\boldsymbol{p}_j$ receives the force $\boldsymbol{F}_j^C = -\boldsymbol{F}_{ij}^C$ according to Newton's third law.

A similar approach is taken for modelling collisions with rod-shaped cells. For the two colliding rod lineThe ratio along the rods which the points lie P_i and P_j are calculated as $P_i = \frac{|\boldsymbol{c}_i - \boldsymbol{p}_i^a|}{l_i}$ and $P_j = \frac{|\boldsymbol{c}_j - \boldsymbol{p}_j^a|}{l_j}$. We calculate the overlap x_{ij} between rods to be $x_{ij} = r_i + r_j - |\boldsymbol{l}_{ij}|$. We then calculate the total force that rods exert on each other in the same form as Equation 3.4, the distribution of this force onto the rod's end points follows the same approach as implemented in previous modelling work. [107]. Where $\boldsymbol{F}_i^{\mathrm{Ca}}$ is the force applied to point $\boldsymbol{p}_i^a$,

$$\boldsymbol{F}_i^{\mathrm{Ca}} = -(1 - P_i)\boldsymbol{F}_i^{\mathrm{C}} \tag{3.5}$$
$$\boldsymbol{F}_i^{\mathrm{Cb}} = -P_i\boldsymbol{F}_i^{\mathrm{C}} \tag{3.6}$$
$$\boldsymbol{F}_j^{\mathrm{Ca}} = (1 - P_j)\boldsymbol{F}_j^{\mathrm{C}} \tag{3.7}$$
$$\boldsymbol{F}_j^{\mathrm{Cb}} = P_j\boldsymbol{F}_j^{\mathrm{C}}. \tag{3.8}$$

Collisions between a sphere and rod are solved as a partial form of rod-rod collisions. For a sphere at position $\boldsymbol{p}_i$, we find the position $\boldsymbol{c}_j$ on the rod line segment $\boldsymbol{l}_j$ which forms the shortest line between them $\boldsymbol{l}_{ij} = \boldsymbol{c}_j - \boldsymbol{p}_i$. We calculate the overlap x_{ij}, forces $\boldsymbol{F}_i$ amd $\boldsymbol{F}_j$ and ratio P_j in the same manner as for rod-rod collisions. The sphere receives the full force $\boldsymbol{F}_i$ and $\boldsymbol{F}_j$is distributed onto the rods constituent spheres in the same manner as Equations 3.7 and 3.8. A schematic can be seen in Figure 3.5 (a).

Collision force responses may be modelled with Hertzian theory rather than the force expression in Equation 3.4. Hertzian theory models the elastic contact between colliding cells. In Equation 3.4 $K_{\mathrm{C}}x_{ij}\hat{\boldsymbol{\theta}}_{ij}$ is substituted with $E(r_i + r_j)^{1/2}x_{ij}^{3/2}$, where E is the parameter representing the elastic modulus of a cell[60, 71, 208].

3.2.1.2 Surface-mediated physical interactions

Adhesin receptor interactions are modelled as springs connecting cell geometries. An interaction between an adhesin-receptor pair q and s has a specific force constant K_{qs}^{S} associated with it. The extension of the spring is calculated as $\alpha_{ij} = l_{\mathrm{a}} - l_{\mathrm{r}}$. Where $l_{\mathrm{a}} = |\boldsymbol{p}_i - \boldsymbol{p}_j|$ is the actual length of the spring, and $l_{\mathrm{r}} = R_l(r_i + r_j)$ is the resting length of the spring. R_l being a spring relaxation factor allowing the spring to leave an offset between cell surfaces.

$$\boldsymbol{F}_{ij}^{\mathrm{S}} = \begin{cases} K_{qs}^{\mathrm{S}}\alpha_{ij}\hat{\boldsymbol{\theta}}_{ij} & \text{if } \alpha_{ij} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.9}$$

An adhesin-receptor interaction is reversible if a sufficiently large force pulls the cells apart. This is modelled as a maximum extension that the spring may reach before breaking. We calculate the maximum extension to be $\alpha_{ij}^{\max} = C_{pq} \cdot l_{\mathrm{r}}$,

where C_{pq} is the extension factor for adhesin and receptor p and q. If $\alpha_{ij} > \alpha_{ij}^{\max}$ the interaction spring is removed.

Cells experience forces due to non-specific interactions such as van der Waals interactions and electrostatic repulsion when their membranes are in close range. An established method for modelling these forces is DLVO theory [85, 167]. However this model operates on distances in the order of nanometers which are negligible in Simbiotics. We have a similar representation, modelling a proportional adhesive force as a two cell surfaces approach. $\boldsymbol{K}_{\boldsymbol{ij}}^{\mathrm{E}}$ is the adhesive force constant and d_{ij} is the distance between the cell centers defined as $d_{ij} = |\boldsymbol{p}_i - \boldsymbol{p}_j|$. Two cells interact if they are within range of each other's extended sphere of influence, defined as the cells radius r_i multiplied by a range factor r_E.

$$\boldsymbol{F}_{ij}^{\mathrm{E}} = \begin{cases} \frac{K_{ij}^{\mathrm{E}}}{d_{ij}^2}\hat{\boldsymbol{\theta}}_{ij} & \text{if } d_{ij} < \frac{r_E(r_i+r_j)}{2} \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

3.2.1.3 Passive motility

To calculate the force random fluid motion has on a free-floating cell, we use Equation 3.7 to find the force on a given particle at each moment in time. $\boldsymbol{K}_{\mathrm{R}}$ is a constant describing the maximum force the cell experiences. We generate a random number between 0 and $\boldsymbol{K}_{\mathrm{R}}$ and multiple it by a random unit vector $\hat{\boldsymbol{\eta}}$ to calculate the current force:

$$\boldsymbol{F}_i^{\mathrm{R}} = K_{\mathrm{R}}\hat{\boldsymbol{\eta}}. \tag{3.11}$$

In the numerical integration, we must take care to normalize the force by the inverse square root of the integration step.

To describe the effect of friction for each cell we calculate a drag force which is proportional to the friction coefficient K_{F}, representing the viscosity of the medium. The drag force is also proportional to the velocity of the cell $\boldsymbol{v}_i$:

$$\boldsymbol{F}_i^{\mathrm{F}} = -K_{\mathrm{F}}\boldsymbol{v}_i. \tag{3.12}$$

Gravity is modelled as a constant force acting on a cell,

$$\boldsymbol{F}_i^{\mathrm{G}} = K_{\mathrm{G}} m_i \hat{\boldsymbol{\gamma}}, \tag{3.13}$$

where K_{G} is the gravitational acceleration constant, m_i is the mass of the cell and $\hat{\boldsymbol{\gamma}}$ is the unit vector describe the direction of the force, pointing to negative y.

3.2.1.4 Active motility

Bacteria can be actively motile due to flagella micro-motility or pili mediated twitching-motility, these processes may be deployed to accomplish a random walk or chemotaxis [168, 169, 214].

Micro-motility in species such as *Escherichia coli* involve run and tumble phases, in which bacteria alternate between accelerating forwards and rotating in place[107]. We model this by probabilities $\boldsymbol{p_{endrun}}$ and $\boldsymbol{p_{endtumble}}$ with which the bacteria switch from a run or a tumble into the alternate state. During the run phase a constant force $\boldsymbol{F_{endrun}}$ is applied to the bacteria in the direction it is facing $\hat{\boldsymbol{\psi}}$. During the tumble phase we assign a new orientation $\hat{\psi}$ to the cell by generating a random unit vector. No directional force is applied to the bacteria when tumbling.

Twitching motility is modelled using the same algorithm as the micro-motility with different parameters. Both $\boldsymbol{p_{endrun}}$ and $\boldsymbol{p_{endtumble}}$ are relatively high, resulting in low persistence rapid movements.

Chemotaxis is modelled using a modified version of the micro-motility run and tumble dynamics, implemented similar to the Keller-Segel method [117, 224]. Cells perform a run and tumble and sample the concentration of the chemoattractant at periods of $\Delta t_{\mathrm{memory}}$ representing their sensory memory. Cells compare their current concentration $C(t)$ with the previous concentration they experienced $C(t - \Delta t_{\mathrm{memory}})$. This is calculated by $C(t) - C(t - \Delta t_{\mathrm{memory}})$, if the value is less than 1 the cell is descending the gradient and has a high probability to tumble. If the value is greater than 1 we know we are ascending a gradient or traversing a plateau, we calculate the gradient strength by how much $C(t) - C(t - \Delta t_{\mathrm{memory}})$ is above 1. The cell has a probability to tumble $\boldsymbol{p_{endrun}}$ that is inversely propor-

tional to the gradient strength, such that cells ascending a gradient are less likely to stop running.

3.2.2 Chemistry

Simbiotics allows for custom definition of chemical species with their respective diffusion and degradation rate constants. Chemicals can exist in the extracellular space or within cells and can be transported across membranes via a variety of mechanisms. Chemical reactions occur in intracellular compartments that are elaborated on in the *Cell growth and death* section (Section 3.2.3.1).

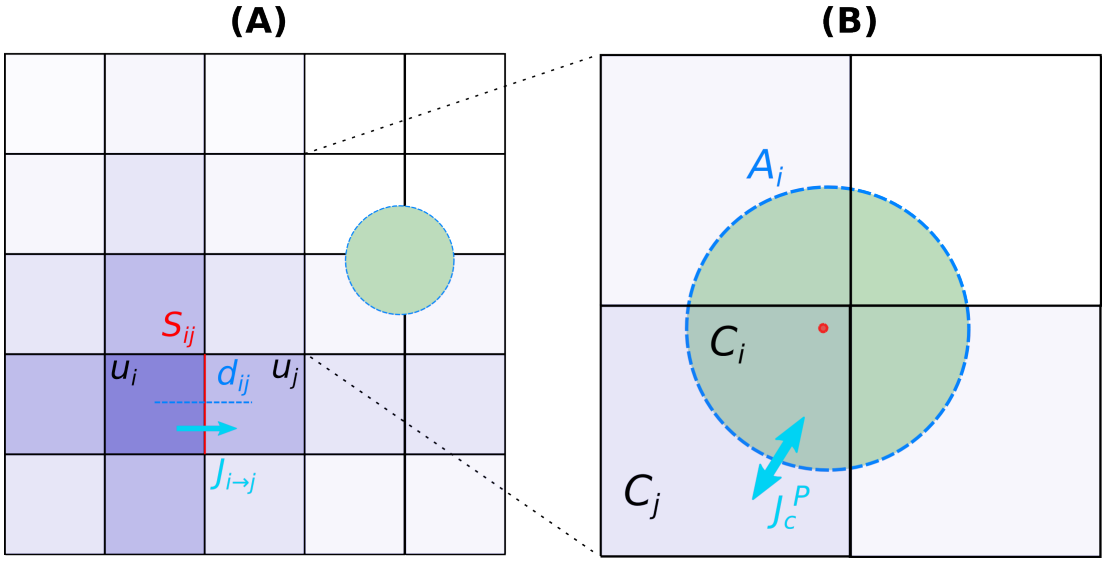


Figure 3.6: **(A)** Schematic of extracellular diffusion, showing localised chemical concentrations in the domain rasterisation, and the flux between the voxels of that rasterisation. **(B)** Membrane transport occurs between the intracellular compartment and the grid voxel the cells center point exists in.

3.2.2.1 Extracellular diffusion

Extracellular diffusion is implemented with the finite volume method [16]. The simulation domain is decomposed into regular non-overlapping subdomains. The flux between neighbouring subdomains is calculated for each chemical species as

follows:

$$J_{\mathrm{i}\,\rightarrow\,\mathrm{j}} = D_c \frac{S_{\mathrm{ij}}}{d_{\mathrm{ij}}}(u_{\mathrm{j}} - u_{\mathrm{i}}), \tag{3.14}$$

where $\boldsymbol{u_i}$ and $\boldsymbol{u_j}$ are the concentrations of a chemical species in the two neighbour subdomains, D_c is the corresponding diffusion coefficient for that chemical species and S_{ij} is the cross-section connected the two subdomains, and d_{ij} is the distance between the center points of the two subdomains. A schematic can be seen in Figure 3.6 (a).

The only extracellular reaction modelled is degradation, to calculate this a rate law can be defined for each chemical species. Where A is a chemical species and k_A is its rate of degradation:

$$A \xrightarrow{k_A} \emptyset. \tag{3.15}$$

One may also describe chemical sources and sinks, a chemostat adjacent to any simulation domain boundary, a flux of bacteria into the domain through boundaries, and a basic flow-chamber which models a constant flow across the entire domain.

3.2.2.2 Membrane transport

Chemicals can pass through cell membranes via either passive or active transport mechanisms. Passive membrane transport is solved in a similar manner as described in the Diffusion section (Section 3.2.2.1), such that the flux is only from high to low concentrations [142]. The flux due to passive transport mechanisms for a given chemical species is denoted by J_c^{P}, where A_i is the surface area of the cell, C_i is the concentration of the chemical in the cell, and C_j is the concentration of the chemical in the extracellular compartment which the cell center point reside in. A membrane permeability factor for individual chemical species P_c can be defined such that the flux is proportional to a chemical's permeability:

$$J_c^{\mathrm{P}} = P_c\, A_i\, (C_{\mathrm{j}} - C_{\mathrm{i}}). \tag{3.16}$$

A schematic can be seen in Figure 3.6 (b).

Active transport is modelled via a Monod function that calculates the flux based on the source concentration, it is a unidirectional flux and the source can be set to be either intracellular or extracellular [215]. The flux due to active transport mechanisms for a given chemical species is denoted by J_c^{A}, where C is the chemical concentration at the source, K_c is the half-saturation constant of the chemical flux, and Q_c is the maximum flux at which the active transport mechanism can work for that chemical species:

$$J_c^{\mathrm{A}} = Q_c \frac{K_c}{K_c + C}. \tag{3.17}$$

3.2.3 Biology

A wide range of biological processes are implemented, including cell growth kinetics and metabolic rules, cell division, motility, quorum-sensing through membrane transport, cell-cell and cell-surface adhesion as well as gene regulatory networks. Bacteria can also produce extracellular polymeric substances, which can form an extacellular matrix. We describe next the set of modelling decisions and simplifications made in order to capture and integrate these various processes.

3.2.3.1 Cell growth and death

As bacteria grow their mass increases and we calculate the change in mass $\boldsymbol{\Delta m}$ for the current timestep based on growth and maintenance kinetics. Here we describe kinetic representations that are valid in Simbiotics, we follow a similar approach to iDynoMiCS [126]. The change in the mass of a bacterial cell is based on the calculated growth rate μ_i., where μ_i is a function of the depending nutrient concentration S_i in the local extracellular compartment $V_{\boldsymbol{p}_i}$,

$$\frac{dm_i}{dt} = \mu_i(S_i). \tag{3.18}$$

Bacterial growth can be modelled as a constant process ignoring substrate dependence. Bacteria grow according to a growth rate μ_i that is calculated by two parameters, the first parameter $\boldsymbol{G}_{\mathrm{r}}$ is the mean growth rate, and the second parameter $\boldsymbol{G}_{\mathrm{v}}$ is the growth variation value. The actual growth rate μ_i is calcu-

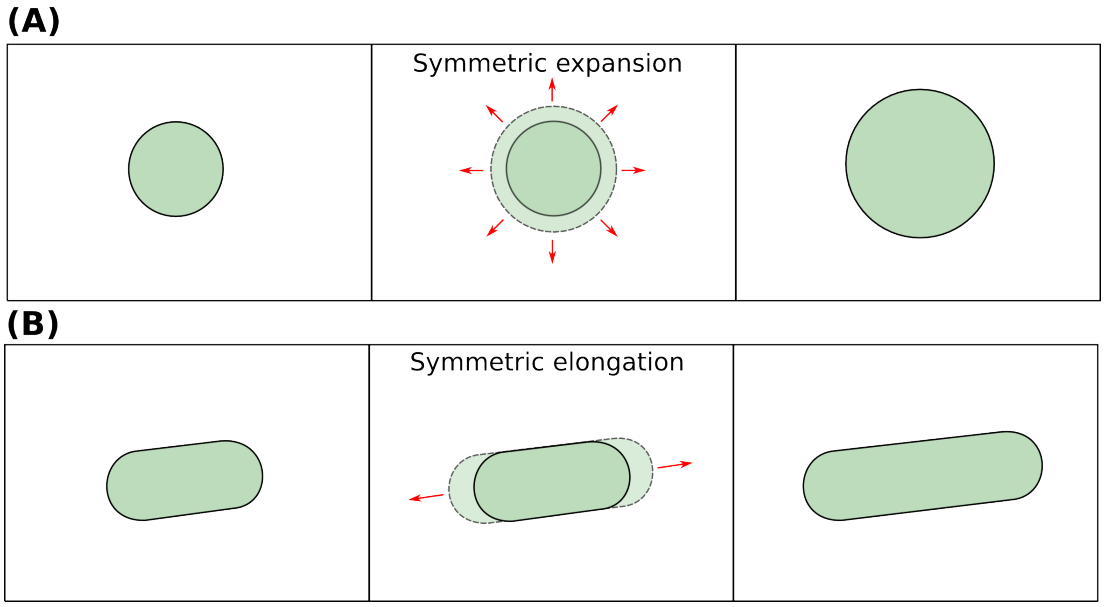


Figure 3.7: Schematics of growing cells. **(A)** Spherical (cocci) cells increase in radius as their mass grows, expanding symmetrically. **(B)** Rod-shaped cells (bacilli) increase in length as their mass grows, elongating symmetrically.

lated as follows, with rnd being a uniformly distributed random number between 0 and 1.

$$\mu_i = G_\mathrm{r} - (G_\mathrm{v} * G_\mathrm{r}) + (2 * rnd * G_\mathrm{r} * G_\mathrm{v}) \tag{3.19}$$

More complex growth dynamics can be specified using reaction kinetics, considering factors such as cell maintenance and available nutrient concentration. A variety of reaction kinetics are implemented which are listed in Table 2, one can compose these kinetics to design custom nutrient-based growth dynamics.

Additionally the modeller can write ordinary differential equations (ODEs) describing the growth of bacterial. This is achieved by describing how the mass changes over time ($\frac{dM}{t}$), and can include variables such as the cell's current intracellular of extracellular molecular concentrations.

Spherical cells die if their radius is below a minimum radius r_min, and rod-shaped cells die if their length is below a minimum length r_len. When a cell dies its geometry is completely removed from the simulation, any intracellular chemicals are then moved to the extracellular compartment in the diffusion grid

Growth kinetic	Equation
First-order kinetic	$\mu = G_r \pm G_v$
Monod kinetic	$\mu = \frac{S}{K_S+S}$
Simple inhibition	$\mu = \frac{K_i}{K_i+S}$
Hill kinetic	$\mu = \frac{S^h}{K_S^h+S^h}$
Haldane kinetic	$\mu = \frac{S}{K_S+S+\frac{S^2}{K_i}}$

Table 3.1: Growth kinetic equations, were μ is the growth rate, S is a given substance concentration and K is the half-saturation constant of a given substance.

which contained the cell's center of mass. Cell growth and death are implemented as an extended version of the dynamics used in the iDynoMiCS software [130]. Growth dynamics may now be coupled to either extracellular or intracellular cell chemical concentrations, additionally the distribution of intracellular chemicals upon cell death is implemented.

We assume that cell biomass density remains constant throughout the cell cycle [209], therefore when a cell grows in mass it is expressed by a growth in volume. As a coccus cell grows its radius r_i increases, expanding symmetrically as can be seen in Figure 3.7 (A). For a bacillus cell, growth is only along the length of the cell l_i, as variations in its width are negligible in comparison [35], elongating symmetrically as seen in Figure 3.7 (B).

3.2.3.2 Cell division

A binary fission library module implemented. Cell division occurs upon a cell reaching twice its original mass [35, 180]. We consider child cells to inherit about half of the mass of the parent cell [154].

This is implemented for coccus cells, however an issue arises with this method for bacillus cells. Preserving mass results in the two child cells being longer than the parent cell, meaning that they may be placed in the model overlapping adjacent cells, resulting in collisions producing unrealistic growth patterns. Instead, bacilli cells are modelled as dividing upon reaching twice their original length, such that the two child cells can be positioned within the volume of the parent cell.

Spherical cells

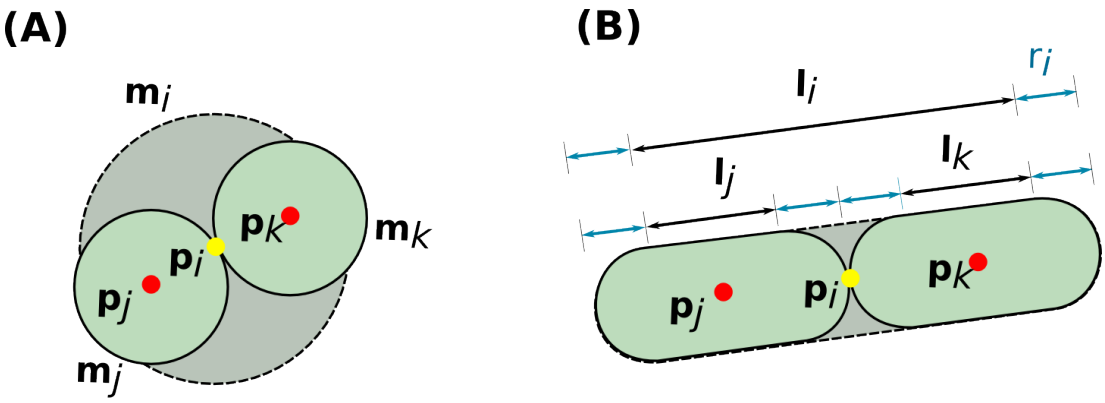


Figure 3.8: **(A)** Schematic of cocci cell dividing. Division occurs upon reaching twice the original cell mass. The child cells are positioned within the parent cell's volume such that they don't overlap. $\boldsymbol{p}_j$ is the center position of the dividing parent cell, which has mass $\boldsymbol{m}_j$. The child cells masses preserve the mass of the parent cell, such that $\boldsymbol{m}_k + \boldsymbol{m}_l = \boldsymbol{m}_j$. Their center of masses $\boldsymbol{p}_k$ and $\boldsymbol{p}_l$ are determined by calculating the radius of the child cells based on their mass, and thus the offset they must be in order to not overlap. **(B)** Schematic of bacillus cell dividing. Division criteria is the cell reaching twice its original length (including the radius of hemispherical caps). The child cells are positioned within the parent's volume and do not overlap, $\boldsymbol{p}_j$ and l_j are the position and length of the dividing cell. The positions of the child cells are $\boldsymbol{p}_k$ and $\boldsymbol{p}_l$, and they have lengths l_k and l_l. Both child cells inherent the same radius r_j.

Dividing spherical cells form two child cells which are placed either side of the parent's centre of mass, as seen in Figure 3.8 (A). The size and placement of child cells are determined by first finding the mass that the children inherit, and then calculating the corresponding volumes and radii, followed by the offset between the child cells in order for them not to overlap.

The parent cell's mass is divided by ratio D_r, which is calculated as

$$D_\mathrm{r} = 0.5 + ((D_\mathrm{v} * rand) - (\frac{D_\mathrm{v}}{2},)) \tag{3.20}$$

where $rand$ is a uniformly distributed random number between 0 and 1, and D_v is a parameter to the amount of noise around exactly a 50/50 split of mass between children. The two masses m_j and m_k are then calculated as

$$m_j = D_\mathrm{r} m_\mathrm{T} \tag{3.21}$$
$$m_k = (1 - D_\mathrm{r}) m_\mathrm{T}, \tag{3.22}$$

where m_T is the total mass of the dividing parent cell. The resulting volumes the child cells are then calculated using this mass, assuming a constant density.

$$V_j = m_l / \rho_i \tag{3.23}$$
$$V_k = m_k / \rho_i \tag{3.24}$$
$$. \tag{3.25}$$

The radii of the child r_i and r_k cells can then be calculated with $r^3 = \frac{V}{\frac{4}{3}\pi}$. These radii can then be used with a randomly generated unit vector $\hat{\boldsymbol{\psi}}_i$ to find the center points $\boldsymbol{p}_j$ and $\boldsymbol{p}_k$ of the child cells,

$$\boldsymbol{p}_j = \boldsymbol{p}_i + (r_j)\hat{\boldsymbol{\psi}}_i \tag{3.26}$$
$$\boldsymbol{p}_k = \boldsymbol{p}_i - (r_k)\hat{\boldsymbol{\psi}}_i. \tag{3.27}$$

The two child cells are created at these positions, such that they do not overlap but are touching. All intracellular chemical molecular amounts are also divided according to this ratio. Additionally all cell defined processes are copied across to the new child cell.

Rod-shaped cells

Rod-shaped bacteria replicate upon reaching twice their original length [50]. This is modelled as two child cells being placed within the volume of the dividing parent cell, as can be seen in Figure 3.8 (B). This is slightly different to dividing spherical cells, as it does not conserve mass in order to make sure there is not additional volume that child cells consume.

For a dividing rod whose center of mass is at position $\boldsymbol{p}_i$ and has an orientation $\hat{\boldsymbol{\psi}}_i$, we calculate the position of child cell centers of mass $\boldsymbol{p}_j$ and $\boldsymbol{p}_k$ as follows,

$$\boldsymbol{p}_j = \boldsymbol{p}_i + (0.5l_i + r_i)\hat{\boldsymbol{\psi}}_i \tag{3.28}$$

$$\boldsymbol{p}_k = \boldsymbol{p}_i - (0.5l_i - r_i)\hat{\boldsymbol{\psi}}_i. \tag{3.29}$$

Both child cells inherit the same radius as the parent cell and are of identical lengths, we must take care to subtract the radius from the child cell length, so that both child cells fit within the volume of the parent cell, $l_j = l_k = 0.5l_i - r_i$.

3.2.3.3 Extracellular polymeric substances

Bacteria can produce extracellular polymeric substances (EPS) [128, 195]. EPS can be modelled via two mechanisms. The first is an implicit form modelling EPS via mass-spring dynamics connecting adjacent cells. This implementation utilises the same algorithm as the specific cell-surface interactions as described in the Physics section (Section 3.2.1.2). This representation assumes that when two cells are close by their relative positions are constrained by the presence of adhesive EPS, thus a spring is formed between two neighbouring geometries where the distance between their center positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ is less than the sum of their radii multiplied by some range factor $R_{\text{EPS}}(r_i + r_j)$.

An alternative form is to model EPS as particles that exist as geometric agents in the environment. This is modelled in a similar manner to iDynoMiCS [130]. Bacterial cells have capsular EPS which is bound to their membrane, this capsule has a volume V_i^C associated with it, and it is added to the cell's volume to calculate the cells total radius considering both active (cellular) and inactive (EPS) biomass. Upon V_i^C reaching a threshold V_{EPS}, an EPS particle is added to the local environment at a random position adjacent to the cell. The EPS particle has the same volume as V_{EPS} and the capsule volume V_i^C is reset to 0.

EPS particles are modelled as passively-motile spheres which may undergo specific and non-specific interactions with neighbouring EPS particles and cells, as described in the Physics section (Section 3.2.1.2).

3.2.3.4 Boolean networks, ODE, Gillespie and SBML integration

Intracellular processes such as gene regulation and metabolism can be modelled using either SBML models, Boolean networks, Gillespie models, or sets of ordi-

nary differential equations, all being widespread formalisms to model cell internal dynamics.

Boolean network representations may be used to represent networks of genes, or higher level concepts such as phenotype states or decision making rules. Nodes in the network are in one of the discrete states *on* or *off*, with directed arcs between nodes to describe an *activation* or *inhibition* relation. Arcs into a node are turn into propositional logic states, form transition rules between nodes. All node transitions are solved synchronously and then updated.

The modeller may specific their own sets of ordinary differential equations, for which a 4th order Runge-Kutta integrator is implemented.

A Gillespie simulation module is implemented, allowing for stochastic chemical processes to be represented and integrated using the general method.

An SBML solver LibSBMLsim[210] is integrated, allowing for each cell to potentially have its own SBML model. SBML can be used to describe the metabolic or gene regulation of a bacterial cell. Any state variable or parameter of the SBML model can be *set* or *get* by other submodels of a cell in Simbiotics, allowing for the full integration of SBML (except for *events*).

For all representations the variables are accessible from other modules, this enables a coupling between cell internal dynamics and interactions with their environment. For example these methods can be linked with the membrane transports module to allow for chemical species to permeate the cell membrane, simulating chemical signalling. Another example includes modelling of membrane structure expression based on gene regulation, such that gene regulation effects how cells adhere to each other.

3.3 Analysis

Simbiotics has a built-in analysis suite; this consists of additional submodels that can be attached to a model specification to perform measurements. A virtual lab is implemented for more in-depth analysis, offering some typical wetlab instruments and mathematical analysis features, as well as scheduled model interactions. Analysis tools and data exporters can be attached to the model specification and used to collect data and process it throughout the simulation. Users may

define schedules which automate model analysis modules, programming specific model interactions or data collection and processing events.

The virtual lab currently consists of microsensors for sampling chemical field, biomass, biofilm height and gene expression profiling. Model events such as pipetting a chemical at a given time are also included. A simulated spectrophotometer to obtain optical density measurements is also implemented. Additional statistics tools include measurements of the mean squared displacement and velocity autocorrelation function of bacteria, as well as detailed data gathering regarding cell interactions, gene expression and spatially distributed biomass concentrations. One can also run a biofilm height measurement algorithm that can encode a heatmap image of biofilm heights as well as measuring the average and standard deviation of measurements. A general data collector is implemented, which allows the modeller to append desired properties of the system they wish to know, such as cell species number, chemical concentrations, simulation execution time, gene expression and number of cell-cell interactions.

An optical real time 3D rendering is provided by the interface, this allows for the custom rendering of different model components. Live graph-plotting is available to show model statistics during simulations. Snapshots and videos of the simulation can be taken, with optional filters to allow for Z-stack slices, filtered cell populations or cell state highlighting. Snapshots consist of all agent geometry encodings and user-selected states, they may be loaded back into Simbiotics which reconstructs the physical state and allows for the navigation of the 3D model. Additionally a basic PovRay exporter can convert a Simbiotics snapshot into a PovRay image file to be rendered.

All lab modules may be attached to a model specification in the same way modules are attached to describe system dynamics. Modules have parameters for users to tune their behaviour. Characterisation of systems using the virtual lab may be achieved through parameter sweeps (parameter sensitivity analysis). The user may set a model parameter to be a sweep, such that the simulation will run multiple version each with a different parameter value in the sweep.

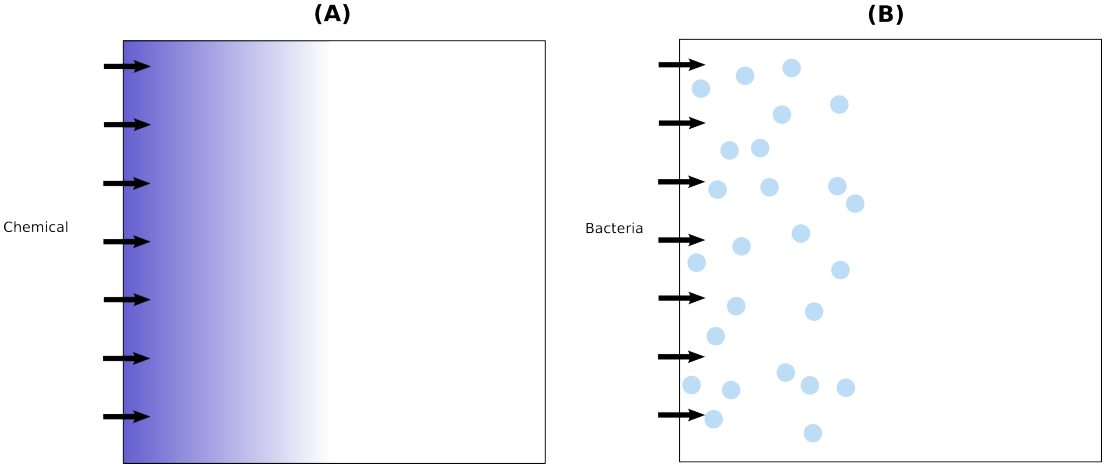


Figure 3.9: **(A)** A chemostat models a chemical flux at a boundary of a domain. **(B)** A bactostat models an influx of bacteria at a boundary of the simulation domain.

3.3.1 Virtual lab

Chemostat

A chemostat is implemented which models fluxes of chemicals into and out of the simulation domain. Chemostats can be attached to a domain boundary, and have rates assigned to them describing the flux rate of specific chemicals defined in the model. A given chemical C is introduced into (or removed from) the domain at a flux rate $\boldsymbol{r_C}$. Chemostats can also be set to achieve a desired concentration level. Defined fluxes can be set to be per unit area if the modeller chooses. A schematic can be seen in Figure 3.9 (a) showing the flux of chemicals into the domain.

Bactostat

A bactostat is similar to a chemostat, it represents the flux of bacteria with the larger environment beyond the simulation domain, and can be attached to the boundaries of the domain. The bactostat has bidirectional flux, bacteria which are leaving the simulation domain are removed, and rate $\boldsymbol{r_{bac}}$ describes the probability a bacteria will be introduced into the simulation per unit time. As with the chemostat, defined fluxes can also be set to be per unit area if the modeller chooses. The rate $\boldsymbol{r_{bac}}$ is proportional to the density of cells outside

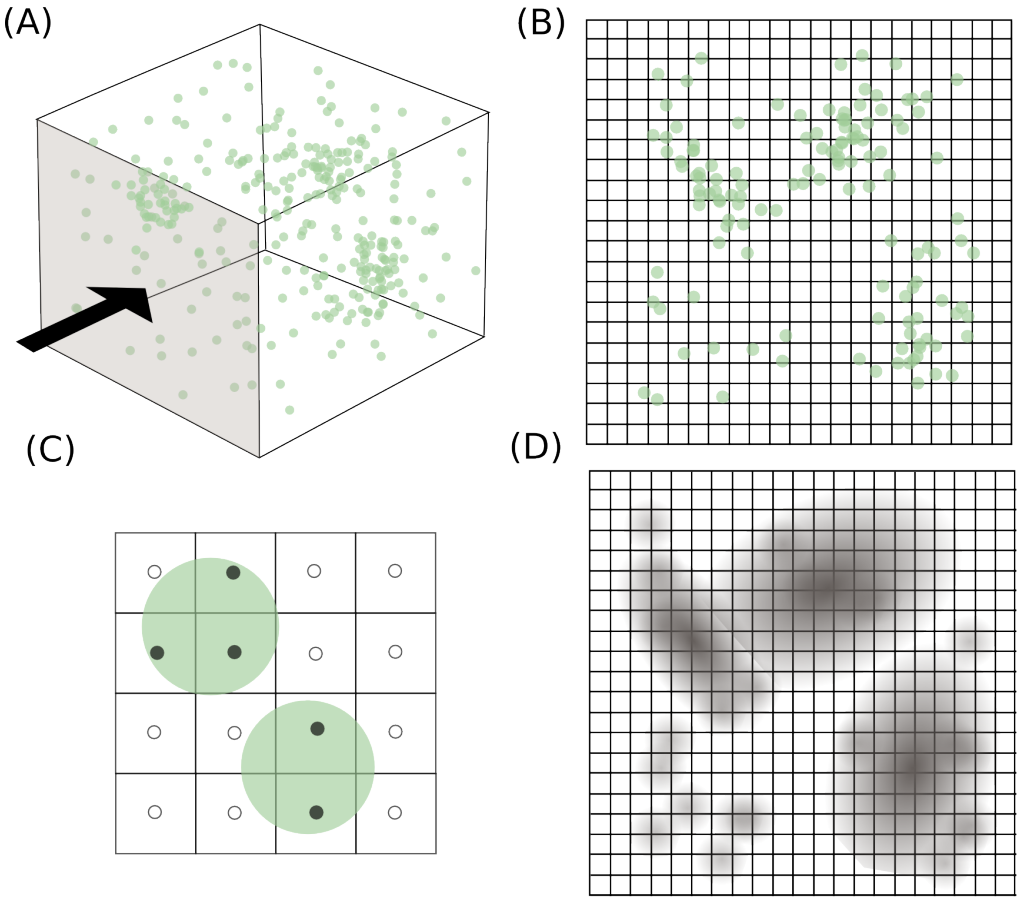


Figure 3.10: Schematic of simulated spectrophotometer in Simbiotics. **(A)** Spectrophotometers work by projecting light into one side of a sample and detecting the ratio of light which exits the other side. **(B)** In Simbiotics this is achieved by projecting the population onto the boundary that the light enters, and rasterising that space into a 2D grid. **(C)** Each grid cell in the rasterisation can be considered to be a trajectory of light rays travelling through the sample. If the center point of a grid cell intersects with any projected cells, then it's considered that the light is blocked. **(D)** The entire rasterisation is assessed as seen in (C), determining the ratio of light that has successfully travelled through the sample.

of the simulation domain $\boldsymbol{d_{bac}}$. The bactostat can have $\boldsymbol{d_{bac}}$ as a static density, such that the flux of bacteria into the domain is constant regardless of domain dynamics. Alternatively $\boldsymbol{d_{bac}}$ can be an accumulative density, such that bacteria leaving the domain increase the flux back into the domain. A schematic can be seen in Figure 3.9 (b) showing the flux of bacteria into the domain.

Spectrophotometer

A simulated spectrophotometer allows for the collection of optical density readings from the model. This is achieved by projecting the cells onto the face of the domain from which the light enters, then partitioning this into a 2D grid. We consider the light not to pass through a grid voxel if a cell intersects it. This method does not consider the diffraction of light, thus some constant is applied to optical density readings to account for this effect. A schematic showing this process can be seen in Figure 3.11.

3.3.2 Mathematical tools

Mean squared displacement

Measuring the mean squared displacement (MSD) of agent geometries involves considering the initial position $\boldsymbol{p(0)}$ of all geometries in a given set $\boldsymbol{G}$. At a given time $\boldsymbol{t}$ we calculate the displacement $\boldsymbol{r}$ for all geometries from their initial position $\boldsymbol{p(0)}$ and takes the average. This average is squared to give the mean squared displacement $\boldsymbol{msd}$.

$$msd \quad = \quad < r_i(t)^2 > \quad = \quad < (p_i(t) - p_i(0))^2 > \tag{3.30}$$

Velocity autocorrelation function

The velocity autocorrelation function (VAC) of agent geometries is calculated by considering the initial velocity $\boldsymbol{v(0)}$ of all geometries in a given set $\boldsymbol{G}$. At a given time $\boldsymbol{t}$ we calculate the dot product of all geometry's initial velocity $\boldsymbol{v(0)}$ with their current velocity $\boldsymbol{v(t)}$ and take the average, this is the velocity autocorrelation function $\boldsymbol{vac}$

$$vac \quad = \quad < v_i(0) \cdot v_i(t) > \tag{3.31}$$

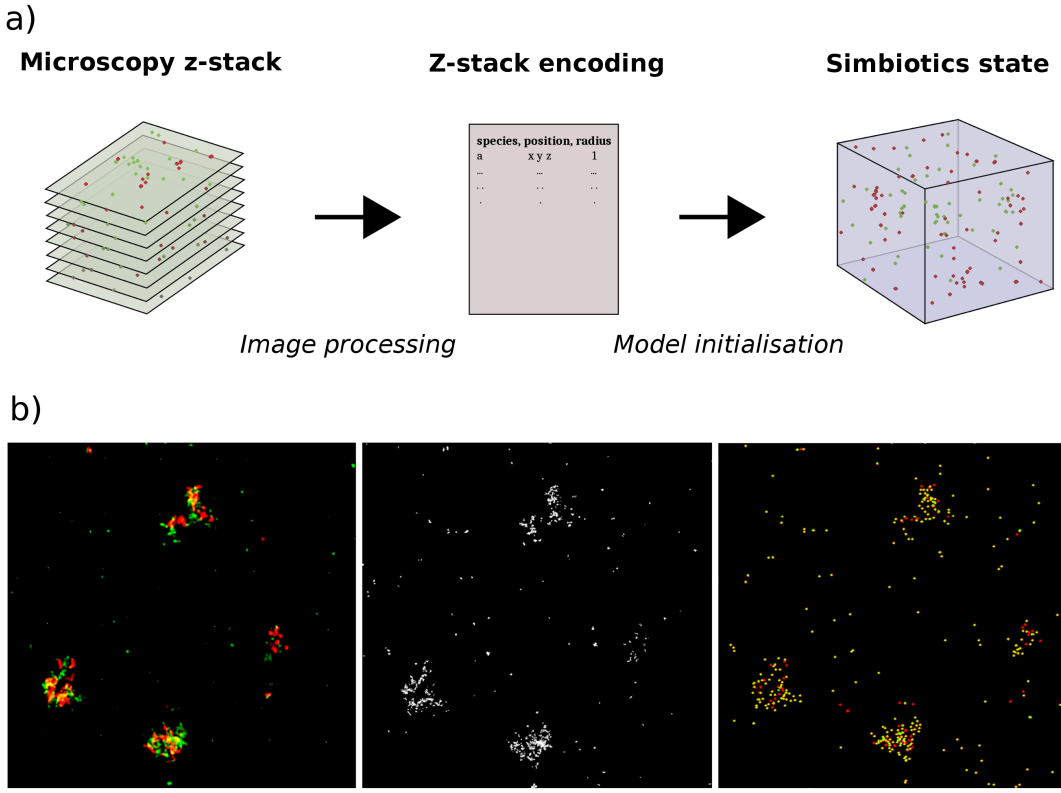


Figure 3.11: Microscopy image loading, showing the processing from the original microscopy z-stack to Simbiotics model state. (a) Schematic showing that a z-stack undergoes image processing to extract and encode features such as cell positions. This encoding is then used in Simbiotics model initialisation to model state. (b) Example of microscopy image processing. left: 2D projection of microscopy z-stack. middle: 2D projection of image processed z-stack, from which image features may be extracted. right: 2D projection of Simbiotics model, showing loaded cellular agents in the same configuration as the original z-stack.

3.3.3 Microscopy image processing

A major contribution of Simbiotics is the ability to process microscopy images of 2D and 3D bacterial conformations. This allows for the initialisation of simulations from realistic biological configurations.

To initialise the spatial configuration of bacteria one may use microscopy Z-stack images. Image processing requires three steps: we first apply a threshold image segmentation that generates binary data representing the Z-stack, we then

identified individual cells and encode details such as center position and radius in a data file. The data file can then be loaded in the configuration file that Simbiotics uses to initialise the cell population. This process is depicted in Figure 3.11 where a Z-stack image is loaded into Simbiotics.

For a multispecies population one may use image analysis techniques to identify cell species, for visually similar species one may use staining techniques to differentiate. Once the cell species has been identified this can be used to attach relevant model processes describing the cell's behaviour, such as an SBML model and other Simbiotics submodels.

This microscopy image processing allows for the simulation of an imaged population, as well as the simulation of a subset of the population through some filtering process. Through this one may observe the effect the filtered subpopulation has on the development of the population by the divergence of the filtered model from the original.

One may also compare the simulated state and the experimental state as the system evolves, iteratively changing model parameters to fit them to the experimental dynamics. This process could be automated, allowing for parameter fitting of models through refinement of the original specification based on the actual data.

3.4 Simbiotics integration loop

All of the processes described above can be integrated by the Simbiotics engine once composed into a model. This is done by a core integration loop that iterates through all model objects and integrates their solutions. The loop has the following form:

```
Start loop
– Solve extracellular chemical diffusion
– Solve cell behaviour modules
– Solve the physics (collisions and motion)
– Apply analysis modules (such as data exporting)
End loop
```

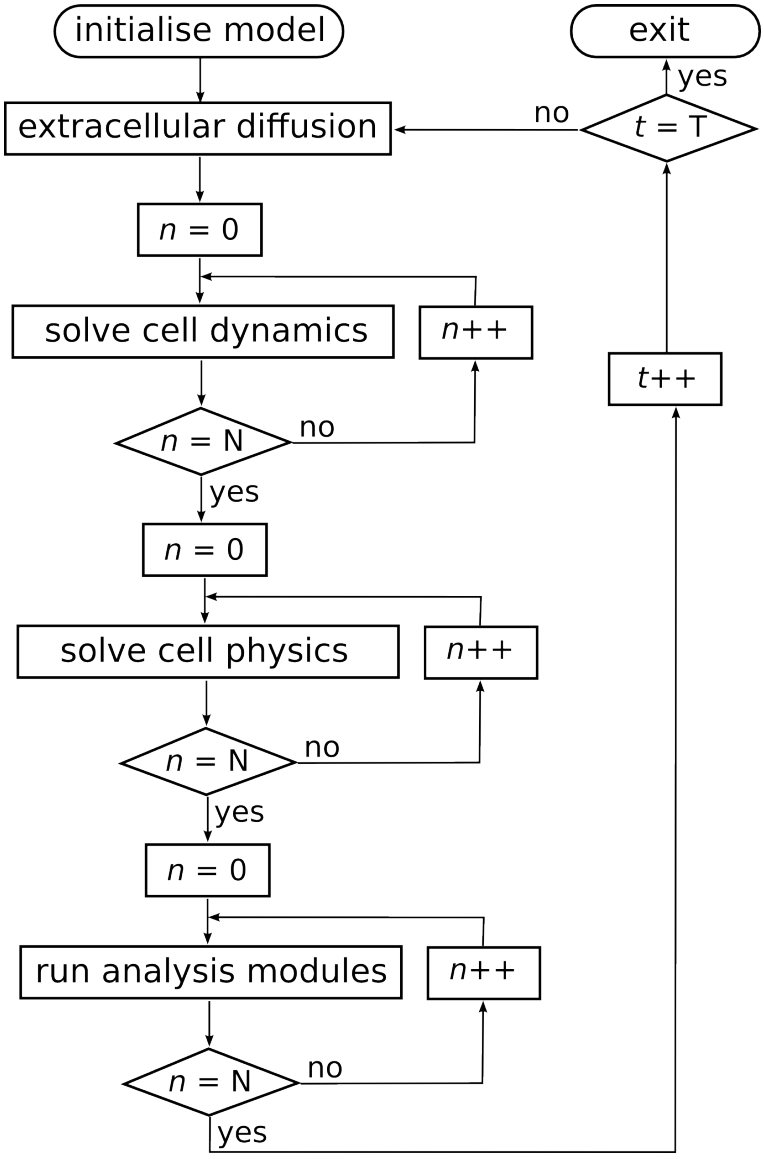


Figure 3.12: **Flow diagram of the simulation algorithm.** After initializing the model with N cells, the outer loop iterates the system through time. In each iteration the algorithm first solves for diffusion and decay of molecules in the environment using a finite difference approximation to the Fick equation. Then, for each cell in the simulation, its intracellular dynamics are solved and updated, then afterwards all cell motion is calculated. Finally the analysis modules (such as data exporters and model events) are executed. The algorithm is stopped after T time units have been simulated, or some other user-defined stopping criteria has been met.

The iteration starts by solving extracellular diffusion, calculating how chemicals in the extracellular space move around the domain (excluding membrane transport, as this is modelled as a cell behaviour process). Next all cell behaviour dynamics are solved, such as cell growth, division, gene regulation and membrane permeation of chemicals. The next step is calculating the physical forces in the simulation and resultant motion of agents. The final step is applying any programmatic changes to the model, such as the user specifying the pipetting of a chemical at a certain time in the simulation, and exporting any specified data to file. Further details on the program flow can be seen in Figure 28.

The loop integrates the module process across a time step t, that is a fixed value set in the Simbiotics configuration. This means processes that occur on a fast timescale, such as diffusion, constrain t to be a small value, with which all processes are integrated over.

3.5 Selecting parameter values

The values for parameters for the modelling features described above have been deduced in a range of manners. Some parameters can be found through literature data, others through experimentation and measurement, and some need to be fit through simulation. Not all parameters correspond to a physical property, and are used in modelling abstractions, however they must still be informed by the natural systems they are designed to model.

The core parameters for the Simbiotics modelling features are listed here (in the order they were presented in the modelling section for this chapter), with typical values and reference.

Parameters listed as having been derived from literature have been found in the papers cited in this chapter and the background, they include well known values such as cell radii that can be found from various sources. Parameters listed as having been derived from simulation fitting involve running simulations with different values until realistic behaviour is produced by the simulation, based on some measurable output (such as calibrating collision forces based on natural growth patterns and collision behaviour, or calibrating the diffusion grid depth based on stable diffusion dynamics).

Parameter	Description	Typical value	Source
G_d	Grid depth	0-7	Simulation fitting
r	Radius (cell)	$0.5\mu m$	Literature
m	Mass (cell)	$11pg$	Literature
l	Length (bacilli cell)	$2\mu m$	Literature
K_C	Collision spring constant	20-100	Simulation fitting
K_{qs}^S	Specific interaction pair	1-100	Simulation fitting
K_{ij}^E	Non-specific interaction pair	1-100	Simulation fitting
K_R	Passive motility force constant	0.1 - 2.0	Simulation fitting
K_F	Friction coefficient	0.01 - 10.0	Simulation fitting
K_G	Gravity coefficient	0.01 - 10.0	Simulation fitting
D_c	Diffusion coefficient	0.001 - 1000.0	Literature
J_{pc}	Membrane diffusion coefficient	0.001 - 1000.0	Simulation fitting
G_r	Growth rate	0.0001 - 0.01	Simulation fitting
G_v	Growth rate variation	0.0 - 0.5	Simulation fitting
D_r	Division ratio	0.5	Literature
D_v	Division ratio variation	0.0 - 0.1	Simulation fitting
t	Time step	0.0001 - 0.1	Simulation fitting

These basic parameters for Simbiotics vary in value depending on the system being simulated. The platform is unit agnostic, that being it does not force the value units, and it is down to the modeller to be consistent and correct with their inputs. Due to this if the modeller changes the time step t, they must change their other parameters that are time dependent taking into account this change to t.

Each model built with Simbiotics involves different features and thus different parameters. Parameters often need to be fit for each specific study, however models of similar systems can often have shared parameter values. For example parameters found in the validation tests (Chapter 4) were often valid in case study models, such as the collision force constant K_C. Other parameters however, such as the random walk force constant K_R, needed to be fit for each model in order to match the experimental diffusion coefficient.

The problem associated with finding parameters and justifying the their values when modelling is common, and Simbiotics provides many modules with parameters that can be used in different ways (depending on which method the modeller chooses to represent a specific phenomena, and then how they connect

those methods via model composition). Simbiotics provides an environment for rapid-prototyping and trial and error, where model development involves finding parameters that are available in literature first, and fitting other parameters based on those. This may take numerous iterations and can require control experiments with which to calibrate the model against.

3.6 Summary

This chapter has presented Simbiotics, a framework for integrative modelling of multicellular systems. The Simbiotics library provides a range of modelling and analysis features primarily for studying bacterial dynamics. The main features have been presented here, elaborating on their implementation and parameters. The analysis features have also been presented, showing how Simbiotics can act as a simple *in-silico* lab where virtual experiments can be designed, simulated and analysed. This chapter has also discussed parameter values for simulations, how these can be determined and typical values that are used.

There are numerous simulators for population dynamics, however there is yet to be a standardized platform for modelling bacterial populations in a multi-scale manner. Simbiotics provides an extendable modular framework in which the user can integrate a wide range of processes, including interfacing with standard formats such as SBML for modelling individual cells and microscopy images for describing spatial composition of populations. The extendable library and distributed CPU parallelization features allow for the scaling of Simbiotics functionality as it is further developed.

Chapter 4

Validation and testing of Simbiotics

This chapter presents a series of validation tests performed on Simbiotics model features. Stress tests and analysis of Simbiotics perfomance scaling are also presented. Literature results from existing population modelling tools are reproduced with Simbiotics. This chapter further addresses Objective 1, ensuring that the developed software's implementation is correct and produces accurate outputs.

4.1 Overview

To ensure the implementation of Simbiotics features are correct, and that they accurately model the intended biological phenomena, a series validation tests were performed. These act as sanity checks, for example ensuring a chemical concentration can not be negative, as well as ensuring that mass is conserved in the simulation. Additionally these tests deduced the stable parameter ranges for the diffusion integrator. Validation of features involved testing small library modules in simple models, checking that they gave the correct output based on their intended mathematical implementation. More complex models through the composition of library modules ensures that these methods are correctly integrated together.

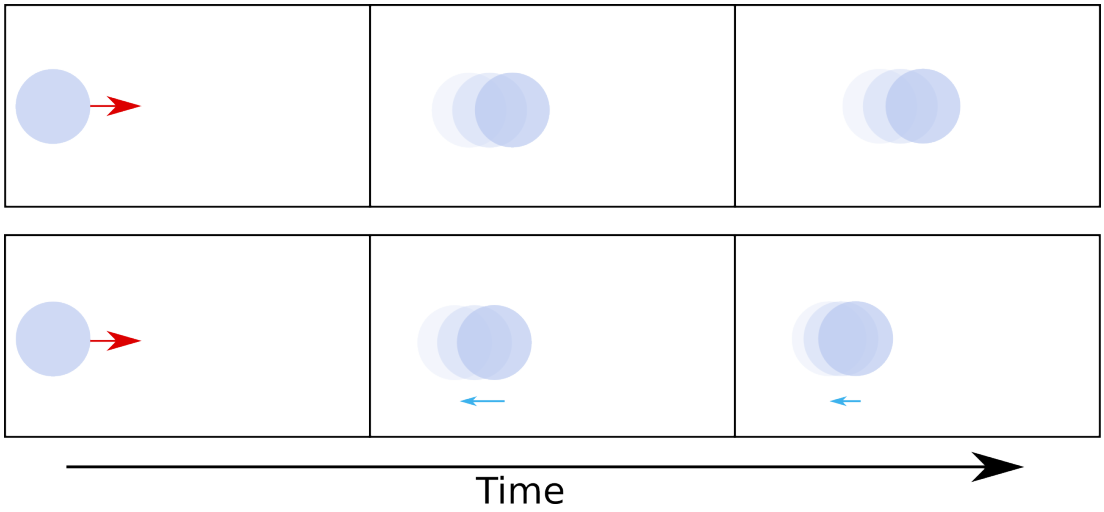


Figure 4.1: Schematic showing Test 1 - An instantaneous force is applied to a cell causing it to move. **Top row:** The cell does not experience any other forces and moves unimpeded. **Bottom row:** The cell experiences a frictional force (modelled as a drag force) as it moves through the space.

4.2 Validation tests

4.2.1 Physical integrator

To ensure the physical integrator (Strömer-Verlet integration) is correctly working, a series of increasingly complex tests were performed. To quantify system dynamics the following properties are obtained from the simulation: the velocity of cells, their squared displacement, and their velocity autocorrelation function (VAC).

4.2.1.1 Forces

Test 1 - Instantaneous force

A single cell is created in a cubic simulation domain, it is at rest (velocity = 0) and is in an environment void of any forces (including no friction.) An momentary artificial force $F = 10$ is then applied to the cell in the +X direction after 10 seconds. A schematic of Test 1 can be seen in Figure 4.1.

As seen in Figure 4.2, the cell's velocity, squared displacement and vac remain at 0 until the applied force at 10 seconds. The instantaneous force causes an

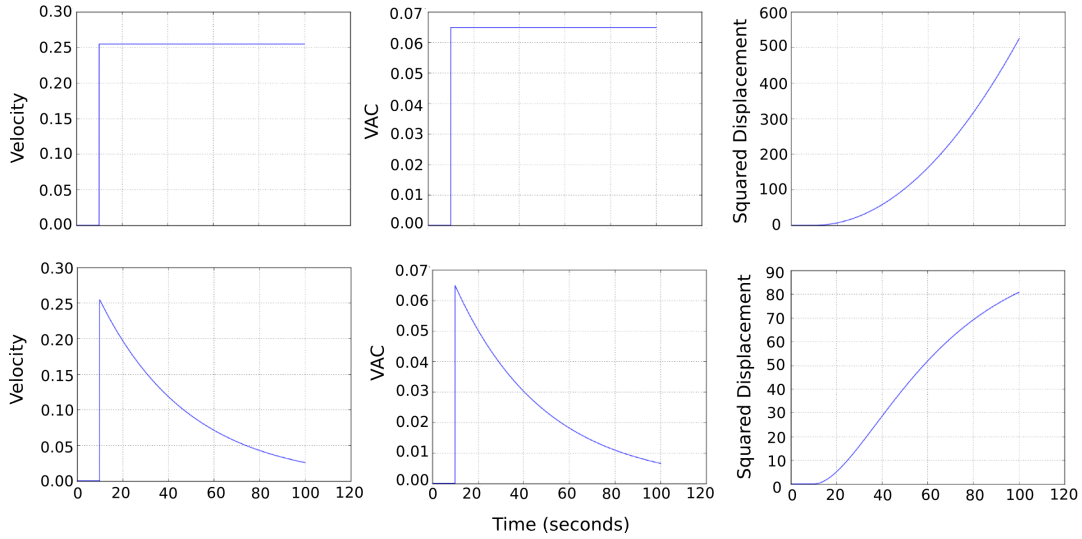


Figure 4.2: **Top row:** Test 1 without friction. An instantaneous force is applied to a cell at t = 10. The velocity of the cell increases and stays constant at 10 seconds, as does the velocity autocorrelation function. The mean-squared displacement increases quadratically due to this constant velocity away from its point of origin. **Bottom row:** Test 1 with friction. The velocity and vac decrease over time due to friction. This slowing in velocity can be seen in the mean-squared displacement no longer being quadratic.

increase in velocity, which remains constant as there are no other forces acting on cell.

The test is run again with the additional of friction force component (viscous drag force), where the frictional coefficient $K_F = 0.01$. This can be observed as a decay in the velocity of cell.

Test 2 - Constant force

With the same set up as for Test 1, we set $K_F = 0.0$ so that there is no friction in the system and the force $F = 0.01$. The artificial force is also applied constantly to the cellular agent (rather than an instantaneous force as in Test 1). A schematic of Test 2 can be seen in Figure 4.3. As seen in Figure 4.4 this causes an acceleration of the cell as anticipated.

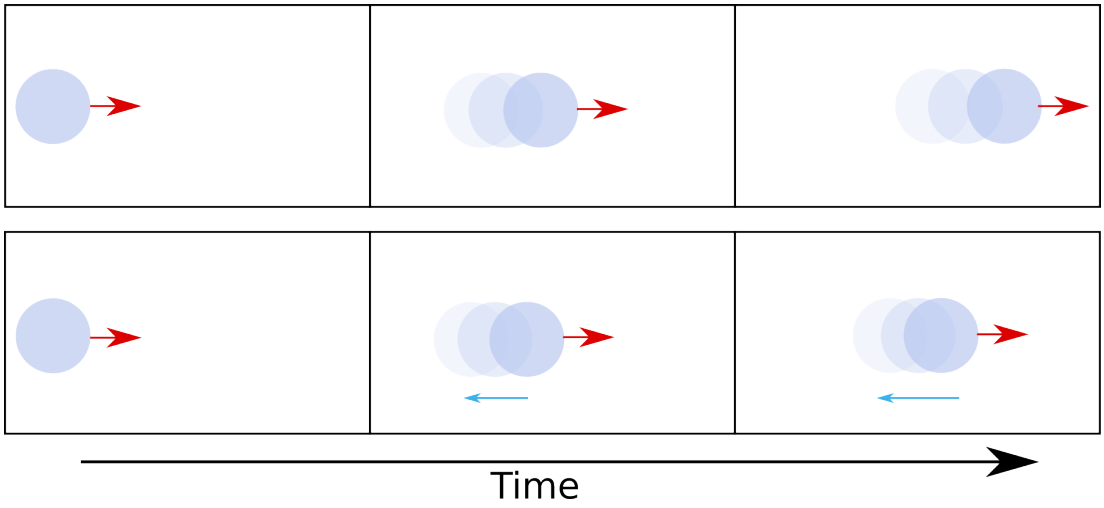


Figure 4.3: Schematic showing Test 2 - An constant force is applied to a cell causing it to accelerate. **Top row:** The cell does not experience any other forces and moves unimpeded. **Bottom row:** The cell experiences a frictional force (modelled as a drag force) as it moves through the space.

4.2.1.2 Collisions

Test 3 - Boundary Collisions

Agents (such as cells) collide with solid boundaries, which is modelled as a reflection of the velocity. This calculation has an elasticity coefficient E_C associated with it, where $E_C = 1.0$ is a completely elastic collision, and $E_C = 0.0$ is completely inelastic. A schematic of Test 3 can be seen in Figure 4.5.

The first test is test up with the same conditions as Test 1, with an instantaneous force resulting in a constant velocity (no frictional force component.) The cell is initial near to a solid domain boundary, and collides with it. As seen in Figure 4.6 the velocity remains constant, and upon collision the VAC value is inverted, as is the squared displacement gradient.

The elasticity of the boundary collision is set to be $E_C = 0.5$, such that the cell loses half of its velocity (kinetic energy) upon collision. The velocity is halved at the moment of collision with the boundary, and the VAC value and the squared displacement gradient half and invert.

A frictional component is added to the model to ensure it behaves as expected in this circumstance. A friction force coefficient of $K_F = 0.001$ is set. This causes

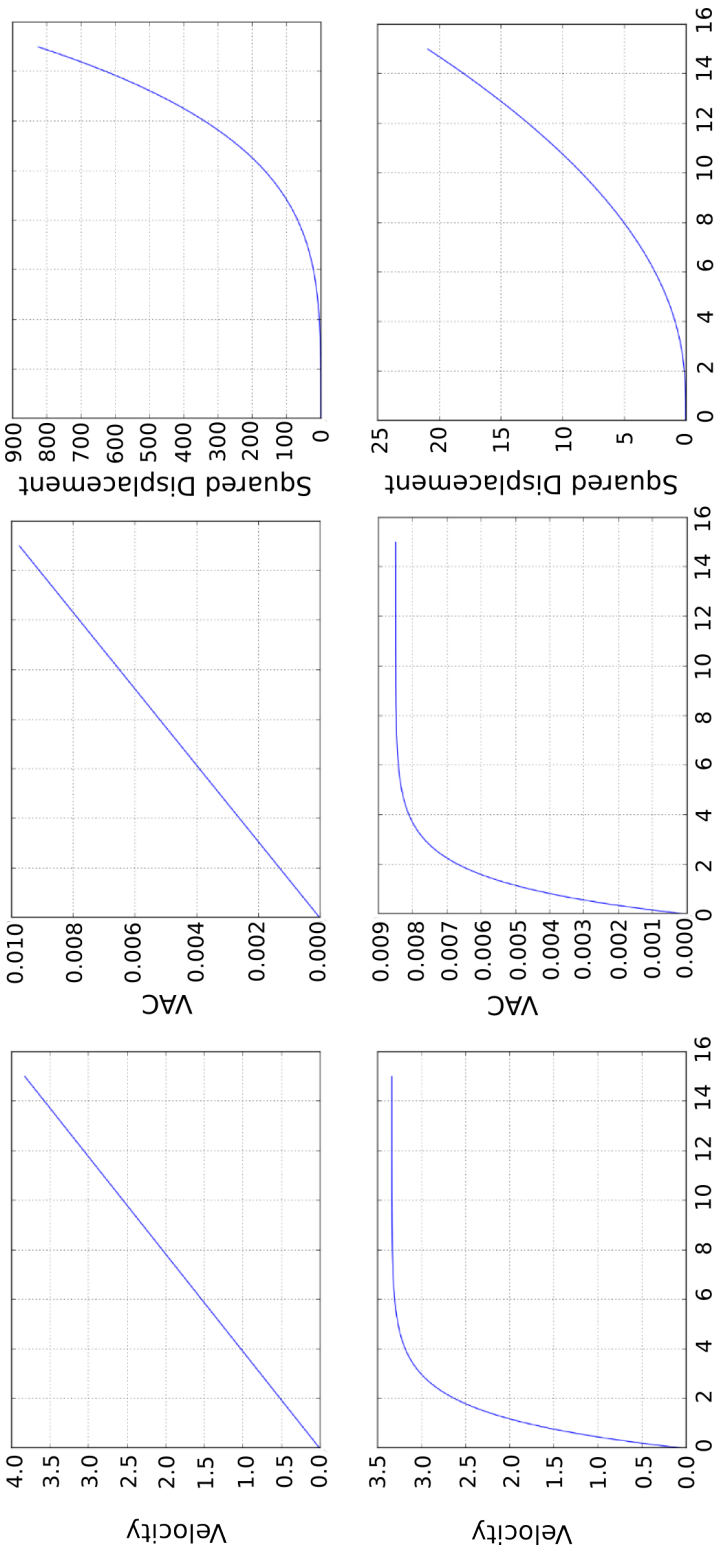


Figure 4.4: **Top row:** Test 2 without friction. As a constant force is applied to the cell its velocity increases linearly, as does its velocity autocorrelation. Its squared displacement is to the fourth order, as its displacement is increasing quadratically. **Bottom row:** Test 2 with friction. The frictional force is causes a decay in velocity, resulting in a terminal velocity for the applied force. This lower velocity results in the cell travelling a shorter distance. The terminal velocity means the squared displacement decays from scaling to the fourth order back to being quadratic.

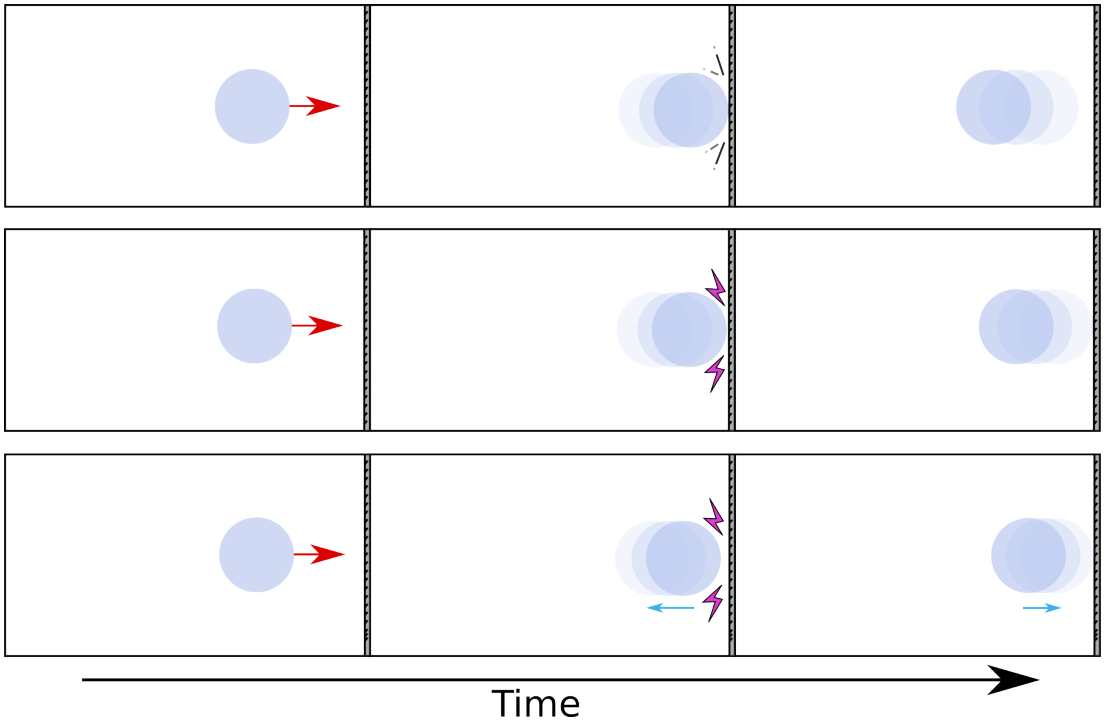


Figure 4.5: Schematic showing Test 3, where a force is applied to a spherical agent (a cell) causing it the collide with a domain boundary. **Top row:** The collision with the boundary is completely elastic. **Middle row:** The collision with the boundary is inelastic, and some energy is lost upon collision. **Bottom row:** As well as an inelastic collision, the cell experiences a drag force.

a decrease in velocity of the cell over time, increasing the duration it takes to hit the solid boundary.

Test 4 - Agent collisions

To test the collisions between agents, two cells are created. An instantaneous force is then applied to one cell, firing it directly at the other. A schematic of Test 4 can be seen in Figure 4.7. As seen in Figure 4.8, the kinetic energy is transferred completely elastically from the first to the second cell upon collision. The squared displacement shows that the first cell travels forward and collides with the second cell, at which point is stops completely, as desired.

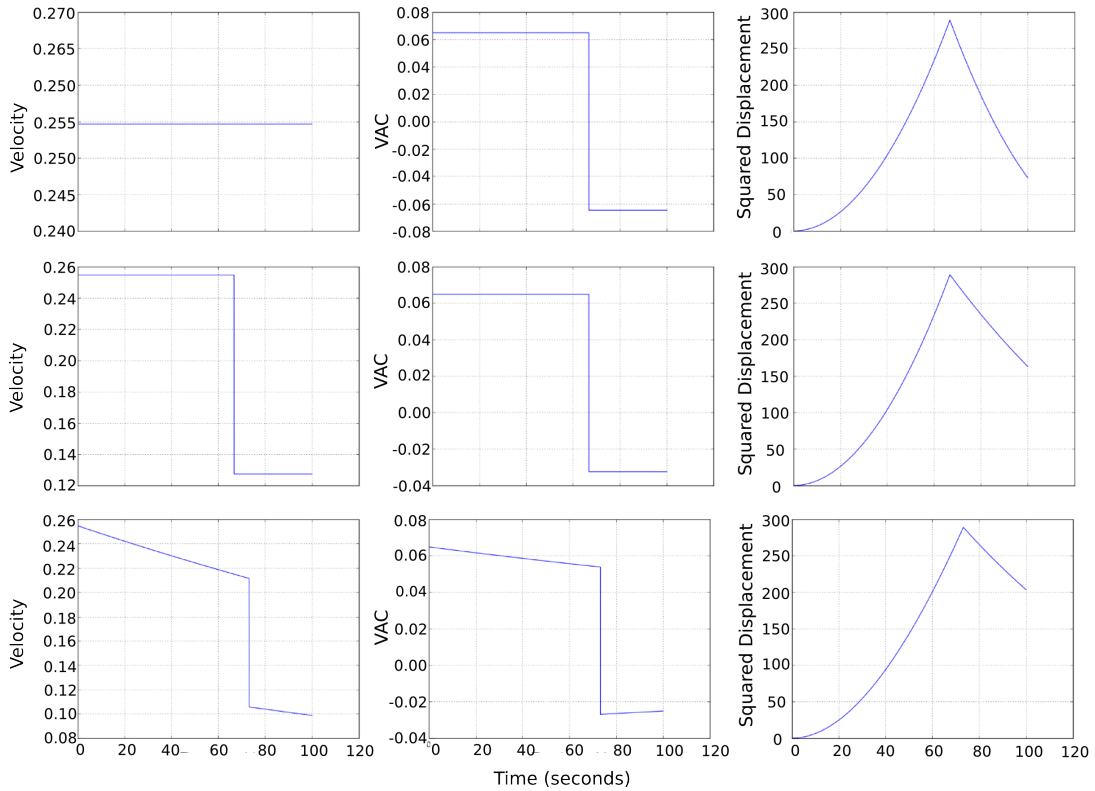


Figure 4.6: **Top row:** Test 3 without friction and dampening. When the cell collides with the boundary it rebounds in the reflected direction. It does not lose any velocity, however as it changes direction is VAC flips to be negative. Its squared displacement begins to decrease at the same rate as it increased as the rebounded cell returns to its original position. **Middle row:** Test 3 with dampening but no friction. The dampening introduced to the system causes the cell to lose some energy upon colliding with the boundary, this can be seen by its lower velocity at 65 seconds when it collides. Its VAC still flips as it changes direction, but loses some magnitude, as does the gradient of its squared displacement. **Bottom row:** Test 3 with both friction and dampening. With both friction and dampening in the system, it can be seen that the cell loses energy during its trajectory through the medium, not only upon collision.

4.2.1.3 Random walk

Test 5 - Random walker cell

Testing the translational diffusion force involved a similar setup to Test 1 (a single cell suspended in 3D domain.) The cell now experiences two types of force. The first is a 'kick' force in a random direction, where the force magnitude is sampled

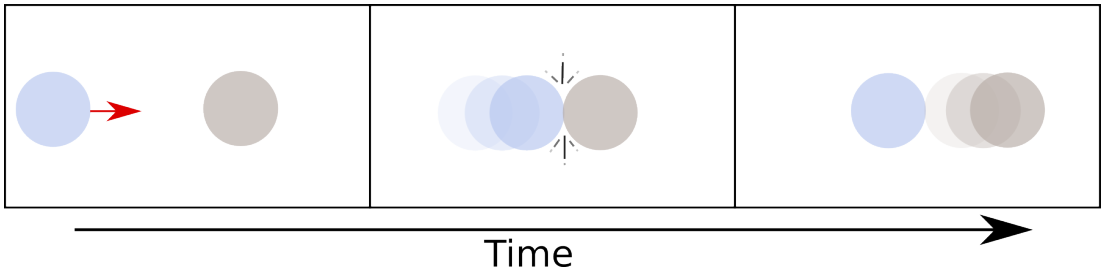


Figure 4.7: Schematic showing Test 4 - A force is applied to a spherical agent (cell) to induce a collision with a second static spherical agent.

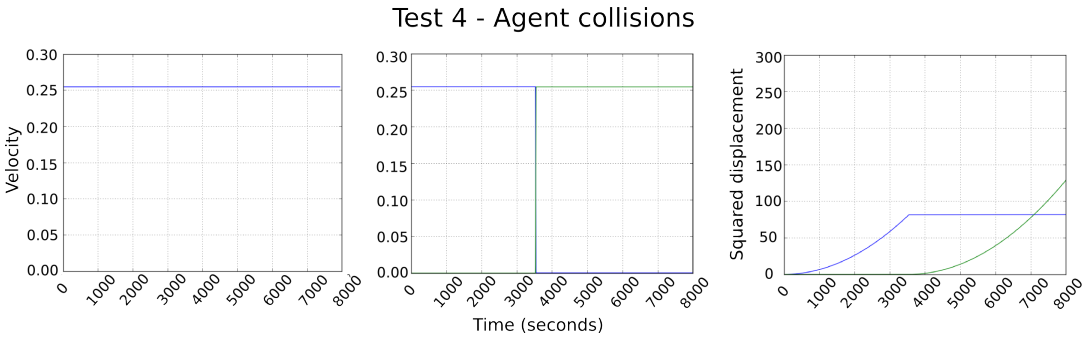


Figure 4.8: Test 4 Agent collisions. Two cells *C1* and *C2* are placed in the domain which experience no force. *C1* is then accelerated directly at *C2*. The two cells collide at T = 30s. Left: The total velocity remains constant in the system as seen in on the left. Middle: There is a complete transfer of momentum from *C1* to *C2*, with the velocity of *C1* reducing to 0.0 and *C2* achieving the same velocity as *C1* initially had (0.26). Right: The squared displacement of *C1* increases then plateaus upon collision, at which point *C2* starts to move away from its original position.

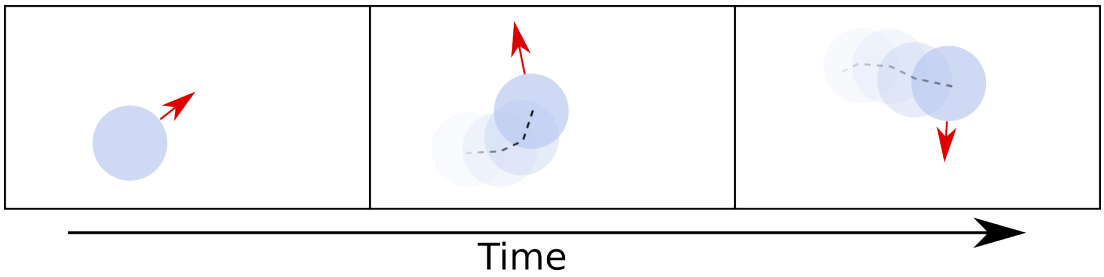


Figure 4.9: Schematic showing Test 5 - a cell undergoes a random walk through the simulation domain.

from a normal distribution around an average force. The second is a frictional force to ensure that there is a maximum velocity the cell can reach. A schematic of Test 5 can be seen in Figure 4.9.

As seen in Figure 4.10, we observe the that the velocity is changing randomly oscillating around an average of $3.0\mu ms^{-1}$. The squared displacement is also randomly changing, however as the domain boundaries are solid walls, its squared displacement can not go above the value determined by the long diagonal of the simulation squared. In this instance the world domain is $100 * 100 * 100\mu m$

The boundaries are set to be cyclical such that the cell can travel an unimpeded distance. One can see from the squared displacement in Figure 4.10 that it is increasing linearly, however due to it being a single cell this general trend is not so clear amongst the fluctuations. The diffusion coefficient of the bacteria can be determined from the mean squared displacement gradient, this calculation will be shown for the average squared displacement of a larger population of cells.

Test 6 - Population of random walker cells

A population of 1000 cells are created in a cubic domain of length $100\mu m$. Cyclical boundaries are defined such that cell positions are not confined within the $100\mu m$ domain. Cells are initial in an evenly distributed spatial arrangement. Each cell experiences an independent translational diffusion force, as seen in Test 5, causing it to do a random walk through the simulation domain. Cells also experience drag force. The parameters are $K_R = 1.0$ and $K_F = 1.0$.

In the first test cells do not collide and can pass through each other, as seen in Figure 4.11 (a). The second test includes cell collisions, shown in Figure 4.11 (b). The third test includes cell collisions and the formation of a spring between colliding cells, shown in Figure 4.11 (c). These three tests were run to test that the integrator remains stable even when new forces are introduced into the system.

As seen in Figure 4.12, the average velocity of cells remains constant, just above $1.0\mu ms^{-1}$. The average VAC oscillates around 0.0, showing that the cells are constantly changing direction. The average mean squared displacement (MSD) of the population of cells shows a linear increase, as they drift away from their initial positions. The integrator can be seen to remain stable for all of the test variations A, B and C.

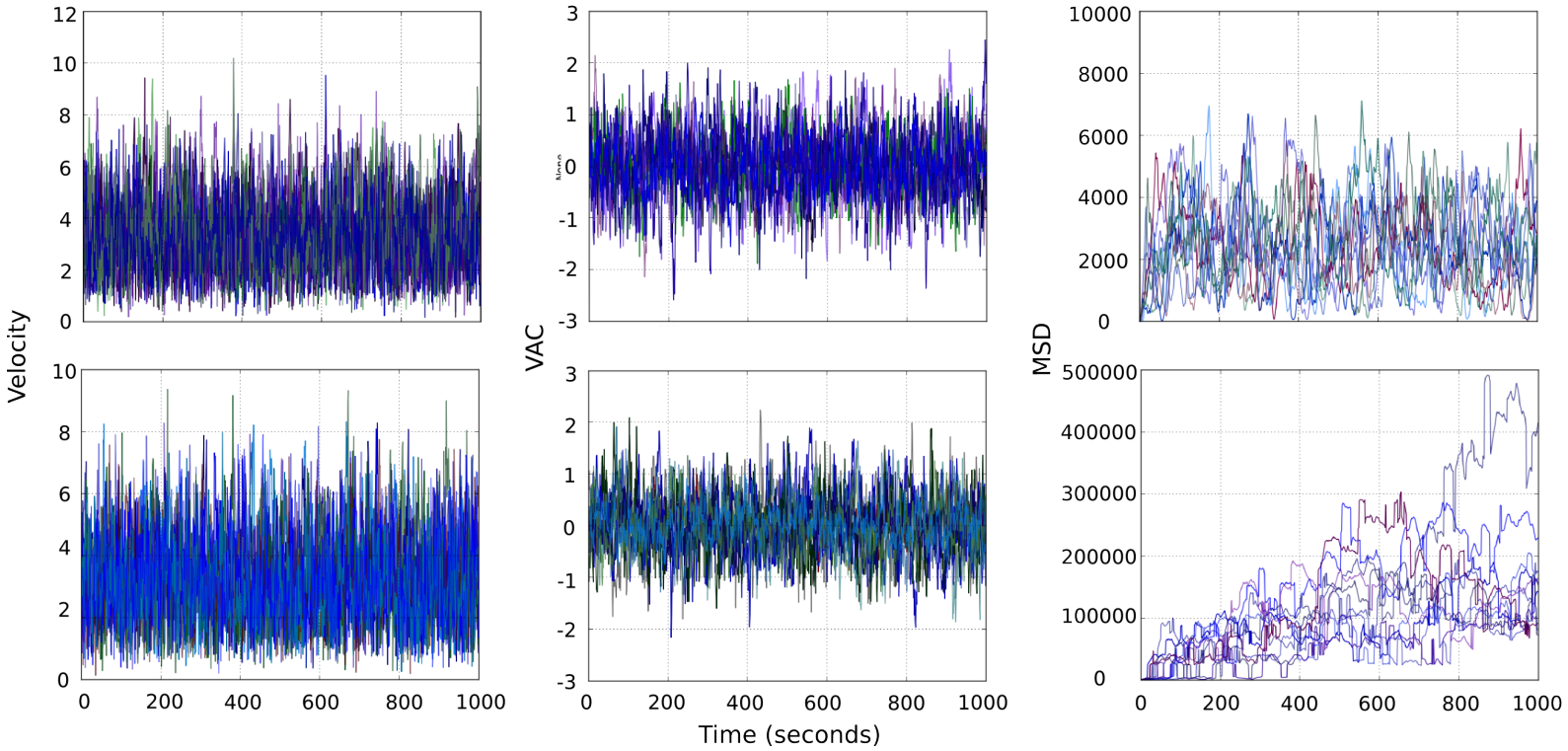


Figure 4.10: **Top row:** Test 5 with solid domain boundaries. A single cell does a random walking through the domain, as the boundaries are solid it can not travel outside the 100*100*100 μm domain. Due to this its squared displacement remains under 10000 (as maximum displacement from original position is 100, thus its squared displacement can be no more than 100^2.) Its velocity fluctuates around a mean value of $4.0mums^{-1}$, and its VAC fluctuates around 0. **Bottom row:** Test 5 with cyclic domain boundaries. The single cell can leave one side of the domain and re-enters on the opposite side, and we consider its position as if it had travelled outside of the domain. Due to this its squared displacement is unbound, as can be seen by its increase. The velocity and VAC remain stable. The graphs show 10 repeats of the test.

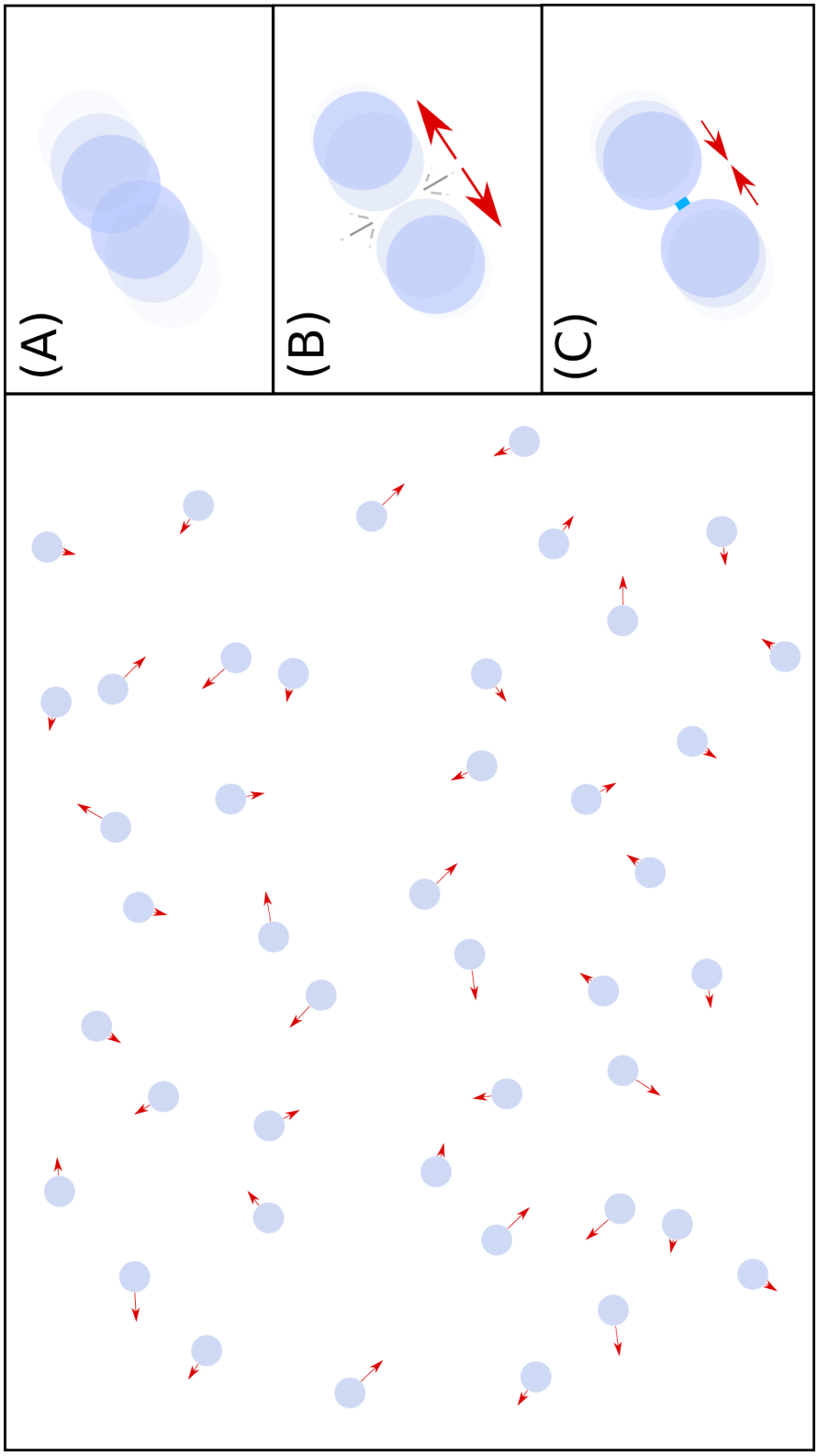


Figure 4.11: Schematic showing Test 6 - A population of random walker cells are free to roam around a domain with cyclical boundaries. **a)** In the first test cells do not collide and simply pass through each other. **b)** In the second test cells collide with each other. **c)** In the third test cells which collide also adhere together, modelled as a loose and breakable spring between the cells.

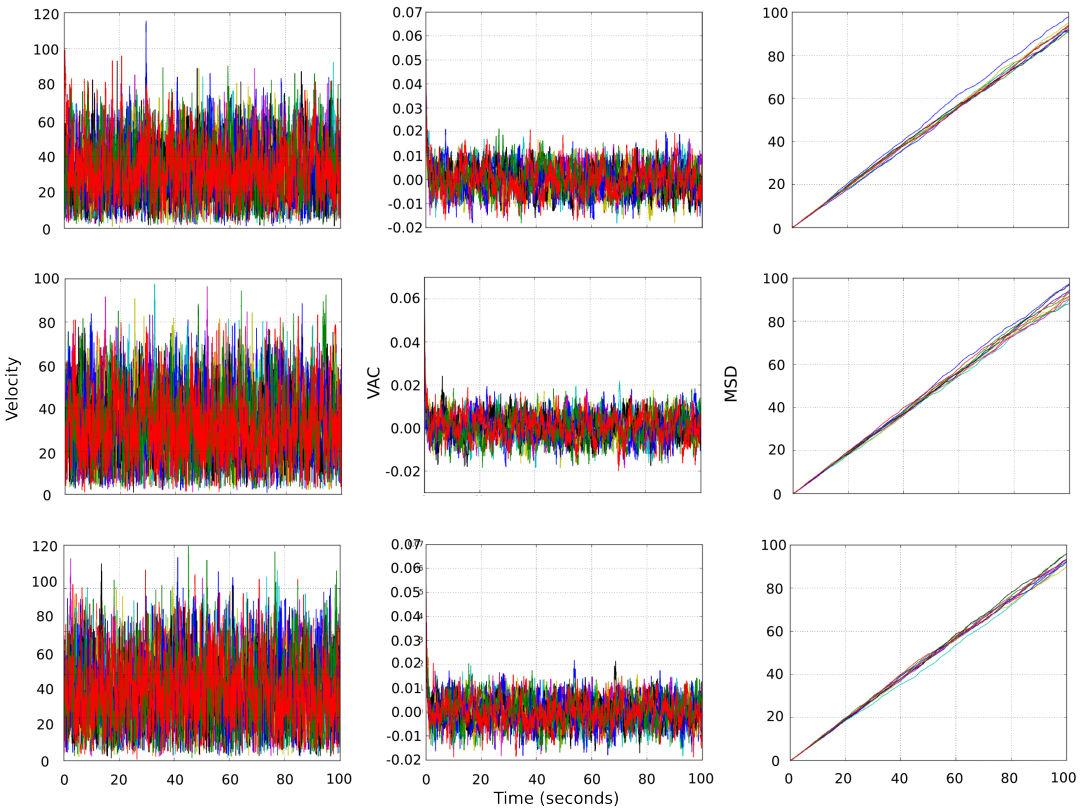


Figure 4.12: Test 6 results - Population of random walkers. **Top row:** results for system A (where cells don't collide), showing stable total velocity and VAC, with a linearly increasing MSD. Middle row: results for system B (where cells do collide), again showing stable total velocity, VAC and MSD. **Bottom row:** results for system C (where cells which collide form a loose spring connecting them), showing stable total velocity, VAC and MSD trajectories. The graphs show 10 repeats of the test.

4.2.2 Chemical integrator

To ensure that the numerical approximation of diffusing chemicals are correct we perform a series of tests. First we verify that the finite volume method implemented according to Fick's laws is working correctly, simulating how chemicals diffuse through the extracellular space, in relation to the analytical solution. We then establish the stable parameter regions of the chemical diffusion integrator. Finally we verify that the membrane transport mechanisms which carry chemicals between the extracellular space and intracellular compartments are correct.

4.2.2.1 Extracellular diffusion

A cuboidal simulation domain is created of dimensions 100x100x100 μm. We define a chemical S1 which can diffuse through space with a diffusion coefficient $D = 10^{-5} ms^{-2}$. S1 has a degradation coefficient of $K_d = 0.0$ so it does not degrade over time. We pipette 100,000 molecules of S1 into the system center of the simulation domain, which was rasterised with a grid depth of $G_d = 7$, and measure the spatial concentration profile over time. A schematic showing this test can be seen in Figure 4.13.

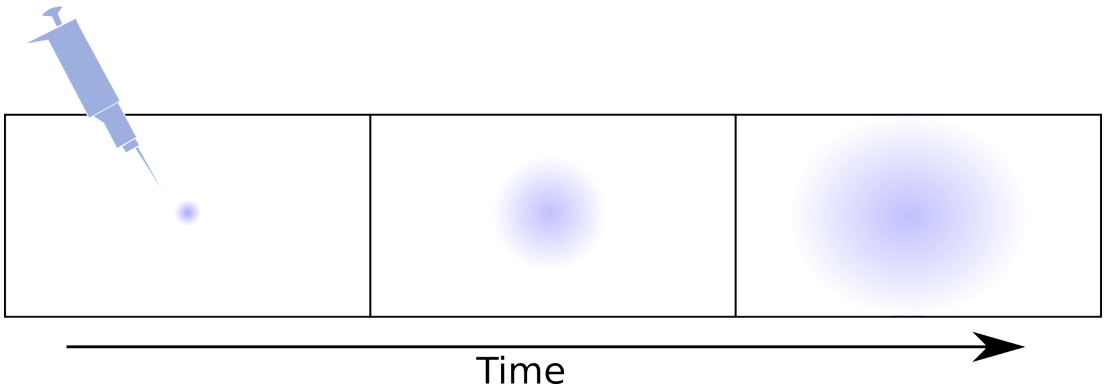


Figure 4.13: Schematic showing Test 7 - Chemical S1 is pipetted into the center of the 3D domain and left to diffuse. The concentration for different distances from the original pipette position are recorded.

We expect that as the chemical profile should show a peak in the center, which over time forms a Gaussian distribution and eventually flattens out as it reaches an equilibrium. The overall amount of chemical S1 should be the same

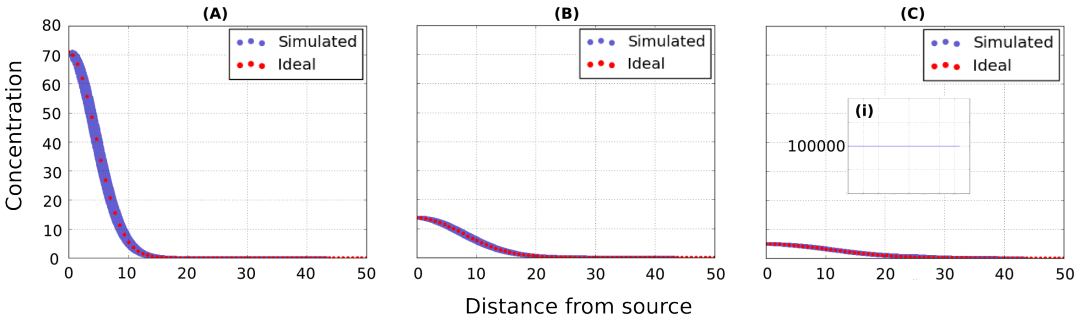


Figure 4.14: Results for the extracellular diffusion test, showing simulated results (blue) and analytical solution (red) at (A) 10 seconds, (B) 30 seconds, and (C) 60 seconds.

as we pipetted, as it does not degrade. This can be seen in Figure 4.14, with the analytical solution overlaid. The analytical solution was found using the diffusion equation:

$$c(x, y, z, t) = \frac{M}{(\sqrt{4\pi Dt})^3} exp(-\frac{x^2 + y^2 + z^2}{4Dt}). \tag{4.1}$$

To find the stable parameter regions of the diffusion integrator, we run this same test with different input parameters, and observe if the integrator is stable by checking whether fluctuations in the quantity of the molecules varies more than 0.1 percent. The sensitive parameters are the simulation *time step* **t**, the diffusing chemical's *diffusion coefficient* **D**, and the *grid depth* **G**. The heatmaps in Figure 4.15 show the stable regions for common values of these parameters.

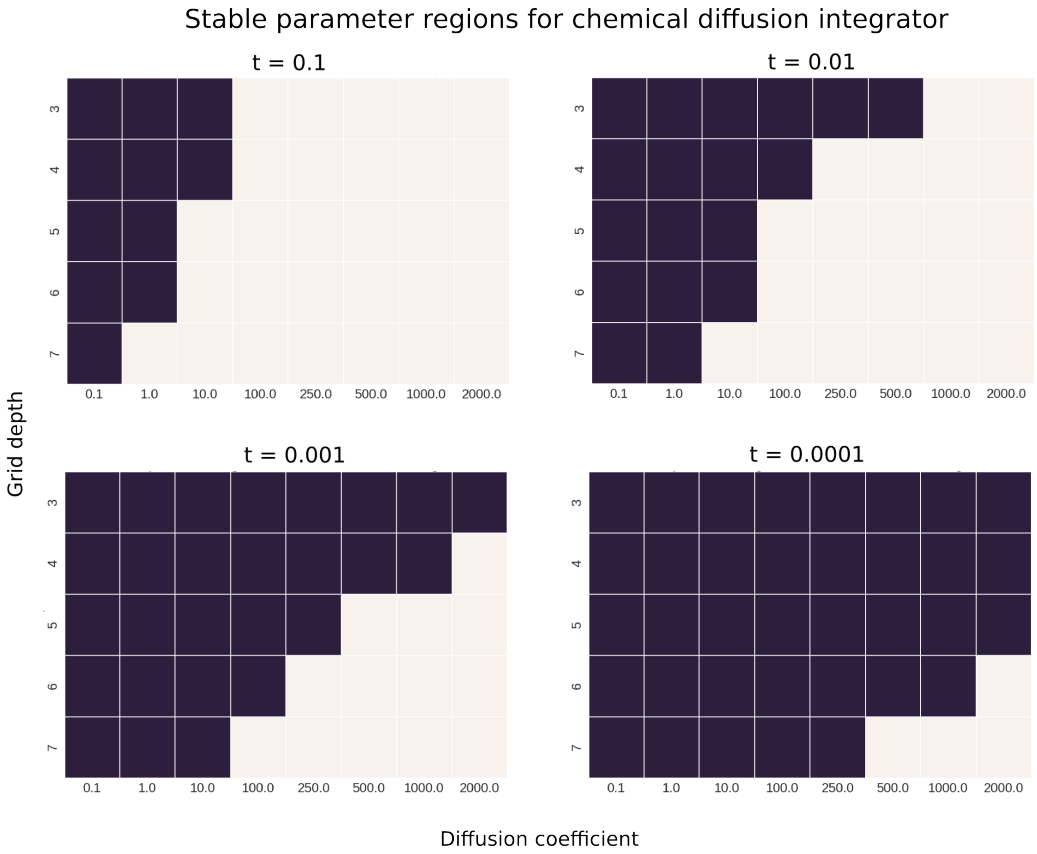


Figure 4.15: Heatmaps showing stable parameter regions of the extracellular diffusion integrator. Black indicates the integrator is stable for those parameters, and light pink indicates it's unstable. A smaller simulation *time step* **t** is needed for stable integration of chemicals with high *diffusion coefficients*. As the *grid depth* (which is the resolution of the grid as described in Chapter 3) increases, then the *time step* may need to be decreased to stability integrate the same diffusion coefficient.

4.2.2.2 Membrane diffusion

To ensure that membrane transport mechanisms are correctly implemented, we devise a small system and check that the expected results of different types of transport are yielded. We create a cell in a small 3D world, the extracellular space is fill with a chemical *S1*.

There are two types of membrane transport. Passive transport represents osmosis, where chemicals can move across the membrane from high to low concentrations. Active transport represents the proteins which can move chemicals across the membrane against the gradient. A schematic can be seen in Figure 4.16.

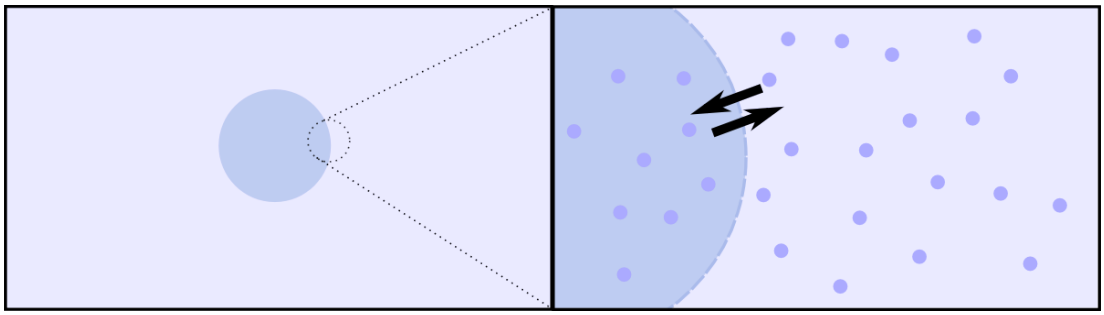


Figure 4.16: Schematic showing Test 8 - molecules of a chemical S1 fill the extracellular space, and can be transported across the cell membrane via either passive mechanisms (osmosis) or by active mechanisms (transporter protein).

Additionally we test the two numerical approximation methods for calculating the flux across the membrane. The first is the continous implementation, where the flux is deterministically calculated. The second is a discrete implementation, utilising a Poisson sampling to obtain a flux rate from a distribution around the mean rate.

Active transport

The active transport mechanism uptakes the chemical S1 from the extracellular space into the intracellular compartment as seen in Figure 4.17. The number of S1 molecules in the extracellular space decreases, and increases in the intracellular compartment. The intracellular concentration becomes greater than the extracellular concentration, as the active transport can move molecules against

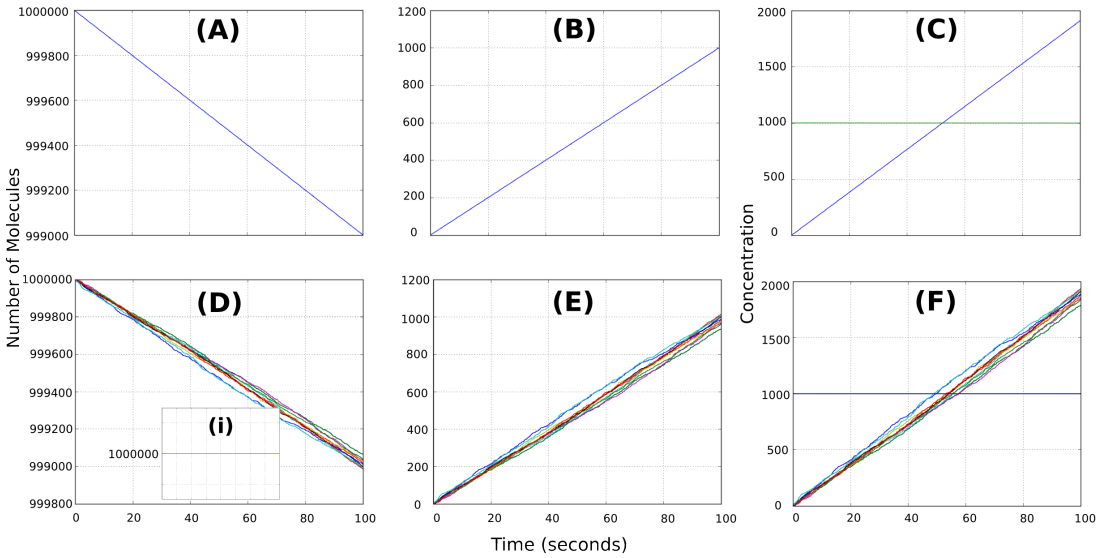


Figure 4.17: Results for active membrane transport mechanism, showing molecules being moved from the extacellular to intracellular compartment even against the concentration gradient. **Top row:** shows the continuous method. (A) the extracellular number of molecules. (B) the intracellular number of molecules. (C). The intracellular (blue line) and extracellular (green line) concentrations. **Bottom row:** shows the discrete method. (D, E, and F) correspond to the same as (A, B and C). There are 10 repeats of the discrete method.

the gradient. The total amount of chemical S1 in the entire system remains constant (Figure 4.17 (i)). This correct behaviour is seen for both continuous and discrete methods.

Passive transport

The passive transport mechanism uptakes the S1 from the extracellular space into the intracellular space until the concentrations are equal, as seen in Figure 4.18. The total amount of chemical S1 in the entire system remains constant. This correct behaviour is seen for both discrete and continous method implementations.

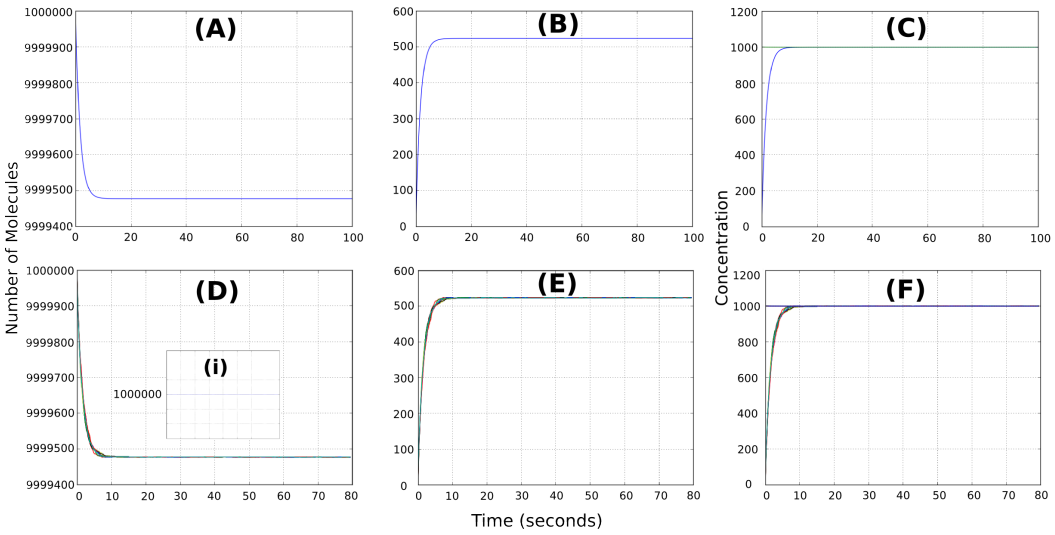


Figure 4.18: Results for passive membrane transport mechanism, showing that transport occurs until there's no concentration difference between the intracellular and extracellular space. **Top row:** shows the continuous method. (A) the extracellular number of molecules. (B) the intracellular number of molecules. (C). The intracellular (blue line) and extracellular (green line) concentrations. **Bottom row:** shows the discrete method. (D, E, and F) correspond to the same as (A, B and C). There are 10 repeats of the discrete method.

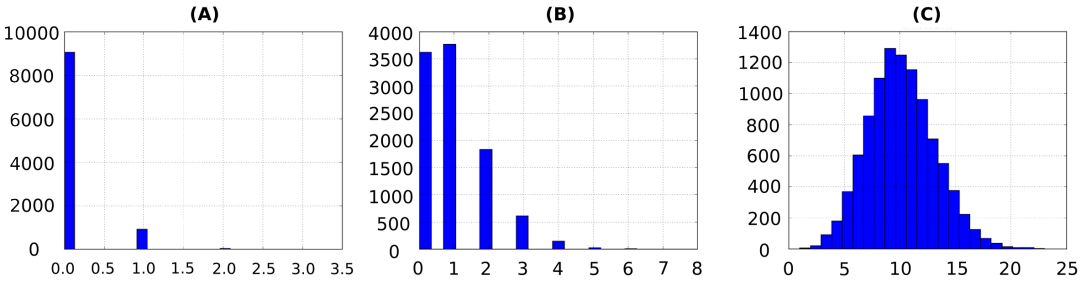


Figure 4.19: Results of the Poisson sampler for different lambda values. **(A)** 10,000 runs for $\lambda = 0.1$. **(B)** 10,000 runs for $\lambda = 1.0$. **(C)** 10,000 runs for $\lambda = 10.0$.

Poisson sampler

We ensure that the Poisson sampler is correctly implemented by running tests for different *lambda* values. The results as seen in Figure 4.19 show 10,000 iterations of the sampler for a given *lambda* value, showing the correct behaviour of

producing a normal distribution around that mean.

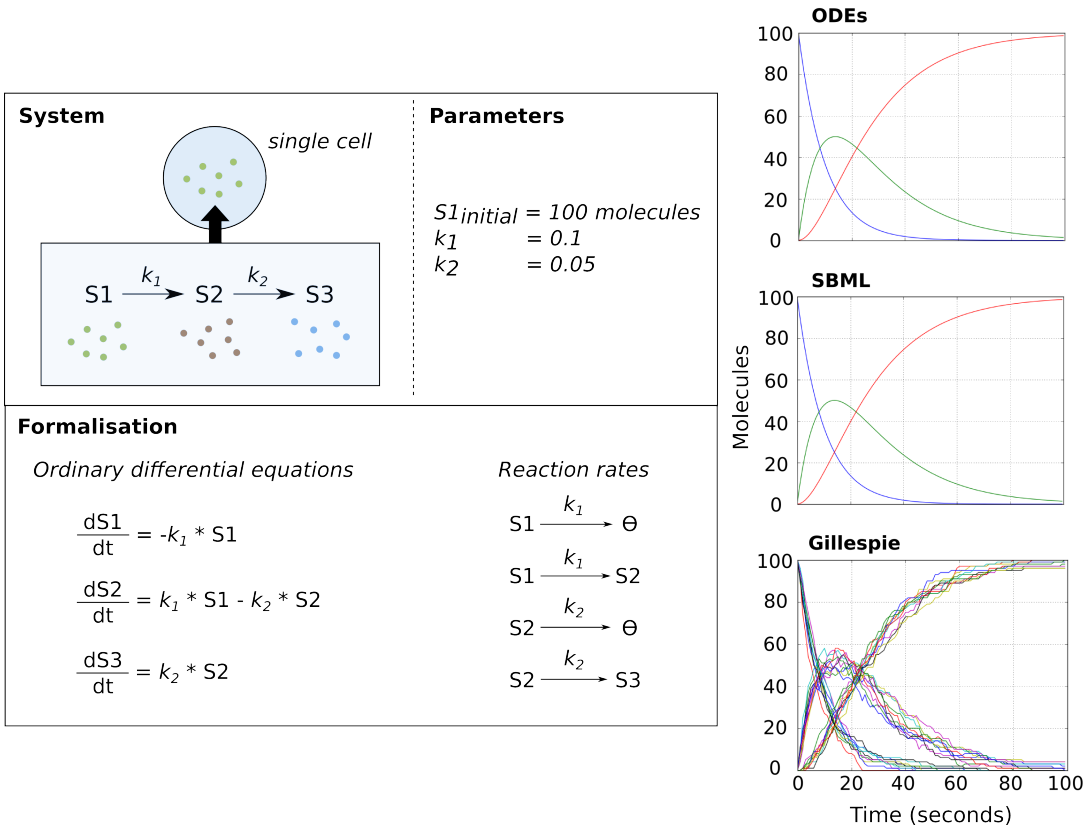


Figure 4.20: **Left:** Schematic showing the system for testing intracellular integrators. The system is a basic chemical reaction network that converts S1 into S2 at a rate K_1, and converts S2 into S3 at rate K_2. This network can be represented either as a set of differential equations, or as mass-action kinetics. **Right:** The results of the different intracellular integrators. The ODE and SBML methods are deterministic methods, where as the Gillespie method is stochastic. The graph shows 10 repeats for Gillespie method.

4.2.3 Intracellular dynamics integrator

To ensure the methods for modelling intracellular dynamics are correct we develop small models of cells with metabolisms and gene regulatory networks. We first check that a self contained cell with these dynamics behaves correctly, verifying that components work. We then check create ensuring these components work as expect in conjunction with other features, such as the cells growing and dividing

into child cells. Finally we validate that integrating these systems gives accurate results.

4.2.3.1 ODE, SBML and Gillespie submodels

We define a simple chemical reaction network which occurs within a cell, and ensure all methods yield correct results. A cell is defined with a basic *metabolism* which converts S1 into S2 at rate K_1, and then converts S2 into S3 at rate K_2. This system can be modelled either as ordinary differential equations (ODEs), an SBML model, or a Gillespie simulation. We test that each of these methods gives the same result for this system, this can be seen in Figure 4.20.

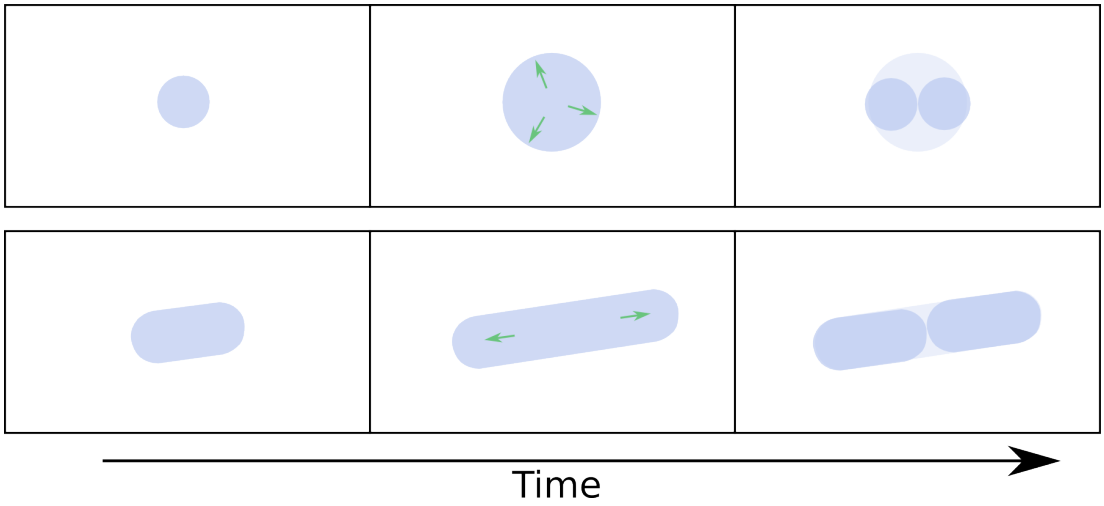


Figure 4.21: Schematic showing Test 9, illustrating the growth and division of a cocci cell (top) and a bacilli cell (bottom).

4.2.3.2 Cell Mitosis

To validate that the cell mitosis is correctly implemented, we develop a simple model of a single cell which grows and divides. We do this test for the two morphologies implemented in the Simbiotics library, coccus (spherical) and bacillus (rod-shaped). A schematic showing these two cell morphologies growing can be seen in Figure 4.21. The cells are set to grow at a constant rate, and divide upon reaching around twice their original size (see Chapter 3 for full details on how

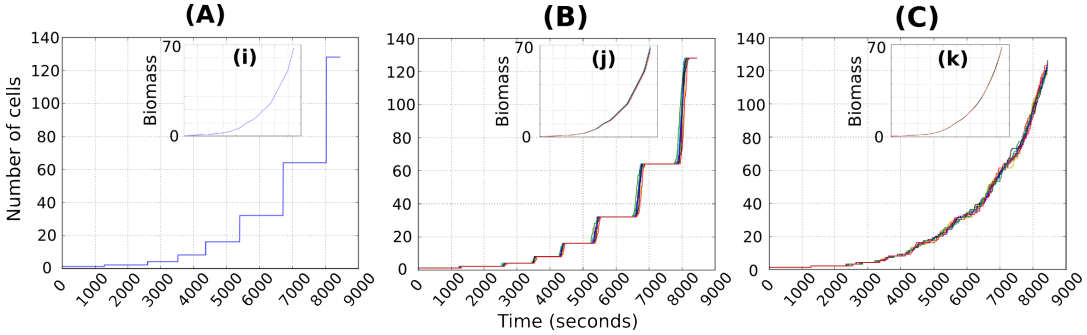


Figure 4.22: Schematic showing results for Test 9. Tests for both cocci and bacilli cells were conducted, where the parameter for adding noise to the growth rate, G_v, and parameter to add a random offset to the division ratio between children during mitosis, D_r, were varied. **Bottom:** The results of the tests stacked on top of each other, showing 5 tests with cocci and 5 test with bacilli. **(A)** $G_v = 0.0$ and $D_r = 0.0$. As can be seen the growth rate of cells is consistent and synchronised, causing a doubling of the colony size roughly ever 1000 seconds. **(B)** $G_v = 0.1$ and $D_r = 0.0$. The growth rate of cells becomes slightly desynchronised. **(C)** $G_v = 0.1$ and $D_r = 0.1$. Adding fluctuations to how mass is dividing between child cells during mitosis results in a complete desynchronisation of mitosis events across the population. The insets *(i)* showing the total biomass in the system.

mitosis is implemented). The number of cells and the total biomass is measured over time.

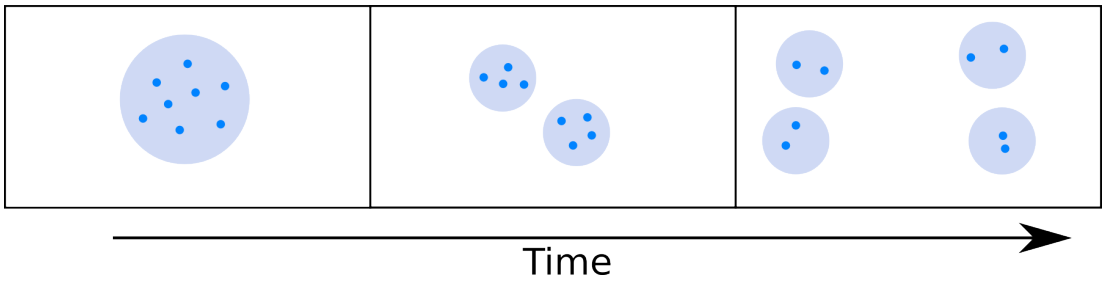


Figure 4.23: A single cell with an initial chemical quantity grows and divides. The chemical is not consume or degraded, and is assumed to be well mixed upon mitosis.

As expected the generation time (time at which population size doubles) remains constant, and goes up in steps as all the cells divide simultaneously, as seen in Figure 4.22 **A**. To add noise into the system so that cells do not grow and divide synchronously, we first add random fluctuations to the growth rate of

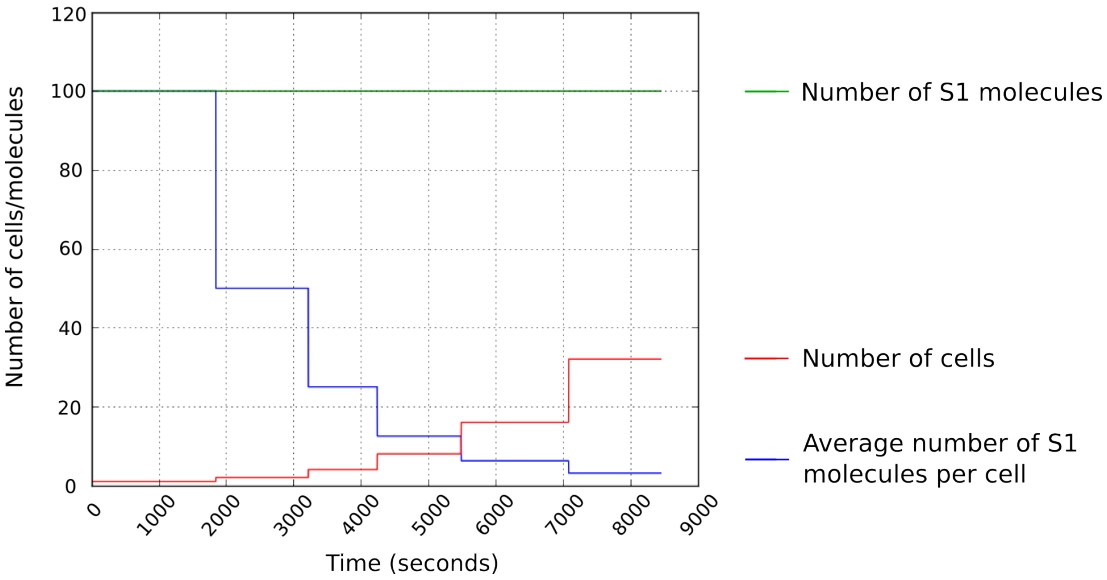


Figure 4.24: As the cell population grows, it can be seen that the total amount of S1 in the system is preserved.

cells. As seen in **B** the step function has some curve to it, as not all cells are dividing at exactly the same time. To further desynchronise the mitosis event of cells, we define a random fluctuation in the mass attributed to the two daughter cells upon mitosis. As seen in **C**, the number of cells gradually increases rather than forming a step function.

A final validation test is performed to ensure that intracellular chemical numbers are preserved during mitosis events. We create an equivalent model to the original mitosis verification test, and we seed the initial cell with 100 molecules of chemical S1 inside it. As the population grows, we record the total number of S1 molecules in the system.

4.3 Performance tests

In this section performance tests of Simbiotics are presented, including an overview of the large models built in case studies.

The bacterial coaggregation case study model (Chapter 7) involved simulating 100,000 cellular agents in a cubic simulation domain of length $400\mu m$. Simulating

4 hours of aggregation took 7 hours of computational time.

The biofilm case study model (Chapter 8) involved simulating an initial population of 3,000 cellular agents in a cuboidal domain of size 300x50x300μm. Simulating 12 hours of biofilm growth took 20 hours of computational time, at this time there were 750,000 cellular agents in the simulation domain.

Both case study models were simulated on a high performance computing cluster. The simulations were run distributed across 5 nodes in the cluster, with each node specification being Xeon E5-2690 v2 with 265GB of RAM.

Performance analysis shows that calculating the physics of the system such as cell-cell collisions, movement and interaction forces, consume the largest amount of computational time. The scaling of the physics calculations is linear $O(N)$ with the number of cells, however becomes exponential $O(N^2)$ when cells are packed together extremely closely such as when aggregates form or dense biofilms.

4.3.1 Stress tests

We preformed stress tests on the Simbiotics platform, analysing the performance scaling for different population and domain sizes. Tests were performed on two cores of a single node of the HPC.

4.3.2 Performance scaling

First we tested how performance scales for cellular populations of the same density for varied domain volumes. We performed the tests for 3 population densities, $\frac{1.25e^5 \text{cells}}{mL}$, $\frac{5e^5 \text{cells}}{mL}$ and $\frac{1e^6 \text{cells}}{mL}$. For each density we scale the population and domain size, maintaining the same density and observing how performance scaled.

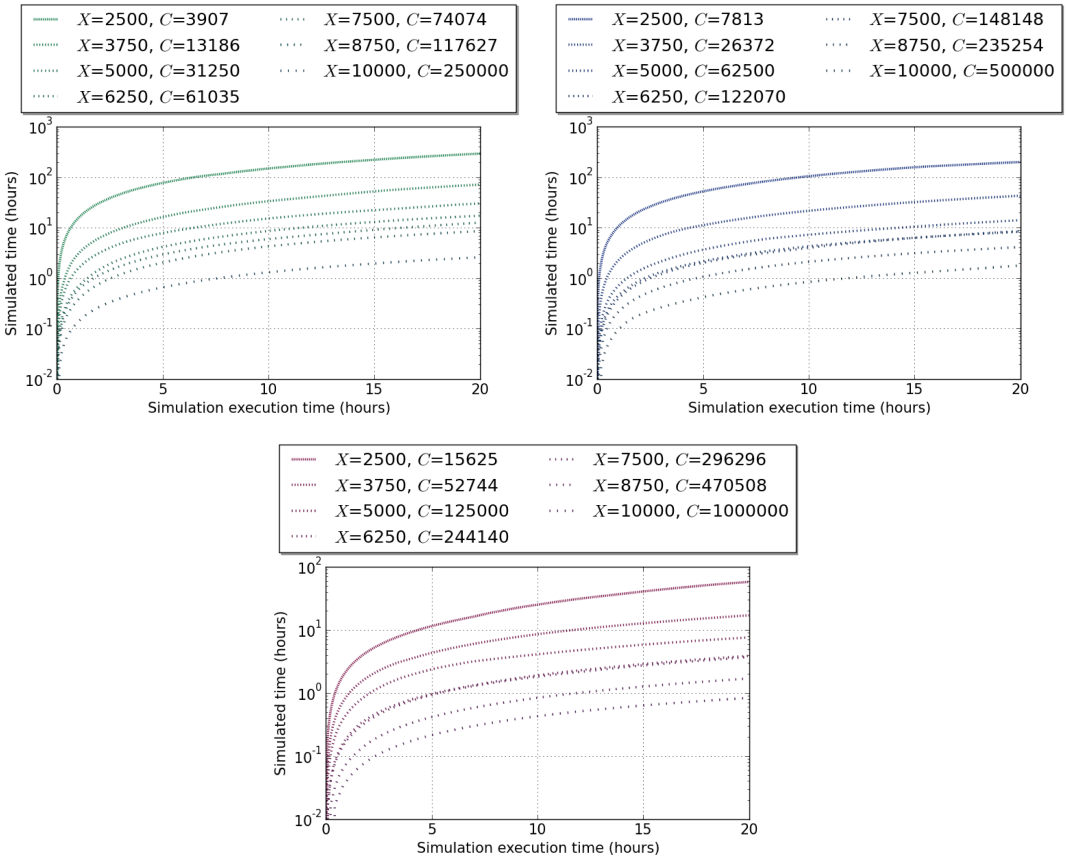


Figure 4.25: Stress test showing log of simulated time against the simulation execution time. Stress tests were performed with spherical cells, suspended in a fluid experiencing a random mixing force. For each test, N is the number of cells, and M is the length of one side of the cubic simulation domain in μm. (a) Results for a density of $\frac{1.25e^5 \text{cells}}{mL}$. (b) Results for a density of $\frac{5e^5 \text{cells}}{mL}$. (c) Results for a density of $\frac{1e^6 \text{cells}}{mL}$. All tests were performed on two cores of a single node of the HPC.

4.3.3 Sphere and Rod comparison

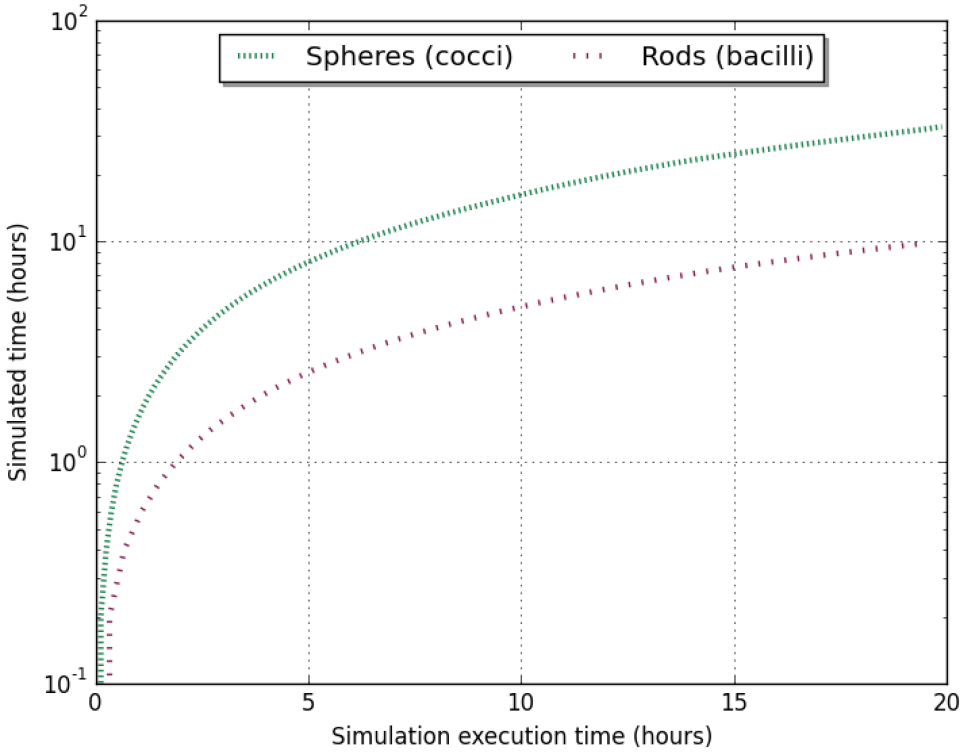


Figure 4.26: A stress test of sphere and rod-shaped cells, showing the log of simulated time against the simulation execution time (both in hours). 100,000 spherical and 100,000 rod-shaped cells were simulated independently, suspended in a fluid volume of 1mL. Cells experience a random mixing force to induce collisions. Tests were performed on two cores of a single node of the HPC.

4.4 Reproducing literature results

To further validate Simbiotics I conducted some brief studies reproducing literature results from some other related simulators. The motivation behind this was to confirm that Simbiotics could reproduce a wide range of existing simulator findings, both ensuring the correctness of Simbiotics and ensuring that a range of existing models can be represented in Simbiotics. This is especially pertinent in a discipline where lack of reproducibility has been a challenge [204].

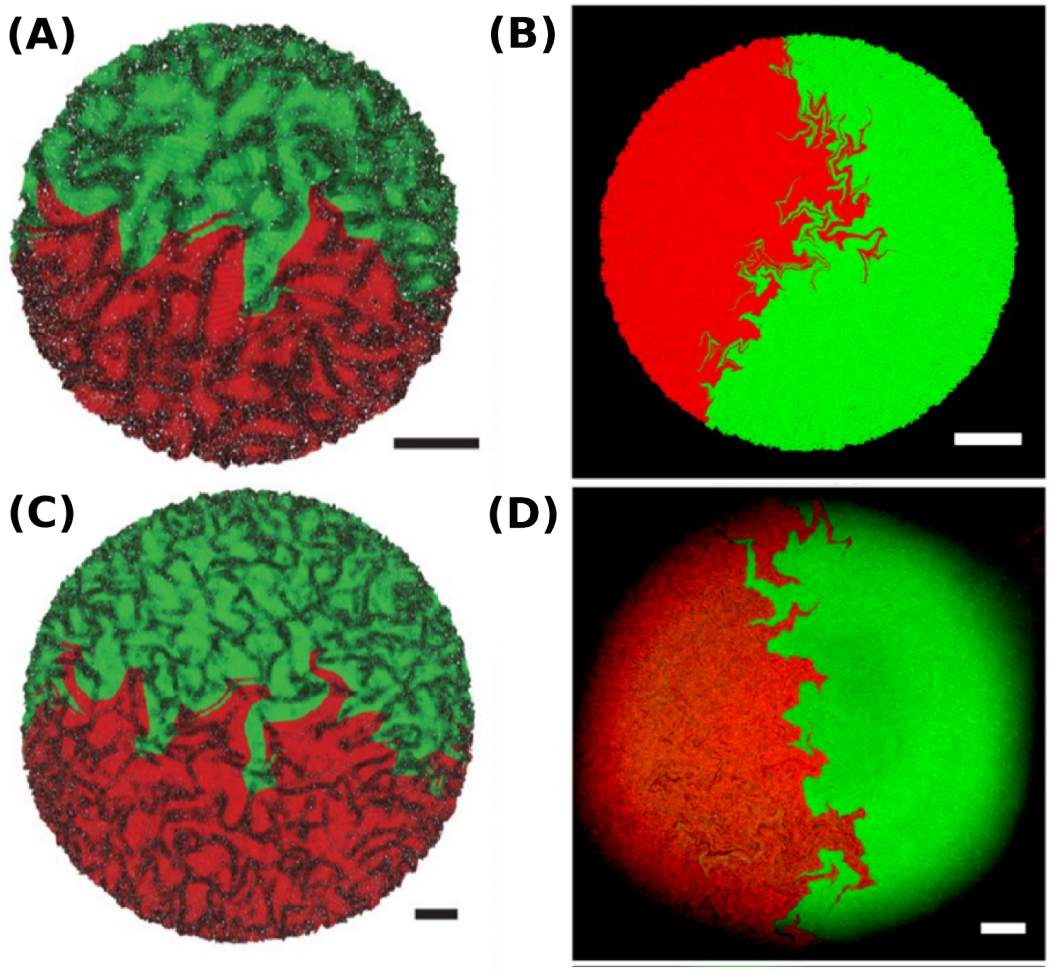


Figure 4.27: CellModeller4 results showing fractal patterns forming at colony boundaries. Figure adapted from [182]

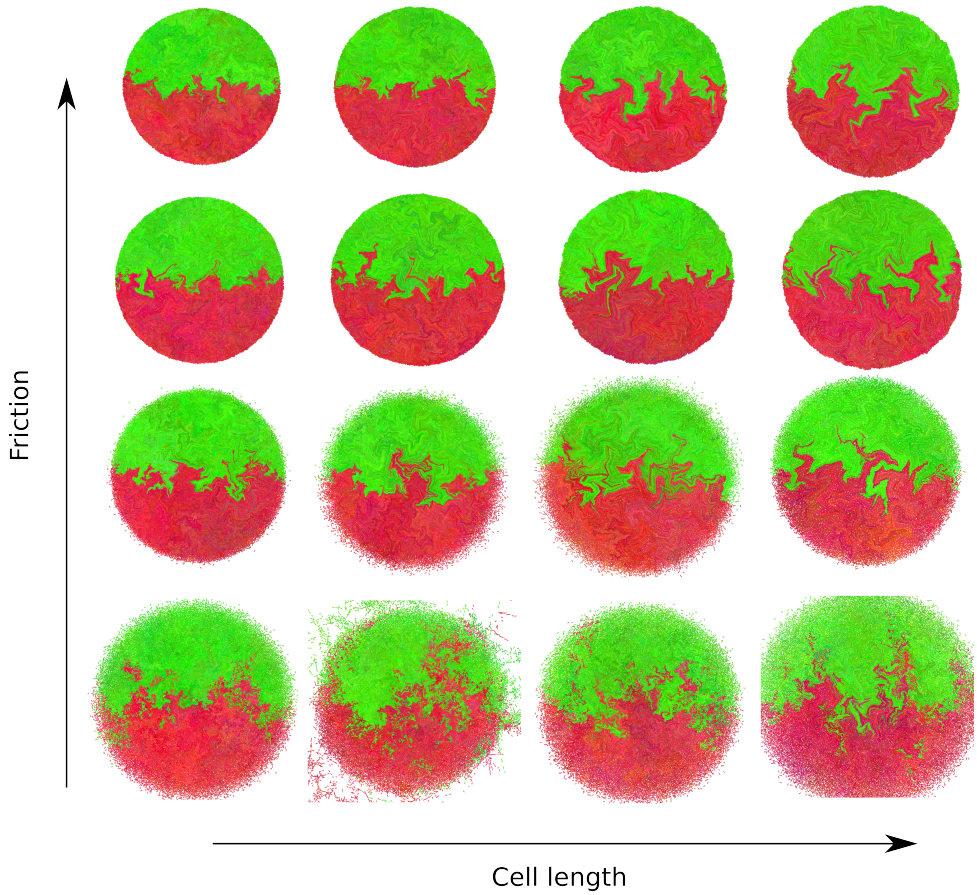


Figure 4.28: Simulation results generated by the Simbiotics model reproducing previous literature findings of fractal pattern formation at the boundary of growing bacillus cells [182]. The system demonstrates the same phenomena as reported in the original publication. The simulation was run with different *friction* and *cell length* parameters, studying the effect these system properties have on the fractal structures.

4.4.1 Emergence of fractal colony boundaries

The spontaneous emergence of fractal colony boundaries due to mechanical instability was demonstrated [182]. The model in the study was developed using *CellModeller4*, a multicellular modelling tool developed at Cambridge University. Their model was of a colony growing in 2D, simulating cell growth and division coupled with physical shoving interactions between cells. Their results can be seen in Figure 4.27.

I developed a model simulating the same system; two species of rod-shaped cells growing on a 2D surface. My simulation findings show the same fractal patterns forming at the boundary of colonies emerging purely from physical shoving forces. The model is simulated for varying friction coefficients and cell lengths, the results for which can be seen in Figure 4.28.

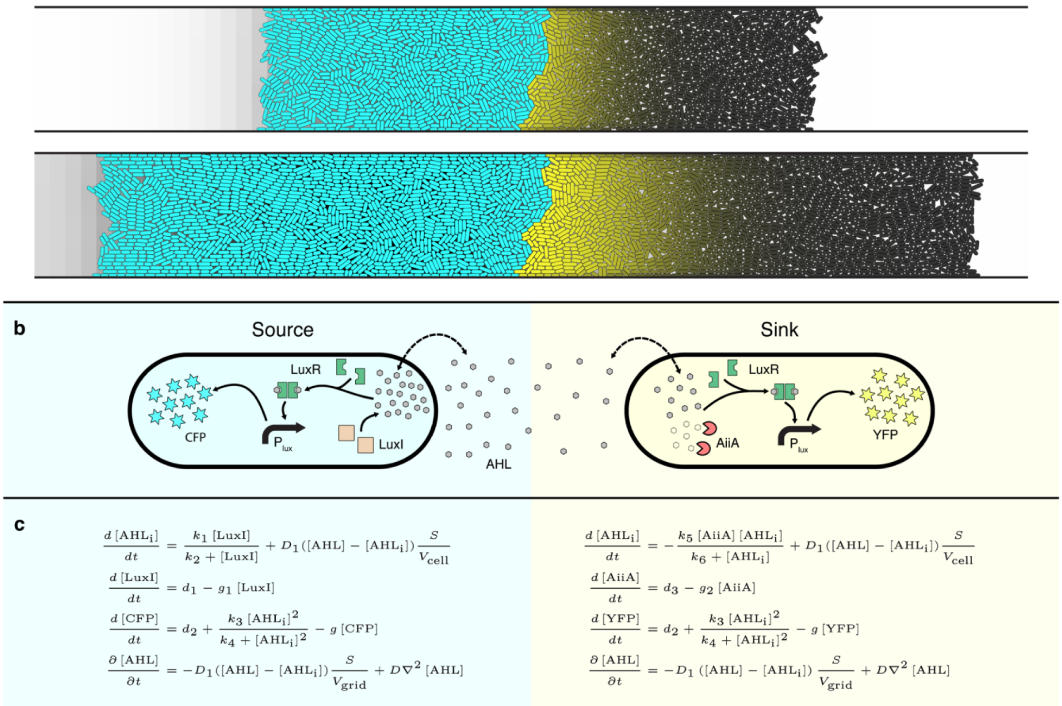


Figure 4.29: Edge detection circuit in CellModeller4. Figure taken from [181]

4.4.2 Colony boundary detection

Boundary detection between two synthetic colonies was studied with *CellModeller4* [181]. Their model consisted of two growing bacterial species in a microfluidics environment, where one species (the source) produces a diffusible signal molecule *AHL*, and the second species (the sink) produces a diffusible molecule *AiiA* which degrades *AHL*. *AHL* activates the production of a fluorescent protein in both populations (CFP for source cells, and YFP for sink cells). This produces a system where YFP is synthesized by the sink cells along the boundary with the source cells. Their model can be seen in Figure 4.29.

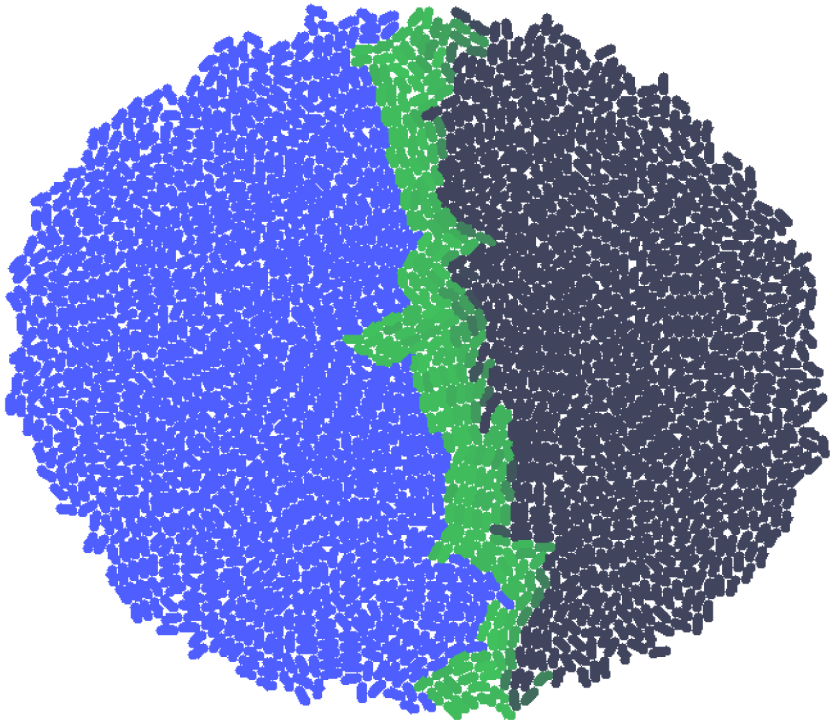


Figure 4.30: Snapshot of the edge detection circuit in *Simbiotics*, the circuit demonstrates the same qualitative behaviour.

Simbiotics was used to develop an equivalent model, for which the results can

be seen in Figure 4.30. Results for our model can be seen in Figure 4.32. The models show the same YFP expression band forming at the colony boundary, with a decaying intensity in YFP away from the boundary.

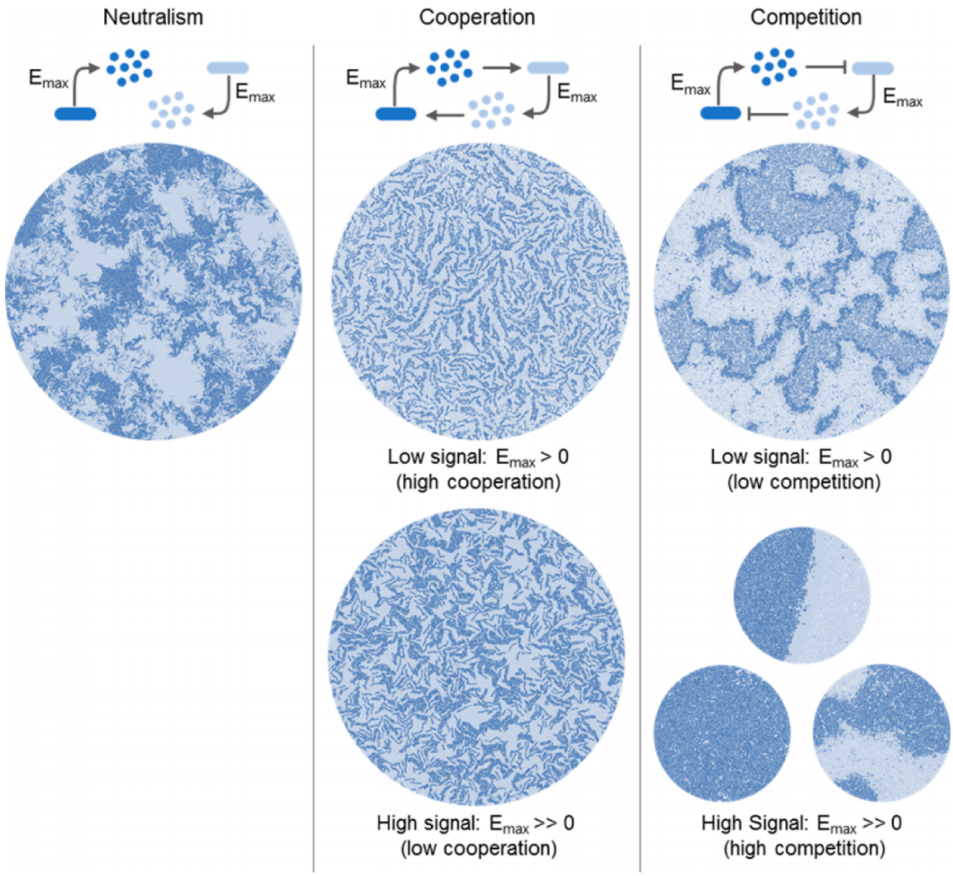


Figure 4.31: Bacterial ecology model studied with *gro*. Figure taken from [79]

4.4.3 Bacterial ecology model

Research into the ecological interactions of bacteria was conducted with *gro* [79], a simulation platform developed at Washington University. They studied ecological models of bacterial populations, observing system dynamics under different mutual relationships. Three ecology models involving a dual species population were considered. The first model treated the species as neutral to either other, they only interact through physical shoving and do not activate or inhibit each

others growth. The second modelled a cooperative relationship between the two species, where cells secrete a molecule that the other species uses for growth, encouraging cells to grow in close proximity. The third model was of a competitive relationship, where the species secreted a molecule which is toxic to the other species and inhibits its growth, resulting in separation of colonies of extinction of one of the species. Their results can be seen in Figure 4.31.

Models of the same ecological relationships between bacteria were developed in Simbiotics. The results of the simulations can be seen in in Figure 4.32, showing the same qualitative trends as observed in the original study.

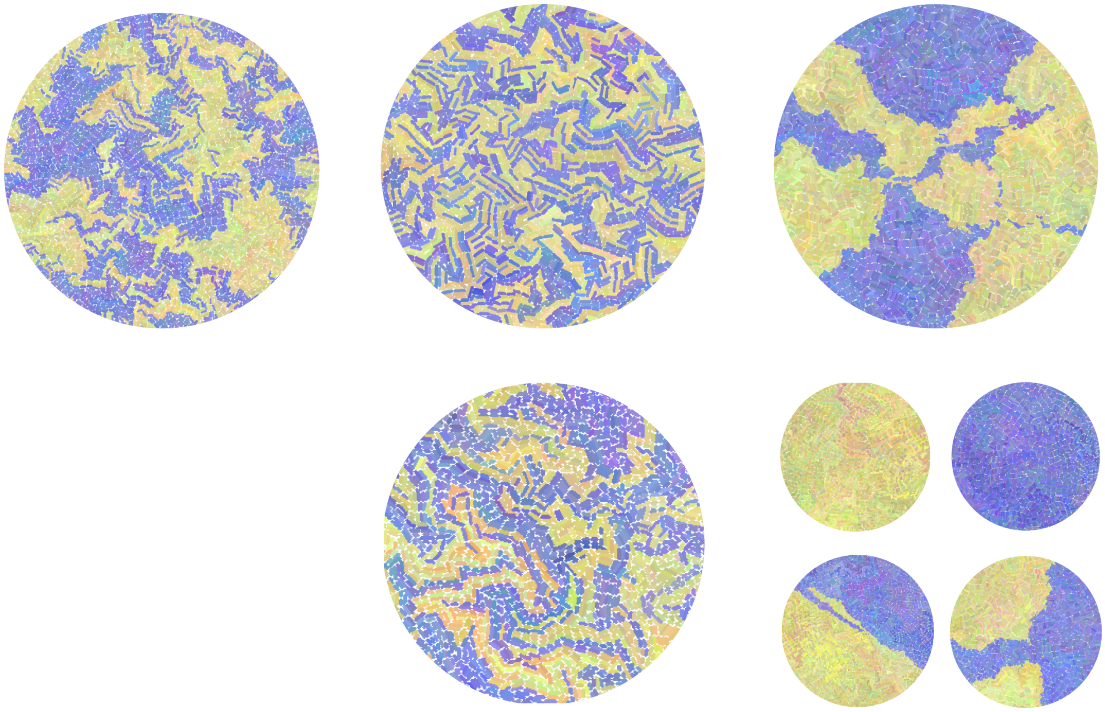


Figure 4.32: Snapshots of the ecology model in *Simbiotics*. The three models - *mutualism*, *cooperation* and *competition* - demonstrated the same qualitative behaviour and studied in *gro*.

4.5 Summary

This chapter has presented the validation of Simbiotics functionality through a series of small tests on models, acting as a sanity check on the implementation

of the platform and ensuring the correct implementation of simulation features. Some stress tests and performance analysis have been conducted, showing that the physics integration is the main bottle-neck as cell populations grow to large numbers. The performance of modelling cells as sphere (cocci) vs rod-shaped (bacilli) has also been presented, showing the performance benefits if the model can be simplified to cells being represented as spheres. The reproduction of existing models in the domain of multicellular simulations support their findings, and also demonstrate Simbiotics' flexibility in modelling a wide range of systems.

The models built in this section help fit parameters to produce realistic behaviour. For example, the constants used in physical force calculations (such as friction coefficient, and spring constant used in cell collisions) are fit by developing basic models of cell motion and collisions, ensuring these produce the expected results. These parameters are then used in more complex models, such as the models built reproducing literature results, that include additional dynamics such as cell growth. The parameters for these additional dynamics can then be fit, with those initial parameters fixed to the values found in previous models.

Chapter 5

Biomodel and numerical methods representation

This chapter presents the design concepts of the Simbiotics model and library, and describe the file format used for representing and managing them. This chapter address both Objective 1 and 2, implementing a data format to represent models and simulation methods in an flexible and extendable manner through modular design, that can be used to support a higher level modelling tool.

5.1 Overview

When developing models of systems there are two major concerns; how to represent the system of interest, and how to apply rules to generate meaningful information. Devising a way to represent a system is non-trivial, especially for complex-systems such as bacterial populations, for reasons discussed in Chapters 1 and 2. There are however underlying patterns in complex-systems, and therefore common principles can be used to model these systems. Similarly, though many different modelling techniques exist, they can be represented and arranged in some logical structure in order to be applied to a model appropriately.

In this chapter I discuss the design concepts in the Simbiotics framework which has been built to accommodate for these two modelling concerns. An explanation of how models and methods are represented is provided, followed by

a description of a data exchange format which was conceived to encode these models and methods. The data format allows for their reuse and communication, and also serves as part of the software API allowing for the loose coupling of software with Simbiotics.

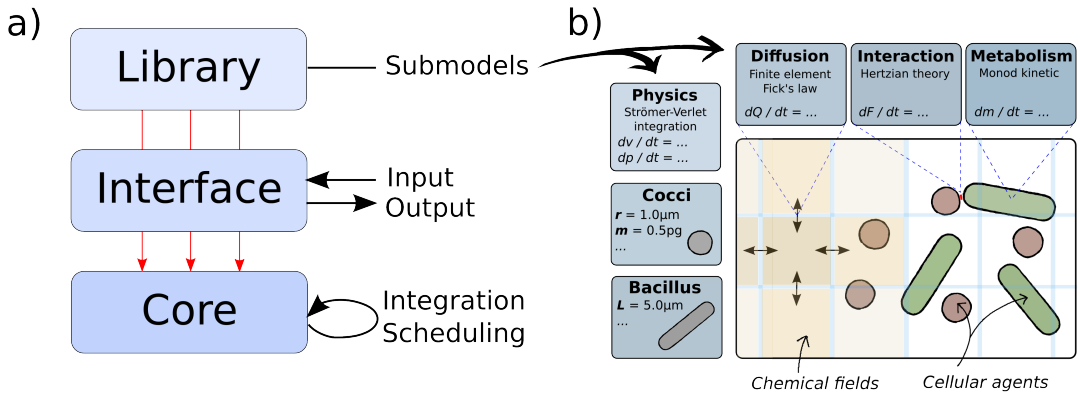


Figure 5.1: **a)** Simbiotics architecture overview, showing that the interface is a bridge with which the user attaches library models to the simulation core. **b)** A schematic of a basic Simbiotics model, composed of a set of 6 library submodels. The model specification utilises specific library modules for each feature of the system, ranging from cell shape to metabolic behaviour. The model specification can be composed and run via the interface layer. For a running model, the core layer integrates all model defined processes and schedules their execution for multi-threaded and multi-CPU environments.

5.2 Simbiotics Implementation

Simbiotics is developed in Java; it utilises the spatial representation, multi-threaded and multi-CPU parallelized execution, and the 3D rendering as implemented in the Cortex3Dp platform [177, 238]. The platform is designed with a modular architecture, allowing for model features to be represented as discrete components that can be readily added, removed and modified for the specific modelling application. This is achieved via a three component architecture comprising of a simulation core, a modelling library and a modelling interface. This

plug n' play framework allows for rapid model prototyping and reiterative designs for the reification of models. The software architecture is depicted in Figure 5.1 (a).

The core of Simbiotics is the computational engine, it deals with representing the system state, integrating all model defined processes and scheduling commands for parallel execution.

The modelling library contains a collection of modules, which are discrete submodels describing specific model behaviour. These range from physical law integrators and chemical diffusion-reaction solvers, to bacterial geometries, cellular dynamics and boundary conditions. Modules describing virtual lab components and scheduling are also present, accompanied by analysis and data exporter modules. Modules are all parameterisable to allow for their customisation. An exhaustive list of present library modules can be found in the Simbiotics Guide (user manual) in Appendix B.

The modelling interface allows the user to specify the inputs and outputs of the platform. Models can be designed by composing library modules in a Java class, or alternatively in a JSON model file which is then parsed into the corresponding Java objects. The interface also allows for the optional real-time 3D rendering of the simulation, with live graph plotters visualising model statistics and on-the-fly analysis.

Simbiotics is packaged into a stand-alone *jar* file, which can be run from command-line. It requires a configuration file which contains the Simbiotics parameters and file path to the JSON model file. A full description of how to compose models in Simbiotics can be found in the User Manual.

Note that Simbiotics is unit agnostic, such that it does not enforce the units for the parameter values. This means that the modeller is responsible for ensuring parameters are converted to consistent units.

5.2.1 Representation of a bacterial cell

Bacteria exhibit a wide range of behaviours as discussed in Chapters 1 and 2. A single bacterial cell can also differentiate throughout its lifespan, resulting in potentially significant changes to its behaviour. In order to capture this in a

model, we start by representing a single cell as a discrete entity which has a *morphology*, and a set of behaviours. The *morphology* represents the physical shape of the cell, this may be a single object or complex structure consisting of multiple geometric parts. The *behaviours* represent the cell's processes and interactions. Each behaviour is an independent module concerned with modelling a particular process. This model forms a tree, with the species object being the root node, and the *morphology* and *behaviours* two child nodes. A schematic showing the representation of a single cell can be seen in Figure 5.2.

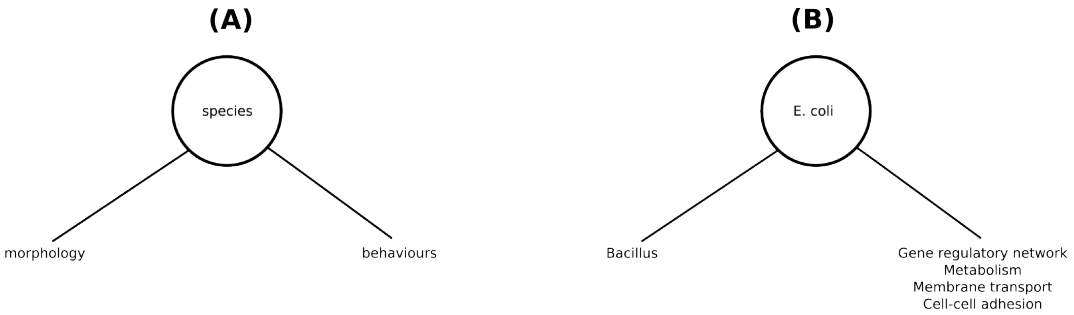


Figure 5.2: Basic model of a single cell. It's represented as a tree where the root node is the species unique id, which has two child nodes, describing the species' morphology and associated behaviours (cellular processes). **(A)** The abstract structure of the basic single cell representation. **(B)** An example of basic cell species with a bacillus morphology and some associated cellular behaviours.

This method of modelling a single cell is rather rigid as it has a static set of behaviour modules which are independent processes, whereas in reality these processes are interdependent and may change over time. To accommodate for this, we introduce a set of *states* and *links* to the model, as seen in Figure 5.3. The *states* can be used to represent cell properties and abstract decision making flags, which the behaviour modules can read/write allowing for the integration of separate behaviours. The *links* can connect states, behaviours and the morphology between each other, allowing for dynamic changes to the cell's behaviour during simulation. This way of programmatically representing a cell allows for the integration of a wide range of cellular machinery and can also capture differentiation of individual cells.

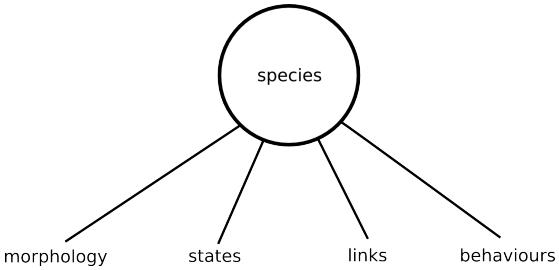


Figure 5.3: Model of a single cell which allows for more complex dynamics to be represented. The species has four child nodes, where the *morphology* describes its shape, the *states* describe its properties, the *behaviours* describes it processes, and the *links* describe relations between these parts.

5.2.2 Representation of a population model

The representation of a population model is a super-set of how we represent a single bacteria. A model can contain many species of bacteria and environmental factors relevant to the system such as extracellular chemical gradients and physical forces. Additionally a model may contain data collectors, devices for interacting with the model, and schedules to define events or periodic actions.

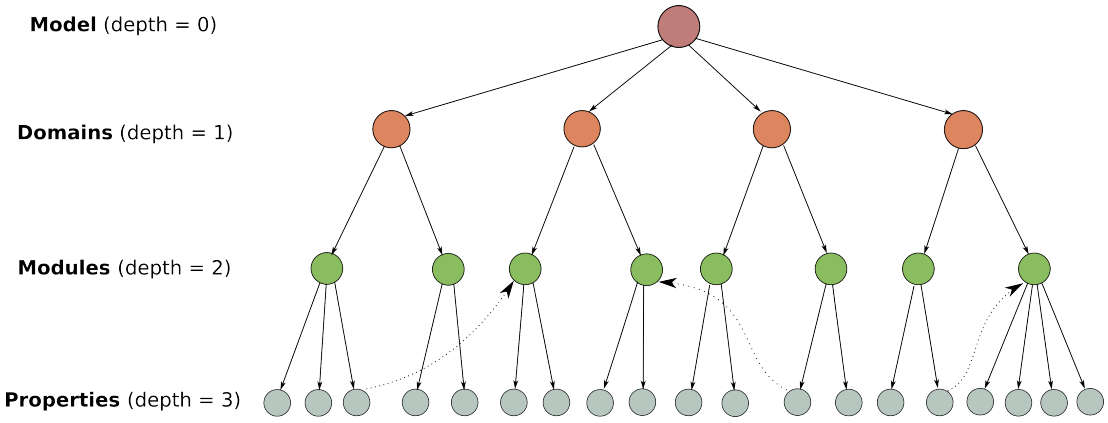


Figure 5.4: A model is represented as a tree, where each depth in the tree corresponds to a different modelling concept.

The model is also represented as a tree, where each depth of nodes relates to a different modelling concept, as can be seen in Figure 5.4. The root node (depth = 0) is the model object which has many child nodes (depth = 1). Each of these child nodes corresponds to different *domain*, where a domain is a sub-

section of processes in the model. For example, there are different domains for defining bacterial species, data collectors, schedules and environmental factors. Each domain can also have many child nodes (at depth = 2), where each of these nodes represents a *module* in that *domain*. A *module* implements a specific model feature. For example, in the domain relating to environmental factors, there is a module implementing chemical diffusion in the extracellular space, and another to define how gravity acts on cell morphologies. Finally, each module may have many child nodes (at depth = 3), where each child node is a *property* of that module. An example of module properties has already been seen in the model of a single bacterial cell in the section above; the species is a *module*, and the morphology, behaviours, states and links are each a *property* of that module. Figure 5.5 is a schematic showing part of a population model which follows this structure, with the single cell model highlighted in blue.

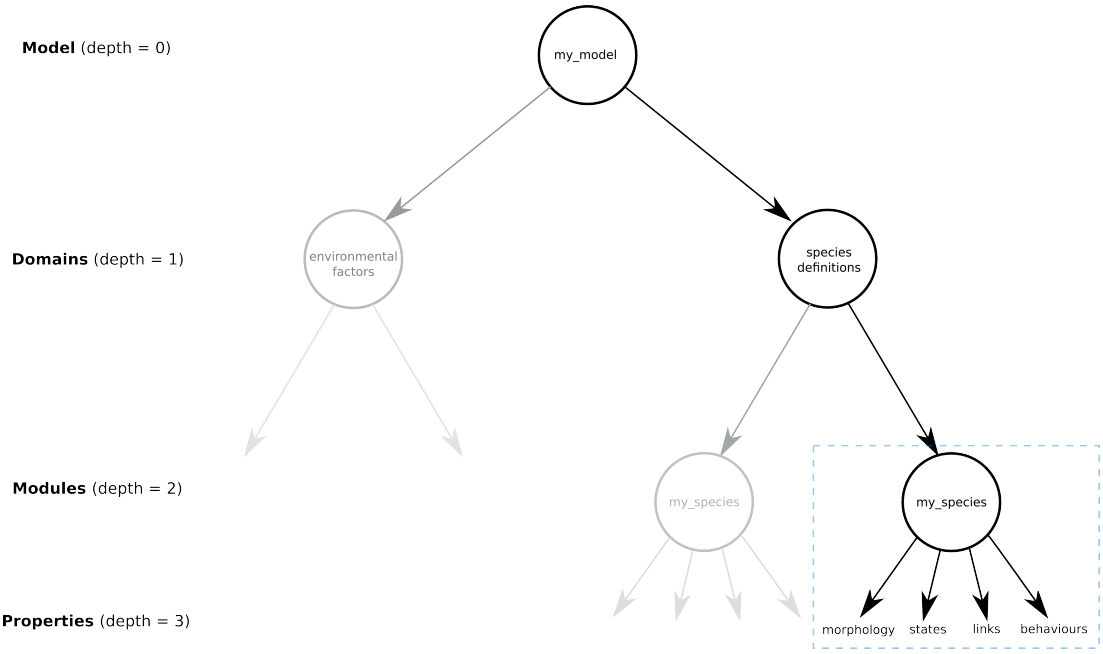


Figure 5.5: Schematic depicting part of a model with the single species model in its species definition domain (highlighted in blue).

The properties of modules can point to other modules via a unique string ID system, allowing for the weak referencing between modules to compose complex simulation objects. For example, a species definition has a *morphology* property,

this property is the string ID of the target morphology module which is defined in the *morphologies* domain. This can be seen in Figure 5.6. This method of binding modules together is somewhat similar to an entity-component system, as described in Chapter 2. It is a compositional design pattern where functional units are composed by tying together modular parts in a loosely coupled manner, meaning that minimal changes need to be made when whole program or model when a module is modified or substituted.

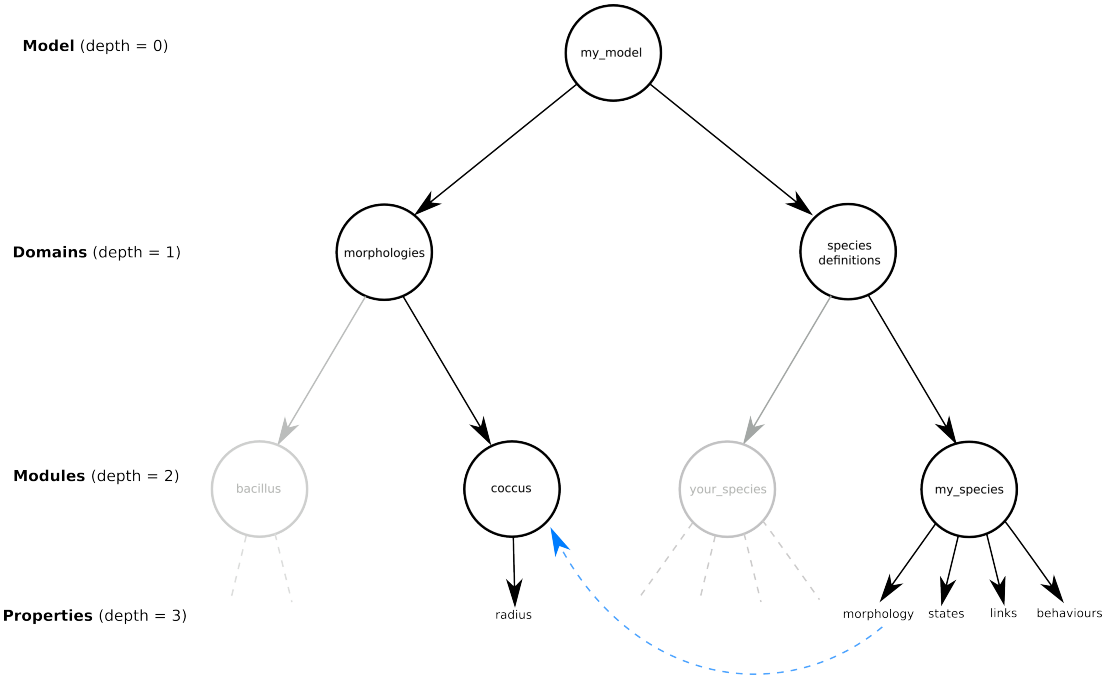


Figure 5.6: Complex models can be developed by connecting modules in different domains together, this can be done by setting a module property point to another module.

This method also allows for the reuse of defined modules in the model, for example two species may have the same morphology or behaviour process, therefore they both refer to the same module rather than having to define the same thing twice. An example can be seen in Figure 5.7, showing two bacterial species defined, where *your_species* has a *mitosis* behaviour module describing how the cell divides into two child cells, and *my_species* has this *mitosis* module as well as a *flagellar* module describing its motility.

Representation of models by this method is flexible, the modular design allows

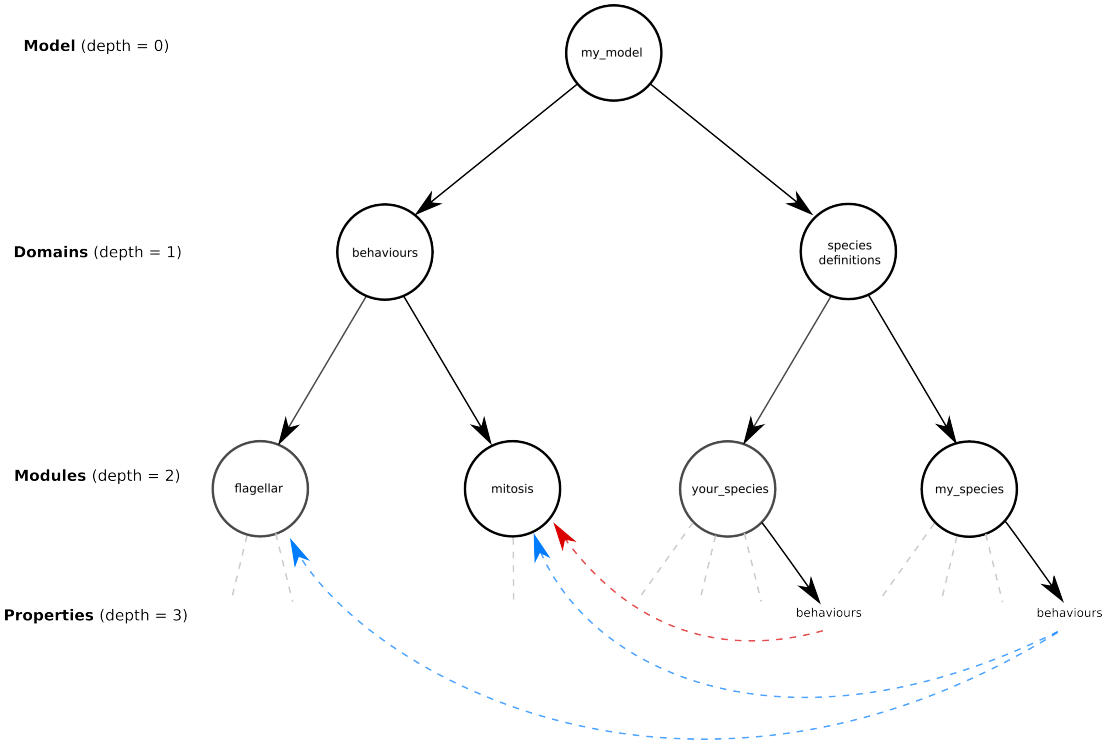


Figure 5.7: Modules can be seen as templates which describe a specific behaviour, this means that a module can be used by numerous other modules, as they instantiate their own version the module. For example if a behaviour module describing cellular mitosis is defined, this can be used by all cell species that undergo that type of mitosis, rather than having to redefine it for each. This structure is similar to an entity-component system.

for new *domains*, *modules* and *properties* to be easily introduced and connected together into functional units. Now we have addressed the first modelling concern which is devising a way to represent the system, now describe the method to apply rules to the model (numerical methods) to generate meaningful information.

5.2.3 Representation of numerical methods library

Numerical methods are implemented in Java as part of the Simbiotics platform, and are stored in the modelling library. This modelling library is a tree structure, similar to the model representation. Each numerical method is implemented as a *module*, and these modules are arranged into *domains*, as can be seen in Figure

5.8. The domains, modules and properties in the library are equivalent to those in a valid model.

The different depths of the library tree are the same as the modelling library, except for an additional intermediate layer (at depth = 3) containing the metadata for modules, between the *modules* and *properties*. Each module has two metadata nodes, the first is class name of the Java object this module corresponds to, and the second is the template containing the *properties* with their default values.

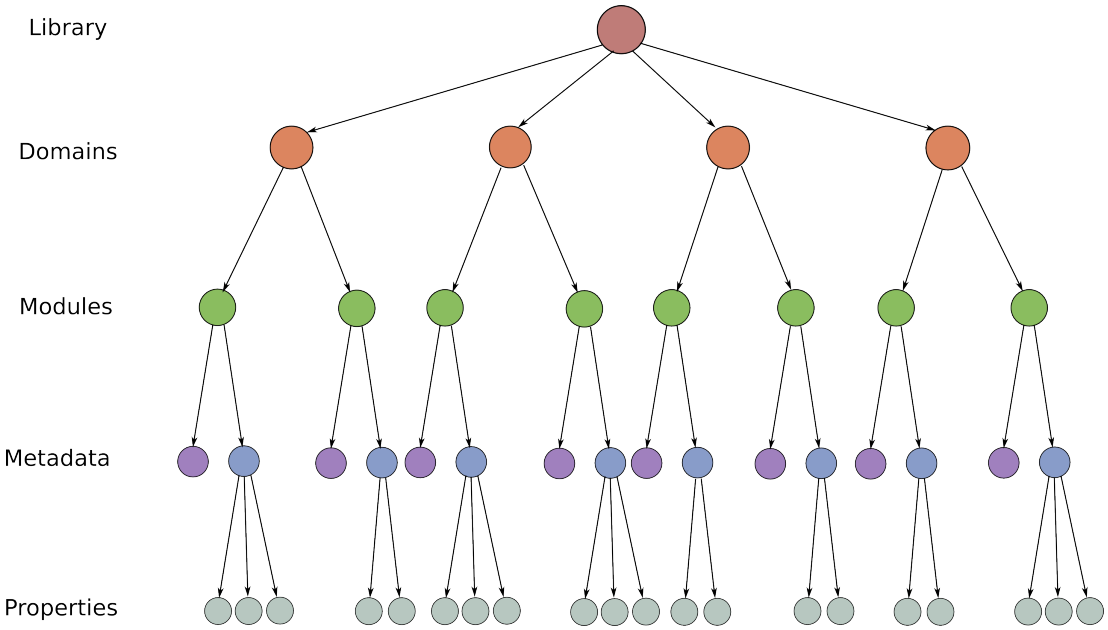


Figure 5.8: Schematic showing structure of the numerical methods library. The library is structured as a tree similar to the model, however there is an intermediate *metadata* layer in-between *modules* and *properties*. This metadata node contains one node containing information such as the Java class name this module corresponds to, the other containing the template of properties for that module with default values.

Models are constructed by composing library modules together, and library module may have many instances defined in the model. Each module in the library has a unique string identifier which is used in the model definitions. Each module also has a name, which can be set by the modeller to differentiate between different instances of a module.

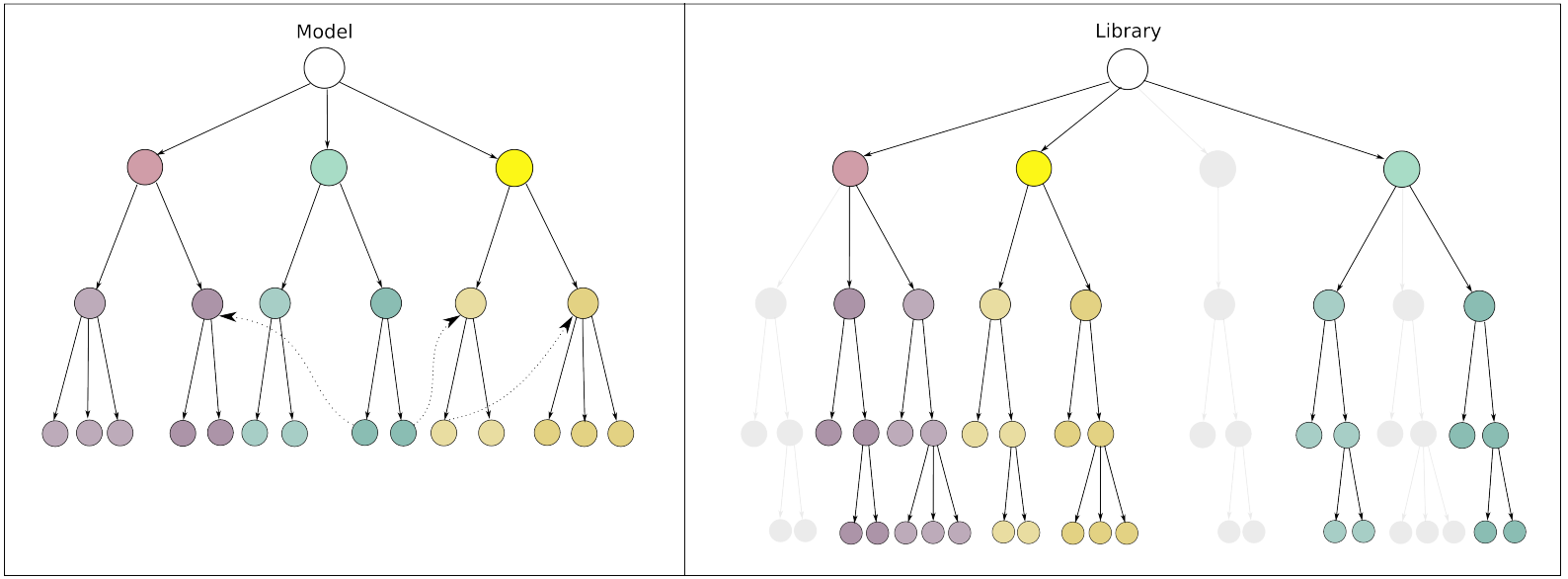


Figure 5.9: Example of a *model* and *library*, where the model is made up of parts from that library, as depicted by the node colours. The model has modules connecting together to create functional components. The library has an extra layer of nodes containing the metadata of the modules. The grey nodes in the library depict modules which are not used in the model.

This representation provides an easy mapping between a model and the numerical methods to simulate them. Changing which method is used to simulate a particular aspect of a model simply involves changing the string id to a different library module in that domain. A schematic can be seen in 5.9 showing how the contents of a model and of the library related, a model is a subset of the features available in the library. Additionally this structure allows for the library to be readily extended, as new branches can be added to the tree horizontally, adding new domains, modules and properties with ease.

```
{
  "type": "object",
  "required": ["id", "data"],
  "properties": {
    "id": { "type": "string" },
    "data": {
      "anyOf": [
        {
          "type": ["string",
                           "number",
                           "boolean",
                           "integer",
                           "object",
                           "null" ]
        },
        {
          "type": "array",
          "items": {
            "anyOf": [
              { "$ref": "#" },
              { "type": "string" },
              { "type": "boolean" },
              { "type": "number" },
              { "type": "integer" }
            ]
          }
        }
      ]
    }
  }
}
```

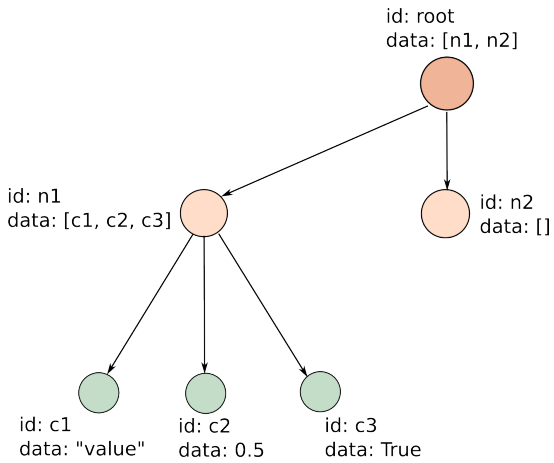


Figure 5.10: Schematic showing the base structure of the exchanged JSON data files. Files consist of a single JSON object which is considered to be the root node of a tree. A node has an "id" field which is its unique string identifier, and a "data" field which may be many child nodes or a primitive value such as a number or string.

5.3 Files and related schemata

A model tree and library tree can both be represented in a machine tractable file format. This file format chosen is JSON (Javascript Object Notation), as it has simple syntax, data structures, and there are many available open-source software libraries for reading and writing JSON. This allows for data exchange with Simbiotics in a loosely coupled manner.

To represent a tree in JSON we consider a node in the tree to be a JSON object, which has an string identifier and some data describing its children. This structure is encoded as a JSON schema, as seen in Figure 5.10, where there may only be one root node, and each node must have an *id* and *data* field present. The *id* must be a string identifier, and *data* can be any value, including one or many child nodes.

This is the basic structure that the model and library JSON files follow. These files do have other constraints, such as that each depth of the tree is allocated to a modelling concept (*depth=1* maps to *domains*), and that there is a set depth that the trees must be in order to be parsed by Simbiotics. More complex schema are generated to enforce the specific structure of model files and library files, as explained below.

5.3.1 Model file

A model file encodes the tree structure as seen in Figure 5.4, and a depiction of it can be seen in 5.11. It contains the domain modules and properties for that specific module. A Simbiotics Java model can be exported to a JSON model file. The JSON model file can then be used as an input to Simbiotics, which parses it back into its Java form for simulation. A valid model file for Simbiotics may only contain *domains*, *modules* and *properties* which are present in the numerical methods library. To ensure this, a model schema is generated by Simbiotics based on the library, it enforces the *id* and *data* tree structure with constraints on which domains, modules and properties are permitted at each depth of the tree.

Specifically, the model schema enforces:

- The *id* of the root node is the string name of the model, and its *data* contains a list of nodes which encode the *domains*. Its *id* is constrained to be a string.

- Each domain has its *id* set to its unique string identifier, and its *data* contains a list of nodes which encode the *modules*. The *id* value is constrained to be one of the domain names present in the library.

- Each module has its *id* set to its unique string identifier, and its *data* contains a list of node which encode the *properties*. The *id* value is constrained to be one of the module name present in the library domain it is defined in.

- Each property has its *id* set to its unique string identifier, and its *data* can be any primitive or data structure (array or map), it may not have any child nodes. The *id* value is constrained to be one of the property names present in the library for that specific module.

The model file schema is generated directly from the Simbiotics source code, making it very easy to update the schema as the platform is further developed. The generation is achieved via the following steps:

- Each *domain* is a map in the Simbiotics Library class. The map's variable name is used as the domain *id*.

- Each *module* is an object which exists with one of the domain maps. Modules are assigned a unique identifier which are used as the module *id*.

- Each *property* is a public variable in a module class. The variable name is used as the property *id*, and its value is used the default value for that property, stored in its *data* field.

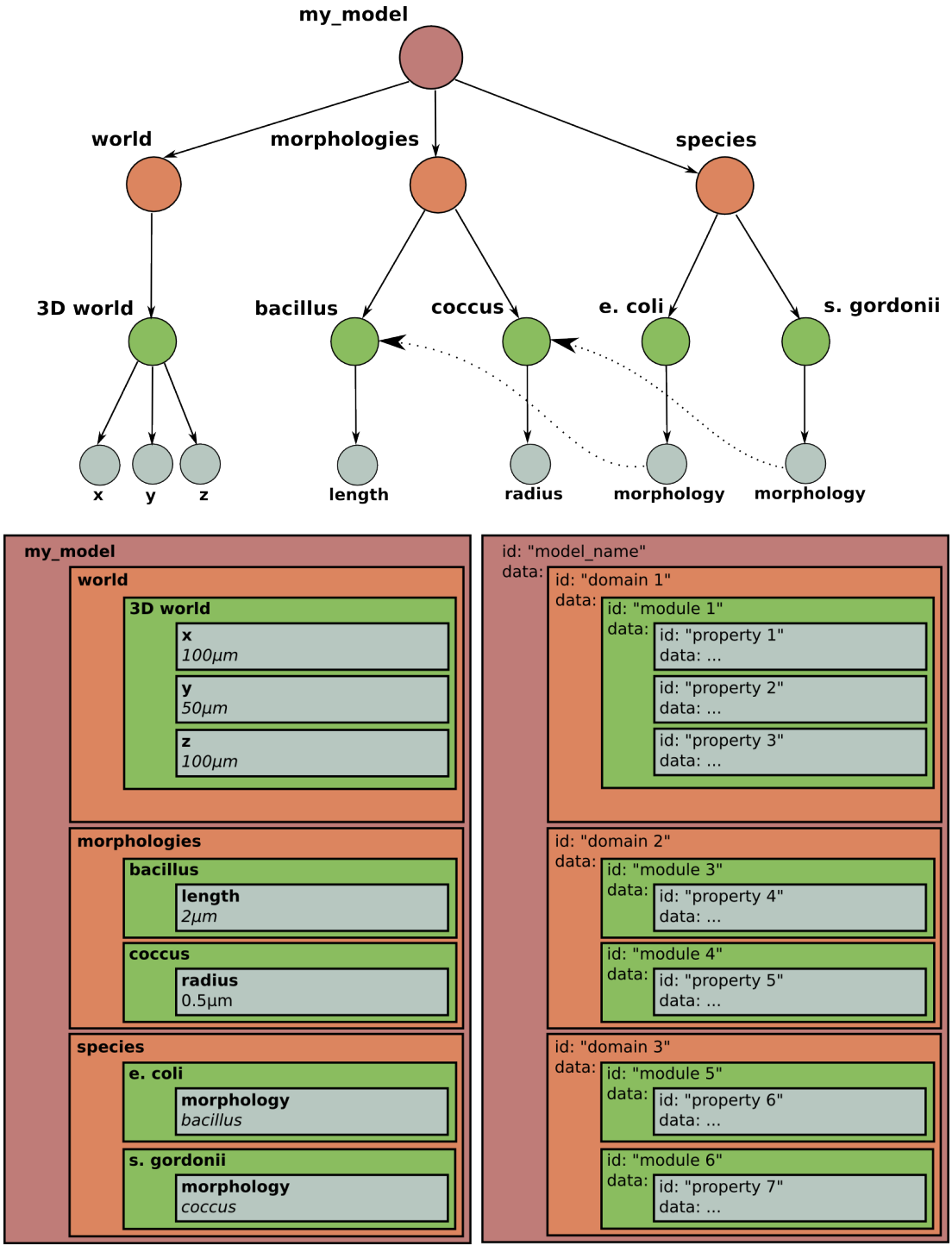


Figure 5.11: Schematic showing structure of a JSON model file. The colours depict the relationships between the different forms. **Top:** A model in its tree representation **Bottom Left:** The encoded model, using JSON objects and arrays to preserve the hierarchy of the tree. **Bottom Right:** The encoded model file showing the preserved *id* and *data* pattern.

5.3.2 Library file

The library file encodes the numerical modelling library, abiding by the structure seen in Figure 5.8. A depiction of the library file can be seen in Figure 5.12. It contains all of the numerical methods that are present in the Simbiotics library, and is generated directly from the Simbiotics source code in a similar way to how the model schema was generated. The library file schema enforces the *id* and *data* tree structure, and that the *metadata* structure is followed (only 2 child nodes, one which may not have child nodes).

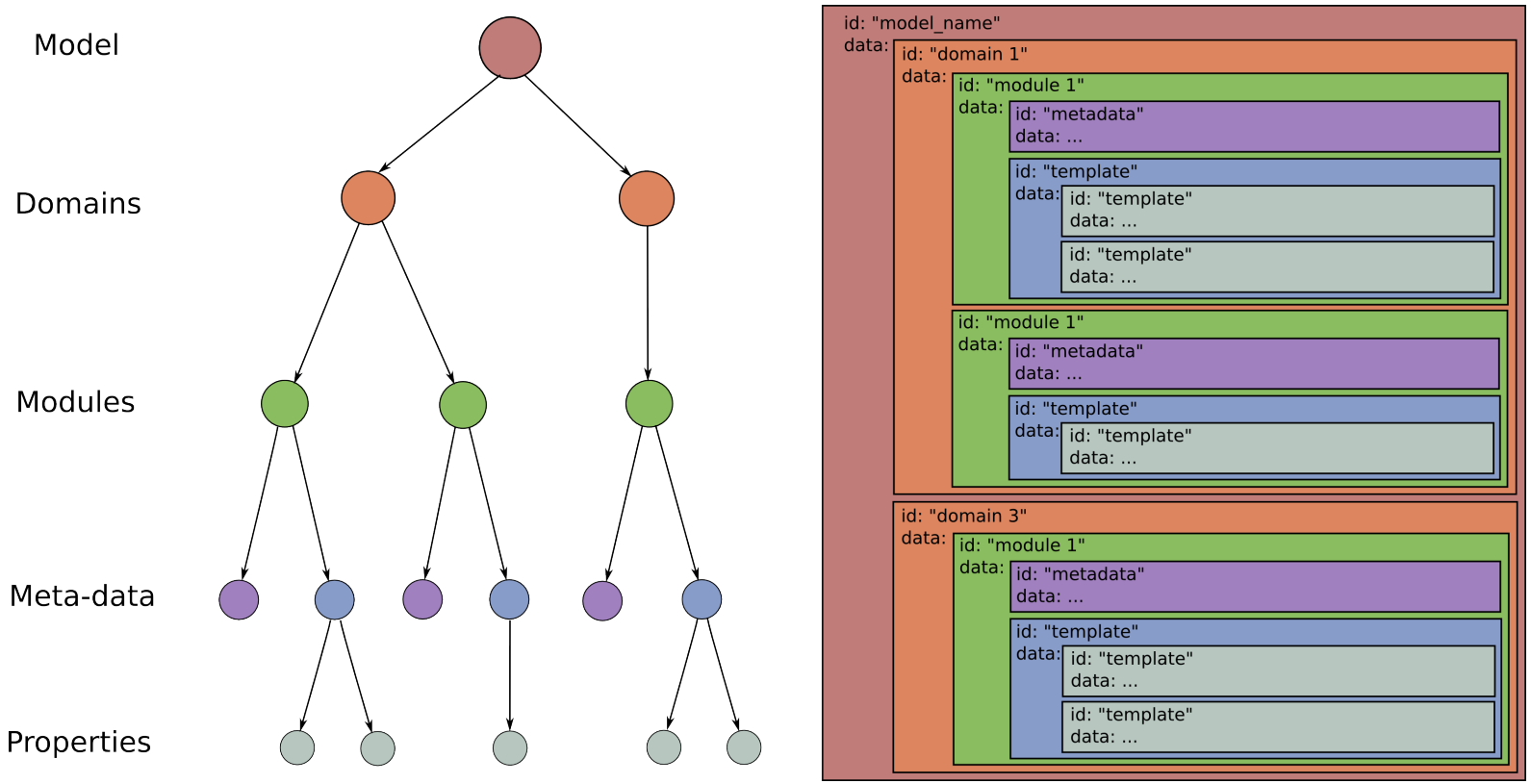


Figure 5.12: Schematic showing structure of a JSON model file. **Left:** A model is structure is a tree where each depth corresponds to a different part of the model. **Right:** Each depth of the tree corresponds to a different element in the JSON file. The colours depicted the relations between the two representation.

5.4 Standard data exchange format for population biomodels

The model and library representations developed, along with their corresponding file formats, allow for the communication of population models and simulation methods. These files can be used as intermediate formats for exchanging population models, parsed by software APIs to connect simulators, model development environments and other related software in a loosely coupled manner. This allows for a potential ecosystem of tools being used to develop/modify/simulate/analyse population models. This could act as a partner to SBML (described in Chapter 2), which is an XML flavour used for modelling single cell dynamics, and has a large number of tools which can read/write it. The format we propose can embed SBML models into physically interacting populations with additional behaviours that SBML does not represent, allowing for the specification of interacting SBML models.

5.5 Summary

In this chapter I have described the representation of biomodels and the Simbiotics library. Models are comprised of library modules which are composed together, allowing the direct mapping to the numerical methods required to solve a model. They can easily be edited, with model features being substituted with other modules from the modelling library. The model and library can both be encoded into a flexible and extendable file format, with a structure that is enforced by schemata, allowing for their reuse, communication and parsing by common programming frameworks. This conceived model file format offers an extension to the SBML standard, allowing for populations of interacting SBML models to be embedded in spatial population models. The file formats are a flavour of JSON, which means they can be easily read and written by other software across many programming languages. This provides the basis on which population biomodels can be shared and integrated into related software tools beyond Simbiotics.

Chapter 6

Easybiotics - a graphical environment enabling rapid population modelling and analysis

This chapter presents Easybiotics, a graphical user interface (GUI) for the use of Simbiotics without programming experience. The implementation and features of Easybiotics are described, elaborating on the modelling environment, as well as how it interfaces with Simbiotics in a flexible manner. A brief example of model prototyping and refinement is presented, demonstrating how Easybiotics can be used to facilitate biomodel design and analysis. This chapter addresses Objective 2 - Development of an easy to use interface to enable those with minimal programming experience to build models of mixed consortia of bacteria.

6.1 Overview

To ease the use of integrative modelling (IM) techniques by domain experts I have developed Easybiotics, a graphical user interface (GUI) for multicellular modelling and analysis. Easybiotics acts as an abstraction layer to a simulation framework (in this case Simbiotics), freeing the modeller to focus on developing and analysing biomodels, rather than on programming simulator and model implementation details.

The design of Easybiotics has been determined by an informal requirements analysis based on conversations with Masters students who are using Simbiotics and similar tools, as well as experimentalists I have met at conferences and workshops. They provided me with a clear picture of what types of models they are building, the questions they would like to ask the model and their workflow. For example the inclusion of an interface to allow simulating populations of SBML models was motivated by conversations with modellers at conferences. Furthermore the awareness that many people engaging in multicellular modelling are not programmers by training, the requirements for the software included being a simple to use tool which does not require programming knowledge.

An easy to install process and minimal software dependencies allow for the use of tool in many contexts, reducing the barriers to entry for using integrative modelling techniques. Easybiotics is applied to Simbiotics, visualising the Simbiotics library and allowing for simple model composition, simulation and analysis. This allows for use of Simbiotics without having to program Java models or use the command-line to run simulations.

Easybiotics is developed in Python 2.7, and uses the Kivy library for creating graphical widgets. The Pandas module is used for data and file handling, and Matplotlib is used for rendering graphs. The software is developed in a standalone manner, and has an API which can interface with Simbiotics or potentially a different simulator.

6.1.1 Loose-coupling with Simbiotics

To ensure the scalability of Easybiotics as modelling techniques advance, it is developed in a dynamic manner which populates the interface based on the *library* file (as described in Chapter 5). This loose-coupling between Easybiotics and Simbiotics ensures minimal if any changes need to be done to either software if the other one changes. Specifically Easybiotics and Simbiotics have to communicate about the following:

1. Simbiotics needs to tell Easybiotics what modelling features it has so that Easybiotics can display the library and model interactively;

2. Easybiotics needs to tell Simbiotics to run a model file.

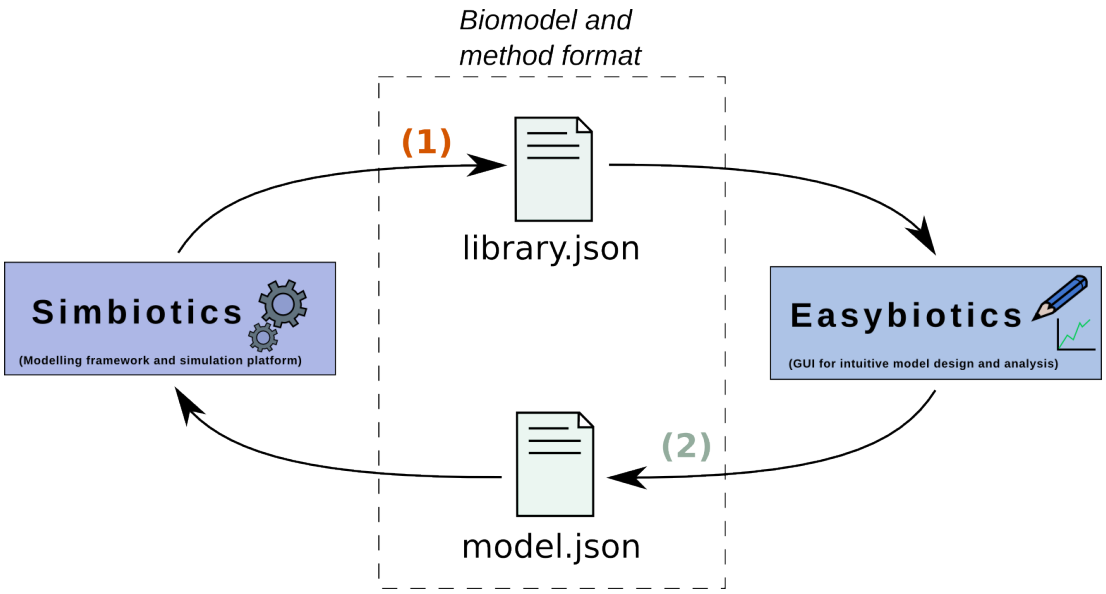


Figure 6.1: Schematic showing exchange protocol between Simbiotics (back-end) and Easybiotics (front-end). The requirements (1) and (2) are illustrated, showing the files output my Simbiotics which are used by Easybiotics. Simbiotics exports the modelling library JSON file, which informs Easybiotics on what modelling features it has. Easybiotics then displays these features and lets the user create a model JSON file. This model file must then be checked against the model schema provided by Simbiotics, which ensures the model structure and content is valid.

To achieve (1) Simbiotics exports a *library.json* file which is interpreted by Easybiotics. To achieve (2) Easybiotics calls Simbiotics via its command-line API, passing the model file. Figure 6.1 illustrates this.

6.1.2 Dynamic population of Easybiotics interface

To ensure the scalability of Easybiotics as modelling techniques advance, it is developed in a dynamic manner which populates the interface based on the library file (as described in Chapter 5). The library file (a JSON tree structure) is parsed and stored as a dictionary of dictionaries in Python, as to preserve the hierarchy of the tree.

The available *domains* are loaded from the library and visualised in the *model editor*. When the user selects to add a *module* to a model *domain*, the entry in the library dictionary for that domain is rendered by Easybiotics as a sub-library

for the user to browse. When the user clicks on a *module*, its *properties* are retrieved from the map and visualised dynamically. Properties of type *String* or *Number* are rendered as text input boxes, and *Boolean* properties are displayed as toggle switches. This can been seen in Figure 6.2, where (A) shows a model loaded in Easybiotics, and (B) and the corresponding model file structure. This

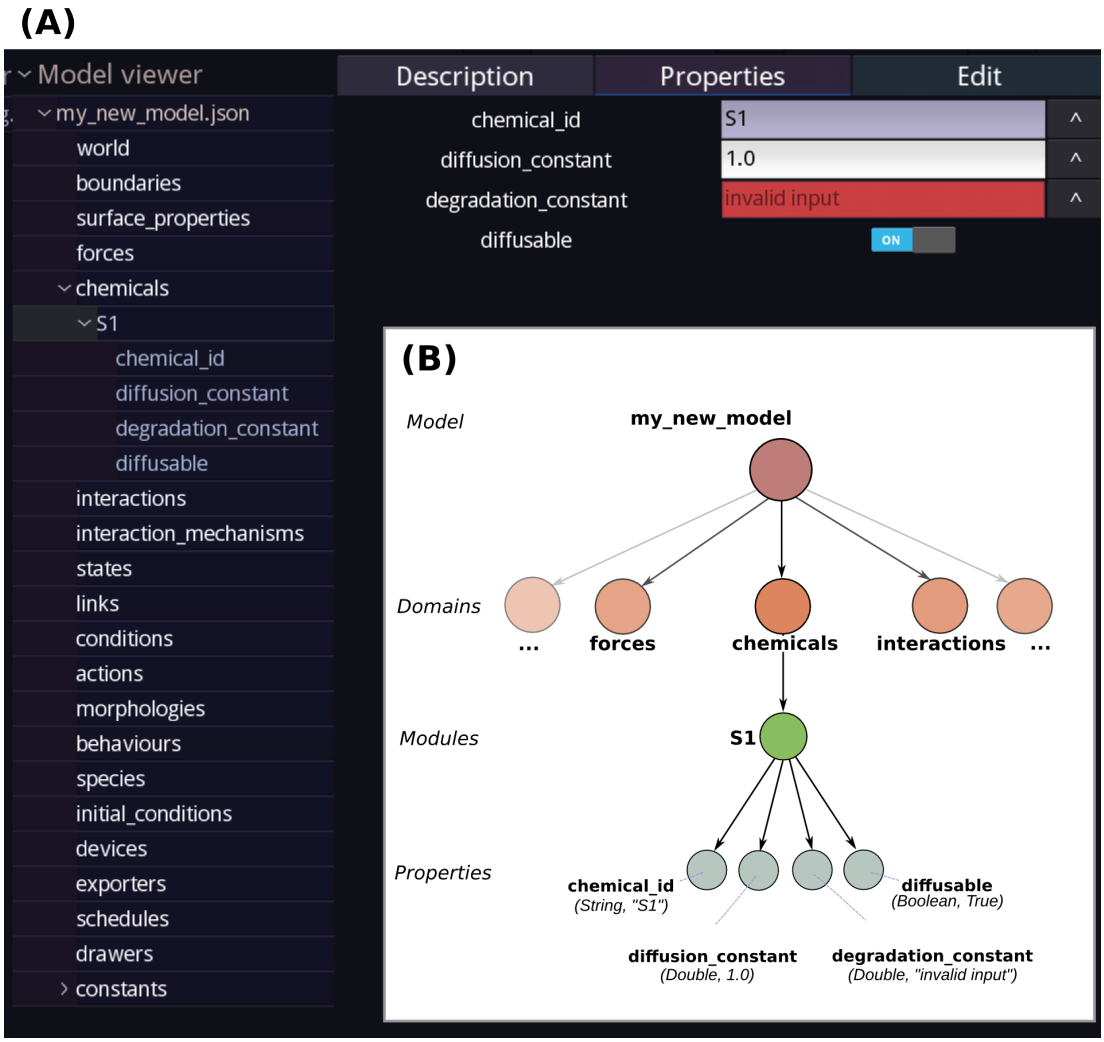


Figure 6.2: The Easybiotics interface is dynamically populated from input files. **(A)** Screenshot of Easybiotics with a simple model loaded, consisting of a single module, *S1*, in the *chemicals* domain. The properties of S1 are visualised in the model tree, and if the S1 module is selected its properties also also display panel property viewer (highlighted in blue). **(B)** A schematic showing the model file structure.

method allows for automatic detection of invalid inputs types, as can be seen by the *degradation_constant* parameter highlighted in red in Figure 6.2 (A).

Currently Simbiotics is the provider of the *library* file, however theoretically another simulator could have an API which achieves the same thing, due to the flexible modular design of this Java <-> Python protocol. Note that for the case of Simbiotics, the *domains* correspond to *maps* in the Simbiotics java library class, the *modules* correspond to java objects in those maps, and *properties* of a module are the public variables in the corresponding java object (as described in Chapter 5). This makes it a completely automatic process to integrate new modules (and even domains) into Easybiotics as they are developed in Simbiotics.

6.2 User friendly integrated-development environment for biomodelling

Easybiotics provides a simple GUI for complex modelling with relative ease and speed relative to programming Simbiotics models in Java. A simple click-and-select interface allows for models to be composed from library modules, and visualised as an interaction modelling tree (highlighted in green in Figure 6.3). The model items can be selected and a description of them, as well as access to the modifying module properties is available in the display panel (highlighted in red in Figure 6.3). The settings to the Simbiotics simulator, such as whether to load the real-time model visualisation or SBML integration, can be modified in the configuration editor (highlighted in blue in Figure 6.3). Additional features such as live graph plotting and parameter sensitivity analysis are available from the file-bar (highlighted in yellow in Figure 6.3). Regularly used functions such as running and saving a model are available on the button panel (highlighted in orange in Figure 6.3), as well as by hot-keys (which are listed in the *Help* menu on the file-bar). Information regarding which model is loaded, the project name, and the currently selected model item are displayed in the information panel (highlighted in purple in Figure 6.3).

The interface contains descriptions of the menu system and functionality, as well as information detailing the library modules. A series of example projects are

Figure 6.3: Overview of the Easybiotics modelling interfaces. The file bar is highlighted in yellow, the Simbiotics configuration editor in blue, the model specification editor in green, the display panel in red, and the button panel in orange.

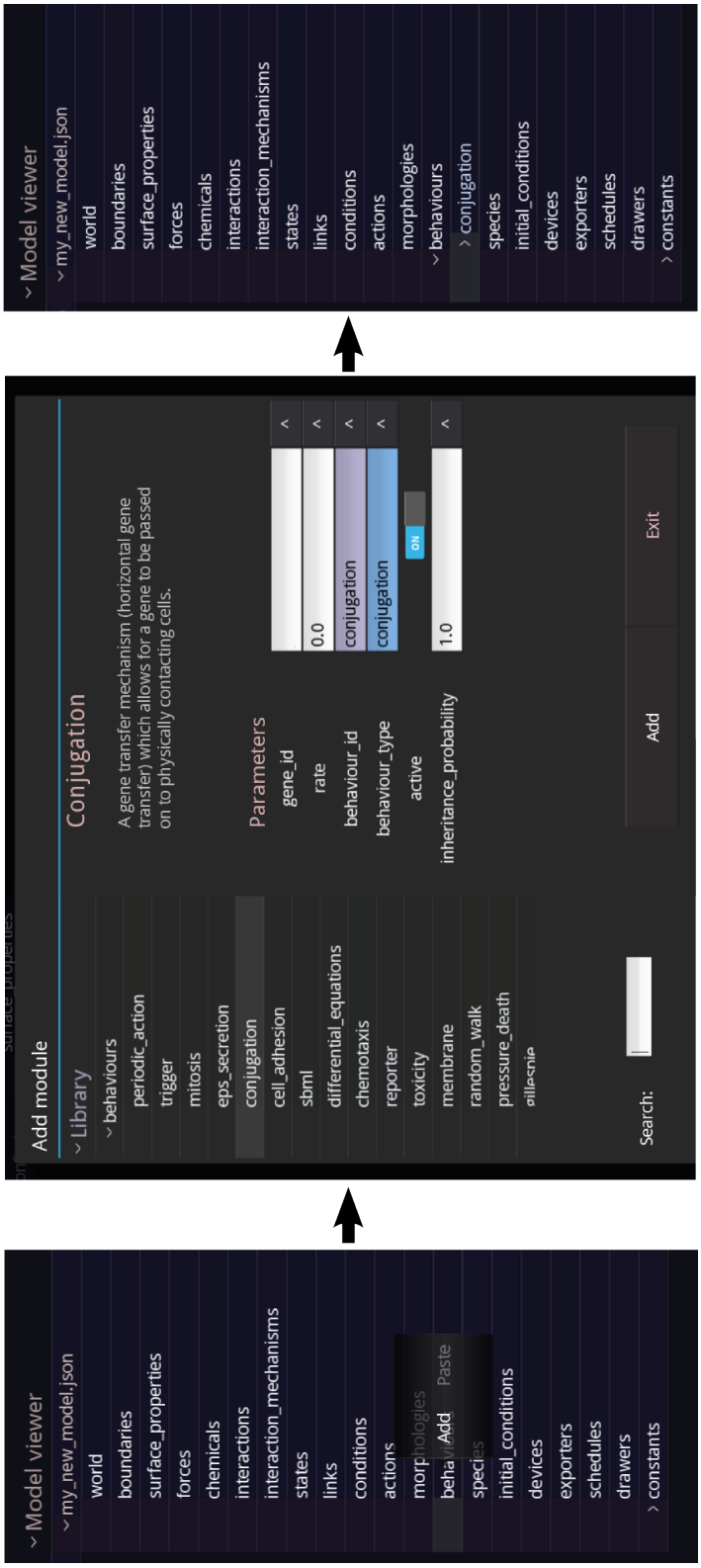


Figure 6.4: (A) shows the model specification right where the user has right-clicked on *behaviours*, and is presented with the **Add** button. (B) When the add button is selected the *behaviours* sub-library is displayed in pop-up box. Note: there are hotkeys to perform equivalent actions (which are found in the *Help] menu on the file bar.*

included in the software to illustrate model composition and analysis methods. Library module parameter inputs have warnings notifying the user if an incorrect value type is entered.

The amount of RAM that the JVM (Java Virtual Machine) uses for the simulation can be set by selecting 'Settings - Run Settings' from the file bar. Here you can set the initial amount allocated as well as the maximum amount the simulation may use. Models can be run by either clicking the 'Run' button in the button panel, or by selecting one of the run options on the file-bar.

6.2.1 Intuitive model development

Model development in Easybiotics is simple and compositional. Right-clicking on a model *domain* and selecting **Add** opens the corresponding sub-library, in which the *modules* of that domain can be browsed and selected. An example of this can be see in Figure 6.4. Module *properties* (parameters) may be set during module selection as well as being modifiable in the display panel once attached to the model. Once a module is added, it appears as children under the corresponding domain in the model specification tree, as seen in Figure 6.4 (C).

Specific modules defined in a model can be composed together through the use of their unique ID (as described in Chapter 5). For example when defining a bacterial species, the value which goes into its morphology property entry is the unique ID of the corresponding morphology module. An example of this can be seen in Figure 6.5 (A), with an inset (B) showing the model structure this encodes for.

To facilitate this binding process and make self evident where it can be done, all *properties* which can be bound have a drop down menu allowing for the selection of any valid modules which exist in the model. An example can be seen in 6.5 (C). Some other properties may be constrained to specific values, for which a drop down menu is also implemented, as can be seen in 6.5 (D). If a property should point to a file, then a *browse* button is rendered next to that parameter, as seen in Figure 6.5 (E).

Model analysis can be conducted by attaching data collectors to the model, the user may choose which data they gather for the model. Parameter sensitivity

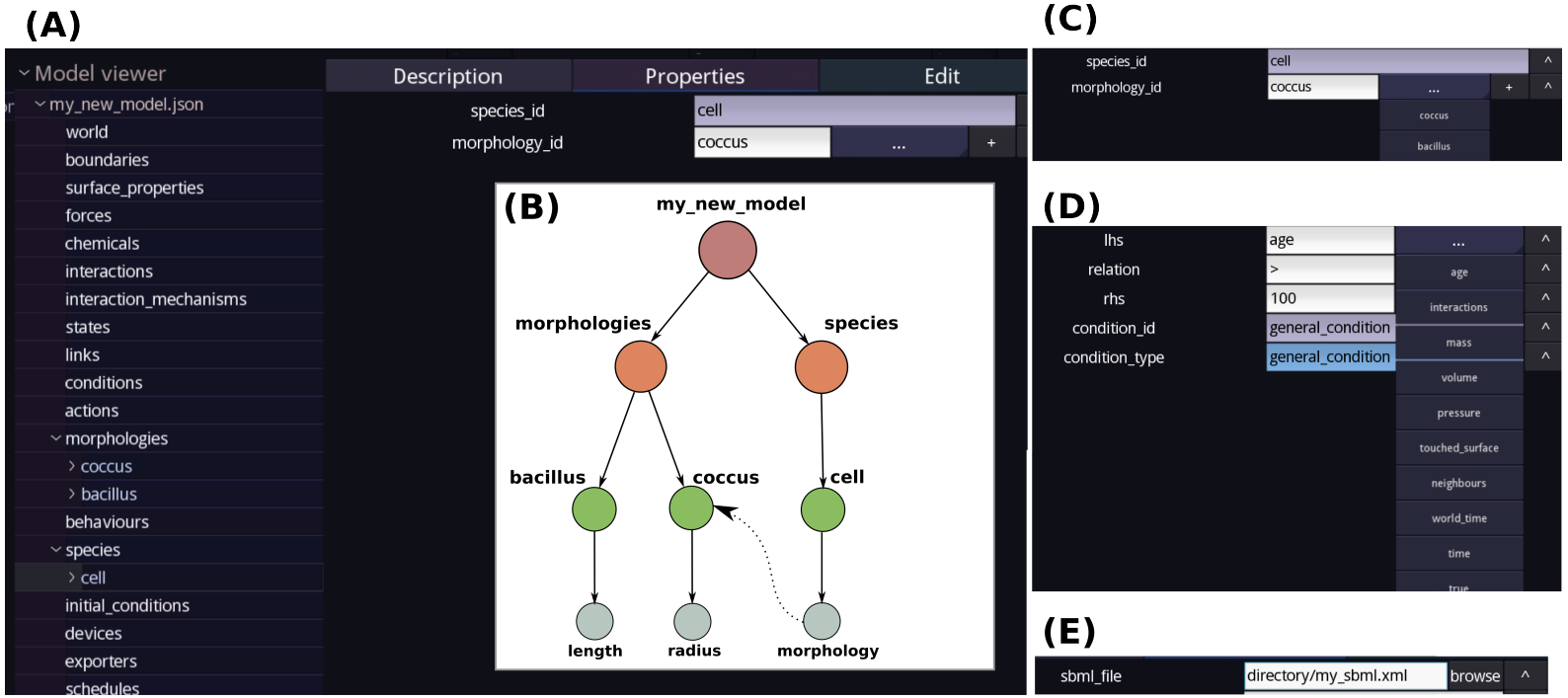


Figure 6.5: Binding model properties in Easybiotics. **(A)** Screenshot showing a model where a binding between the *morphology* property of a *cell* species module is made to the *coccus* morphology module. **(B)** A schematic showing the corresponding model tree representation of the model, showing the binding with a dashed arrow. **(C)** The binding of properties is facilitated by a drop down menu which shows all available bindings in the model. **(D)** Some properties don't bind to other modules, but rather must be one of a set range of values, in these instances a drop down menu is also provided. **(E)** Certain properties bind to an external file, for example when integrating an SBML into the model, in these cases a browse button is provided to navigate to the file.

can be characterised by running parameter sweeps, which involves running the model under a range of parameters to observe how system dynamics change. Easybiotics also provides functionality to plot live graphs directly from the data collectors attached to the model.

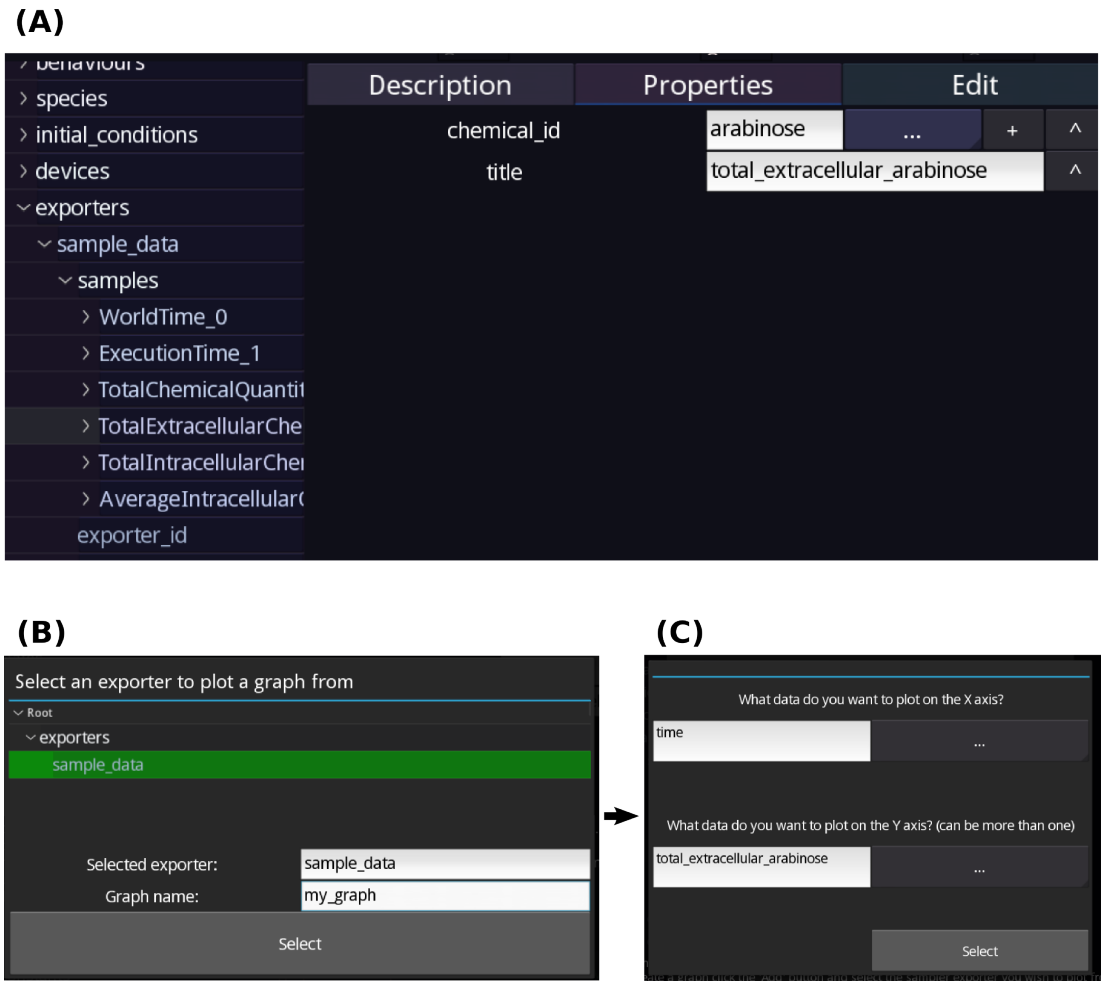


Figure 6.6: Screenshots of Easybiotics showing the process to create real-time graphs of data collected from a model simulation. **(A)** A *sampler* module called *"sample_ data"* is attached to the *exporters* domain of the model, and has numerous samples such as recording the simulation time and chemical quantities in the intracellular and extracellular space. **(B)** Selecting to add a create a graph from the file bar loads a pop-up box prompting the user to select one of their defined exporter modules. **(C)** When an exporter is selected the user is prompted to select which samples to plot on the X and Y, facilitated by drop-down boxes which are automatically populated with the defined samples.

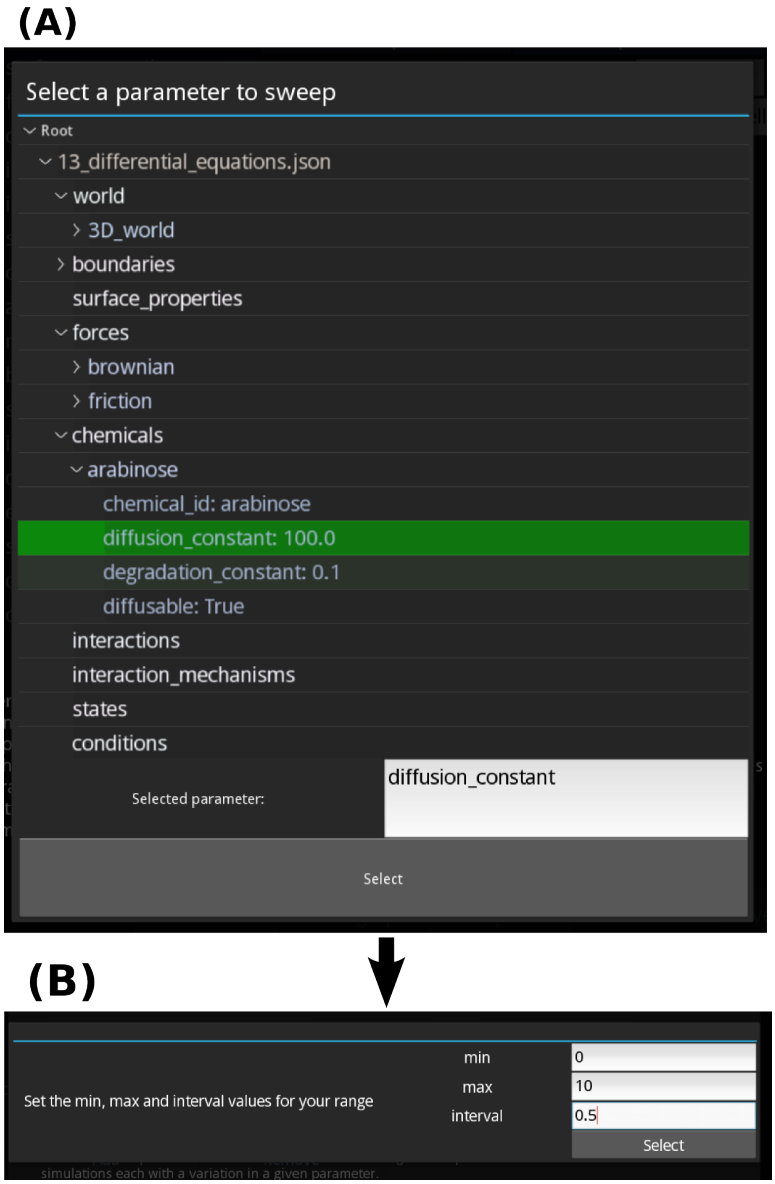


Figure 6.7: Screenshots show the process to create parameter sensitivity analysis tests in Easybiotics, called *parameter sweeps*. **(A)** The user is prompted to select an iterable parameter of the simulation (currently only properties which are of a *number* type are valid to sweep). **(B)** Once a valid parameter is selected then the user is prompted to enter the range of values for the parameter. As an alternative to entering a *min*, *max* and *interval* for the parameter values, they may specify a list of values.

6.2.2 Data collection and live graph plotting

Easybiotics provides functionality to use the analysis tools in Simbiotics, and allows for live graph plotting from the data files that Simbiotics exports. There are many properties of a model which can be measured, such as biomass, spatial chemical distributions and gene regulation activity. The model specification allows for data exporters to be attached to the simulation. Numerous exporters can be defined, where each collects specific data about the simulation and writes it to file. Graphs may then be defined, which are set to plot data from the exporters. Graphs can be saved and loaded to/from file for easy reuse, they are separate to the model specification. To run real time graphs during the simulation run, select the 'Run - Run with live graph plotting' option from the file bar, and specifying the graphs which are to be rendered.

6.2.3 Parameter sensitivity analysis

Population models are often complex with many parameters and potential system states. In order to characterise systems it is often useful to simulate a model with changes in one or a small number of parameters. Easybiotics provides a feature to support this type of analysis, called *parameter sweeps*. The model is run for each of the defined values in the parameter sweep range, and the results are stored in separate folders (along with a copy of the model file and parameters that it was run with). Many parameter sweeps can be defined, where each sweep is a specific model parameter and a list of values for it to take. Parameter sweeps can be run independently, where each parameter is explored with the default value for all other parameters. Alternatively they can be run exhaustively, where all combinations of parameters are simulated. Live graph plotters may also be attached to parameter sweeps, plotting the data from all simulations on one graph for easy comparison. Similar to graphs, parameter sweep objects can be defined and saved to a separate file.

Managing and analysing data output from many simulations is currently assisted by a set of Python scripts. The scripts allow for the extraction of specific data from a set of different simulation runs. For example, when doing parameter sensitivity analysis, it is often necessary to obtain the same set of data from the

different results folders in order to compare them.

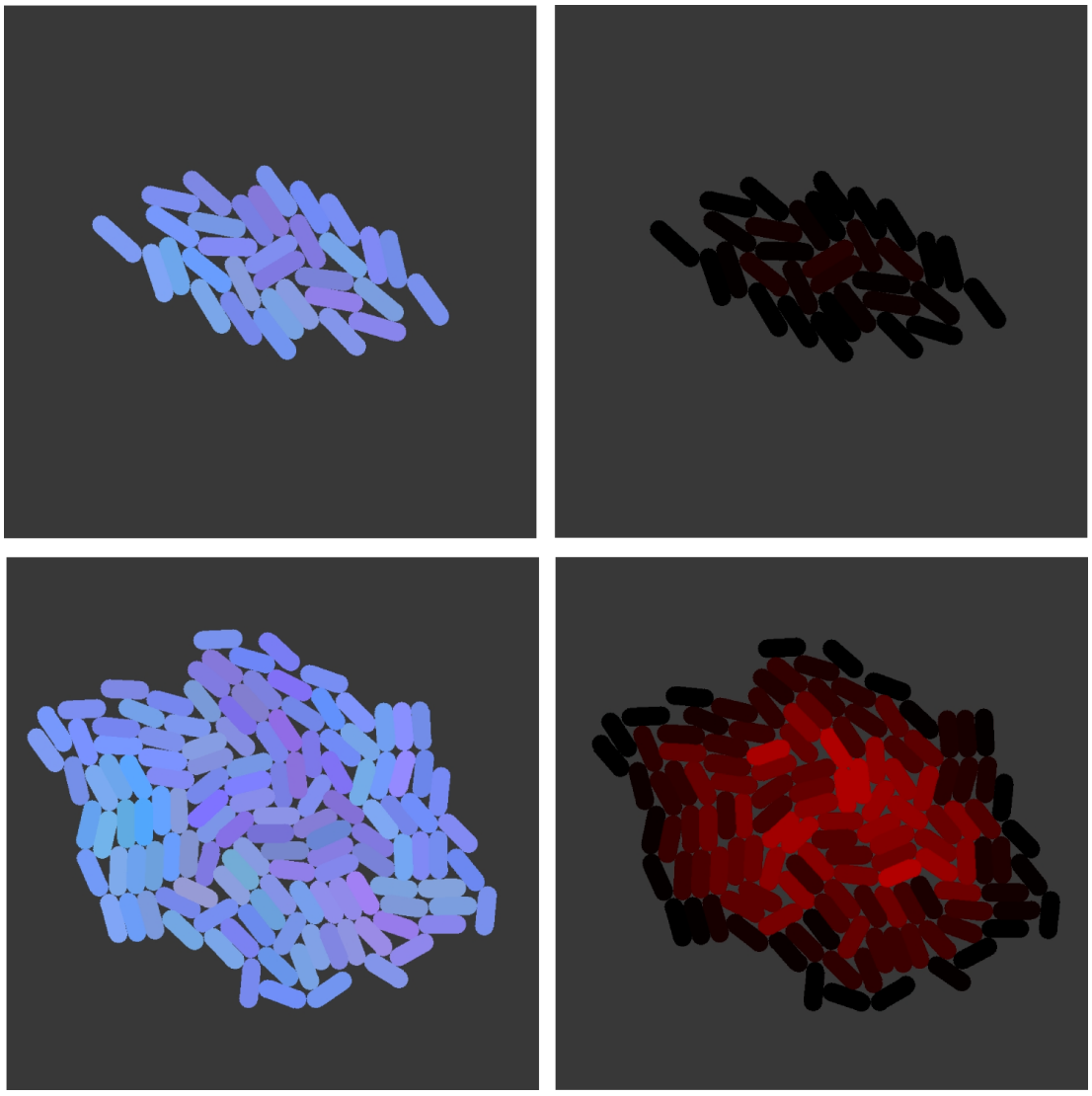


Figure 6.8: Post-simulation visualisations of a growing colony of bacillus cells. **(A)** Rendering of cells where the colours show the lineage of cell lines, as upon mitosis one of the daughter cells has a small mutation in colour, exposing growth patterns. **(B)** Rendering of cells where the colours show the physical pressure the cell experiences, with black being low pressure and red being high.

6.2.4 Visualisation of models

Easybiotics provides functionality to render visualisations of simulations after they have finished executing, as an alternative to a real-time visualisation. This can be achieved by attaching a certain type of exporter, called a *geometry_imager*, to the model specification. This exporter writes all geometry properties to a file periodically, writing a new file for each time point. It produces a series of indexed files which can be found in the results folder you set for the exporter.

Each geometry image file can be rendered independently into an static 3D scene, which is loaded in the Simbiotics GUI allowing the user to move the camera around the scene and modify which properties are visualised. An example can be seen in Figure 6.8, where a growing colony can also be visualised by the pressure cells experience.

A sequence of geometry images can also be loaded into an animated 3D scene. The user may set the delay between the animation frames, and whether the renderer should skip indexes of geometry image files. The animation renderer also runs a camera to record the animated 3D scene and convert it into an *.avi* file.

6.3 Rapid model prototyping

The features provided by Easybiotics allow for the rapid prototyping of models, providing a similar platform as any CAD tools in traditional engineering domains offer. Models can be constructed, run and analysed without technical (programming) challenges, models can be saved and modified in a different version, giving freedom to explore system dynamics and try alternative methods of simulating particular model features. Small and medium sized models can be run on laptops and desktop machines, however a HPC is often needed for large simulations (100,000+ cells).

To exemplify the Easybiotics environment Figure 6.9 shows a running simulation with live graph plots of data the simulation is generating. The system consists of a species of synthetic bacteria that have the metabolic pathways

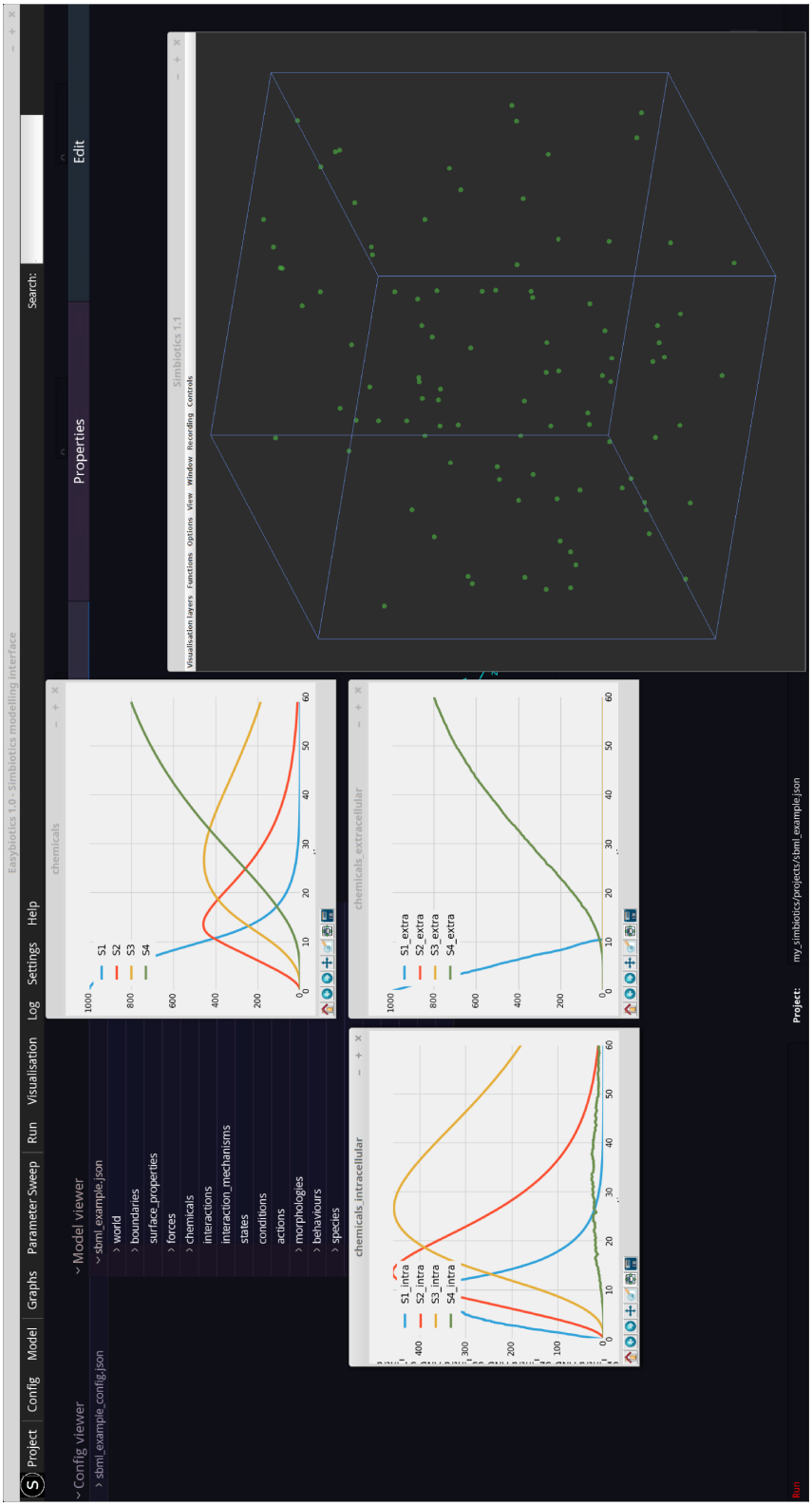


Figure 6.9: A screenshot of model running in Easybiotics - showing the model in the background, with four pop-out windows: The Simbiotics simulation window on the right, and three graph windows on the left. The top graph shows the total quantity of the chemical species in the simulation, the bottom left graph shows the total intracellular amounts of the chemicals, and the bottom right graph shows the total extracellular amounts of the chemicals. One can see that the extracellular chemical $S1$ is up-taken by the cells, and the product $S4$ is secreted by them.

$$\text{S1} \xrightarrow{k_1} \text{S2} \tag{6.1}$$
$$\text{S2} \xrightarrow{k_2} \text{S3} \tag{6.2}$$
$$\text{S3} \xrightarrow{k_3} \text{S4} \tag{6.3}$$

where $k_1 = 0.2$, $k_2 = 0.1$ and $k_3 = 0.05$. The cells are free-floating in a cubic domain as if it were the fluid phase. The domain is filled with $S1$, and the cells have an active membrane transport mechanism to uptake $S1$ into the cell, where it is involved in the reactions stated above. The product $S4$ is then secreted by the cell via an active transport mechanism.

The interface Easybiotics allows for the building of the model relatively simply; the metabolic pathways were defined in an SBML file using *Copasi* [87]. Alternatively these could have been defined as a set of ODEs or a Gillespie model, both possible representations of which can be seen in Figure 6.10 (A) and (B), the membrane transport definitions in (C), the initial conditions in (D), and the data exporter in (E). Upon hitting *Run* the live Simbiotics simulation viewer shows as an individual window, as the graphs defined from the data exporters (as seen in the *Data collection and live graph plotting section* above) display as individual windows for each graph (as seen in Figure 6.9).

Once a specific design has been achieved and prototyped in small models, the modeller can turn off the live visualisation and graph plotting and run a much large simulation by modifying the model parameters. The generated files from the data exporters can then be used for post-simulation graph plotting and analysis. Furthermore, parameter sweeps could be defined defined, which automates the running of models with different parameters - this allows the modeller to schedule a series of *in-silico* experiments.

Further elaboration of how Easybiotics can be used to rapidly prototype and explore models can be found in the Easybiotics user guide model building Tutorials, found in Appendix C.

Figure 6.10: **(A)** An example of how the same metabolic pathways could be modelling using a Gillespie model - ComplexReaction_0 is selected and its properties (parameters) can be seen on the right, where the reactants, products and rate can be set. **(B)** An example of how the same metabolic pathways could be modelled using ODEs - DifferentEquation_0 is selected and its properties displayed, where an equation describing the change in $S2$ is defined. **(C)** The membrane transport mechanism in the model - MembraneFlux_0 is selected showing it models a non-osmotic flux of $S1$ going into the cell, and the flux is picked from a Poisson distribution. **(D)** The initial conditions to the model - initial_population is selected, with its properties showing it defines 100 instances of the *cell* species.

6.4 Summary

Research in the Synthetic Biology domain is growing [193], and as more researcher engage in developing novel biological devices, the formalisation of design and testing methods becomes ever more important. Developing a robust synthetic system typically requires multiple iterations around the specify->design->build->test cycle to meet specifications. This process is laborious and expensive for both the computational and laboratory aspects, hence any improvement in any of the workflow steps would be welcomed.

In this spirit I have developed Easybiotics, an abstraction-layer for hiding model implementation details from model building and analysis. It allows for full access of Simbiotics functionality through a loosely-coupled interface ensuring maintainability of the software. Easybiotics provides an intuitive graphical environment for the rapid-prototyping of multicellular models, and gives the user access to features for characterising those models through data collection, graph plotting and parameter sensitivity analysis. The platform (in conjunction with Simbiotics) can act as a virtual lab that can be used to develop, store, run and analyse *in-silico* experiments - offering a means by which these experiments can be reliability reproduced on different people's computers.

With the push for computer-aided design in Synthetic Biology [33, 69, 232], and development of numerous tools for designing and simulating genetic circuits and metabolic pathways [32, 116, 136], we hope Easybiotics provides a suitable platform for assisting in modelling distributed multicellular biodevices.

Part II

Part II Abstract

In this part of the thesis I present four case studies conducted with Simbiotics. The case studies were selected to study the influence of physical and chemical interactions independently, exercising the modelling features of Simbiotics in discrete stages.

The first case study investigates the influence of cell-cell physical interactions, and the second study investigates both cell-cell and cell-substratum interactions coupled with cell growth. The third case study focuses on chemical-signalling between cells in a static spatial domain, and the fourth study investigates how synthetic chemical-signalling can be harnessed to form spatial patterns of gene regulation in immotile cell populations.

To more clearly delineate the library modules that were used in these four case studies (and used in the other models presented in this thesis), a graphical depiction can be seen below in Figure 6.11. This provides a key depicting the set of model components used in all of the studies (this is not the exhaustive list of Simbiotics features), and they are then shown to be present or not for each of the models.

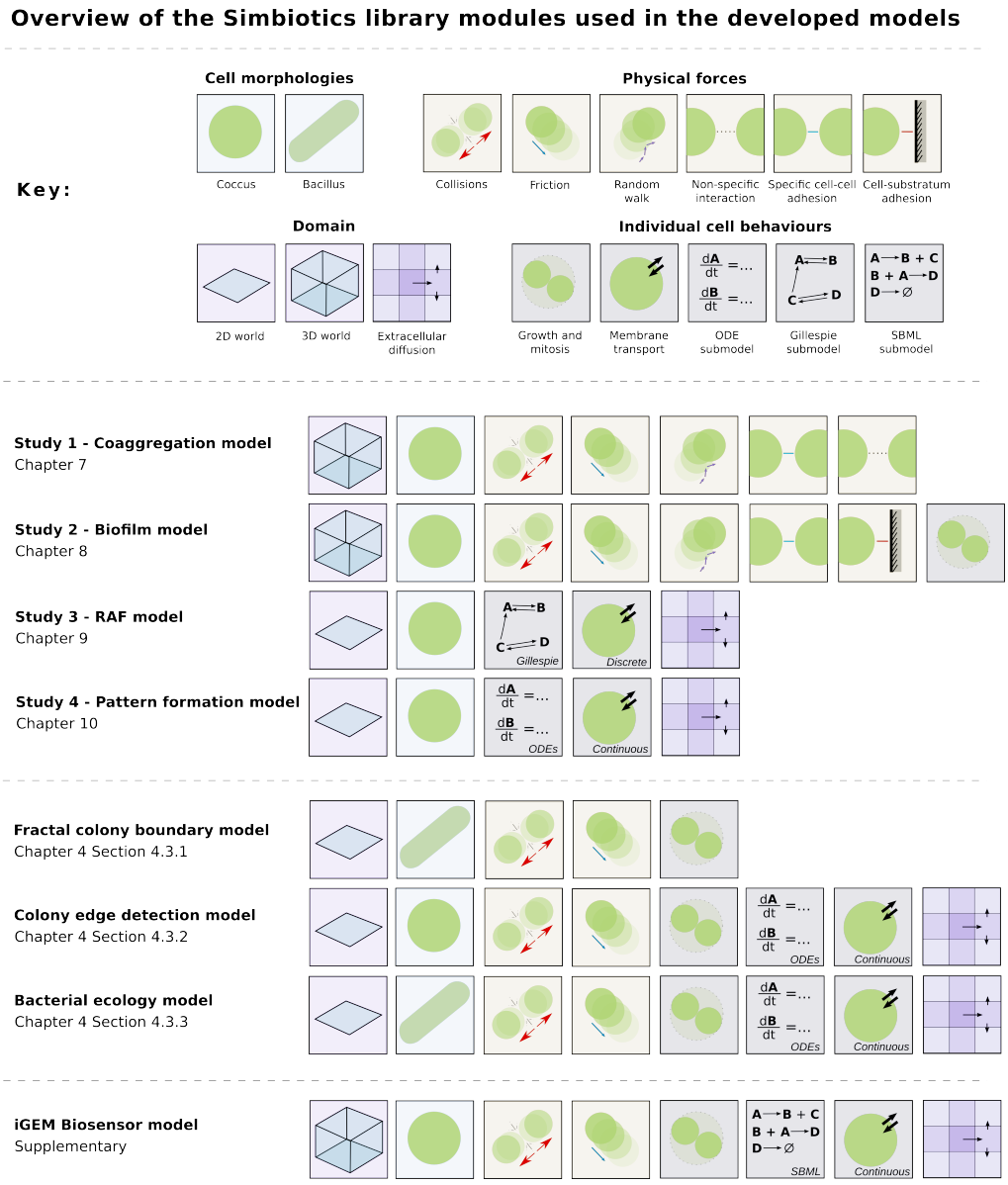


Figure 6.11: An overview of the library modules used in the *Simbiotics* models. Each study required the representation and integration of different processes. Cell morphology modules have a blue background, physical force component modules have a yellow background, single cell dynamics have a grey background, and extracellular diffusion is shown as purple. Note that the *membrane transport* module is used in its discrete form in Study 3, its continuous form in Study 4. For simplicity most icons are drawn with a coccus bacteria, though the processes are not exclusive to these geometries.

Chapter 7

Study 1 - Dental plaque: Bacterial coaggregation

In this chapter we present a study on coaggregation of two bacterial species involved in dental plaque formation. The influence of physical interactions mediated by receptor-adhesin, electrostatic and van der Waals forces on aggregation rate are considered. This study was done in collaboration with Waleed Mohammed and Nick Jakubovics from the School of Dentistry at Newcastle University with Waleed doing the experiments. The chapter addressed the first part of Objective 3 - Studying the effect of physical shoving and cell-cell adhesion on bacterial coaggregation and biofilm formation. The model and results of this chapter have been published: J. Naylor, H. Fellermann, Y. Ding, W.K. Mohammed, N.S. Jakubovics, F. Dafhnis-Calas , S. Heeb, M. Camara, J. Mukherjee, C.A. Biggs, P.C. Wright, N. Krasnogor **Simbiotics: A Multiscale Integrative Platform for 3D Modeling of Bacterial Populations** in *ACS Synthetic Biology*, 6(7):1194-1210, July 2017.

7.1 Overview

This case study was motivated by the investigation of purely physical interactions on the aggregation of bacteria in the fluid phase. This developed model involved simulating only physical processes, allowing for the physical modelling compo-

nents in the Simbiotics library to be verified with experimental data generated by our collaborators at the School of Dentistry.

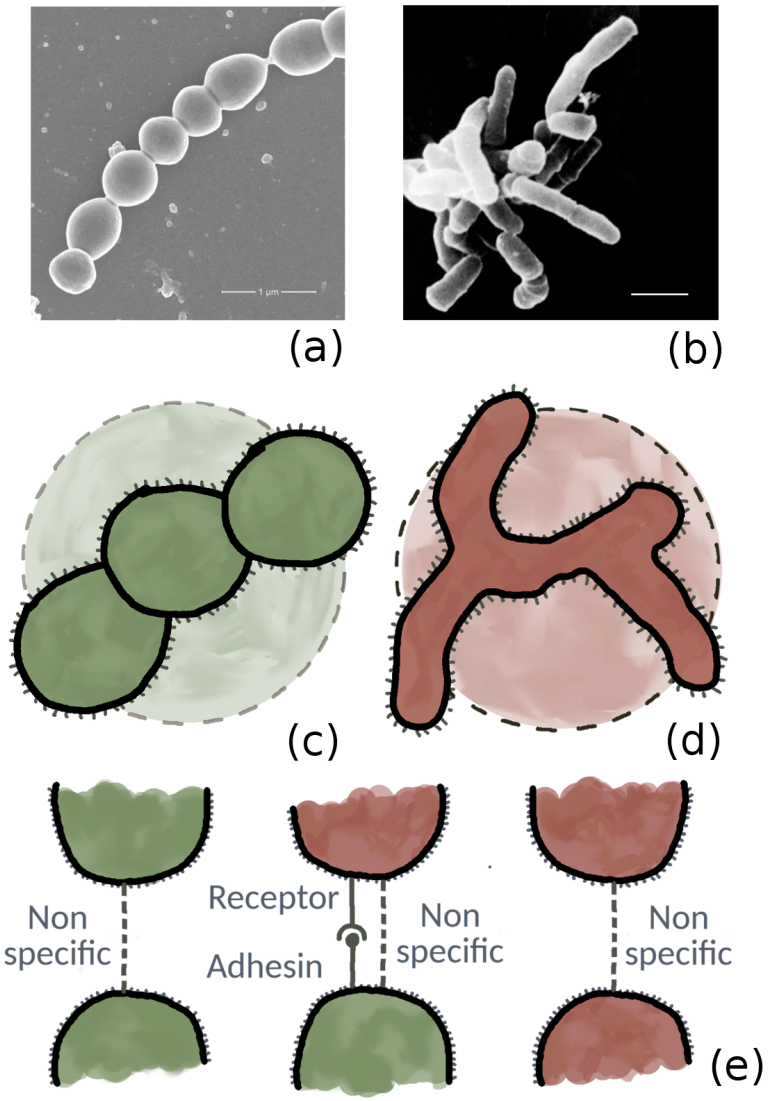


Figure 7.1: Microscopy images showing *Streptococcus gordonii* (a) and *Actinomyces oris* (b) where the scale bar is $1\mu m$. (c, d) *S. gordonii* and *A. oris* are modelled as spheres. (e) Depiction of the modelled interactions between *S. gordonii* and *A. oris*.

7.2 Introduction

Dental plaque is a biofilm caused by colonisation of the teeth by oral bacteria [149]. The coaggregation of bacteria plays a part in plaque formation [176], where bacteria of different species physical adhere to each other via specific molecules, leading to gene regulatory changes in the participant cells [155]. Understanding the mechanisms and dynamics of coaggregation can allow for the development of new dental hygiene products for disrupting plaque formation [2, 138, 223].

The process of coaggregation is heavily dictated by the surface characteristics of bacterial cells. Cell surface charge effects the strength of van der Waals and electrostatic forces between cells, referred to as non-specific interactions. Surface adhesins and receptors may also be present, which undergo specific interactions if they have the appropriate structure to form an adhesive bond - this bond has a key-lock mechanism [29, 186].

We consider the influence of these specific and non-specific interactions on the aggregation of two bacteria found in the mouth, *Streptococcus gordonii* and *Actinomyces oris*, that have a matching adhesin and receptor pair [14] (depicted in Figure 7.1). We are interested in investigating the driving forces behind aggregation, analysing aggregation dynamics for different specific and non-specific interactions. The system was studied experimentally and with a computational model built in Simbiotics.

7.3 Methods

7.3.1 Experimental

To isolate the process of surface-mediated interactions without metabolic behaviour, the cells were initially washed in sodium azide, such that their biological activity is ceased but their physical properties were preserved. Through this we ensure the system is of minimal complexity, isolating the process of cell-to-cell adhesion without biological activity or active motility. The aggregation of bacteria in a cuvette of 1mL solution was measured by following changes in optical density. We started with a well-mixed population and use a spectrophotometer to

obtain a time series of OD 600 measurements. As aggregates formed the optical density of the population decreased as more light could pass through the cuvette.

7.3.2 Model

To capture the core features of the system, we decompose it into its core *components* and *processes*. Components are considered to be the definable objects of the system, and processes are the mechanisms through which they interact.

Components

1. 3D Fluid Environment
The fluid medium is represented as a cubic domain in which the cells can move around. All faces of the cube are periodic, such that when a cell exits the cube via one of these faces it enters through the opposing face.

2. Cells (S. gordonii* and *A. oris)
S. gordonii and *A. oris* cells are represented as rigid-body spheres.

3. Surface adhesins and receptors
Surface adhesins and receptors are represented implicitly on the surface on bacterial cells, stored in its properties.

Processes

1. Specific adhesin and receptor interactions
Specific interactions between adhesins and receptors are defined, where each interaction has an associated rate at which two physically contacting cells have a specific interaction, and an associated force. These interactions are modelled as a spring that forms between two cells.

2. Non-specific electrostatic/van der Waal interactions
Non-specific interactions between cells such a electrostatic repulsion and van der Waals forces are modelled as forces applied on neighbouring cells that are within a close proximity.

3. Random motion of fluid

The random motion of fluid acting on a cell is modelled by two force components. The first being a random force exerted on a particle at each point in time, the second being the viscous drag force a cell experiences as it moves through the fluid medium.

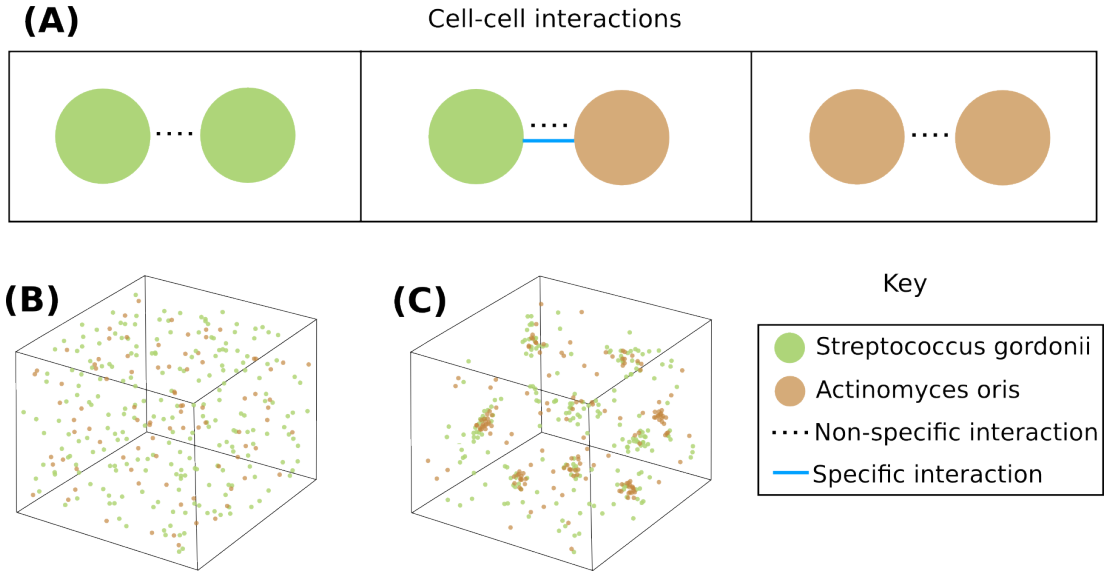


Figure 7.2: **(A)** The cell interactions in the model. *Left and Right* show the non-specific interactions between cells of the same species *Center* shows the additional specific interaction between the two species. **(B)** The initial condition of the coaggregation model - a well mixed balanced population of the two species. **(C)** An illustration of the coaggregated cells.

Model description

Bacterial cells are modelled as spheres with surface properties. Each cell performs a random-walk due to the effect of Brownian motion causing the population to mix. Cells have an extended sphere of influence to represent their surface charge effects, these are modelled as non-specific interactions as described in the Chapter 3. *S. gordonii* cells have adhesins on their surface and *A. oris* have a matching receptor, an interaction between the two is modelled as a specific interaction as described in Chapter 3.

As the main parameters to the simulation we consider the strength of non-specific interactions due to surface charge, K_{E}. We also take the probability that two colliding cells with a matching receptor-adhesin will interact P_{S}, representing the density of adhesins and receptors on the cell surfaces. Further more we consider the strength of an adhesin-receptor interaction K_{S}.

We start with a well mixed population of individual bacteria and use the simulated spectrophotometer as described in the Simbiotics Analysis section to obtain a time-series of optical density measurements. Figure 7.2 further illustrates the model.

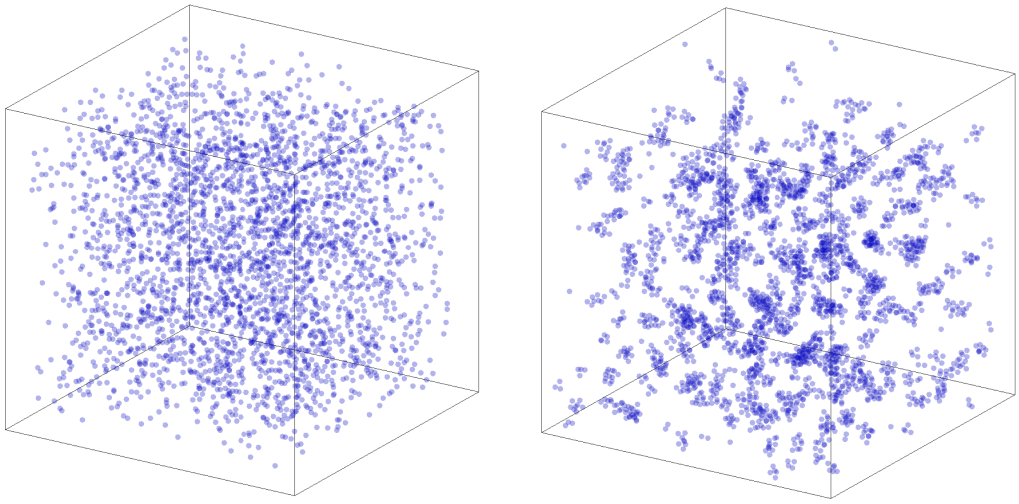


Figure 7.3: **Left:** Initial population of simulated *S. gordonii*. **Right:** Snapshot of simulated *S. gordonii* showing aggregation.

7.4 Results

To understand the dynamics of coaggregation we first isolate the processes of non-specific mediated mono-aggregation. After this we consider the independent effect of a specific interaction between two species. The third case is the combined model of non-specific and specific interactions. Finally we consider the effect of cell population density on the system, performing experiments and simulations

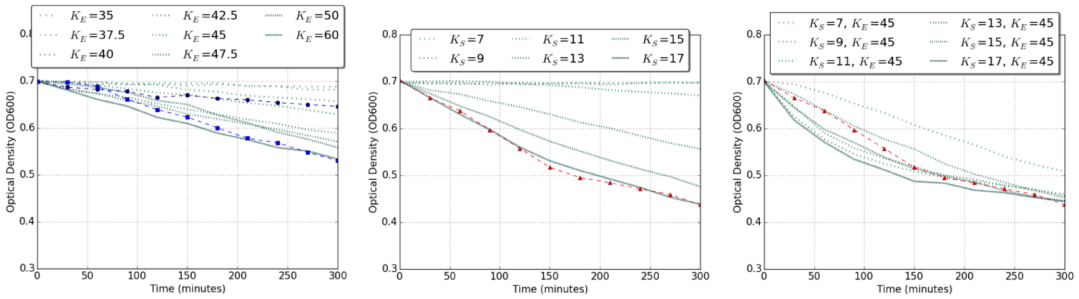


Figure 7.4: Simulated and experimental optical density measurements showing aggregation. Dashed lines are experimental results, solid lines are simulated. (a) Simulated aggregation due to non-specific interactions with different force constant K_{E} values, compared to experimental optical density curves for single population aggregation. (b) Simulated aggregation due to specific receptor-adhesin interactions with different force constants K_{S} values, compared to experimental optical density curve of mixed population coaggregation. (c) Simulated aggregation due to both non-specific and specific interactions with different force constants $K_{\mathrm{S/E}}$ values, compared to experimental optical density curve of mixed population coaggregation.

Model feature	Parameter	Symbol	Value	Unit
Sphere	*S. gordonii* cell radius	r_{gordonii}	0.5	μm
Sphere	*A. oris* cell radius	r_{oris}	0.5	μm
Brownian motion	Force constant	K_{R}	2.2	$\frac{\mu m}{cs^{\frac{3}{2}}}$
Friction	Force constant	K_{F}	2.0	$\frac{\mu g}{cs}$
Gravity	Force constant	K_{G}	0.0002	$\frac{\mu m}{cs^2}$
Non-specific interactions	Force constant	K_{E}	25-50	$\mu g \frac{\mu m^3}{cs^2}$
	Range	r_{E}	3.0	$\frac{interactions}{cs}$
Specific interactions	Force constant	K_{S}	6-7	$\frac{\mu g}{cs^2}$
	Probability	P_{S}	0.1-10	$\frac{interactions}{cs}$

Table 7.1: Model features and their parameters for the basic coaggregation case study model

of mono and coaggregation at 3 different initial cell densities, 0.5x, 2.0x and 4.0x that of the original system.

Analysis of non-specific interactions involved changing the force bacteria exert on each other capturing different surface charges. Figure 7.4 (a) shows optical

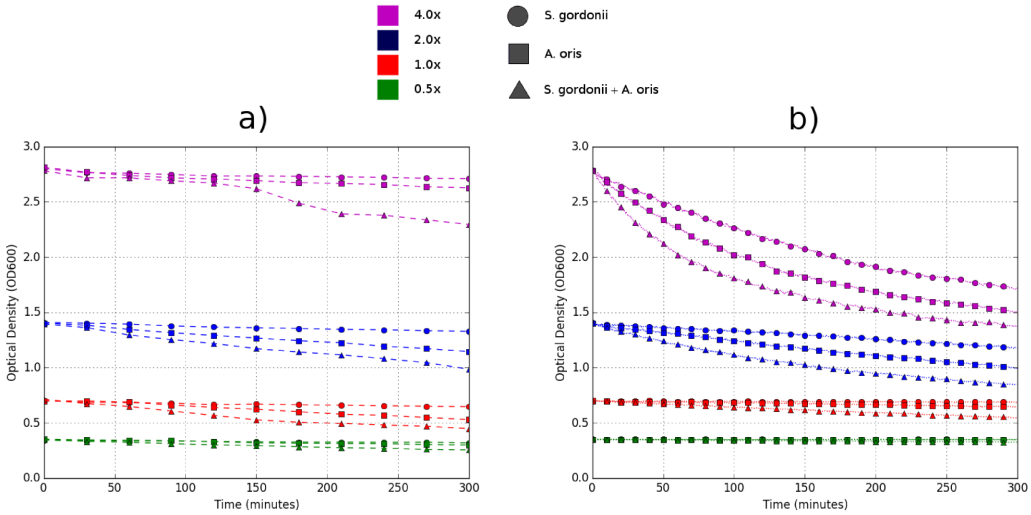


Figure 7.5: Simulated and experimental optical density measurements showing aggregation. Dashed lines are experimental results, solid lines are simulated. (a) Experimental optical density curves of two mono-aggregating and one coaggregating population. The three curves are shown for each density (from top to bottom) 4.0x, 2.0x, 1.0x and 0.5x. (b) Simulated optical density curves of aggregation curves for *A. oris* and *S. gordonii* aggregation independently, and one curve for coaggregation of a mixed population. The three curves are shown for each density (from top to bottom) 4.0x, 2.0x, 1.0x and 0.5x.

density measurements for both the experimental and simulated tests. The experimental curves show the single-species aggregation behaviour of *S. gordonii* and *A. oris* on their own. As the non-specific interaction force constant K_{E} is increased the rate at which aggregates form increases, however it saturates at values of $K_{\mathrm{E}} > 50$. With high force constant values regularly sized aggregates typically form as seen in Figure 7.6 (b), low force constant values lead to irregular aggregation at around $K_{\mathrm{E}} = 35$ as seen in Figure 7.6 (d). Aggregation does not occur when $K_{\mathrm{E}} \leq 30$, this is due to the attractive electrostatic force not being sufficient to prevent Brownian motion from causing the cells to dissociate.

Analysis of additional specific interactions involved changing the probability P_{S} at which two colliding bacteria with matching receptor and adhesin will interact specifically, and the strength K_{S} of this interaction. Figure 7.4 (b) shows

optical density measurements for experimental and simulated coaggregation. Experimental (dashed lines) show coaggregation of a mixed *S. gordonii* and *A. oris* population. Simulated (solid lines) show coaggregation optical density measurements, a parameter sweep over K_S and P_S was performed. Figure 7.4 (c) shows the same experimental results with the coaggregation results of the combined

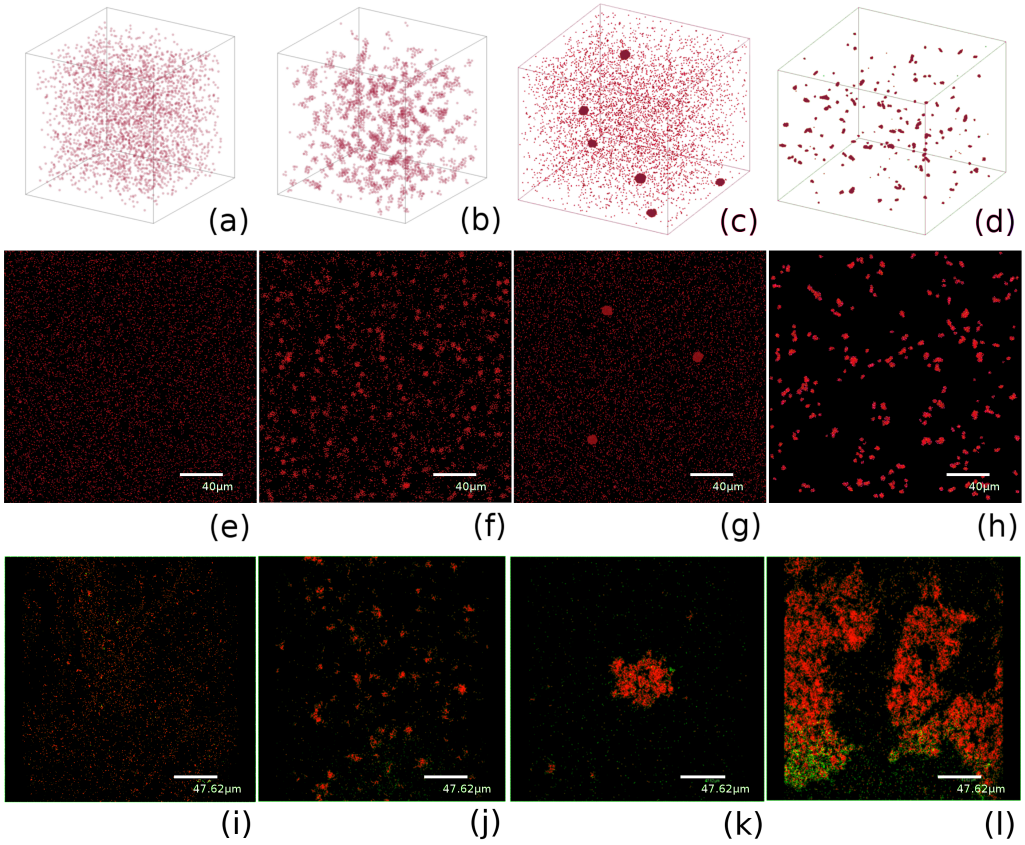


Figure 7.6: Simulation snapshots and microscopy images showing aggregation in both simulated and actual system. In microscopy images *A. oris* are red and *S. gordonii* are green. (a) Initially well-mixed population of simulated cells (both *A. o* and *S. g*). (b) Uniformly distributed aggregates after 4 simulated hours. (c) Large aggregates among many un-bonded cells. (d) Asymmetrically distributed aggregate sizes with few un-bonded cells. (e-h) 2D projections of cross-sections taken from (a-d). (i) Microscopy image of experiment showing an initially well-mixed population. (j, k, l) Microscopy images showing different aggregation structures. All images show the microscopy/simulation after 4 hours.

specific and non-specific interaction model. One can see the aggregation rates from purely specific interactions as seen in Figure 5 (b) are enhanced by the presence of non-specific interactions as seen in Figure 7.4 (c). An explanation for why this may occur is due to non-specific interactions having an extended sphere of influence, interacting with neighbours which aren't in direct contact, whereas specific interactions only occur for cells which are in physical contact.

Strong specific interactions typically lead to uniform aggregate sizes similar to non-specific interactions, however variations of low K_S and P_S values lead to irregular aggregation as seen in Figure 7.6 (c) and (d). The reason for this may be due to the fact that single cell-cell interactions easily dissociate due to Brownian motion, however if small aggregates persist, other cells may join to form a larger aggregate. The presence of additional cells creates a network of specific interactions between neighbouring cells in the aggregate. A synergistic effect occurs where each cell has multiple interactions, stabilising the aggregate and leading to a few large aggregates forming in a generally well-mixed population. When interactions are strong enough, cells do not dissociate once they adhere to each other. In a well-mixed solution with uniformly distributed Brownian motion this leads to regularly sized aggregates.

Our model does not produce the large aggregate islands as seen in the microscopy images shown in Figure 7.6 (h). The reason for this may be due to additional forces present in the actual experiment such as hydrodynamics which we do not model explicitly.

Aggregation at varied population densities

To consider the effect of population density on aggregation experiments with *0.5x*, 2x and 4x the initial population size were performed. Figure 7.5 (a) shows experimental results, and (b) shows simulation results.

We find the aggregation rate of bacteria is proportional to the population density. This can be explained by the mean free path that a cell travels before interacting with another cell decreasing as density increases, therefore a higher probability of a physical adhesion as density increases. An additional mechanism is that large clusters tend to sink faster, as the combined motions of its constituent cells cancel, leading to a stronger effect of gravity. The sinking of aggregates leads

to a decrease in the OD_{600} reading.

We find that aggregation rate is more sensitive to population density in the simulation, however overall simulation results show strong qualitative trends with the experimental findings. We note that spectrophotometry is a generalised technique which has been applied to the measurement of aggregation, and due to the nature of the method it is not a perfect deduction of aggregate formation.

7.5 Discussion of Simbiotics

This study heavily informed the design of Simbiotics as it coincided with the early development of the software. The requirements of the software were based on our collaborator's needs, ensuring the platform could simulate the relevant features of their system. The model needed to simulate bacterial adhesion, however processes such as hydrodynamics were not required due to the questions they wished to ask the model. The features relevant to the study were implemented as modules in the software, such as modules describing two types of cell adhesion (specific and non-specific) and another module describing passive motility of cells (random walk).

This study involved developing a variety of models, some using different feature modules than others (for example the coaggregation models required the addition of a specific interaction module). Additionally, each of those models had numerous parameters that were to be explored. Due to this is was clear that some form of file format was required to represent models, and that the platform had to be able to run a given model with many different parameter values.

The library of modelling features proved very powerful for building these different models, allowing for the easy design, composition and modification of each system. Additionally having the model specifications written to text file allowed them to easily be used as the input to Simbiotics on the high-performance computing cluster.

7.6 Summary

This case study has investigated the influence of physical interactions on bacterial coaggregation in the fluid phase. The effect of specific and non-specific interactions have been characterised individually and in combination, both experimentally and with a computational model built in Simbiotics. A model was developed to reproduce the experimental behaviour, and then to characterise the influence of the relevant parameters. Model findings show that the influence of non-specific electrostatic interactions accelerate the rate of aggregation due to their extended sphere of influence, and that aggregation rate scales with population density due to the increased rate of cell physical interactions.

This study provides a basic model on which future investigations can be conducted. The model could be extended to include pellicle colonisation and cell growth, investigating how *S. gordonii* colonise may colonise a surface, allowing *A. oris* (which do not directly adhere to the pellicle) to be recruited to the plaque biofilm through the interactions with *S. gordonii*. Additionally, as our knowledge of the genes relating the adhesion expands, we can extend the model to investigate how physical adhesion feeds back to gene regulation [84, 139, 144]. This avenue of investigation may offer insights into how we can disrupt this process by developing advanced oral care products.

The work was done in collaboration with Newcastle School of Dentistry, with Waleed Mohammed conducting the experiments.

Chapter 8

Study 2 - Synthetic *E. coli* biofilms

In this chapter we present a study on biofilm formation conducted with Simbiotics. The influence of physical interactions between cells and substratum are investigated, studying how these lead to different biofilm architectures. This study was done in collaboration with Joy Mukherjee, Felix Dafhnis-Calas, Miguel Camara, Stephan Heeb, Catherine Biggs and Phillip Wright, with Joy conducting the experiments. This chapter further addresses Objective 3 - Studying the effect of physical shoving and cell-cell adhesion on bacterial coaggregation and biofilm formation. The model and results of this chapter have been published: J. Naylor, H. Fellermann, Y. Ding, W.K. Mohammed, N.S. Jakubovics, F. Dafhnis-Calas , S. Heeb, M. Camara, J. Mukherjee, C.A. Biggs, P.C. Wright, N. Krasnogor **Simbiotics: A Multiscale Integrative Platform for 3D Modeling of Bacterial Populations** in *ACS Synthetic Biology*, 6(7):1194-1210, July 2017.

8.1 Overview

This case study was motivated by the investigation of physical interactions on biofilm formation and development. This was a natural extension of Study 1, where only physical interactions between cells in the fluid phase were modelled. Here we coupled physical interactions with cell growth, modelling planktonic cells initially in the fluid phase which can attach to the substratum. The modelling components used for simulating bacterial growth could be further validated in

this study by comparison with experimental data generated by our collaborators.

8.2 Introduction

Biofilms pose a serious concern to public health due to the potential to cause infections and their resistance to antimicrobial agents [52]. Developing solutions to combat biofilms involve destroy existing biofilms, treating surfaces to prevent cell adhesion, and interruption of bacterial communication systems discoordinating biofilm formation [34, 57, 134, 140]. However, due to limitations in these approaches, novel solutions may be counter-intuitive, and involve the growth of synthetic 'friendly' biofilms, which act as intelligent surfaces to prevent infection. Development of synthetic biofilms with such properties is a dream of Synthetic Biology, and only the first steps toward the conception of such a system have been achieved. Synthetic biofilms have been grown to better understand the roles of biofilm components, relationships within the biofilm, and functions of biofilms [4, 203].

The more basic study of how biofilms can be controlled is investigated here. We consider the influence that cell surface properties have on biofilm formation and development, in the hope to better understand how these can be tailored to produce specific biofilm architectures. Three *Escherichia coli* mutants that form different biofilm architectures are investigated, and a model is constructed to study the potential driving forces behind these architectures.

8.3 Methods

8.3.1 Experimental

Three strains of *E. coli* are used in the experiments, *DH5-α*, a *csrA* strain with higher surface charge, and a *PgaA* strain with an even higher surface charge.

The strains were cultivated overnight for 16 hours in a 3ml Synthetic Urine media with the addition of 0.1% glucose [28] aerobically at $30°C$ and 120 rpm. Overnight grown cultures were then re-inoculated into fresh Synthetic Urine media (1:100 dilution) and $200\mu L$ was grown in a 96 well plate in the static condition

for 48 hours. The supernatant was then removed and its optical density was measured. The optical density of the biofilm formed on the surface was also measured by re-suspending the biofilm with the synthetic urine media, and the planktonic/biofilm ratio was considered. The biofilm was also imaged by staining the biofilm formed on these 96 well plates using the Live/Dead BacLight stain (ThermoFisher Scientific, UK) using a Leica SP2 confocal laser scanning microscope.

8.3.2 Model

We aim at developing a minimal model of the system, allowing for the dynamics emerging from physical interactions to be clear and well characterisable, rather than having a complex model for which the relationships are convoluted by many interacting processes.

To establish the features that should be present in the model, the experimental system is decomposed into key *components* and *processes*.

Components

1. 3D Fluid Environment with Substratum
The fluid medium is represented as a cubic domain. Horizontal faces (X and Z) are periodic, such that when a cell exits the cube via one of these faces it enters through the opposing face. The Y faces are solid, such that cells can not pass them. The base of the cube (plane at Y minimum) has adhesive structures on it that cells can adhere to.

***2.* E. coli**
E. coli cells are represented as rigid-body spheres.

3. Extracellular polymeric substances (EPS)
An implicit representation of EPS is used where loose springs which form between cells upon cell-to-cell adhesion

Processes

1. Cell motility

Cell motility is represented as a random force that each cell experiences at each time point. The magnitude of this force is significantly smaller for cells which are sessile than those which are planktonic, in order to capture the shift from flagellar to pili-mediated motility.

2. Cell-to-cell and cell-substratum adhesion

Cell-cell adhesion events are represented by springs which form at a given rate between bacterial cells that are physically contacting.

3. Cell growth and mitosis

Cell growth is represented by the spherical cells growing in mass (and thus volume). Mitosis occurs upon cells reaching twice their initial mass.

Model description

Individual cells metabolisms and gene regulation are not explicitly modelled. Rather, a high level 'brain' is implemented in each cell which computes whether the cell is in a planktonic or a sessile state. These two states toggle parameter values relating to the cell's behaviour, providing an implicit differentiation in the cell's metabolism and gene regulation, thus two phenotypes are present in the model.

PLANKTONIC

Cells are considered planktonic if they have no adhesion partners, or if their adhesion partner is also in the planktonic state. Characterised by their motility and slow growth rate, they experience large random motion forces, implicitly capturing their flagellar-mediated motility. Planktonic cells have a low probability that they will undergo cell-to-cell adhesion with another planktonic cell.

SESSILE

Cells are considered sessile if they have adhered to the substratum, or if one of

there adhesion partners is also in the sessile state. They are immotile with a high growth rate, experiencing small random motion forces emulating the twitching-motility mediated by pili. They have a high probability of undergoing cell-to-cell adhesion.

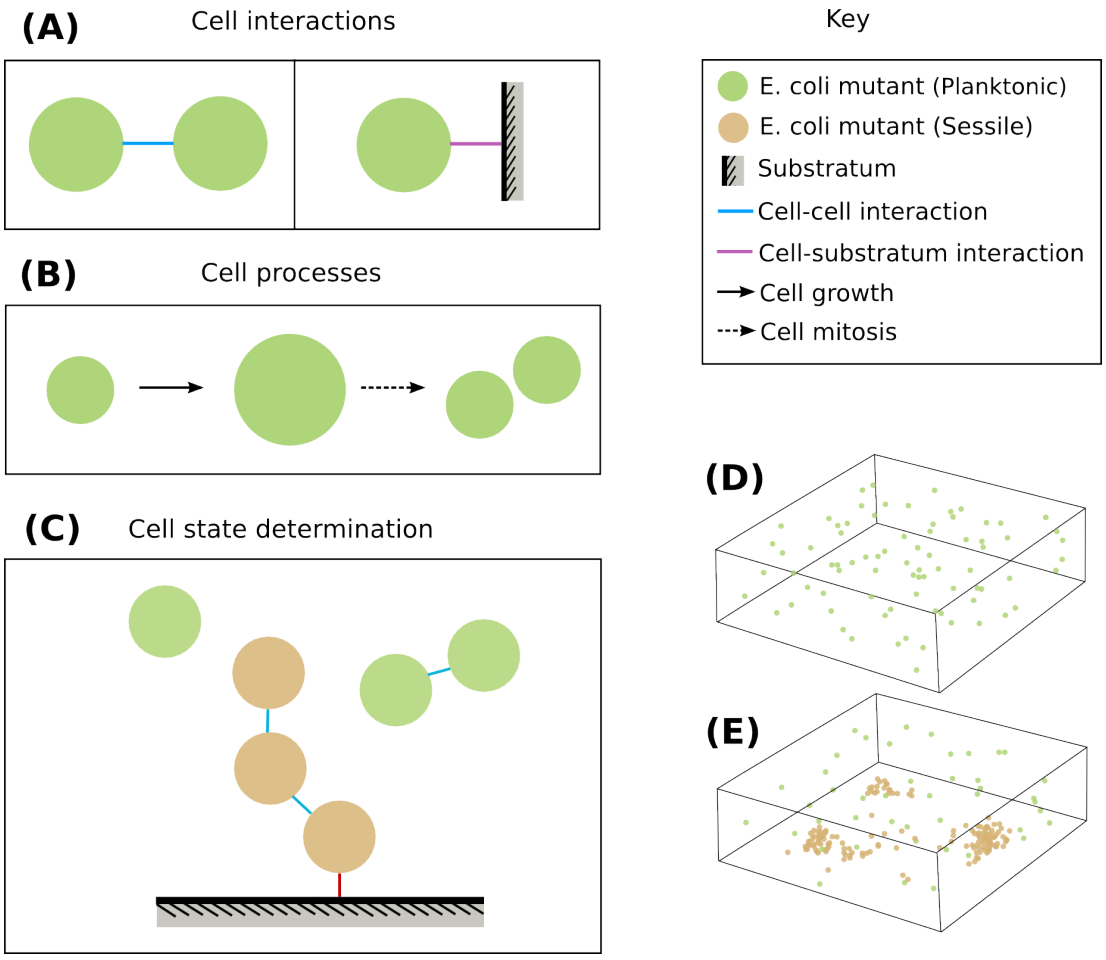


Figure 8.1: **(A)** The cell interactions in the model. *Left* shows the adhesion between two cells *Right* shows the adhesion between a cell and the substratum. **(B)** The cell processes in the model; a cell grows in biomass (modelled as sphere increasing in volume) and divides (undergoes mitosis) upon reaching twice it's original mass. **(C)** An illustration of how cell states are determined; cells which have adhered to the substratum, or part of a cell cluster that has adhered to the substratum at some point, are considered *sessile*. Cells which are still free-floating are *planktonic* (event if adhered to another planktonic cell). **(D)** The initial condition of the model - a well mixed population of cells. **(E)** An illustration of biofilm formation in the model, with cells adhering to the substratum.

Our model of biofilm development represents cell growth, cell motility, surface-mediated interactions and basic gene regulation. Cells are modelled as initially free-floating spheres, and experience Brownian motion with a force constant $K_r p$. Cells contacting the substratum may adhere to it with the rate P_s and an interaction strength K_s. Once attached to the substratum, cells experience a lower magnitude of Brownian motion, $K_r s$, and may adhere to other contacting cells at a rate P_c and an interaction strength K_c. These cells are then considered to be part of the biofilm and experience lower Brownian motion and may adhere to other cells. Cells grow at a rate G_r with growth dynamics as described in the Simbiotics modelling section. All model parameters can be seen in Table 8.1.

Initially in the planktonic state, a population of motile cells are free to move within the cubic world. Planktonic cells do not tend to adhere to one another. Upon colliding with the substratum, these cells adhere to the surface, shifting from the planktonic state to sessile. This process is reversible, and if the cell experiences enough force to pull it away from the surface, this cell will again become planktonic. Sessile cells have a higher growth rate, along with a higher propensity to undergo cell-to-cell adhesion, thus they proliferate forming communities of bacteria on the surface. An illustration of the model can be seen in Figure 8.1.

Model feature	Parameter	Symbol	Value	Unit
Sphere	*E. coli* cell diameter	r	1.0	μm
Brownian motion	Force constant	K_{R}	2.2	$\frac{\mu m}{cs^{\frac{3}{2}}}$
Friction	Force constant	K_{F}	2.0	$\frac{\mu g}{cs}$
Gravity	Force constant	K_{G}	0.0002	$\frac{\mu m}{cs^2}$
Cell growth	Growth rate	G_r	0.00025	$\frac{\mu g}{cs}$
Specific interactions	Cell-cell force constant	K_c	0.1-10	$\frac{\mu g}{cs^2}$
	Cell-surface force constant	K_s	0.1-10	$\frac{\mu g}{cs^2}$
	Cell-cell probability	P_c	0.1-10	$\frac{interactions}{cs}$
	Cell-surface probability	P_s	0.1-10	$\frac{interactions}{cs}$

Table 8.1: Model features and their parameters for the biofilm case study models. For the models all parameters remained the same except for K_s, K_c, P_s and P_k.

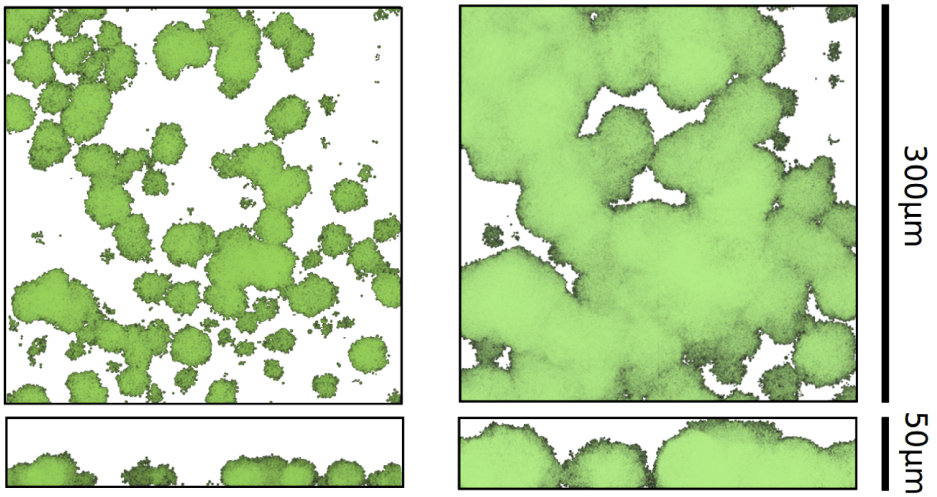


Figure 8.2: Snapshots of simulated biofilm with $K = 10$ and $P = 10$, shown after 12 hours on the *left* and after 24 hours in *right*. Both have a top-down view and a side view displayed below.

8.4 Results

We consider the height distribution of biofilms to characterise their morphologies. Biofilms which are flat and uniform produce a low standard deviation in height, whereas lumpy and irregular biofilms produce a larger height variation. Through this process we can relate local cell surface interactions to colony level spatial organisation.

We observe the effect of cell surface charge by growing biofilms with different cell parameters. The parameters modified are the rate $\boldsymbol{P_s}$ with which a cell adheres to a surface it is in contact with, strength $\boldsymbol{K_s}$ of the interaction with the surface, rate $\boldsymbol{P_c}$ with which a cell will adhere to another contacting cell and strength $\boldsymbol{K_c}$ of that cell-cell interaction. Snapshots of a simulated biofilm can be seen in Figure 8.2.

First we set cell-surface and cell-cell parameters to be symmetric, such that cells have the same rate at which they interact with other cells and surfaces, and they interact with other cells and surfaces with the same force constant.

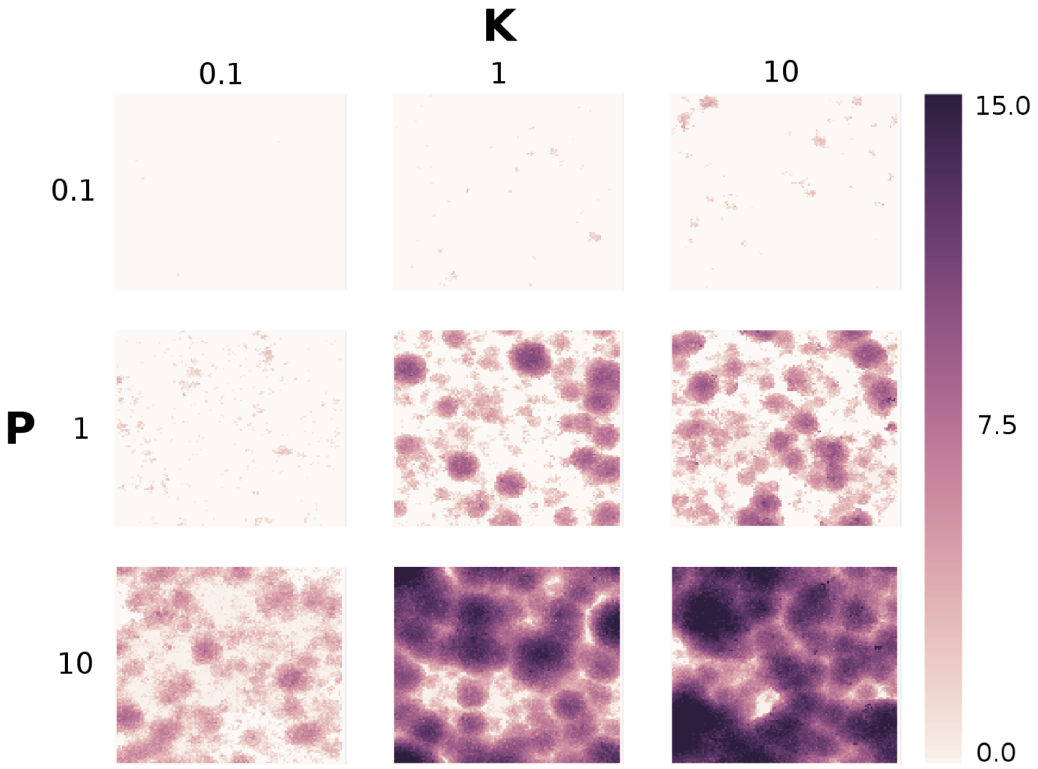


Figure 8.3: Heatmaps showing height (μm) of simulated biofilms under different parameters. The strength of interaction K and rate of interaction P are varied. Cell-surface and cell-cell interactions have the same rates in these simulations. These images show the biofilm after 12 hours.

From Figure 8.3 one can see that as we increase the rate at which cells interact $P_{s/c}$, the biofilm covers more surface area due to more bacteria attaching directly to the substratum. Clusters then form as other planktonic cells attach to those already in the biofilm. As we increase the strength of cell interactions $K_{s/c}$ we observe taller and denser biofilms, this may be explained by the fact that bacteria stick to each other more firmly and thus the biofilm can grow stable mushroom-like structures which extend from the substratum into the fluid medium.

By changing the parameters $P_{s/c}$ and $K_{s/c}$ we obtain varied biofilm development. However these parameters produces a consistent biofilm architecture, with hemispherical clusters of bacteria spreading across the surface forming lumpy and irregular biofilms.

We consider the effect that an asymmetrical cell adhesion to other cells than

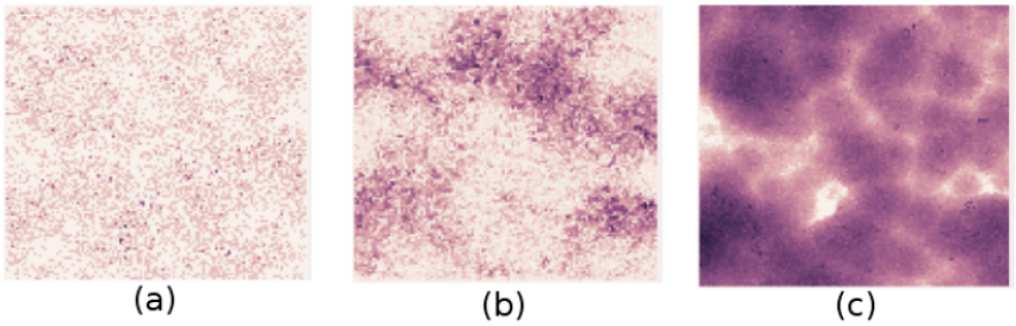


Figure 8.4: (a) Biofilm grown with high cell-surface interaction rates and low cell-cell interaction rates $[K_s = 10, P_s = 10, K_c = 0.1, P_c = 0.1]$. (b) Biofilm grown with high cell-surface interaction rates and medium cell-cell interaction rates $[K_s = 10, P_s = 10, K_c = 0.25, P_c = 0.25]$. (c) Biofilm as seen in Figure 7 (c) $[K = 10, P = 10]$ which has equal K and P values for both cell-cell and cell-surface ($K_c = K_s$ and $P_c = P_s$).

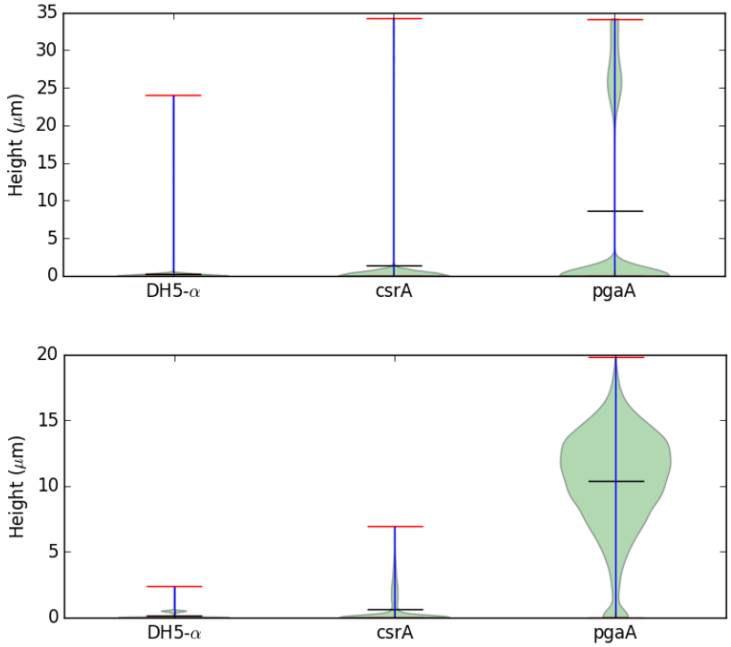


Figure 8.5: Height distribution of experimental biofilm (top) and simulated biofilm (bottom). The general trend of *DH5-α* forming very little biofilm (less than $1\mu m$), *csrA* forming slightly more flat biofilm (less than $5\mu m$), and *pgaA* forming a larger and lumpier biofilm (varied heights up to $20\mu m$ or more).

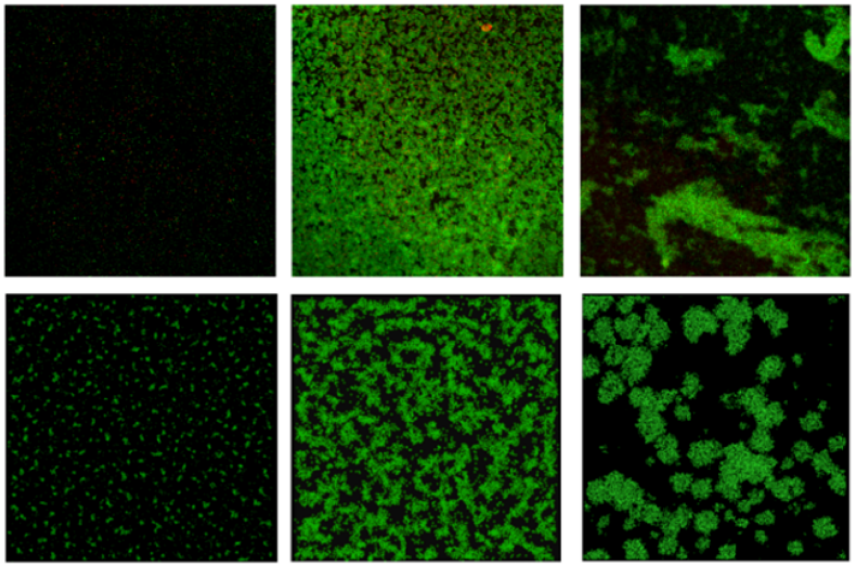


Figure 8.6: Comparison between experimental biofilms (top row) and simulated biofilms (bottom row). The images show the three different *E. coli* strains in the case study (from left to right), *dh5-α*, *csrA* and *pgaA*. The model and experiments look qualitatively similar, as well as having quantitative similarities as seen in Figure 8.5.

to surfaces would have. In Figure 8.4 (a, b, c) we compare biofilms grown with symmetric cell-cell and cell-surface adhesion to a biofilm grown with asymmetric parameters, such that the probability a cell will bind to a surface $\boldsymbol{P_s}$ and the strength of that cell-surface interaction $\boldsymbol{K_s}$ are relatively high in comparison to cell-cell interaction probability $\boldsymbol{P_c}$ and strength $\boldsymbol{K_c}$. This results in significantly reduced biofilm formation, with a spreading of cells across the surface leading to a more uniform structure.

Our model and analysis offer an explanation as to how cell-surface interactions can influence biofilm architecture. When cells interact with the environmental surfaces and other cells at a similar rate biofilms tend to form an irregular and lumpy structure. This can be explained by early colonisation leading to clustered growth on the surface forming an irregular structure, as lumps increase the probability that planktonic cells will adhere to them as they protrude into the fluid due to strong cell-cell interactions. Cells that interact weakly with each other

but strongly with a surface tend to form flat and uniform biofilms. This can be explained by cells in the biofilm being able to detach from other cells, allowing them to spread across the surface or becoming planktonic in the fluid; they may then colonise the surface elsewhere. Over time cells populate the surface, but due to weak cell-cell interactions a thick layer of cells does not emerge until the surface is covered forcing growth in height.

Model findings reinforce the observations made in experiments, Figure 8.5 (A) shows experimental biofilm height distributions and (B) simulated height distributions. Strains such as *PgaA* which have a higher surface charge have stronger cell-cell interactions which lead to large irregular biofilms, where as a low surface charge strain *DH5-α* produce less biofilm with a uniform structure. Visual comparisons between experimental and modelled biofilms can be seen in Figure 8.6.

8.5 Discussion of Simbiotics

This study was conducted alongside Study 1, and played a large part in informing the design of Simbiotics. This study involved the simulation of more biological features, such as cell growth and division, which were not present in the first case study. These further set the requirements of Simbiotics, driving the implementation of cell growth and division modules. The system also required the representation of a substratum (solid surface) on which bacteria could attach, and helped inform how the boundary conditions in Simbiotics could be implemented and specified by the user.

The modelling library architecture supported additional biology modules and boundary conditions well, allowing them to be defined as independent blocks and attached to the model specification as other features are. This process solidified the architecture of the library system (as described in Chapter 5), with a library consisting of subdomains, each with their own modules and sub-parameters. This structure was able to capture a range of features and having them dynamically linked by the modeller, giving them freedom to express models flexibly.

Challenges faced during this study include scaling to large simulation sizes. The model is more complicated than that shown in Study 1, and due to cell growth

the population of cells increases over time. Larger populations and clustering of cells as they grow on surfaces induces many collisions, which begins to slow the physics integrator at large population sizes. Simbiotics could simulate large system sizes, with a maximum of about 750,000 cells, which gave confidence in Simbiotics' capacity to simulate meaningful system sizes.

8.6 Summary

This case study investigated how physical interactions can influence biofilm formation and structure. A model including cell growth, motility and adhesion was developed, simulating the colonisation of a surface and expansion into a biofilm. Simulations were comparable to experimental findings both qualitatively and quantitatively, and showed that an asymmetric cell-cell and cell-substratum adhesion could be a potential explanation for the differences in biofilm architecture between the species.

These types of effects could potentially be harnessed in the pursuit of developing synthetic biofilms with targeted behaviour, providing insights into how cells can be modified to produce specific architectures. Control over the physical structure of biofilms may help us developing robust and efficient bio-devices, such as for the synthesis of nanomaterials [164], or the development of biosensors and intelligent surfaces [62, 146].

The work was done in collaboration with Sheffield Department of Chemical and Biological Engineering, with Joy Mukherjee conducting the experiments.

Chapter 9

Study 3 - Population Dynamics of Autocatalytic Sets in a Compartmentalized Spatial World

This chapter presents my contribution to a study on population dynamics of autocatalytic chemical sets conducted with Simbiotics. This case study was an unexpected use of Simbiotics, being an Origins of Life study exploring potential dynamics of protocells, in contrast to modelling bacterial population. The system of interest was very similar to models of bacterial populations, and thus Simbiotics was an appropriate tool. This Chapter overviews my contribution to a published manuscript, which was written by Wim Hordijk. My contributions involved developing the presented model, with which Wim studied a variety of system dynamics, and extending the Simbiotics modelling library to include relevant modelling features such as a discrete membrane transport implementation.

This chapter addresses Objective 4 - Studying the effect of synthetic chemical signalling and gene-regulation on biophysical patterning in bacterial populations, however in this case it is chemical signalling between compartments of chemical reaction networks.

This work is published: W. Hordikj, J. Naylor, H. Fellermann, N. Krasnogor **Population Dynamics of Autocatalytic Sets in a Compartmentalized Spatial World** in *Life*, 8(3):33, August 2018.

9.1 Overview

This case study investigates the influence of chemical signalling and intracellular dynamics on populations of protocells (referred to as compartments). The processes of intracellular dynamics and intercellular signalling were isolated in this study, excluding the modelling of physical interactions, considering the system to consist of spatially distributed immotile compartments. The model used simulation features not used in previous casestudies, namely for simulating intracellular chemical reactions with the Gillespie method, simulating membrane diffusion as a discrete process, and simulating extracellular diffusion. The study provided a good use of Simbiotics outside of its original scope, exercising the platform's versatility in represented various types of multicellular system. This study also drove the further development and testing of the platform, including adding additional features for modelling and streamlining the model building process.

9.2 Introduction

This case study was established when Wim Hordijk, a visitor at Newcastle University, became interested in using Simbiotics for an origins of life study. The study investigates how protocells may evolve, modelling autocatalytic chemical networks within compartments that can interact through diffusable molecules, simulating how these potential protocells may evolve over time. An example of one of the autocatalytic sets used in the study can be seen in Figure 27.

This collaboration involved me developing a model of the system, and Wim using this model to study different system dynamics. The study was primarily conducted by Wim and he wrote a now published manuscript (available in Appendix C). The system and modelling methods are described below, and my contribution to the study is outlined.

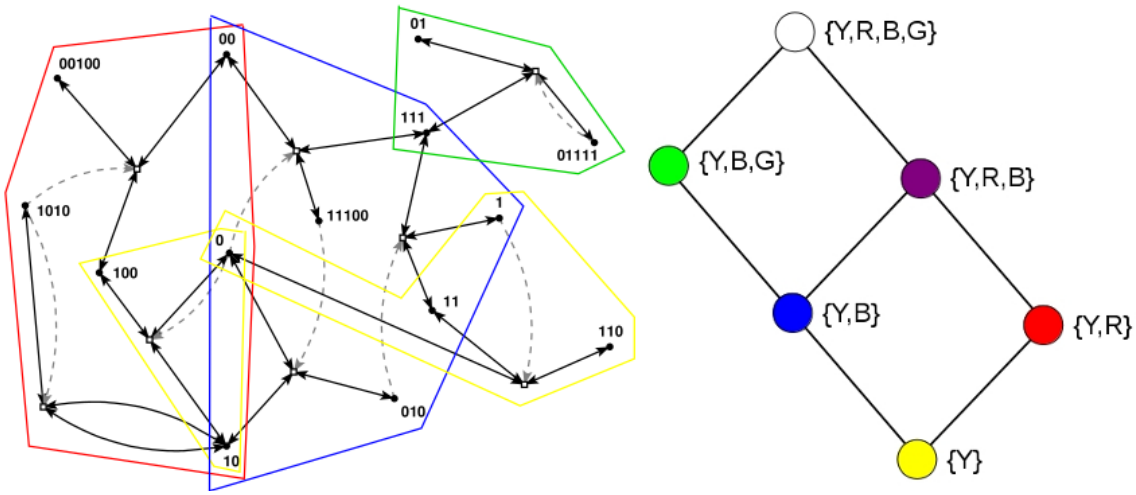


Figure 9.1: **RAF set example.** Left: An example of a RAF set as found in an instance of the binary polymer model. Black dots (labelled with bit strings) represent the molecule types, and white boxes represent reactions. Solid black arrows indicate molecules going into and coming out of a reaction, while dashed grey arrows indicate catalysis. Coloured polygons indicate some of the RAF subsets. Right: The six closed RAFs (colour coded) and their mutual subset relationships.

9.3 Methods

Model

As with previous studies, the systems are decomposed into their abstract *components* and *processes* relevant to this level of modelling.

Components

1. Substratum

The substratum is represented as a 2D domain on which compartments can exist.

***2.* Compartments**

Compartments are represented as rigid-body spheres.

Processes

1. Chemical reaction network

Chemical reaction networks are represented as stochastic Gillespie models embedded within compartments.

2. Membrane transport

Transport of chemicals across the cell membrane are modelled as discrete fluxes between the intracellular and extracellular spaces.

3. Extracellular diffusion

Diffusion of extracellular chemicals are modelled by rasterizing the 2D domain into voxels, and solving the flux of chemicals between these voxels.

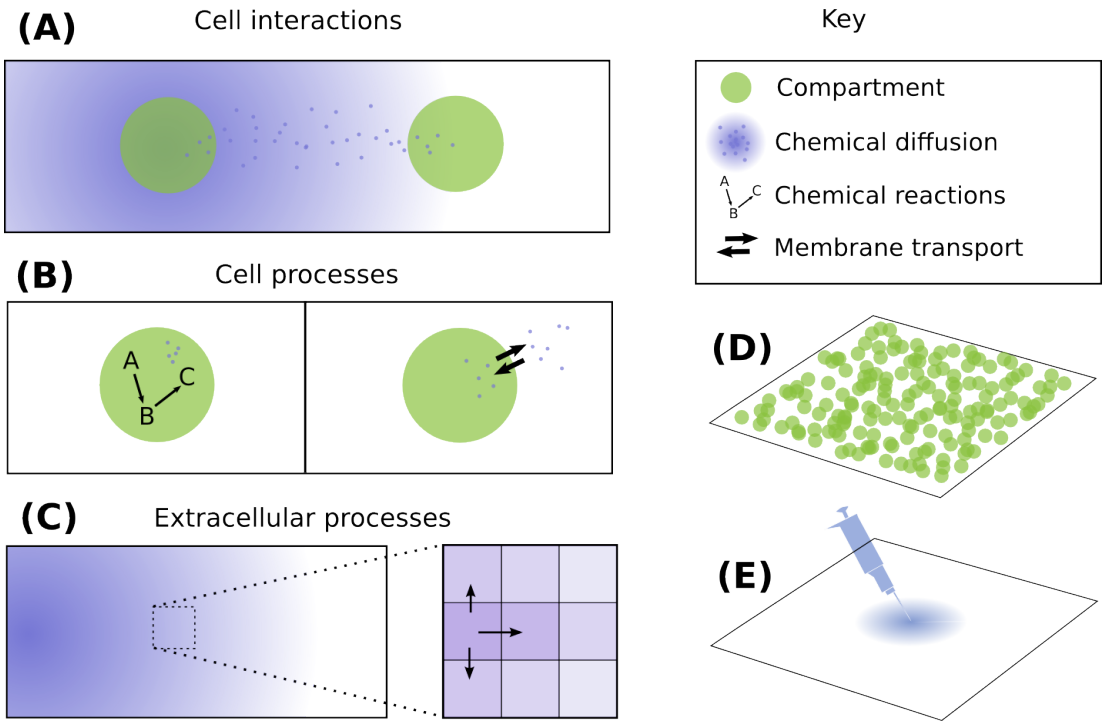


Figure 9.2: **(A)** The cell interactions in the model, showing that cells communicate via diffusable chemical signals. **(B)** The cell processes in the model. *Left* shows that cells have metabolic pathway activity. *Right* shows that cells have a membrane transports mechanism allowing chemicals to be transporting in and out the cell. **(C)** The extracellular processes in the model are chemical diffusion. **(D)** The initial condition of the model - a well mixed population of cells on a 2D surface. **(E)** Systems are induced by pipetting chemicals into the center of the domain.

Model description

The model is initialised with compartments (spheres) randomly distributed across the 2D domain, with initial chemical concentrations of some chemicals in the extracellular space. Compartments do not move or grow, and have chemical networks within them which are modelled with the Gillespie method. The membrane of each compartment is modelled with a discrete membrane implementation, using a Poisson sampler to calculate the current chemical fluxes. Chemicals in the extracellular space can diffuse, allowing compartment's reaction networks to interact with each other.

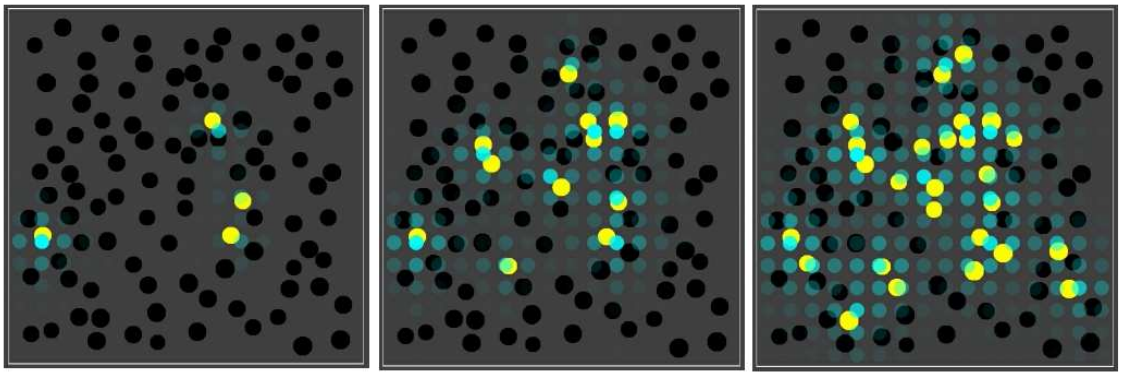


Figure 9.3: **The influence of a permeable inducer.** Three snapshots over time from a simulation where the RAF set produces an permeable inducer that can diffuse through the lattice. The blue spheres indicate the concentration of this inducer in the different grid locations.

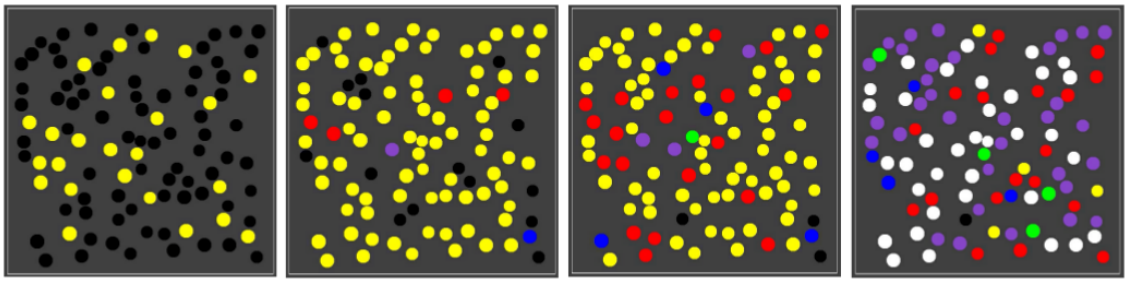


Figure 9.4: **A population of compartments.** Four snapshots over time from a simulation with 100 compartments.

The most active reactions within the compartment set its state, which is implemented as the compartment changing colour in the simulation. A snapshot of the model can be seen in Figure 34, showing a population of cells in different states (black or yellow) and diffusable chemicals (light blue). A later snapshots of the model can be seen in Figure 31, showing diversity in the states of compartments in the population, forming sub-populations of different phenotypes.

The time evolution of compartment's state changes were tracked, this was considered for numerous different chemical network sets. Full details on the models and study can be found in Appendix C.

9.4 My contribution

My contribution to this study involved developing a model of Wim's system in Simbiotics based on his requirements. This involved building and testing additional features that were desired for the model. Additionally I developed an interface to manipulate the features and parameters of the core model to specify his desired system, allowing him to conduct a variety of *in-silico* experiments with this model.

Wim's requirements for simulating his model were the following:

1. The model had to be able to simulate a chemical reaction network (as seen in Figure 26) using a Gillespie implementation.
2. Compartments should be simulated as permeable entities in a 2D domain
3. Chemicals should be diffusable between compartments
4. There should be some form of source to introduce chemicals to the system
5. The simulation should be able to handle low molecule numbers (meaning a discrete representation of chemical molecules is desired).

Simbiotics did not have a Gillespie module implemented, having only differential equations to represent the chemical network which were not appropriate to model low-molecule numbers as they are a continuous representation. Due to this, I developed a Gillespie module that could be used to describe the internal dynamics of a cell, which could be used as an alternative to differential equations. Additionally I implemented a module to describe permeation of chemicals through a membrane in a discrete form by implementing a Poisson sampler to calculate flux rates. The addition of these two new modules allowed for the system to be modelled in Simbiotics.

Wim had numerous systems of interest, and to facilitate this I developed a small modelling interface in the form of a text file input. This text file allowed Wim to input his own custom chemical networks, and set other parameters of the model such as the domain size, number of compartments, as well as initial chemical amounts. Developing this interface allowed for Wim to express a wide range of systems in a simple form, enabling him to use Simbiotics unsupervised.

9.5 Discussion of Simbiotics

The models for this case study involved significantly different features to studies 1 and 2. This study included simulating diffusable intracellular and extracellular chemicals, and specific chemical reactions networks within compartments. The simulations run for this study consisted of much small domain and cell population sizes, and did not require the modelling of cells growing, moving or colliding. Rather, intracellular reactions and extracellular chemical diffusion were the main aspects, and very low molecule numbers had to be handled.

Simbiotics already had the functionality for simulating these phenomena in some form, however they were not appropriate for this study. Specifically the reaction networks should be modelled using the Gillespie method rather than as differential equations, due to very low numbers of molecules. Additionally due to low molecule numbers, the membrane transport module (which solves how molecules move between intracellular and extracellular compartments) needed to be a discrete implementation, rather than continuous.

To include these new features was simple in Simbiotics, the appropriate section of the modelling library just required a new module to be added. The Gillespie method was implemented as a biology module, such that it could be attached to a species definition and could simulate intracellular reactions. The membrane transport mechanism was also implemented as a biology module, utilising a Poisson sampler to solve the rate at which molecules cross a membrane in a discrete form. This process further demonstrated how crucial it was for a framework such as Simbiotics to contain a set of cross-compatible features to not only model different features, but also provide different methods to model the same feature.

Developing an interface with which Wim could run different types of simulations drove the development of the user interface, Easybiotics (Chapter 6). In this case the interface was developed as an input script for this study, which would be parsed alongside the core model and used to set the details of the model. This process informed how the architecture of a front-end interface could be made, and which types of work-flows are useful for modellers.

9.6 Summary

This study investigated the influence of chemical signalling on compartmentalised chemical reaction networks, in the context of protocell evolution. This provided a simple study with which to use the software to model intercellular signalling in intracellular dynamics, whilst ignoring growth and movement of cells/compartments.

Wim Hordijk drove the study and used a model I built of his system to simulate numerous systems of interest. The resulting publication was written by Wim and can be found in Appendix C. This chapter overviewed my contribution to the study, including the model I developed and the features I implemented based on Wim's requirements.

Chapter 10

Study 4 - Pattern formation via synthetic cell signalling

In this chapter we present a study on pattern formation in multicellular systems based on synthetic cell signalling. This study was done in collaboration with Francisco Romero Campero at Universidad de Sevilla. This chapter further addresses Objective 4 - Studying the effect of synthetic chemical signalling and gene-regulation on biophysical patterning in bacterial populations.

10.1 Overview

This case study is motivated by the investigation of synthetic chemical signalling on the spatial patterning of gene regulation in multicellular systems. The processes of intracellular dynamics and intercellular signalling are isolated, modelling the system as being composed of spatially distributed immotile cells. This study naturally built upon the modelling components used in the previous study on chemical signalling (Chapter 9), using these components to explore how synthetic genetic circuits and chemical signalling can be *designed* to generate specific spatial patterns of gene regulation.

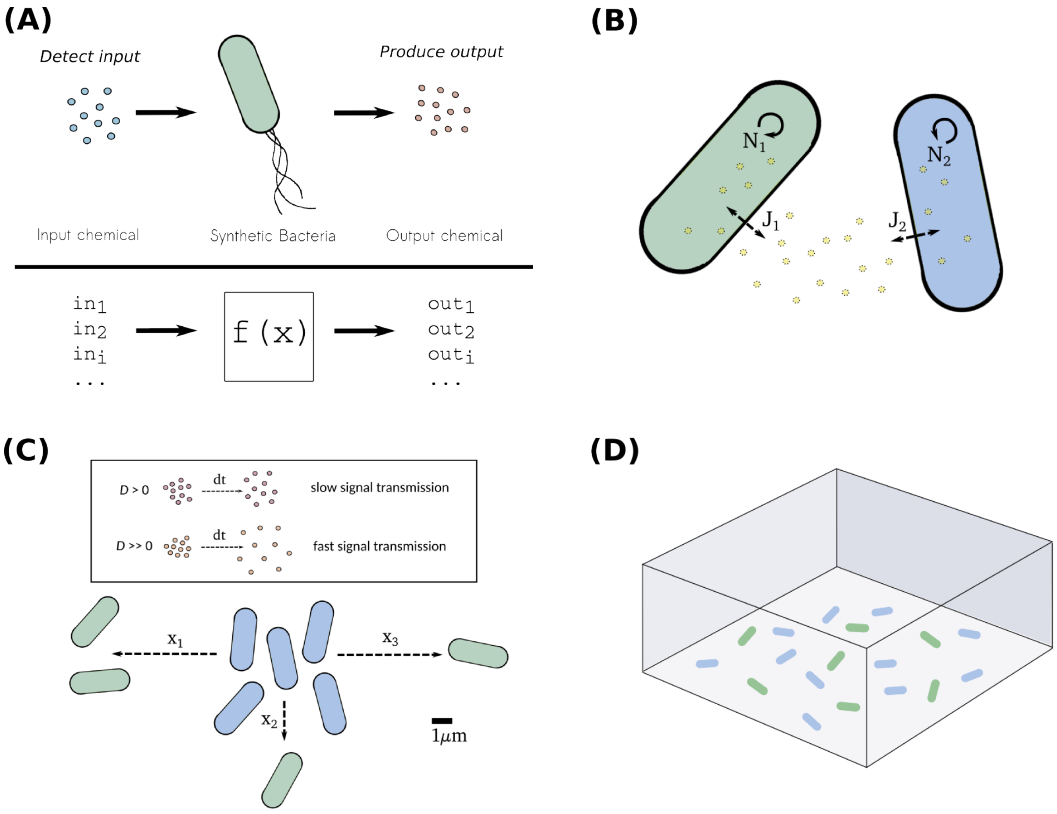


Figure 10.1: **(A)** Synthetic bacteria can act as individual processing units, or functions, which takes some input(s) and producing some output(s). **(B)** Different species of synthetic bacteria, each with their own targeted behaviour, can be connected by diffusable chemical species acting as 'wires'. **(C)** The chemical 'wires' are driven by chemical diffusion, and thus the properties of the diffusing molecule are important. Slowly diffusing molecules can be used as slow signal transmissions, and rapidly diffusing molecules for fast signal transmission **(D)** Different species of synthetic bacteria are spatially distributed in the most robust manner based on how signals should optimally propagate around the population to produce device functionality.

10.2 Introduction

Gene modification techniques have allowed for the re-programming of cellular dynamics. This allows for the design of targeted behaviour in cells, considering them as computational units or bio-factories which may act alone or in parallel with other programmed cells [11, 160, 178, 179]. Due to the complexities

associated with integrating large circuits into single cells, the development of distributed bio-computation is a more reliable method [143]. Techniques have been developed to accelerate the design of synthetic genetic circuits (SGRNs) [166], and early studies of spatially distributed populations of SGRNs call for further investigation of these systems [17, 18, 109].

Here we investigate two systems of SGRNs that produce targeted population behaviour. The first is a system that generates a pulse of gene expression, similar to the band-detector system [19]. The second is a pattern forming system that creates alternating stripes of gene regulation. Simbiotics is used to model the systems, allowing for design-space exploration and robustness testing.

Figure 10.1 overviews how we see cells as functional components in a device. We may see each species as an individual component that gives a specific output for a specific input, where those inputs and outputs are diffusable molecules. The transformation from input molecules to output molecules can be achieved by an SGRN within the cell. The cell should also have membrane transport mechanisms to uptake inputs and secrete outputs. Individual cells can then work in tandem to produce a population-scale device, possibly with many species each performing a specific part of the bio-device function. The communication between cells depends on the diffusible chemicals, thus chemical properties such as diffusion and degradation rate are crucial for effectively 'wiring' together different cells. The properties of particular diffusing molecules may be designed to control slow and fast signal propagation in the device, and the final bio-device spatially arranged in a manner which produces optimal functionality based on these designs.

10.3 Methods

10.3.1 Model

To model the developed systems we consider a simple population of cells that have SGRNs and can communicate via chemical signalling. These processes are isolated by excluding cell movement of growth from the model. The cells are modelled as rigid-body spheres that are spatially distributed across a 2D domain, as if they were on a substratum.

As with previous studies, the systems are decomposed into their abstract *components* and *processes* relevant to this level of modelling.

Components

1. Substratum

The substratum is represented as a 2D domain on which cells can exist.

***2.* Cells**

Cells are represented as rigid-body spheres.

Processes

1. Gene regulation

Gene regulatory networks are modelled as sets of ordinary differential equations.

2. Membrane transport

Transport of chemicals across the cell membrane are modelled as continuous fluxes between the intracellular and extracellular spaces.

3. Extracellular diffusion

Diffusion of extracellular chemicals are modelled by rasterizing the 2D domain into voxels, and solving the flux of chemicals between these voxels.

Model description

A population of cells are evenly distributed across a flat substratum, each cell has a gene network modelled as ODEs which are solved deterministically using a Runge Kutta 4th order integrator. Membrane transport between the cells and extracellular space are modelled as continuous fluxes, and the extracellular diffusion is modelling using a finite-difference implementation of Fick's law to calculate chemicals fluxes between the rasterized voxels. An illustration of the model can be seen in Figure 10.2. This model is used as the basis for simulating the two example systems, the further details of which can be found below in their corresponding results sections.

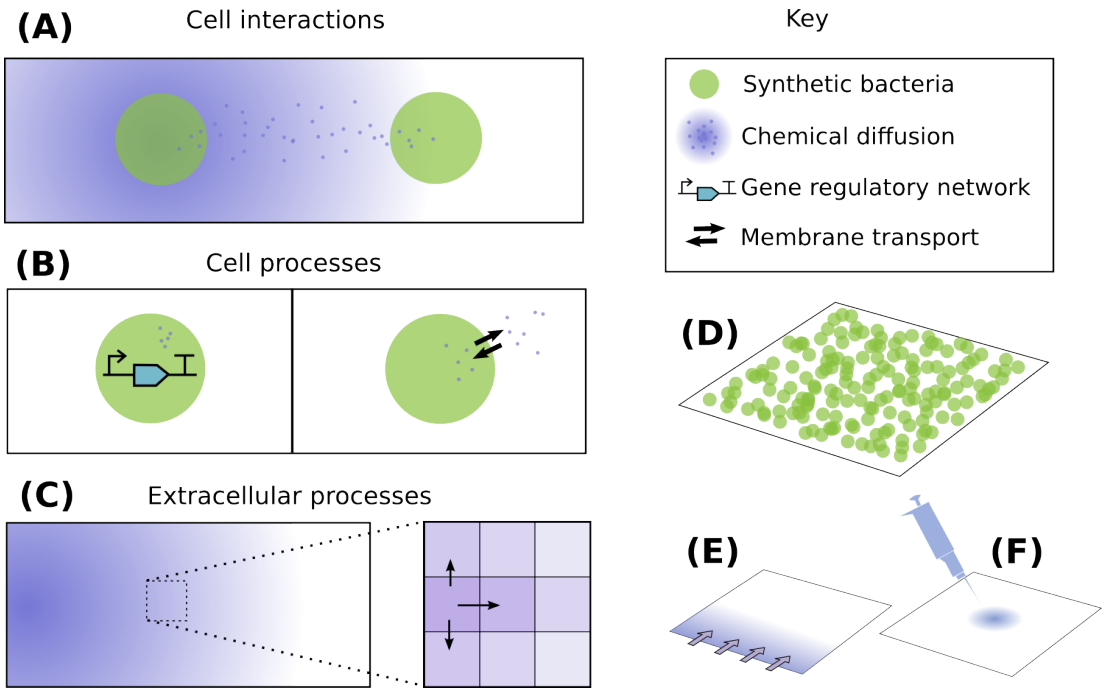


Figure 10.2: **(A)** The cell interactions in the model, showing that cells communicate via diffusable chemical signals. **(B)** The cell processes in the model. *Left* shows that cells have gene regulatory network activity. *Right* shows that cells have a membrane transports mechanism allowing chemicals to be transporting in and out the cell. **(C)** The extracellular processes in the model are chemical diffusion. **(D)** The initial condition of the model - a well mixed population of cells on a 2D surface. **(E)** An example of type of system induction used in the model - turning on a chemostat to input chemicals into the domain. **(F)** Another form of induction used in the study - pipetting chemicals into the center of the domain.

10.4 Results

10.4.1 Pulse generator system

The pulse-generator system consist of two species of cells, a sender species and a receiver species. The sender species produces a signal which the receiver can detect, triggering the receiver cell to produce GFP for a limited duration. Specifically, the sender species produces AHL, which is transported across the membrane and into the extracellular space. The AHL may diffuse through the extracellular space, and can be transported across the receiver cell's membrane, where it

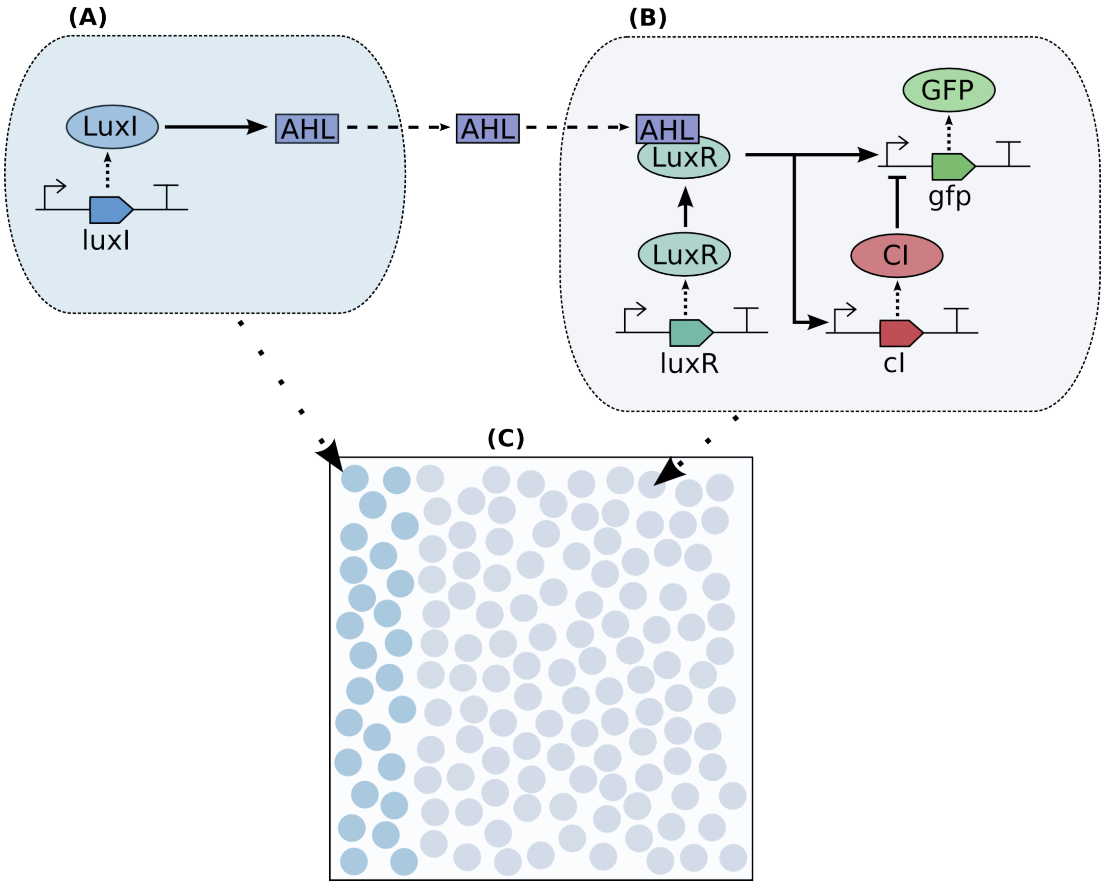


Figure 10.3: The sender and receiver genetic networks for the pulse generator case study. The sender cell synthesizes AHL which may diffuse through the extracellular space into the a receiver cell. AHL causes a pulse of GFP production in the receiver cell by activating GFP synthesis and a slower activation on CI production which ends up inhibiting GFP expression.

involved in the receiver's gene regulatory network. This causes GFP to be expressed by the receiver, as well as CI at a lower rate, which eventually inhibits GFP synthesis. As the signal molecule AHL diffuses through the spatial domain, it causes a pulse of GFP to propagate through the population, emanating from the position(s) where AHL was added. The signal molecule and GFP pulse decay over time. A variation of this system is also modelled where the receiver cell relays the signal when active, by producing and secreting AHL in addition to GFP being synthesized. This propagates the signal and prevents the GFP pulse from decaying. The influence of the diffusing signal molecule on these two systems is

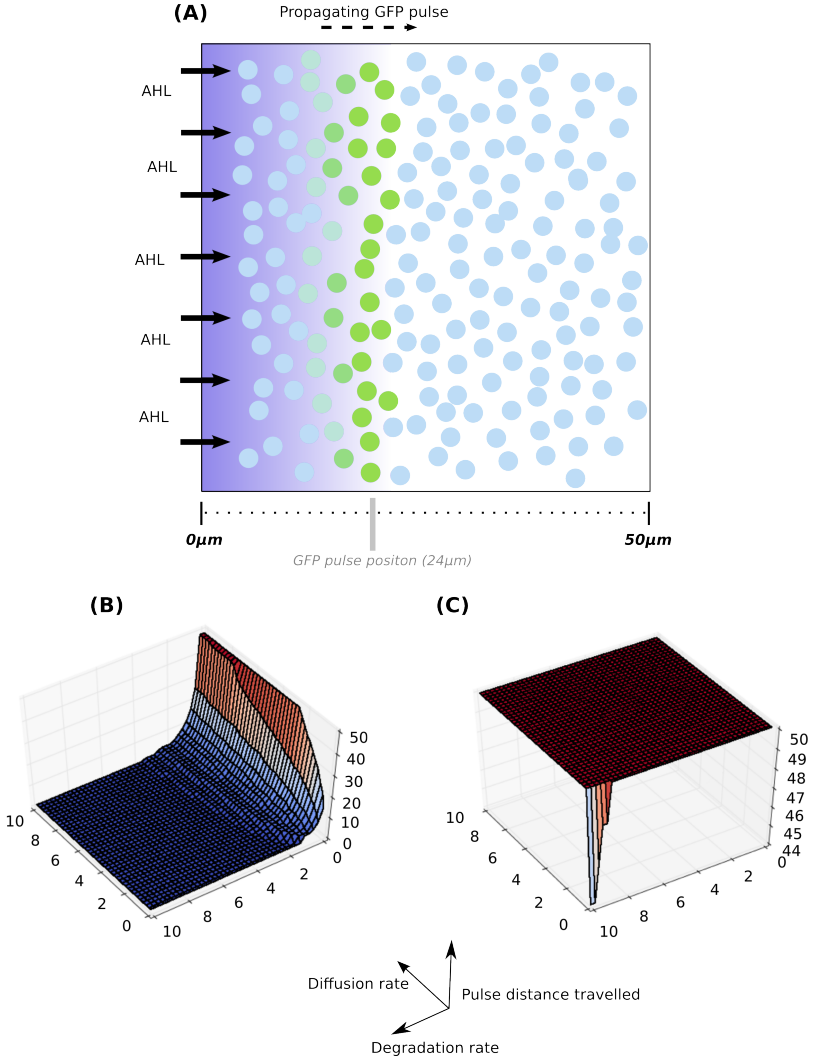


Figure 10.4: **(A)** A schematic showing the experiment for measuring pulse propagation response of the receiver cells. AHL is introduced via a chemostat on the left of the simulation domain supplies an initial amount of AHL and is then switched off. **(B)** Heatmap showing the distance that the pulse propagated to for the circuit which does not produce additional AHL. Low diffusion rates and high degradation rates cause the pulse to decay before travelling across the whole domain. **(C)** Heatmap showing the pulse distance travelled for the circuit with relay (that produces AHL as well as GFP). The pulse is propagated further, typically reaching the end of the domain, however for very low diffusion rates it can be seen that the pulse still decays and does not continue indefinitely.

Model feature	Parameter	Symbol	Value	Unit
Domain	Width	x	100	μm
	Height	y	0	μm
	Depth	y	50	μm
	Grid depth	G_d	7	
Sphere	*E. coli* cell diameter	r	1.0	μm
Membrane diffusion	Permeation rate	$D_{\mathrm{i}}^{\mathrm{j}}$	1.0	$\mu m^2 s^{-1}$
Extracellular diffusion	Diffusion rate	D_{i}	0.001 - 10.0	$\mu m^2 s^{-1}$
	Degradation rate	K_{i}	0.001 - 10.0	$\mu m^3 s^{-1}$
GRN parameters		k_1	0.6	$\mu m^3 s^{-1}$
		k_2	0.01	$\mu m^3 s^{-1}$
		k_3	1.0	$\mu m^3 s^{-1}$
		k_4	0.001	$\mu m^3 s^{-1}$
		k_5	0.5	$\mu m^3 s^{-1}$
		k_6	0.0001	$\mu m^3 s^{-1}$
		k_7	0.0005	$\mu m^3 s^{-1}$
		k_8	2.0	$\mu m^3 s^{-1}$
		k_9	0.5	$\mu m^3 s^{-1}$
		k_{10}	0.00001	$\mu m^3 s^{-1}$
		v_1	0.6	$\mu m^3 s^{-1}$
		v_2	0.45	$\mu m^3 s^{-1}$
		v_3	2.0	$\mu m^3 s^{-1}$

Table 10.1: Model features and their parameters for the biofilm case study models. For the models all parameters remained the same except for K_s, K_c, P_s and P_k.

studied, considering the effect signal diffusion and degradation coefficients have on the velocity and duration of the propagating GFP pulse.

A schematic showing the sender and receiver (without the relay) can be seen in 10.3, and the equations describing this system one below:

ODEs describing *sender* cell's GRN

$$\frac{dAHL}{dt} = v_1 - (k_1 * AHL) \tag{10.1}$$

ODEs describing *Pulse receiver* cell's GRN

$$\frac{dAHL}{dt} = -k_1 * AHL \tag{10.2}$$

$$\frac{dLuxR}{dt} = v_2 - (k_2 * LuxR) \tag{10.3}$$

$$\frac{dLuxRAHL}{dt} = (k_3 * AHL * LuxR) - (k_4 * LuxRAHL) \tag{10.4}$$

$$\frac{dGFP}{dt} = \frac{v_3 * LuxRAHL}{LuxRAHL + k_5} * \frac{1}{1 + (CI/k_6)} - (k_6 * GFP) \tag{10.5}$$

$$\frac{dCI}{dt} = \frac{v_4 * LuxRAHL}{LuxRAHL + k_5} - k_7 * CI \tag{10.6}$$

The simulations are performed in a 2D domain of dimensions $50 * 50\mu m$. For simplicity in characterising the receiver cell's dynamics, the sender cells are represented as a chemostat on the left side of the simulation domain which serves as the influx of an initial amount of AHL to induce the pulse. A schematic of this can be seen in Figure 10.4 (A).

The first test performed analyses the distance travelled by the pulse for the two variations of the genetic circuit. In the case of the receiver circuit which does not produce AHL, it can be seen that the distance travelled by the pulse is proportional to the diffusion rate and inversely proportional to the degradation rate, as seen in Figure 10.4 (B). In the case of the receiver circuit with relay (that produces AHL as well as GFP), it can be seen that the pulse travels a further distance, and for the parameters simulated tends to reach the other side of the simulation domain ($50 * 50\mu m$), as seen in Figure 10.4 (C). The pulse does not propagate indefinitely even when receiver has the relay circuit, as can be seen for very low diffusion rates and high degradation rates.

The second test performed focuses on the variation of the circuit which produces AHL as well as GFP. The velocity of the pulse was measured for different diffusion and degradation coefficients. Pulse velocity scales scales proportionally with the diffusion coefficient, and inversely proportional to the degradation coefficient, as seen in Figure 10.5 (A).

The robustness of the system under different spatial arrangements of cells is also studied. A patchy spatial arrangement of receiver cells can inhibit the propagation of the GFP pulse, as seen in 10.5 (B) and (C). It is observed that for low diffusion rates, the pulse propagation is robust and is not greatly affected by potentially sparse patchy receiver cell spatial arrangements, however it has a low signal propagation speed. High diffusion rates tend to be less stable, even for low degradation rates, this is most likely due to the signalling molecule (in this case AHL) being able to diffuse away freely into the space where there are no cells a degrade before sufficiently activating receiver cells genetic circuits to propagate the pulse at a high velocity.

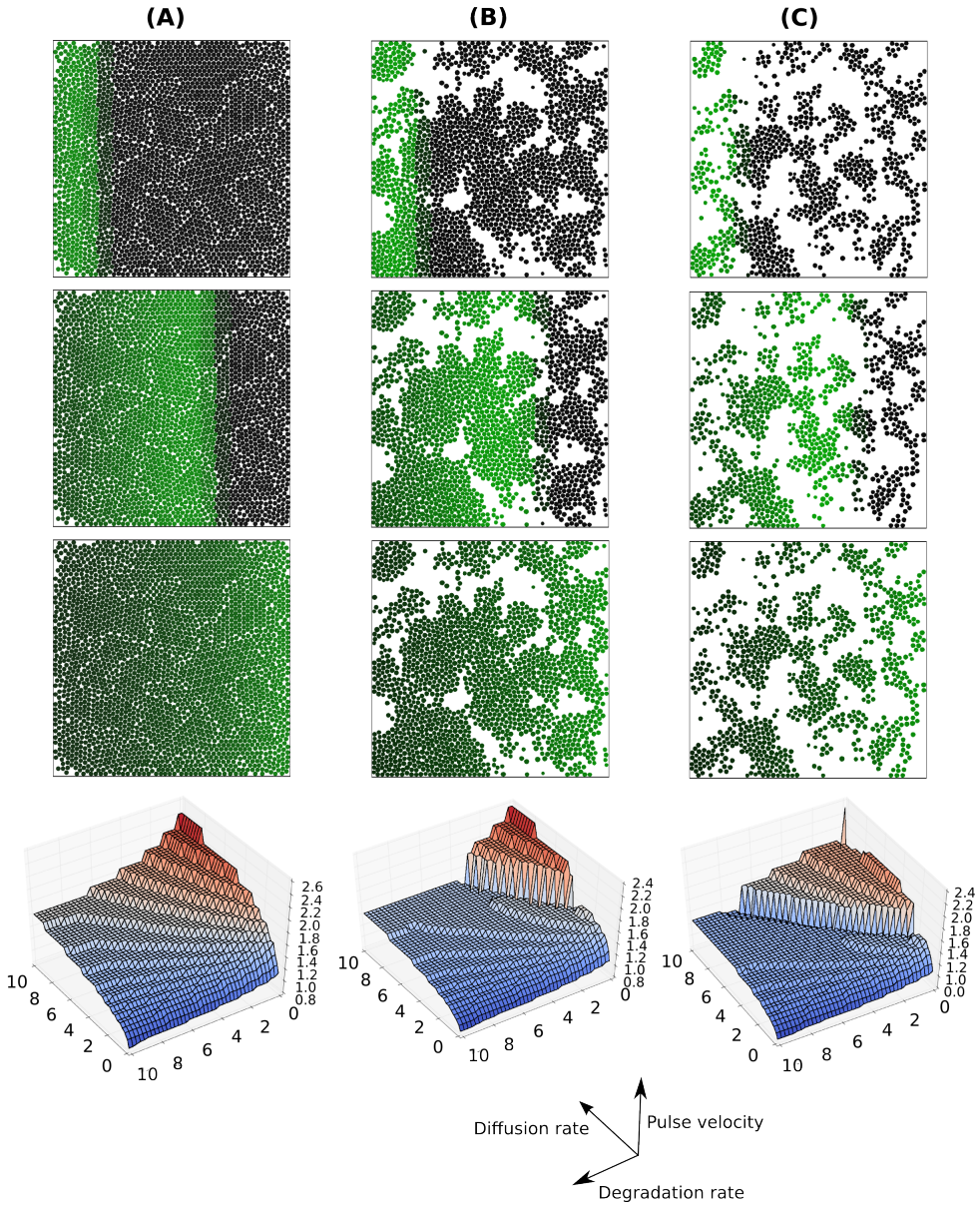


Figure 10.5: The pulse velocity for receiver circuit which relays the signal is measured for different diffusion and degradation coefficients of the signal molecule (AHL). The robustness of signal propagation is considered by performing the same experiment for different spatial arrangements of cells.

Pulse generation through colonies

To investigate how the pulse generator worked in a larger population of cells with a differential spatial arrangement, the receiver circuit (without relay) was embedded

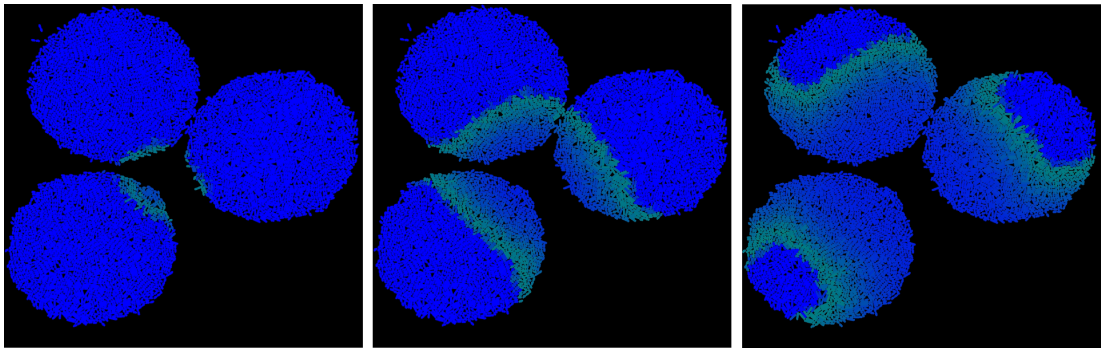


Figure 10.6: The pulse generator receiver circuit (without relay) embedded in cells part of three adjacent colonies. The system is induced by pipetting AHL into the center of the domain - this causes a pulse to propagate through the colonies from that position. It can be observed that the pulse becomes distorted around the edges of the colony, this is due to the AHL being able to diffuse more freely in the empty space, where as it is consumed as it diffuses through areas with cells in it - this causes the diffusion cloud to wrap around the colonies.

in three adjacent colonies of bacillus (rod-shaped) cells. The system was induced by pipetting AHL was pipetted into the center of the domain. The results can be seen in 10.6, showing the pulse emanate from the center and through the colonies. An interesting phenomenom observed in the model is that the pulse appears to wrap around the colonies and deforms the pulse propagation. An explanation for this may be that the AHL can diffuse freely (without being consumed) in the empty spaces between colonies, causing it to diffuse relatively quick to the AHL diffusing through the colonies.

10.4.2 Pattern formation system

The pattern-formation system consists of a receiver cell with a more complex GRN, seen in Figure 10.7. Receiver cells have parallel genetic circuits, one which produces fluorescent protein *F1*, and the other produces a different fluorescent protein *F2*. Synthesis of *F1* is induced by a diffusable signal molecule *S2*, and synthesis of *F2* is induced by a different signal molecule *S1*. The genetic circuit that synthesizes *F1* when induced also produces a repressor *R1*, which inhibits the activity of the circuit that synthesizes *F2*, and vice versa. This means that when a receiver cell receives input signal *S1* or *S2*, it commits to that genetic

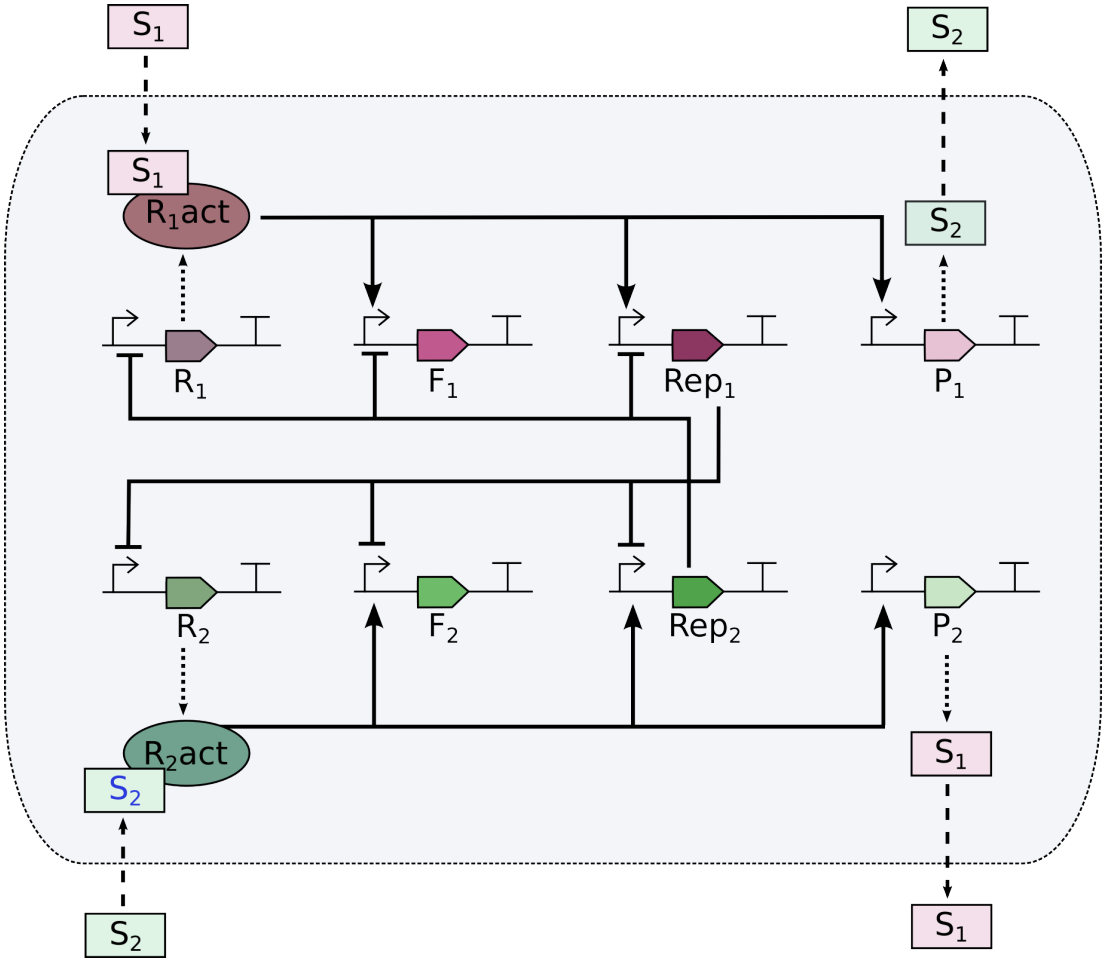


Figure 10.7: The genetic circuit for the receiver cell in the pattern-formation case study. There are two parallel circuits, which are activated by signalling molecules S1 or S2 respectively. Circuit activation causes synthesis of the respective fluorescent protein (F1 or F2), inhibition of the parallel circuit, and synthesis of the opposite signalling to that which it was activated by. This causes a leap-frog effect and stripes of gene expression as the signals diffuse through a spatially distributed cellular population, as S1 triggers S2 production, S2 then triggers S1 production.

circuit being activated, repressing the activity of the other. Furthermore, when a genetic circuit is active, is being synthesizing the signal molecule for the other genetic circuit, which is then transported out of the cell. This specifies that a receiver cell producing fluorescent *F1* produces a signal telling its neighbours to produce *F2*.

ODEs describing *Pattern formation receiver* cell's GRN

$$\frac{dR_1}{dt} = \frac{v_1}{1+\frac{Rep_2}{k_i}} - (k_r * R_1) - (k_c * R_1 * S_1) \tag{10.7}$$

$$\frac{dR_2}{dt} = \frac{v_1}{1+\frac{Rep_1}{k_i}} - (k_r * R_1) - (k_c * R_2 * S_2) \tag{10.8}$$

$$\frac{dF_1}{dt} = v_2 * \frac{R_1act}{R_1act + km} * \frac{1}{1+\frac{Rep_2}{k_i}} - (k_d * F_1) \tag{10.9}$$

$$\frac{dF_2}{dt} = v_2 * \frac{R_2act}{R_2act + km} * \frac{1}{1+\frac{Rep_1}{k_i}} - (k_d * F_2) \tag{10.10}$$

$$\frac{dR_1act}{dt} = (k_c * R_1 * S_1) - (k_d * R_1act) \tag{10.11}$$

$$\frac{dR_2act}{dt} = (k_c * R_2 * S_2) - (k_d * R_2act) \tag{10.12}$$

$$\frac{dRep_1}{dt} = v_3 * \frac{R_1act}{R_1act + k_m} * \frac{1}{1+\frac{Rep_2}{k_i}} - (k_d * Rep_1) \tag{10.13}$$

$$\frac{dRep_2}{dt} = v_3 * \frac{R_2act}{R_2act + k_m} * \frac{1}{1+\frac{Rep_1}{k_i}} - (k_d * Rep_2) \tag{10.14}$$

$$\frac{dP_1}{dt} = v_4 * \frac{R_2act}{R_2act + k_m} - (k_d * P_1) \tag{10.15}$$

$$\frac{dP_2}{dt} = v_4 * \frac{R_1act}{R_1act + k_m} - (k_d * P_2) \tag{10.16}$$

$$\frac{dS_1}{dt} = (v_5 * P_1) - (k_s * P_1) - (k_c * R_1 * S_1) \tag{10.17}$$

$$\frac{dS_2}{dt} = (v_5 * P_2) - (k_s * P_2) - (k_c * R_2 * S_2) \tag{10.18}$$

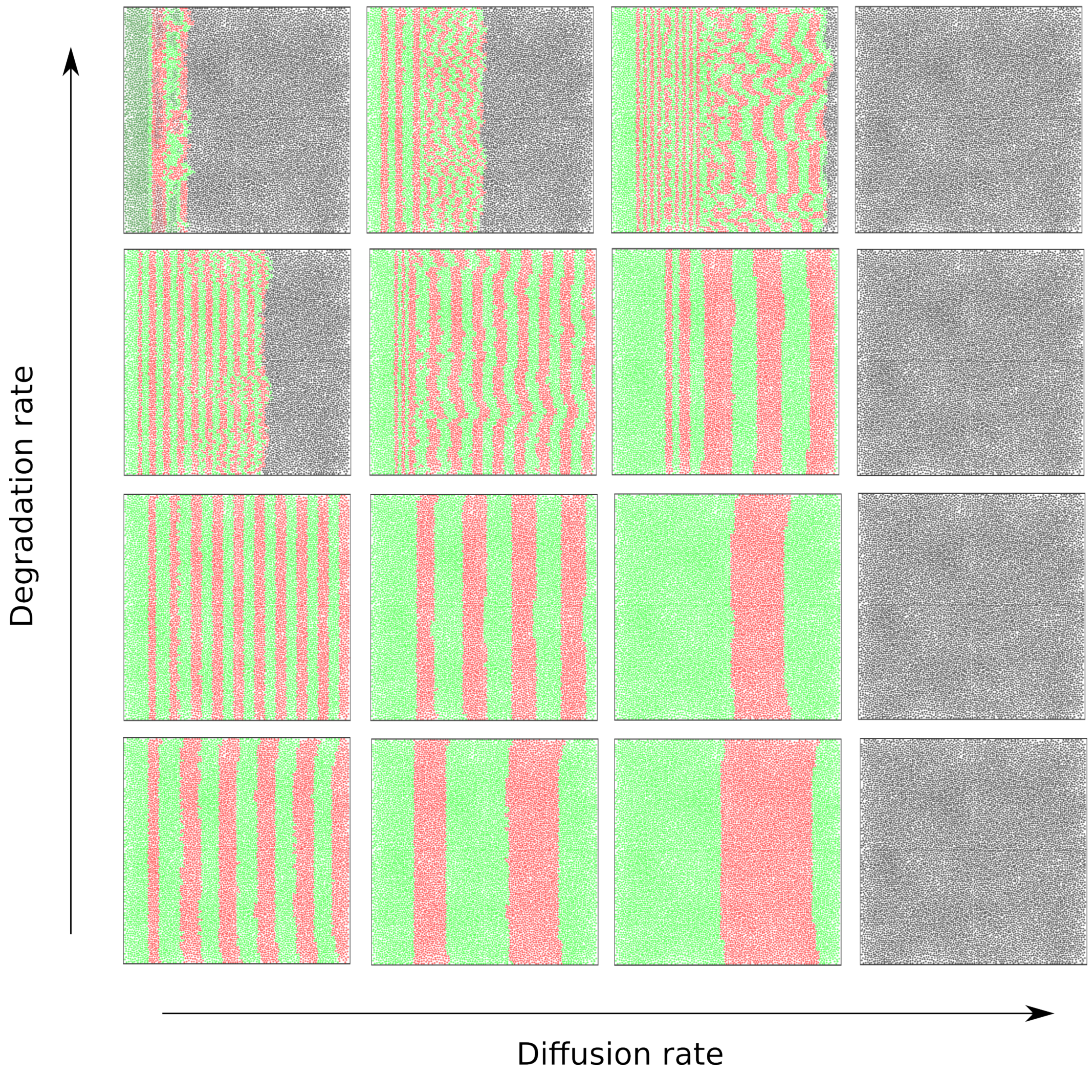


Figure 10.8: Simulations of the *pattern formation* system forming linear stripes. The system is induced by a chemostat being temporarily turned on at the left side of the domain. Results for different diffusion and degradation rates are displayed - showing that stripe frequency is inversely proportional to diffusion rate, and proportional to degradation rate.

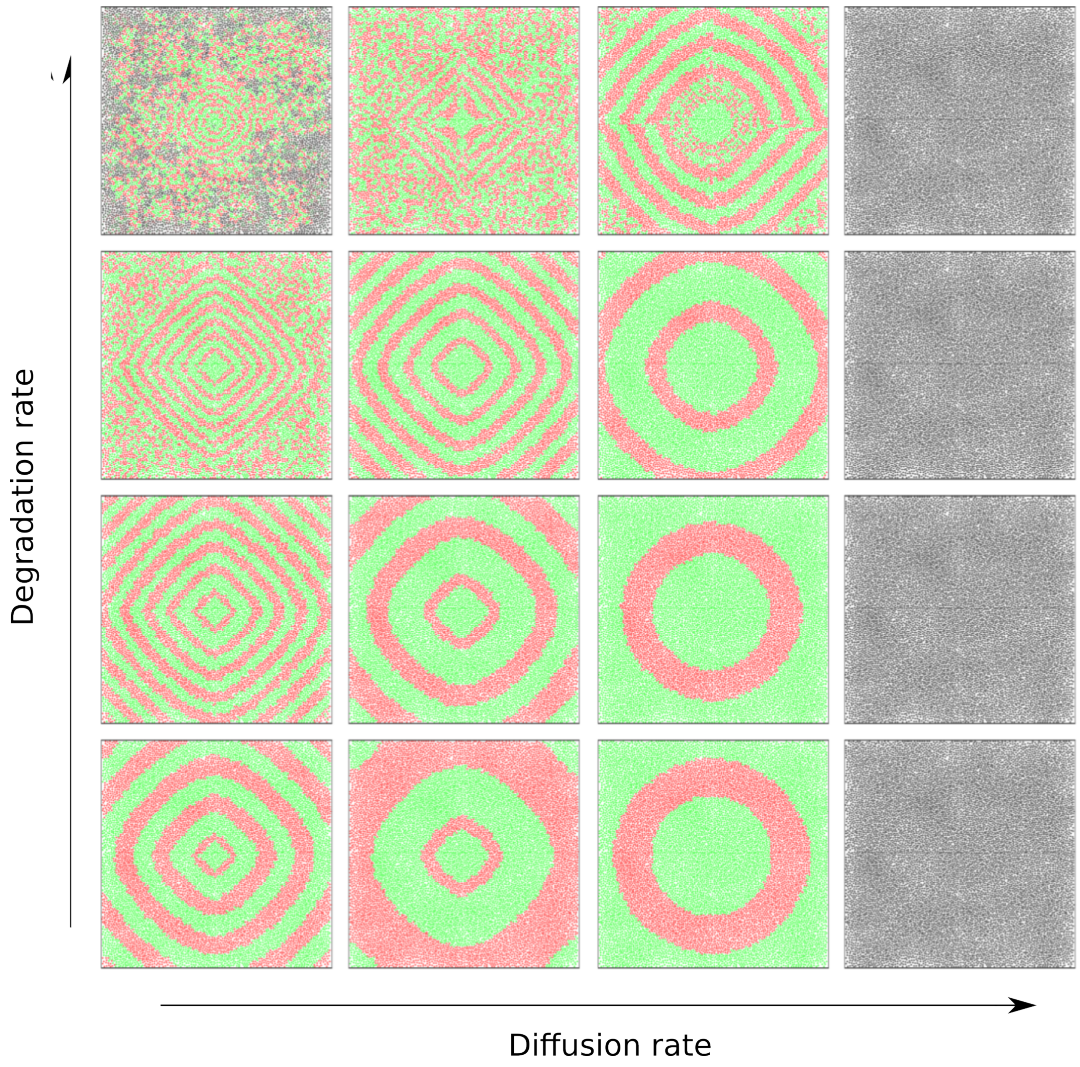


Figure 10.9: Simulations of the *pattern formation* system forming linear stripes. The system is induced by a pipetting into the center. Results for different diffusion and degradation rates are displayed - showing the same trends as the linear system. The spatial rasterization begins to cause artefacts at low diffusion rates and high degradation rates, causing the concentric rings to appear more diamond shaped.

The emergent behaviour of this system is that from the initial position(s) of induction with either *S1* or *S2*, stripes of alternating production of *F1* and *F2* emanate from the induction position(s). We consider the influence of the diffusion and degradation coefficients of the diffusing signals *S1* and *S2* on the frequency of these stripes.

The simulations are performed in a 2D domain of dimensions $100*100\mu m$, and two types of system induction are performed. The first is by temporarily turning on a chemostat to create an influx of inducer molecule $S2$ into the system, in the same way AHL was induced into the pulse generator study. The results for this can be seen in Figure 10.8 showing linear stripes. The second induction method is by pipetting $S2$ into the center of the domain. The results for this can be seen in Figure 10.9, showing concentric rings emanating from the induction point.

The frequency of stripes is observed to be inversely proportional to the diffusion coefficient, and proportional to the degradation coefficient, as seen in Figure 10.8. High values for diffusion and degradation tend to inhibit the pattern formation process. In the case of diffusion, high values result in the activation of the genetic circuit not occurring sufficiently to produce fluorescent proteins, as the signal molecule diffuses to low concentrations too rapidly. In the case of degradation, high values result in distortion of the pattern formation process, leading to the disorganisation of the stripes into unpatterned distributions of fluorescent protein expression, and the decay of the patterns propagating further.

10.5 Discussion of Simbiotics

This case study required similar features to Study 3, consisting of a static cell population in a 2D or 3D domain, with intracellular dynamics and diffusable molecules. The implementation of the model was not a challenge in Simbiotics, and did not require the implementation of additional modelling features. The differential equation module was used to model individual cell gene networks, and the membrane transport mechanism was used in its continuous form.

The main challenge with the study was finding the parameter regions where the desired phenomena occured, as simulations may take some time for population dynamics to emerge, and there were numerous parameters to the models.

Simbiotics was a powerful tool here, as it easily allowed for models to be run with different parameters, however for exploring many parameters and storing the results in a meaningful manner was still a mostly manual process. This motivated the development of a parameter sweep module, which was developed in Python as an external module. The python script cycled through parameters and started simulations, then saved the results to a folder system organised by modified parameter values. This module was used in Easybiotics as part of the parameter sweep and graph plotting implementations (as described in Chapter 6).

10.6 Summary

This case study has explored theoretical functionalisation and patterning of multicellular systems through the design of synthetic chemical signalling. The geometric effect of spatially distributed cells and chemical signalling can be controlled to generate designed behaviours in populations, allowing for the conceptualisation of distributed genetic circuits. These studies offer insights into how basic functional components of a more complex distributed multicellular device could be developed.

Specifically, a synthetic circuit for propagating a pulse of gene regulation across a population of cells has been characterised, exposing how such a system could be robustly implemented for signal propagation. Further investigation into this system could involve randomising patchy populations of cells and measuring pulse velocity, in order to better understand the relationship pulse propagation has with population density and geometric distribution of cells.

A pattern formation system for implementing stripes of alternating gene regulation has also been studied, considering the influence of diffusable molecule properties on stripe formation and width. An extension of this case study could explore the functionalisation of the stripes, with cells grown and patterned on a 2D surface acting as a signal processing array.

Simbiotics has provided a CAD environment for designing and analysing these systems. The potential design space of a system can be explored through rapid-model prototyping, and characterised through methods such as parameter sensitivity analysis, identifying driving forces and key relationships in the system.

This form of *in-silico* modelling allows for feasibility and robustness testing of systems prior to synthesis, aiding in the design of distributed genetic circuits in synthetic multicellular systems.

Chapter 11

Discussion and Conclusions

This chapter presents the discussion and conclusions of this research. The original motivations and goals are revisited, then a reflection on the contributions is presented. This is followed by a discussion of the limitations of the research, and the future outlook for this area of study.

11.1 Overview of thesis motivation and goals

The natural world exhibits a wide range of phenomena that emerge from common parts. The versatility and performance of natural systems is undeniable, with robust systems on many scales having evolved. How nature achieves this is a salient question, and pursuit of this can help us understand natural phenomena as well as how we may advance our own engineering capabilities, benefiting from those design principles found in nature.

Biological multicellular systems are a prime example of a natural system that exhibits complex, robust and versatile behaviour. A comprehensive understanding of how these systems behave has been paved by years of research in the domain of Biology, and accelerated by the emergence of Systems Biology, allowing for analysis of large data sets and formalisation of knowledge into predictive models. Functional biological components have been identified, that Synthetic Biology aims to repurpose for bioengineering purposes.

Facilitating these practices are computational methods and tools, of which

many have been developed for simulating particular processes of cells, however there does not exist an effective tool for integrating these methods into a multi-scale model of cell populations in an accommodating manner.

My thesis aims to integrate the modelling and analysis methods for bacterial populations into an easy to use framework, enabling domain experts who may have minimal programming experience to engage in modelling using cutting edge simulation techniques. Additionally I aimed to apply the developed framework to modelling physical and biochemical interactions in multicellular systems.

The specific aims of this thesis as stated in Chapter 1 are:

- **Aim 1:** Develop an easy to use, flexible and extendable workbench for integrative modelling of multicellular populations.
- **Aim 2:** Model and analyse multicellular populations patterned by physical and biochemical interactions.

11.2 Reflection on contributions

Through addressing these aims, my research strived to expand computational methods for integrative modelling of multicellular systems. Key challenges laid in integrating cell processes into a flexible single-cell model, and coupling this with physically realistic modelling techniques to simulate interacting populations of those single-cell models. Embedding these features in a software framework that other programmers could include and build with was the next step, followed by the development of a higher level software tool allowing domain experts who may only have minimal programming experience to use the tool.

Addressing these challenges involved conception of a modular software architecture in which a library of state-of-the-art methods could be implemented as modules, and provided to the modeller as 'building blocks' with which they can construct a model (presented in Chapters 3 and 5). The library focuses on the features of bacterial systems, however the library can be easily extended to include modules relating to other multicellular systems (such as for the modelling of tissue development). Notably the library modules for simulating intracellular dynamics include Boolean networks, ODEs, Gillespie simulations and SBML

integration, which can be used in conjunction with membrane and extracellular diffusion methods for simulating populations of interacting single-cell models in a spatial domain.

The extendibility of the platform was achieved through the programming of modular interfaces, allowing for new functionality to be added to the platform and library in a self-contained manner. This proved useful during case study modelling, which often drove the implementation of specific methods to be included in the library. For example in Study 3 (Chapter 9) the systems simulated involved low numbers of diffusable molecules, therefore a discrete version of the membrane diffusion method was implemented in order to realistically simulate the model. The addition of this discrete membrane diffusion method involved writing a single Java class that extending one of the Simbiotics interfaces. This class is then included in the library, which automatically encodes the module in the library file, and is seen as a valid module in the model file without any modification.

The versatility of the platform allowed for a range of multicellular systems to be modelled. This was achieved by implementing the methods in the library as modular 'building blocks' which could be composed with other modules via interfaces, giving the modeller freedom to integrate only the methods they wished to use. This compositional method for model development proved to be a powerful way to create the case study models (Chapters 7-10) and the models reproduced from literature (Chapter 4.3). Each of those models required the use of different methods, such as modelling intracellular dynamics using the Gillespie method in Study 3, and using differential equations in Study 4. A figure illustrating which modules were used in the case study and others models can be found in the Part II Abstract.

Simbiotics has been used to model the multicellular systems patterned by physical and biochemical interactions (Chapters 7-10). The findings of these studies have shown the valuability of Simbiotics for modelling emergent dynamics in natural systems, and in designing emergent behaviours in synthetic ones. Significant system sizes (up to 750,000 cells) have been simulated, and investigated through parameter sensitivity analysis to determine driving forces of the system and relationships between parameters. The literature findings reproduced

in Chapter 4 Section 4.3 act as confirmation that the implemented features in Simbiotics produce realistic simulations of physical and biochemical processes in multicellular systems.

Notably Simbiotics has been used to study systems outside of the initially intended domain of bacterial multicellular systems, being applied to Origins of Life study regarding auto-catalytic sets in interacting spatial compartments, as seen in Chapter 9. The versatility of the tool has found uses by other researchers; it was used by the Newcastle 2017 iGEM team to model a distributed biosensor - where the SBML models simulated in Simbiotics were designed in Copasi; David Nettleship used Simbiotics for his Undergraduate dissertation *Investigating programmable pattern formation in synthetic bacterial colonies*, where Copasi was again used alongside Simbiotics; Simbiotics is currently being used by the Newcastle 2018 iGEM team for a spatial chemotaxis model.

The application of the software outside its original scope is promising, however currently researchers who have used the software have had programming expertise. The ability for non-programmers (especially experimentalists who work *with* the systems that we informaticians model) to have access and use these tools is a challenge which has not yet been fully addressed. Existing work in this domain of multi-scale modelling has typically been done by Engineering faculties. The introduction of Easybiotics has aimed to remedy this, serving as an intuitive *in-silico* lab, providing features such as those seen in Chapter 6 which give non-programmers/engineers to engage in modelling. Easybiotics is currently in the pre-release stage, thus its use within the modelling community has not yet been determined.

A challenge facing the modelling and simulation community is in the reproducibility of results by third parties, even with a full description of how the original model worked [204]. For this reason it is crucial that formalisation is brought to the modelling and simulation practice, ensuring its reliability as a scientific instrument [202]. To address this, I introduced a file format encoding population models and methods for their reuse and communication (Chapter 5). A significant contribution provided by this format is that it allows for populations of interacting SBML models to be embedded in a physically realistic model, and encoded in a machine tractable file format. This file format also enabled

the externalisation of Simbiotics models and methods, providing an intermediate format for loosely interfacing with other software (such as Easybiotics).

The contribution of the software to the natural sciences, particularly the understanding of multicellular systems, is yet to be realised beyond some relatively basic models. The case study simulations and model findings presented in this thesis have been valuable for formally representing our knowledge of a system, and investigating the system dynamics once a sufficient model has been constructed as seen in the literature modules and case studies (Chapter 4.3 and Chapters 7-10). In all these cases, they call for more research to be done to realise novel aspects of biological systems.

11.3 Limitations

The methods developed in this thesis are a small step forward for the integrative modelling domain, and there are numerous limitations to the techniques presented. These are addressed here, and are decomposed into limitations regarding: modelling, usability, extendibility, performance and conceptual aspects.

With regards to the modelling, Simbiotics lacks in the modelling of some features which may be desirable to consider when developing multicellular models. One of these is that Simbiotics does not exclude cell volumes from the calculated volume of an extracellular voxel. This means that even if a voxel is full of cells, the extracellular diffusion algorithm does not account for this excluded volume when calculating fluxes, which may cause artefacts in models where there are large populations of densely packed cells interacting with chemical fields. Furthermore, currently Simbiotics does not have a fluid dynamics module (such as an implementation of the *Navier-Stokes* equations). This aspect is crucial for the physically realistic modelling of biofilms, and thus right a range of investigations into biofilms can not be conducted with the tool. Other key modelling limitations in the platform include: lack of extracellular chemical reactions; lack of 3D meshes to model more complex geometries; and lack of periodic boundary conditions for chemical diffusion. Due to these limitations the types of systems that can be modelled, and the types of questions that the model can answer, are currently constrained.

With regards to extendibility of the platform, some of Simbiotics' core implementation constrains the direction in which the software can be readily extended. The core implementation is as an Agent-based Model (ABM), where the agents exist in a continuous space that is then rasterized into subvolumes to represent localised chemical concentrations, with which agents can interact. The implementation of this makes it difficult for an alternative representation of physical cells to be added to the framework. For example, vertex models are a possible method to represent densely packed populations of cells such as tissues, however developing a vertex model and coupling it with the existing framework would require significant work. This is in contrast to implementing a new method for simulating gene regulation, or addition of fluid dynamics to the grid rasterisation, which both fit with the existing design pattern of the software.

With regards to usability, installation of Simbiotics and Easybiotics onto a users native system is currently done manually rather than by an installation program. Though Simbiotics and Easybiotics have minimal dependencies, this is still potentially technical work that has to be done by the user to get a working copy of the software. To address this we have released a virtual machine image with the software installed allowing for out-of-the-box use, however this method of distribution is not easily manageable as new versions of the are released. Virtual machines also complicate the users experience with the software.

In the cases that Simbiotics or Easybiotics crash (such as a corrupt model file, or an error in the execution) the error messages that are output to the user can appear cryptic. Additionally, even though the user is notified of basic things such as what the valid input type is for a parameter, there is currently no feature to inform the modeller if their parameter value is within a stable range for the integrator. Additionally if the Simbiotics integrators become unstable it is not apparent to the user. These issues can be confusing to a user, and may serve to undermine the trust a user has in the software.

With regards to performance, the Simbiotics platform utilises the Cortex3Dp module for multi-threaded and multi-CPU execution, enabling the platform to take advantange of high-performance computing clusters (HPCs). Simulations run on HPCs can capture large system sizes (100,000+ cells) for significant periods of time (1h+). Most potential users of Simbiotics would most likely be using it

natively on their machine (possibly a laptop) and thus gain no significant benefit from the current scheduler optimisations. Due to this, there is currently a bottleneck in regards to how large of a system you can run for a significant amount of time on a single machine, depending on the complexity and factors in the model.

The limitations of the work are underpinned by its conceptual design and target use. The platform was constructed to aid in the modelling of multicellular systems where the single cell is treated as a discrete individual that is free to move, and to have its own unique intracellular dynamics that may change throughout its lifespan. This enforces a level of abstraction on the modeller, and restricts how easily the software can be scaled up to simulate system sizes often observed in nature (over 10^9 cells), as cells have to be represented as individuals, rather than as clusters or motifs.

11.4 Future work

Outlook for this area of research is promising, with many research papers being published around the world in both the Systems and Synthetic biology domains. Simbiotics was designed taking into consideration this rapid growth and constant generation of new knowledge and modelling techniques, allowing it to be easily maintained as science progresses. Targeted future work aims to remedy the main limitations that have been identified in the methods presented here, and to keep to keep up with the changing domain.

In future work, we plan to expand the Simbiotics modelling library and virtual lab to accommodate for more cellular behaviours and ways of analysing model properties. To increase the platform's capabilities for biofilm modelling, the addition of a hydrodynamics module to the library is planned, as is the implementation of excluded cell volumes (such that the extracellular diffusion grid calculates it's volume considering that cells take up volume). Further more, we plan to integrate Simbiotics and Easybiotics with the Infobiotics Workbench 2.0 design suite for synthetic genetic designs [24], which also gives access to synthetic circuit model checking, and biocompilation.

To ease the usability of the platform a cross-platform installer is also to be developed, allowing for the non-technical installation of Simbiotics and Easybi-

otics without the need to use the virtual machine. This alongside a series of video tutorials should provide domain experts with the resources they need to engage in modelling. Additionally, a feature to enable the specification of parameter units is to be implemented in Simbiotics, allowing the user to more freely enter the parameter values of the system leaving Simbiotics to handle the conversion into the correct units. This feature could be propagated up to Easybiotics, signalling to the user what realistic parameter values are. Additionally the implementation of a feature which informs the user if the integrator becomes unstable, and if so which part is unstable, is to be reported to the user.

We plan to optimise the simulation execution scheduler, making it more powerful for running on single machines. This could be achieved by the implementation of a dynamic time-step solver would also be beneficial. This would offer some speed up for simulations by unconstraining the simulation of all process from the global timestep, and solving the steady state of relatively fast processes and allowing for larger timesteps for less time-sensitive processes.

Regarding future modelling work - now that we have a series of basic models constructed and trust in the tool, we plan on simulating more complex models of bacterial coaggregation and biofilm formation, where both physical and biochemical processes are considered. Further studies involve more collaboration between experimental and computational sides, striving for predictions to be made by *in-silico* models and tested experimentally.

11.5 Summary

In this thesis I have discussed state of the art methods for computationally modelling multicellular systems, establishing some key issues preventing those methods from being more readily integrated into the study of multicellular systems.

To address these issues, we have introduced Simbiotics, an integrative framework for modelling and analysing multicellular systems. The Simbiotics library of modelling tools allows for the flexible representation of multi-species cellular populations, integrating previously distributed state-of-the-art methods into a common platform. Easybiotics enables domain expert such as microbiologists and chemical engineers to use Simbiotics with minimal programming experience, over-

coming a challenge I found presented to me regularly by many experimentalists at conferences who wished to model their systems. Additionally, a light-weight and versatile data format has been introduced for representing and sharing multi-cellular population models, encapsulating SBML (an existing biomodel standard) and building on such formalisations for denoting biological systems.

Case study findings have indicated that Simbiotics can be a valuable tool when studying emergent multicellular dynamics, both in the studying of natural systems and in the exploration of design spaces for novel synthetic devices. Model development can verify that our understanding of the experimental system is correct, and explain the driving forces behind population behaviour. It can also expose discrepancies between the real system and the simulated one, thereby revealing areas of insufficient system understanding. My thesis further sets the stage for more complex models to be developed of multicellular systems, studying how cell internal dynamics and population level organisation are connected.

Appendix A - Simbiotics user guide

This appendix contains the Simbiotics user guide. The mainual contains information on how to get and install Simbiotics, tutorials on how to build models, as well as a tutorial on how to add new features (by programming new library modules). The manual also elaborates on the software usage and content.

.1 Introduction

Welcome to the Simbiotics user guide! In these guide we'll take you through what the software does, how to install it, and how to use it. In brief Simbiotics is a java simulator which lets you construct models of multicellular systems, primarily populations of mixed bacterial species. Simbiotics can be used via a graphical user interface called Easybiotics. Once you have installed Simbiotics, you can go over to the *easybiotics_guide.pdf* if you wish to use that for model building/analysis.

You can also try Simbiotics in a Virtual Machine for easy out-of-the-box use, it can be found on the website along with video tutorials on how to use the software.

https://bitbucket.org/simbiotics/simbiotics/wiki/Home

Overview

Simbiotics is a 3D modelling platform which allows for the design, simulation and analysis of multicellular systems. The platform is focussed on modelling of bacterial populations, allowing for the representation of unique cellular species, where their individual behaviour and interactions can be defined. Through this one can simulate the emergent behaviours exhibited by the population, arising from the

interplay between micorprocesses such as individual cell's genetic regulation, and macroscale processes such as dynamic spatial arrangement.

Simbiotics provides a standard modelling library for simulating typical processes of bacteria, such as growth, motility, gene regulation, metabolic activity and cell-surface appendages (receptors and adhesins). The library also provides some models of environmental factors such as a fluid mixing force, bouyancy/-gravity, friction and a primitive flow chamber.

Additionally one can attach virtual devices to a model in order to probe or interact with it, allowing for a partial virtual lab experience. Data exporters can also be attached to a model in order to collate, format and write data to file. The inclusion of auxiliary programs to model specifications allows for initial conditions, repetitive tasks and desired interactions with the model to be automated.

In order to design a model in Simbiotics, modules from the Library can be attached to a model. This means that the modeller can fine tune the simulation content, designing a specification in a compositional manner in order to build bacterial and environmental models. In addition this means that only processes relevant to a model will be simulated.

License

Technical overview

Simbiotics is written in Java 1.7. It utilises the spatial representation and parallelised scheduler as implemented in Cortex3Dp [238]. Simbiotics was developed using LibSBML 5.13 and LibSBMLSim 1.3.

Terminology

To clarify some of the terminology used in this document, we list some keywords and their meaning.

Term	Meaning
Library module behaviour	Java classes within the Simbiotics library which describe model-specific
$SIMBIOTICS	The main Simbiotics folder, which contains the *src* folder

Table 1: User manual terminology

.2 Getting Simbiotics

Simbiotics

Simbiotics is available at:

https://bitbucket.org/simbiotics/simbiotics/wiki/Home

Simbiotics is developed in Java 1.7, you must have the following installed:

- Open JDK >= 6 (GNU General Public Licence + classpath exception) or Oracle Java SE >= 6 (Oracle Binary Code Licence)

And optionally, if you wish to use SBML integration, you must have:

- libSBML - http://sbml.org/Software/libSBML (GNU LGPL)
- libSBMLSim - http://fun.bio.keio.ac.jp/software/libsbmlsim/ (GNU LGPL)

If you aren't using SBML integration you can skip to **3. Running Simbiotics** section.

Getting Dependencies

Simbiotics can be used in its minimal form without any dependencies, however if you wish to use the SBML integration you must have the following software packages installed, and linked to the project.

LibSBML

The first dependency is the Systems Biology Markup Language (SBML) library. SBML is a language format for representating computational models of biological processes, such as the metabolism and gene regulation of individual cells. Simbiotics can handle SBML through these libraries to represent populations of SBML

models interacting with each other. LibSBMLj is a java interface for the SBML format, and can be downloaded at the link below:

Download LibSBML

LibSBMLsim

The second dependency is LibSBMLsim, a simulator which is used to run SBML models. It is written in C++ and LibSBMLsimj is the java interface for using the library, and can be downloaded from the link below:

Download LibSBMLsim

Linking dependencies

Once they are installed on your system (make sure you have the correct ones for your operating-system and cpu-architecture), locate *libsbmlj.jar/libsbmlsimj.jar* and *libsbmlj.so/libsbmlsimj.so* files on your system, and copy them into the *$SIMBIOTICS/jars* folder (overwrite any existing versions which are in that folder).

Compiling Simbiotics

The supplied Simbiotics source code must be compiled to an executable jar if you wish to run it from command-line. This is a simple one stage process; from command-line enter the $SIMBIOTICS root folder and run the *make* command, as such:

Listing 1: Compling Simbiotics source to executable jar

```
cd $SIMBIOTICS
make
```

.3 Running Simbiotics

Simbiotics can be run in multiple ways allowing the user to choose which is most appropriate for them. Simbiotics can either by run via command-line, an IDE, or Easybiotics (see easybiotics_guide.pdf).

Using Simbiotics by command-line

Simbiotics can be launched by command line using the *jar* file, as seen below.

Listing 2: Minimal use of jar file

```
cd $SIMBIOTICS
java -jar simbiotics.jar
```

A specific configuration file may also be provided as a command line argument. The configuration file (config) contains the launch parameters for the software. Example configuration files can be found in *$SIMBIOTICS/examples/configs*. The configuration file is described in Section 6.1.

There are a range of command line parameters (arguments) for simbiotics to supply other information to simbiotics:

Listing 3: Parameters

```
-config      the configuration file
-model       the model file
-parameters  the model parameter file
-results     the target results directory
```

If a parameter which is already in the configuration file is provided as a command-line argument, it will override the value specific in the config.

Below are some other examples of launching Simbiotics from command-line.

Listing 4: Loading a model

```
#setting a custom java model
java -jar simbiotics.jar -config configs/default.json -model
    simbiotics.examples.Model1_Aggregation
```

```
#setting a custom java model and custom results directory
java -jar simbiotics.jar -config configs/default.json -model
    simbiotics.examples.Model1_Aggregation -results my_results/

#setting a custom JSON model and custom results directory
java -jar simbiotics.jar -config configs/default.json -model
    examples/models/1_aggregation.json -results my_results/
```

Using an Integrated Development Environment (IDE)

For this user manual the IDE we will use is Intellij 14.1, which can be downloaded at the link below.

Download Intellij 14.1

The following steps are how to open the project in Intellij, version 14.1 was used for this user guide.

1. File - New - Project from Existing Sources
2. Select the *simbiotics* main folder.
3. Create project from existing sources
4. Name the project
5. Make sure the *simbiotics* src folder path is selected
6. Make sure the libraries are selected
7. Finish

The dependencies may need to be manually linked in the IDE.

1. Navigate to File - Project Structure... (Ctrl+Shift+Alt+S)
2. Click on the Libraries tab on the left
3. Click New Project Library (Green +)
4. Choose Java
5. Navigate to the *$SIMBIOTICS/jars* folder
6. Choose one of the *.jar* or *.so* files
7. Choose to add it to the *simbiotics* module
8. Repeat this for all of the files in *$SIMBIOTICS/jars* (both *.jar* and *.so* files)

You can test that Simbiotics is running correctly by navigating the one of the example models, such as *"src simbiotics examples Model1_ Aggregation.java"* and run the java application with that class as the main entry.

Once you have verified that you can launch Simbiotics from the IDE, please see the section below entitled "Developing Simbiotics models in Java" for tutorials on how to build models.

.4 Live visualisations of simulations

To run simulations with a live visualistion, set the ***gui*** variable in the configuration file to be ***true***. This loads the Simbiotics GUI, which renders a 3D scene which can be navigated with a camera. It also provides a tool bar with additional functions which are described below.

Visualisation layers

The renderer can be set to only display certain layers of the simulation.

Functions

Functions can be performed such as running a spectrophotometer scan on the system to take an optical density measurement.

Options

In the options menu you can pause/unpause the simulation. Additionally you may allocate more CPU threads.

View

The camera position can be modified/reset here.

Window

Popup windows can be shown to show the details of the simulation.

Recording

Both images and videos may be taken of the visualisation. Images work in the same way as having a *geometry_image* exporter attached to the model specification - it writes the properties of all the geometries in the simulation to a file.

Video recording generates an *.avi* which can be found in the *$SIMBIOTIC-S/results* folder.

.5 Developing Simbiotics models in Java

To illustrate how to build Simbiotics models in Java, we run through some basic examples. The first tutorial is a step-by-step overview of how to create a basic model, withe the following building upon those ideas to develop more complex models.

Tutorial 1 - Creating your first model

In this first tutorial we will describe how to construct a basic model, followed by how to attach some library modules to describe model functionality and perform basic analysis and data collection.

The complete model can be found in the Simbiotics project at:
simbiotics.examples.Tutorial1_AggregationOpticalDensity

Creating a model class

First we define a new model class which extends Model. Make sure it this new class is in the Simbiotics source code folder. This class needs two functions to work, a Java *main* method to so you can start the simulation from the model class, and a *build* method in which the model specification definitions are. The *main* method should have a call to the *initialise* function, and should pass the *.class* variable of the model you are defining. The *build* method contains the model specification, and is used by calling desired *define* functions and passing in modules from the Simbiotics library. Additionally one may override the *prestep* and *poststep* methods, which are called before/after solving each iteration of the simulation, and can used for direct injection of commands as the simulation runs..

```
// define a new class which extends Model
public class MyModel extends Model {

   // define a main method in which this objects static class
      variable is passed into the initialise function
   public static void main(String[] args){
```

```
        initialise(MyModel.class);
    }

    // override the Model build method
    public void build(){
        // model definitions go here
    }

    // optionally override the Model prestep method
    public void prestep(){
        // custom modeller definitions
    }

    // optionally override the Model poststep method
    public void poststep(){
        // custom modeller definitions
    }
}
```

In the *build* function, the modeller is required to define the simulation domain size (world size). This is shown below where a world of size 100*50*100 micrometers is specified.

We also define boundary conditions which describe the behaviour at the domain boundaries. Here we set the X and Z axes to be cyclical (periodic) boundaries, such that agents which leave a face of the cuboidal domain on the X and Z axes enter from the opposing face of the domain. By default boundary conditions are set to be solid walls, in this case the Y axis (top and bottom faces of the cube) are impassable.

```
// define the world domain to be 100*50*100 micrometers (in form {x,
    y, z})
defineWorldSize(100, 50, 100);

// define the world X and Z boundaries to be cyclical
defineBoundary(Axis.X, new CyclicalBoundary());
```

```
defineBoundary(Axis.Z, new CyclicalBoundary());
```

Three solver systems for the model are required, namely the physical intergration solver, reaction-diffusion solver and the goemetry collisions solver. You may use different libraries modules for these, if none are loaded them the default solvers (standard solvers) are used. This is shown below, where we use the default library modules for each of the solvers.

The *StandardPhysics* module implements a verlet integrator which describes how forces are translated into velocities and positions for agent geometries. The *StandardDiffusion* module implements a finite-volume method of Fick's Law for solving the diffusion of chemicals in the world domain. The *StandardCollisions* module implements a mass-spring law to describe how intersecting agent geometries exert forces on each other.

```
// define the physics solver (StandardPhysics implements verlet
    integration) and add force components
definePhysics(new StandardPhysics());

// define the diffusion solver (StandardDiffusion implements a
    finite-volume method of Fick's law)
defineDiffusion(new StandardDiffusion());

// define the collision solver (StandardCollisions implements a
    mass-spring system)
defineCollisions(new StandardCollisions());
```

The modeller can define cell species using a *CellSpecies*, which describes the name and functionality of the species. Below we define two species, *"species_a"* which is red and is represented as a sphere of diameter 0.9 micrometers, and *"species_b"* which is green a sphere of 1.1 micrometers.

Populations of the two species are then defined, 300 *"species_a"* cells and 200 *"species_b"* cells by creating an initial condition.

```
// define the coccus morphology (spherical geometry)
```

```
defineMorphology(new CoccusMorphology(0.5), "coccus");

// define two species of cells
defineCellSpecies(new CellSpecies("species_a", Color.RED, "coccus");
defineCellSpecies(new CellSpecies("species_b", Color.GREEN, "coccus");

// define a population of the species
defineInitialCondition(new InitialPopulation("species_a", 300),
    "initial_species_a");
defineInitialCondition(new InitialPopulation("species_b", 200),
    "initial_species_b");
```

Loading the model in its current state results in a static scene with the inanimate cell populations suspended in the domain. This is the first step of building a typical model, providing the core components on which model functionality will be layered.

Extending the model

Defining environmental forces is done via the physics solver system. The *StandardPhysics* module can take a set of force component parameters, which describe the forces equations due to specific mechanisms. Force components are found in the *simbiotics.library.physics.components* package.

We define two force components, Brownian dynamics and friction dynamics, with force coefficients passed into their constructors.

```
// define the physics solver (StandardPhysics implements verlet
    integration) and add force components
definePhysics(new StandardPhysics(new Brownian(2.4), new
    Friction(2)));
```

Binding sites can also be used to represent targets for interactions, typically representing cell surface proteins and carbohydrates. We define two binding sites *"adhesin_a"* and *"adhesin_b"*. We then define an interaction called *"interaction_a_b"* which occurs between the two spcies of adhesin. An *InteractionTem-*

plate describes interaction parameters, here we set the interacton force coefficient to be 40 and the interaction rate to be 30.

```
defineBindingSite(new BindingSite("adhesin_a"));
defineBindingSite(new BindingSite("adhesin_b"));

// define the interaction and its mechanismwhich occur between
    adhesins
defineInteractionMechanism(new SpringMechanism(40, 30), "spring")
defineInteraction(new SpecificInteraction("interaction_a_b", new
    Pair("adhesin_a", "adhesin_b"), "interaction1"));
```

Now we have defined binding sites which have an interaction between them, we can add the binding sites our cell species definitions, this is achieved via adding a behaviour library module to the species. Below we define two behaviour modules, both instances of *CellAdhesion* which is a module implementing how cells detect binding site interactions with neighbouring cells. This module takes a parameter list of Strings, being the IDs of the binding sites which are present in that module. For our modules *"adhesion_ a/b"* have their corresponding adhesin as their constructor parameter.

We then modify the cell species definitions we defined earlier; cell templates can take a parameter list of Strings after the cell geometry (sphere) parameter, these are the IDs of the behaviour modules as we defined above. Cell species *"species_ a/b"* have their corresponding cell adhesion behaviour module attached to their definition, *"adhesin_ a/b"* are then implicitly represented on the surface of *"species_ a/b"*.

```
// define the cell behaviour module which implements cell-adhesin
    functionality
defineCellBehaviour(new CellAdhesion("adhesin_a"), "adhesion_a");
defineCellBehaviour(new CellAdhesion("adhesin_b"), "adhesion_b");
...

// add the new behaviour modules to the cell species templates using
```

```
    their unique keys
defineCellSpecies(new CellSpecies("species_a", Color.RED, "coccus",
    "adhesion_a");
defineCellSpecies(new CellSpecies("species_b", Color.GREEN, "coccus",
    "adhesion_b");
```

Binding stes can be used to define environmental structures such as binding targets on solid boundaries. We define a binding site called *"boundary_structure"*, and an interaction *"boundary_interaction"* which occurs between *"adhesin_a"* and the new boundary structure with a force coefficient of 100 and a rate of 100.

We then define a boundary condition on the Y axis, at the face of the cube where the Y coordinate is the maximum of the world domain (in Simbiotics Y max is the top face of the cuboid domain). The boundary is set to be a solid wall, and has a property object assigned to. In the property object we defined property called *"structures"*, which takes a String array of the binding sites which are present, in this case only the new binding site *"boundary_structure"*.

```
// define the new environmental binding site
defineBindingSite(new BindingSite("boundary_structure"));
...

// define the interaciton between species_a's adhesin, adhesin_a, and
    the environmental_structure
defineInteractionMechanism(new SpringMechanism(100, 100), "spring_2")
defineInteraction(new SpecificaInteraction("boundary_interaction",
    new Pair("adhesin_a", "boundary_structure"), "spring_2"));
...

//define the world Y boundaries to be solid, and the top substratum
    has a surface structure which interacts with species_a
defineBoundary(Axis.Y, AxisFace.MAX, new SolidBoundary(new
    BoundaryData(new Pair("structures", new
    String[]{"boundary_structure"}))));
```

Collecting data from the model

To collect data from the model we can define exporters, these are library modules which read desired model state information and writes it to file. Additionally the modeller can define devices, which are programs that perform built-in analysis on the model state such as measurements or interactions with the model, device data can then be used by exporters.

For this model we can measure the aggregation of the bacterial population using a simulated spectrophotometer, emulating the process a biologist would go through to acquire such data. We first define the spectrophotometer module, then an exporter module which uses the data from this spectrophotometer. This is achieved by using the ID of the spectrophotometer in the constructor of the exporter. We take a spectrophotometer scan and export the data every 10 seconds, this sample period is the second parameter to the exporter.

```
// define the optical density device
defineDevice(new Spectrophotometer(), "spectrophotometer");

// define the optical density
defineExporter(new SpectrophotometerExporter("spectrophotometer",
    10), "od600_data");
```

Tutorial 2 - Biofilm

In this tutorial we will develop a more advanced model, building on concepts we covered in the first tutorial. We first develop a primitive single species biofilm model, where planktonic cells can colonise a surface. We then extend the model, introducing a second bacterial species which performs chemotaxis towards a chemical which is produced by the first species biofilm, resulting in the second species adhering the the biofilm. Growth kinetics are introduced, as well as a boundary interface which describes a flux of new chemicals and bacteria into the world domain. Analysis is then performed to measure the biofilm height profile and this data is written to file.

The complete model can be found in the Simbiotics project at:
simbiotics.examples.Tutorial2_BiofilmHeight

Environment setup

We first define a world domain size of 100*50*100 micrometers followed by definition of cyclical (periodic) boundaries on the X and Z axes, as we did in the first tutorial. The domain boundary at the minimum value of the Y axis (bottom face of the cuboid domain) is then set to be solid with binding sites present.

We then define the solver systems for the physics, diffusion and collisions in the model. The physics system has three force components, namely forces due to gravity, Brownian dynamics and friction (viscous drag force).

```
// define a world domain of 100*50*100 micrometers
defineWorldSize(100, 50, 100);

// define the world X and Z boundaries to be cyclical
defineBoundary(Axis.X, new CyclicalBoundary());
defineBoundary(Axis.Z, new CyclicalBoundary());

// define the world Y boundaries to be solid, and the top substratum
    has a surface structure which interacts with species_a
```

```
defineBoundary(Axis.Y, AxisFace.MIN, new SolidBoundary(new
    BoundaryData(new Pair("structures", new
    String[]{"boundary_structure"}))));

// define the boundary structure binding site
defineBindingSite(new BindingSite("boundary_structure"));

// define the physics solver (StandardPhysics implements verlet
    integration) and add force components
definePhysics(new StandardPhysics(new Gravity(0.1), new
    Brownian(2.4), new Friction(2)));

// define the diffusion solver (StandardDiffusion implements a
    finite-volume method of Fick's law)
defineDiffusion(new StandardDiffusion());

// define the collision solver (StandardCollisions implements a
    mass-spring system)
defineCollisions(new StandardCollisions());
```

Bacterial species

A bacterial species is then defined; it's represented as a red sphere of diameter 0.9 micrometers, and has a binding site *"adhesin_a"* on its surface which may interact with the *"boundary_structure"* binding site. We then create 100 instances of the species.

```
// define the binding site
defineBindingSite(new BindingSite("adhesin_a"));

// define the interaction between species_a's adhesin (adhesin_a),
    and the boundary
defineInteractionMechanism(new SpringMechanism(100, 100),
    "spring_mechanism")
```

```
defineInteraction(new SpecificInteraction("interaction_a_boundary",
    new Pair("adhesin_a", "boundary_structure"), "spring_mechanism"));

// define the cell behaviour module which implements cell-adhesin
    functionality
defineCellBehaviour(new CellAdhesion("adhesin_a"), "adhesion_a");

// define the morphology
defineMorphology(new CoccusMorphology(0.5), "sphere");

// define the cell species
defineCellSpecies(new CellSpecies("species_a", new Color.RED,
    "sphere", "adhesion_a"));

// define cell population
defineInitialCondition(new InitialPopulation("species_a", 100),
    "initial_species_a");
```

Multiple bacterial species

To develop the biofilm model further we introduce a second species. We define *"species_b"*, which is represented by a blue sphere of diameter 1.1 micrometers, it has a binding site *"adhesin_b"* on its surface which may interact with *"adhesin_a"* on *"species_a"* cells.

```
// define the binding site
defineBindingSite(new BindingSite("adhesin_b"));

// define the interactions which occur between adhesins
defineInteractionMechanism(new SpringMechanism(50, 50),
    "spring_mechanism_2")
defineInteraction(new SpecificInteraction("interaction_a_b", new
    Pair("adhesin_a", "adhesin_b"), "spring_mechanism_2"));
```

```
// define the second morhology
 defineMorphology(new CoccusMorphology(0.65), "larger_sphere");

// define the second cell species
defineCellSpecies(new CellSpecies("species_b", Color.BLUE,
    "larger_sphere", "adhesion_b"));

// define second cell population
defineInitialCondition(new InitialPopulation("species_b", 50),
    "initial_species_b");
```

Bacterial growth

We use two forms of bacterial growth in this model. The first is a constant growth module which is not dependent on any factor, the second is a nutrient dependent growth which depends on an extracelluar nutrient. In order to represent an extracellular nutrient which undergoes reaction-diffusion dynamics, we must define the diffusion grid resolution and chemical species.

To define the diffusion grid resolution we pass a value of 3 to the StandardDiffusion constructor, this means a binary split will be recursively performed on the cuboidal domain 3 times. For our domain size of 100*50*100 micrometers, 3 binary splits mean our diffusion voxel resolution is 12.5*6.75*12.5 micrometers.

We then define the *"substance_b"* chemical which represents the nutrient, it has a diffusion rate of 50 and a degradation rate of 0.5.

We also define a *"chemotaxis"* behaviour module, which describes motility dynamics in order to ascend a chemical gradient. We set the chemoattractant to be *"substance_b"*.

```
// define the diffusion solver (StandardDiffusion implements a
    finite-volume method of Fick's law) and an integer of how many
    binary divisions to preform on the world domain
defineDiffusion(new StandardDiffusion(3));

// define substance_b with its diffusion and degradation rates
```

```
defineChemicalSpecies(new Chemical("substance_b", 50, 0.5));

// define species_b's oxygen chemotaxis module
defineCellBehaviour(new Chemotaxis("substance_b", 50, 50, 50),
    "chemotaxis");
```

We define two forms of growth in the model. For *"species_a"* a constant growth module is used, which has a growth rate of 0.0004 ± a variation of $0.0004 fgs^{-1}$.

For *"species_b"* a nutrient dependent growth module is used. We first create a reaction called *"growth_reaction"*, defining its as non-autocatalytic, then setting the maximum growth rate and reaction yield coefficient. We then add a kinetic factor describing the form of the reaction, using a MonodKinetic we set the depending substance to be *"substance_b"* and the half-saturation value to be 0.5 We then create a ReactionKineticGrowth behaviour module and attach the growth reaction we had defined. Then we set the stoichiometric yield coefficients of the reactants and products in the reaction. We set the yields to be *"substance_b"* decreasing by one unit as the cells *"biomass"* increases one unit.

Cells will divide (undergo mitosis) upon reaching twice the diameter they were at birth.

```
// define the species_a's constant growth module
defineCellBehaviour(new ConstantGrowth(0.0004, 0.0004), "growth_a");

// define the reaction kinetics for substrate-dependent growth
KineticReaction growth_reaction = new
    KineticReaction("growth_reaction");
growth_reaction.setAutocatalytic(false);
growth_reaction.setMaxRate(0.001);
growth_reaction.setYield(1.0);
growth_reaction.addKineticFactor(new MonodKinetic("substance_b",
    0.5));

// define species_b's substance dependent growth module
ReactionKineticGrowth dependent_growth = new ReactionKineticGrowth();
dependent_growth.addReaction(growth_reaction);
```

```
dependent_growth.addYield("substance_b", -1.0);
dependent_growth.addYield("biomass", 1.0);
defineCellBehaviour(dependent_growth, "growth_b");
```

The new modules must then be added to the cell species definitions by their IDs. We modify the *"species_a"* definition to add the constant *"growth_a"* module, and modify *"species_b"* to have the nutrient-dependent *"growth_b"* module and *"chemotaxis"* module.

```
// define the cell species
defineMorphology(new CoccusMorphology(0.45), "sphere1");
defineCellSpecies(new CellSpecies("species_a", Color.RED, "sphere1",
    "adhesion_a", "growth_a")
);
defineMorphology(new CoccusMorphology(0.6), "sphere2");
defineCellSpecies(new CellSpecies("species_b", Color.BLUE, "sphere2",
    "adhesion_b", "growth_b", "chemotaxis")
);
```

Bacterial differentiation

To introduce bacterial differentiation to model we can embed some decision making into the cells. A cell can be in a set of discrete states, which can be turned on/off based on local environment factors. For this tutorial we represent this decision making at a high level of abstraction by using a single state, indicating whether the cell has adhered to the substratum. These states then effect the behaviour that the cell has, changing the way it interacts with its environment.

First, we will set up some cell behaviours which can be turned on when the cell attaches to the substratum. A secretor will be turned on which secretes *substance_b* at given rate. Extracellular-polymeric substances (EPS) also start being produced, EPS are represented as soft spheres.

```
// define the secretor which species_a has to secrete substance_b
defineCellBehaviour(new Secretor("substance_b", 100),
```

```
    "secrete_substance_b");

// define the species_a's constant growth module
defineCellBehaviour(new SecretingCapsule(0.002, 0.002, 0.05),
    "secreting_capsule");
```

Secondly we set the states of the species, in this instance both have one state *"SESSILE"* which is true if the cell is attached to the surface.

Links are set up, which connect cell behaviours to cell states. For both *"species_a/b"* there is a *BiofilmSensor* link, which connects their *"adhesion_a/b"* to the *"SESSILE"* state, setting the state to be true if the cell has adhered to the substratum (boundary structure) or to a cell is already sessile.

If a *"species_a"* cell is sessile it has the following behaviour:

- Turns on secretion of *"substance_b"* (StateToBehaviourLink)
- Increases its growth rate and variation (VariableChanger)

If a *"species_b"* cell is sessile is has the following behaviour:

- Turns on secretion of EPS (StateToBehaviourLink)
- Decreases its chemotaxis propel speed (VariableChanger)

```
// define the cell states
States states_a = new States();
states_a.add("SESSILE", false);
States states_b = new States();
states_b.add("SESSILE", false);

// define links
Links links_a = new Links();
links_a.add(new BiofilmSensor("adhesion_a", "SESSILE"));
links_a.add(new StateToBehaviourLink("SESSILE",
    "secrete_substance_b"));
links_a.add(new VariableChanger(new Pair("SESSILE", "growth_a"), new
    Pair("growth_rate", 0.00125)));
```

```
links_a.add(new VariableChanger(new Pair("SESSILE", "growth_a"), new
    Pair("deviation", 0.0005)));
Links links_b = new Links();
links_b.add(new BiofilmSensor("adhesion_b", "SESSILE"));
links_b.add(new StateToBehaviourLink("SESSILE", "secreting_capsule"));
links_b.add(new VariableChanger(new Pair("SESSILE", "chemotaxis"),
    new Pair("run_force", 1)));
```

We must then attach the newly defined behaviours, states and links to the cell species definitions, modify the original definitions.

```
// define the cell species
defineCellSpecies(new CellSpecies(
   "species_a", Color.RED, states_a, links_a, new Sphere(0.9),
   "adhesion_a", "growth_a", "secrete_substance_b")
);
defineCellSpecies(new CellSpecies(
   "species_b", Color.BLUE, states_b, links_b, new Sphere(1.2),
   "adhesion_b", "growth_b", "chemotaxis", "secreting_capsule")
);
```

Chemostat and bactostat

We define a flux of new bacteria and chemicals into the system. This is achieved by defining a chemostat (for chemical fluxes) and a bactostat (for bacterial fluxes), and assigning them an environment interface which describes which domain boundary they operate on.

Below we define two lists of Fluxes, one for chemicals representing a flux of acid into the system, and one for bacteria representing the flux of the two species into the domain. Flux declarations have the flux rate as their second parameter.

For chemicals we have flux of *"substance_b"* at a rate of 0.01 $\mu M s^{-1}$ μm^2. For bacteria we have a flux of *"species_a"* at a rate of 0.6 cells s^{-1}, and of *"species_b"* at 0.4 cells s^{-1}.

We then define an environment interface, describing which domain boundary

this flux occurs at. Here we specific that the *MAX* boundary of the *Y* axis is where the fluxes occur, meaning that cells and chemicals are introduced from the top face of the cuboid simulation domain.

We then define the two devices, a Chemostat and a Bactostat, passing their constructors the corresponding fluxes and the target environment interface. They are also identifiable by their unique device IDs, *"chemostat"* and *"bactostat"*.

```
// set up the fluxes used for the chemostat
ArrayList<ChemicalFlux> chemical_flux = new ArrayList<>();
chemical_flux.add(new ChemicalFlux("substance_b", 0.001));

// set up the fluxes used for the bactostat
ArrayList<BacterialFlux> bacteria_flux = new ArrayList<>();
bacteria_flux.add(new BacterialFlux("species_a", 0.6));
bacteria_flux.add(new BacterialFlux("species_b", 0.4));

// define the environment interface
EnvironmentInterface environment_interface = new
    EnvironmentInterface(Axis.Y, AxisFace.MAX)

// define up the chemostat and bactostat devices with their
    respective fluxes
defineDevice(new Chemostat(chemical_flux, environment_interface),
    "chemostat");
defineDevice(new Bactostat(bacteria_flux, environment_interface),
    "bactostat");
```

Biofilm height measurements

To analyse the model we take measurements of the biofilm height. This gives us both the average and standard deviation of the biofilm height, as well as a 2D heatmap which encodes the biofilm height profile.

First we define the biofilm height measuring device which samples the height of the biofilm across the entire world domain. Its scan resolution is defined in its con-

structor by as X and Z resolution, here we set that resolution to be 2 micrometers on both the X and Z axes. We give it a device ID of *"biofilm_height_measurer"*.

We then define a data exporter specifically for this device. We pass the ID of the device we defined above to instruct the exporter to use data collected from this device. The second parameter is the sample period of data collection, it's set to export the data every 25 seconds. The exporter unique ID *"biofilm_height_data"* is the name of the file which will hold this default, it can be found in the results directory which is defined in the Simbiotics configuration.

```
// define the biofilm height measuring device
defineDevice(new BiofilmHeight(2, 2), "biofilm_height_measurer");

// define the biofilm height exporter
defineExporter(new BiofilmHeightExporter("biofilm_height_measurer",
    25), "biofilm_height_data");
```

.6 Input/output

Configuration file

The configuration file is the first argument when loading Simbiotics from command-line, it is the only compulsory argument. It describes the parameters for Simbiotics which can be seen in Table 2 below. When developing in an IDE, the configuration parameters exist in the SimbioticsConfig class.

Listing 5: Simbiotics configuration file

```
{
  "model_file": "simbiotics.examples.Model1_Aggregation",
  "results_dir": "results/",
  "duration": 0,
  "simple_workers": 1,
  "complex_workers": 4,
  "max_nodes_per_pm": 20000,
  "node_depth": 0,
  "slot_resolution": 20,
  "balance_round": 300
  "verlet_update": 10,
  "view_width": 1280,
  "view_height": 800,
  "parallel": true,
  "profiling": false,
  "gui": true
}
```

Keyboard/mouse interactivity

There are some default key bindings provided in Simbiotics. These can only be run when the Simbiotics GUI is also loaded ($gui = true$ in configuration file).

Parameter	Description
model_file	The path to the model class/file to be simulate
results_dir	The default results directory for data exportin
duration	Number of simulated seconds before exiting, 0 means
simple_workers	Number of simple worker threads
complex_workers	Number of complex worker threads
max_nodes_per_pm	Number of agent geometries in partition before it is split int
node_depth	Number of binary splits of the cuboid domain into the diffusion
slot_resolution	Number of voxels in each subpartition
balance_round	Number of iterations before the domain is checked if it should be s
verlet_update	Number of iterations before updated a cells verlet list (near
view_width	Width of the GUI frame in pixels
view_height	Height of the GUI frame in pixels
parallel	Whether the simulation should be run in a parallelized
profling	Whether the simulation profiling data should be dis
gui	Whether the simulation should be run with a G

Table 2: Simbiotics configuration parameters

Input	Action
Left click + drag	Translates the model visualisation
Right click + drag	Rotates the model visualisation
‘	Toggles pause
a	Saves data for all exporters
q	Takes a 3D snapshot of all geometric agents (for post rendering)
Spacebar	Toggles the colour scheme

Table 3: Simbiotics input commands

Inputs

Microscopy images

Microscopy images can be processed and loaded into Simbiotics to specify the initial spatial arrangement of bacteria. This is achieved by using the *MicroscopyLoader* class, which is available in the *simbiotics.loader* package.

Calling the *generatePopulation* function requires 3 parameters, in the following form:

Listing 6: Loading microscopy images into Simbiotics

```
MicroscopyLoader.generatePopulation(image_file, image_dimensions,
    world_dimensions);
```

image_file is a csv file encoding the microscopy image. *image_dimensions* is 3D double array containing the size of the image file in pixels. *world_dimensions* is the size the image will be scaled down to in the simulation.

Listing 7: Loading microscopy images into Simbiotics

```
PopulationEncoding my_population =
    MicroscopyLoader.generatePopulation("encoding.csv", new
    double[]{1024, 1024, 1024}, new double[]{92, 92, 92});

definePopulation(my_population);
```

SBML models

SBML models can be embedded in agents in Simbiotics. This is achieved by using the *SBMLModule* behaviour class, which is available in the *simbiotics.library.behaviour.sbml* package.

Listing 8: Loading SBML models into Simbiotics

```
SBMLModule my_sbml = new SBMLModule("my_sbml_file.xml", 1, 0.1);
```

The SBML module can then be defined as a behaviour and attached to a cell species definition as such:

```
defineCellBehaviour(my_sbml, "sbml_metabolism");
defineCellSpecies(new CellSpecies("my_species", Color.BLUE, new
    Sphere(1.0), "sbml_metabolism"));
```

Outputs

Data exporting

Data exporters output files to the $results_dir$ defined in the configuration file, unless their $file\_path$ variable is set, in which case that specific exporter outputs data to that folder, whilst the results outputs to the main results directory. Simulations also copy a version of the model used to run the simulation.

.7 Modelling library

world

2D_world 2D simulation domain
3D_world 3D simulation domain

boundaries

solid solid domain boundary for agents
cyclical cylical domain boundary for agents
no_return no return domain boundary for agents

surface_properties

adhesive adhesive structure on a solid domain boundary

forces

gravity force of gravity on agents (-Y axis force)
brownian force of brownian motions agents (random walk)
friction force of friction on agents (drag force)
interactions_force force of interactions on agents (the defined specific interactions)
collisions force of collisions between only spherical (coccus) agents
collisions_extended force of collisions between only rod-shaped (bacillus) agents
collisions_complete force of collisions between mixed spherical (coccus) and rod-shaped (bacillus) agents
collisions_hertzian force of collisions modelled as hertzian interaction between agents
non_specific force of non-specific interactions (van der Waals and electrostatic appropximation)
dlvo force of non-specific interactions according to DLVO theory

chemicals

chemical a chemical that can diffuse in the extracellular space
intracellular_only_chemical a chemical that can only exist in an intracellular compartment

interactions

specific_interation a specific interaction between two binding sites on agents surfaces

interaction_mechanisms

spring_mechanism a specific interaction is modelled as a Hookian spring forming between the interacting agents

states

state a qualitative intracellular state (boolean)
quantitative_state a quantitative intracellular state (continuous value)

links

state_to_state connect two states, such that state 2 always updates to be state 1's value
state_to_behaviour connect a state and a behaviour, such that the behaviour's activity (on or off) is equal the the states boolean value
behaviour_to_state connect a state to a behaviour, such that the states boolean value is equal to the behaviours activity variable (on or off)
state_is_external_concentration connect a state (quantitative state) to an external chemical, such that the states value is equal to the extracellular concentration of that chemical
surface_sensor connect a state to a surface sensor, such that the state is true if the agent is interacting with a solid boundary

conditions

general_condition define a custom condition
concentration_threshold check if a chemical concentration in relation to a set threshold
touched_cell check if the agent is touching a cell of a specific species
world_time check if a certain duration of global simulation time has elapsed
concentration_vs_concentration check the concentration of two chemicals against each other
has_interactions check if the cell has an interaction of a specific type
touched_surface_with check if the agent is touching a boundary with a specific surface property

actions

general_action define a custom action
change_state change the value of a state
new_behaviour add a new behaviour module to the agent
remove_behaviour remove a behaviour module from the agent
behaviour_activity set the activity of a behaviour module to be on or off
change_colour change the colour of the agent
kill_cell kill the agent
divide trigger the agent to divide (mitosis)
produce_child trigger the agent to create a child agent
delayed_action do an action after a given duration of time
probabilistic action do an action with a given probability
break_interactions remove all interactions (specific interactions) of a given type

morphpologies

coccus representation of a coccus (spherical) cell morphology
bacillus representation of a bacillus (rod-shaped) cell morphology

species

cell representation of a cell agent, which has states, links, behaviours and a morphology
eps representation of an eps agent, which has a morphology

behaviours

periodic_action an action that occurs periodically
trigger a list of conditions and actions, where once all conditions are met, all actions are executed
mitosis a cell divides upon reaching twice of its original mass
eps_secretion cells secrete EPS (agents, represented as small spheres) at some rate
conjugation models conjugation between physically contacting bacteria
cell_adhesion models membrane surface structures and specific interactions between cells
sbml models intracellular dynamics as SBML models which are solved with LibSBMLsim
differential_equations models intracellular dynamics as sets of ordinary differential equations
chemotaxis models chemotaxis of bacteria - run and tumble dynamics ascending chemical gradients
reporter changes the colour of the cell based on the value of a state
toxicity kills the cell upon it experiencing over a threshold of a specific chemical
membrane models membrane transport of a cell (active or passive mechanisms can be defined)
random_walk models a random walk of a cell (similar to brownian motion force)
pressure_death kills the cell upon it experiencing more than a define threshold of physical pressure
gillespie models intracellular dynamics as a stochastic Gillespie model
constant_growth models a cell growing at a constant rate
boolean_reporter agent changes between two colours based on a boolean state

boolean_grn models intracellular dynamics as a boolean network
run_tumble models flagellar based run and tumble motility dynamics

initial_conditions

initial_chemical_concentration define an initial chemical concentration at a position
initial_chemical_quantity define an initial chemical quantity at a position
initial_chemical_concentration_everywhere define an initial chemical concentration throughout the whole domain
initial_chemical_quantity_everywhere define an initial chemical quantity throughout the whole domain
initial_intracellular_chemical_concentration define an initial chemical concentration inside cells of a specific species
initial_intracellular_chemical_quantity define an initial chemical quantity inside cells of a specific species
initial_cell_state_activity define the initial value of a specific state of a specifc species
initial_cell_position define a cell (agent) at a specific position
initial_population define a well mixed population of cells
initial_population_in_area define a well mixed population of cells within a specific area (or volume)
initial_population_image define the initial spatial arrangement of cells from a previous simulation state
initial_population_microscopy_image define the initial spatial arrangement of cells from a processed microscopy image encoding
initial_grid_of_cells define a uniform grid of cells of a specific species

devices

chemostat define a chemostat attached to a domain boundary
bactostat define a bactostat attached to a domain boundary
chemical_pool define a chemical pool at a specific point
chemical_source define a chemical source at a specific point

chemical_sink define a chemical sink at a specific point
canera define a camera to record the simulation

exporters

sampler samplers collection custom dave from the simulation
geometry_image geometry images (population images) export the spatial arrangement of cells (can be used to initialise models)
timers exports the timers profiling the Simbiotics integrator
microsensor exports the chemical concentration at a given position
positional_cell_chemical exports a spatial description of intracellular chemical quantities (for heatmaps etc)
orientation exports the orientation of agents

schedules

save_and_exit saves all data collection to file and exits the simulation
export_periodically flushes and saves data exporters to file periodically
pipette_event schedules a pipette event (adding chemicals or agents to the domain)
camera_rotate schedules the camera to rotate at a given rate
camera_pan schedules the camera to pan at a given rate

Model definitions

Java functions

defineWorldSize

Defining the world size sets the simulation domain dimensions.

```
void defineWorldSize(double world_size)
                                                    (1)
void defineWorldSize(double world_x, double world_y, double world_z)
                    (2)
```

Where *world_size* is the length of a cubic domain. Alternatively one can have a cuboidal domain, where*world_x* is the length of the domain along the X axis, *world_y* the length of the domain along the Y axis, and *world_z* the length of the domain along the Z axis.

In Simbiotics, the X axis is right/left, the Y axis is up/down and the Z axis is back/front, with the positive/negative values being the respective direction for each axis.

defineBoundary

Defining boundaries sets the behaviour of agent geometries when they interact with the sides of the cuboidal world domain. Specific boundary behaviours can be set to particular faces of the domain by specifying the Axis and AxisFace parameters (2), if no AxisFace parameter is passed (1) then the boundary condition is applied to both the minimum and maximum faces of the given axis.

```
void defineBoundary(Axis axis, BoundaryCondition boundary_condition)
                          (1)
void defineBoundary(Axis axis, AxisFace axis_face, BoundaryCondition
    boundary_condition) (2)
```

Where *axis* is the target axis (X, Y, Z), *axis_face* is which face of the cube along that axis (MIN, MAX) and *boundary_condition* is an implementation module describing boundary mechanics.

definePhysics

Defining the physics solver sets the integration method for calculating how agent geometries positions change due to forces.

```
void definePhysics(PhysicsSolver physics_solver)
```

Where *physics_solver* is an implementation module of the physics solver.

defineDiffusion

Defining the diffusion solver sets the method used for calculating chemical fluxes between domain subvolumes.

```
void defineDiffusion(DiffusionSolver diffusion_solver)
```

Where *diffusion_solver* is an implementation module of the diffusion solver.

defineCollisions

Defining the collision solver sets the method used for calculating the forces geometries which are colliding exert on each other.

```
void defineCollisions(CollisionSolver collision_solver)
```

Where *collision_solver* is an implementation module of the collision solver.

defineChemicalSpecies

Defines a chemical species to be part of the model with given ID and properties.

```
void defineChemicalSpecies(Chemical chemical)
```

Where *chemical* is an implementation module of a chemical, which can be present in extracellular and intracellular compartments.

defineChemicalInterface

Defines a flux of chemicals at a point position in the domain, which can be identified with an ID.

```
void defineChemicalInterface(ChemicalInterface chemical_interface,
    String id)
```

Where *chemical_interface* is an implementation module of a chemical interface, and *id* is the name of that interface.

defineBindingSite

Defines a binding site which can represent a physical binding location on the surface of cellular geometries and boundary interfaces.

```
void defineBindingSite(BindingSite binding_site)
```

Where *binding_site* is an implementation module of a binding site.

defineInteraction

Defines an interaction which can represent the physical mechanism between two binding sites.

```
void defineInteraction(PhysicalInteraction interaction)
```

Where *interaction* is an implementation module of a PhysicalInteraction.

defineCellBehaviour

Defines a behaviour module to be identified by its ID and key, which can then be bound to cell species definitions to describe cell dynamics.

```
void defineCellBehaviour(iBehaviour behaviour, String module_id, String
    module_key, Boolean active)
```

Where *behaviour* is an implementation module of an iBehaviour, *module_id* is its unique identifier, *module_key* is the type of behaviour corresponding to the Simbiotics library keys, and *active* is a boolean whether the behaviour is active (on) or inactive (off).

defineCellSpecies

Defines a cell species with a particular implementation, such as their spatial representation, behaviour and state information.

```
void defineCellSpecies(CellSpecies cell_species)
```

Where *cell_species* is an implementation module of CellSpecies.

definePopulation

Defines the initial population size of the cell species, their positions are distributed normally throughout the cubic domain.

```
void definePopulation(String species_id, int population_size)
```

Where *species_id* is the target species ID, and *population_size* is the number of cells .

defineCellAtPosition

Defines a cell of the given species at a position, can also have a unique cell name to track an individual cell throughout the simulation.

```
void defineCellAtPosition(String species_id, double[] position)
void defineCellAtPosition(String species_id, double[] position, String
    cell_name)
```

Where *species_id* is the target species ID, *position* is the coordinates of the cell, and *cell_name* is the unique name of that cell.

defineInitialVelocity

Defines the initial velocity for all cells in at the initial state of the model with some random deviation.

```
void defineInitialVelocity(double velocity, double standard_deviation)
```

defineDevice

Defines a device which may interact with or probe the model state, indentifiable by its ID.

```
void defineDevice(iDevice device, String device_id)
```

defineExporter

Defines an exporter to write model data to file, it's identifiable by its ID and has an optional file path of where to write the data to. If no file path is supplied then the default results folder as defined in the Simbiotics configuration will be used.

```
void defineExporter(Exporter exporter, String exporter_id)
void defineExporter(Exporter exporter, String file_path, String
    exporter_id)
```

defineAuxiliary

Defines an auxiliary program which may automate interactions or events in the model, identifiable by a unique ID.

```
void defineAuxiliary(iAuxiliary auxiliary, String auxiliary_id)
```

defineDrawer

Defines a model component to visual for 3D rendering output.

```
void defineDrawer(Drawer drawer)
```

defineConstant

Defines a constant for the simulation engine, such as the global *"TIME_STEP"*.

```
void defineConstant(String id, double value)
```

.8 Building new modules

The Simbiotics library can be extended by designing new modules in Java. This is achieved by meeting the requirements of one of the Simbiotics interfaces. The interfaces are all stored in the same Java package, found in: *simbiotics.plugs.interfaces*. The base classes implementing these are found in: *simbiotics.plugs.base*.

When developed a new module, one may wish to *extend* one of the base classes, or for finer control, they may wish to directly *implement* the interface. For example, when developing a new module for bacterial behaviour, the *Behaviour* class could be *extended*, or one may *implement* the *iBehaviour* interface directly.

We exemplify this through how the *Mitosis* behaviour is implemented. We start by overriding the *iBehaviour* interface, which can be seen below.

```
public interface iBehaviour extends Serializable, CustomSerializable {

   /** Execute (run) the behaviour module **/
   void execute();

   /** Apply the changes of the behaviour module **/
   void apply();

   /** Get a copy of the module**/
   iBehaviour getCopy();

   /** Divide the module (with a given ratio) **/
   iBehaviour divide(double ratio);

   /** If returns true, this module is copied to the child cells during
       division **/
   boolean isCopiedWhenCellDivides();

   /** Return the agent this module belongs to **/
   iAgent getAgent();

   /** Set the agent this module belongs to **/
```

```
    void setAgent(iAgent cell);

    /** Get the probability of this behaviour module being inherited
        during division**/
    double getInheritanceProbability();

    /** Set the probability of this behaviour module being inherited
        during division **/
    void setInheritanceProbability(double inheritance_probability);

    /** Get behaviour module id (String name) **/
    String getBehaviourId();

    /** Set the behaviour module id (String name) **/
    void setBehaviourId(String module_id);

    /** Set the behaviour module type (String type) **/
    void setBehaviourType(String module_type);

    /** Get the behaviour module type **/
    String getBehaviourType();

    /** Get whether the behaviour module is active (switched on) **/
    boolean isActive();

    /** Set the behaviour module activity to be on or off **/
    void setActive(boolean active);

    /** Returns true if the module has an associated volume (biomass) **/
    boolean hasVolume();

    /** Get the volume of this module **/
    double getVolume();
}
```

This interface has a set functions which have to be implemented in order for it to be treated as a valid behaviour module. The base implementation of this interface is the *Behaviour* abstact class, which can be seen below:

```
public abstract class Behaviour implements iBehaviour {

   public String behaviour_id = "";
   public String behaviour_type = "";

   public boolean active = true;
   public double inheritance_probability = 1.0;
   protected iAgent agent;

   /** Behaviour Constructor(s) **/
   public Behaviour() {
      this(true, 1.0);
   }
   public Behaviour(String behaviour_type){
      this.behaviour_type = behaviour_type;
      this.behaviour_id = behaviour_type;
   }
   public Behaviour(boolean active){
      this(active, 1.0);
   }
   public Behaviour(boolean active, double inheritance_probability){
      setActive(active);
      setCopyOnDivide(inheritance_probability);
   }

   /** Execute (run) the cell behaviour - toggled by 'active' variable
       **/
   public void execute(){
      if(active)
         run();
   }
```

```
/** Run the module at each iteration **/
public abstract void run();

/** Apply the module at each iteration **/
public abstract void apply();

/** Get a copy of the module */
public abstract iBehaviour getCopy();

/** Get a copy of the module */
public iBehaviour divide(double ratio){
   return getCopy();
}

/** Get the cell this module is associated with **/
public iAgent getAgent() {
   return agent;
}

/** Set the cell this module is associated with **/
public void setAgent(iAgent cell) {
   this.agent = cell;
}

/** Set whether the module is copied on divide **/
public void setCopyOnDivide(double copy_on_divide){
   this.inheritance_probability = copy_on_divide;
}

/** Returns true if this module is copied upon mitosis (for child
    cell) **/
public boolean isCopiedWhenCellDivides() {
   return inheritance_probability > 0;
}
```

```
/** Set the module ID **/
public void setBehaviourId(String module_id){
   this.behaviour_id = module_id;
}

/** Set the module type **/
public void setBehaviourType(String module_type){
   this.behaviour_type = module_type;
}

/** Get the module type **/
public String getBehaviourType(){
   return behaviour_type;
}

/** Get the module ID **/
public String getBehaviourId(){
   return behaviour_id;
}

/** Returns true if the module is currently active **/
public boolean isActive(){
   return active;
}

/** Set whether the module is currently active or not **/
public void setActive(boolean active){
   this.active = active;
}


public double getInheritanceProbability(){
   return inheritance_probability;
}
```

```
   public void setInheritanceProbability(double inheritance_probability){
      this.inheritance_probability = inheritance_probability;
   }
   public boolean hasVolume(){
      return false;
   }

   public double getVolume(){
      return 0;
   }
}
```

This is an abstract class and thus can not be instantiated - it simply provides the general implementation that most behaviour modules need so that code doesn't have to be repeated in those classes. Often this *Behaviour* class is sufficient to extend, and one can override all the methods in the class as they please - however at some point it makes more sense to implement the *iBehaviour* class directly to reduce any efficiency costs associated with instantiating many instances of a class with a chain of super constructors.

In the case that the *Behaviour* class is sufficient to build on top of, it can be used to specific specific behaviour of an agent. Below we show a *Mitosis* class implemented to divide a coccus (spherical) cell upon reaching twice of its original volume:

```
public class Mitosis extends Behaviour {

    protected boolean divide;

    /** Constructor(s) and copy **/
    public Mitosis(){
        super("mitosis");
        setActive(true);
    }
    public Mitosis getCopy(){
        return new Mitosis();
```

```
    }

    /** Run this cell internal (store results for a synchronous update)
        **/
    public void run(){
        divide = false;
        if(agent.getMass() >= agent.getBirthMass() * 2)
            divide = true;
    }

    /** Update all of the cells synchronously **/
    public void apply(){
        if(divide)
            agent.divide();
    }
}
```

And here is an example of a variation of the *Mitosis* class which implements a bacillus (rod-shaped) cell dividing upon twice its original length:

```
public class Mitosis extends Behaviour {

    protected boolean divide;

    /** Constructor(s) and copy **/
    public Mitosis(){
        super("mitosis");
        setActive(true);
    }
    public Mitosis getCopy(){
        return new Mitosis();
    }

    /** Run this cell internal (store results for a synchronous update)
        **/
    public void run(){
```

```
        divide = false;
        CapsularBody body = agent.getBody();
        if(body.getLength() >= 2 * body.getBirthLength())
            divide = true;
    }

    /** Update all of the cells synchronously **/
    public void apply(){
        if(divide)
            agent.divide();
    }
}
```

Please note: neither of these *Mitosis* class implementations are the one actually used in the platform, they are simply examples of how a new module can be built.

Appendix B - Easybiotics user guide

.9 Introduction

Overview

Easybiotics is a graphical user interface (GUI) for the Simbiotics platform. Easybiotics allows for the design, simulation and analysis of Simbiotics models via an easy to use graphical interface which does not require programming experience to operate. It is a light-weight program developed in Python which has minimal dependencies.
This document describes how to get and install Easybiotics, as well as some tutorials on how to use it. **You can also try Easybiotics in a Virtual Machine for easy out-of-the-box use, it can be found on the website along with video tutorials on how to use the software.**

https://bitbucket.org/simbiotics/simbiotics/wiki/Home

License

Technical overview

Easybiotics is written in Python 2.7, and depends on the Kivy, Matplotlib and Pandas library.

Terminology

To clarify some of the terminology used in this document, we list some keywords and their meaning.

Term	Meaning
Library module	Java classes within the Simbiotics library which describe model-specific behaviour.
$SIMBIOTICS	The main Simbiotics folder, which contains the *src* folder

Table 4: User manual terminology

.10 Getting Easybiotics

Downloading and installing

Simbiotics and Easybiotics are distributed together and can be downloaded at https://bitbucket.org/simbiotics/simbiotics/wiki/Home.

Simbiotics is developed in Java 1.7, and Easybiotics in Python 2.7.

You must have the following packages installed:

- Open JDK >= 6 (GNU General Public Licence + classpath exception) or Oracle Java SE >= 6 (Oracle Binary Code Licence)
- Python >= 2.7 - (https://www.python.org/)
- Kivy >= 1.8 - (https://kivy.org/)
- Matplotlib >= 1.4.2 - (https://matplotlib.org/index.html)
- Pandas >= 0.17 - (https://pandas.pydata.org/)

And optionally, if you wish to use SBML integration, you must have:

- libSBML - http://sbml.org/Software/libSBML (GNU LGPL)
- libSBMLSim - http://fun.bio.keio.ac.jp/software/libsbmlsim/ (GNU LGPL)

If you already have these dependencies installed, skip to **2.3. Running easybiotics**. Note that you must have Simbiotics installed to use Easybiotics. If you do not have Simbiotics installed please refer to the *simbiotics_guide.pdf* document.

Getting Dependencies

Easybiotics depends on modules from the Kivy, Pandas and Matplotlib libraries, which can be found here:

[Kivy]
[Pandas]
[Matplotlib]

If you are using a linux system, these can be installed on command line via aptitude with the following commands:

Listing 9: Getting Easybiotics dependencies via aptitude

```
sudo apt-get install python-kivy
sudo apt-get install python-pandas
sudo apt-get install python-matplotlib
```

Running Easybiotics

Simbiotics can be run in multiple ways allowing the user to choose which is most appropriate for them. Simbiotics can either by run via Easybiotics, by command-line, or opened in an IDE.

To run Easybiotics in Linux/MAC open a terminal/command prompt, navigate to the *$SIMBOTICS* directory and run the *"start_ easybiotics"* script. For example:

Listing 10: Running Easybiotics

```
cd $SIMBIOTICS
./start_easybiotics
```

If you are using Windows you must use start the python application with *gui/qSimbiotics.py*

.11 Developing models in Easybiotics

Here we elaborate on the features of Easybiotics and how to use them. First describing what each part of the GUI is for. Examples of how to use Easybiotics can be found the in tutorials section of this document, and in the Easybiotics video tutorials which can be found at

https://bitbucket.org/simbiotics/simbiotics/wiki/Tutorials

Easybiotics provides features to accomplish the following:

1. Develop models
2. Run models
3. Analyse models
4. Render visualisations

Develop models

The model development environment allows for Simbiotics models to be composed easily. The model specification is represented as an interactive tree structure to which library modules can be attached.

Run models

Models can be run from inside the model development environment. Models can be run with optional real time rendering of the simulation domain.

Analyse models

Simulation data can be exported to file by attaching exporters to the model specification. This data can optionally be visualised in live plots during the simulation run. The Simbiotics library also contains some analysis modules for characterisation of the model during run time.

Render visualisations

In addtion to real-time simulation rendering, simulations can be visualised after run time as 3D scenes. You can create static 'image' 3D scenes, or animated 3D scenes composed of a sequence of static scenes.

Overview

Creating a model

Models can be developed in Simbiotics by selecting the "Develop and run models" on the home screen. You may either create a new model, or load an existing model. There are a collection of example models which can be loaded directly from the load screen. Additionally if you select browse and navigate to *examples/models/* you will find some more example models.

We will first go over the basics on model development in Easybiotics. A tutorial can be found below this section which walks through the development of a basic model.

Once you have created a new model or loaded an existing one, you will be presented with the model development screen. The model development screen has 5 major components: the Filebar, the Config Editor, the Model Editor, the Display Panel and the Button Panel.

Filebar

The filebar (highlighted in yellow on Figure 3) gives access to functions such as the Easybiotics settings, handling configuration and model files, running simulations with optional live visualisation, graphs and parameter sweeps, along with information about Easybiotics.

Config Editor

The configuration editor (highlighted in blue on Figure 3) gives access to the Simbiotics platform settings, such as whether to run the simulation with real time rendering, and how many CPU threads should be created for the simulation.

Model Editor

The model editor (highlighted in green on Figure 3) allows for the manipulation of a model specification. This includes adding/removing modules, setting the modules parameters, and connecting modules together to represent the target system.

Display Panel

The display panel (highlighted in red on Figure 3) has three tabs. The description tab displays details of the selected module. The properties tab allows for the modification of any parameters of that module. The edit tab allows for the direct manipulation of the model specification file values.

Button Panel

The button panel (highlighted in orange on Figure 3) has three buttons - to run the current model, to quicksave the current model, and to go back to the Easybiotics homepage.

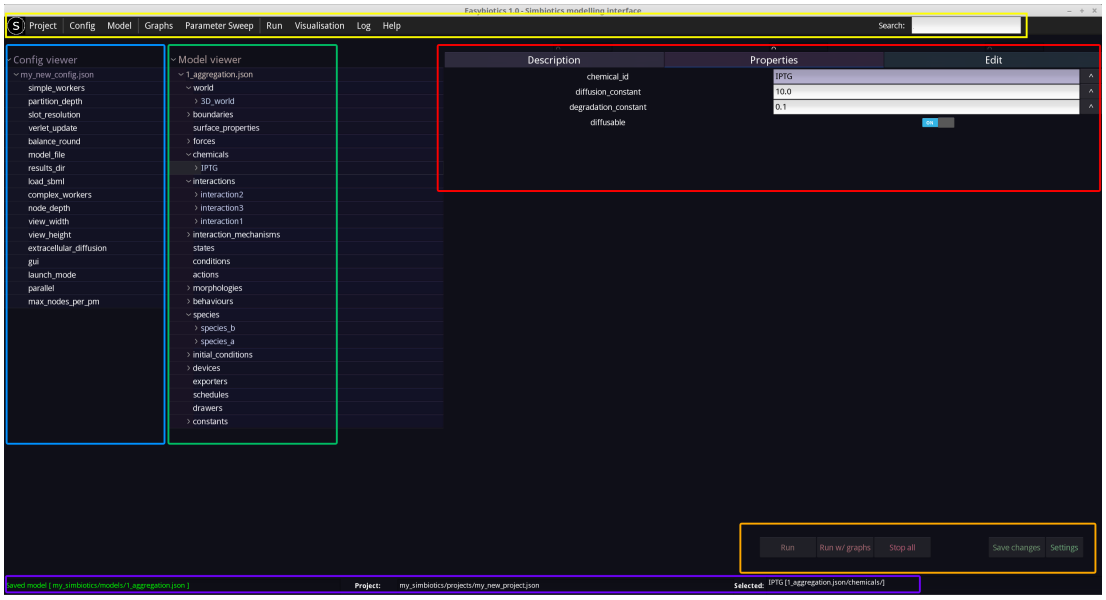


Figure 1: Overview of the Easybiotics modelling interfaces. The file bar is highlighted in yellow, the Simbiotics configuration editor in blue, the model specification editor in green, the display panel in red, and the button panel in orange.

Running the model

Models can be run by either clicking the 'Run' button in the button panel, or by selecting one of the run options on the filebar.

You may select the amount of RAM that the JVM (Java Virtual Machine) uses for the simulation by selecting 'Settings - Run Settings' from the file bar. Here you can set the initial amount of RAM allocated as well as the maximum amount of RAM the simulation may use.

The simulation can be run with an optional live rendering. This is set in the configuration editor, by setting the *gui* variable to *true* or *false*. If using the live renderer, the Simbiotics GUI is opened, more information can be found in the Live Visualisation of Simulations section.

Analysing the model

Models can be analysed by attaching exporters to the model specification. Numerous exporters can be defined, where each collects specific data about the simulation and writes it to file. Graphs may then be defined, which are set to plot data from the exporters. Graphs can be saved and loaded to/from file for easy reuse. To run real time graphs during the simulation run, select the 'Run - Run with live graph plotting' option from the file bar, and specifying the graphs which are to be rendered.

Easybiotics also provides a feature to perform parameter sweeps. Similar to graphs, parameter sweep objects can be defined, which iterate over properties in the model. All simulations are run for the defined parameter sweep ranges and their results saved to independent directories. Live graph plotters may also be attached to parameter sweeps, plotting the data from all simulations on one graph for easy comparison.

In addition, the library contains numerous analytical tools such as those able to calculate the mean squared displacement of agents and their velocity autocorrelation function. There are also some simulated lab tools, such as microsensors and a spectrophotometer. This allows for probing of the simulated experiment as would typically be done for the real experiment.

Examples of analysis with the tool are shown in the Easybiotics tutorials

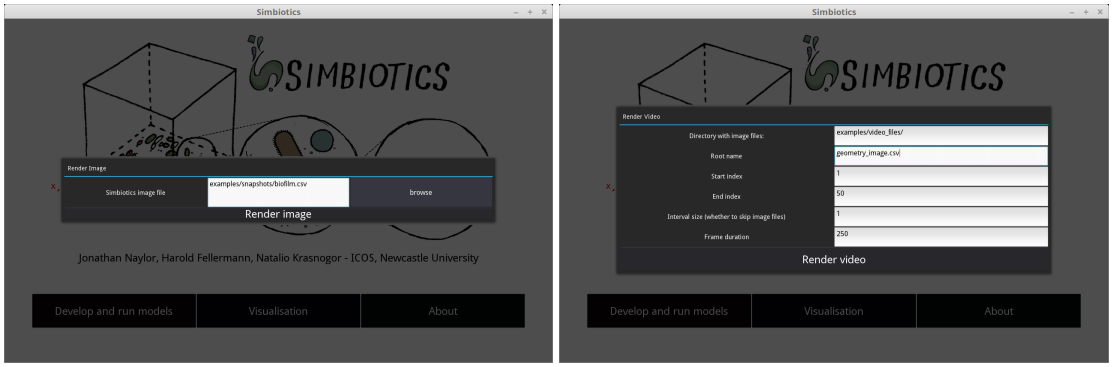
below.

Rendering visualisations

Visualisations of simulations can be rendered after they have finished executing, as an alternative to a live visualisation. This can be achieved by attaching a certain type of exporter, called a *geometry_image*, to the model specification. This exporter writes all geometry properties to a file periodically, writing a new file for each time point. It produces a series of indexed files which can be found in the results folder you set for the exporter.

Each geometry image file can be rendered independently into an static 3D scene, which is loaded in the Simbiotics GUI allowing the user to move the camera around the scene and modify what properties are visualised.

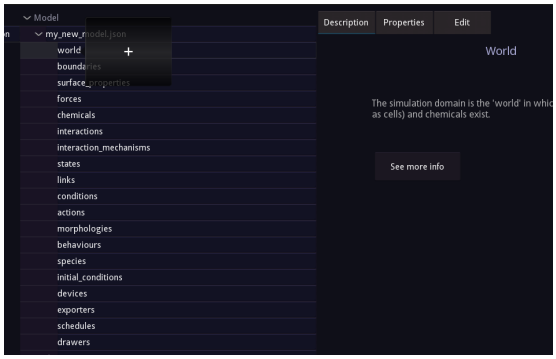
Alternatively, a sequence of geometry images can be loaded into an animated 3D scene. The user may set the delay between the animation frames, and whether the renderer should skip indexes. The animation renderer also runs a camera to record the animated 3D scene and convert it into an *.avi* which can be found in the *$SIMBIOTICS/results* folder.

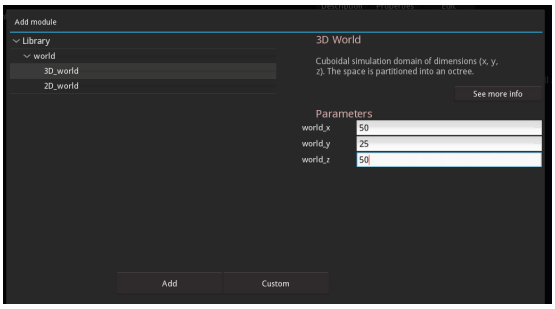


Tutorial 1 - Creating your first model

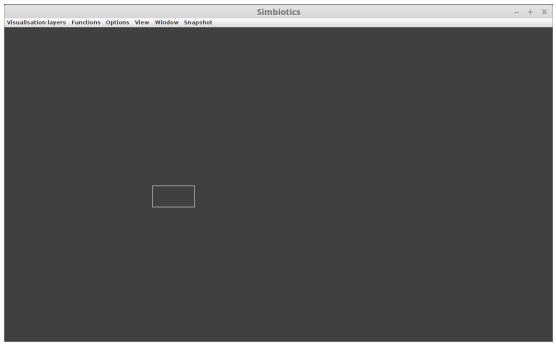
To create a new model in Easybiotics selected *Develop and run models - New model* from the Easybiotics home page, and enter the file name for your model, then press *Create*. The modelling interface should then display, showing the configuration editor on the left, and the model specification editor in the middle.

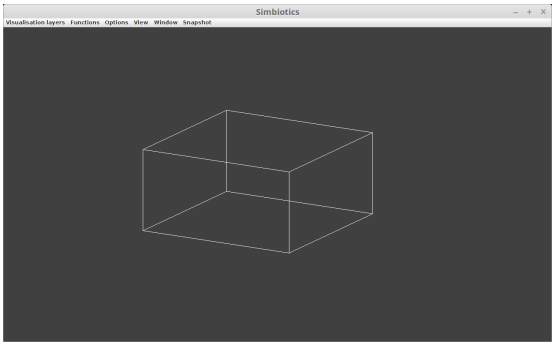
Firstly, we need to create a simulation domain, which we call a *world*. To do this, right-click on the *world* node on the model specification tree. This will display all world modules in the Simbiotics library. For this example, select the *3D_world* module, and set the world dimensions to be 50*25*50, then press the *Add* button. An example of this can be seen below.





To see that baby in action, hit run (either from the filebar or from the button bar bottom right). Make sure you have the *gui* property in the configuration set to *true*! The Simbiotics GUI should open, showing the simulation domain and nothing else. You can rotate the camera by right-clicking and dragging, and scrolling to zoom in and out. For more information on the Simbiotics GUI, see the Live visualisation of simulations section nearer the start of this document.

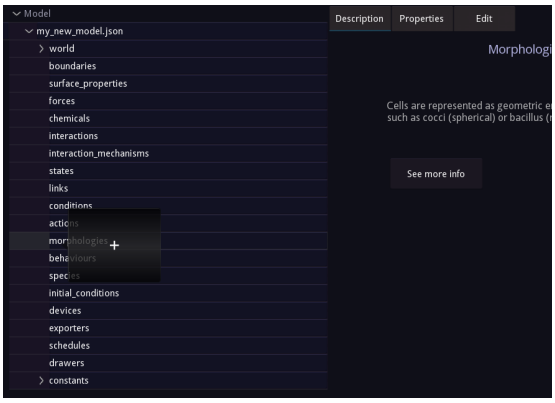


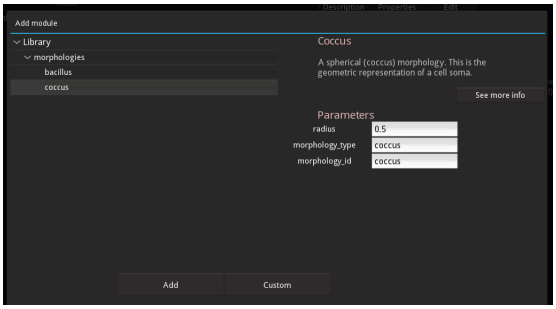


Ok, lets add some cells!

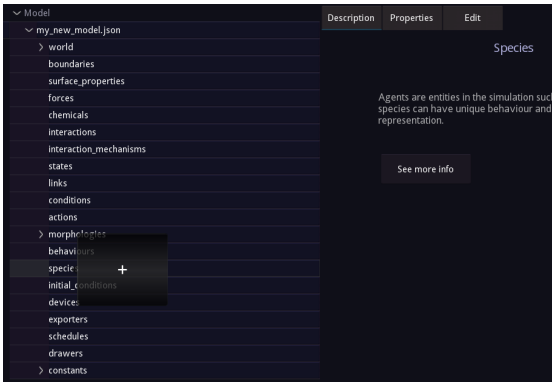
In brief, a cell is considered as an agent (an individual) in the model, and each agent is represented by a physical geometry in the simulation. In the model specificaiton, a physical geometry can be created by add a *morphology*. To do

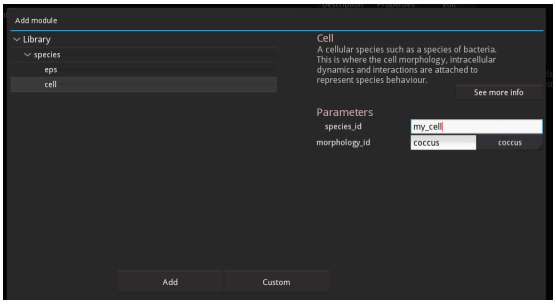
this, right click on the *morphologies* node on the model specificaiton, and select the *coccus* (spherical) module. The default parameters are ok, so just press *Add*. This process can be seen below.





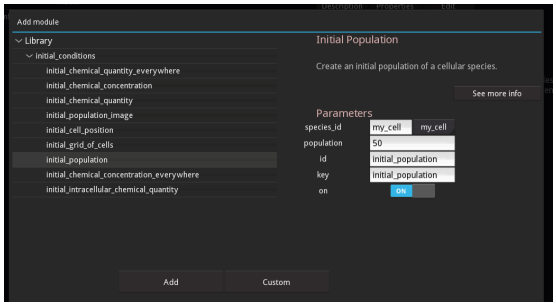
Next, we need to create a cellular species (a type of agent) which has the morphology as its geometric shape. To do this, right-click on the *species* node. Select the *cell* module, and in the parameters set the *morphology_id* to be the *coccus* morphology we have just defined. When linking modules in this way, you either have to type in the id, or can select one of the valid modules you've already defined by the ... drop down menu. Let's also change the name of this cellular species, set the *species_id* to be *"my_cell"*.

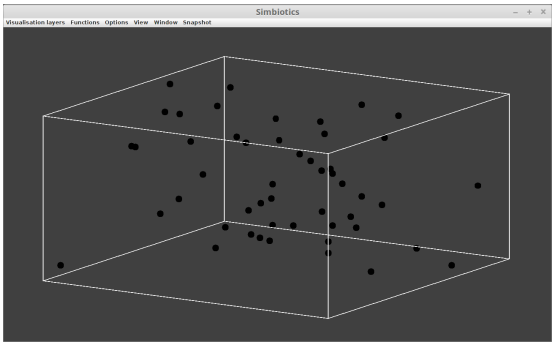




Finally, let's create a population of cells. This can be done by right clicking on the *initial_conditions* node and adding an *initial_population* module. Set the *species_id* to be *"my_cell"*, add set the population size to be 50, and press add.

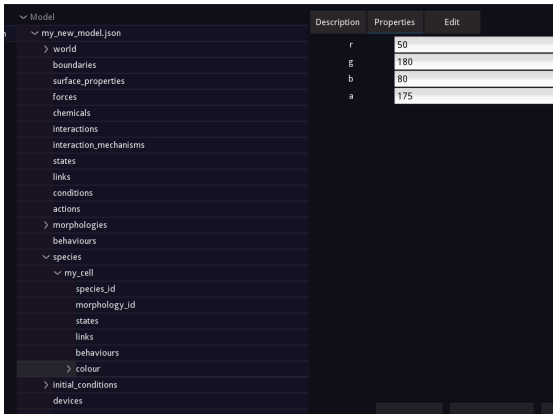
Ok... so the cells aren't doing much, lets make this more interesting. To set the colour of the cells, navigate to the *colour* property of the *cell* species you

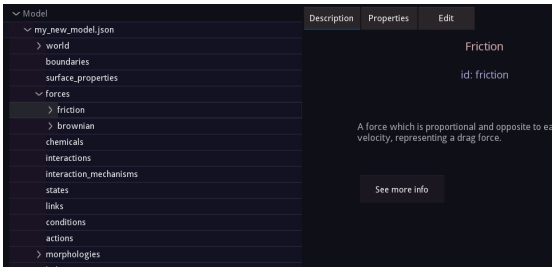




defined (by clicking on the dropdown icons on the model specification.) Select the *Properties* tab in the display panel on the right, it should show the options to set the RGBA values. Set the colour to whatever you like, we'll do a nice green (50, 180, 80, 175).

Also, let's add some movement to the system. We'll do this by adding a *friction* and *brownian* module to the *forces* node with their default parameters.

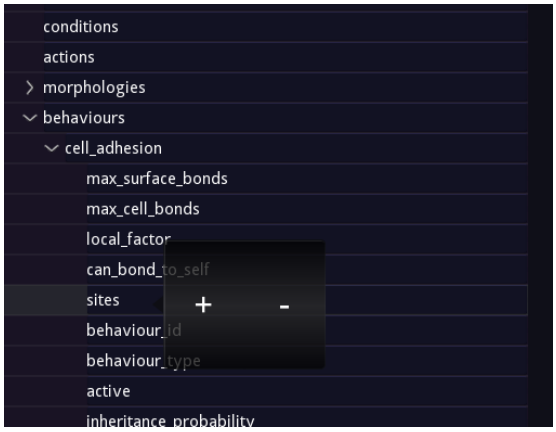


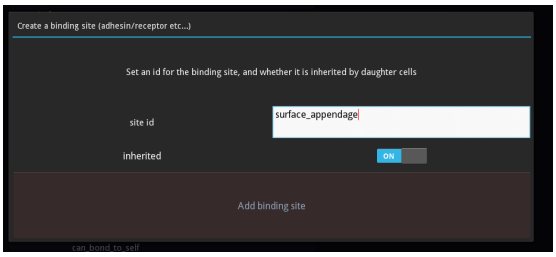


Now, when you press *Run*, you should see a population of moving cells which are the colour you set!

Let's now add some basic behaviour to the cells to finish off tutorial 1. We'll model that the cells have surface appendages that cause them to aggregate. To do this, we must create a *behaviour* module, called *cell_adhesion*. Add this module to the model specification with its default parameters. We must then add a property to this *cell_adhesion* module representing a surface appendage. To do this, open up the property list for the *cell_adhesion* module, and right on *sites* then add a site called *"surface_appendage"* as seen below. The *inherited* boolean

sets whether any child cell would directly inherit this surface structure, though for this model it is irrelevant we will not include cell growth.





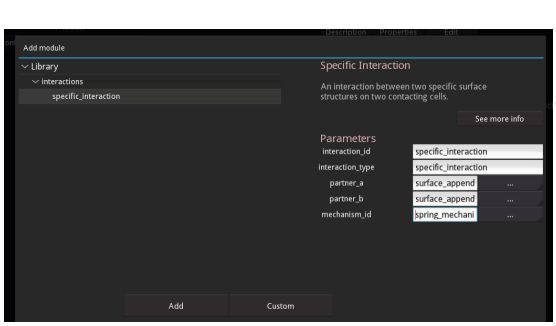
Next we need to define an *interaction_mechanism*, which is the sub-model describing how two surface appendages interact. Add a *spring_mechanism*, which describes the interaction as a Hookian spring. This forms a spring connecting the two interacting geometries according to the springs parameters. For this model, we can leave the parameters as default.

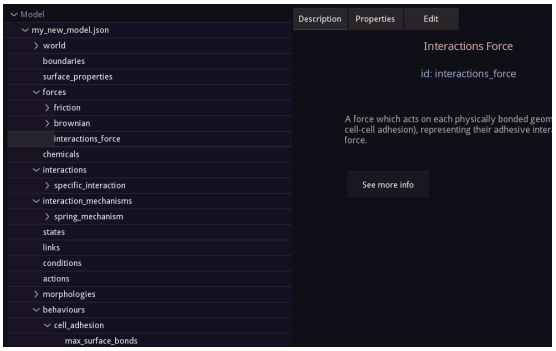
We must then create an *interaction*, which associates the surfaces appendages we defined with the mechanism. Add a *specific_interaction* module, and set the partners both to be the *"surface_appendage"*, and the *mechanism_id* to be the *"spring_mechanism"* we defined above.

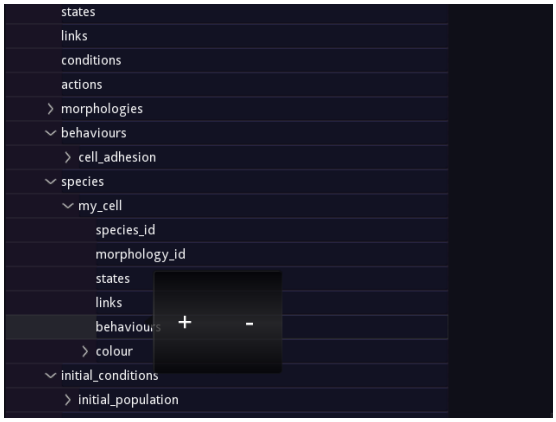
Finally, we must add the *interaction_force* module to our *forces*. This is a module which includes the physical force generated by interactions to the total force that a cell experiences. Without this force module, interactions will have no physical effect.

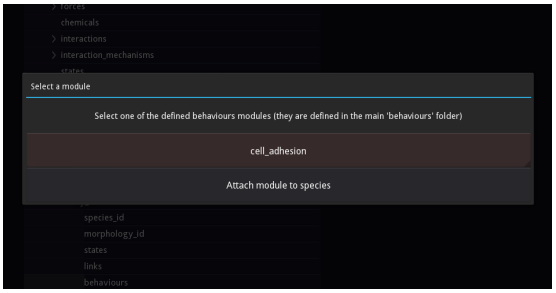
Ok, almost there! Now we have defined a *behaviour* module which has a surface appendage, and we have defined an *interaction* that states if two of those surface appendages (from two different cells) comes into contact, then the *interaction mechanism* we also defined is used to model that interaction. The only thing left is to say that our cell species, *"my_cell"*, has this type of behaviour.

To do this do, open up the species definition, and right click on the *behaviours* property which is inside the *"my_cell"* definition, press the + button, and select the the *"cell_adhesion"* module from the drop down box, and select *Attach*.

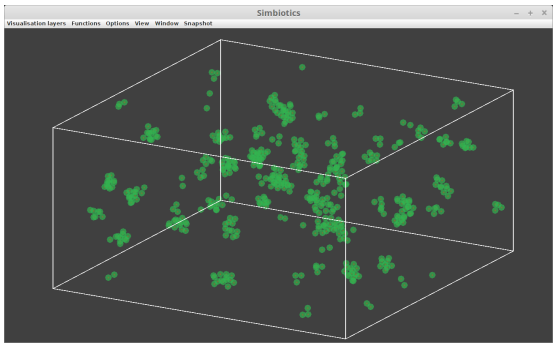


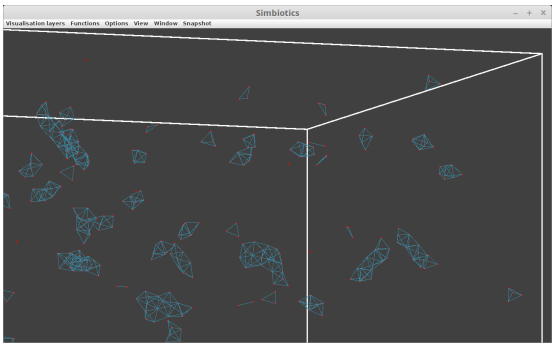






Now when you run the model, the cells should begin to aggregate, such as you see below. You may want to increase the population size so that the aggregation is more apparent. To render the wire-frame of the interactions, turn off the rendering of *"my_cell"* in the *Visualisation layers* menu of the Simbiotics GUI, and turn on the *"interactions"* layer.





This concludes tutorial 1. You can play around with the parameters in the model, and more information on these parameters can be found in the publications

surrounding Simbiotics, which can be found in the *Related publications* section.

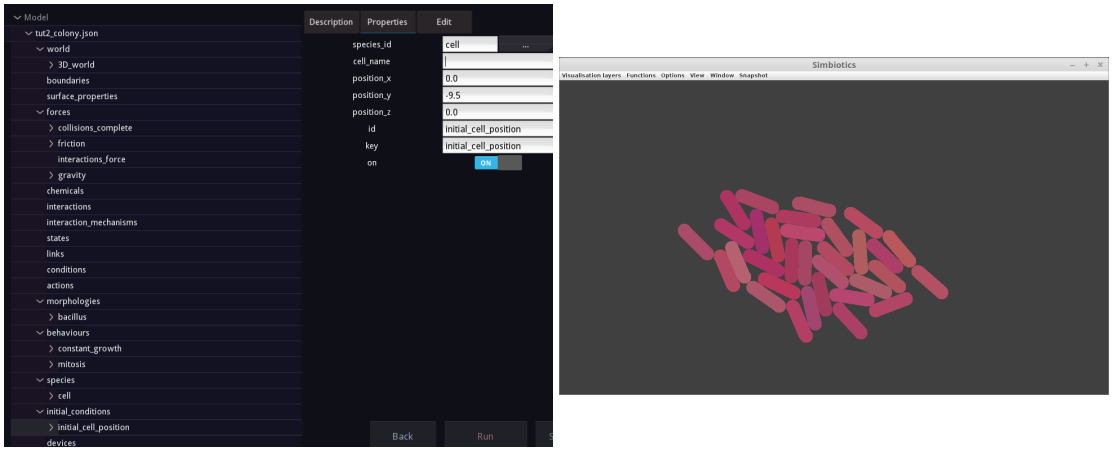
Tutorial 2 - Collecting some data from a model

Here we run through an example of collecting data from a model, including how to visualise it live during the simulation.

We will illustrate this example through a model of a colony of bacillus (rod-shaped) cells growing on a surface. We will assume a constant nutrient supply and thus uninhibited growth. The bacillus cells secrete a chemical which diffuse out of the cell membrane into extracellular space.

To start, please create a new model, and define the following specification:

1. To world, add a 3D_world which is 50*20*50.
2. To forces, add a collisions_complete module, with a range of 10 and a k value of 50.
3. To forces, add a friction module, with a constant value of 1.0.
4. To forces, add a interaction_force module.
5. To forces, add a gravity module, with a constant value of 0.2.
6. To morphologies, add a bacillus module with a length of 1.0 and radius of 0.5.
7. To behaviours, add a constant_growth module with a growth rate of 0.025.
8. To behaviours, add a mitosis module.
9. To species, add a cell module with a the bacillus morphology.
10. In the species - modules property, attach the constant_growth and mitosis modules.
11. To initial_conditions, add an initial_cell_position, setting the species_id to the cell species you defined, and the position to be x=0.0, y=9.5, z=0.0.



Press run, and you should see a single bacillus cell at the bottom of the simulation domain which is growing and dividing.

Next, we will add the module that describes their cell behaviour, specifically that they synthesize a chemical, and secrete it out of their membrane.

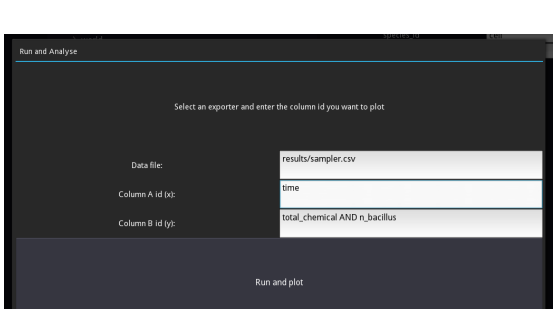
This is done by defining a chemical species. Add a *chemical* module to the *chemicals* definitions with a diffusion_coefficient of 1.0, a degradation_coefficient of 0.1 and diffusable set to true (ON).

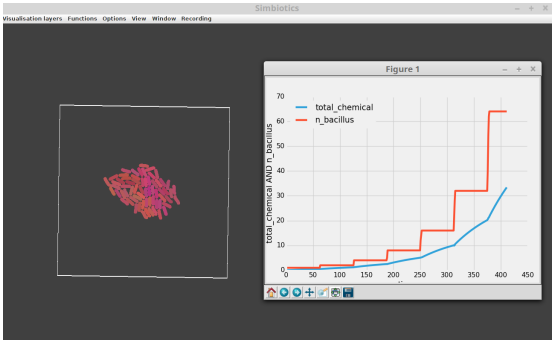
Next, add a *synthesizing_grn* to the *behaviours* definitions. Set the chemical_id to be equal to the chemical you just defined, ensure the velocity_constant is 1.0 and add the module. Add this new behaviour module to the species, in the same way that you attached the mitosis and constant_growth modules to the species.

To check that this works correctly, we can plot the amount of the chemical in simulation. To do this, we must first collect the data - add a *sampler* module to the *exporters* definitions, with the the file_path set to be "results/". Now we have a sampler defined, open it up in the model specification and right click on *samples*, and add a *TotalChemicalQuantity* sample, with the *chemical_id* set to the chemical you defined, and sample title to be "total_chemical". Add a *WorldTime* sample, and title it *time*, and a *CellNumber* sample with the species_id set to the bacillus species you defined.

To periodically write the data to file, add an *export_periodically* module to the *schedules* definitions. We can now run the simulation with live graph plotting.

Click *Run* on the top file bar, then select *Run with live graph plotting.* Set the data file to be the sampler exporter you defined, which is the [file path+exporter id+file extension]. In our instance, this is *results/sampler.csv* (the default results folder is in the $SIMBIOTICS main folder). The graph axes must then be defined, and they must be one of the column headers (sample titles) in the csv data file. We set the X axis to be the time data, and on the Y axis we plot both the total_chemical quantity (scale = molecule number) and the number of the bacillus cell species (scale = cell number).





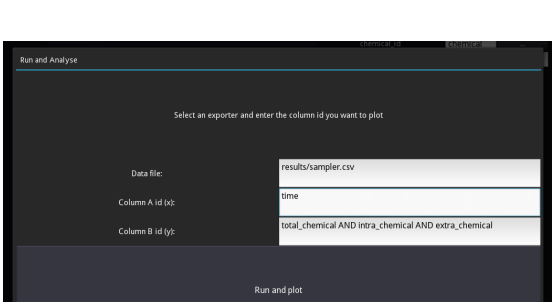
We see that as the population of cells is growing at a constant rate, doubling every 60 units of time (Note: The Simbiotics platform is unit-agnostic, meaning that whatever units you put in are the units you get out. For more information on this see the related publications section.) We also see that as their are more cells, the total rate of production of the chemical increases.

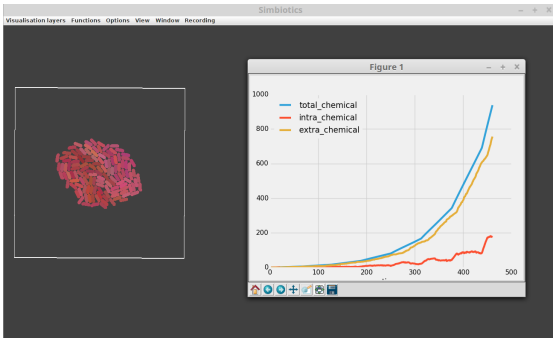
Next, we add the behaviour to the cell describing that the chemical can diffuse out of the cell membrane into the extracellular space. To do this, add a *membrane* module to the *behaviours* definition, leave its parameters as default. Open up the *membrane* definition in the model specification, and right click on the *membrane_fluxes* property. Add a flux with the chemical_id the chemical you defined, and the rate to be 0.2. Set poisson to be true (ON), which sets the solver to take a poisson distributed around the average permeation rate. Set osmotic to be true, which means that the flux direction is always from high to low concentrations. Interpolated can be left on false.

Now, we must attach that membrane module to our cell species definition, in the same way we added cell_adhesion, mitosis and the sythnesizing_grn. To ensure this works, let's plot the intracellular and extracellular quantity of the

chemical over time. To do this, navigate back to the *samples* defined our *sampler* in the *exporters* definitions. Add a *TotalIntracellularChemicalQuantity* and a *TotalExtracellularChemicalQuantity*, with the titles *"intra_ chemical"* and *"extra_ chemical"* respectively, which record the defined chemical_id. Also, change the velocity_constant of the synthesizing_grn to 10.0, and set the *chemical* to have an degradation_constant of 0.0.

We then plot the total_chemical, intra_chemical and extra_chemical over time.





Tutorial 3 - Intracellular dynamics with Gillespie submodels + multiple bacterial and chemical species

In this tutorial we will create a 2D model with a static population of bacteria (species_1). We will model a chemical influx (chemical_1) from the left hand side (X axis minimum face) and an influx of a seocnd species bacteria (species_2) from the right hand side (X axis maximum face.) The bacterial species_2 will have an active motility, called chemotaxis, such that they ascend the chemical gradient and try find the highest concentration of the chemical. The bacterial species has an active transport mechanism, taking the chemical in the extracellular space inside the cell. It also has metabolic behaviour transforming the chemical into a second chemical (chemical_2), this second chemical can diffuse out of the membrane in the extracellular space. Chemical_2 is toxic to bacterial species_1 in high concentrations, resulting in cell death.

To start, please create a new model, and define the following specification:

1. To world, add a 2D_world which is 100*100
2. To forces, add a collisions module,
3. To forces, add a friction module, with a constant value of 1.0.
4. To chemicals, define chemical_1 with a diffusion constant of 2.0 and degradation 0.1.
5. To chemicals, define chemical_2 with a diffusion constant of 1.0 and degradation 0.0.
6. To forces, add a friction module, with a constant value of 1.0.
7. To morphologies, add a coccus module with a radius of 0.75.
8. To morphologies, add a coccus module with a radius of 0.5.
9. To behaviours, add a chemotaxis module with the chemical_id set to chemical_1, and interpolated set to true (ON).

10. To species, create species_1 with the first coccus morphology.

11. To species, create species_2 with the second coccus morphology, and attach the chemotaxis module to its behaviour list.

12. To initial_conditions, add an initial_population, setting the species_id to the species_1, and the population size to 50.

13. To devices, add a chemostat with default parameters.

14. Open the chemostat properties in the model specification editor, right click on fluxes and add a flux of chemical_1 with a rate of 1.0.

15. Right click on the environment_interfaces property of the chemostat, add set it to be the axis to be X, and the axis_face to be MIN.

16. To devices, add a bactostat with default parameters.

17. Open the bactostat properties in the model specification editor, right click on fluxes and add a flux of species_2 with a rate of 0.1.

18. Right click on the environment_interfaces property of the bactostat, add set it to be the axis to be X, and the axis_face to be MAX.

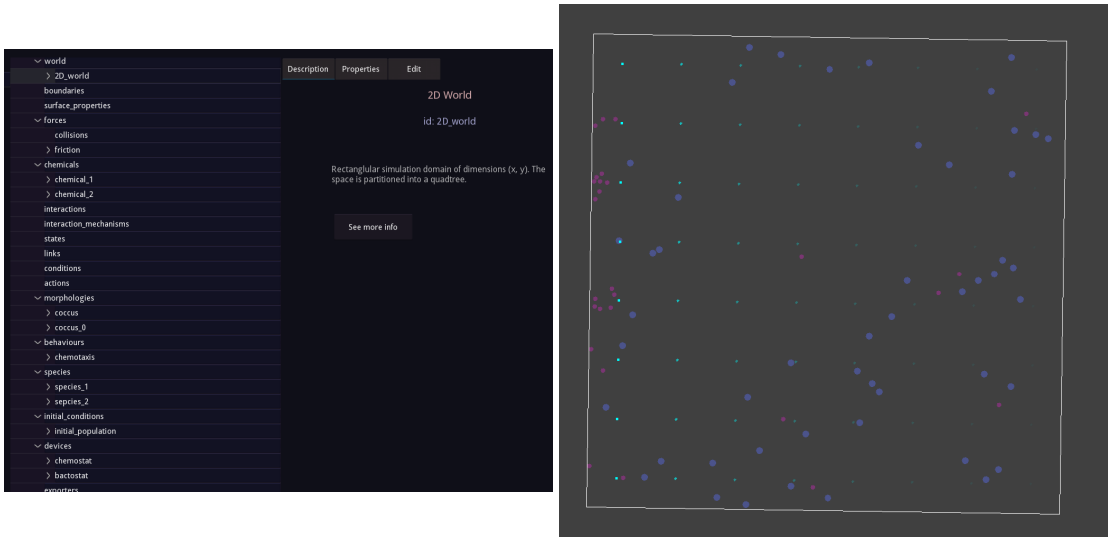
Press run, and you should see a static population of species_1, and species_2 cells coming into the simulation domain from the right hand size (X Max interface), which move around and hunt down the high concentration of chemical_1, causing them to migrate left to the X Min interface.

To add the membrane and metabolic pathway to species_2, do the following:

1. To behaviours, add a membrane module with default parameters.

2. Open the membrane module properties in the model editor, and add a membrane_flux of chemical_1, with a flux=0.1, osmotic=false(OFF), poisson=true(ON), interpolated=false(OFF).

3. To behaviours, add a gillespie module with default parameters.



4. Open the gillespie module in the model editor, and add a reaction (right click on reactions and click +). The id is the reaction name, call it something like "my_reaction". Set the reactants to be "chemical_1" and the products to be "chemical_2", with a rate of 0.1.

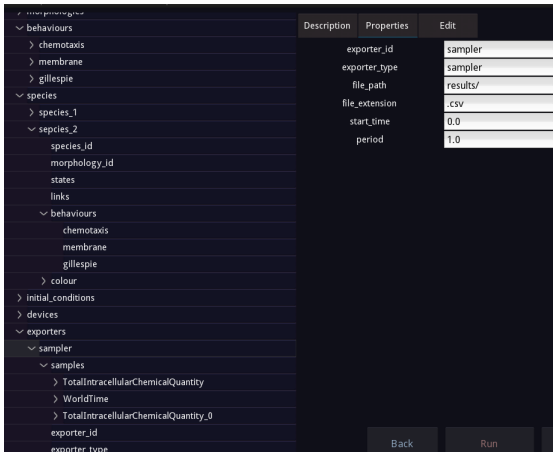
5. Add the gillespie and membrane modules to the cell species_2 behaviour definitions.

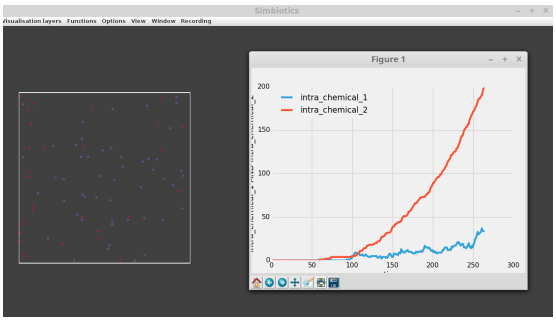
6. To exporters, add a sampler with the file path set to "results/"

7. Open the sampler in the model editor, and add 3 samples: WorldTime with title "time", and 2 TotalIntracellularChemicalQuantity modules, one for chemical_1 called "intra_chemical_1" and one for chemical_2 called "intra_chemical_2".

If you run the model with live plotting, and run with a custom graph, plot data file="results/sampler.csv", x="time" and y="intra_chemical_1 AND intra_chemical_2". You should see a similar output to below.

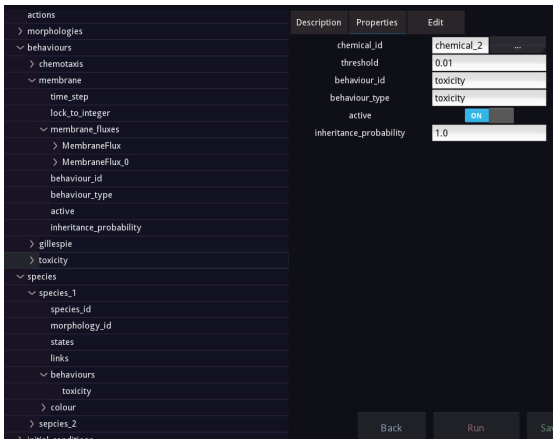
Next, we must add the transportation of chemical_2 across bacterial species_2's membrane, with a rate of 0.5 and set as osmotic process (osmotic on, poisson on, interpolated off). This causes chemical_2 to diffuse into the extracellular space. We then define a toxicity module in in our behaviour definitions, setting the

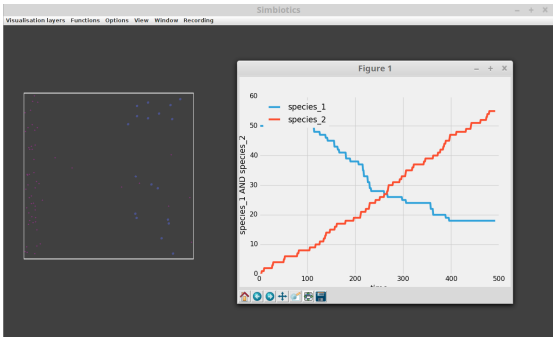




chemical_id to be chemical_2, and the threshold to be 0.01. We must then add the toxicity module to bacterial species_1's behaviour list.

Additionally, we'll add two new samples to our sampler (in exporters). Add two CellNumber modules, one for species_1 and one for species_2, with their titles the same as their species_ids. The result should be that after some time species_1 cells begin to die caused by species_2 cells secreting the toxic chemical_2 product. Below we plot the cell numbers over time.





Tutorial 4 - Cellular logic and decision making with Triggers

One way to represent cellular logic is with triggers. Triggers are composed of conditions and actions. If all conditions are true, then all actions are executed.

As a brief example, we will create a population of cells which may adhere to a surface with two types of interaction. We will tell the cell to 'differentiate' by changing colour depending on the dominant interaction it is having with the surface.

1. To world, add a $3D\_world$ with default parameters

2. To boundaries, add a $solid\_boundary$ with $axis = Y, axis\_face = MIN, property\_id$ $adhesive$

3. To surface_properties, add an $adhesive$ module with $surface\_structures =$ $structure1\ AND\ structure2$

4. To forces, add a $collisions, interaction\_force, brownian$ and $friction$ module with default parameters, then add a $gravity$ module with $gravity\_constant =$ 0.25

5. To behaviours, add a $cell\_adhesion$ module with default parameters - then add a $site$ with $site\_id = adhesin$ (via the model editor drop down menu of the $cell\_adhesion$ behaviour module)

6. To interaction_mechanisms, add two $spring\_mechanism$ modules, and set $rate = 50$ for both

7. To interactions, add a two $specific\_interaction$ modules, the first should have $partner\_a = adhesin$ and $partner\_b = structure1$, with $mechanism\_id =$ $spring\_mechanism$. The second should have $partner\_a = adhesin$ and $partner\_b = structure2$, with $mechanism\_id = spring_mechanism_0$

8. To conditions, add two $has\_interactions$ modules, the first with $interaction\_id =$ $specific\_interaction$, $relation$ => and $value = 1$. The second with $interaction\_id = specific\_interaction_0$, $relation$ => and $value = 1$

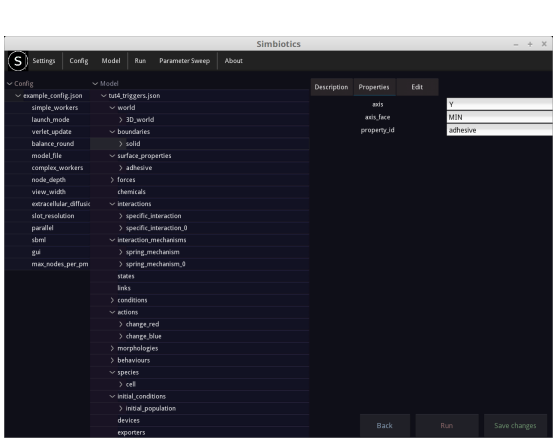
9. To actions, add two *change_colour* modules, set the first one to have *action_id* = *change_red*, and an RGBA value of 100, 0, 0, 0. The second should have *action_id* = *change_blue*, and an RGBA value of 0, 0, 200, 100.

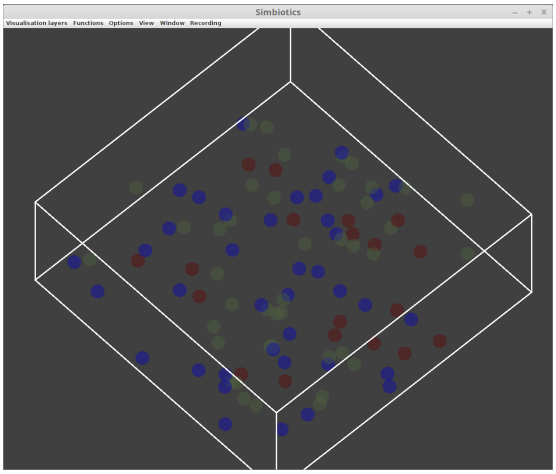
10. To behaviours, add two *trigger* moudles, for the first set *conditions* = *has_interactions*1, *actions* = *change_red*. For the second set *conditions* = *has_interactions*2, *actions* = *change_blue*

11. To morphologies, add a *coccus* module with default parameters

12. To species, add a *cell*. Set *morphology_id* = *coccus*, and in the model editor attach the three behaviour modules (cell_adhesion, trigger and trigger_0) to the species. Also, set the *colour* property of the cell to be 80, 100, 60, 100

13. To initial_conditions, add an *initial_population* module with *species_id* = *cell* and *population* = 100





The resulting model will be a population of cells which are motile, and when they stick to the surface they will either turn blue or red, signifying which of the two interactions is dominant.

The model works as follows: cells may adhere to the Y MIN surface (bottom domain boundary), this occurs by their *adhesin* interacting with either *structure*1

or *structure*2 on the boundary. The cells have *triggers* which check which interaction is dominant, and the cell changes colour to indicate this. A trigger consists of *conditions* and *actions*, if all the conditions are true, then all the actions are executed.

The parameters to note are *rate* property of the *spring_mechanism* interaction mechanism module. This is the rate at which that interaction occurs, we originally set it to 50% for both interactions, therefore we got roughly an equal distribution of red and blue cells. The *cell_adhesions* property $max\_surface\_bonds$ is the maximum number of interactions the cell may have with the surface (domain boundary). The default value for this is 3, which is why in the conditions we check if the number of a given interacton is > 1 (as a cell could have two type 1 interactions and one type 2 interaction, visa versa, or have all 3 interactions 'occupied' by a single type of interaction.)

You can test out changing these parameters and seeing how the model behaviours - you could even say that a given type of interaction has a weak *spring_constant* meaning the interaction could be reversed (the spring can break easier due to the random motion on the cells).

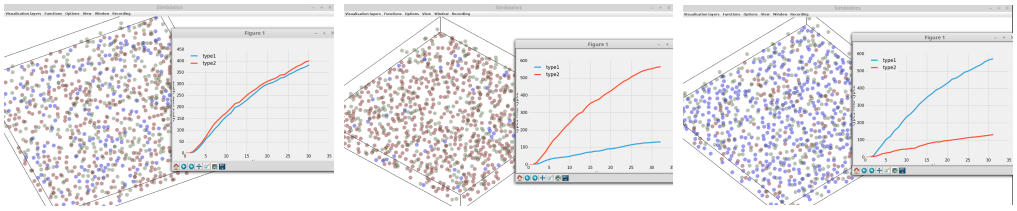
To plot a graph showing the number of red and blue cells can be achieved by adding a *sampler* to the exporters definitions, and adding two *CellCondition* modules. The first should have $conditions = has\_interactions1$, and set the title to something like *type*1, the second should have $conditions = has\_interactions$ and $title = type2$. Again, you may wish to add a *export_periodically* module to the schedules (alternatively press 'a' on the keyboard to write all buffered results to file).

Try playing around with the parameters to see what you get, bellow we show results for a some different rates of the type1 and type2 interactions.

Tutorial 5 - Intracellular dynamics using SBML files

Another way to represet cellular logic/dynamics is through an SBML model. An SBML model can be loaded as a behaviour - as with all modules, each cell has its own 'version' of this model.

Before we start the SBML tutorial, note the following steps when using SBML



(a) (b) (c)

Figure 20: **(a)** rate of type 1 = 50, rate of type 2 = 50 **(b)** rate of type 1 = 25, rate of type 2 = 75 **(c)** rate of type 1 = 75, rate of type 2 = 25

files:

1. Each file should only have a single compartment (which represents a cell's volume)
2. Any species you want to diffuse in/out of the cell need to be defined in the Simbiotics model (with the exact same ID!)
3. Those diffusable species will be handled by the Simbiotics membrane transport system that you'll define, therefore you do not need membrane related reactions in your SBML model.
4. Remember, you can just use 1 SBML file, and create many unique instances of it, or you may use many SBML files (to define multiple species)
5. PLEASE CHECK THE SBML FILE VALIDITY. You can use this link .
6. The SBML simulator Simbiotics uses, libSBMLsim, does not handle SBML events

SBML is integrated into Simbiotics by calling the SBML simulator (libSBMLsim) to solve each cell's intracellular dynamics. The SBML simulator is called every *time_step* of simulation time (we'll discuss the parameters below), and the solver has its own internal *sbml_time step*, which is solved for the whole *time_step*. Simbiotics then integrates the SBML simulator result.

For this exercise, we will create 2 species of cells using 2 SBML model files. These files can be found in $\$SIMBIOTICS/examples/$, called "tut5_species1.xml" and "tut5_species2.xml".

The species 1 SBML model has a reaction which turns chemical A into chemical B, and the species 2 SBML model has a reaction which turns chemical B into chemical C. The A to B reaction is set to be at a rate double that of the B to C reaction.

1. To world, add a $3D\_world$ with default parameters

2. To forces, add a *brownian* and $friction$ module with default parameters

3. To chemicals, add 3 *chemical* modules, called S1, S2 and S3. All of them shouldd have $diffusable = ON$, $diffusion\_rate = 10$ and $degradation\_rate = 0$

4. To behaviours, create a *membrane* module. Add two $membrane\_fluxes$, the first should transports S1 inside the cell at a rate of 1.0, and set only *poisson* to be ON. The second should transport S2 outside the cell at a rate of -1.0, and again set only *poisson* to be true.

5. To behaviours add another *membrane* module, add a flux that that transports S2 at a rate of 1.0 and one that transports S3 at a ratio of -1.0. Again, both should only have *poisson* set to be ON.

6. To behaviours, add two *sbml* modules. For the first, set the $sbml\_file$ to $examples/sbml/tut5\_species1.xml$, and the $time\_step$ to be 0.1 and $sbml\_time\_step$ to 0.01. The second should be the same except its $sbml\_file$ should be set to $examples/sbml/tut5\_species2.xml$. Note: the $time\_step$ variable is the step between the SBML model being solved, and the $sbml\_time\_step$ is the internal time step for the SBML solver

7. To morphologies, add a *coccus* module

8. To species, add two *cell* species modules. Try and guess what's next - we're going to attach the first *membrane* and the first *sbml* module (the ones

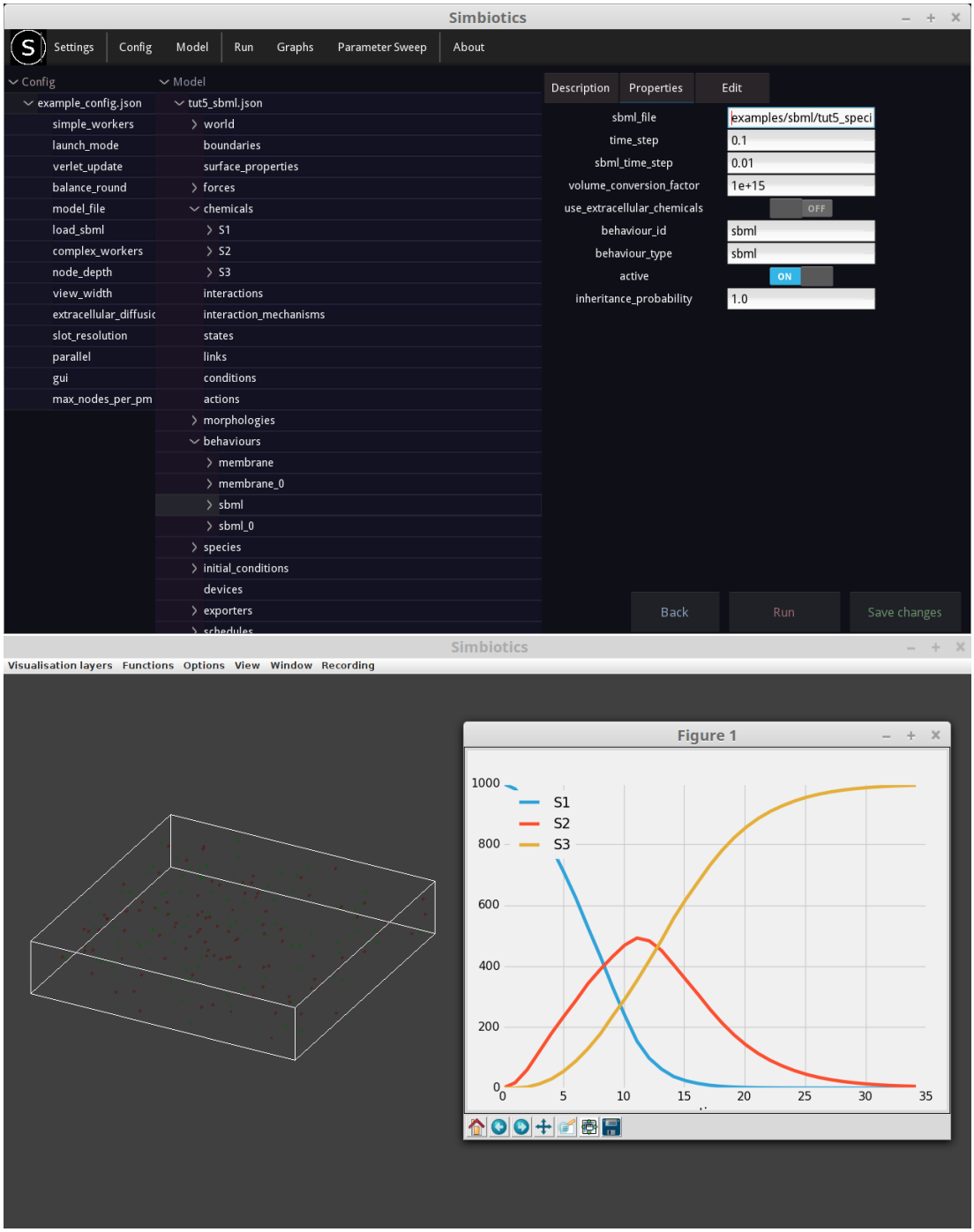
that deal with S1 and S2, to the first *cell*, and to the second *cell* we add the second *membrane* and *sbml* modules, which deal with S2 and S3.

9. To initial_conditions, add two *initial_population* modules, creating 100 of each of the *cell* species.

10. To initial_conditions, add an *initial_chemicalß$_q$uantity*, and set it to be 1000 molecules of S1. (These molecules will be placed at the *position* defined, it's 0, 0, 0 by default, which is the very center of the simulation domain).

11. To exporters, add a *sampler*, and add $TotalChemicalQuantity$ samples for all 3 chemicals S1, S2 and S3. If you want, you could also add a $TotalIntracellularChemicalQuantity$ and a $TotalExtracellularChemicalQuantity$ for each of them too.

12. To schedules, add an *export_periodically* module.


Simbiotics
Settings
Config
Model
Run
Graphs
Parameter Sweep
About
Config
example_config.json
simple_workers
launch_mode
verlet_update
balance_round
model_file
load_sbml
complex_workers
node_depth
view_width
extracellular_diffusic
slot_resolution
parallel
gui
max_nodes_per_pm
Model
tut5_sbml.json
world
boundaries
surface_properties
forces
chemicals
S1
S2
S3
interactions
interaction_mechanisms
states
links
conditions
actions
morphologies
behaviours
membrane
membrane_0
sbml
sbml_0
species
initial_conditions
devices
exporters
Description
Properties
Edit
sbml_file
examples/sbml/tut5_speci
time_step
0.1
sbml_time_step
0.01
volume_conversion_factor
1e+15
use_extracellular_chemicals
OFF
behaviour_id
sbml
behaviour_type
sbml
active
ON
inheritance_probability
1.0
Back
Run
Save changes
Simbiotics
Visualisation layers
Functions
Options
View
Window
Recording
Figure 1
1000
800
600
400
200
0
S1
S2
S3
0
5
10
15
20
25
30
35

Tutorial 6 - Intracellular dynamics using differential equations

Intracellular dynamics can also be specified using differential equations. The *grn* behaviour module is used to do this. Note: despite the name 'grn' it can be used to represent things other than a gene regulatory network, for example it may be used to modify an agents mass (it can be used for biomass growth).

To exemplify use of a differential equation module, we will create a population of a single bacterial species, in an domain (world) filled with chemical S1. The cell species can uptake the chemical though its membrane, and consumes it to both grow in mass and synthesize chemical S2, which it secretes out into the extracellular space. We'll set chemical S1 not to degrade in the extracellular space, and chemical S2 to degrade quite fast in the extracellular space.

You can find the model at "$SIMBIOTICS/examples/models/tut6_equations.json". We will not go through all the steps of the model building as they can be found above, rather we'll focus on the differential equation module.

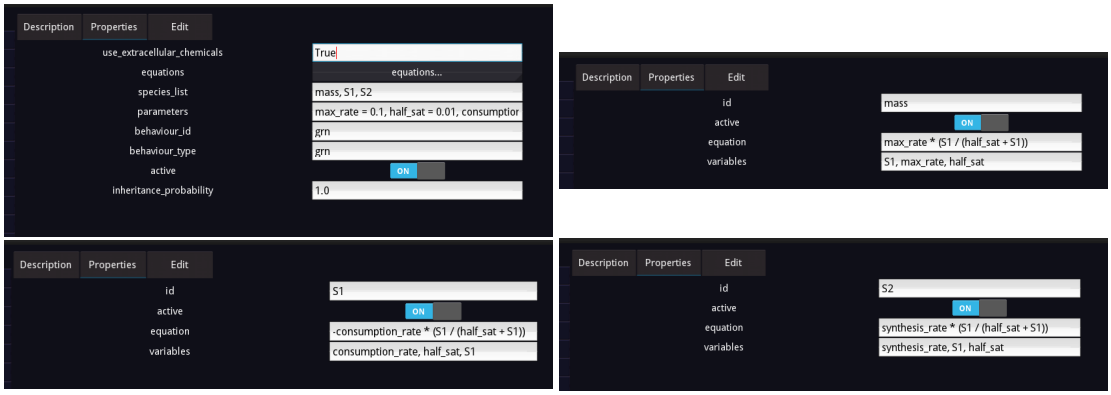
Create a world set up with a single cellular species in it, and defined the two chemical species, set S1's degradation rate to 0.0, and S2's degradation rate to 0.1. Add a membrane flux so that S1 is transported into the cell at a rate of 1.0, and S2 out of the cell at a rate of -1.0, both with the poisson sampler set to be on.

Now, create the *grn* behaviour module. You must first set the $species\_list$ - as we'll be working with S1, S2 and modifying the cell's $mass$ (which is an accessible property via its ID), we must define those three, separated by a comma:

$species\_list = mass, S1, S2$

Now that we have defined the species, we must add some equations. Right click on the *equations* property in the *grn* module (in the model editor view). The equation *id* should be the species you are modifying (either mass, S1 or S2). The *equation* field sets the calculation to work out $\frac{dS_i}{dT}$. It may be an expression with variables/constants too, for example you may refer to any of the species in the $species\_list$ you just defined. You may also refer to a custom named variable, for example k, and you must define it in the *parameters* field, in the form of $variable = value$.

An example can be seen in the figure below of us setting the equations for this system.

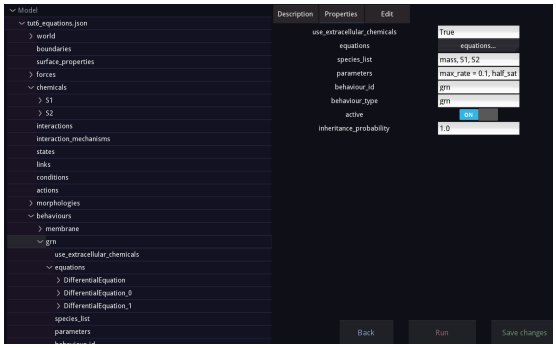


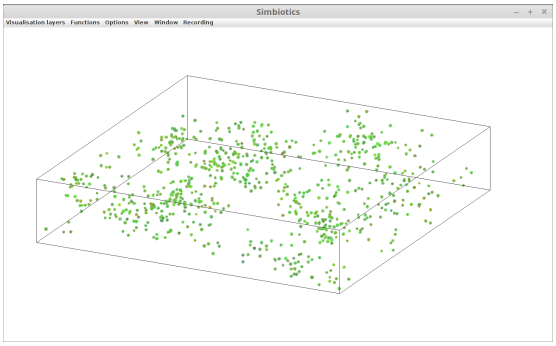
Here are all the values for the *grn* behaviour module:

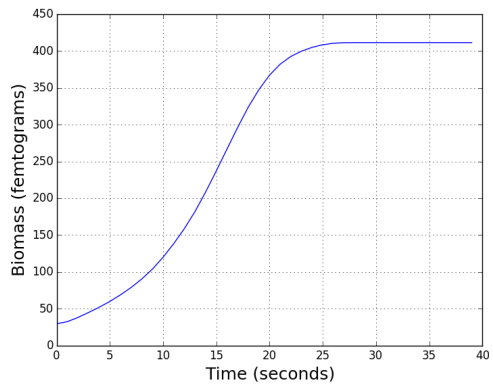
1. species_list = mass, S1, S2
2. parameters = max_rate = 0.1, half_sat = 0.01, consumption_rate = 0.1, synthesis_rate = 0.1
3. Equation 1, id: = mass
4. Equation 1, equation: = max_rate * (S1 / (half_sat + S1))
5. Equation 1, variables: = S1, max_rate, half_sat
6. Equation 2, id: = S1
7. Equation 2, equation: = -consumption_rate * (S1 / (half_sat + S1))
8. Equation 2, variables: = consumption_rate, half_sat, S1
9. Equation 3, id: = S2
10. Equation 3, equation: = synthesis_rate * (S1 / (half_sat + S1))
11. Equation 3, variables: = synthesis_rate, S1, half_sat

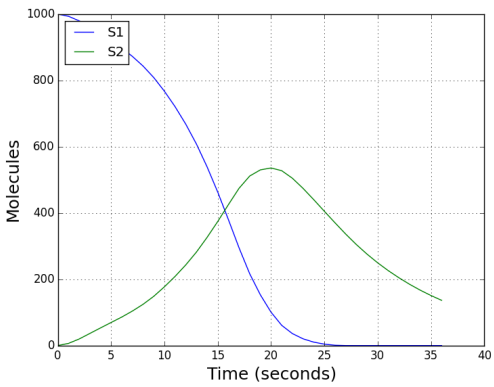
Now create an initial population of the cell species, and add an initial amount of S1 into the extracellular space. Also, don't forget to attach the behaviours you defined to the cell species! (Plus attach some data exporters if you want to generate some graphs)

You can run the model get a result similar to what we see below. The cells should stop growing and producing S2 once all of S1 is consumed - the remaining S2 then degrades in the extracellular space. Try playing around with the parameters, or add another species which consumes S2.









Tutorial 7 - Live graph plotting

Live graph plotting can be achieved by defining graph objects and attaching them to a simulation run. The defined graphs can be saved to file and loaded again for easy reuse.

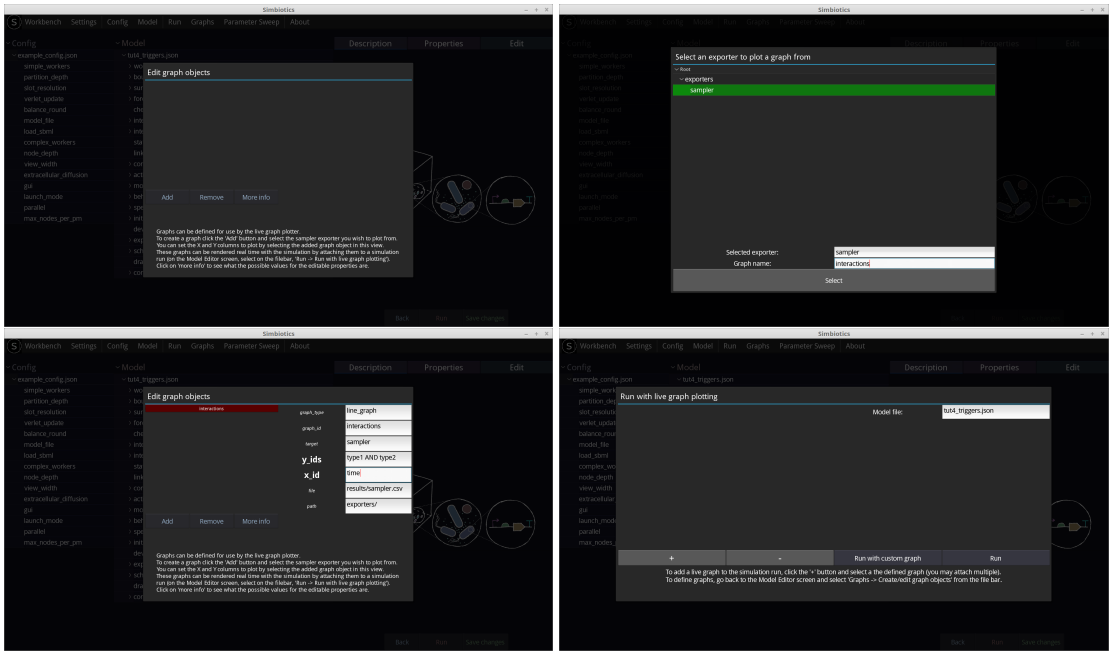
In this tutorial we'll reuse the model from Tutorial 4, and build the graph objects to render our graphs, rather than using the custom graph mode as we did in Tutorial 4. So, load up that model (a copy of it can be found at *examples/-models/tut4_triggers.json*.

To create a graph, click on *Graphs* on the file bar, and select *Create/edit graph objects*. Click on *Add*, and select the *sampler* exporter. You can name the graph, in our case we call it *interactions* (see figures below), then press *Select*.

Now you have added a graph for that exporter, you can set what X and Y values you want to plot. Click on the graph in the list view, and it will bring up the properties on the right hand side. You may set the **bold** variables. *y_ids* are the columns in the data file to be plotted on the Y axis, and *x_id* is, as you can guess, is the column to be plotted for the X value. The values for *y_ids* and *x_id* are the sample titles that you wish to plot (the titles become the column headers in the *.csv* file which is exporter by the sampler). To see your sample titles, go back to the model editor view, open the nodes for *exporters - sampler - samples*, then you can click on each individual sample, and see/set its title in the property view.

Note: the *x_id* should only be one value (one sample title), but the *y_ids* may be set to be many values. This is achieved by chaining together sample titles separated by *AND*. See the figures below for clarification on this.

For our graph we'll plot *time* on the X axis, and for the Y axis we'll plot *type1 AND type2*. Once you've set the X and Y columns to plot, go back to the model editor. We can now run a simulation with this graph attached to the run. To do this, select *Run - Run with live graph plotting* from the file bar. Now you can click the + button, and select your graph, then press *Attach*. You can now run the model, and you should have a lovely live graph alongside your simulation!


Simbiotics
Edit graph objects
Add
Remove
More info
Graphs can be defined for use by the live graph plotter.
To create a graph click the 'Add' button and select the sampler exporter you wish to plot from.
You can set the X and Y columns to plot by selecting the added graph object in this view.
These graphs can be rendered real time with the simulation by attaching them to a simulation
run (on the Model Editor screen, select on the FileBar, 'Run -> Run with live graph plotting').
Click on 'more info' to see what the possible values for the editable properties are.
Select an exporter to plot a graph from
exporters
sampler
Selected exporter:
Graph name:
interactions
Select
Run with live graph plotting
Model file:
tut4_triggers.json
Run with custom graph
Run
To add a live graph to the simulation run, click the '+' button and select a the defined graph (you may attach multiple).
To define graphs, go back to the Model Editor screen and select 'Graphs -> Create/edit graph objects' from the file bar.

Tutorial 8 - Parameter sweeps

Parameter sweeps can be conducted. The value of a selected parameter is iterated across a defined range, and a simulation is run for each value in that range. The data for each individual simulation is stored in its own subfolder for further processing. You can also run live graphs/post rendererd graphs with the parameter sweep, where each individual simulation data is plotted on the same graph for easy comparison.

Many parameter sweeps can be defined, and you can set the to run in a combinatorial manner to explore the entirity of the parameter space (all combinations of the attached parameter sweeps are simulated), or alternatively you can run them indepedently to observe the effect of each indiviudal parameter on the system.

In this tutorial we'll use the model as we left it after the previous tutorial (Tutorial 7), so load that up. To create a parameter sweep, click on *Parameter sweeps - Creat/edit parameter sweep objects* on the file bar of the model editor. Same as for the graphs, press *Add* and select the associated model object. For parameter sweeps you can only select properties with a numerical value. In this tutorial we'll select the *rate* variable for the *spring_mechanism*.

Once you have your parameter object defined, click on it in the list view, and edit the properties which appear on the right hand side. The *range* property can take values which match the following forms:

range = [A,B,C,...,Z]
range = [A-Z]

Where A-Z are placeholders for numerical values. And the *interval* is a numeric value describing the interval between each value when iterating through the range. If you use the first form then the *interval* setting is ignored, as each value (separated by a comma) is iterated through, so you can leave the parameter value empty. For this tutorial, we want to set the interaction rate (of *type1* interactions occuring) to be 0, then 50, then 100. There are two ways we can do this, choose either:

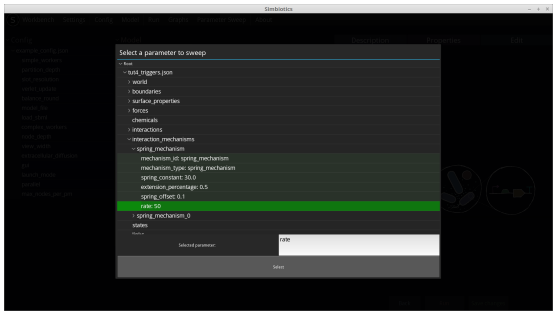
range = [0, 50, 100]
range = [0-100], interval = 50

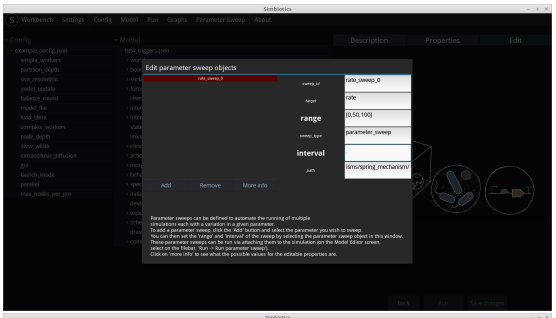
Now that you have your parameter object and its properties set, you can go back to the model editor view, and press *Run - Run parameter sweep* from the file bar. To attach the defined parameter sweep object to this simulation run, press the + button and attach your sweep object. You're now good to go! But first, lets look at the options we have:

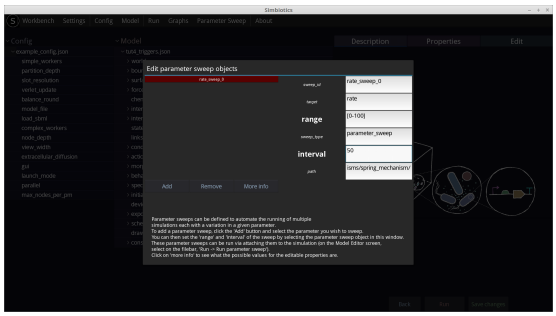
Run each sweep separately? If there are multiple parameter sweep objects attached to a run, you can either sweep each parameter individually, or you can run all combinations of all sweeps. (ON runs them separately, OFF runs all combinations).

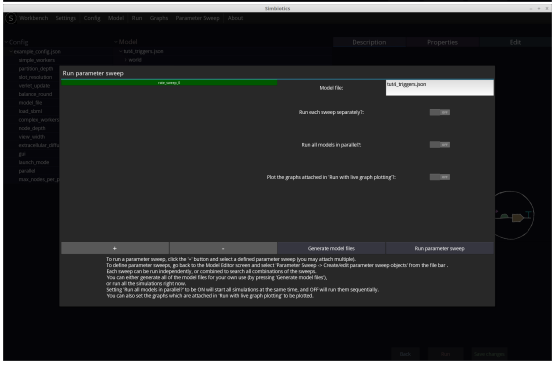
Run all models in parallel? Sets whether to run all simulations at the same time, or whether to run them sequentially one after another. Be careful if you run all simulations at the same time (in parallel), as this could consume a lot of RAM and potentially freeze your machine, so please determine how much RAM a single simulation consumes before thinking about running many at the same time. (OFF runs them one after another, ON runs them at the same time)

Run the graphs attached in 'Run with live graph plotting' If you have graphs attached then these can also be run with the parameter sweeps. (ON renders the graphs, OFF doe... forget it, you get the picture)


Simbiotics
Select a parameter to sweep
Selected parameter:
rate
Select


Simbiotics
Edit parameter sweep objects
range
interval
Add
Remove
More info


Simbiotics
Edit parameter sweep objects
range
interval
Add
Remove
More info


Simbiotics
Run parameter sweep
Model file:
Run each sweep separately?:
Run all models in parallel?:
Plot the graphs attached in 'Run with live graph plotting'?:
Generate model files
Run parameter sweep

Appendix C - Population Dynamics of Autocatalytic Sets in a Compartmentalized Spatial World

This appendix presents the publication associated with Chapter 9. The manuscript shown below was primarily written by Wim Hordijk, for which I developed the model for.

.12 Introduction

Autocatalysis, i.e., the ability of molecules to catalyse their own synthesis, is a hallmark of virtually any origin of life scenario, since it is the chemical equivalent to biological replication—the fundamental feature of living entities to "make more of themselves". Yet, such autocatalytic molecules, directly catalysing their own production, are rare, and it is unlikely that life kick-started with such "selfish" autocatalytic chemicals. However, autocatalysis can also be obtained at a systems level, if a chemistry features a set of mutually catalytic molecules in which the formation of every member is catalysed by other members of the set. Such a set is then able to collectively catalyse all its constituents, even if none of its members is a true (i.e., direct, or "selfish") autocatalyst [92, 114, 163].

The study of such collectively autocatalytic sets (CAS) has revealed that they are likely to emerge spontaneously in sufficiently diverse chemistries, even under modest catalytic activity [91], and that they are able to dynamically upconcentrate their members in order to maintain themselves. While CAS are able to draw

resources from potential competitors [90], they have been criticised for displaying little to no evolvability [217]. The argument goes that once an autocatalytic cycle establishes in a random chemistry, there is nothing that destabilises this cycle in order to make room for the emergence of other autocatalytic cycles, since the concentration of all involved molecules increases exponentially. Even if a random chemistry allows for multiple catalytic cycles as hypothetical individual units of selection, these would eventually just coexist, leaving no room for further evolutionary adaptation.

It has been suggested that the limited evolvability of CAS could be overcome by embedding autocatalytic sets into compartments [114, 218]. Such encapsulated reaction systems are able to draw resources from and potentially release products back into the environment. Encapsulation with environmental coupling, so the claim, might reconstitute selection among competing CAS, since different compartments can host different active autocatalytic cycles, which can be destabilised through resource competition and random fluctuations during compartment division. As Kauffman, who introduced the concept of CAS, writes [114]: "Theoretical work and experimental work on CAS both support their plausibility as models of openly evolvable protocells, if housed in dividing compartments such as dividing liposomes." This intuition has recently been confirmed by computational investigations that put CAS into flow reactors in order to mimic encapsulation in semi-permeable compartments [218].

Encapsulated reaction systems have been studied extensively in the origins of life context under the term protocells [61, 175, 191]. Protocells are simple metabolisms occurring within compartments (e.g., lipid or fatty acid vesicles) that have the capacity for growth and self-replication. Potentially equipped with inheritable chemical "information" they are generally regarded as primitive units of (limited) evolution (through compositional inheritence), eventually leading to true open-ended Darwinian evolution. While protocells have not yet been fully implemented in the laboratory, both theoretical and experimental investigations have uncovered numerous necessary requirements about the involved chemicals and their coupling [23, 31, 44, 49, 152, 174].

Related to protocells are models from the realm of the lipid world, where an explicit covalent metabolism is replaced by conceptually simpler cross-catalytic asso-

ciation and dissociation dynamics of compartment forming amphiphilic molecules [189]. It has been demonstrated that even in such lipid models, random network properties of the constituting molecules can give rise to heredity [189], speciation [78], and population dynamics [148], although it has been argued whether systems lacking covalent chemistry are able to undergo full evolution [147, 218].

So far, though, relatively little attention has focused on studying collectively autocatalytic sets as metabolisms and inheritable information for protocells. A few studies have shown that such "autocatalytic protocells" indeed have the ability, in principle, to synchronise their internal metabolism and membrane dynamics, and may evolve [90, 192, 218, 220]. However, these studies on autocatalytic sets did not explicitly model populations of protocells in a spatial environment. Here, we make an important first step in this direction by using a recently developed software tool to simulate the emergence and dynamical behavior of autocatalytic sets in a population of simple compartments that exist in a spatially explicit world. We present several illustrative initial results, discuss how these could be relevant in the context of the origin and early evolution of life, and provide suggestions for further work, in combination with experimental studies.

.13 Background

The concept of autocatalytic sets was originally introduced by Kauffman [111, 112, 113], and subsequently formalised and further developed as RAF theory [91]. An autocatalytic set (or RAF set) is defined as a set R of reactions and associated molecules that is:

1. *Reflexively autocatalytic* (RA): each reaction in R is catalysed by at least one molecule from R itself; and

2. *F-generated* (F): all reactants in R can be created from some food set F by using a sequence of reactions from R itself.

The food set F is a set of molecule types that are assumed to be available from the environment. This notion of autocatalytic sets has been defined mathematically more rigorously, and an efficient (polynomial-time) algorithm for finding RAF

sets in general reaction networks has been developed [88, 93, 95]. RAF theory has been applied extensively to simple polymer-like models of chemical reaction networks, showing that autocatalytic sets are highly likely to exist at chemically realistic levels of catalysis, and under a wide variety of model assumptions [88, 90, 93, 156, 199]. Importantly, these results show that autocatalytic sets often consist of a hierarchy of smaller and smaller autocatalytic subsets, i.e., smaller subsets of reactions that themselves are RAF sets [94, 95]. Finally, the formal RAF framework has also been applied successfully to analyse real chemical and biological reaction networks [89, 200].

Many of these earlier results are based on a simple model of reaction networks known as the binary polymer model [59]. In this model, molecules are represented by bit strings up to a maximum length n, with the food set consisting of all bit strings up to a small length t (usually $t = 2$, i.e., the monomers and dimers). The chemical reactions consist of the possible ligations (gluing two bit string together into a longer one) and cleavages (cutting a bit string into two smaller ones). Finally, catalysis is assigned randomly, with a fixed probability p that a given molecule (bit string) catalyses a given reaction (a ligation and its corresponding cleavage). Figure 26(left) shows an example of a RAF set R, consisting of eight reactions, that was found in an instance of the binary polymer model with $n = 5$, $t = 2$, and $p = 0.0045$, with some of its RAF subsets indicated by the coloured polygones.

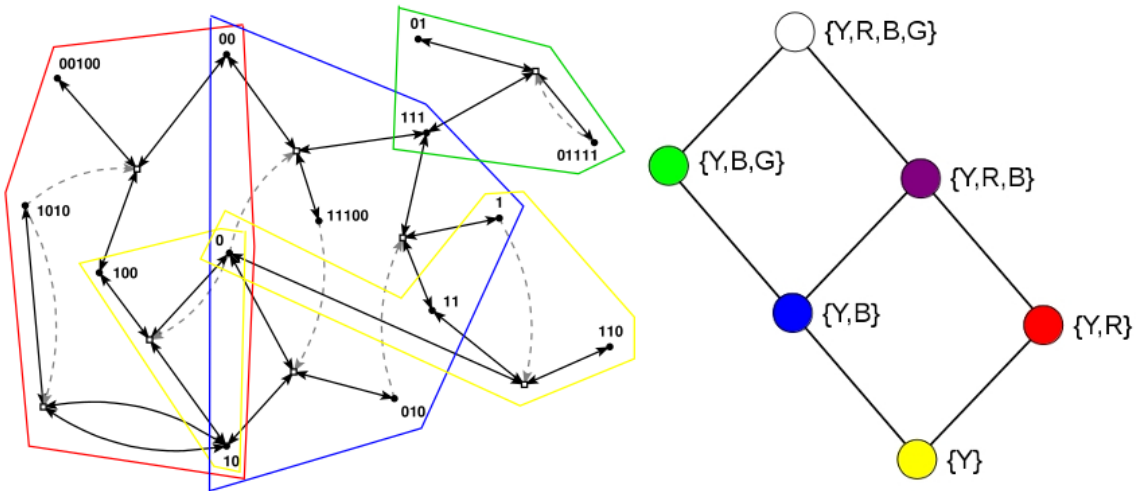


Figure 26: **RAF set example.** Left: An example of a RAF set as found in an instance of the binary polymer model. Black dots (labeled with bit strings) represent the molecule types, and white boxes represent reactions. Solid black arrows indicate molecules going into and coming out of a reaction, while dashed gray arrows indicate catalysis. Coloured polygones indicate some of the RAF subsets (see text). Right: The six closed RAFs (colour coded) and their mutual subset relationships.

Note that catalysis is considered an "all-or-nothing" feature in this context. However, (relative) catlaysis rates can also be taken into account , as well as inhibition, i.e., molecules that *prevent* reactions from happening [95]. In the simulations described below, (relative) reaction rates are explicitly used, and in one instance a form of inhibition is also included.

Recently it was argued that the main RAF subsets of interest, in particular in the context of the origin of life, are the so-called closed RAF sets, in which all reactions for which a catalyst is present (given the set of molecule types currently present in the system) are included [96, 199]. For example, when only food molecules (monomers and dimers) are present initially, only the reactions within the yellow RAF subset can proceed catalysed. This yellow subset thus forms the smallest closed RAF (the "yellow" closed RAF), and is (necessarily) always part of any larger closed RAF as well.

However, for the three reactions in the red RAF subset, not all catalysts are present yet when only the yellow subset exists. One of its reactions will have to

happen uncatalysed to create the required but missing catalyst spontaneously. Of course, all reactions can happen without a catalyst, but they do so at a lower rate, which means there is usually some (stochastic) waiting time before this happens. Once it does happen, though, the red subset comes into existence, with all its reactions proceeding catalysed. Thus, the yellow and red subsets combined form another closed RAF (the "red" closed RAF).

Similarly, the blue subset requires any one of its three reactions to happen spontaneously before the full subset can come into existence. When it does, the yellow and blue subsets combine to form yet another closed RAF (the "blue" closed RAF). When the yellow, blue, and red subsets all exist, they form an even larger closed RAF (the "purple" closed RAF).

Finally, the green subset, which also requires a spontaneous (uncatalysed) reaction reaction, can form an extension of the blue RAF subset, but only once the blue subset itself already exists. Thus, the yellow, blue, and green subsets combined form a closed RAF as well (the "green" closed RAF). When all subsets (yellow, red, blue, and green, i.e., the full autocatalytic set R) exist, they form the largest possible closed RAF (the "white" closed RAF).

These six possible closed RAF sets are shown in the diagram in Figure 26(right) with their respective colours, the combination of RAF subsets they are made up of, and where an edge between two nodes means that the closed RAF at the lower end of the edge is a direct subset of the closed RAF at the upper end of the edge.

Earlier it was shown that the autocatalytic set R in Figure 26(left) actually contains 29 RAF subsets, but that only six of these are closed RAFs (the ones indicated here) [96]. Therefore, from a dynamical point of view, the other 23 RAF subsets are of little interest, as they would immediately expand into the larger closed RAF that they are part of. In other words, RAF subsets that are not closed are transient, whereas closed RAFs are stable over long time spans, until some spontaneous but rare reaction happens that allows an even larger closed RAF to come into existence.

The RAF set R as shown in Figure 26(left) and its six closed RAFs as shown in Figure 26(right) are used here to illustrate, through computer simulations, the emergence of different autocatalytic (sub)sets in a population of compartments that exist in an explicit spatial environment.

.14 Methods

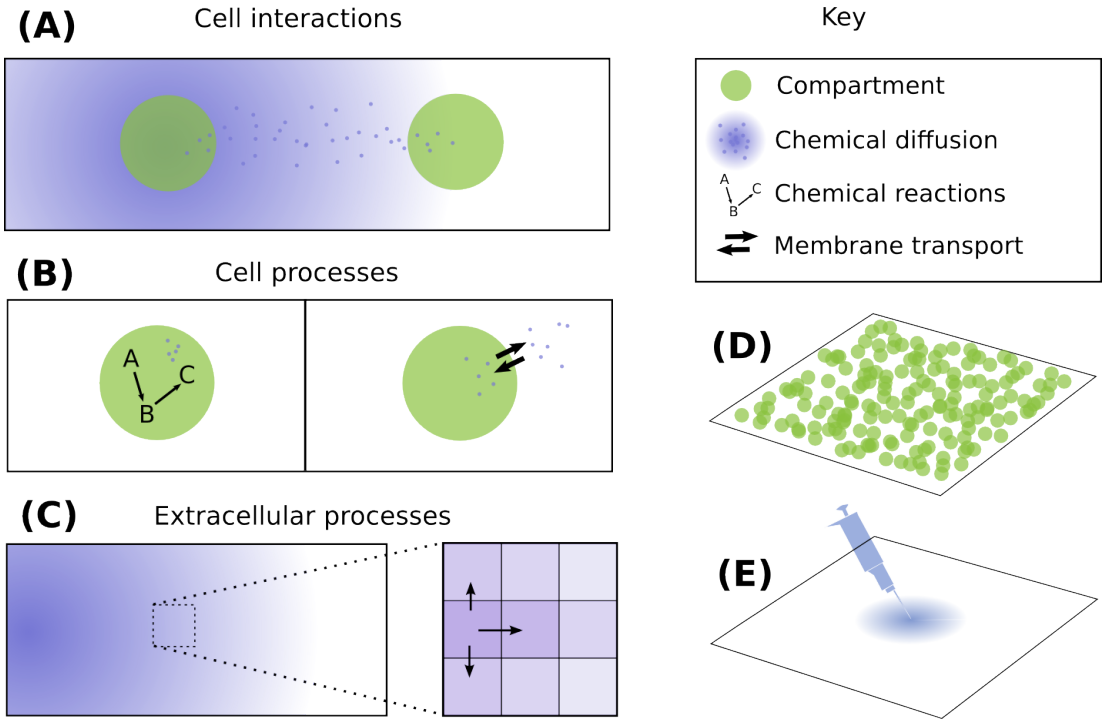


Figure 27: **(A)** The cell interactions in the model, showing that cells communicate via diffusable chemical signals. **(B)** The cell processes in the model. *Left* shows that cells have metabolic pathway activity. *Right* shows that cells have a membrane transports mechanism allowing chemicals to be transporting in and out the cell. **(C)** The extracellular processes in the model are chemical diffusion. **(D)** The initial condition of the model - a well mixed population of cells on a 2D surface. **(E)** Systems are induced by pipetting chemicals into the center of the domain.

Simulations were performed with the Simbiotics package [158]. Simbiotics is a multicellular simulator which represents cells as individual physical entities embedded in chemical gradients, where each cell can have defined dynamics and can interact with its environment as well as other cells. The simulation toolkit provides a versatile set of data collection and analysis tools, and can be easily extended with user-defined modules and libraries. A more detailed description of Simbiotics is available elsewhere [158]. Here, we briefly describe the particular features used in this study.

We model the simulation domain as a 2D rectangle with periodic boundary conditions. Chemicals in the environment (e.g. food molecules) are defined as continuous fields over this rectangle and the usual Fick law is taken to express their diffusion. For computational purposes, Simbiotics rasterises this space into a grid of finite sized voxels and integrates the deterministic diffusion and decay dynamics using a finite difference method. A voxel is a sub-area of the 2D simulation domain, which stores the chemical quantities which exist there, enabling for the representation of localised concentrations and chemical fluxes between neighbouring voxels. An overview of the Simbiotics model can be seen in Figure 27.

A constant flow of monomers and dimers, which we regard as food molecules, is provided by introducing these molecules at the center of the grid at a given constant rate, which then diffuse to neighboring grid locations depending on a given diffusion rate. Molecules diffuse from higher concentrations to lower concentrations. Food molecules in the environment decay with a given constant rate which models outflow of the environment. Depending on the inflow, diffusion, and decay rates, in the absence of any other dynamics, an steady state in the concentration of food molecules over the entire grid is eventually reached.

We introduce compartments by randomly placing spheres of constant radius into the space. Compartments are not allowed to overlap and are immotile throughout the simulation. Each compartment can hold molecules in their interior. Molecules are allowed to permeate compartment membranes if their lengths do not exceed a certain threshold. Permeation is proportional to the concentration difference between the compartment interior and the surrounding local environment (taken as the concentration at the grid cell that the compartment resides in), the compartment surface area, and a permeation rate constant. The actual number of molecules permeating the membrane is sampled from a Poisson distribution whose mean is the permeation rate. In the following, we allow bit strings of up to length two (i.e., all food molecules) to permeate membranes, whereas longer strings are strictly contained within compartments.

Chemical reactions within compartments are based on the reaction network presented in Figure 26(left), i.e., an autocatalytic set that occurs in an instance of the binary polymer model. For simplicity, the simulations only consider liga-

tion reactions. For each such reaction, an uncatalysed and a catalysed instance is included, but with different rate constants such that the rate constant for the uncatalysed reaction instance is lower than that of the catalysed reaction instance. The actual chemical dynamics within each compartment is simulated using Gillespie's stochastic simulation algorithm [72, 73]. A flow diagram of the simulation algorithm is presented in Figure 28.

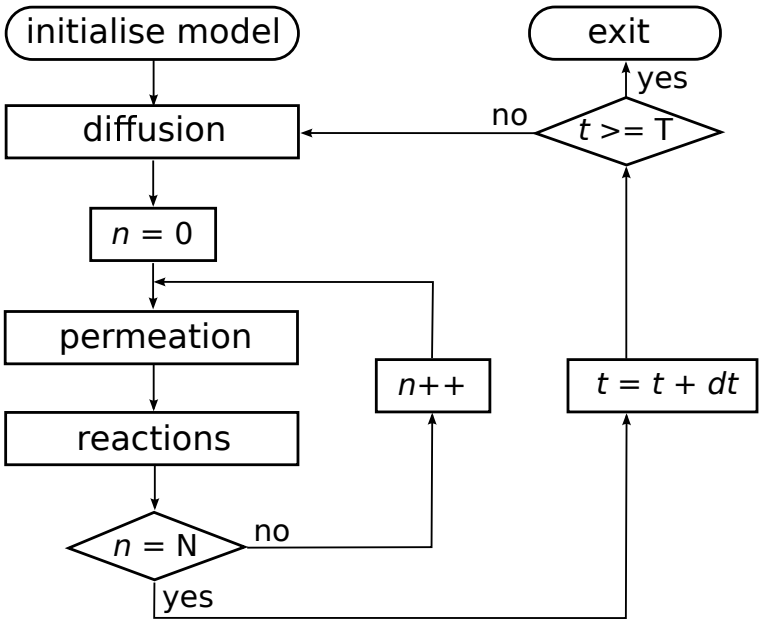


Figure 28: **Flow diagram of the simulation algorithm.** After initializing the model with N compartments, the outer loop iterates the system through time. In each iteration the algorithm first solves for diffusion and decay of molecules in the environment using a finite difference approximation to the Fick equation. Then, for each compartment in the simulation. intracompartment molecular counts are updated by first solving permeation processes, and then running a Gillespie algorithm within each compartment. The algorithm is stopped after T time units have been simulated.

Note that we do not simulate chemical reactions in the environment. This can be justified by assuming that even if some reactions would take place in the environment, the reaction products would mostly diffuse away and out of the environment, and no real sustained chemistry would be possible, other than inside compartments [192].

Figure 29 shows the basic simulation setup at two different time steps. The

spatial environment consists of a 16x16 grid, with 100 randomly distributed compartments (black spheres). Blue spheres indicate the (relative) concentration of food molecules in each of the grid locations. Faint (transparent) spheres indicate low concentration, and bright (solid) spheres indicate high concentration (relative to the grid location with the highest current concentration). Early on during the simulation (left frame), the food molecules are just starting to diffuse throughout the grid, while being introduced into the environment at a constant rate in the center of the grid. Later on during the simulation (right frame), an equilibrium distribution of food molecules as been reached. Food molecules will also have entered the compartments, but there is no actual chemistry going on yet (i.e., all reaction rate constants have been set to zero).

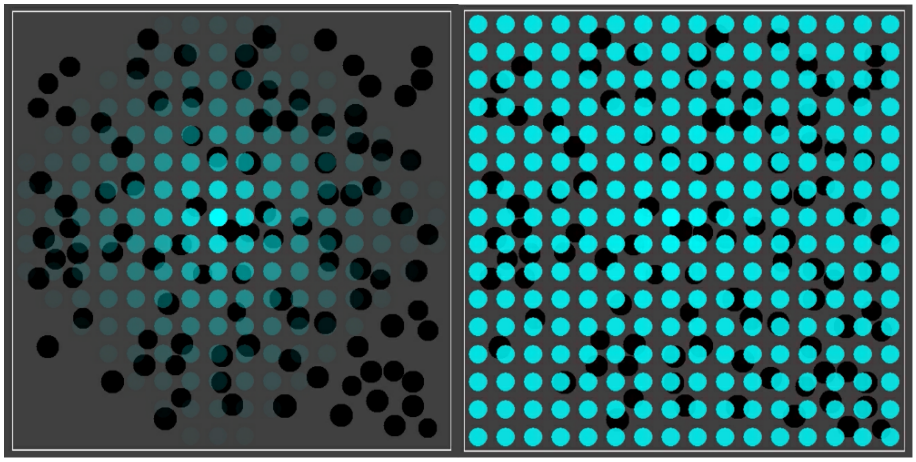


Figure 29: **The basic simulation setup.** A 16x16 grid with 100 randomly distributed compartments (black spheres) and concentration of food molecules (blue spheres) throughout the grid. Left: Shortly after starting the simulation. Right: After an equilibrium distribution of food molecules has been reached.

In most of the simulations presented below the rates of food inflow, diffusion, and decay were set such that, once the equilibrium phase has been reached, there are about five molecules of each food type (i.e., monomers and dimers) within each compartment when no chemistry takes place. When the reaction rate constants are set to positive values, though, reactions will happen within compartments (starting from the food molecules). Over time, the different closed RAFs will appear inside compartments, and each compartment is coloured according to

which closed RAF it currently contains (with a molecule count threshold of two), using a colour scheme as in Figure 26(right). Detailed parameter settings for each of the simulations are given in Appendix B.

It should be noted here that, since the simulated chemical reaction networks come from an abstract polymer model, time and volume are arbitrary. In other words, the time units could, in principle, represent seconds, hours, or even days. However, what is important here is the relative difference between rate constants of catalysed and uncatalysed reactions. The absolute magnitude of these rate constants were chosen such that a single simulation takes just a few minutes, rather than hours. But the parameters can easily be scaled up or down to change the absolute time scale. The overall behavior would remain the same, though.

Similarly, molecular quantities are somewhat arbitrary. We have chosen a threshold of two product molecules to exist before an autocatalytic subset is considered to be present. The idea behind this is that the first product generally has to be produced through a spontaneous reaction, but if there are two or more products, it is highly likely that they have been produced through catalysed reactions, and that the corresponding autocatalytic subset indeed is present in full. Such low molecular counts may seem unrealistic, but again, since the units in the system are arbitrary, these could also be interpreted as, e.g., micromolar quantities. Moreover, the origin of life most likely did happen in a low molecular-count scenario.

Each of the simulations described below can be downloaded from http://ico2s.org/data/extras/compartments/, which also contains the movies referred to in the results section.

.15 Results

.15.1 Dynamics of a single compartment

Before presenting results on simulating a population of compartments, we start by showing the kinds of dynamical behavior that can occur within a single compartment. This will help in understanding the subsequent results.

Figure 30 shows the results of two simulations with just a single compartment,

located at the center of the grid (i.e., where the food molecules flow in). On the horizontal axis in these plots is time (in arbitrary units), and the vertical axis shows total number of molecules. The yellow line represents the number of molecules of type 110 that exist inside the compartment over time. Recall from Figure 26(left) that this molecule type (bit string) is a product of the yellow RAF subset, but one that is not used up in any other reaction. Similarly, the red line represents the number of molecules of type 00100 inside the compartment. This molecule is produced by the red RAF subset, but not used in any reactions. The blue line represents the number of molecules of type 11100, produced by the blue RAF subset. This molecule acts as a catalyst, but not as a reactant in any of the reactions, and is thus also not used up. Finally, the green line shows the number of molecules of type 01111, which are produced (but not used up) by the green RAF subset (although they are also a catalyst).

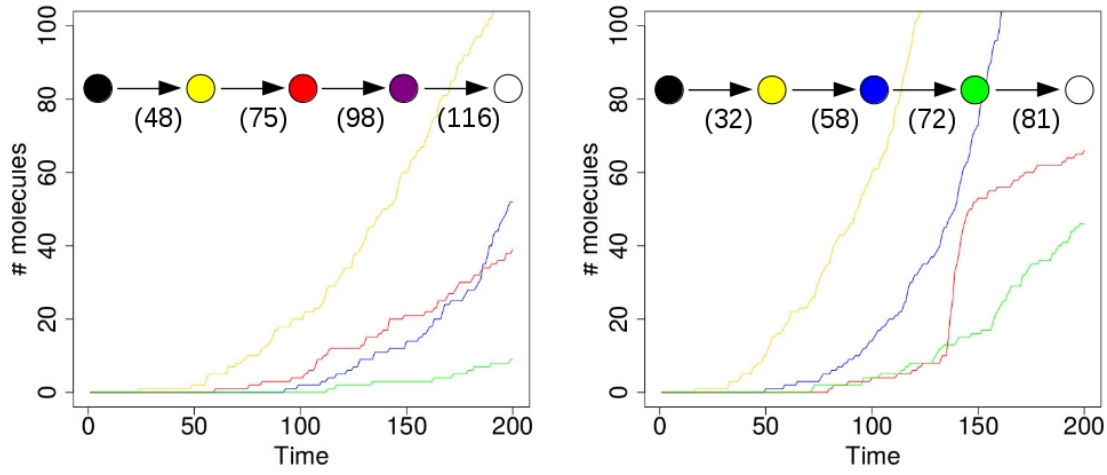


Figure 30: **A single compartment.** Left: A simulation run with a single compartment where the red RAF subset appears first. Right: A simulation run with a single compartment where the blue RAF subset appears first. Insets: The sequence of compartment colour changes in the simulations. Numbers indicate at which time steps during the simulation the respective colour changes happened.

In the simulation that produced the results shown in the plot on the left of Figure 30, the first molecule of type 110 (a "yellow" molecule) is produced at time step 23. At the start of the simulation there are not many food molecules yet, as they only just start flowing into the environment. So, it takes a while

before the first (catalysed) reactions will actually start happening, once enough food molecules are present. The second molecule of type 110 is produced at time step 48, at which point the compartment in the simulation visualization turns from black to yellow. In other words, once there are at least two molecules of type 110, the compartment turns yellow, indicating that the yellow closed RAF (consisting of the yellow RAF subset in Figure 26(left)) is currently present inside the compartment.

However, for the other RAF subsets (red, blue, and green in Figure 26(left)), a spontaneous reaction is required first, as explained above. Since these spontaneous (uncatalysed) reactions happen at a lower rate than the catalysed ones, there is an additional waiting time before any of these RAF subsets comes into existence. In the plot on the left in Figure 30, by chance the red subset comes into existence first, due to a spontaneous reaction. As soon as at least two molecules of type 00100 exist inside the compartment (which happens at time step 75 in this simulation), the compartment turns from yellow to red, indicating that the red closed RAF (i.e., the yellow and red RAF subsets combined) is currently present inside the compartment.

Similarly, the blue RAF subset comes into existence after one of its reactions has happened spontaneously, and as soon as at least two molecules of type 11100 are present (which happens at time step 98), the compartment turns purple, indicating that the purple closed RAF (i.e., the yellow, red, and blue subsets combined) is now present inside the compartment. Finally, the green subset comes into existence (which can only happen once the blue subset exists), and a second molecule of type 01111 is produced at time step 116, at which point the compartment turns white, indicating that the white closed RAF (i.e., all RAF subsets together) is now present. In short, the compartment has gone through the sequence of colours as shown in the inset in Figure 30(left), where the numbers underneath the arrows indicate at which time step the change in colour happened.

However, in the second simulation (shown on the right in Figure 30), there is a different sequence of events. In this case, the blue subset comes into existence first, then the green one, and finally the red one. So, the compartment goes through the sequence of colours as shown in the inset of Figure 4(right). Also note that the time steps of the changes are different between the two simulations,

showing that it truly is a stochastic process.

In these simulations, the difference between the rate constants for catalysed and uncatalysed reactions is kept relatively small (about one order of magnitude), so that the changes in colour actually happen within a reasonable amount of time. However, in real chemical systems this difference will generally be larger (often several orders of magnitude [230]), so the waiting times between colour changes (i.e., new RAF subsets coming into existence) will also be much larger. In fact, in principle it could even be the case that only the red closed RAF actually comes into existence, but never the blue one, or vice versa, if the required spontaneous reaction never happens within the total simulation time.

These simulations confirm the postulated lack of evolvability [217] of this particular RAF: once an autocatalytic set comes into existence, it continues to catalyse its members, which are never consumed by future reactions. No matter the chain of events, the compartment will ultimately display the fully developed white RAF. If the presence or absence of RAF subsets are taken to be evolutionary traits, these traits can never be selected against in evolutionary competition dynamics. With the basic one-compartment dynamics explained in detail, we can now move on to populations of compartments.

.15.2 Dynamics of a population of compartments

Using the same parameter values (including the reaction rate constants), we next ran the same simulation, but with 100 compartments randomly spread out in the grid. As the one-compartment simulation already suggests, different compartments in the population go through different sequences of colour changes at different times, giving rise to a population of mixed compartment "types" (i.e., some with only the yellow closed RAF, some with the red closed RAF, some with the blue, etc.). Figure 31 shows four snapshots from one such simulation.

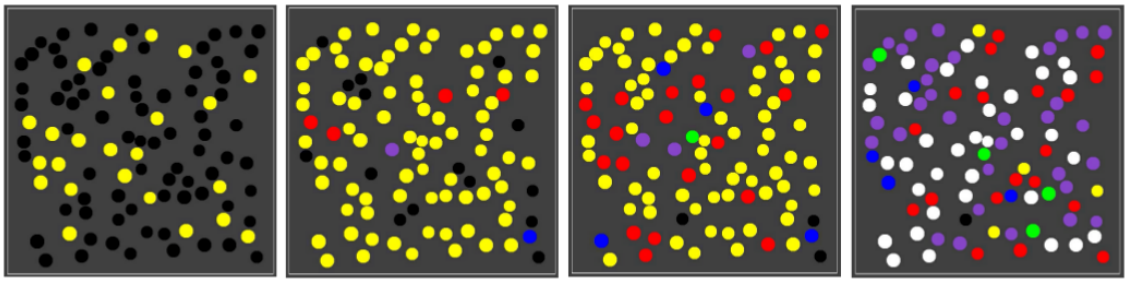


Figure 31: **A population of compartments.** Four snapshots over time from a simulation with 100 compartments.

As the figure shows, early on most compartments are still black (i.e., no chemistry is going on yet, but some are have already gathered enough food molecules to have the yellow closed RAF in existence. A little later most compartments are in the yellow state, and there are also already a few red, blue, or purple compartments. This compartment diversity then increases, until finally most compartments have turned purple or white, although there are also still several other colours, including a few yellow ones. A movie of this simulation (for 150 time units) is provided at the following web page: http://ico2s.org/data/extras/compartments/, where a copy of the used parameter file can also be viewed or downloaded.

Szathmáry, Kauffman, and colleagues have shown that such variability is exactly one of the main conditions for autocatalytic sets to be evolvable [218]. Having different combinations of autocatalytic subsets (i.e., different closed RAFs) existing inside different compartments can give rise to competition for resources between compartments, and new autocatalytic subsets coming into existence in some compartments, due to rare spontaneous reaction events, give rise to variation.

These mutations are immediately evident in the movie as the changes in colour of the compartments. Competition between compartments is less evident, but is also present. Note that even at the end of the simulation there are still a few yellow and even one black compartment. Because other compartments already have larger closed RAFs existing inside them, those other compartments are using up food molecules at a relatively high rate. Given that molecules are introduced at a constant rate and then diffuse through the grid, they tend to diffuse mostly

towards those compartments that use them up at high rates, simply due to the resulting concentration differences. Therefore, food resources are diverted away from compartments that do not have much chemistry going on yet (e.g., they may only have the yellow closed RAF present), and are therefore "starved", becoming even less likely to ever go beyond the black or yellow state. A similar type of competition was shown to exist in principle, in a one-compartment scenario with two competing RAF subsets, in earlier simulation studies [90].

Note, though, that in the reaction network used in this simulation, only ligation reactions are included, but not cleavage reactions. Therefore, mutations can only happen in one direction: only new autocatalytic subsets can come into existence, giving rise to a larger closed RAF existing inside a compartment. As such, the lack of evolvability observed in the last section is not automatically remedied by investigating populations of compartments. What would make the dynamics more interesting is a mechanism for mutations where an autocatalytic subset is lost.

.15.3 The influence of a toxic element

Suppose that molecule type 00100, produced by the red RAF subset, can spontaneously turn into a "toxic" element that catalyses the destruction of molecule type 11100, which is produced by and acts as a catalyst of the blue RAF subset. In other words, once the red RAF subset is present, it can suppress the existence of the blue RAF subset. This way, it is possible to have "mutations" where an autocatalytic subset is lost. Such an event is illustrated in Figure 32.

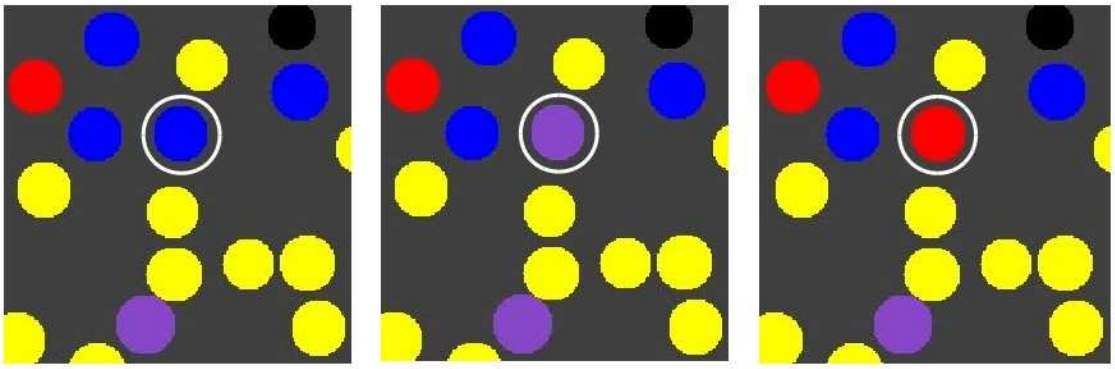


Figure 32: **The influence of a toxic element.** The production of a toxic element by the red RAF subset can cause the blue RAF subset to be lost again, making a compartment change from blue to purple to red (indicated by the white circle).

The three images in Figure 32 show snapshots of the same area in the grid, but at different time steps, from a simulation that includes the toxic element, and where the reaction making up the green RAF subset is left out (for illustrative purposes). All other parameter values are kept the same as in the previous case.

Note that the compartment indicated by the white circle changes from blue to purple to red. In the previous simulation this would not have been possible, since concentrations of long polymers cannot decrease. However, what happened in this simulation is that the given compartment acquired the blue RAF subset first (becoming blue), then the red one (becoming purple), but then the toxic element produced by the red subset destroyed the blue subset, causing the compartment to change to red. On the other hand, the red subset can also (temporarily) destroy itself. If all the molecules of type 00100 produced by the red subset turn into toxic elements, the red subset itself does not exist anymore either, until new 00100 molecules are produced. So, a compartment could also oscillate between yellow and red. Both of these situations happen in various locations and at various times in the simulation, a movie of which (for 100 time units) is available on the mentioned web page.

Another way to see the influence of the toxic element, compared to the base case from the previous subsection, is to look at the number of compartments of each type (i.e., colour) over time. Figure 35(left) shows such a comparison for

two representative simulations for each case. As this plot shows, in the toxic element case (dashed lines), the number of purple compartments is clearly suppressed, as it is now more difficult to have the red and blue RAF subsets existing simultaneously.

Taken together, the simulations presented so far demonstrate that populations of RAF sets allow for competition and selection dynamics and ultimately for (limited) evolution, if the chemistry allows for molecules of RAF sets to be consumed. We next showcase some other dynamical features that can occur in spatially embedded populations of compartmentalised RAF sets.

.15.4 The influence of a permeable inducer

So far, the various compartments do not interact with each other or the environment, other than taking up food molecules. In the next simulation, we also include the secretion of an element produced by the compartments, in particular one that can induce other compartments to acquire an autocatalytic set (if they do not already have one).

To illustrate the effect in its simplest form, we use a different chemical reaction network, shown in Figure 33, than in the previous simulations. This network also forms a RAF set that could exist in the binary polymer model. For this simulation, only molecule types 0 and 11 are food molecules. Note that this RAF set also needs at least one spontaneous reaction to come into existence. As before, a compartment turns yellow as soon as at least two molecules of type 110 are present.

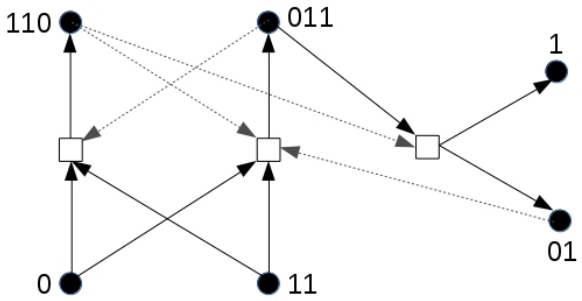


Figure 33: **A reaction network with an inducer.** The reaction network used to show the influence of an inducer (molecule type 01). As before, dashed arrows indicate catalysis.

However, note that one of the products of this RAF set, molecule type 01 (not part of the food set), acts as an additional catalyst for one of the two reactions that initially need to happen spontaneously. We assume that this molecule can cross the compartment boundary since it has the same length as one of the food molecules, and then diffuse (at low rate) through the grid. What can then happen is that one compartment that already has the RAF set present secretes one or more molecules of type 01 into the environment, which then slowly diffuse through the grid and enter another compartment. If this other compartment does not have the RAF set existing yet, but it has acquired enough food molecules (0 and 11), the inducer molecule (01) can catalyse the required reaction for the RAF set to come into existence, rather than having to wait for an uncatalysed reaction.

In other words, molecule type 01 can act as an "inducer" for RAF sets to come into existence in nearby compartments. This situation is shown in Figure 34, where the blue spheres indicate the concentration of molecule type 01 (outside of compartments) in each grid location in the environment.

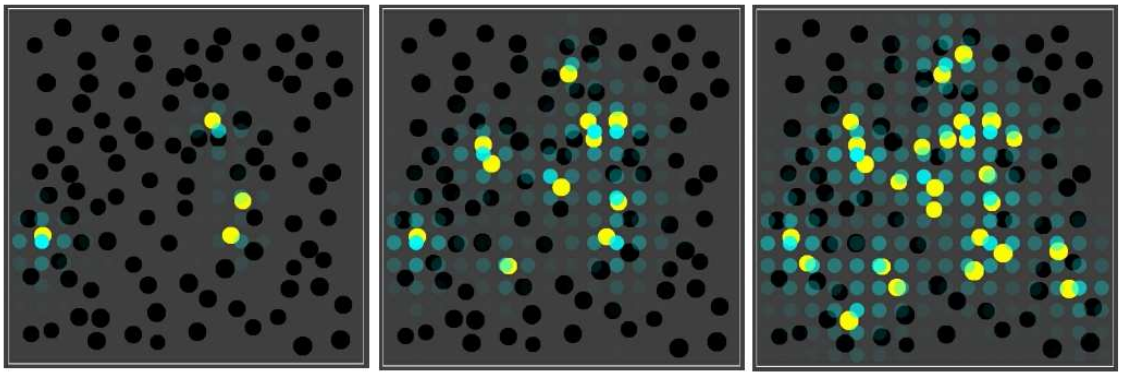


Figure 34: **The influence of a permeable inducer.** Three snapshots over time from a simulation where the RAF set produces an permeable inducer that can diffuse through the lattice. The blue spheres indicate the concentration of this inducer in the different grid locations.

In the corresponding movie (available on the same web page again, including the parameter file), it is clear that black compartments tend to turn yellow preferentially in the vicinity of other compartments that are already yellow, where the highest concentrations of the inducer are found. The end result is that the yellow compartments are clustered, rather than distributed homogeneously throughtout the space.

To show that it is indeed the inducer that causes this phenomenon, we first performed ten simulations (120 time units each) where the inducer (molecule type 01) is not allowed to cross the compartment membrane. In this case there are (on average) 16.3 yellow compartments (out of 100) at the end of the simulation. Then we performed another ten simulations (also 120 time units each) where the inducer is allowed to cross the membrane (as in Figure 34 and the corresponding movie). In the latter case there are (on average) 26 yellow compartments at the end of the simulation. So, there are significantly more yellow compartments due to the inducer (p-value = 0.0003). Figure 35(right) shows a comparison of the number of yellow compartments over time between a simulation with and a simulation without the permeable inducer.

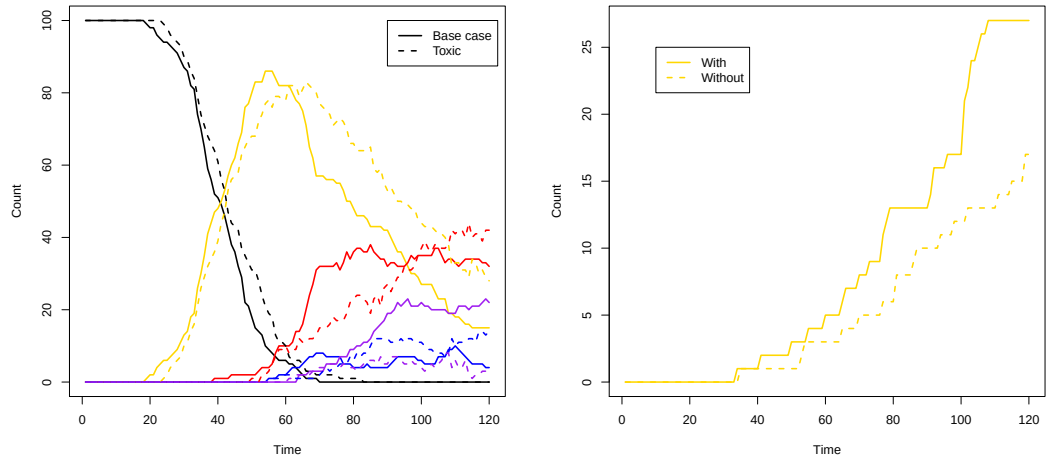


Figure 35: **Compartment counts.** Left: A comparison of compartment type counts between the base case and the influence of a toxic element. Right: A comparison of yellow compartment counts with or without the inducer.

.16 Discussion

We have presented results of computational simulations of autocatalytic sets emerging in compartments. As far as we know, these simulations are the first demonstration of such dynamics explicitly combining (1) collectively autocatalytic sets in (2) *populations* of compartments in (3) a *spatial* environment. This provides a significant step forward towards modeling the emergence and evolution of autocatalytic sets in simple protocells.

Our simulations show that the main requirements for autocatalytic sets to be evolvable are met when encapsulating them into compartment populations: the existence of different combinations of autocatalytic subsets (i.e., closed RAFs) in a population of compartments, giving rise to different "cell types" and competition between them. Mutations in such cell types are caused by occasional new autocatalytic subsets coming into existence due to rare spontaneous reaction events, or the loss of autocatalytic subsets due to, e.g., one subset producing a toxic substance for another subset. This requirement had already been shown to be satisfied, in principle, in earlier studies [90, 218], but here it is shown for the

first time in a spatially explicit population setting.

We have also demonstrated that populations of encapsulated autocatalytic sets can give rise to ecological dynamics that act somewhat orthogonal to evolutionary competition and selection dynamics [228, 229]. As an example of such phenomena, we have shown how permeable inducers produced in one compartment can trigger the appearance of autocatalytic sets in neighboring compartments. We regard this as an example of population dynamics, and concur that similar ecological relationships (e.g. mutualism, parasitism, etc.) should also be observable in spatially coupled populations of RAF sets.

Note, though, that the current simulations cannot yet be considered to represent true protocells. In particular, there is no coupling between the internal (autocatalytic set) dynamics and the (fixed) compartment boundary [152]. The main focus in the current study has been on the population and spatial aspects. This could, for example, model chemistry in the porous structure of hydrothermal vents [150]. However, Simbiotics can also simulate movement, growth, and division of compartments. Coupling internal dynamics with compartment growth and division is one of the hallmarks of protocell research [23, 31, 152, 191], and will be a focus of future work.

Importantly, once implemented, compartment division and the generation of offspring will allow us to study inheritance (another requirement for evolvability [90, 218]). When a compartment divides, it will distribute its biomass among its two offspring cells. Assuming sufficiently high molecular counts, both offspring cells are likely to contain all the necessary catalysts to continue the chemical dynamics of autocatalytic subsets that were already present in the parent, without having to wait for any spontaneous reactions again. However, especially at low concentrations, it might happen that one or more essential catalysts are missing in one of the offspring cells, due to stochastic fluctuations at division. In that case, one or more of the autocatalytic subsets that were present in the parent cell can be lost, which would provide another way for mutations to happen, as was already suggested [218].

Finally, although the results presented here are computational simulations using an abstract chemical reaction network, there are direct links to experimental systems. For example, recently the emergence and dynamics of autocatalytic

sets of RNA molecules have been studied in microfluidics [8], which provides an experimental simulation of compartments. These RNA autocatalytic sets were created in the lab [216], and have been studied in more detail using the formal RAF framework [89]. Thus, there is a direct and natural connection between our simulations and these microdroplet experiments, which we hope to explore in future work.

Autocatalytic sets have been shown to have a high probability of existence, also for moderate and chemically plausible levels of catalysis [88, 156]. Furthermore, several experimental autocatalytic sets have been constructed in the lab, either with nucleotide sequences [120, 196, 216] or with peptides [12]. Finally, they have been shown, in principle, to be evolvable [90, 218]. Here, we have taken a first step towards a more realistic demonstration of this by simulating the emergence and dynamics of autocatalytic (sub)sets in populations of compartments in a spatially explicit environment.

Clearly, this has consequences for how we might think about the origin of life. If autocatalytic sets have a high chance of emerging spontaneously in simple compartments (e.g., lipid membranes), and can grow and evolve to become more complex, this could provide a plausible way for life to have arisen. Further simulation studies along these lines, also in combination with experimental studies as indicated, seem to be a promising avenue to shine light on this pathway to life.

Parameter	**Value**
Spontaneous rates	
c_{100},c_{110}	0.02
c_{11100},c_{111},c_{010}	0.003
c_{1010},c_{0111}	0.005
c_{00100}	0.001
Catalysed rates	
c_{11100},c_{111},c_{010}	0.005
c_{1010}	0.03
c_{00100}	0.02
c_{0111}	0.01

Table 5: Model parameters for Chapter 9 Figure 9.5-9.7

Parameter	Value
Catalysed rates	
c_{01111}	0.005

Table 6: Modified model parameters for Chapter 9 Figure 9.7

Parameter	Value
Spontaneous rates	
c_{110},c_{011}	0.0001
Catalysed rates	
c_{110},c_{011}	0.05
$c_{[011+110 \rightarrow 01+1+110]}$	0.05
$c_{[0+11+01 \rightarrow 011+01]}$	0.5

Table 7: Model parameters for Chapter 9 Figure 9.8

Parameter	Value
World size	20 x 20
Compartments	100
Compartment radius	0.5
Voxel size	2.5 x 2.5
Numerical time step	0.01

Table 8: Model parameters for Chapter 9 Figures 9.4-9.10

Parameter	Value
Diffusion coefficients	
D_0,D_1,D_{00},D_{01},D_{10},D_{11}	20
D_{010},D_{100},D_{110},D_{111},D_{1010},D_{00100},D_{01111},D_{11100}	10
Decay rate	
K_0,K_1,K_{00},K_{01},K_{10},K_{11}	0.013
K_{010},K_{100},K_{110},K_{111},K_{1010},K_{00100},K_{01111},K_{11100}	0

Table 9: Model parameters for Chapter 9 Figure 9.6

Bibliography

[1] Front matter. In H.W. Doelle, editor, *Bacterial Metabolism*, page iii. Academic Press, 1969. ISBN 978-1-4832-3135-8. doi: https://doi.org/10.1016/B978-1-4832-3135-8.50001-1. URL http://www.sciencedirect.com/science/article/pii/B9781483231358500011. 22, 23

[2] An innovative solution to help with plaque management. *British Dental Journal*, 218(6):364–364, mar 2015. doi: 10.1038/sj.bdj.2015.234. URL https://doi.org/10.1038/sj.bdj.2015.234. 145

[3] Sameera Abar, Georgios K. Theodoropoulos, Pierre Lemarinier, and Gregory M.P. O'Hare. Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33, may 2017. doi: 10.1016/j.cosrev.2017.03.001. URL https://doi.org/10.1016/j.cosrev.2017.03.001. 33

[4] Michael Abrahamson, Zbigniew Lewandowski, Gill Geesey, Gudmund Skjak-Braek, Wenche Strand, and Bjorn E. Christensen. Development of an artificial biofilm to study the effects of a single microcolony on mass transport. *Journal of Microbiological Methods*, 26(1-2):161–169, jul 1996. doi: 10.1016/0167-7012(96)00908-6. URL https://doi.org/10.1016/0167-7012(96)00908-6. 156

[5] Uri Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits (Chapman & Hall/CRC Mathematical and Computational Biology)*. Chapman and Hall/CRC, 2006. ISBN 1584886420. 6

[6] Uri Alon. Network motifs: theory and experimental approaches. *Nature*

Reviews Genetics, 8(6):450–461, jun 2007. doi: 10.1038/nrg2102. URL https://doi.org/10.1038/nrg2102. 6

[7] Rudolf Amann and Ramon Rossello-Mora. After all, only millions? *mBio*, 7(4), jul 2016. doi: 10.1128/mbio.00999-16. URL https://doi.org/10.1128/mbio.00999-16. 22

[8] S. Ameta, S. Arsene, P. Nghe, N. Lehman, and A. D. Griffiths. Autocatalytic sets of RNA replicators in origin of life. In *XVIIIth International Conference on Origin of Life*, 2017. 319

[9] J. S.J. Anderson. The 3' to 5' degradation of yeast mRNAs is a general mechanism for mRNA turnover that requires the SKI2 DEVH box protein and 3' to 5' exonucleases of the exosome complex. *The EMBO Journal*, 17(5):1497–1506, mar 1998. doi: 10.1093/emboj/17.5.1497. URL https://doi.org/10.1093/emboj/17.5.1497. 23

[10] Ioan I. Ardelean and Matteo Cavaliere. *Natural Computing*, 2(2):173–197, 2003. doi: 10.1023/a:1024943605864. URL https://doi.org/10.1023/a:1024943605864. 33

[11] Nina Bjork Arnfinnsdottir, Vegar Ottesen, Rahmi Lale, and Marit Sletmoen. The design of simple bacterial microarrays: Development towards immobilizing single living bacteria on predefined micro-sized spots on patterned surfaces. *PLOS ONE*, 10(6):e0128162, jun 2015. doi: 10.1371/journal.pone.0128162. URL https://doi.org/10.1371/journal.pone.0128162. 178

[12] G. Ashkenasy, R. Jegasia, M. Yadav, and M. R. Ghadiri. Design of a directed molecular network. *PNAS*, 101(30):10872–10877, 2004. 319

[13] M. R. Atkinson, M. A. Savageau, J. T. Myers, and A. J. Ninfa. Development of genetic circuitry exhibiting toggle switch or oscillatory behavior in Escherichia coli. *Cell*, 113(5):597–607, May 2003. 7

[14] C.R. Back, S.K. Douglas, J.E. Emerson, A.H. Nobbs, and H.F. Jenkinson. Streptococcus gordoniidl1 adhesin sspb v-region mediates coaggregation via

receptor polysaccharide ofactinomyces orist14V. *Molecular Oral Microbiology*, 30(5):411–424, Jun 2015. ISSN 2041-1006. doi: 10.1111/omi.12106. URL http://dx.doi.org/10.1111/omi.12106. 145

[15] Samuel Baron. *Medical Microbiology*. Univ of Texas Medical Branch, 1996. ISBN 0963117211. 26

[16] Timothy Barth and Mario Ohlberger. Finite volume methods: Foundation and analysis. *Encyclopedia of Computational Mechanics*, Nov 2004. doi: 10.1002/0470091355.ecm010. URL http://dx.doi.org/10.1002/0470091355.ecm010. 51

[17] S. Basu, R. Mehreja, S. Thiberge, M.-T. Chen, and R. Weiss. Spatiotemporal control of gene expression with pulse-generating networks. *Proceedings of the National Academy of Sciences*, 101(17):6355–6360, apr 2004. doi: 10.1073/pnas.0307571101. URL https://doi.org/10.1073/pnas.0307571101. 179

[18] Subhayu Basu, David Karig, and Ron Weiss. Engineering signal processing in cells: Towards molecular concentration band detection. *Natural Computing*, 2(4):463–478, 2003. ISSN 1567-7818. doi: 10.1023/b:naco.0000006774.27778.f0. URL http://dx.doi.org/10.1023/B:NACO.0000006774.27778.f0. 179

[19] Subhayu Basu, Yoram Gerchman, Cynthia H. Collins, Frances H. Arnold, and Ron Weiss. A synthetic multicellular system for programmed pattern formation. *Nature*, 434(7037):1130–1134, apr 2005. doi: 10.1038/nature03461. URL https://doi.org/10.1038/nature03461. 179

[20] Yasmine Belkaid and Timothy W. Hand. Role of the microbiota in immunity and inflammation. *Cell*, 157(1):121–141, mar 2014. doi: 10.1016/j.cell.2014.03.011. URL https://doi.org/10.1016/j.cell.2014.03.011. 6

[21] Eshel Ben-Jacob, Ofer Schochet, Adam Tenenbaum, Inon Cohen, Andras Czirok, and Tamas Vicsek. Generic modelling of cooperative growth patterns in bacterial colonies. *Nature*, 368(6466):46–49, mar 1994. doi: 10.1038/368046a0. URL https://doi.org/10.1038/368046a0. 33

[22] Frank T. Bergmann, Stefan Hoops, Brian Klahn, Ursula Kummer, Pedro Mendes, Jurgen Pahle, and Sven Sahle. COPASI and its applications in biotechnology. *Journal of Biotechnology*, 261:215–220, nov 2017. doi: 10.1016/j.jbiotec.2017.06.1200. URL https://doi.org/10.1016/j.jbiotec.2017.06.1200. 32

[23] E. Bigan, L. Pauleve, J. Steyaert, and S. Douady. Necessary and sufficient conditions for protocell growth. *Journal of Mathematical Biology*, 73:1627–1664, 2016. 298, 318

[24] J. Blakes, J. Twycross, F. J. Romero-Campero, and N. Krasnogor. The infobiotics workbench: an integrated in silico modelling platform for systems and synthetic biology. *Bioinformatics*, 27(23):3323–3324, oct 2011. doi: 10.1093/bioinformatics/btr571. 33, 34, 203

[25] Ginger Booth. Gecko: A continuous 2d world for ecological modeling. *Artificial Life*, 3(3):147–163, jul 1997. doi: 10.1162/artl.1997.3.3.147. URL https://doi.org/10.1162/artl.1997.3.3.147. 5

[26] Benedict Borer, Robin Tecon, and Dani Or. Spatial organization of bacterial populations in response to oxygen and carbon counter-gradients in pore networks. *Nature Communications*, 9(1), feb 2018. doi: 10.1038/s41467-018-03187-y. URL https://doi.org/10.1038/s41467-018-03187-y. 28

[27] Kjell Bovre and Leif Oddvar Froholm. Variation of colony morphology reflecting fimbriation in moraxella bovis and two reference strains of m. nonliquefaciens. *Acta Pathologica Microbiologica Scandinavica Section B Microbiology and Immunology*, 80B(5):629–640, aug 2009. doi: 10.1111/j.1699-0463.1972.tb00189.x. URL https://doi.org/10.1111/j.1699-0463.1972.tb00189.x. 6

[28] H.W. Brooks, D.G. White, A.J. Wagstaff, and A.R. Michell. Evaluation of a glutamine-containing oral rehydrationsolution for the treatment of calf diarrhoea using an escherichia coli model. *The Veterinary Journal*, 153(2):

163–169, Mar 1997. ISSN 1090-0233. doi: 10.1016/s1090-0233(97)80036-6. URL http://dx.doi.org/10.1016/S1090-0233(97)80036-6. 156

[29] Hendrik J. Busscher and Anton H. Weerkamp. Specific and non-specific interactions in bacterial adhesion to solid substrata. *FEMS Microbiology Letters*, 46(2):165–173, jun 1987. doi: 10.1111/j.1574-6968.1987.tb02457.x. URL https://doi.org/10.1111/j.1574-6968.1987.tb02457.x. 145

[30] Jiguo Cao, Xin Qi, and Hongyu Zhao. Modeling gene regulation networks using ordinary differential equations. In *Next Generation Microarray Bioinformatics*, pages 185–197. Humana Press, nov 2011. doi: 10.1007/978-1-61779-400-1_12. URL https://doi.org/10.1007/978-1-61779-400-1_12. 31

[31] T. Carletti, R. Serra, I. Poli, M. Villani, and A. Filisetti. Sufficient conditions for emergent synchronization in protocell models. *Journal of Theoretical Biology*, 254:741–751, 2008. 298, 318

[32] Deepak Chandran, Frank T. Bergmann, and Herbert M. Sauro. Computer-aided design of biological circuits using tinkercell. *Bioengineered Bugs*, 1 (4):276–283, jul 2010. doi: 10.4161/bbug.1.4.12506. URL https://doi.org/10.4161/bbug.1.4.12506. 138

[33] Deepak Chandran, Frank T. Bergmann, Herbert M. Sauro, and Douglas Densmore. Computer-aided design for synthetic biology. In *Design and Analysis of Biomolecular Circuits*, pages 203–224. Springer New York, 2011. doi: 10.1007/978-1-4419-6766-4_10. URL https://doi.org/10.1007/978-1-4419-6766-4_10. 138

[34] Meng Chen, Qingsong Yu, and Hongmin Sun. Novel strategies for the prevention and treatment of biofilm related infections. *International Journal of Molecular Sciences*, 14(9):18488–18501, sep 2013. doi: 10.3390/ijms140918488. URL https://doi.org/10.3390/ijms140918488. 156

[35] An-Chun Chien, NorbertÂăS. Hill, and PetraÂăAnne Levin. Cell size control in bacteria. *Current Biology*, 22(9):R340–R349, May 2012. ISSN 0960-

9822. doi: 10.1016/j.cub.2012.02.032. URL http://dx.doi.org/10.1016/j.cub.2012.02.032. 55

[36] B. Chopard, J. Borgdorff, and A. G. Hoekstra. A framework for multi-scale modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2021):20130378–20130378, jun 2014. doi: 10.1098/rsta.2013.0378. URL https://doi.org/10.1098/rsta.2013.0378. 5

[37] Gabriel Ciobanu. Theory and applications of p systems. *International Journal of Computer Mathematics*, 83(7):525–527, jul 2006. doi: 10.1080/00207160601065272. URL https://doi.org/10.1080/00207160601065272. 33

[38] S. Cortassa, J. C. Aon, M. A. Aon, and J. F. Spencer. Dynamics of metabolism and its interactions with gene expression during sporulation in saccharomyces cerevisiae. *Adv. Microb. Physiol.*, 43:75–115, 2000. 23

[39] Robert Costanza, Lisa Wainger, Carl Folke, and Karl-Goran Maler. *Modeling Complex Ecological Economic Systems: Toward an Evolutionary, Dynamic Understanding of People and Nature*, pages 148–163. Springer New York, New York, NY, 1996. ISBN 978-1-4612-4018-1. doi: 10.1007/978-1-4612-4018-1_15. URL https://doi.org/10.1007/978-1-4612-4018-1_15. 5

[40] J. W. Costerton. overview of microbial biofilms. *J. Ind. Microbiol.*, 15(3): 137–140, Sep 1995. 6, 28, 29

[41] JonathannbspR Karr JayoditanbspC Sanghvi DereknbspN Macklin MiriamnbspV Gutschow JarednbspM Jacobs Benjamin Bolival Jr Nacyra Assad-Garcia JohnnbspI Glass MarkusnbspW Covert, Jayodita C Sanghvi, Derek N Macklin, Miriam V Gutschow, Jared M Jacobs, Benjamin Bolival, Jr., Nacyra Assad-Garcia, John I Glass, and Markus W Covert. A whole-cell computational model predicts phenotype from genotype. *Cell*, 150(2): 389–401, July 2012. 6

[42] Dennis G. Cvitkovitch, Yung-Hua Li, and Richard P. Ellen. Quorum sensing and biofilm formation in streptococcal infections. *Journal of Clinical Investigation*, 112(11):1626–1632, dec 2003. doi: 10.1172/jci200320430. URL https://doi.org/10.1172/jci200320430. 27

[43] Agnieszka Cydzik-Kwiatkowska and Magdalena Zielinska. Bacterial communities in full-scale wastewater treatment systems. *World Journal of Microbiology and Biotechnology*, 32(4), mar 2016. doi: 10.1007/s11274-016-2012-9. URL https://doi.org/10.1007/s11274-016-2012-9. 6

[44] Bruce Damer and David Deamer. Coupled phases and combinatorial selection in fluctuating hydrothermal pools: A scenario to guide experimental approaches to the origin of cellular life. *Life*, 5(1):872–887, mar 2015. doi: 10.3390/life5010872. URL https://doi.org/10.3390/life5010872. 298

[45] Tal Danino, Octavio Mondragon-Palomino, Lev Tsimring, and Jeff Hasty. A synchronized quorum of genetic clocks. *Nature*, 463(7279):326–330, jan 2010. doi: 10.1038/nature08753. URL https://doi.org/10.1038/nature08753. 7

[46] Oladipo Dare-Abel, Obioha Uwakonye, and Akunnaya Opoko. THE EFFECT OF CAD ON ARCHITECTURE STUDENTS' CREATIVITY AND ENTHUSIASM. In *INTED2016 Proceedings*. IATED, mar 2016. doi: 10.21125/inted.2016.0984. URL https://doi.org/10.21125/inted.2016.0984. 35

[47] Maria I. Davidich and Stefan Bornholdt. Boolean network model predicts cell cycle sequence of fission yeast. *PLoS ONE*, 3(2):e1672, feb 2008. doi: 10.1371/journal.pone.0001672. URL https://doi.org/10.1371/journal.pone.0001672. 30

[48] Hidde de Jong. Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology*, 9(1):67–103, Jan 2002. ISSN 1557-8666. doi: 10.1089/10665270252833208. URL http://dx.doi.org/10.1089/10665270252833208. 30

[49] Veronica DeGuzman, Wenonah Vercoutere, Hossein Shenasa, and David Deamer. Generation of oligonucleotides under hydrothermal conditions by non-enzymatic polymerization. *Journal of Molecular Evolution*, 78(5):251–262, may 2014. doi: 10.1007/s00239-014-9623-2. URL https://doi.org/10.1007/s00239-014-9623-2. 298

[50] W. D. Donachie, K. J. Begg, and M. Vicente. Cell length, cell growth and cell division. *Nature*, 264(5584):328–333, Nov 1976. doi: 10.1038/264328a0. URL http://dx.doi.org/10.1038/264328a0. 57

[51] Robin Donaldson and David Gilbert. A model checking approach to the parameter estimation of biochemical pathways. In *Computational Methods in Systems Biology*, pages 269–287. Springer Berlin Heidelberg, 2008. doi: 10.1007/978-3-540-88562-7_20. URL https://doi.org/10.1007/978-3-540-88562-7_20. 32

[52] Rodney M. Donlan. Biofilm formation: A clinically relevant microbiological process. *Clinical Infectious Diseases*, 33(8):1387–1392, oct 2001. doi: 10.1086/322972. URL https://doi.org/10.1086/322972. 156

[53] Rodney M. Donlan. Biofilms: Microbial life on surfaces. *Emerging Infectious Diseases*, 8(9):881–890, Sep 2002. ISSN 1080-6059. doi: 10.3201/eid0809.020063. URL http://dx.doi.org/10.3201/eid0809.020063. 4, 27

[54] Diana M. Downs. Understanding microbial metabolism. *Annual Review of Microbiology*, 60(1):533–559, 2006. doi: 10.1146/annurev.micro.60.080805.142308. URL https://doi.org/10.1146/annurev.micro.60.080805.142308. PMID: 16771650. 23

[55] D. Dykhuizen. Species Numbers in Bacteria. *Proc. Calif. Acad. Sci.*, 56(6 Suppl 1):62–71, Jun 2005. 22

[56] Michael B. Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, jan 2000. doi: 10.1038/35002125. URL https://doi.org/10.1038/35002125. 7

[57] A. K. Epstein, T.-S. Wong, R. A. Belisle, E. M. Boggs, and J. Aizenberg. Liquid-infused structured surfaces with exceptional anti-biofouling performance. *Proceedings of the National Academy of Sciences*, 109(33): 13182–13187, jul 2012. doi: 10.1073/pnas.1201973109. URL https://doi.org/10.1073/pnas.1201973109. 156

[58] Arthur Fabel. The dynamics of the self-organizing universe. *CrossCurrents*, 37(2/3):168–177, 1987. ISSN 00111953, 19393881. URL http://www.jstor.org/stable/24459044. 3

[59] J.Doyne Farmer, Stuart A Kauffman, and Norman H Packard. Autocatalytic replication of polymers. *Physica D: Nonlinear Phenomena*, 22 (1-3):50–67, oct 1986. doi: 10.1016/0167-2789(86)90233-2. URL https://doi.org/10.1016/0167-2789(86)90233-2. 300

[60] F. D. C. Farrell, O. Hallatschek, D. Marenduzzo, and B. Waclaw. Mechanically driven growth of quasi-two-dimensional microbial colonies. *Physical Review Letters*, 111(16), Oct 2013. ISSN 1079-7114. doi: 10.1103/physrevlett.111.168101. URL http://dx.doi.org/10.1103/PhysRevLett.111.168101. 48

[61] H. Fellermann. Wet artificial life: The construction of artificial living systems. In *Principles of Evolution*, pages 261–280. Springer Berlin Heidelberg, 2011. 298

[62] Rosalind A. Le Feuvre and Nigel S. Scrutton. A living foundry for synthetic biological materials: A synthetic biology roadmap to new advanced materials. *Synthetic and Systems Biotechnology*, 3(2):105–112, jun 2018. doi: 10.1016/j.synbio.2018.04.002. URL https://doi.org/10.1016/j.synbio.2018.04.002. 166

[63] Hans-Curt Flemming and Jost Wingender. The biofilm matrix. *Nature Reviews Microbiology*, 8(9):623–633, aug 2010. doi: 10.1038/nrmicro2415. URL https://doi.org/10.1038/nrmicro2415. 28

[64] Hans-Curt Flemming, Jost Wingender, Ulrich Szewzyk, Peter Steinberg, Scott A. Rice, and Staffan Kjelleberg. Biofilms: an emergent form of

bacterial life. *Nature Reviews Microbiology*, 14(9):563–575, sep 2016. doi: 10.1038/nrmicro.2016.94. URL https://doi.org/10.1038/nrmicro.2016.94. 5, 6

[65] A. E. Friedland, T. K. Lu, X. Wang, D. Shi, G. Church, and J. J. Collins. Synthetic gene networks that count. *Science*, 324(5931):1199–1202, may 2009. doi: 10.1126/science.1172005. URL https://doi.org/10.1126/science.1172005. 7

[66] Eileen Fung, Wilson W. Wong, Jason K. Suen, Thomas Bulter, Sun gu Lee, and James C. Liao. A synthetic gene-metabolic oscillator. *Nature*, 435 (7038):118–122, may 2005. doi: 10.1038/nature03508. URL https://doi.org/10.1038/nature03508. 7

[67] Timothy S. Gardner, Charles R. Cantor, and James J. Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767):339–342, jan 2000. doi: 10.1038/35002131. URL https://doi.org/10.1038/35002131. 7

[68] Trevor Roger Garrett, Manmohan Bhakoo, and Zhibing Zhang. Bacterial adhesion and biofilms on surfaces. *Progress in Natural Science*, 18(9):1049–1056, Sep 2008. ISSN 1002-0071. doi: 10.1016/j.pnsc.2008.04.001. URL http://dx.doi.org/10.1016/j.pnsc.2008.04.001. 29

[69] Yves Gendrault, Morgan Madec, Christophe Lallement, Francois Pecheux, and Jacques Haiech. Computer-aided design in synthetic biology. In *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies - ISABEL '11*. ACM Press, 2011. doi: 10.1145/2093698.2093869. URL https://doi.org/10.1145/2093698.2093869. 138

[70] Alf Gerisch, Raimondo Penta, and Jens Lang, editors. *Multiscale Models in Mechano and Tumor Biology*. Springer International Publishing, 2017. doi: 10.1007/978-3-319-73371-5. URL https://doi.org/10.1007/978-3-319-73371-5. 5

[71] Pushpita Ghosh, Jagannath Mondal, Eshel Ben-Jacob, and Herbert Levine. Mechanically-driven phase separation in a growing bacterial colony. *Proceedings of the National Academy of Sciences*, 112(17):E2166–E2173, Apr 2015. ISSN 1091-6490. doi: 10.1073/pnas.1504948112. URL http://dx.doi.org/10.1073/pnas.1504948112. 48

[72] D. T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22:403–434, 1976. 305

[73] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977. 305

[74] Angel Goni-Moreno and Martyn Amos. *DiSCUS: A Simulation Platform for Conjugation Computing*, chapter DiSCUS: A Simulation Platform for Conjugation Computing, pages 181–191. Springer International Publishing, Cham, 2015. ISBN 978-3-319-21819-9. doi: 10.1007/978-3-319-21819-9_13. URL http://dx.doi.org/10.1007/978-3-319-21819-9_13. 33, 34

[75] Thomas E. Gorochowski, Antoni Matyjaszkiewicz, Thomas Todd, Neeraj Oak, Kira Kowalska, Stephen Reid, Krasimira T. Tsaneva-Atanasova, Nigel J. Savery, Claire S. Grierson, and Mario di Bernardo. Bsim: An agent-based tool for modeling bacterial populations in systems and synthetic biology. *PLoS ONE*, 7(8):e42790, Aug 2012. ISSN 1932-6203. doi: 10.1371/journal.pone.0042790. URL http://dx.doi.org/10.1371/journal.pone.0042790. 34

[76] Anthony J.F. Griffiths, Jeffrey H. Miller, David T. Suzuki, Richard C. Lewontin, and William M. Gelbart. *Introduction to Genetic Analysis*. W. H. Freeman, 2000. ISBN 0716735202. 26

[77] Gary M. Grobman. Complexity theory: A new way to look at organizational change. *Public Administration Quarterly*, 29(3/4):350–382, 2005. ISSN 07349149. URL http://www.jstor.org/stable/41288239. 3

[78] Renan Gross, Itzhak Fouxon, Doron Lancet, and Omer Markovitch. Quasispecies in population of compositional assemblies. *BMC Evolutionary Biology*, 14(1):265, December 2014. ISSN 1471-2148. 299

[79] Martin Gutierrez, Paula Gregorio-Godoy, Guillermo Perez del Pulgar, Luis E. Munoz, Sandra Saez, and Alfonso Rodriguez-Paton. A new improved and extended version of the multicell bacterial simulator gro. *ACS Synthetic Biology*, 6(8):1496–1508, may 2017. doi: 10.1021/acssynbio.7b00003. URL https://doi.org/10.1021/acssynbio.7b00003. 100

[80] B.S. Guttman. Prokaryotes. In *Encyclopedia of Genetics*, page 1549. Elsevier, 2001. doi: 10.1006/rwgn.2001.1031. URL https://doi.org/10.1006/rwgn.2001.1031. 22

[81] Janus A. J. Haagensen, Susse K. Hansen, Bjarke B. Christensen, Sunje J. Pamp, and Soren Molin. Development of spatial distribution patterns by biofilm cells. *Applied and Environmental Microbiology*, 81(18):6120–6128, jun 2015. doi: 10.1128/aem.01614-15. URL https://doi.org/10.1128/aem.01614-15. 28

[82] Babu Halan, Katja Buehler, and Andreas Schmid. Biofilms as living catalysts in continuous chemical syntheses. *Trends in Biotechnology*, 30 (9):453–465, sep 2012. doi: 10.1016/j.tibtech.2012.05.003. URL https://doi.org/10.1016/j.tibtech.2012.05.003. 6

[83] Roland Maximilian Happach and Meike Tilebein. Simulation as research method: Modeling social interactions in management science. In *Collective Agency and Cooperation in Natural and Artificial Systems*, pages 239–259. Springer International Publishing, 2015. doi: 10.1007/978-3-319-15515-9_13. URL https://doi.org/10.1007/978-3-319-15515-9_13. 5

[84] Uta Henssge, Thuy Do, Steven C. Gilbert, Steven Cox, Douglas Clark, Claes Wickstrom, A. J. M. Ligtenberg, David R. Radford, and David Beighton. Application of MLST and pilus gene sequence comparisons to investigate the population structures of actinomyces naeslundii and actinomyces oris.

PLoS ONE, 6(6):e21430, jun 2011. doi: 10.1371/journal.pone.0021430. URL https://doi.org/10.1371/journal.pone.0021430. 154

[85] Malte Hermansson. The dlvo theory in microbial adhesion. *Colloids and Surfaces B: Biointerfaces*, 14(1-4):105–119, Aug 1999. ISSN 0927-7765. doi: 10.1016/s0927-7765(99)00029-6. URL http://dx.doi.org/10.1016/S0927-7765(99)00029-6. 49

[86] Alfons G. Hoekstra, Jiri Kroc, and Peter M.A. Sloot. Introduction to modeling of complex systems using cellular automata. In *Understanding Complex Systems*, pages 1–16. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-12203-3_1. URL https://doi.org/10.1007/978-3-642-12203-3_1. 5

[87] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI–a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067–3074, oct 2006. doi: 10.1093/bioinformatics/btl485. URL https://doi.org/10.1093/bioinformatics/btl485. 136

[88] W. Hordijk and M. Steel. Detecting autocatalytic, self-sustaining sets in chemical reaction systems. *Journal of Theoretical Biology*, 227(4):451–461, 2004. 300, 319

[89] W. Hordijk and M. Steel. A formal model of autocatalytic sets emerging in an RNA replicator system. *Journal of Systems Chemistry*, 4:3, 2013. 300, 319

[90] W. Hordijk and M. Steel. Conditions for evolvability of autocatalytic sets: A formal example and analysis. *Origins of Life and Evolution of Biospheres*, 44(2):111–124, 2014. 298, 299, 300, 312, 317, 318, 319

[91] W. Hordijk and M. Steel. Chasing the tail: The emergence of autocatalytic networks. *BioSystems*, 152:1–10, 2017. 297, 299

[92] W. Hordijk, J. Hein, and M. Steel. Autocatalytic sets and the origin of life. *Entropy*, 12(7):1733–1742, 2010. 297

[93] W. Hordijk, S. A. Kauffman, and M. Steel. Required levels of catalysis for emergence of autocatalytic sets in models of chemical reaction systems. *International Journal of Molecular Sciences*, 12(5):3085–3101, 2011. 300

[94] W. Hordijk, M. Steel, and S. Kauffman. The structure of autocatalytic sets: Evolvability, enablement, and emergence. *Acta Biotheoretica*, 60(4): 379–392, 2012. 300

[95] W. Hordijk, J. I. Smith, and M. Steel. Algorithms for detecting and analysing autocatalytic sets. *Algorithms for Molecular Biology*, 10:15, 2015. 300, 301

[96] W. Hordijk, M. Steel, and P. Dittrich. Autocatalytic sets and chemical organizations: Modeling self-sustaining reaction networks at the origin of life. *New Journal of Phyiscs*, 20:015011, 2018. 301, 302

[97] M. C. Horner-Devine, K. M. Carney, and B. J. M. Bohannan. An ecological perspective on bacterial biodiversity. *Proceedings of the Royal Society B: Biological Sciences*, 271(1535):113–122, jan 2004. doi: 10.1098/rspb.2003.2549. URL https://doi.org/10.1098/rspb.2003.2549. 6

[98] R. Van Houdt and C.W. Michiels. Biofilm formation and the food industry, a focus on the bacterial outer surface. *Journal of Applied Microbiology*, 109(4):1117–1131, aug 2010. doi: 10.1111/j.1365-2672.2010.04756.x. URL https://doi.org/10.1111/j.1365-2672.2010.04756.x. 29

[99] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, , the rest of the SBML Forum:, A. P. Arkin, B. J. Bornstein, D. Bray, and et al. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, Mar 2003. ISSN 1460-2059. doi: 10.1093/bioinformatics/btg015. URL http://dx.doi.org/10.1093/bioinformatics/btg015. 7, 32

[100] David T. Hughes and Vanessa Sperandio. Inter-kingdom signalling: communication between bacteria and their hosts. *Nature Reviews Microbiology*, 6(2):111–120, feb 2008. doi: 10.1038/nrmicro1836. URL https://doi.org/10.1038/nrmicro1836. 27

[101] C. Hung, Y. Zhou, J. S. Pinkner, K. W. Dodson, J. R. Crowley, J. Heuser, M. R. Chapman, M. Hadjifrangiskou, J. P. Henderson, and S. J. Hultgren. Escherichia coli biofilms have an organized and complex extracellular matrix structure. *mBio*, 4(5), sep 2013. doi: 10.1128/mbio.00645-13. URL https://doi.org/10.1128/mbio.00645-13. 28

[102] Sun-Tak Hwang. Fundamentals of membrane transport. *Korean Journal of Chemical Engineering*, 28(1):1–15, dec 2010. doi: 10.1007/s11814-010-0493-z. URL https://doi.org/10.1007/s11814-010-0493-z. 26

[103] Trey Ideker, Timothy Galitski, and Leroy Hood. A new approach to decoding life: Systems biology. 2:343–72, 02 2001. 6

[104] V. V. Isaeva. Self-organization in biological systems. *Biology Bulletin*, 39 (2):110–118, Apr 2012. ISSN 1608-3059. doi: 10.1134/S1062359012020069. URL https://doi.org/10.1134/S1062359012020069. 4, 6

[105] Seunghee S. Jang, Kevin T. Oishi, Robert G. Egbert, and Eric Klavins. Specification and simulation of synthetic multicelled behaviors. *ACS Synthetic Biology*, 1(8):365–374, Aug 2012. ISSN 2161-5063. doi: 10.1021/sb300034m. URL http://dx.doi.org/10.1021/sb300034m. 33, 34

[106] E. Jantsch. *The Self-Organizing Universe: Scientific and Human Implications of the Emerging Paradigm of Evolution (Systems Science and World Order Library. Innovations in Systems Science)*. Pergamon, 1980. ISBN 0080243126. 3

[107] Albertas Janulevicius, Mark C.M. van Loosdrecht, Angelo Simone, and Cristian Picioreanu. Cell flexibility affects the alignment of model myxobacteria. *Biophysical Journal*, 99(10):3129–3138, Nov 2010. ISSN 0006-3495. doi: 10.1016/j.bpj.2010.08.075. URL http://dx.doi.org/10.1016/j.bpj.2010.08.075. 47, 50

[108] Pahala Gedara Jayathilake, Prashant Gupta, Bowen Li, Curtis Madsen, Oluwole Oyebamiji, Rebeca Gonzalez-Cabaleiro, Steve Rushton, Ben Bridgens, David Swailes, Ben Allen, A. Stephen McGough, Paolo Zuliani,

Irina Dana Ofiteru, Darren Wilkinson, Jinju Chen, and Tom Curtis. A mechanistic individual-based model of microbial communities. *PLOS ONE*, 12(8):e0181965, aug 2017. doi: 10.1371/journal.pone.0181965. URL https://doi.org/10.1371/journal.pone.0181965. 33

[109] Oleg Kanakov, Tetyana Laptyeva, Lev Tsimring, and Mikhail Ivanchenko. Spatiotemporal dynamics of distributed synthetic genetic circuits. *Physica D: Nonlinear Phenomena*, 318-319:116–123, apr 2016. doi: 10.1016/j.physd.2015.10.016. URL https://doi.org/10.1016/j.physd.2015.10.016. 179

[110] J.B. Kaplan. Biofilm dispersal: Mechanisms, clinical implications, and potential therapeutic uses. *Journal of Dental Research*, 89(3):205–218, feb 2010. doi: 10.1177/0022034509359403. URL https://doi.org/10.1177/0022034509359403. 29

[111] S. A. Kauffman. Cellular homeostasis, epigenesis and replication in randomly aggregated macromolecular systems. *Journal of Cybernetics*, 1(1):71–96, 1971. 299

[112] S. A. Kauffman. Autocatalytic sets of proteins. *Journal of Theoretical Biology*, 119:1–24, 1986. 299

[113] S. A. Kauffman. *The Origins of Order*. Oxford University Press, 1993. 299

[114] S. A. Kauffman. Approaches to the origin of life on earth. *Life*, 1(1):34–48, 2011. 297, 298

[115] S.A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, mar 1969. doi: 10.1016/0022-5193(69)90015-0. URL https://doi.org/10.1016/0022-5193(69)90015-0. 30

[116] Yiannis N. Kaznessis. SynBioSS-aided design of synthetic biological constructs. In *Methods in Enzymology*, pages 137–152. Elsevier, 2011. doi: 10.1016/b978-0-12-385120-8.00006-1. URL https://doi.org/10.1016/b978-0-12-385120-8.00006-1. 138

[117] Evelyn F. Keller and Lee A. Segel. Model for chemotaxis. *Journal of Theoretical Biology*, 30(2):225–234, Feb 1971. ISSN 0022-5193. doi: 10.1016/0022-5193(71)90050-6. URL http://dx.doi.org/10.1016/0022-5193(71)90050-6. 50

[118] Rohini Keshava, Rohan Mitra, Mohan L. Gope, and Rajalakshmi Gope. Synthetic biology. In *Omics Technologies and Bio-Engineering*, pages 63–93. Elsevier, 2018. doi: 10.1016/b978-0-12-804659-3.00004-x. URL https://doi.org/10.1016/b978-0-12-804659-3.00004-x. 7

[119] Vikas Khanna and Bhavik R. Bakshi. Integrated multiscale modeling of economic-environmental systems for assessing biocomplexity of material use. In *Proceedings of the 2010 IEEE International Symposium on Sustainable Systems and Technology*. IEEE, may 2010. doi: 10.1109/issst.2010.5507702. URL https://doi.org/10.1109/issst.2010.5507702. 5

[120] D-E. Kim and G. F. Joyce. Cross-catalytic replication of an RNA ligase ribozyme. *Chemistry & Biology*, 11:1505–1512, 2004. 319

[121] Hiroaki Kitano. Computational systems biology. *Nature*, 420(6912):206–210, nov 2002. doi: 10.1038/nature01254. URL https://doi.org/10.1038/nature01254. 6

[122] E. Klipp, R. Herwig, A. Kowald, C. Wierling, and H. Lehrach. *Systems Biology in Practice*. Wiley-VCH Verlag GmbH & Co. KGaA, feb 2005. doi: 10.1002/3527603603. URL https://doi.org/10.1002/3527603603. 23, 30, 31

[123] H. Kobayashi, M. Kaern, M. Araki, K. Chung, T. S. Gardner, C. R. Cantor, and J. J. Collins. Programmable cells: Interfacing natural and engineered gene networks. *Proceedings of the National Academy of Sciences*, 101(22):8414–8419, may 2004. doi: 10.1073/pnas.0402940101. URL https://doi.org/10.1073/pnas.0402940101. 7

[124] Savas Konur, Marian Gheorghe, Ciprian Dragomir, Laurentiu Mierla, Florentin Ipate, and Natalio Krasnogor. Qualitative and quantitative analysis of systems and synthetic biology constructs using p systems. *ACS*

Synthetic Biology, 4(1):83–92, aug 2014. doi: 10.1021/sb500134w. URL https://doi.org/10.1021/sb500134w. 33

[125] Savas Konur, Harold Fellermann, Larentiu Marian Mierla, Daven Sanassy, Christophe Ladroue, Sara Kalvala, Marian Gheorghe, and Natalio Krasnogor. *An Integrated In Silico Simulation andÂăBiomatter Compilation Approach toÂăCellular Computation*, pages 655–676. Springer International Publishing, Cham, 2017. ISBN 978-3-319-33921-4. doi: 10.1007/978-3-319-33921-4_25. URL http://dx.doi.org/10.1007/978-3-319-33921-4_25. 33

[126] J.-U. Kreft. Conflicts of interest in biofilms. *Biofilms*, 1(4):265–276, Oct 2004. ISSN 1479-0513. doi: 10.1017/s1479050504001486. URL http://dx.doi.org/10.1017/S1479050504001486. 27, 53

[127] J.-U. Kreft, G. Booth, and J. W. T. Wimpenny. Bacsim, a simulator for individual-based modelling of bacterial colony growth. *Microbiology*, 144(12):3275–3287, Dec 1998. ISSN 1465-2080. doi: 10.1099/00221287-144-12-3275. URL http://dx.doi.org/10.1099/00221287-144-12-3275. 7, 33, 34

[128] Tapan Kumar Singha. Microbial extracellular polymeric substances: Production, isolation and applications. *IOSR Journal of Pharmacy (IOSRPHR)*, 2(2):276–281, Jan 2012. ISSN 2250-3013. doi: 10.9790/3013-0220276281. URL http://dx.doi.org/10.9790/3013-0220276281. 58

[129] J. Stephen Lansing. Complex adaptive systems. *Annual Review of Anthropology*, 32(1):183–204, 2003. doi: 10.1146/annurev.anthro.32.061002.093440. URL https://doi.org/10.1146/annurev.anthro.32.061002.093440. 5

[130] Laurent A. Lardon, Brian V. Merkey, Sonia Martins, Andreas Dotsch, Cristian Picioreanu, Jan-Ulrich Kreft, and Barth F. Smets. idynomics: next-generation individual-based modelling of biofilms. *Environmental Microbiology*, 13(9):2416–2434, Mar 2011. ISSN 1462-2912. doi:

10.1111/j.1462-2920.2011.02414.x. URL http://dx.doi.org/10.1111/j.1462-2920.2011.02414.x. 7, 33, 34, 55, 58

[131] Nicolas Le Novere, Benjamin Bornstein, Alexander Broicher, Melanie Courtot, Marco Donizelli, Harish Dharuri, Lu Li, Herbert Sauro, Maria Schilstra, Bruce Shapiro, Jacky L. Snoep, and Michael Hucka. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research*, 34(Database issue):D689–D691, Jan 2006. 32

[132] Gavin Lear, Julia Bellamy, Bradley S Case, Jack E Lee, and Hannah L Buckley. Fine-scale spatial patterns in bacterial community composition and function within freshwater ponds. *The ISME Journal*, 8(8):1715–1726, feb 2014. doi: 10.1038/ismej.2014.21. URL https://doi.org/10.1038/ismej.2014.21. 28

[133] David Levy. Applications and limitations of complexity theory in organization theory and strategy. 79, 01 2000. 5

[134] K. Lewis. Riddle of biofilm resistance. *Antimicrobial Agents and Chemotherapy*, 45(4):999–1007, apr 2001. doi: 10.1128/aac.45.4.999-1007.2001. URL https://doi.org/10.1128/aac.45.4.999-1007.2001. 156

[135] Yung-Hua Li and Xiaolin Tian. Quorum sensing and bacterial social interactions in biofilms. *Sensors*, 12(3):2519–2538, feb 2012. doi: 10.3390/s120302519. URL https://doi.org/10.3390/s120302519. 27

[136] Vincent Libis, Baudoin Delepine, and Jean-Loup Faulon. Expanding biosensing abilities through computer-aided design of metabolic pathways. *ACS Synthetic Biology*, 5(10):1076–1085, mar 2016. doi: 10.1021/acssynbio.5b00225. URL https://doi.org/10.1021/acssynbio.5b00225. 138

[137] Nabil Litayem, Bhawna Dhupia, and Sadia Rubab. Review of cross-platforms for mobile learning application development. *International Journal of Advanced Computer Science and Applications*, 6(1), 2015. doi: 10.14569/ijacsa.2015.060105. URL https://doi.org/10.14569/ijacsa.2015.060105. 35

[138] Yuan Liu, Pratap C. Naha, Geelsu Hwang, Dongyeop Kim, Yue Huang, Aurea Simon-Soro, Hoi-In Jung, Zhi Ren, Yong Li, Sarah Gubara, Faizan Alawi, Domenick Zero, Anderson T. Hara, David P. Cormode, and Hyun Koo. Topical ferumoxytol nanoparticles disrupt biofilms and prevent tooth decay in vivo via intrinsic catalytic activity. *Nature Communications*, 9(1), jul 2018. doi: 10.1038/s41467-018-05342-x. URL https://doi.org/10.1038/s41467-018-05342-x. 145

[139] C. Y. Loo, D. A. Corliss, and N. Ganeshkumar. Streptococcus gordonii biofilm formation: identification of genes that code for biofilm phenotypes. *J. Bacteriol.*, 182(5):1374–1382, Mar 2000. 154

[140] L.T. Lui, X. Xue, C. Sui, A. Brown, D.I. Pritchard, N. Halliday, K. Winzer, S.M. Howdle, F. Fernandez-Trillo, N. Krasnogor, and C. Alexander. Bacteria clustering by polymers induces the expression of quorum sense controlled phenotypes. *Nature Chemistry*, 5:1058–1065, 2013. doi: 10.1038/nchem.1793. 156

[141] Avi Ma'ayan. Complex systems biology. *Journal of The Royal Society Interface*, 14(134):20170391, sep 2017. doi: 10.1098/rsif.2017.0391. URL https://doi.org/10.1098/rsif.2017.0391. 5

[142] Robert I. Macey. Mathematical models of membrane transport processes. *Physiology of Membrane Disorders*, pages 111–131, 1986. doi: 10.1007/978-1-4613-2097-5_7. URL http://dx.doi.org/10.1007/978-1-4613-2097-5_7. 52

[143] Javier Macia, Francesc Posas, and Ricard V. Sole. Distributed computation: the new wave of synthetic biology devices. *Trends in Biotechnology*, 30 (6):342–349, jun 2012. doi: 10.1016/j.tibtech.2012.03.006. URL https://doi.org/10.1016/j.tibtech.2012.03.006. 179

[144] T. S. Mahoney, A. S. Weyrich, D. A. Dixon, T. McIntyre, S. M. Prescott, and G. A. Zimmerman. Cell adhesion regulates gene expression at translational checkpoints in human myeloid leukocytes. *Proceedings of the Na-*

tional Academy of Sciences, 98(18):10284–10289, aug 2001. doi: 10.1073/pnas.181201398. URL https://doi.org/10.1073/pnas.181201398. 154

[145] R MAITHREYE, C SUGUNA, and SOMDATTA SINHA. Collective dynamics of multicellular systems. *Pramana*, 77(5):843–853, oct 2011. doi: 10.1007/s12043-011-0197-x. URL https://doi.org/10.1007/s12043-011-0197-x. 6

[146] Mario Andrea Marchisio and Fabian Rudolf. Synthetic biosensing systems. *The International Journal of Biochemistry & Cell Biology*, 43(3):310–319, mar 2011. doi: 10.1016/j.biocel.2010.11.012. URL https://doi.org/10.1016/j.biocel.2010.11.012. 166

[147] Omer Markovitch and Doron Lancet. Excess Mutual Catalysis Is Required for Effective Evolvability. *Artificial Life*, 18(3):243–266, June 2012. ISSN 1064-5462. 299

[148] Omer Markovitch and Doron Lancet. Multispecies population dynamics of prebiotic compositional assemblies. *Journal of Theoretical Biology*, 357:26–34, September 2014. ISSN 0022-5193. 299

[149] Philip D Marsh. Dental plaque as a biofilm and a microbial community - implications for health and disease. *BMC Oral Health*, 6(Suppl 1):S14, 2006. ISSN 1472-6831. doi: 10.1186/1472-6831-6-s1-s14. URL http://dx.doi.org/10.1186/1472-6831-6-S1-S14. 145

[150] W. Martin, J. Baross, D. Kelley, and M. J. Russell. Hydrothermal vents and the origin of life. *Nature Reviews Microbiology*, 6:805–814, 2008. 318

[151] John S. Mattick. Type IV pili and twitching motility. *Annual Review of Microbiology*, 56(1):289–314, oct 2002. doi: 10.1146/annurev.micro.56.012302.160938. URL https://doi.org/10.1146/annurev.micro.56.012302.160938. 26

[152] F. Mavelli and K. Ruiz-Mirazo. Theoretical conditions for the stationary reproduction of model protocells. *Integrative Biology*, 5:324–341, 2013. 298, 318

[153] Melissa B. Miller and Bonnie L. Bassler. Quorum sensing in bacteria. *Annual Review of Microbiology*, 55(1):165–199, oct 2001. doi: 10.1146/annurev.micro.55.1.165. URL https://doi.org/10.1146/annurev.micro.55.1.165. 6, 27

[154] M. Mir, Z. Wang, Z. Shen, M. Bednarz, R. Bashir, I. Golding, S. G. Prasanth, and G. Popescu. Optical measurement of cycle-dependent cell growth. *Proceedings of the National Academy of Sciences*, 108(32):13124–13129, Jul 2011. ISSN 1091-6490. doi: 10.1073/pnas.1100506108. URL http://dx.doi.org/10.1073/pnas.1100506108. 55

[155] Waleed K Mohammed, Natalio Krasnogor, and Nicholas S Jakubovics. Streptococcus gordonii challisin protease is required for sensing cell–cell contact with actinomyces oris. *FEMS Microbiology Ecology*, 94(5), mar 2018. doi: 10.1093/femsec/fiy043. URL https://doi.org/10.1093/femsec/fiy043. 145

[156] E. Mossel and M. Steel. Random biochemical networks: The probability of self-sustaining autocatalysis. *Journal of Theoretical Biology*, 233(3):327–336, 2005. 300, 319

[157] Raqeyah Jawad Najy. The role of computer aided design (CAD) in the manufacturing and digital control (CAM). *Contemporary Engineering Sciences*, 6:297–312, 2013. doi: 10.12988/ces.2013.3736. URL https://doi.org/10.12988/ces.2013.3736. 35

[158] J. Naylor, H. Fellermann, Y. Ding, W. K. Mohammed, N. S. Jakubovics, J. Mukherjee, C. A. Biggs, P. C. Wright, and N. Krasnogor. Simbiotics: A multiscale integrative platform for 3D modeling of bacterial populations. *ACS Synthetic Biology*, 6(7):1194–1210, 2017. 303

[159] Jonathan Naylor, Harold Fellermann, Yuchun Ding, Waleed K. Mohammed, Nicholas S. Jakubovics, Felix Dafhnis-Calas, Stephan Heeb, Miguel Camara, Joy Mukherjee, Catherine A. Biggs, Phillip C. Wright, and Natalio Krasnogor. Correction to simbiotics: A multiscale integrative platform for 3d modeling of bacterial populations. *ACS Synthetic Biology*,

7(3):952–952, mar 2018. doi: 10.1021/acssynbio.8b00077. URL https://doi.org/10.1021/acssynbio.8b00077. 39

[160] Darren N. Nesbeth, Alexey Zaikin, Yasushi Saka, M. Carmen Romano, Claudiu V. Giuraniuc, Oleg Kanakov, and Tetyana Laptyeva. Synthetic biology routes to bio-artificial intelligence. *Essays In Biochemistry*, 60(4): 381–391, nov 2016. doi: 10.1042/ebc20160014. URL https://doi.org/10.1042/ebc20160014. 178

[161] A. Z. Nevesinjac and T. L. Raivio. The cpx envelope stress response affects expression of the type IV bundle-forming pili of enteropathogenic escherichia coli. *Journal of Bacteriology*, 187(2):672–686, jan 2005. doi: 10.1128/jb.187.2.672-686.2005. URL https://doi.org/10.1128/jb.187.2.672-686.2005. 26

[162] T. J. Newman and R. Grima. Many-body theory of chemotactic cell-cell interactions. *Physical Review E*, 70(5), Nov 2004. ISSN 1550-2376. doi: 10.1103/physreve.70.051916. URL http://dx.doi.org/10.1103/PhysRevE.70.051916. 47

[163] P. Nghe, W. Hordijk, S. A. Kauffman, S. I. Walker, F. J. Schmidt, H. Kemble, J. A. M. Yeates, and N. Lehman. Prebiotic network evolution: Six key parameters. *Molecular BioSystems*, 11:3206–3217, 2015. 297

[164] Peter Q. Nguyen. Synthetic biology engineering of biofilms as nanomaterials factories. *Biochemical Society Transactions*, 45(3):585–597, jun 2017. doi: 10.1042/bst20160348. URL https://doi.org/10.1042/bst20160348. 166

[165] Muaz Niazi. Complex adaptive systems modeling: A multidisciplinary roadmap. 1, 01 2013. 5

[166] A. A. K. Nielsen, B. S. Der, J. Shin, P. Vaidyanathan, V. Paralanov, E. A. Strychalski, D. Ross, D. Densmore, and C. A. Voigt. Genetic circuit design automation. *Science*, 352(6281):aac7341–aac7341, mar 2016. doi: 10.1126/science.aac7341. URL https://doi.org/10.1126/science.aac7341. 179

[167] B.W. Ninham. On progress in forces since the dlvo theory. *Advances in Colloid and Interface Science*, 83(1-3):1–17, Dec 1999. ISSN 0001-8686. doi: 10.1016/s0001-8686(99)00008-1. URL http://dx.doi.org/10.1016/S0001-8686(99)00008-1. 49

[168] M Paoluzzi, R Di Leonardo, and L Angelani. Effective run-and-tumble dynamics of bacteria baths. *Journal of Physics: Condensed Matter*, 25 (41):415102, 2013. URL http://stacks.iop.org/0953-8984/25/i=41/a=415102. 50

[169] A. E. Patteson, A. Gopinath, M. Goulian, and P. E. Arratia. Running and tumbling with e. coli in polymeric solutions. *Scientific Reports*, 5: 15761, Oct 2015. ISSN 2045-2322. doi: 10.1038/srep15761. URL http://dx.doi.org/10.1038/srep15761. 50

[170] Koushik Paul, Vincent Nieto, William C. Carlquist, David F. Blair, and Rasika M. Harshey. The c-di-GMP binding protein YcgR controls flagellar motor direction and speed to affect chemotaxis by a "backstop brake" mechanism. *Molecular Cell*, 38(1):128–139, apr 2010. doi: 10.1016/j.molcel.2010.03.001. URL https://doi.org/10.1016/j.molcel.2010.03.001. 26

[171] Gheorghe Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, aug 2000. doi: 10.1006/jcss.1999.1693. URL https://doi.org/10.1006/jcss.1999.1693. 33

[172] Alexandre Persat, Carey D. Nadell, Minyoung Kevin Kim, Francois Ingremeau, Albert Siryaporn, Knut Drescher, Ned S. Wingreen, Bonnie L. Bassler, Zemer Gitai, and Howard A. Stone. The mechanical world of bacteria. *Cell*, 161(5):988–997, may 2015. doi: 10.1016/j.cell.2015.05.005. URL https://doi.org/10.1016/j.cell.2015.05.005. 26

[173] Addy Pross. How does biology emerge from chemistry? *Origins of Life and Evolution of Biospheres*, 42(5):433–444, oct 2012. doi: 10.1007/s11084-012-9305-2. URL https://doi.org/10.1007/s11084-012-9305-2. 3

[174] Sudha Rajamani, Alexander Vlassov, Seico Benner, Amy Coombs, Felix Olasagasti, and David Deamer. Lipid-assisted synthesis of RNA-like polymers from mononucleotides. *Origins of Life and Evolution of Biospheres*, 38(1):57–74, nov 2008. doi: 10.1007/s11084-007-9113-2. URL https://doi.org/10.1007/s11084-007-9113-2. 298

[175] S. Rasmussen, M. A. Bedau, L. Chen, D. Deamer, D. C. Krakauer, N. H. Packard, and P. F. Stadler, editors. *Protocells*, 2008. MIT Press. 298

[176] Alexander H Rickard, Peter Gilbert, Nicola J High, Paul E Kolenbrander, and Pauline S Handley. Bacterial coaggregation: an integral process in the development of multi-species biofilms. *Trends in Microbiology*, 11(2):94–100, feb 2003. doi: 10.1016/s0966-842x(02)00034-3. URL https://doi.org/10.1016/s0966-842x(02)00034-3. 145

[177] Douglas Rodney. Cx3dp: A parallel framework for modeling the growth and development of neural tissue. *Frontiers in Neuroinformatics*, 5, 2011. doi: 10.3389/conf.fninf.2011.08.00106. URL https://doi.org/10.3389/conf.fninf.2011.08.00106. 104

[178] Guillermo Rodrigo, Javier Carrera, Santiago F Elena, and Alfonso Jaramillo. Robust dynamical pattern formation from a multifunctional minimal genetic circuit. *BMC Systems Biology*, 4(1):48, 2010. doi: 10.1186/1752-0509-4-48. URL https://doi.org/10.1186/1752-0509-4-48. 178

[179] FRANCISCO J. ROMERO-CAMPERO, JAMIE TWYCROSS, MIGUEL CAMARA, MALCOLM BENNETT, MARIAN GHEORGHE, and NATALIO KRASNOGOR. MODULAR ASSEMBLY OF CELL SYSTEMS BIOLOGY MODELS USING p SYSTEMS. *International Journal of Foundations of Computer Science*, 20(03):427–442, jun 2009. doi: 10.1142/s0129054109006668. URL https://doi.org/10.1142/s0129054109006668. 33, 178

[180] Lawrence I Rothfield and Chun-Rui Zhao. How do bacteria decide where to divide? *Cell*, 84(2):183–186, Jan 1996. ISSN 0092-8674.

doi: 10.1016/s0092-8674(00)80971-x. URL http://dx.doi.org/10.1016/S0092-8674(00)80971-X. 55

[181] Timothy J Rudge, Paul J Steiner, Andrew Phillips, and Jim Haseloff. Computational modeling of synthetic microbial biofilms. *ACS Synthetic Biology*, 1(8):345–352, August 2012. 33, 34, 98, 99

[182] Timothy J. Rudge, Fernan Federici, Paul J. Steiner, Anton Kan, and Jim Haseloff. Cell polarity-driven instability generates self-organized, fractal patterning of cell layers. *ACS Synthetic Biology*, 2(12):705–714, 2013. doi: 10.1021/sb400030p. URL https://doi.org/10.1021/sb400030p. PMID: 23688051. 96, 97, 98

[183] Daven Sanassy, Harold Fellermann, Natalio Krasnogor, Savas Konur, Laurentiu M. Mierla, Marian Gheorghe, Christophe Ladroue, and Sara Kalvala. Modelling and stochastic simulation of synthetic biological boolean gates. *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, Aug 2014. doi: 10.1109/hpcc.2014.68. URL http://dx.doi.org/10.1109/HPCC.2014.68. 30, 33

[184] Daven Sanassy, Pawel Widera, and Natalio Krasnogor. Meta-stochastic simulation of biochemical models for systems and synthetic biology. *ACS Synthetic Biology*, 4(1):39–47, Jan 2015. ISSN 2161-5063. doi: 10.1021/sb5001406. URL http://dx.doi.org/10.1021/sb5001406. 7, 31

[185] Tim Sandle. Bacterial adhesion: an introduction. 17, 06 2013. 26

[186] Maximilian M. Sauer, Roman P. Jakob, Jonathan Eras, Sefer Baday, Deniz Eris, Giulio Navarra, Simon Berneche, Beat Ernst, Timm Maier, and Rudi Glockshuber. Catch-bond mechanism of the bacterial adhesin fimh. *Nat Comms*, 7:10738, Mar 2016. ISSN 2041-1723. doi: 10.1038/ncomms10738. URL http://dx.doi.org/10.1038/ncomms10738. 145

[187] E. R. Scerri. Top-down causation regarding the chemistry-physics interface: a sceptical view. *Interface Focus*, 2(1):20–25, nov 2011. doi: 10.1098/rsfs.2011.0061. URL https://doi.org/10.1098/rsfs.2011.0061. 3

[188] James Scott-Brown and Antonis Papachristodoulou. sbml-diff: A tool for visually comparing SBML models in synthetic biology. *ACS Synthetic Biology*, 6(7):1225–1229, jan 2017. doi: 10.1021/acssynbio.6b00273. URL https://doi.org/10.1021/acssynbio.6b00273. 32

[189] Daniel Segree, Dafna Ben-Eli, David W. Deamer, and Doron Lancet. *Origins of Life and Evolution of the Biosphere*, 31(1/2):119–145, 2001. doi: 10.1023/a:1006746807104. URL https://doi.org/10.1023/a:1006746807104. 299

[190] Annalese B. T. Semmler, Cynthia B. Whitchurch, and John S. Mattick. A re-examination of twitching motility in pseudomonas aeruginosa. *Microbiology*, 145(10):2863–2873, oct 1999. doi: 10.1099/00221287-145-10-2863. URL https://doi.org/10.1099/00221287-145-10-2863. 6

[191] R. Serra and M. Villani. *Modelling Protocells*. Springer, 2017. 298, 318

[192] R. Serra, A. Filisetti, M. Villani, A. Graudenzi, C. Damiani, and T. Panini. A stochastic model of catalytic reaction networks in protocells. *Natural Computing*, 13(3):367–377, 2014. 299, 305

[193] Philip Shapira, Seokbeom Kwon, and Jan Youtie. Tracking the emergence of synthetic biology. *Scientometrics*, 112(3):1439–1469, jul 2017. doi: 10.1007/s11192-017-2452-5. URL https://doi.org/10.1007/s11192-017-2452-5. 7, 138

[194] James A. Shapiro. Bacteria as multicellular organisms. *Scientific American*, 258(6):82–89, jun 1988. doi: 10.1038/scientificamerican0688-82. URL https://doi.org/10.1038/scientificamerican0688-82. 33

[195] Guo-Ping Sheng, Han-Qing Yu, and Xiao-Yan Li. Extracellular polymeric substances (eps) of microbial aggregates in biological wastewater treatment systems: A review. *Biotechnology Advances*, 28(6):882–894, Nov 2010. ISSN

0734-9750. doi: 10.1016/j.biotechadv.2010.08.001. URL http://dx.doi.org/10.1016/j.biotechadv.2010.08.001. 58

[196] D. Sievers and G. von Kiedrowski. Self-replication of complementary nucleotide-based oligomers. *Nature*, 369:221–224, 1994. 319

[197] T. J. Silhavy, D. Kahne, and S. Walker. The bacterial cell envelope. *Cold Spring Harbor Perspectives in Biology*, 2(5):a000414–a000414, apr 2010. doi: 10.1101/cshperspect.a000414. URL https://doi.org/10.1101/cshperspect.a000414. 26

[198] Szymon Skoneczny. Cellular automata-based modelling and simulation of biofilm structure on multi-core computers. *Water Science and Technology*, 72(11):2071–2081, aug 2015. doi: 10.2166/wst.2015.426. URL https://doi.org/10.2166/wst.2015.426. 33

[199] J. Smith, M. Steel, and W. Hordijk. Autocatalytic sets in a partitioned biochemical network. *Journal of Systems Chemistry*, 5:2, 2014. 300, 301

[200] F. L. Sousa, W. Hordijk, M. Steel, and W. F. Martin. Autocatalytic sets in e. coli metabolism. *Journal of Systems Chemistry*, 6:4, 2015. 300

[201] Kansuporn Sriyudthsak, Fumihide Shiraishi, and Masami Yokota Hirai. Mathematical modeling and dynamic simulation of metabolic reaction systems using metabolome time series data. *Frontiers in Molecular Biosciences*, 3, may 2016. doi: 10.3389/fmolb.2016.00015. URL https://doi.org/10.3389/fmolb.2016.00015. 31

[202] Susan Stepney and Fiona A.C. Polack. *Engineering Simulations as Scientific Instruments: A Pattern Language: With Kieran Alden, Paul S. Andrews, James L. Bown, Alastair Droop, Richard B., T. Sampson, Jon Timmis, Alan F.T. Winfield*. Springer, 2018. ISBN 3030019373. 200

[203] Elizabeth J. Stewart, Mahesh Ganesan, John G. Younger, and Michael J. Solomon. Artificial biofilms establish the role of matrix interactions in staphylococcal biofilm assembly and disassembly. *Scientific Reports*, 5

(1), aug 2015. doi: 10.1038/srep13081. URL https://doi.org/10.1038/srep13081. 156

[204] Victoria Stodden, Jennifer Seiler, and Zhaokun Ma. An empirical analysis of journal policy effectiveness for computational reproducibility. *Proceedings of the National Academy of Sciences*, 115(11):2584–2589, mar 2018. doi: 10.1073/pnas.1708290115. URL https://doi.org/10.1073/pnas.1708290115. 96, 200

[205] P. Stoodley, K. Sauer, D. G. Davies, and J. W. Costerton. Biofilms as complex differentiated communities. *Annual Review of Microbiology*, 56(1): 187–209, oct 2002. doi: 10.1146/annurev.micro.56.012302.160705. URL https://doi.org/10.1146/annurev.micro.56.012302.160705. 6

[206] Jesse Stricker, Scott Cookson, Matthew R. Bennett, William H. Mather, Lev S. Tsimring, and Jeff Hasty. A fast, robust and tunable synthetic gene oscillator. *Nature*, 456(7221):516–519, oct 2008. doi: 10.1038/nature07389. URL https://doi.org/10.1038/nature07389. 7

[207] Tamas Szekely and Kevin Burrage. Stochastic simulation in systems biology. *Computational and Structural Biotechnology Journal*, 12(20-21):14–25, nov 2014. doi: 10.1016/j.csbj.2014.10.003. URL https://doi.org/10.1016/j.csbj.2014.10.003. 31

[208] tagkey1986ii. Other titles in the course of theoretical physics by l.d. landau and e.m. lifshitz. In E.M. Lifshitz, A.M. Kosevich, and Pitaevskii L.P., editors, *Theory of Elasticity (Third Edition)*, pages ii –. Butterworth-Heinemann, Oxford, third edition edition, 1986. ISBN 978-0-08-057069-3. doi: http://dx.doi.org/10.1016/B978-0-08-057069-3.50001-2. URL http://www.sciencedirect.com/science/article/pii/B9780080570693500012. 48

[209] tagkey1991ii. Dedication. In Stephen Cooper, editor, *Bacterial Growth and Division*, pages ii –. Academic Press, San Diego, 1991. ISBN 978-0-12-187905-1. doi: http://dx.doi.org/10.1016/B978-0-08-091747-4.

50001-1. URL http://www.sciencedirect.com/science/article/pii/B9780080917474500011. 55

[210] H. Takizawa, K. Nakamura, A. Tabira, Y. Chikahara, T. Matsui, N. Hiroi, and A. Funahashi. Libsbmlsim: a reference implementation of fully functional sbml simulator. *Bioinformatics*, 29(11):1474–1476, Apr 2013. ISSN 1460-2059. doi: 10.1093/bioinformatics/btt157. URL http://dx.doi.org/10.1093/bioinformatics/btt157. 59

[211] Ming te Lu and Wing lok Yeung. A framework for effective commercial web application development. *Internet Research*, 8(2):166–173, may 1998. doi: 10.1108/10662249810211638. URL https://doi.org/10.1108/10662249810211638. 35

[212] Hiroyuki Terashima, Seiji Kojima, and Michio Homma. Chapter 2 flagellar motility in bacteria. In *International Review of Cell and Molecular Biology*, pages 39–85. Elsevier, 2008. doi: 10.1016/s1937-6448(08)01402-0. URL https://doi.org/10.1016/s1937-6448(08)01402-0. 26

[213] Elizabeth Thursby and Nathalie Juge. Introduction to the human gut microbiota. *Biochemical Journal*, 474(11):1823–1836, may 2017. doi: 10.1042/bcj20160510. URL https://doi.org/10.1042/bcj20160510. 6

[214] M. J. Tindall, P. K. Maini, S. L. Porter, and J. P. Armitage. Overview of mathematical approaches used to model bacterial chemotaxis ii: Bacterial populations. *Bulletin of Mathematical Biology*, 70(6):1570–1607, Jul 2008. ISSN 1522-9602. doi: 10.1007/s11538-008-9322-5. URL http://dx.doi.org/10.1007/s11538-008-9322-5. 50

[215] Akif Uzman. Molecular biology of the cell (4th ed.): Alberts, b., johnson, a., lewis, j., raff, m., roberts, k., and walter, p. *Biochemistry and Molecular Biology Education*, 31(4):212–214, Jul 2003. ISSN 1539-3429. doi: 10.1002/bmb.2003.494031049999. URL http://dx.doi.org/10.1002/bmb.2003.494031049999. 53

[216] N. Vaidya, M. L. Manapat, I. A. Chen, R. Xulvi-Brunet, E. J. Hayden, and N. Lehman. Spontaneous network formation among cooperative RNA replicators. *Nature*, 491:72–77, 2012. 319

[217] V. Vasas, E. Szathmary, and M. Santos. Lack of evolvability in self-sustaining autocatalytic networks constrains metabolism-first scenarios for the origin of life. *PNAS*, 107(4):1470–1475, 2010. 298, 310

[218] V. Vasas, C. Fernando, M. Santos, S. Kauffman, and E. Sathmary. Evolution before genes. *Biology Direct*, 7:1, 2012. 298, 299, 311, 317, 318, 319

[219] V. Vemuri. *Modelling of Complex Systems: An Introduction (Operations research and industrial engineering)*. Academic Press Inc, 1978. ISBN 0127165509. 5

[220] M. Villani, A. Filisetti, A. Graudenzi, C. Damiani, T. Carletti, and R. Serra. Growth and division in a dynamic protocell model. *Life*, 4:837–864, 2014. 299

[221] Vishwas Mahesh. Role of computer aided design and engineering in product development. 2013. doi: 10.13140/rg.2.1.4551.4323. 35

[222] Anton Vitvitsky. Cellular automata simulation of bacterial cell growth and division. In *Designing Beauty: The Art of Cellular Automata*, pages 121–123. Springer International Publishing, 2016. doi: 10.1007/978-3-319-27270-2_19. URL https://doi.org/10.1007/978-3-319-27270-2_19. 33

[223] William G. Wade and Martin A. Slayne. Controlling plaque by disrupting the process of plaque formation. *Periodontology 2000*, 15(1):25–31, oct 1997. doi: 10.1111/j.1600-0757.1997.tb00101.x. URL https://doi.org/10.1111/j.1600-0757.1997.tb00101.x. 145

[224] Charles C.N. Wang, Ka-Lok Ng, Yu-Ching Chen, Phillip C.Y. Sheu, and Jeffrey J.P. Tsai. Simulation of bacterial chemotaxis by the random run and tumble model. *2011 IEEE 11th International Conference on Bioinformatics*

and Bioengineering, Oct 2011. doi: 10.1109/bibe.2011.41. URL http://dx.doi.org/10.1109/BIBE.2011.41. 50

[225] Jue D. Wang and Petra A. Levin. Metabolism, cell growth and the bacterial cell cycle. *Nature Reviews Microbiology*, 7(11):822–827, oct 2009. doi: 10.1038/nrmicro2202. URL https://doi.org/10.1038/nrmicro2202. 23, 24

[226] Rui-Sheng Wang, Assieh Saadatpour, and Reka Albert. Boolean modeling in systems biology: an overview of methodology and applications. *Physical Biology*, 9(5):055001, sep 2012. doi: 10.1088/1478-3975/9/5/055001. URL https://doi.org/10.1088/1478-3975/9/5/055001. 30

[227] James Michael Whitacre. Biological robustness: Paradigms, mechanisms, and systems principles. *Frontiers in Genetics*, 3, 2012. doi: 10.3389/fgene.2012.00067. URL https://doi.org/10.3389/fgene.2012.00067. 3

[228] C. R. Woese. On the evolution of cells. *Proceedings of the National Academy of Sciences*, 99(13):8742–8747, jun 2002. doi: 10.1073/pnas.132266999. URL https://doi.org/10.1073/pnas.132266999. 318

[229] C. R. Woese and G. E. Fox. Phylogenetic structure of the prokaryotic domain: The primary kingdoms. *Proceedings of the National Academy of Sciences*, 74(11):5088–5090, nov 1977. doi: 10.1073/pnas.74.11.5088. URL https://doi.org/10.1073/pnas.74.11.5088. 318

[230] Richard Wolfenden and Mark J. Snider. The depth of chemical time and the power of enzymes as catalysts. *Accounts of Chemical Research*, 34(12): 938–945, dec 2001. doi: 10.1021/ar000058i. URL https://doi.org/10.1021/ar000058i. 310

[231] Stephen Wolfram. Computation theory of cellular automata. *Communications in Mathematical Physics*, 96(1):15–57, mar 1984. doi: 10.1007/bf01217347. URL https://doi.org/10.1007/bf01217347. 33

[232] Kang Wu and Christopher V Rao. Computational methods in synthetic biology: towards computer-aided part design. *Current Opinion in Chemical*

Biology, 16(3-4):318–322, aug 2012. doi: 10.1016/j.cbpa.2012.05.003. URL https://doi.org/10.1016/j.cbpa.2012.05.003. 138

[233] Spyros Xanthopoulos and Stelios Xinogalos. A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics on - BCI '13*. ACM Press, 2013. doi: 10.1145/2490257.2490292. URL https://doi.org/10.1145/2490257.2490292. 35

[234] Fang Xie, Edward M. Eddy, and Marco Conti. Analysis of signaling pathways controlling flagellar movements in mammalian spermatozoa. In *Methods in Enzymology*, pages 91–104. Elsevier, 2013. doi: 10.1016/b978-0-12-397945-2.00006-8. URL https://doi.org/10.1016/b978-0-12-397945-2.00006-8. 26

[235] Lingchong You, Robert Sidney Cox, Ron Weiss, and Frances H. Arnold. Programmed population control by cell-cell communication and regulated killing. *Nature*, 428(6985):868–871, apr 2004. doi: 10.1038/nature02491. URL https://doi.org/10.1038/nature02491. 7

[236] Kevin D Young. Bacterial morphology: why have different shapes? *Current Opinion in Microbiology*, 10(6):596–600, dec 2007. doi: 10.1016/j.mib.2007.09.009. URL https://doi.org/10.1016/j.mib.2007.09.009. 26

[237] Vipul Periwal Zoltan Szallasi, Jorg Stelling. *System Modeling in Cellular Biology: From Concepts to Nuts and Bolts (The MIT Press)*. The MIT Press, 2006. ISBN 0262195488. 31

[238] Frederic Zubler. A framework for modeling the growth and development of neurons and networks. *Front. Comput. Neurosci.*, 3, 2009. doi: 10.3389/neuro.10.025.2009. URL http://dx.doi.org/10.3389/neuro.10.025.2009. 45, 104, 209

[239] Paolo Zuliani. Statistical model checking for biological applications. *International Journal on Software Tools for Technology Transfer*, 17(4):527–536, aug 2014. doi: 10.1007/s10009-014-0343-0. URL https://doi.org/10.1007/s10009-014-0343-0. 32