Towards the Implementation of Distributed Systems in Synthetic Biology

Neythen J. Treloar

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Cell and Developmental Biology

University College London

February 3, 2022

I, Neythen J. Treloar, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

The design and construction of engineered biological systems has made great strides over the last few decades and a growing part of this is the application of mathematical and computational techniques to problems in synthetic biology. The use of distributed systems, in which an overall function is divided across multiple populations of cells, has the potential to increase the complexity of the systems we can build and overcome metabolic limitations. However, constructing biological distributed systems comes with its own set of challenges. In this thesis I present new tools for the design and control of distributed systems in synthetic biology. The first part of this thesis focuses on biological computers. I develop novel design algorithms for distributed digital and analogue computers composed of spatial patterns of communicating bacterial colonies. I prove mathematically that we can program arbitrary digital functions and develop an algorithm for the automated design of optimal spatial circuits. Furthermore, I show that bacterial neural networks can be built using our system and develop efficient design tools to do so. I verify these results using computational simulations. This work shows that we can build distributed biological computers using communicating bacterial colonies and different design tools can be used to program digital and analogue functions. The second part of this thesis utilises a technique from artificial intelligence, reinforcement learning, in first the control and then the understanding of biological systems. First, I show the potential utility of reinforcement learning to control and optimise interacting communities of microbes that produce a biomolecule. Second, I apply reinforcement learning to the design of optimal characterisation experiments within synthetic biology. This work shows that methods utilising reinforcement learning show promise for complex distributed bioprocessing in industry and the design of optimal experiments throughout biology.

Impact Statement

The work presented here has the potential to impact both academia and industry. I have developed new theoretical frameworks for spatially distributed digital and analogue computing using populations of cells. Furthermore, I have developed and demonstrated design methods for the construction of digital and analogue computers. Within academia, this work will have impact in the field of synthetic biocomputing and could lay foundations for many more advances. More widespread impact could include advances such as point of care medical diagnostic devices with complex information processing capabilities.

I have also investigated the use of deep reinforcement learning to control microbial communities in bioreactors and the design of optimal experiments within synthetic biology. As the capabilities of synthetic biology increase, there will be a strong incentive to leverage the capabilities of microbial communities in industrial bioprocessing. Two barriers to adoption is the difficulty of controlling such communities and the challenge of deriving accurate system models. This work has tackled both of these problems and opens the door to such approaches in industry. Furthermore, it will form the basis of future experimental work in academia such as characterisation experiments using desktop bioreactors or microfluidics.

Acknowledgements

Firstly, I would like to thank Prof. Chris Barnes for the support he has given me throughout my PhD. I am grateful for all his guidance and for giving me the autonomy to explore many ideas beyond those presented here. I am very thankful that I got the opportunity to work in his group exploring such exciting ideas and I am looking forward to continuing with this work in the near future. I have achieved more in the last three years than I would've thought possible when I started.

I would also like to thank everyone in the Barnes group for creating such a welcoming environment. Special thanks go to Jack, Luca and Bez for being great office mates and friends and showing the me best places to buy scones and korean fried chicken. To Alex Fedorec for all his help and guidance, and to Ke Yan for bringing some of the theoretical work presented here to life in the lab. I would also like to thank all the collaborators who took the time to share ideas with me, especially Brian Ingalls who supported me during my time in his group and made much of the work in this thesis possible. I would also like to thank the ERC for funding my research and the Bogue fellowship for awarding me funding to spend a summer in Canada.

Lastly, I would like to thank my friends and family both in London and at home who have supported me throughout this time. Especially those who made the times of national lock down bearable.

Contents

List of Figures

List of Tables

Chapter 1

Background

Over the last couple of decades advanced DNA manipulation techniques have led to the emergence of the field of synthetic biology. Synthetic biology aims to apply engineering principles to the design and construction of biological systems from the ground up. One of the key challenges is the development and application of new design and control methodologies. Traditionally, synthetic biology has involved engineering the genetic reaction networks (GRNs) within single cell populations. However, just like natural systems such as multi-cellular organisms and bacterial communities, there are many advantages to having multiple populations of cells including specialisation and task distribution. Additionally, there are metabolic limits on the complexity of circuits that can be built into any single bacterial cell. Due to the complexity of the living cells there are many challenges unique to engineering within biological systems. These include context dependence of parts, non-orthogonality of signals, growth dependent effects and potential unintended consequences of an engineered perturbation to the genome. The complexity of these problems is compounded when designing and controlling distributed biological systems, potentially composed of many different cell types. In this thesis I develop new methods for the design, control and understanding of distributed synthetic biological systems for use in biological computing and bio-processing. Chapters 3 and 4 are concerned with the design of spatially distributed biological computers, investigating both digital (Chapter 3) and analogue (Chapter 4) computing. Chapters 5 and 6 use a technique from machine learning called reinforcement learning to tackle two problems in synthetic biology. Chapter 5 applies reinforcement learning to the control of synthetic communities performing distributed bioprocessing and Chapter 6 uses it for optimal experimental design.

1.1 Overview of computational concepts

1.1.1 Digital computing

A digital function is restricted to operating in the binary domain, mapping inputs in the form of 0s and 1s to a similarly restricted output. Computing in the purely digital domain has advantages such as reliability, resistance to noise and easy implementation using transistors. This has lead to the advent of general purpose digital computers which continue to have utility in almost every facet of modern life. The input-output mapping of a digital function is commonly represented using a truth table, which defines which combinations of inputs are mapped to ON and OFF. Just like an algebraic equation that can be rearranged and simplified while still expressing the same relation, for any given truth table there are many equivalent digital functions of varying complexity that encode it. A key consideration when building an electronic digital circuit is to find the simplest form of a given truth table for physical implementation. This led to the development of a manual method based on grouping outputs using a visual representation of a function called a Karnaugh map [1] and later an automated design algorithm called the Espresso algorithm [2] which is now ubiquitous in electronic digital circuit design.

1.1.2 Analogue computing with neural networks

Inspired by the networks of biological neurons that form the brain, artificial neural networks (ANNs) are composed of many artificial neurons connected in a flexible manner that allows learning. Each artificial neuron receives input, either externally or from other neurons, and performs a small computation. The neuron then passes the result onto subsequent neurons or out as the networks output. The strength of connection between two neurons is dictated by a weight. Inputs to a neural network are vectors or multidimensional matrices of numbers. Like sensory inputs, these can represent many things including images, time series data or sound waves. The output of the network can represent predictions, classifications or any other function of the inputs. Neural networks are not restricted to operating in the digital domain and can therefore be considered as performing analogue computations.

There are multiple architectures of neural network which connect the composite neurons in different ways. Feedforward networks are composed of multiple layers of neurons, each neuron receives input from the neurons in the previous layer and passes its output on to the neurons in the next layer. Convolutional networks are inspired by biological vision and excel at feature extraction and image classification. Recurrent networks maintain a persistent internal state meaning they excel at processing sequences of inputs, such as sentences or time series data. These different architectures can be used together and a single network often contains convolutional, recurrent and feedforward parts.

ANNs are trained on data, usually using a method called backpropagation [3]. First the data is usually split into small groups called mini-batches, the data-points within a mini-batch are processed simultaneously in parallel. During backpropagation mini-batches of training data are sent through the network to calculate the predicted output for each data point, this is called the forward pass. The error in the predicted output is calculated using a loss function, commonly the mean squared error. Using the chain rule of differentiation, this error is propagated backwards through the network to find the change in the network's weights that would reduce the error over the current mini-batch. The weights are then moved incrementally in this direction and the size of the change is dictated by a hyper parameter called the learning rate. If necessary this process can be repeated for many epochs, where each epoch is a run through the entire training data set, until the network is fully trained and can predict the targets with low error. Neural networks are proven universal function approximators when they have at least one hidden layer [4, 5]. They underpin the field of deep learning within machine learning and have been instrumental in many of the recent advancements in artificial intelligence. Their power lies in the ability to learn complex functions from data and the flexibility to be applied to many types of problem. They have been applied to image classification [6], non linear regression [7] and protein folding prediction [8], and have been used in the field of deep reinforcement learning to achieve superhuman performance in video [9, 10] and board [11] games.

1.2 Computing in synthetic biological systems

The unique properties of biological molecules can be exploited by using them as computational substrates. This could result in superior performance over traditional computers in certain applications [12]. DNA programming has been used to solve an NP-complete Hamiltonian path problem [13] and to construct AND, NOT and XOR logic gates [14]. These logic gates were then combined in the construction of a molecular half adder [15] and then a full-adder [16]. Further building on this, three games were designed in which a human can supply input using oligonucleotides and a fluorescence output is given [15, 17, 18]. DNA computing has even been used to implement a small neural network of four fully connected neurons [19]. A non-deterministic universal Turing machine was built out of DNA [20], capable of solving non-deterministic polynomial (NP) time problems in polynomial time. Like DNA, RNA can be exploited as a computational substrate. An RNA based circuit capable of sensing microRNAs and classifying cancer cells was demonstrated for the selective apoptosis of HeLa cells [21]. A later work also showed that microRNAs can be used to build logic circuits in mammalian cells and demonstrated the selective elimination of target cells [22]. RNA based logic has also been demonstrated in *E. coli* [23], where circuits including a 12 input digital function were built. Automated design programs have been developed for the *in silico* design of RNA based circuits, including smallRNA based multi-state devices [24] and microRNA based classifiers [25]. Reactions mediated by enzymes can be seen as logic gates, for example a reaction of the form $A + B \rightarrow C$ can be seen as an AND gate with respect to inputs A and B, with C the output. In similar ways many logic gates have been built, including AND [26], OR [27], NAND [28], NOR [26], and XOR [26]. These logic gates can be combined to make logical networks [29]. These have been shown to have medical applications, for example in the processing on bio-markers related to traumatic brain injury and soft tissue injury to give an accurate diagnosis [30]. Another approach [31] used agents propelled by molecular motors to explore a nano-fabricated network and solve the NP subset sum problem, here an exponentially growing number of agents was required in place of exponential time.

The first synthetic biology papers engineered a toggle switch [32], oscillator [33] and autoregulation [34], which can be used as fundamental components in engineering a computer [35]: memory, clock and noise filter. Since then, the tools necessary for engineering microbes for computation have been extensively developed over the last two decades of synthetic biology research. Consequently engineering the genetic reaction networks (GRNs) within cells has emerged as another dominant paradigm of biological computing. Transcriptional networks that implement digital logic gates have been extensively investigated. An AND gate that integrates the output of two promoters has been implemented in single cells [36] and later more complex digital circuits were created by wiring together multiple layers of orthogonal AND gates [37]. We now have libraries of orthogonal repressor-promoter NOT gates [38], as well as the ability to produce de novo CRISPR-dCas9 gates [39], that can be wired together to make complex digital functions [40]. These advances, along with tools to reduce DNA context effects [41, 42, 43] have enabled the construction of digital circuits with a great of deal complexity in common lab strains of bacteria as well as strains relevant to microbiome engineering [44]. This level of circuit complexity is only achievable through the use of automated design tools, such as Cello [40], which match the empirical properties of genetic logic gates to ensure they will function together.

Biological processes in cells, based on the continuous concentration of metabolites and other molecules, are naturally analogue. Analogue computing is more efficient, in terms of the rate of ATP consumption and the number of protein molecules required, for doing addition with a genetic circuit at the ranges of precision that are metabolically feasible in single cells [45]. Additionally, it has been shown that building the equivalent circuit using analogue logic can require orders of magnitude fewer genetic parts [46, 47]. Analogue sensing, addition, and ratiometric and power law computations have been implemented using only three transcription factors [47]. Perceptrons have been implemented using enzymes that transduce different inputs into a common output molecule, benzoate, and a synthetic actuator circuit that sensed benzoate [48]. This was used to build a cell based adder and cell free metabolic perceptrons in which enzyme concentrations acted as weights [48].

1.2.1 Capabilities of realised distributed computers

One of the key engineering principles that synthetic biology strives to adhere to is modularity, so that biological components can be recombined and interchanged to build new systems. A successful example within the context of synthetic biological distributed systems is the decomposition of a complex digital function into multiple subunits, each engineered within a different population of cells that communicate with each other (Figure 1.1A). This mirrors a common approach in electronics where two universal logic gates, for example NOR and NAND, are wired together to produce any digital function. In this manner all 16 two input logic gates have been created using bacterial colonies on agar plates, containing genetically engineered NOR gates, and communicating via diffusible molecules [49]. A similar approach consisted of a community of yeast cells that carried out the functions AND, NIMPLIES, NOT and IDENTITY [50]. These are chemically wired together using diffusible communication molecules to produce complex functions. Mathematical work into the optimal design of computational communities implementing distributed genetic logic gates given realistic constraints on the number of logic gates possible per cell and the number of orthogonal quorum molecules has been done [51]. Another automated design framework for the construction of user specified functions using DNA recombinase NOT and IDENTITY gates distributed over multiple cell types enables the design of a consortium of bacteria to perform the desired digital computation [52]. This framework was then used to build consortia capable of four input digital logic [53]. The standard mathematical proof that any Boolean function can be decomposed into a double summation of IDENTITY and NOT logics was used to build multicellular circuits encoding the IDENTITY and NOT logic into cells and then performing sums by mixing cell cultures together [54]. A subsequent paper simplified the implementation of digital functions by printing multiple cell populations onto branched paper devices using ‘cellular ink’ [55]. It was shown that these devices could be stored for up to ten days and still be effective. Furthermore, conjugation between bacterial cells has been exploited as a direct message passing system, and has been used to design, *in silico*, a community of distributed NOR gates wired together for a population level XOR gate. Finally, a distributed implementation of an N-bit counter has been designed, *in silico*, in which a set of N single bit counters and connector modules are connected using diffusible molecules [56].

A key component for computation is memory. Quorum sensing has been combined with a genetic toggle switch, resulting in a population level toggle switch [57]. A synthetic community composed of E. coli strains has been used to record the order, duration and timing of chemical events [58]. A bistable switch was built across two distinct cell types, controllable by two different yeast pheromones, that switched the community between two states [59]. The simulation of a design for a flip flop memory device distributed over four populations of cells show that its function is robust to changes in parameters and that circuit behaviour can be tuned by changing experimental conditions [60]. Another computational investigation showed how a co-culture of two bacterial strains could be used to do associative learning, with both short- and long-term memory [61].

Unlike electronic computers, biological systems are able to change their “hardware” depending on the task at hand by, for example, dynamically controlling the constituents of a community (Figure 1.1C). Two independent auxotrophic E. coli populations have been designed so that their growth is tuneable by inducing production of amino acids [63]. Using a community of microbes that inhabit slightly different temperature niches, a temperature cycling scheme is able to dynamically tune the community [66]. Methods of intrinsic community composition control can be built into cells genetically. This has been done using self-inhibition using quorum molecule signalling [67]. Simulation results also show that a population of cells containing a reconfigurable logic gate that can be switched between NOR and NAND behaviour [68]. Furthermore, a rock-paper-scissors system of three

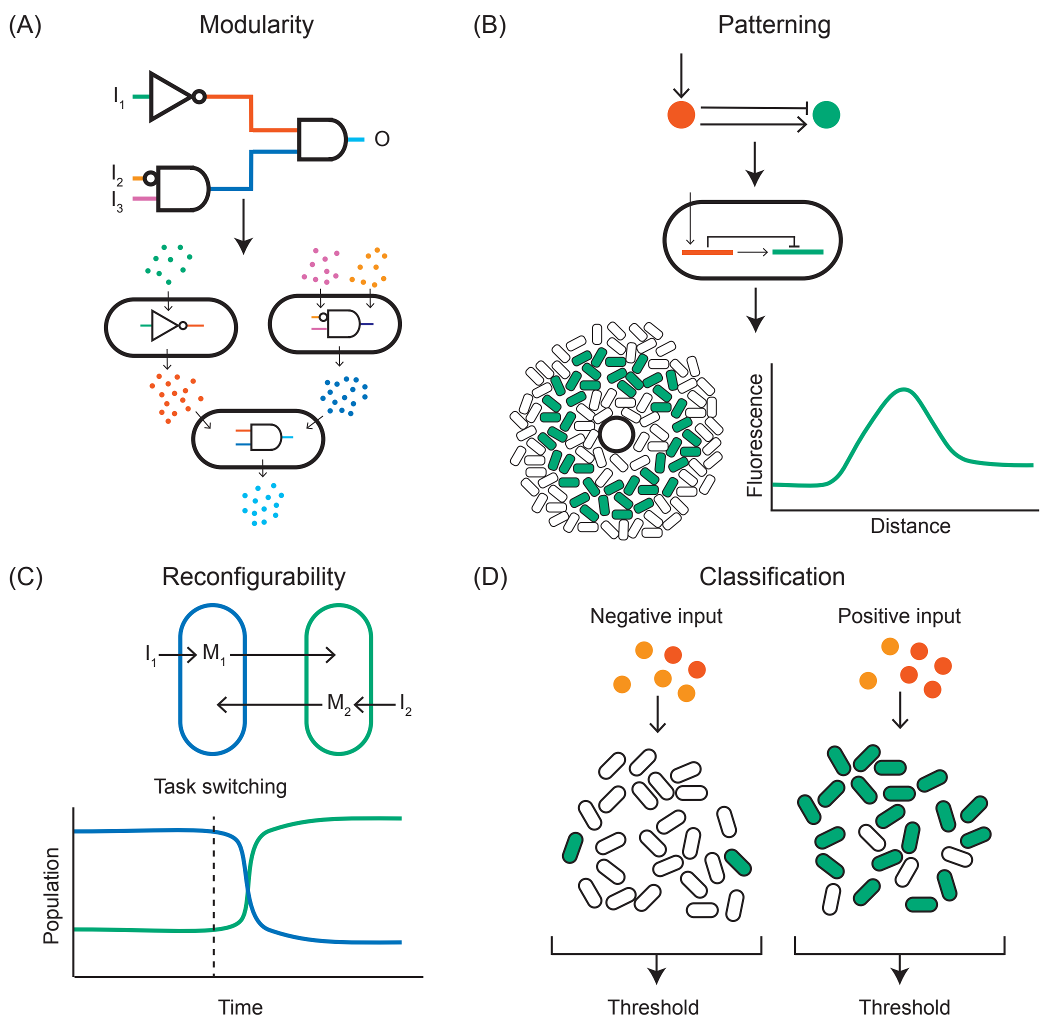


Figure 1.1: Example capabilities of computational communities. A) A complex circuit can be split into modules, distributed across different populations of cells. Adapted from [50] . B) Computational methods can be used to find networks capable of stripe formation, these can be programmed into cells using genetic circuits, the expressed phenotype of each cell depends on its position relative to a source of signalling molecule. Adapted from [62]. C) Reconfigurability could be a key capability of biological computing. Here the composition of a bacterial community can be controlled through inducers I1 and I2. This capability could be used to task switch in a computational bacterial community. Adapted from [63]. D) Bacterial communities are naturally applicable to complex functions such as ensemble classification. Adapted from [64]. This figure is reproduced from [65].

populations of E. coli that cyclically inhibit one another, combined with population dependent synchronised lysis, shows the capability to cycle the community composition through the three strains [69]. A further method has been proposed, *in silico*, that uses the manipulation of plasmid copy numbers to embed multiple computations in a population of cells [70].

Classifiers aim to identify which category an observation belongs to. Biological classifiers have been built to identify cancer cells using miRNA [21, 25]. A key concept in machine learning is the use of ensemble methods. These combine the output of many individual weak classifiers, which perform at least slightly better than random choice, and produce an overall output with much greater accuracy. This methodology can naturally be applied to a community of cells, where each cell contains a genetically encoded weak classifier and the overall community output is computed by combining the individual outputs of all cells (Figure 1.1D). This approach has been investigated *in silico*. For each data point in a training data set a heterogenous population of cells containing weak classifiers vote on the answer [64]. The community learns as cells are stochastically pruned from the population; cells that voted incorrectly are removed with a higher probability. A multi-input classifier composed of a community of cells containing either a linear or a bell-shaped classifier was simulated and found to be able to represent practically arbitrary shapes in the input space [64]. Other numerical results on a similar population of cells showed that complex classification problems could be tackled [71].

Both multicellular organisms and communities of unicellular organisms have the ability to cooperate to produce spatial structures that allow them to better perform complex functions. The prime example of this phenomena is development in multicellular organisms, in which cells containing identical DNA differentiate and organise themselves spatially to assemble a complex organism. Harnessing this capability could mean the realisation of biological computers that can self-assemble and reproduce in a manner that is not currently possible with silicon systems. The first step in this direction was taken by engineering E. coli ‘receiver cells’ which respond to a quorum molecule with a band detect activation [72]. Sources of the quorum molecule could then be used to produce different patterns of fluorescence in a lawn of E. coli. This approach was complemented by the development of quorum molecule producing ‘sender’ cells [73]. Work has also been undertaken using senders and receivers to produce 3D patterning of mammalian cells [74]. It is possible that sender and receiver cells could be combined to produce dynamic pattern formation in response to environmental changes. The value of using computational modelling to investigate pattern formation and design spatially structured synthetic communities has been shown (Figure 1.1B) [62]. Here the space of two and three-node, stripe forming networks was investigated computationally, and used to inform wet laboratory experiments. Further computational investigation using the modelling platform GRO [75, 76] acts as a proof of concept for the design of bacterial colonies capable of self-assembling into spatial structures including L and T shapes [77]. It has also been shown that synthetic communities engineered to grow with a ring shaped pattern show scale invariance, similar to natural systems [78]. An artificial symmetry breaking mechanism was combined with domain specific cellular regulation resulting in artificial patterning and cell differentiation reminiscent of a simple developmental process [79]. Interactions between motile and non-motile bacteria when grown together in a biofilm have been shown to trigger the emergence of complex patterns over time [80].

1.3 Distributed bio-processing

The ability to engineer cells at the genetic level has enabled the research community to make use of biological organisms for many functions, including the production of biofuels [81, 82, 83], pharmaceuticals [84] and the processing of waste products [85]. Co-cultures consisting of multiple distinct populations of cells have been shown to be more productive than monocultures at performing processes such as biofuel production [82, 83, 86] and alleviate the problem of metabolic burden that occurs when a large pathway is built within a single cell [87]. For these reasons co-cultures should be fundamental to the advancement of bioprocessing. However, maintaining a co-culture presents its own set of problems. The competitive exclusion principle states that if multiple populations are competing for a single resource and there are no other interactions, a species with an advantage will drive the others to extinction. An additional challenge is that the interactions between different populations of bacteria can make long term behaviour in a co-culture difficult to predict [88]; the higher the number of distinct populations, the greater the challenge becomes to ensure system stability [89]. Previous methods of co-culture population control have been engineered into cells genetically. For example, using predator-prey systems [90], mutualism [87, 91] or amensalism and competitive exclusion [92]. However, processes such as horizontal gene transfer and mutation make the long term genetic stability of a population hard to guarantee [88], meaning that genetic control methods can become less effective over time. Another potential problem is the increased metabolic load imposed on a cell due to the control genes, which can leave less resources for growth and the production of useful products [93]. These downsides can be avoided by exerting control over the environment, which is the dominant approach in industry. Established techniques are Proportional-Integral-Derivative controllers [94], Model-Predictive-Controllers [95, 96, 97] or the development of feedback laws [98, 99, 100, 101]. However in more recent work reinforcement learning has been used *in silico* and found to require less system knowledge and perform favourably under conditions where sampling is difficult [102].

1.4 Reinforcement learning

Reinforcement learning algorithms involve an agent which learns by observing its environment. Time is discretised and the environment can take on any one of an arbitrary number of possible states at each discrete time step. At each time step the agent chooses between an arbitrary number of possible actions, the chosen action then affects the state of the environment and the agent receives a reward depending on the result of the chosen action (Figure 1.2). The training of a reinforcement learning agent is often broken up into *episodes*. An episode is defined as a temporal sequence of states and corresponding actions (generated by the agent interacting with the environment) which progress until a terminal state is reached. The total reward obtained during an episode is called the return.

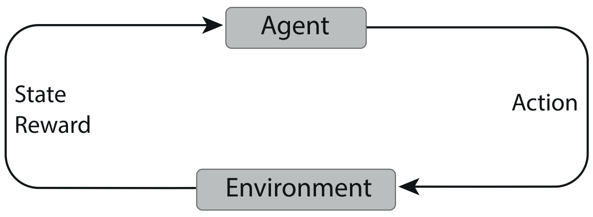


Figure 1.2: The basic reinforcement learning feedback loop. At each timestep the agent chooses an action, which effects its environment. The state of the environment is observed by the agent and a reward is received depending on the favourability of the new state. Adapted from [103]

The reward function defines the agents goal. It maps each environmental state to a scalar reward, communicating the desirability of each state to the agent. Most reinforcement learning algorithms, and the ones used in this thesis, use the action-reward loop in Figure 1.2 to estimate a value function. The value of a state $V(s)$ is defined as the expected return from being in that state. Where the return is defined as the total discounted future reward $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where t is the current time-step, the sum over k is a sum over all future time-steps, r is the reward received at time-step $t+k+1$ and γ is a discount factor. The discount factor dictates how much the prospect of future rewards weigh in on an agent's decision and the agent's goal is to maximise the return. We can also define the value of a state-action pair $Q(s,a)$ in a similar way, as the expected return after taking action a in state s. The functions $V(s)$ and $Q(s,a)$ can be learned by the agent through the rewards it receives at each time step and are the basis of its decision making. The policy $\pi(s)$ is a mapping of states to actions, at each timestep the agent will choose an action $a_t = \pi(s_t)$ and the policy will depend on the agents current value function and how likely the agent is to take an exploratory action. An exploratory action is taken at random with the intention of exploring new regions of state-action space. In contrast, an exploitative action queries the value function and the action that is estimated to have the highest value is chosen. Both value functions are dependent on the policy π used by the agent and so can be written as [103]:

$$V_\pi(s) = E_\pi\{R_t | s_t = s\} \tag{1.1}$$

$$Q_\pi(s,a) = E_\pi\{R_t | s_t = s, a_t = a\} \tag{1.2}$$

1.4.1 The Markov property

An environment is said to have the Markov property if all relevant information required for decision making at a time point t is contained in the state at t (S_t). In other words the historic state transitions are irrelevant, meaning the future is only dependent on the present, not the past. If the state transitions of the environment satisfy the Markov property then selecting an action is known as a Markov decision process. In this case the behaviour can be defined by specifying only [103]:

$$P\{S_{t+1} = S', r_{t+1} = r | S_t, a_t\} \quad \forall S', r, S_t, a_t$$

where S_t is the state at time t, a_t is the action taken by the agent at time t, r_{t+1} is the reward received at time $t+1$ and S' is a possible next state of the system. The Markov property ensures that the next state and reward can be predicted using only the current state. Many of the convergence guarantees of reinforcement learning theory assume Markov decision processes [103].

1.4.2 The Bellman equations

If the Markov property is assumed to hold, then Equation 1.1 can be written as [103]:

$$\begin{aligned} V_\pi(s) &= E_\pi\{r_{t+1} + \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\} \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V_\pi(s')] \end{aligned} \tag{1.3}$$

where $P^a_{ss'} = P\{s_{t+1} = s' | s_t = s, a_t = a\}$ and $R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$. This is the Bellman equation for the policy π. This is a linear system and can be solved by inversion [103]. The objective of reinforcement learning is to learn the optimal policy π_* by finding the solution to the Bellman equation for the optimal policy:

$$\begin{aligned} V_*(s) &= \max_a E_\pi\{r_{t+1} + \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\} \\ &= \max_a \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V_*(s')] \end{aligned} \tag{1.4}$$

This is now non linear and as such cannot be solved directly by inversion. Similarly, we can define the non linear Bellman equation for the optimal policy with respect to state-action values:

$$Q_*(s,a) = \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma \max_{a'} Q_*(s',a')] \tag{1.5}$$

The non linear Bellman equations for V_* and Q_* can be solved approximately by many iterative methods, the main ones being dynamic programming, Monte Carlo and temporal difference methods [103].

1.4.3 Q-learning

Q-learning [104] is a type of temporal difference method which learns a value function over state-action pairs, $Q(s,a)$, where the state and the action spaces are both discrete. The values of each state-action pair are updated at each time step according to the Q-learning update rule:

$$Q(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1},a)) \tag{1.6}$$

where α is the learning rate. The term $\max_a Q(s_{t+1},a)$ gives an estimate of the total future reward obtained by entering state s_{t+1}. The term $(r_t + \gamma \max_a Q(s_{t+1},a))$ is often referred to as the Q-learning target. Q-learning is an off-policy method, meaning that the update rule is not dependent on the policy.

A natural way to store the state-action values is a lookup table with an entry for each state-action pair and this can be updated according to Equation 1.6 at each time step, this is refereed to as tabular Q-learning. Another option is to use a function approximator to learn a state-action value function. When doing reinforcement learning with non-linear function approximators, such neural networks, to represent the Q-function, the learned Q-values can be unstable or even diverge [9]. There are two ways to overcome this problem and methods based on both are used in this thesis. The first way is to reset and completely retrain the neural network as in Fitted Q-learning [105]. This leads to a very data efficient method, although for large networks the need to retrain from scratch each episode can be time prohibitive. The second involves two modifications to the deep Q-learning algorithm that have been shown to solve this instability. When both modification are applied the network is known as a deep Q-network (DQN) [9]. The first is the addition of an experience replay buffer. The buffer stores state transitions experienced by the agent in the form (s_t, a_t, r_t, s_{t+1}). At each time step experience is randomly sampled from the buffer to train the agent. This reduces the temporal correlation between subsequent training observations, a major source of instability. The second modification is the addition of a target Q-network. A target network is a separate neural network used to generate the Q-learning targets used for training. The target network's weights are only updated periodically, much less frequently than the primary Q-network. This reduces the correlation of the Q-learning target to the primary Q-network, the other major source of instability.

Q-learning has been applied to many problems. Video games provide the perfect training environment for such agents as large amounts of data can easily be generated. A key paper in the field introduced DQN to learn how to play 49 Atari games [9]. It was found that the DQN architecture outperformed the best reinforcement learning results at the time in 43 of the games, despite not using any prior knowledge of the games that had been used in the previous results. The DQN agent also performed comparably to professional human players, achieving over 75% of the human high score in more than half the games. However they found that the DQN agent still struggled with games that required long term planning. A natural method to enable the agent to plan over long timeseries is to use a recurrent neural network. A DQN agent using a recurrent network containing a long-short-term-memory (LSTM) cell was recently used to learn how to play DOOM [10]. These results show that DQN agents can be used to learn tasks that are designed to be challenging to humans. Reinforcement learning has also been applied to more practical problems. The control of a multistable system consisting of a periodically kicked rotor in the presence of noise was achieved with Q-learning [106]. It was found that the agent could learn to stabilize the system in a selected steady state even with high noise. Deep reinforcement learning has also been successfully used to optimize chemical reactions [107]. The agent out performed a current optimisation algorithm by using 71% fewer steps in simulation and real reactions. Additionally Fitted Q-learning has been shown to be viable, *in silico*, for the control and optimisation of bioreactors containing multiple strains of bacteria [102]. Tabular Q-learning and many of the early deep Q-learning methods are restricted to discrete state and action spaces. However the flexibility of neural networks enables methods such as Fitted Q-learning, which can learn in continuous state but discrete action spaces [105, 102], and Twin Delayed Deep Deterministic Policy Gradient (T3D) [108], which learns in continuous state and action spaces.

1.5 Aims

This thesis was concerned with the development of new methodologies for the use of distributed systems in synthetic biology, specifically in the areas of biocomputing, industrial bioprocessing and experimental design. The major aims of this thesis were therefore:

1. Develop new design algorithms for the construction of distributed biological computers. These are composed of multiple colonies of bacterial cells positioned on a 2D plate, where the colonies communicate using diffusible molecules and respond with different programmable functions

2. Develop a reinforcement learning algorithm for the control of microbial co-cultures. The co-culture will be composed of two strains of competing bacteria. The effectiveness of reinforcement learning to optimise a bioprocess should be demonstrated

3. Develop a reinforcement learning algorithm for optimal experimental design.

This should enable the model based design of characterisation experiments for systems described by sets of non linear differential equations.

1.6 Thesis outline

The layout of this thesis is as follows:

Chapter 2: I describe the methods that are used throughout the thesis

Chapter 3: I investigate the design of programmable digital functions. I develop a mathematical representation of our system and use this to prove statements about the capabilities of the biological parts and verify these results using simulations. Building on this I use the representation to design a circuit optimisation algorithm to encode arbitrary digital functions with minimal biological complexity.

Chapter 4: I explore the possibility of implementing biological neural networks to encode analogue functions. I develop an algorithm for the design of neural networks constructed from patterns of bacterial colonies and show some potential applications. Furthermore, using a computational model of the biological parts I demonstrate the effectiveness of the design algorithm.

Chapter 5: I apply deep reinforcement learning to the control of microbial communities. I show its effectiveness compared to an industry standard approach its ability to optimise a communities' output with no knowledge of the internal mechanism.

Chapter 6: I explore the applicability of using deep reinforcement learning in designing optimal experiments for understanding biological systems. This results in a general method that could be used to understand many systems across synthetic biology and the rest of science.

1.7 Contribution to publications

The work in this thesis has so far contributed to the following publications.

Neythen J Treloar, Alex JH Fedorec, Brian Ingalls, and Chris P Barnes. Deep reinforcement learning for the control of microbial co-cultures in bioreactors. PLoS computational biology, 16(4):e1007783, 2020.

Behzad D Karkaria, Neythen J Treloar, Chris P Barnes, and Alex JH Fedorec. From microbial communities to distributed computing systems. Frontiers in Bioengineering and Biotechnology, 8:834, 2020.

Chapter 2

Methods

2.1 Finite difference simulation of diffusion

The finite difference method solves a system of differential equations by discretising the area of interest and, when using the forward Euler approximation, steps forward through time to simulate dynamics.

The diffusion equation for the concentration of a diffusible molecule A is given by:

$$\frac{\partial A(\mathbf{r},t)}{\partial t} = D\nabla^2 A(\mathbf{r},t) + S(\mathbf{r},t)$$

where $\mathbf{r}$ is the position, t is time, D is the diffusion coefficient, ∇^2 is the Laplace operator and $S(\mathbf{r},t)$ is a term that accounts for sources of the molecule. To solve this, the domain of interest is split into a finite number of discrete grid points and an approximate solution is calculated for each point on the grid. The finite difference for a forward time, central space approximation for the diffusion equation in two dimensions is:

$$\frac{A_{i,j,k+1} - A_{i,j,k}}{\Delta t} = D(\frac{A_{i-1,j,k} - 2A_{i,j,k} + A_{i+1,j,k}}{\Delta x^2} + \frac{A_{i,j-1,k} - 2A_{i,j,k} + A_{i,j+1,k}}{\Delta y^2}) + S_{i,j,k}$$

where $A_{i,j,k}$ is the concentration of diffusible molecule at spatial coordinates i, j and timestep k, Δx and Δy are the size of the discrete points in the x and y directions and Δt is the timestep between successive iterations. This can be written as:

$$\mathbf{A}_{k+1} = \mathbf{A}_k + \Delta t(H\mathbf{A}_k + \mathbf{S}_k)$$

where $\mathbf{A}_k$ and $\mathbf{S}_k$ are vectors of the current concentration and source terms at all the discretisation points at timestep k respectively and H is a matrix which operates on $\mathbf{A}$ to give the central space difference relations. Using this formula we can start from an initial concentration of diffusible molecule,$\mathbf{A}_0$, and iterate through time. The production of additional molecules by bacterial colonies can be included via the source term S_k. This method can be used to model the production of and response to multiple diffusible communication molecules by bacterial colonies.

2.2 Reinforcement learning algorithms

In this thesis two deep RL algorithms were used, Fitted Q-learning [105], and Twin Delayed Deep Deterministic Policy Gradient (T3D) [108]. Python version 3.6.7 was used for all reinforcement learning code, available at `http://www.python.org`. The neural networks were implemented in Google's TensorFlow (version 2.7.0) [109]. Numpy (version 1.121.4) was used throughout [110].

2.2.1 Neural Fitted Q-learning algorithm

In neural Fitted Q-learning [105] a value function is learned which maps state action pairs to values, $Q(s,a)$. The value function is represented by a neural network. Here, a state transition is defined as the tuple $(s_t, a_t, r_t, s_{t+1}, d)$ specifying, respectively, the system state, action taken and reward received at time t, the state of the system at time $t+1$ and an indicator that is 1 if the episode terminated at time $t+1$ otherwise it is 0. The state can be continuous, but the action is limited to being one of a set of discrete choices. From a sequence of these state transitions a sequence of Q-learning targets is created according to:

$$Q(s_t, a_t)_{target} = r_t + (1-d)\gamma \max_a Q(s_{t+1}, a) \tag{2.1}$$

Here, the term $\max_a Q(s_{t+1}, a)$, where a is an action that can be taken by the agent, gives an estimate of the total future reward obtained by entering state s_{t+1}. This is weighted by γ, the discount factor, which dictates how heavily the possible future rewards weigh in on decisions. The neural network is trained on the set of inputs

$\{(s_t, a_t)\,\forall t\}$ and targets $\{Q(s_t, a_t)_{target}\,\forall t\}$ generated from all training data seen so far (Algorithm 1). In Episodic Fitted Q-learning this was done after each episode (Algorithm 2) while in Online Fitted Q-learning this was done after each update interval (Algorithm 3). I used the Adam optimiser [111] to train the neural network, because of its ability to dynamically adapt the learning rate, which is favourable when implementing reinforcement learning with a neural network [112]. The inputs to the network were scaled to be between 0 and 1 to prevent network instability.

In this thesis I use an ε-greedy policy in which a random action is chosen with probability ε and the action $a = \max_a Q(s_t, a)$ is chosen with probability $1-\varepsilon$. The explore rate was initially set to $\varepsilon = 1$ and decayed as $\varepsilon = 1 - \log_{10}(Ae)$ where e is the episode number, starting at 0, and A is a constant that dictates the rate of decay. A minimum explore rate of $\varepsilon = 0$ was set and was reached by the end of training. ε-greedy is a widely used policy that has been proven effective [10, 9] and is easy to implement.

Algorithm 1 Fitted Q-iteration

```
1: input: $\{(s_t, a_t, r_t, s_{t+1})\ \forall t\}$
2: hyperparameter: N                                      ▷ number of Fitted Q-iterations
3: for iter in 1 to N do
4:     reinitialise Q network
5:     $inputs = \{s_t\ \forall t\}$
6:     $targets = \{r_t + \gamma \max_a Q_{iter}(s_{t+1})\ \forall t\}$
7:     train Q network on $(inputs, targets) \rightarrow Q_{iter+1}$
8: return $Q_N$
```

Algorithm 2 Episodic Fitted Q-learning

```
1: hyperparameter: E                                      ▷ number of episodes
2: hyperparameter: tmax                   ▷ number of timesteps in each episode
3: for episode in 1 to E do
4:     for i in 1 to tmax do
5:         $a = \pi(s_{t,env}, Q_N)$                    ▷ get action based on current policy
6:         $(s_t, a_t, r_t, s_{t+1}) = env.step(a)$  ▷ interact with env and observe transition
7:         $\mathcal{D} \leftarrow \mathcal{D} + (s_t, a_t, r_t, s_{t+1})$          ▷ add transition to memory
8:     $Q_N$ = Fitted_Q_iteration($\mathcal{D}$)                  ▷ update agent's policy
9: return $Q_N$
```

Algorithm 3 Online Fitted Q-learning

```
    input: $envs$                                          ▷ set of environments to learn from
 2: hyperparameter: tmax                                   ▷ number of timesteps
    hyperparameter: update_frequency                       ▷ how frequently to update the policy
 4: for t in 1 to tmax do
        for each env do
 6:         $a = \pi(s_{t,env}, Q_N)$                      ▷ get action based on current policy
            $(s_t, a_t, r_t, s_{t+1}) = env.step(a)$       ▷ interact with env and observe transition
 8:         $\mathcal{D} \leftarrow \mathcal{D} + (s_t, a_t, r_t, s_{t+1})$   ▷ add transition to memory
        if iter%update_frequency = 0 then
10:         $Q_N$ = Fitted_Q_iteration($\mathcal{D}$)      ▷ update agent's policy
    return $Q_N$
```

2.2.2 Twin Delayed Deep Deterministic Policy Gradient

Twin delayed deep deterministic policy gradient (T3D) [108] is an off-policy algorithm for continuous deep reinforcement learning (Algorithm 4). At its core it is based on an older algorithm called deep deterministic policy gradient (DDPG) [113], but introduces a few modifications to improve learning stability. The DDPG algorithm is closely related to Q-learning and can be thought of as Q-learning adapted for continuous action spaces. Like deep Q-learning, DDPG uses a neural network to approximate and learn a Q-function $Q(s,a)$ which maps state, action pairs to a value. In addition to this, DDPG also learns a policy, $a = \pi(s)$, which is represented by a second neural network. The policy network maps states to actions and is trained to choose the action, a, that maximises the value of the state action pair for the given state, s, according to the value network: $a = \text{argmax}_a Q(s,a) \approx Q(s,\pi(s))$.

As before a state transition is defined as the tuple (s,a,r,s_{t+1},d) specifying, respectively, the system state, action taken, and reward received at time t, and the state of the system at time $t+1$ and an indicator that is 1 if the episode terminated at time $t+1$ otherwise it is 0. As the agent learns it stores observed state transitions in a replay buffer, $\mathcal{D}$, which can be though of as it's memory. As is standard in deep reinforcement learning algorithms two tricks are used to increase stability of the learning process in DDPG. Firstly, at each update a random sample, $\mathcal{B}$, of past experience is taken from the replay buffer to reduce the temporal correlation in the updates. Secondly, target networks Q_{targ}, π_{targ} are used to generate the Q-learning targets, the parameters of these networks update slowly to the parameters of Q and π by polyak averaging, $\theta_{targ} = \rho\theta_{targ} + (1-\rho)\theta$. This reduces the dependence of the target on the trained parameters and further increases stability.

There are three further additions to DDPG to get the full T3D algorithm. First, the policy network updates are delayed by updating half as frequently as the Q-network. Second, to address a common failure mode of DDPG in which the policy can exploit incorrect sharp peaks in the Q-function the target policy is smoothed by adding random noise to the target actions,

$$a_{t+1}(s_{t+1}) = \text{clip}(\pi_{targ}(s_{t+1}) + \text{clip}(\xi, -c, c), a_{low}, a_{high}),\ \xi \sim \mathcal{N}(0, \sigma)$$

where c is an upper bound on the absolute value of the noise, a_{low} and a_{high} and lower and upper bounds on the target action respectively and σ is the standard deviation of the noise. This effectively regularises the algorithm. Finally, as all Q-learning methods involve maximising over target actions they are prone to overestimate the Q-function. To reduce this tendency in T3D double Q-learning is used, in which two Q-functions, Q_1 and Q_2, are learned and the one that gives the smaller value is used to calculate the Q-learning target. The full T3D update is as follows. From a sequence of state transitions, $\mathcal{B}$, sampled from the replay buffer a sequence of Q-learning targets, y, is created according to:

$$\begin{gathered} a_{t+1} = \text{clip}(\pi_{targ}(s_{t+1}) + \text{clip}(\xi, -c, c), a_{low}, a_{high}), \quad \xi \sim \mathcal{N}(0, \sigma) \\ y(r, s_{t+1}, d) = r + \gamma(1-d) \min_{i=1,2} Q_{i,targ}(s_{t+1}, a_{t+1}) \ \ \forall \ \ (s, a, r, s_{t+1}, d) \in \mathcal{B} \end{gathered}$$

The term $(1-d)\min_{i=1,2} Q_{i,targ}(s_{t+1}, a_{t+1})$ gives an estimate of the total future reward obtained after entering state s_{t+1}. The networks Q_1 and Q_2 are trained on the set of inputs by regressing to the targets with the following losses.

$$\begin{aligned} L_1(\mathcal{D}) &= \mathop{\mathbb{E}}_{\mathcal{B}}[(Q_1(s,a) - y(r, s_{t+1}, d)^2] \\ L_2(\mathcal{D}) &= \mathop{\mathbb{E}}_{\mathcal{B}}[(Q_2(s,a) - y(r, s_{t+1}, d)^2] \end{aligned}$$

Then, every other update, the policy network is updated by training it to maximise Q_1:

$$\max_{\theta} \mathop{\mathbb{E}}_{\mathcal{B}}[Q_1(s, \pi_\theta(s))] \tag{2.2}$$

Finally, the target networks are updated:

$$\theta_{Q_{1,targ}} = \rho\theta_{Q_{1,targ}} + (1-\rho)\theta_{Q_1}$$
$$\theta_{Q_{2,targ}} = \rho\theta_{Q_{2,targ}} + (1-\rho)\theta_{Q_2}$$
$$\theta_{\pi_{targ}} = \rho\theta_{\pi_{targ}} + (1-\rho)\theta_{\pi}$$

I use an ε-greedy policy adapted to the continuous action space; a random action is uniformly chosen between a_{low} and a_{high} with probability ε and the action $a = \text{clip}(\pi(s) + \mathcal{N}(0, 0.2\varepsilon), a_{low}, a_{high})$ is chosen with probability $1-\varepsilon$. The *explore rate* ε was set to decay exponentially as training progressed. The explore rate was initially set to $\varepsilon = 1$ and decayed as $\varepsilon = 1 - \log_{10}(Ae)$ where e is the episode number, starting at 0, and A is a constant that dictates the rate of decay. A minimum explore rate of $\varepsilon = 0$ was set and was reached by the end of training. The Adam optimiser [111] was used to train the neural networks, because of its ability to dynamically adapt the learning rate, which is favourable when implementing reinforcement learning with a neural network [112]. To prevent network instability all quantities were scaled so that they were approximately in the interval [0,1] before being entered into the neural network.

Algorithm 4 T3D

```
    for episode in 1 to E do
2:      for t in 1 to tmax do
            $a = \pi(s_t)$                                  ▷ get action based on current policy
4:          $(s_t, a_t, r_t, s_{t+1}, d) = env.step(a)$     ▷ interact with env and observe
    transition
            $\mathcal{D} =\leftarrow \mathcal{D} + (s_t, a_t, r_t, s_{t+1}, d)$      ▷ add transition to memory
6:          update_count = 0
            if iter%update_frequency = 0 then
8:              $\mathcal{B} \sim \mathcal{D}$
                $a' = \text{clip}(\pi_{targ}(s') + \text{clip}(\xi, -c, c), a_{low}, a_{high}), \quad \xi \sim \mathcal{N}(0, \sigma)$
10:             $y(r, s', d) = r + \gamma(1-d)\min_{i=1,2} Q_{i,targ}(s', a') \quad \forall \quad (s, a, r, s', d) \in \mathcal{B}$
                Train $Q_1$, $Q_2$ networks on $y(r, s', d)$
12:             if update_count % policy_delay == 0 then
                    Train $\pi$ network to $\max_\theta \mathbb{E}_{\mathcal{B}}[Q_1(s, \pi_\theta(s))]$
14:                 $\theta_{targ} = \rho\theta_{targ} + (1-\rho)\theta$      ▷ update target networks
    return $\pi$                                            ▷ return trained policy
```

2.3 Proportional integral derivative control

A proportional integral derivative (PID) controller [114] is a type of controller that is widely used in industrial control systems. A PID controller attempts to minimise error, $e(t)$, where the error is the difference between a measured variable and its target set point. The overall control input $u(t)$ is calculated as the sum of three terms

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}. \tag{2.3}$$

$K_p e(t)$ is the proportional term, $K_i \int_0^t e(\tau) d\tau$ is the integral term and $K_d \frac{de(t)}{dt}$ is the derivative term. Each of the three terms aims to minimise different aspects of the error. The proportional term punishes the current error between the set point and current measured variable. The integral term takes into account past value of the error by integrating the cumulative historic value of the error. The derivative term is an estimate of the future error based on its current rate of change. The contribution of each term to the control input is controlled by a gain K_p, K_i, and K_d for the proportional, integral and derivative terms respectively. The gains are scalar values which need to be chosen for each application of the PID controller so that it performs well on the given control problem. Tuning a PID controller is the process of finding the gains that give good performance. This can be done by hand or with with automated tools. In this thesis I used MATLAB's PID autotune [115]. Often the derivative term is omitted, as it can be difficult to tune, and the resulting *PI* controller is sufficient for most applications.

2.4 Model predictive control

A model predictive controller (MPC) [116] uses a calibrated model of a system to predict optimal control inputs. In the applications in this thesis the model is a set of differential equations

$$\frac{d\mathbf{X}}{dt} = \mathbf{F}(\mathbf{X}, \theta, \mathbf{u}), \tag{2.4}$$

where $\mathbf{X}$ is a vector of system variables, θ is a vector of parameters and $\mathbf{u}$ is a vector of experimental inputs. Assuming that the parameters are known, the MPC can integrate the model over a predefined time interval to optimise an objective function Θ_D, with respect to the control inputs $\mathbf{u}$. The objective function is a function of the system variables that defines the MPC's control objective, common choices can be to minimise the distance between the system state and a target set point or to maximise some productivity objective, such as the product output of a bioreactor. The time horizon the MPC optimises over is a hyperparameter to be chosen. If the control scenario is composed of N sample and hold intervals the time horizon can be between 1 and N. In this thesis I consider two variants of MPC. The first uses a time interval of 1, and I refer to this as a one step ahead optimiser (OSAO). The OSAO is myopic and only concerned with finding the single input that maximises the objective over the next sample and hold interval. For each interval the OSAO uses the model to predict the experimental input to be applied during the next interval. At interval i the OSAO solves the optimisation problem

$$\mathbf{u}_i = \arg\max_{\mathbf{u}} \Theta_D(\mathbf{u}, \theta, t_i, t_{i+1}). \tag{2.5}$$

The second variant is a controller that optimises over the full timeseries of N intervals, and I refer to this as an MPC. At the beginning of the experiment the MPC uses the model to optimise over the full experiment, choosing every input that will be applied by solving the optimisation problem:

$$[\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_N] = \arg\max_{[\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_N]} \Theta_D(\mathbf{u}, \theta, 0, t_N) \tag{2.6}$$

To solve the optimisation problems for both the OSAO and MPC the non-linear solver IPOPT [117] was called from the CasADi library [118].

Chapter 3

Distributed digital biocomputation through spatial diffusion and engineered bacteria

3.1 Introduction

Engineering biological systems capable of computation has long been a goal of synthetic biology. The first papers in the field engineered fundamental computer parts such as a toggle switch [32], oscillator [33] and auto regulator [34] and since then much progress has been made. A common approach is the construction of computationally capable cells by engineering programmable logic into the gene regulatory networks (GRNs) inside cells. The current state of the art of this approach are automated design tools such as Cello [40, 44], which transcribes a given logic function into its corresponding GRN. This allowed a comprehensive suite of three input digital functions to be constructed. However, each function needs to be constructed from scratch which means extensive genetic engineering. Furthermore, there are practical limitations on the complexity of circuits we can program into a single cell, due to crosstalk between regulatory molecules and metabolic burden. Crosstalk is a problem because components in a cell can't be directly wired together, so we cannot reuse gene regulators without interference. In contrast, metabolic burden becomes a problem when the resource requirements of a GRN in a cell approaches the capacity of the cell's metabolism. Under high metabolic load a cell may exhibit increased mutation rates due to stress responses [119] and a competitive advantage to losing or mutating the programmed circuit [120]. These factors reduce the stability of a programmed GRN and place a practical upper limit on the complexity of the constructed function.

To overcome these limitations on single cells computational communities of microbes have been developed in liquid culture. This allows the division of a complex function into smaller, isolated modules which can be programmed into different populations and integrated using inter-population communication. Two and three input logic gates have been implemented in yeast liquid culture [50], microbial consortia capable of four input digital logic [53] have been built in this way and algorithms for the optimal design of digital functions in liquid communities have been developed [51]. However, like the non-orthogonality requirements of GRNs within single cells, building microbial communities that communicate in liquid culture requires the use of a unique communication molecule for each 'wire' and the avoidance of crosstalk between molecules. This has previously been termed the 'wiring problem' of wet computational circuits [121].

In this theoretical work I show the potential to overcome the limitations associated with single cell computing and distributed communities in liquid culture by producing distributed logic gates based on spatially arranged, communicating bacterial colonies in solid culture. I combine a suite of patterning functions that have been shown to be possible in bacteria [62, 122] with the use of spatial structure as a control parameter to produce designs for modular, easily programmable bacterial computers capable of computing complex digital logic. This design allows a function to be split into parts and the parts can be distributed across multiple bacterial colonies and wired together using a diffusible molecule. This means that the GRNs inside each colony are small and isolated, naturally overcoming the limitations of burden and non-orthogonality. Furthermore, the signal produced by a colony is spatially localised, meaning that other colonies can be insulated from it by placing them outside of the signal range and we can largely avoid the wiring problem associated with liquid culture communities. Early work using colonies on solid media combined bacterial logic gates with spatial signalling by using four different quorum molecules to connect bacterial colonies containing NOR gates [49]. In contrast to this approach, we use a suite of four activation functions and offload much of the information processing into the spatial pattern of colonies. This allows us to implement arbitrary digital functions with only one layer of biological signalling (and only one signalling molecule) and with many fewer bacterial colonies. Another work decomposed digital functions into the sum of spatially separated modules, where each module is a negated sum of IDENTITY and NOT functions of the inputs [54]. In this setup the potential complexity of the resulting circuits is great, with up to 2^{n-1} modules, where n is the number of inputs to the function, which can include any combination of inputs or their negations and the physical implementation requires a complex setup of connected growth chambers. A subsequent paper arranged sender and modulator cells spatially by printing functions in a branched architecture on a piece of paper and digital functions such as a three input parity bit were implemented [55]. This work enables the easy printing of digital functions using stamps and 'cellular ink' and the printed functions were shown to be storable for later use. However the worst case complexity of the branching architecture grows quickly for more complex functions, where the maximum number of branches is 2^{n-1} and each branch can contain up to n different modulators. Furthermore, the size of the library of required cells including sender, modulator, amplifier and reporter cells is quite large at $3n+2$. The use of programmable spatial patterns and the range of patterning functions is unique to our approach and enables us to program arbitrary digital functions with minimal genetic engineering. I use distance between colonies as a design parameter, which effectively means each patterning function can be used as a flexible module that encodes different functions at different positions relative to its inputs. Unlike designing and building a complex GRN into a cell, a new pattern can easily be produced using automated pipetting robots.

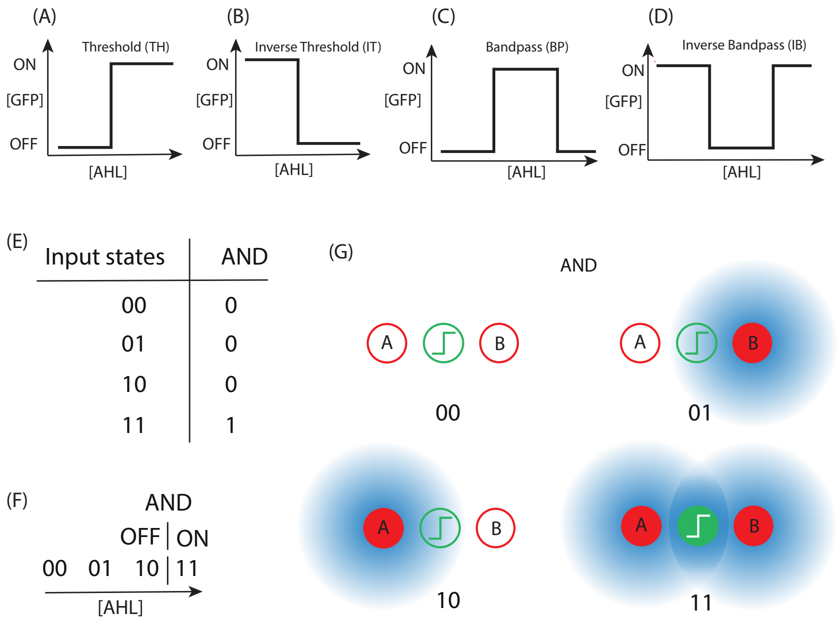
I consider patterns of communicating bacterial colonies growing on a plate where there are two colony types. The first are biosensor or input colonies, these sense environmental inputs and produce the quorum molecule AHL in response. The second type is the output colony, these integrate the signals from the input colonies and produce a fluorescent output that represents the result of the computation. The response of each output colony to diffusible signals is described by one of four activation functions. Because communication by a diffusible molecule is dependent on distance, the bacterial computer can be programmed by specifying the spatial pattern of the colonies. In this chapter I develop a mathematical framework to understand the system and make mathematical statements of its capability. I show that each of our activation functions can be programmed to encode many different two-input logic gates by changing its position relative to the inputs and this allows us to implement any two-input digital function with a single output colony. I then develop an algorithm, the Macchiato algorithm, to optimise the programming of complex digital logic functions distributed over multiple output colonies. A key feature of this algorithm is that it encodes a given logic function in a form that can be implemented in one layer of biological signalling. This minimises the complexity of the spatial patterns and greatly reduces the required genetic engineering. It also guarantees that any function can be implemented using only one communication molecule, minimising the wiring requirements. Finally, I prove the capability to encode any n input logic function with only one level of biological signalling, $n+4$ unique cell types and $n+\left\lceil \frac{n-1}{2} \right\rceil+1$ total colonies.

3.2 The capabilities of a single output colony

In order to make mathematical statements about the capability of the system I developed a simple representation that can capture its key properties and simultaneously express digital logic gates. We define an n input, one output digital function as follows. An input state is represented as the string $\{0,1\}^n$, where n is the number of inputs, a 0 represents the corresponding input being OFF and a 1 represents it being ON. There are 2^n different inputs states for a digital logic function. A one output digital function maps each input state into one of two outputs, ON or OFF, meaning there are 2^{2^n} one output digital functions for a given number of inputs. We represent this output mapping as a boundary dividing the input states into OFF and ON sets. With consideration of the constraints imposed by communicating with a diffusible molecule and how different activation functions partition the input states into OFF and ON sets we can enumerate possible logic gates by finding the allowable permutations of the input states and the boundary.

This will be demonstrated with the two input digital logic gates. Each of the two input logic gates maps each one of the four possible input states $(00, 01, 10, 11)$ to a binary output state (OFF or ON). In our system we represent the input states using the concentration of AHL produced by biosensing strains of bacteria. The input colonies effectively transduce the input state into a concentration field, where the concentration of diffusible molecule at a point will be dependent on the current input state and the position of the point relative to the input colonies. An output colony then maps the concentration of diffusible molecule at its position to an output state using its activation function. The activation function can be one of threshold (TH), inverse threshold (IT), bandpass (BP) or inverse bandpass (IB) (Figure 3.1A,B,C,D respectively) and these are digital approximations of activation functions that have previously been built [62, 122]. Considering the digital approximation of the cellular activation functions simplifies the analysis and has previously been used when constructing digital circuits from cells [49, 54] .

An example of a two input logic gate is AND, an AND gate's output is ON if and only if both of its inputs are ON. A common way of representing a logic gate is using a truth table (Figure 3.1E). I will represent this logic gate as 4 input states (00, 01, 10, 11) divided into two sets (OFF|ON), where input states to the left of the boundary, |, are mapped to OFF and to the right are mapped to ON. This means that the AND gate would be represented as $00, 10, 01|11$ (Figure 3.1F). An example of an instantiation of an AND gate using bacterial colonies is shown in Figure 3.1G, where the distance of the input colonies from the output colonies is tuned such that the concentration of diffusible molecule at the output colony is only sufficient to activate it when both inputs are active. Because the concentration of diffusible molecule at the output colony depends on its position relative to the input colonies we can change



Input states	AND
00	0
01	0
10	0
11	1

Figure 3.1: Digital logic by diffusible communication. A, B, C, D) The activation functions we assume that can be engineered into bacteria, threshold, inverse threshold, bandpass and inverse bandpass. Each activation function responds to the concentration of a diffusible molecule, [AHL] by producing a fluorescent molecule [GFP]. E) The truth table of a two input AND gate. F) The mathematical representation of an AND gate. G) How an AND gate could be physically instantiated using our system, an output colony containing a threshold activation function is distanced from two input colonies such that there is only enough signal to activate the fluorescence response when both input colonies are producing diffusible molecule.

the input-output mapping by moving the output colony. However, there are some constraints on the relative concentrations of AHL which must hold for all positions of the output colony. These are:

$$A_{00} \leq A_{01}$$
$$A_{00} \leq A_{10}$$
$$A_{01} \leq A_{11}$$
$$A_{10} \leq A_{11}$$
$$A_{ij} \geq 0 \, \forall \, i,j \in \{0,1\}$$

where A_{ij} is the concentration of AHL at the output colony when the system's input state is ij. To capture the notion of AHL concentration and these constraints in our representation we impose the convention that from left to right the concentration of diffusible molecule is increasing. e.g. if our input states were ordered $00, 01, 10, 11$ then the concentration of AHL at the output colony would be lowest when the inputs are in the 00 state and highest in the 11 state. The result of the constraints when dealing with two inputs is that by changing the position of the output colony we can swap the ordering of the input states between $00, 01, 10, 11$ and $00, 10, 01, 11$, however the end states, 00 and 11, cannot be moved. To see this imagine a system in the 00 state, with neither of the input colonies activated. For all relative positions of the input colonies and output colony it is true that the concentration of AHL at the output colony must increase or remain the same if the system moves into the 10 or 01 state by turning an input colony on. Equally, the AHL concentration at the output colony must decrease or remain the same if we start in the 11 state and move into 10 or 01.

Next, we can represent the digital approximations to our activation functions as the boundary dividing the input states into OFF and ON sets, for example a threshold activation would be represented as OFF|ON, where all input states to the left of the boundary are mapped to OFF and input states to the right are mapped to ON and the inverse threshold is represented as ON|OFF, with the opposite mapping to the threshold. Figure 3.2 shows how we can change the logic gate encoded by an output colony by moving it with respect to the input colonies (A,B,C) or by using a different activation function (D,E,F). Figure 3.2A shows a threshold output colony adjacent to the A input colony, meaning that the 10 input state will have a much higher AHL concentration than the 01 state and will be sufficient to turn on the output colony. This pattern encodes the digital function A. Figure 3.2B shows the opposite situation, the relative concentrations of the 10 and 01 states have been swapped by moving the output colony adjacent to input B, encoding the function B. Figure 3.2C shows that by moving the output colony so that it is equal distance from each input we can reduce the concentrations of 01 and 10 such that only the 11 input state activates the output colony, resulting in an AND gate. Figure 3.2 (D,E,F) shows the same

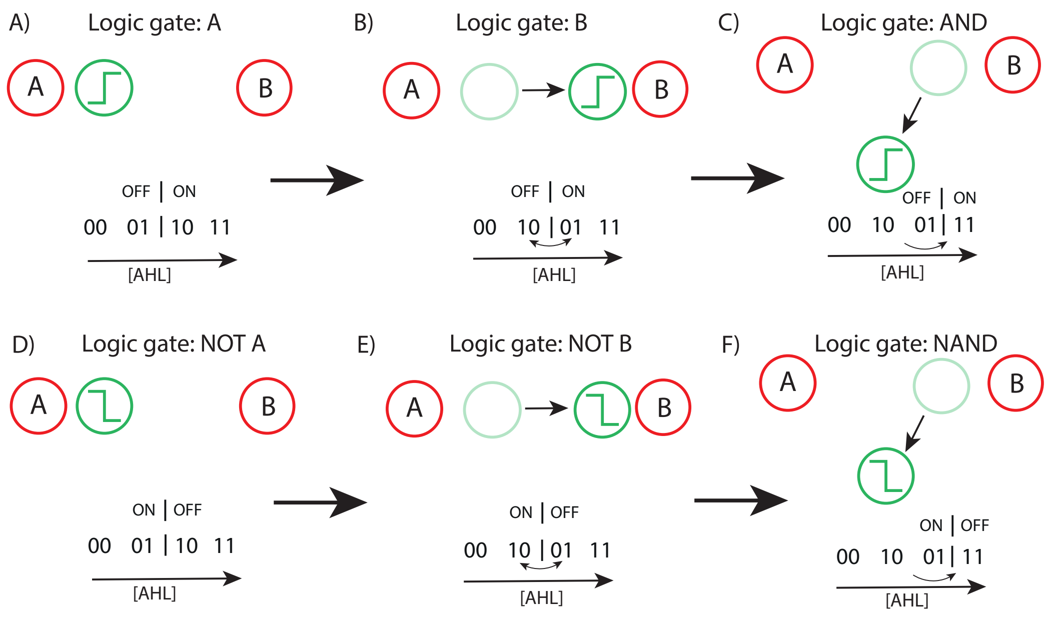


Figure 3.2: Programming logic gates by moving colonies. A) If the output colony is closer to input A then the state 10 will be at a higher AHL concentration and the logic gate encoded will be A. B) if the colony is moved closer to input B then 01 state will have a higher concentration than 10 at the output colony and the logic gate changes to B. C) Moving the colony further away from both colonies will leave the order of the input states unchanged but will move the decision boundary, changing the logic function to AND. (D,E,F) equivalent configurations but using an inverse threshold activation changes the logic gate.

patterns but with an inverse threshold, the inverted output map results in a different set of logic gates for the same spatial patterns. Finally, different permutations of input states and the boundary are distinguishable if and only if they have different output mappings and each distinguishable mapping corresponds to a unique logic gate. For example the mapping $00|01,10,11$ is indistinguishable with $00|10,01,11$, and they therefore both encode the same logic gate with a threshold mapping (OR), but $00,01|10,11$ is distinguishable from $00,10|01,11$ and as such they encode two different logic gates (A and B).

Using this representation we capture the constraints inherent in representing the input states as concentrations of a diffusible molecule and we can naturally represent the activation functions as boundaries partitioning the input states into the two output states. By enumerating the distinguishable permutations of the system, which exactly correspond to the distinct logic gates that can be encoded, we can discover whether this system could allow the encoding of arbitrary, multi-input logic gates. In the following, the two input case, which is simple enough to walk through by hand, will be used to demonstrate the approach and show that with all four activation functions all two-input logic gates can be achieved using only one output colony. Then the results of the computational extension to three inputs will be shown and we shall see that a single output colony is not sufficient for logic gates with more than two inputs.

3.2.1 All two input logic gates can be realised with a single output colony

To enumerate all two input logic gates that are possible in our system, first we take the threshold function (OFF|ON). For each admissible position of the decision boundary we will enumerate all the logic functions that can be obtained by swapping the 01 and 10 input states. There are four positions of the decision boundary we need to consider, where we assume that $|00, 10, 01, 11$ is not possible as at least one input colony must be active to turn the output on. First the $00|10, 01, 11$ position. Here we can see that we obtain one distinct logic gate (OR), as swapping 10 and 01 gives us two indistinguishable mappings. For the next boundary position; $00, 01|10, 11$ is distinct from $00, 10|01, 11$, so we gain two logic gates (A and B). The boundary positions $00, 10, 01|11$ and $00, 10, 01, 11|$ both give us one each (AND and OFF respectively). This means that with the threshold we have a total of 5 logic gates: OR, A, B, AND, OFF (Figure 3.3A). With the inverse threshold we have the same boundary positions, but with the inverse mapping (ON|OFF) and through similar arguments, or through inversion of the 5 threshold logic gates, we gain another 5 gates (NAND, NOT A, NOT B, NOR, ON) (Figure 3.3B). This means that with the threshold and inverse threshold we can achieve 10 of the 16 two input logic gates.

We now investigate the additional logic gates that are possible by adding the bandpass and inverse bandpass to our set of available activation functions. The bandpass can be seen as a threshold function where circular permutations of the states $00, 01, 10, 11$ are allowed (Figure 3.3C). Taking a similar requirement that some AHL must be present for the bandpass to activate, we have the constraint that the 00 input state must be in the lower OFF region of the bandpass, which

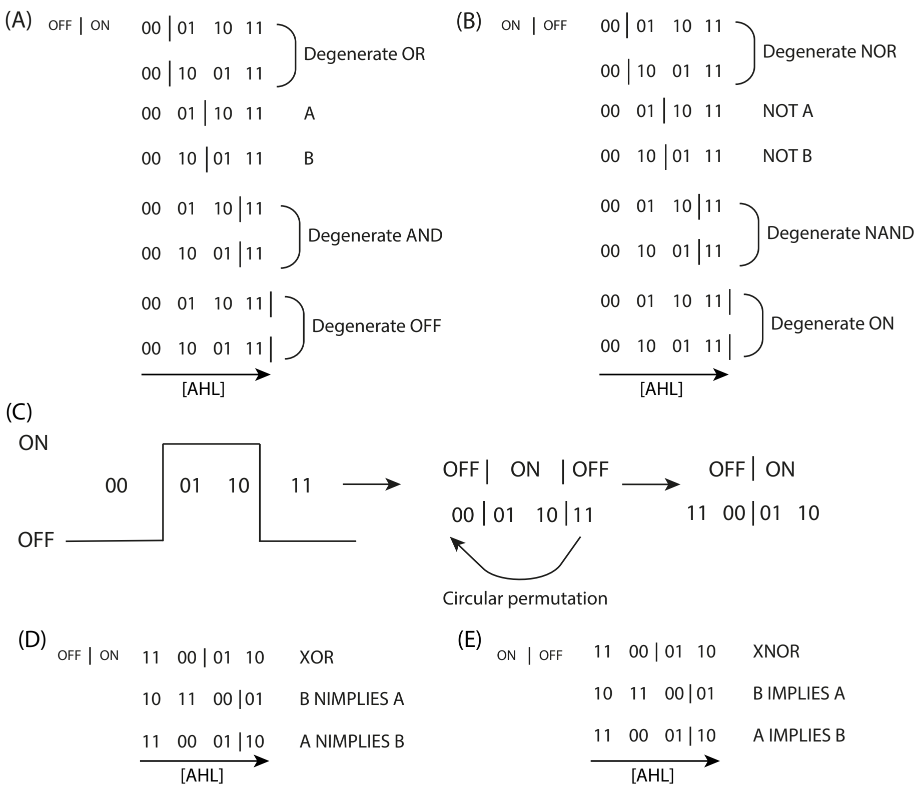


Figure 3.3: Proving two input digital capabilities. The logic gates possible with a threshold (A) and inverse threshold (B), swapping any two state that are within the same set will not change the function, but swapping between sets will. (C) The introduction of the bandpass allows circular permutations of the input states. Additional logic gates are possible with the bandpass (D) and inverse bandpass (E). With the threshold, inverse threshold, bandpass and inverse bandpass we have achieved all 16 two-input logic gates.

imposes the limitation that the 00 input state must be to the left of the boundary. We will examine each circular permutation in turn for the bandpass and place the boundary in each allowable position and enumerate the possible logic gates. The first circular permutation of input states is $00, 10, 01, 11$, which gives us the same logic gates as the threshold. The next circular permutation is $11, 00, 10, 01$ where each of the allowed boundary positions ($11, 00|10, 01$, $11, 00, 10|01$ and $11, 00, 10, 01|$), gives us one (XOR), two (B NIMPLIES A and A NIMPLIES B) and one (OFF) logic gate respectively. Here, the second boundary position gives us two logic gates because swapping 01 and 10 results in two distinguishable mappings. The next circular permutation is $01, 11, 00, 10$ with which the allowable boundary positions ($01, 11, 00|10$ and $01, 11, 00, 10|$) give two (B NIMPLIES A and A NIMPLIES B) and one (OFF) logic gates respectively. The final circular permutation is $10, 01, 11, 00|$, which gives us OFF. We see that we have gained XOR, B NIMPLIES A and A NIMPLIES B in addition to the gates possible with the threshold (Figure 3.3D). The same analysis for the inverse threshold or inverting the bandpass mapping to ON|OFF gives us the remaining logic gates B IMPLIES A, A IMPLIES B and XNOR and we have found the remaining 6 two input logic gates (Figure 3.3E). In this way we have shown that all two-input logic gates are possible using a single output colony.

3.2.2 Computationally enumerating logic gates

The complexity of this analysis increase combinatorially and a computational method for enumerating possible logic gate was developed to allow its application to digital functions with more than two inputs. Here I introduce and demonstrate the method for the two input case.

I define a microstate of the system as a unique permutation of input states and the boundary. I define two microstates as being distinguishable if by observing the output it is possible to distinguish between them, for example $00|01, 10, 11$ and $00|10, 01, 11$ are indistinguishable as they both map 00 to OFF and $01, 10, 11$ to ON, but $00, 01|10, 11$ and $00, 10|01, 11$ are distinguishable as the output mapping of the two microstates is different. If two microstates are distinguishable they are said to belong to two different macrostates, where a macrostate is a set of one or more microstates that are all indistinguishable from each other. In this representation each macrostate of the system is equivalent to a unique logic gate and the number of logic gates possible can be enumerated by finding the number of distinguishable macrostates of the system. An outline of this procedure is as follows. The first step is to enumerate all the possible permutations of input states. Those permutations that violate the constraints imposed by the physics of our spatial computing system are then eliminated. Then, I apply all possible boundary positions that divide the inputs states into OFF and ON sets enabled by the activation functions. This gives the total set of realisable microstates. From this set of microstates I find the number of macrostates (logic gates) by collapsing all sets of indistinguishable microstates. An algorithm which automated these steps was developed (Algorithm 5).

Algorithm 5 Enumeration of logic gates

```
1: permutations = permute(concentration_states)  ▷ all possible orderings of input states
2: permutations = filter(orderings, constraints) ▷ remove any orderings that violate a constraint
3: possible_logic_gates = []
4: for each boundary_position do                 ▷ for each position of the boundary
5:     microstates = apply_boundary(permutations)
6:     macrostates = collapse(microstates) ▷ collapse indistinguishable microstates
7:     possible_logic_gates.append(macrostates)
8: return possible_logic_gates
```

The results of this for two inputs are summarised in Figure 3.4, where each circle represents a possible microstate. Each microstate is one of $5! = 120$ possible permutations of the 5 elements (4 input states and 1 boundary). A physical system with no constraints would be able to realise any microstate. This phase space of 120 different microstates is grouped according to the number of input states mapped to OFF on the x-axis. The microstates are further grouped on the y-axis into macrostates, where each macrostate is a group of indistinguishable microstates that all encode the same logic gate. For example, there are 4 of the total 120 different microstates that encode an XOR gate, all of which have two input states mapped to OFF. To be able to build all two input logic gates at least one microstate corresponding to each of the logic gates must be realisable by the physical system. This phase space was enumerated computationally and the permutations realisable by our system were found by checking against the constraints (Algorithm 5). The realisable permutations are represented by filled in circles, where the colours represent the activation function required to encode the microstate. Note that all microstates possible with the threshold are also possible with the bandpass and all those possible with the inverse threshold are also possible with the inverse bandpass. This shows how the constraints on our system mean that we cannot represent any microstate within the phase space, but we do have sufficient capability to realise any two input logic gate using a single output colony.

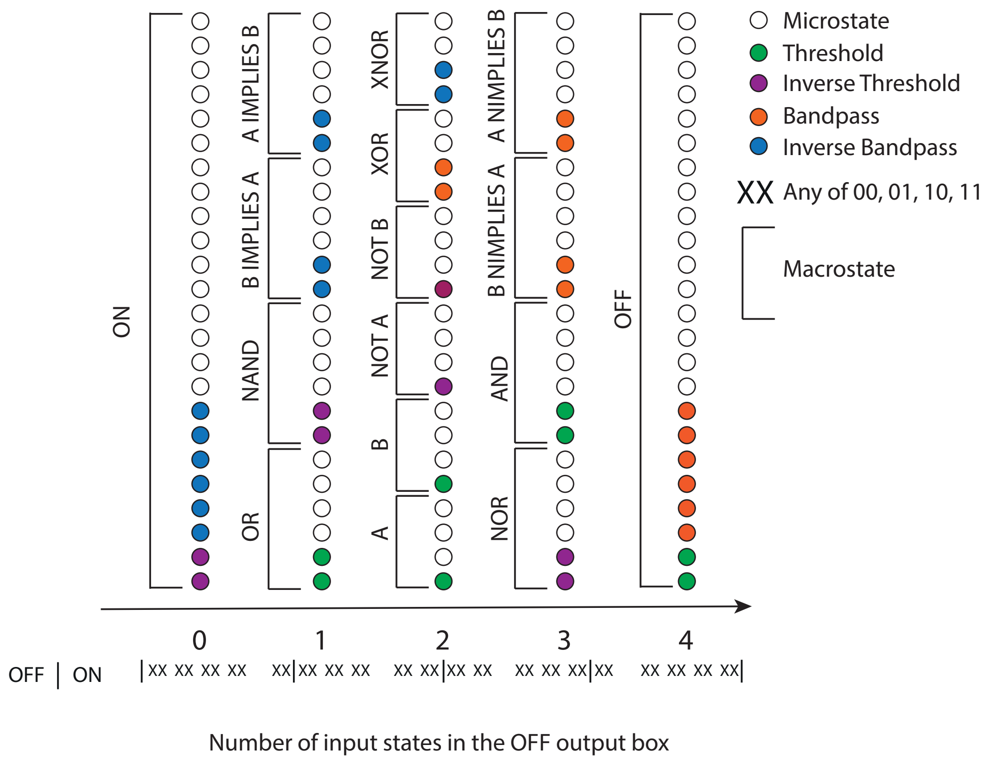


Figure 3.4: Realisable microstates in the phase space of logic gates. Each of the 24 microstates associated with each boundary position and the microstates that are compatible with each macrostate (logic gate). The coloured circles show the microstates that are accessible when using each of the activation functions. For a logic gate to be possible, at least one of it's microstates needs to be accessible.

3.2.3 Simulation verifies the two input mathematical results

These mathematical results were then verified by simulation using a finite difference diffusion model (Section 2.1). Two input colonies, A and B, were simulated for each of the input states 00, 01, 10 and 11, where each input colony acts as a continuous source of AHL if it is active. All simulation units were arbitrary. The four activation functions were applied to the resultant AHL concentration fields for each state and the resulting logic gates were mapped onto each position on the grid. This effectively results in a map of the two-dimensional area around the input colonies, where the colour of each region represents the logic gate that would be encoded by an output colony if it was placed within that region. The results are shown in

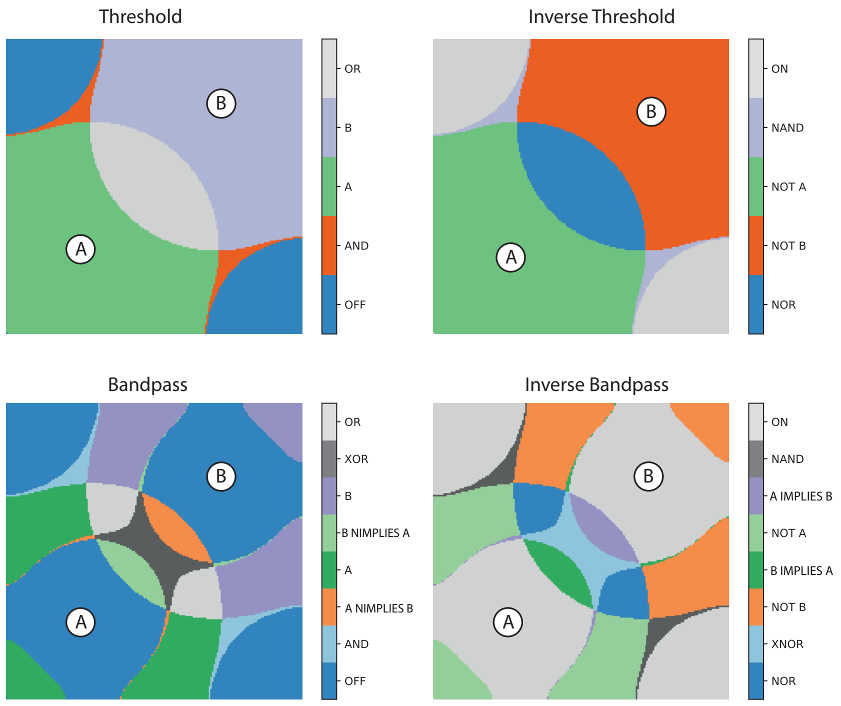


Figure 3.5: Digital logic patterns. Simulation results showing the areas on a grid that correspond to each logic gate for each activation function.

Figure 3.5 and fully corroborate the mathematical predictions. This shows I have successfully abstracted the system, producing a mathematical framework capable of making predictions about the capabilities of the system based on communication via a diffusible molecule. The simulation also clearly shows how we can use spatial patterning to increase the utility of our genetic circuits, each activation function can be used as a module that is capable of many digital functions depending on its position relative to the input colonies.

3.2.4 One output colony cannot realise all three input digital functions

So far in this chapter I have shown that any two-input digital function can be encoded with a single output colony, next the three input capabilities were investigated. There are now have 8 possible input states (000, 001, 010, 011, 100, 101, 110, 111). The set of states with one and two inputs ON will be labelled as $I = \{001, 010, 100\}$ and $II = \{011, 101, 110\}$ respectively. As before we will represent the logic gates by introducing a boundary that partitions the input states that map to ON and input states that map to OFF. For example, three input AND is represented as $000,001,010,011,100,101,110|111$, with a threshold activation (OFF|ON). The use of AHL concentration to represent the input states gives us the following constraints for the three input case:

$$A_{000} \leq A_I, A_{II}, A_{111}$$
$$A_{001} \leq A_{101}, A_{011}$$
$$A_{010} \leq A_{011}, A_{110}$$
$$A_{100} \leq A_{101}, A_{110}$$
$$A_{000}, A_I, A_{II} \leq A_{111}$$

Notice that we can swap the positions of some of the input states in *I* and *II* without violating these constraints. The realisable logic gates were enumerated computationally using Algorithm 5. The result of this was then compared with the number of actual three input logic gates for each boundary position, which is calculated by: $\frac{8!}{(8-i)!i!}$ where i is the position of the boundary or, equivalently, how many input states are mapped to OFF (Table 3.1). This analysis shows that we can do a total of 164 of the 256 three input logic gates and that for functions with three inputs we cannot encode all of the possible digital logic functions with a single output colony.

3.3 The Macchiato algorithm: optimal distributed spatial circuits

To overcome these limitations when building digital logic gates with more than two inputs I developed an algorithm for the optimisation of two-level spatial digital logic. When building electronic digital circuits, two level logic optimisation is used to simplify a logical circuit into the minimal sum of products (OR of ANDs) or product of sums (AND of ORs). The Espresso algorithm is ubiquitous in electronic design automation (EDA) and is a heuristic optimiser that finds an approximate minimal

Boundary position: i (OFF|ON)	$\frac{8!}{(8-i)!i!}$	Realisable gates
0 (| . . . .)	1	1
1 (. | . . . .)	8	8
2 (. . | . . . . . .)	28	22
3 (. . . | . . . . .)	56	32
4 (. . . . | . . . .)	70	38
5 (. . . . . | . . .)	56	32
6 (. . . . . . | . .)	28	22
7 (. . . . . . . | .)	8	8
8 (. . . . . |)	1	1
Total	256	164

Table 3.1: The possible three input logic gates with a single output colony.

form of a given digital function [2]. Despite returning approximate solutions, in practice the Espresso algorithm is extremely effective and its use is widespread in optimising electronic digital functions. The Espresso algorithm attempts to minimise a digital function into the minimal set of OR of ANDs, where each AND term can contain any of the inputs or their negations (Figure 3.6A, top). Taking inspiration from the Espresso algorithm I have developed a novel algorithm, named the Macchiato algorithm, which finds the minimal set of bacterial colonies which encode a given digital function.

The Macchiato algorithm takes the truth table of a multi-input, one output function and returns the approximate minimal set of bandpass, threshold, inverse threshold and inverse bandpass colonies that are required to construct the digital function represented by the truth table. These constitute the analogue of the AND layer and this formulation has increased computational capabilities over a single output colony by distributing different parts of the overall function across multiple colonies. The OR layer can then be done using a threshold colony which activates if any of the previous layer is activated (Figure 3.6A, middle) or it can be done implicitly in the sense that if any colonies are fluorescing we take the output to be ON (Figure 3.6A, bottom). It is for this reason I focus on an analogue of the OR of ANDs (sum of products) form of a digital function, as we gain the flexibility to encode a two level digital function using only one level of biological signalling with a fluorescence output or encode the same function with two levels of biological signalling if a specific biological response is required. The Macchiato algorithm also returns the required connectivity or ’wiring’ specification of the bacterial colonies, which, in principle, can be used to find the requisite pattern of colonies. The algorithm works as follows (Figure 3.6B).

1. A truth table is supplied as input
2. The input states are rearranged to minimise the number of blocks, where a block is defined as a contiguous sequence of ones in the output mapping, while obeying the constraints imposed upon the possible orderings of input states. This is the main part of the algorithm
3. Activation functions are mapped onto the blocks, so that each block is covered by a single output colony. For a block to be covered by an output colony the input states within the block and only those input states must be mapped to ON by the activation function in the output colony. This results in the required minimal set of colonies as well as their connectivity to the inputs and this is returned to the user
4. This output could be used to produce a spatial configuration of colonies that results in the truth table, if any of the green output colonies are activated the output of the function is ON

The Espresso algorithm works by iteratively increasing the size of ‘covers’. A cover is a group of 1s in the output of the digital function that can be represented by one term in its input. The Macchiato algorithm has an analogous concept of blocks, where a block is a group of ones in the output that can be mapped to ON by a single output colony containing one of our four activation functions. The aim of the Macchiato algorithm is to reduce the number of blocks. The Macchiato algorithm is also heuristic, so it is not guaranteed to find the best solution. However, in comparison with an exhaustive, graph search based algorithm found the optimal solution for 100% of the two and three input logic functions, for more complex functions the time and space requirements of the exhaustive search became prohibitive.

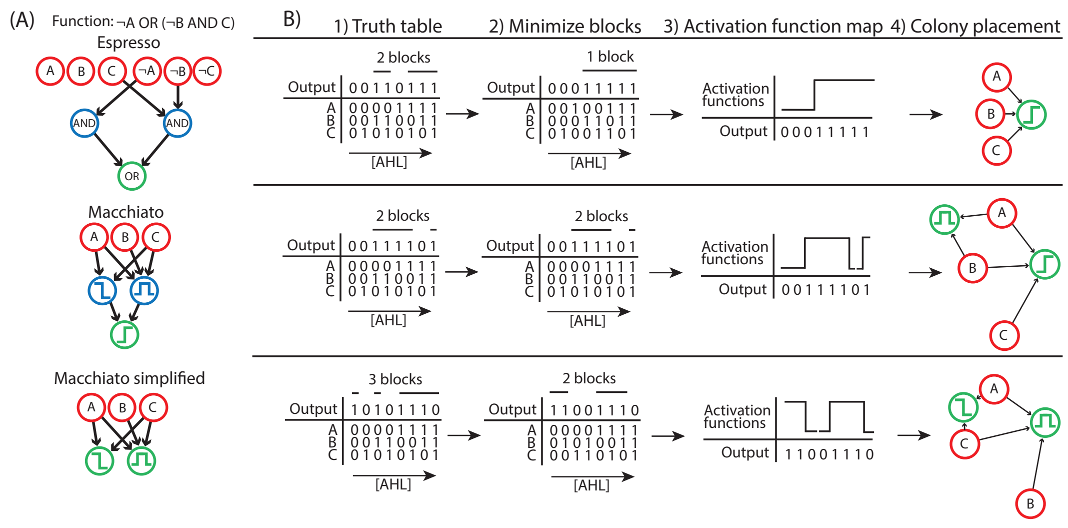


Figure 3.6: The Macchiato algorithm. A) The same digital logic function simplified by the Espresso algorithm (top), Macchiato algorithm with two levels of biological signalling (middle) and Macchiato with one level of biological signalling (bottom). B) The stages of the Macchiato algorithm on three examples, from left to right; 1) a truth table is supplied as input, 2) the input states are rearranged to minimise the number of blocks while obeying the constraints imposed by the system, 3) activation functions are mapped onto the blocks, this is the output return to the user, 4) this output can be used to produce a spatial configuration of nodes that results in the truth table, if any of the green output colonies are activated the output of the function is ON. Top is an example of a function that can be simplified sufficiently that it can be implemented with one colony. Middle cannot be simplified due to the system constraints. Bottom, can be simplified but still requires two colonies.

The main part (2 in Figure 3.6) of the Macchiato algorithm is split into two stages. The first stage is to do a rough optimisation (Algorithm 6) to find the optimal ordering of states assuming no mixing between the different input groups. An input group is defined as a set of input states which all have an equal number of 1s. Because the system constraints permit input states within each input group to be rearranged arbitrarily, we can always simplify an input group that contains both 0s and 1s such that the 1s are all at the low or high AHL concentration end and we can always flip the group so that the 1s are moved to the other end. In this first stage all possible combinations of flips of input groups are searched to find the one with the smallest number of blocks. This acts to give a good initial condition for the second stage. The second stage further minimises the number of blocks by allowing mixing between the input groups (Algorithm 7). The smallest blocks are tried in turn to be eliminated by moving their input states into larger blocks, while obeying the constraints imposed by representing the input states with a diffusible molecule. When no further improvements can be made the algorithm exits. Then the blocks are covered by mapping activation functions onto this set of minimal blocks (3 in Figure 3.6). This works by analysing the minimal set of blocks and applying activation functions such that all blocks are covered by at least one activation function and no zeros in the output are covered (Algorithm 8). Finally, the output, composed of the required activation functions and the input states that need to be in their respective ON and OFF sets, is returned to the user.

Three example applications of the Macchiato algorithm to three input functions are shown in Figure 3.6B. Figure 3.6B-top shows an example of one of the three input functions that can be implemented using a single colony, the input truth table contains two blocks of ones, however by changing the distances between colonies we can group these into a single block and this means that only one output colony is required to implement the function. Figure 3.6B-middle shows an example where the constraints of our system prevent any simplification of the input truth table, therefore there are two blocks that need to be covered and the function is distributed over two output colonies. Figure 3.6B-bottom is an example of a truth table that can be simplified from three blocks to two. Therefore this function is also distributed over two output colonies.

Algorithm 6 Rough Optimisation

```
1: input: truth_table
2: simplified_table = simplify(truth_table)
3: min_blocks = count_blocks(simplified_table)
4: flip_combinations = cartesian_product([False, True], repeat = simplified_table.n_flippable)
5: for combination in flip_combinations do
6:     if count_blocks(combination) < min_blocks then
7:         min_blocks = count_blocks(combination)
8:         best_table = combination
9: initial_truth_table = truth_table(combination)
10: return initial_truth_table
```

Algorithm 7 Minimise Blocks

```
1: input: initial_truth_table
2: current_table = initial_truth_table
3: finished = False
4: while not finished do
5:     blocks = get_blocks(current_table)
6:     block_sort = arg_sort(blocks)    ▷ sort blocks by size, smallest to largest
7:     finished = True   ▷ exit if we are unable to make any further simplifications
8:     for block in block_sort do
9:         test_table = current_table
10:        move_ones(block)    ▷ try and move ones in the output to other blocks
11:        if block is empty then
12:            current_table = test_table
13:            finished = False    ▷ don't exit as we have made a simplification
14:            break    ▷ start again with the simplified truth table
15: return current_table
```

Algorithm 8 Map activation functions

```
1: input: blocks_table
2: if blocks == [1,0,1] then    ▷ This is the only situation where IB is required
3:     return [IBP]
4: activations = []
5: pos = 0
6: if blocks[0:2] == [1,0] then    ▷ Inverse threshold only allowed at beginning
7:     activations.append(IT)
8:     pos += 1
9: while pos < len(blocks) - 1 do
10:    if blocks[pos:pos+3] == [0,1,0] then
11:        activations.append(BP)
12:        pos += 2
13: if blocks[-2:0] == [0,1] then    ▷ Threshold only allowed at end
14:    activations.append(TH)
15: return activations
```

3.3.1 The relative importance of the activation functions for the Macchiato algorithm

The output of the Macchiato algorithm is an OR of the outputs of a set of colonies. This means it is important that input states that are required to be mapped to OFF are not mapped to ON by any of the output colonies. However, an input state that is required to be mapped to ON can be mapped to OFF by multiple output colonies and as long as it is covered by another output colony this won't affect the functions output. Figure 3.7A demonstrates that if an input state that is required to be OFF is mapped to ON by any output colony then the output will differ from the required mapping (left). This makes the bandpass the most important activation function as it can select a block and map those and only those input states within the block to ON (right). The threshold and inverse threshold can only be used if a block of input states that need to be mapped to ON contain ALL 1 and or ALL 0 respectively i.e. the minimum and maximum AHL concentration. As we assume that a non-zero amount of AHL is required to activate the bandpass, the inverse threshold is required to map the state ALL 0 to ON, whereas the input state ALL 1 could be covered by the bandpass if required. The inverse bandpass is only required where there are exactly two blocks of input states that map to ON, one of which includes ALL 0 and the other includes ALL 1 and is effectively a convenience that will simplify a number of functions. This means that the activation functions are ranked bandpass (BP), inverse threshold (IT), threshold (TH), inverse bandpass (IB) in order of most important to least important. This intuition was demonstrated and the capability of our approach to build the three input logic gates was analysed. The Macchiato algorithm was applied to each three input gate using five different sets of available activation functions. For each set of activation functions the number of bacterial colonies required to build each gate was found (Table 3.2). With access to all four activation functions the Macchiato algorithm can achieve all three input logic gates using one or two output colonies. This shows that two-level logic optimisation has overcome the limitations using a single output colony in implementing three input digital functions. If we remove the inverse bandpass from the available activation

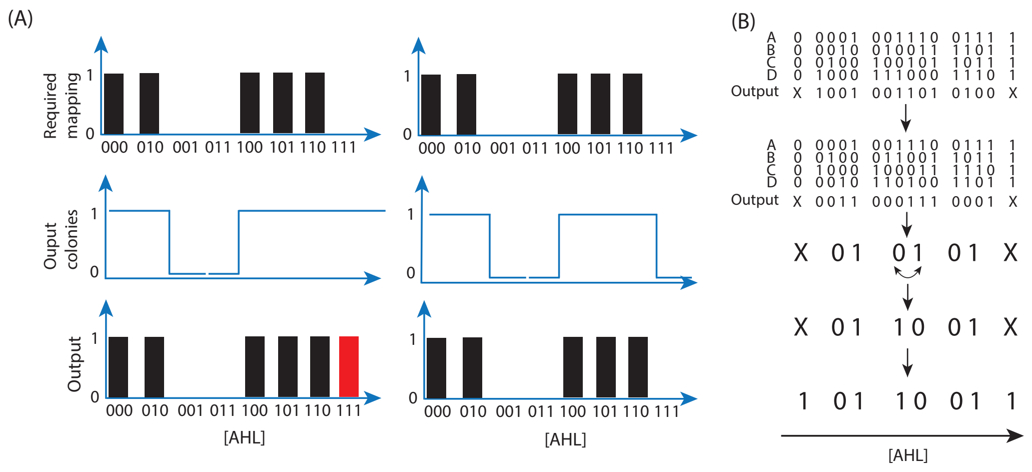


Figure 3.7: A) If an input state is falsely mapped to ON then the output will differ from the required mapping (left), which makes the bandpass the most important activation function (right). B) We can prove the upper bound on the number of output colonies required, from top to bottom; 1) for any given truth table we can rearrange the input states with the same number of activated inputs such that in the worst case each number of activated inputs contains a block of 0s and 1s (2). 3) in the worst case no further simplification can be done and the problem reduces to finding the upper bound number of blocks upon inverting these (4). 5) This is an example of the most pathological case for four inputs.

functions we retain the capability to do all the three input logic gates, however a larger proportion now require two output colonies as expected. If we remove the threshold or inverse threshold the Macchiato algorithm finds 64 logic gates that are now not possible. However, note that a bandpass could, in theory, be used in place of the threshold to map the highest AHL concentration states to ON. If we remove the bandpass activation we are unable to construct the majority of the three input logic gates, demonstrating the importance of the bandpass. This shows that experimental effort should be prioritised to developing first the bandpass and inverse threshold and if possible a threshold, but the inverse bandpass is of much less importance.

3.3.2 Upper bound for the number of output colonies

Here I show that the Macchiato algorithm can be used to encode any given digital function with an arbitrary number of inputs and that the upper bound for the required

		Number of colonies	
Activation functions	Not possible	1	2
TH, IT, BP, IB	0	164	92
TH, IT, BP	0	101	155
TH, BP, IB	64	144	46
IT, BP, IB	64	144	46
TH, IT, IB	155	99	0

Table 3.2: Number of colonies required to implement three input logic gates for different combinations of activation functions

number of output colonies is

$$C_o \leq \left\lceil \frac{n-1}{2} \right\rceil + 1$$

where C_o is the number of output colonies and n is the number of inputs. To show this take a digital function, an example is shown with four inputs in Figure 3.7B, and group the input states into input groups according to the number of activated inputs (Figure 3.7B, top). Due to the constraints of the system we know that the two end states ALL 0 and ALL 1 must be fixed at the end points. We also know that we can change the order within each group of input states arbitrarily without violating any constraints on the concentration of AHL. This means that for any function we can order each group such that it's output is a block of zeros followed by a block of ones (Figure 3.7B second one down) and in the worst case each group will have at least one 0 and one 1 in it's output. Note that in most circumstances we could further simplify this as we are allowed to mix some states between groups, however in the worst case this will not be possible and so we can ignore this complexity. Now, to simplify, we can represent each of the $n-1$ middle groups as a 0 and a 1, representing its outputs that map to 0 and to 1 (Figure 3.7B, middle) and the problem reduces to flipping the input groups such that the number of blocks of ones is minimised (Figure 3.7B, second from bottom). It is apparent that when there are an odd number of inputs (and an even number of middle groups) the number of blocks is $\frac{n-1}{2}$ and when there are an even number of inputs (and an odd number of middle groups) there are $\frac{n-1}{2}+0.5$. Both these facts can be combined to give the $\left\lceil \frac{n-1}{2} \right\rceil$

term, where $\lceil x \rceil$ is the ceiling of x. The only additional complexity comes from the output states of the ALL 0 and ALL 1 input states; there are four combinations 00, 01, 10, 11. Appending a zero onto either end of the sequence can never increase the number of blocks, therefore the state 11 represents the worst case and is the only one that we need to consider (Figure 3.7B, bottom). If the middle groups adjacent to the output (the outer middle groups) are both 1s it is apparent that no additional blocks can be added. If one outer middle group are is 0 and the other is 1, it is apparent that this will increase the number of blocks by one. If both outer middle groups are 0 then at first glance it appears that two blocks will be added, however you can simply invert each of the middle groups, at worst incurring a cost of one additional block. This means that the addition of the end states at worse adds one additional block, and, remembering that each block requires a single output colony to be covered, we have derived our formula.

This result was verified for the 1, 2, 3 and 4 input cases computationally. For each given number of inputs every possible logic gate was enumerated. For each logic gate the input states were grouped such that each group of input states had a block of zeros then a block of ones in the output, while ensuring that the constraints on AHL concentration were being obeyed. Following this, every different configuration possible by flipping these blocks of zeros and ones was generated and the minimum number of blocks possible by flipping was found. For each number of inputs the worst case number of blocks agrees with the derived formula. Verification of the 5 input case was attempted, but the memory requirements were found to be prohibitive.

The relative complexity of our approach was assessed by comparison with two other implementations of spatially distributed biocomputers [54, 55]. The first comparison is with an approach based on the separation of cells into a set of connected growth chambers [54]. The maximum number of independent growth chambers, named modules, was used for the comparison. A subsequent paper implemented similar circuits on branched 2D circuits on pieces of paper [55] and the maximum number of branches was used for comparison. These two measures were compared with the maximum total number of colonies required for any given

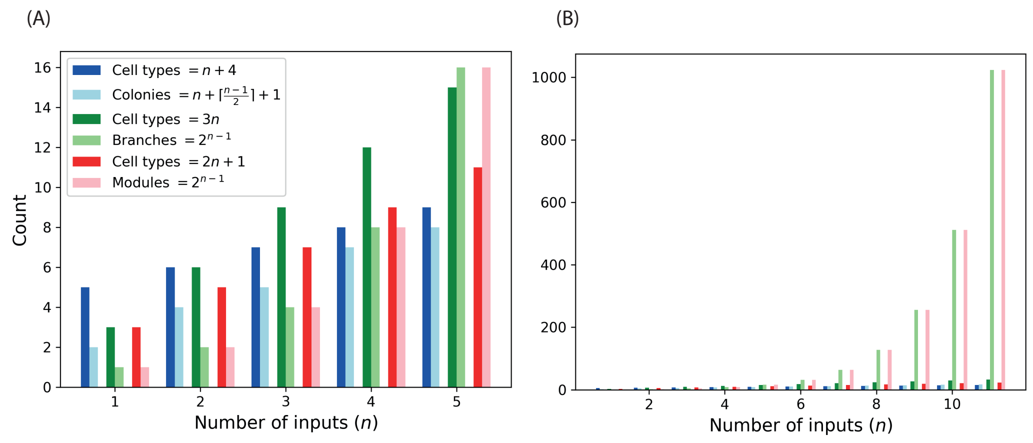


Figure 3.8: A) Comparison of how the number of required cell types and colonies for the Macchiato algorithm scales with the number of inputs (blue) with the number of required cell types and similar measures of complexity for two previous approaches (green [55] and red [54]) to spatially distributed biocomputing. B) How this scales for larger numbers of inputs.

function designed with the Macchiato algorithm. This is equal to the number of input colonies plus the maximum number of output colonies $C_{tot} = n + \left\lceil \frac{n-1}{2} \right\rceil + 1$. The number of unique cell types that need to be constructed was compared for all three approaches. For the Macchiato algorithm this is n input cell types plus 4 patterning functions. It is shown in Figure 3.8 that as the number of inputs increases the complexity of the Macchiato algorithm scales favourably in terms of both the amount of genetic engineering that needs to be done to construct the requisite cells and the resulting complexity of the spatial circuits. This means that the distributed circuits considered here are capable of encoding complex digital functions with less biological complexity than the current state of the art.

3.4 Discussion

In this chapter I have shown that bacterial colonies programmed with realistically achievable patterning functions can be spatially wired together using diffusible molecules to create programmable digital computers. I have proven that a single computing colony is capable of being programmed with any two-input digital logic gate by changing its activation function or its position relative to the inputs. I have then shown that additional complexity is needed to allow the implementation of all three input logic gates. An extension of the method that is capable of distributed digital logic circuits was proposed and, inspired by two level electronic digital logic optimisation, an algorithm was developed for an analogous spatial digital optimisation procedure to find the minimal set of bacterial colonies required to implement a given digital function. The results of this algorithm are analysed to find the requirements on the activation functions and their relative importance, which can be used to prioritise wetlab work. Furthermore, it is shown that any function with an arbitrary number of inputs can be encoded and an upper bound for the number of output colonies required with respect to the number of inputs is found to be $C_o \leq \left\lceil \frac{n-1}{2} \right\rceil + 1$. This means that including the input colonies the total number of colonies required to implement a given function is bounded by $C_{tot} \leq n + \left\lceil \frac{n-1}{2} \right\rceil + 1$. In comparison with previous, similar approaches [49, 54, 55] our system allows the construction of more complex digital functions with less biological complexity (Figure 3.8). This work is based on the use of genetic circuits that already exist [62, 122] and existing lab protocols make automated production of patterns of bacterial colonies easy. This is combined with the modular design that requires at most the genetic engineering of n biosensor strains and 4 patterning functions which can subsequently be reused and programmed into arbitrary digital functions with minimal effort.

To illustrate the approach we showed digital activation functions that transition between the OFF and ON states as a step function. However, the assumption we make about the activation functions is much weaker than the requirement for a digital step function. We only require that each function is able to partition the input space into OFF and ON regions in the required manner. For example, to build an OR gate with the threshold $(00|01,10,11)$, a digital transition from OFF to ON in the region between 00 and 01 is not required. What is needed is that at the concentration of AHL in the 00 state the threshold is OFF and at the $01,10$ and 11 states it has turned ON. The increase of activation in the concentrations between 00 and 01 can be gradual and it will not affect our design. However, in practice activation functions with more step like responses will likely make the construction of functions easier and future work could include the optimisation of activation functions for this purpose. Another consideration is that ideally the output pattern of a constructed function would achieve steady state and after an initial calculation time could be read out at any subsequent time. There are ways that could be investigated the achieve this including expressing AHL degradase enzymes in bacteria [123]. However, during the initial implementation of the method a measurement time could be set at which results are automatically imaged, removing the need for a steady state pattern.

The work in this chapter represents the theoretical foundation required to implement digital functions in patterns of bacterial colonies. However, to implement these designs the minimal set of colonies and the wiring specifications returned by the Macchiato algorithm will have to be translated into a spatial pattern of colonies. This can be done in future work by calibrating a spatial reaction diffusion model to characterisation data of a system. The work in this chapter is system agnostic and the Macchiato algorithm can be applied to any system based on communication using diffusible molecules. However, different systems will require different reaction diffusion models and these models represent the bridge between theory and eventual wetlab implementation. Once a calibrated reaction diffusion model has been obtained the spatial pattern that results in the required connectivity between the minimal set of output colonies and the input colonies can be inferred and subsequently implemented in the lab. This would represent a fully realised implementation pathway from abstract digital logic functions to computing using patterns of bacterial colonies. A further practical consideration is the degrees of freedom available to place colonies in 2D space. This method is based upon the use of spatial proximity as a wiring parameter and, as the distance between each pair of colonies in a multi-colony pattern cannot be varied independently, there will be an upper limit to the complexity of the functions we can build before the wiring requirements become too great. Future work could be done to find the point at which this becomes a problem and if solutions are required. Potential solutions could include the extension to 3D patterns, which would increase the degrees of freedom in colony placement, or the 3D printing of custom made plates incorporating channels that connect and insulate pairs of colonies, which would be cheap and fast using a desktop 3D printer.

In summary I have developed a mathematical representation and design algorithm for distributed digital spatial computing using bacterial colonies. These methods can be used to advance the state of the art in distributed computing in synthetic biology. Furthermore, the eventual implementation of these designs can easily be automated using pipetting robots, paving the way for practical applications of distributed biocomputing such as medical diagnosis. Overall this work has opened an exciting avenue of research for both future lab work and potential real world applications.

Chapter 4

Distributed analogue biocomputation through spatial diffusion and engineered bacteria

4.1 Introduction

In this chapter I investigate the potential to do spatial analogue computing. In contrast to the digital functions considered in the Chapter 3, the inputs and outputs of an analogue function are not restricted to the values 0 and 1, meaning that any continuous value can be taken. The genetic reaction networks (GRNs) within a cell and the quorum molecules used to communicate between cells operate in the domain of concentration, which is effectively a continuous value unless a molecule is at extremely low levels. For this reason, analogue computing might be a more natural way of exploiting the processes inside cells and there have been results that support this. Analogue computing is more efficient, in terms of the rate of ATP consumption and the number of protein molecules required, for doing addition with a genetic circuit at the ranges of precision that are metabolically feasible in single cells [45]. This is due to the mathematical dependence of precision on ATP consumption and number of protein molecules differing for analogue and digital genetic circuits [45]. Additionally, it has been shown that building the equivalent circuit using analogue logic can require orders of magnitude fewer genetic parts [46, 47]. However, analogue processes are much more susceptible to noise, which presents additional challenges to engineering them within synthetic biology. Minimising noise within synthetic constructs and incorporating noise resistance into design methods will likely be a big part of the future of biological analogue computing.

Possibly the most widespread method of implementing analogue functions are artificial neural networks (ANNs). The use of ANNs has driven the recent advancements in the field of deep learning and some work has been done to implement ANNs within synthetic biology. Perceptrons, the building blocks of artificial neural networks, have been implemented using enzymes that transduce different inputs into a common output molecule, benzoate, and a synthetic actuator circuit that sensed benzoate [48]. This was used to build a cell based adder and cell free metabolic perceptrons in which enzyme concentrations acted as weight. Additionally, single layer artificial neural networks have been implemented in liquid culture within cells, where each neuron is a bacterial population and communication is done using diffusible molecules [124]. However, in this work the weights and bias values are adjusted by changing the promoters in the GRNs, meaning that extensive genetic engineering is required to build a new network. In this chapter I propose a method for building bacterial neural networks (BNNs) into spatial patterns of bacterial colonies, where colonies communicate using diffusible molecules. Because the strength of a diffusible signal is a function of distance this would allow us to easily tune the weights between neurons by changing the distance between them.

In this chapter I leverage the similarities between a bacterial colony responding to sources of diffusible molecules and a neuron in a neural network to model communication between bacterial colonies growing on a two dimensional plate as the forward pass of a neural network. This model allows the rapid simulation of the forward pass of a BNN which enables the use of complex design algorithms that require thousands of simulations. I found that backpropagation, the primary method of training ANNs, was unsuitable for our purposes as it would get stuck in local minima when attempting to train networks of the sizes we are interested in. Therefore, I developed an evolutionary algorithm for the design of BNNs. This was used to train BNNs, *in silico*, to perform a wide range of non-linear classification tasks, including a classical example from machine learning, the XNOR function and a medical diagnosis problem to diagnose irritable bowel disease (IBD) and irritable bowel syndrome (IBS). A finite difference model was used to verify one of these designs, which showed that the feed forward model was able to make predictions that held true in a more realistic reaction diffusion situation.

4.2 Modelling bacterial communication networks as artificial neural networks

The most common type of artificial neuron, and those most widely used for deep learning perform a weighted sum of a vector of n inputs, $x_1, x_2 ... x_n$ and a bias term b. A non-linear activation function, σ, is then applied to the result of the sum to calculate the neuron's output (Figure 4.1A). An ANN can be created by connecting multiple neurons together, such that the output of one neuron is passed on as the input of all the neurons it is connected to. In a feed forward network the neurons are arranged in layers, where each neuron receives inputs from the neurons in the previous layer and send its outputs on to the neurons in the next layer. Figure 4.1B shows an example of a simple feed forward network with a single hidden layer. The strength of connection between each neuron is dictated by a weight and training a neural network is the process of finding the set of weights that best performs some defined objective. In the field of deep learning, training is usually done by the process of backpropagation, but other methods such as evolutionary algorithms can be used [125].

We observe that the response of a colony of bacteria to multiple sources of a diffusible molecule greatly resembles an artificial neuron. Consider the situation with two sources of a diffusible molecule at different positions (Figure 4.1C). Each source will produce a concentration field of the molecule which decays as distance from the source increases. Therefore we can use the distance from a source as a design parameter to tune the weight between the source and the colony, the closer the colony is to the source the higher the weight. As diffusion is governed by the

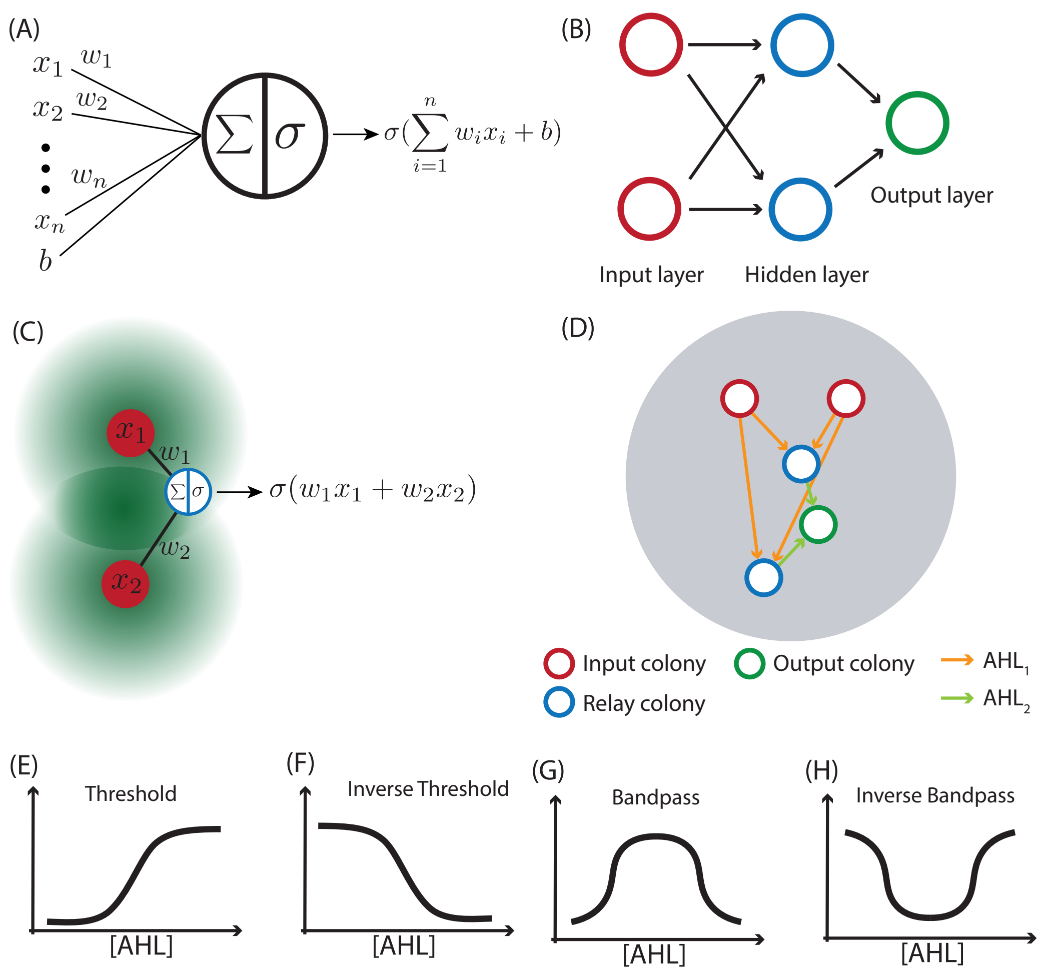


Figure 4.1: Bacterial neurons and neural networks. Diagram of an artificial neuron (A) and how these can be connected into an artificial neural network (B). Diagram of a bacterial neuron (C) and how these can be connected into a bacterial neural network (D). The activation functions we assume that can be engineered into bacteria, where [AHL] is the concentration of the communication molecule. (E) threshold, (F) inverse threshold, (G) bandpass, (H) inverse bandpass.

dynamics of the random motion of independent molecules, in the region where the fields overlap the concentration will be equal to the sum of the contributions from each source. If we consider a bacterial colony placed at some position relative to the two sources the concentration at the colony will be a weighted sum of each of the inputs, where the weight is a function of the distance of the colony from the respective input. We refer to a bacterial colony used in this manner as a bacterial neuron. Like the networks of artificial neurons that make up an ANN, we can create a BNN by connecting multiple bacterial neurons (Figure 4.1D). Bacterial neurons are connected together by positioning them in proximity to each other and the weight between two neurons can be tuned by changing the distance between them. Finally, the response functions that can be engineered into the colonies (Figure 4.1 E-H) are non-linear, a key characteristic of the activation functions used in an ANN. To create the feedforward architecture we can use three distinct types of bacterial neurons. The first type are the input neurons which will contain engineered biosensors that sense environmental inputs and respond with an AHL, these will constitute the input layer of the network. The hidden layers of the network will be instantiated with relay neurons which sense one AHL from the previous layer and respond with a second, orthogonal AHL. We can use relay neurons that sense and respond with different AHLs to build multiple hidden layers. Lastly, we have the output neurons, which sense an AHL and respond with measurable fluorescence and this represents the output layer of the network.

Using these similarities the spatial communication between bacterial colonies can be modelled as a neural network, where each bacterial colony is a neuron and the strength of communication between each pair of colonies via diffusible molecules is represented by a weight.

4.3 Training bacterial neural networks using an evolutionary algorithm

The predominant method of training ANNs is the gradient based method backpropagation [3]. Backpropagation aims to train the weights of a neural network such that a given optimisation objective is achieved. Often this objective is to minimise a loss function, which quantifies the error of the neural network across a training data set. Backpropagation is a local optimisation method which repeatedly adjusts the weights of the network in the direction which would act to reduce the loss over the given data. Because it is a local optimisation algorithm it is not guaranteed to find the global optimal set of weights, in fact it is highly unlikely to do so. However, in practice, it excels at finding a good local optima for complex neural networks, in which the given task is performed well despite not being the globally optimal solution. The, perhaps surprising, effectiveness of back propagation underpins much of the advancement of deep learning. When training small networks of the kind we are interested in here, however, it can be prone to converging to bad local optima, which do not perform the task well. An initial investigation for the use of backpropagation for our application found this to be a problem. Evolutionary algorithms present an alternative which are less susceptible to converging to local optima and have been successfully applied to the training of ANNs [125]. Furthermore, using an evolutionary algorithm makes both the direct optimisation over positions of bacterial neurons on a two dimensional plane and the enforcing of constraints on the available area and minimum distance between neurons trivial.

4.3.1 Constraints

There are some constraints we need to consider that are present in BNNs but not ANNs. First, unlike ANNs the weights in our spatial networks are no longer independent of each other as we set them by moving a colony in two dimensional space. This constraint means that simple networks will be desirable as there will be less dependency among the weights. Second, weights are represented by concentration and can not be negative. This imposes a restriction on the functions we can build by constraining the networks to use non-negative weights. However, this can be at least partially alleviated by the availability of more complex activation functions such as the inverse threshold, bandpass and inverse bandpass or with the use of bacteria that degrade AHL [123]. Lastly, for the results in this chapter we have no bias terms, $\mathbf{b}$, but this could be rectified simply with colonies that constitutively produce AHL.

4.3.2 Modelling bacterial communication as the forward pass of a neural network

I use the similarities between BNNs and ANNs described to approximate the system of communicating bacterial colonies as a neural network and drastically reduce the computation time required to simulate the forward pass compared to using a full finite difference model. To calculate the forward pass of a feedforward ANN with m layers we start with the input layer, $l = 0$, and calculate the activations of each following layer in turn, until we reach the output, $l = m - 1$:

$$\mathbf{a}^{(l+1)} = \sigma^{(l+1)}(\mathbf{W}^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l)}),$$

where $\mathbf{a}^{(l)}$ is the vector of activations of layer l, $\sigma^{(l)}$ is a vector function of the non-linear activation functions of the neurons in layer l, $\mathbf{W}^{(l)}$ is the matrix of weights connecting the output of each neuron in layer l to the input of each neuron in layer $l+1$ and $\mathbf{b}^{(l)}$ is the vector of bias terms associated with layer l. To express the network of communicating bacterial colonies in the same way we first need to make a few simplifying assumptions. The main assumption is that we consider each bacterial colony as a point located at the centre of the colony, this simplifies the model in two ways. Firstly, we can take concentration to be constant within a single bacterial neuron, which is equivalent to considering the average concentration of inducer molecule and the subsequent activation strength across a colony. Secondly, we can use a simple mathematical formula for the diffusion of a molecule produced by a point source to approximate the behaviour of signalling molecule produced by the colony. This is a good approximation in the region outside the area of the colony, which is our region of interest as our design does not allow overlapping colonies. We also assume that bacterial neurons are instantly activated by their respective AHL, which is reasonable as the timescales of the experiments are long compared to the transcription and translation rate of the cells. The limitations of these assumptions will be discussed later.

Under these assumptions the same feed forward method can be used to approximate the output of a BNN with a few considerations. First, the activation function σ for each neuron will be one of the four available functions we can build into the genetic reaction network of an *E.coli* cell, the threshold (TH), inverse threshold (IT), bandpass (BP) and inverse bandpass (IB). The Threshold and inverse Threshold closely resemble the sigmoid function that sees widespread use within deep learning and the inclusion of the bandpass and inverse bandpass is supported by the fact that many non monotonic activation functions found to perform well in neural networks [126]. Secondly, as communication is done by diffusible molecules, the weight between each pair of neurons is dependent on the distance between them, r, and the time since activation of the first neuron, t ($\mathbf{W}^{(l)}(r,t)$).

To obtain an estimate of $\mathbf{W}(r,t)$ Mathematica [127] was used to solve the diffusion equation for a point source of an AHL with diffusion coefficient $3 \times 10^{-6} \frac{cm^2}{s}$ [128], over distances from 1-13.5mm and time 0-27 hours. In this chapter I assume that it takes ten hours for a diffusible signal to propagate from one layer to the next and hence for two bacterial neurons distance r apart we use the value $\mathbf{W}(r, 10 \text{ hours})$ to calculate their weight. The four activation functions were approximated as combinations of the sigmoid function, which was chosen as it resembles the response of bacterial colonies to diffusible molecules. In the future these functions can be found by fitting to characterisation data of the response of a colony to different concentrations of molecule. The activation functions used were

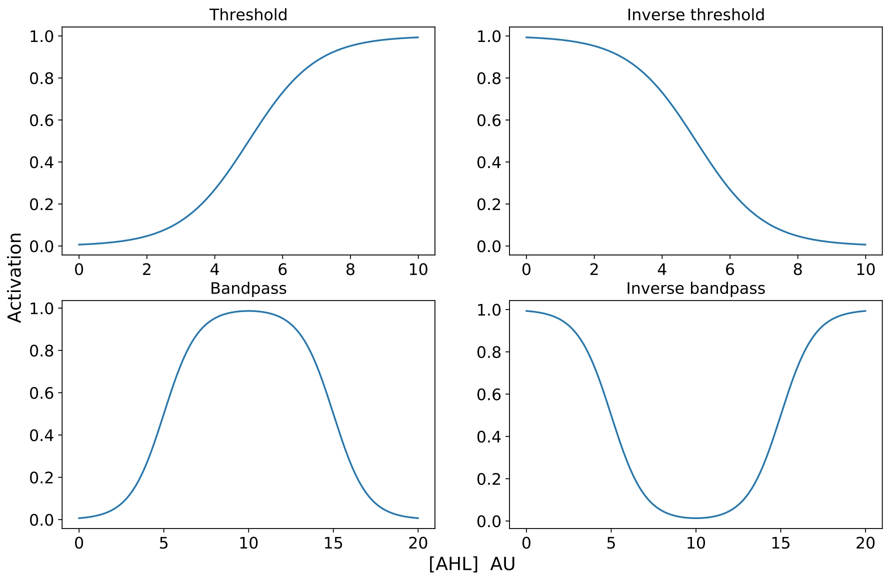


Figure 4.2: Activation functions used for the bacterial neural networks

$$TH = S(A-5)$$
$$IT = 1 - S(A-5)$$
$$BP = S(A-5) - S(A-15)$$
$$IB = S(A-15) - S(A) + 1$$

where S is the sigmoid function $S(x) = \frac{1}{1+e^{-x}}$, TH is the threshold, IT the inverse threshold, BP the bandpass, IB the inverse bandpass and A is the concentration of AHL. These are shown in Figure 4.2.

4.3.3 Evolutionary algorithm

Here I demonstrate the evolutionary algorithm to find optimal spatial patterns of bacterial colonies that implement the forward pass of a BNN such that a given target function is approximated with minimal error. I show the use of the evolutionary algorithm to design a BNN to learn the non-linear XNOR function, a classical classification problem in machine learning that is not possible using a linear classifier. I then use a finite difference model to show that the behaviour predicted by the neural network model is recapitulated in a more realistic reaction diffusion system.

Evolutionary algorithms work by first initialising a random population then evolving it through rounds of mutation, reproduction and selection steps. Just like a biological population the mutation and reproduction steps introduce variation into the population and the selection step drives the population to evolve in a certain direction. The evolutionary algorithm works as follows and the process is shown in Figure 4.3. Before the evolution can take place the user must supply some inputs to the algorithm (Figure 4.3A). Firstly, a list of allowable activation functions, either a subset or the whole suite of four activation functions can be available to be used in the network. Secondly, an initial network topology, which specifies the initial number of layers and number of neurons in each layer in the initial population. Third, the maximum size of the area available is given, this is a rectangle whose size can be specified and provides bounds which neurons are not allowed to move outside. For

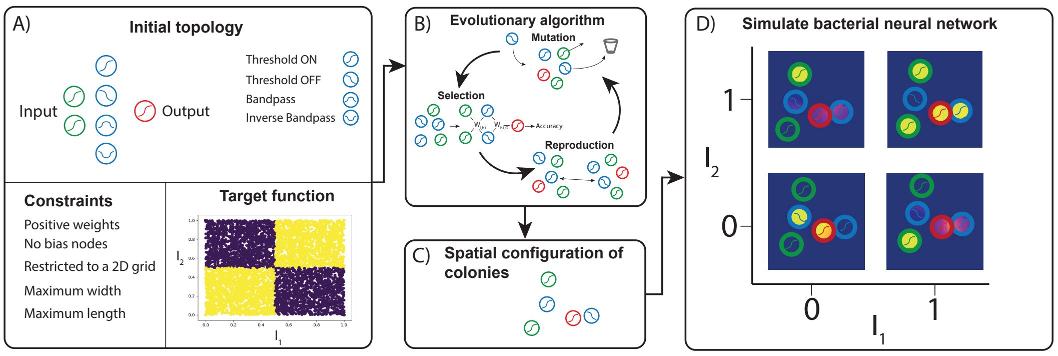


Figure 4.3: The, *in silico*, pipeline for training and simulating a bacterial neural network. A) An initial network topology, target function and constraints that arise from the physical system we are using are passed to the algorithm. B) A two dimensional spatial network is trained using an evolutionary algorithm. The mutation and reproduction operations involve changing the position of neurons randomly or swapping neurons between grids respectively. To select the best grids for the next generation the spatial configurations are converted into a neural network and tested against the target function. The grids with the highest lowest loss go on to the next generation. The best network (C) can then be simulated to verify it functions correctly (D).

this chapter the plate size was set to be 35mm by 35mm, the approximate size of a well in a six well plate. Finally the target function which the BNNs will be trained to approximate is passed in. The target function can be multi dimensional, where the number of neurons in the input layer will be equal to the number of dimensions. The number of output neurons can also vary, depending on the problem. Examples of problems include binary classification in two dimensions in which the areas of the input space should be classified as 0 or 1. This requires only a single output neuron. Multi-class classification, where each class is represented with a different integer and a distinct output neuron, will require as many output neurons as there are classes in the dataset. In this work I have focussed on classification problems, but regression could also be done on functions with continuous output.

The algorithm (Algorithm 9) first generates a random population of BNNs which obey the constraints on plate size, number of layers and the allowable activation functions. For each network the input and output layers are chosen to have threshold activation functions and placed in random positions within the bounds. Then for the hidden layers activation functions are chosen randomly from the allowable set of

Algorithm 9 Evolutionary training algorithm

```
1: input: target_function, constraints, available_activations, initial_topology
2: N = 100                                            ▷ number of evolutionary rounds
3: initial_pop = 1000                                 ▷ initial population size
4: population = generate_random_networks(initial_topology, pop_size)
5: for generation in 1...N do
6:     select(population)
7:     reproduce(population)
8:     mutate(population)
9: return best_network(population)                    ▷ return network with highest fitness
```

functions and also randomly placed on the plate. For the results in this chapter the initial population size was 1000. Each evolutionary round that follows is comprised of a selection, mutation, and reproduction step (Figure 4.3B). For the selection step the performance of each plate is evaluated by converting it into a neural network. This is done using the neural network model of bacterial communication developed above. The sum squared error loss with respect to the target function of the resulting neural network is calculated. A network with low loss is said to have high fitness and vice versa. Selection is applied to the population using stochastic universal sampling in which members of the population go onto the next generation with probability proportional to their relative fitness [129]. High fitness members can be copied multiple times into the next generation and the fittest member of the population is guaranteed to survive. Mutation is then carried out on the new population. The probability that any given network will mutate is equal to 0.1 and if a network does mutate there is a 0.2 probability that the position of any given neuron will be perturbed, a 0.1 probability that any given hidden layer will gain a neuron of random type and a probability of 0.1 that any given hidden layer will lose a random neuron. The final step is reproduction, in which each plate has a 0.3 chance of reproducing. If it does reproduce a second network is chosen at random to reproduce with. Two children are created which have random combinations of the parent's hidden neurons and are added to the population. The reproduction rate of 0.3 was chosen to keep the population approximately constant over the repeated rounds of evolution. The rounds of selection, mutation and reproduction were repeated for 100 generations.

The output of the algorithm is the network with the highest fitness or, equivalently, lowest loss with respect to the target function. The output defines the positions and types of bacterial neurons that form the network and these also satisfy the constraints imposed by placing colonies on a two dimensional plate (Figure 4.3C). In lieu of experimental verification we use a spatial finite difference model (see Section 2.1 for details) that simulates the system of communicating bacterial colonies on a two dimensional area (Figure 4.3D).

Using this algorithm I first investigated the training of a simple neural network with one hidden layer. The threshold, inverse threshold, bandpass and inverse bandpass can be used in the colonies in the hidden layer, whereas the input and output neurons always contain the threshold activation. Using the neural network model and evolutionary training algorithm a BNN was trained to approximate the non-linear XNOR function (Figure 4.3 A-C). The single hidden layer of the resulting BNN consisted of a single threshold and inverse threshold colony. The threshold colony was adjacent to, and therefore strongly strongly connected to, both inputs, while the inverse threshold was adjacent to the output colony. The finite difference simulation was run to verify the efficacy of the ANN model in approximating the diffusion dynamics of BNNs and the evolutionary training algorithm in finding good solutions. The plate was simulated and found to reproduce the XNOR function at the digital extremes (Figure 4.3 D), showing that using the neural network model and an evolutionary algorithm we have successfully designed a network of neurons communicating with a diffusible molecule and the predicted behaviour has been verified in the reaction diffusion system.

4.4 Bacterial neural networks can approximate a wide range of functions

A key problem in the field of deep learning and in the potential application of biocomputers to disease diagnosis is classification. Here I demonstrate the potential of BNNs to perform a wide variety of classification tasks. The results are summarised in Figure 4.4 where the network topology and predicted output of the simplified neural network model after training via the evolutionary algorithm is shown for four different target functions. Figure 4.4A is the predicted output of the network in Figure 4.3, and shows that away from the digital extremes the predicted output diverges from the target. Figure 4.4B shows that allowing the network to use an additional hidden layer increases performance on the XNOR classification task. Increasing network complexity has enabled more complex decision boundaries, the shape and the sharpness of the decision boundaries have improved. Figure 4.4C shows that the bandpass can increase the potential capabilities of the system, for example by introducing the capability to learn a circle with a very simple network. Figure 4.4D further demonstrates the utility of the bandpass, showing the potential to learn a periodic function. To verify the efficacy of the bandpass and inverse bandpass networks were trained on the target functions in Figure 4.4C and D with the bandpass and inverse bandpass omitted from the set of available activation functions. Under these conditions no good approximation to the target function could be found. This shows how the more complex activation functions at least partially alleviate some of the constraints of BNNs, such as the restriction to positive weights, allowing functions to be learned which would otherwise not be possible.

4.5 Application to biosensing

IBD and IBS have many symptoms in common, but are caused by different underlying mechanisms [130, 131]. This causes many patients to undergo expensive and invasive endoscopies. A simple testing kit could save billions of dollars and potentially allow IBD sufferers to monitor their gut health and predict when relapses would occur. PH and inflammation markers are two key factors in the gut that could be monitored [132] and biosensors for both are feasible [133, 134, 135]. Figure 4.4E shows the successful application of a BNN to the problem of diagnosing IBS and IBD, where the classification into 3 disease states (healthy, IBS, IBD) is done based off two biomarkers for health (pH and inflammation). These biosensors could be used to analyse a gut sample and a biosensor for each is contained in the input layer. To test the potential of BNNs to diagnose IBD, IBS and healthy gut states

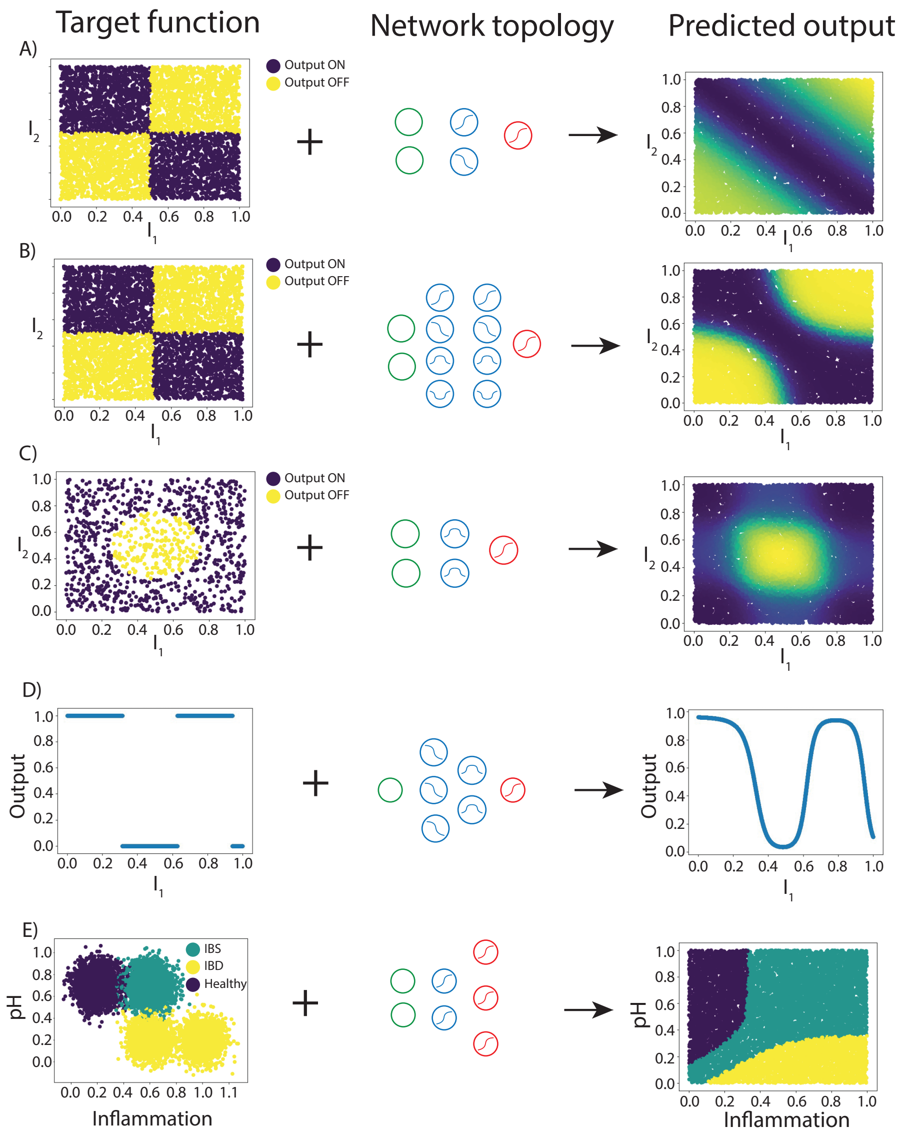


Figure 4.4: Summary of neural network results. Left the target function, middle the given network topology, right the predicted output of the spatial neural network. A) The XNOR function with one hidden layer. B) XNOR function with two hidden layers, the approximation is improved. C) The bandpass allows the learning of a circle. D) learning a simple periodic function. E) A possible application to diagnosis of IBS and IBD, where inputs from pH and inflammation biosensors are used to classify into healthy, IBS or IBD states.

artificial data was generated to represent the position of the three disease states on a two dimensional plane with pH and inflammation axes. A BNN was trained on the artificial data to perform the classification task. As shown, a relatively simple BNN, with a single hidden layer, has decision boundaries that correctly classify the three disease states. This shows that our BNN approach could be applied to medical diagnosis, fulfilling an overall goal of the field of biocomputing.

4.6 Discussion

In this chapter I have demonstrated that implementing small neural networks in two dimensional patterns of bacterial colonies has the potential to achieve analogue computing within a synthetic biological system. Firstly, I developed a model which simplifies the complex reaction diffusion system of communicating bacterial colonies to the forward pass of a neural network. This bridges the gap between neural networks and patterns of bacterial colonies and enables the fast calculation of a forward pass through a BNN. Then, using this model, I developed an evolutionary algorithm for the training of BNNs. The evolutionary training algorithm was demonstrated on the training of a two-dimensional non-linear XNOR classification task, and the predicted behaviour of the neural network was verified using a finite difference simulation of the system. This demonstrates that modelling bacterial communication networks as neural networks can enable the design of BNNs. Furthermore, I have shown the ability of this method to successfully train BNNs on a range of non-linear classification tasks, including a medically relevant diagnostic test to distinguish IBS from IBD, a key medical problem.

Most work in synthetic biocomputing has tried to emulate the success of digital computing within the field of electronics by implementing digital functions into cells. Although this is an important avenue of work, it is equally as important that we remain open to the other computational paradigms that might be better suited to the computational substrates available in biology. This is the first work to investigate the design of multilayer neural networks instantiated in patterns of bacterial colonies and is an important step towards achieving distributed analogue computation in synthetic biology. Neural networks have been implemented in cells before [124]. However, this approach requires complex genetic engineering to build each new network and control over the weights is restricted by the set of available promoters, ultimately limiting its use for practical applications such as intelligent medical diagnosis. The approach proposed here would require genetic engineering only in the initial construction of the neurons for the input, hidden and output layers and following this these neurons could be placed in different patterns using automated pipetting robots to perform many different functions. Integrated into my approach is the flexibility and robustness of neural networks, meaning that it should be possible to learn many different functions and potentially use different organisms for the neurons, such as mammalian cells.

The work in this chapter represents a good first step in the development of BNNs, namely the development of the design methodology and initial demonstration of its potential effectiveness. However, future work needs to be done before these designs can be realised in the laboratory. The activation functions I used in both the neural network model and the finite difference model were constructed from combinations of the sigmoid function that resemble the shape of the activation functions that can be built into bacteria. Future work could be done to gather characterisation data and fit functional forms of these true activation functions which can be used in the design of BNNs. Work could also be done to investigate the effectiveness of these functions for use in BNNs and find targets for future genetic optimisation of these GRNs. It is possible that some of the properties of the constructed functions such as leakiness and linearity at small concentrations could present challenges for building effective BNNs, however a wide variety of activation functions have been used and found to be effective in ANNs [126], so the method is likely to be robust to differences. Using realistic activation functions should be the first step in building a more complete finite difference model of the system. This model should include other effects I didn't consider, mainly cell growth and nutrient consumption and could be used to further verify that different networks behave as expected in a more realistic setting. Additionally, a mathematical investigation could be done into the possible limitations of a diffusion based neural networks given that weights are constrained to be positive and weights are no longer all independent from each other. An ANN lacking these constraints is a proven universal function approximator if it has at least one hidden layer [4, 5] and it would be interesting to see if an equivalent proof was possible for BNNs. This would also inform us of any limitations imposed by these constraints and, if required, possible solutions to these problems. Finally, the designs will need to be built and tested in the lab to verify that our model of BNNs can be used to design spatial bacterial patterns capable of analogue computing. In this chapter I focussed on a set of non-linear classification tasks due to their relevance in medical diagnostics, but this framework is general and could be used to solve many other classical machine learning problems, such as regression

In summary, in this chapter I have made the first steps towards the implementation of neural networks using patterns of bacterial colonies. I have shown that this is a potential area of interest for the future of synthetic biocomputing and I believe that future work on this approach would be fruitful.

Chapter 5

Deep reinforcement learning for the control of microbial co-cultures in bioreactors

5.1 Introduction

In recent years, there has been growing interest in implementing increasingly complex bioprocessing systems composed of multiple, interacting microbial strains. This has many advantages over single culture systems, including enhanced modularization and the reduction of the metabolic burden imposed on each strain. Despite these advantages, the control of multi-species communities (co-cultures) within bioreactors remains extremely challenging and this is a key reason why most industrial processing still use single cultures. In this chapter, I apply recently developed methods from artificial intelligence, namely reinforcement learning combined with neural networks, to the control of multiple interacting species in a bioreactor. This approach is model-free - the details of the interacting populations do not need to be known - and is therefore widely applicable. I anticipate that artificial intelligence has a fundamental role to play in optimizing and controlling processes in synthetic biology as the complexity of the systems we can engineer increases. This work was published in PLoS Computational Biology [102].

For my analysis, I use the chemostat model, which provides a standard description of bioprocess conditions. This model is applicable to a wide range of other systems where cell or microorganism growth is important, including wastewater treatment [136] and the gut microbiome [137]. Such systems can be especially difficult to control because they are often equipped with minimal online sensors [138], limiting the effectiveness of classical control techniques that are hampered by infrequent or delayed system measurements [100, 139].

Reinforcement learning is a branch of machine learning concerned with optimising an agent's behaviour within an environment. The agent learns an optimal behaviour policy by observing environmental states and selecting from a set of actions that change the environment's state (Figure 5.1A). The agent learns to maximise an external reward that is dependent on the state of the environment. The training of a reinforcement learning agent is often broken up into *episodes*. An episode is defined as a temporal sequence of states and corresponding actions (generated by the agent interacting with the environment) which progress until a terminal state is reached. The total reward obtained during an episode is called the return. For this study, I used a data-efficient variant of reinforcement learning called Neural Fitted Q-learning [140, 105, 141] (see Section 2.2.1 for details). Much reinforcement learning research has been done on video games [9] due to the availability of plentiful training data. However, it is also seeing application to more practical problems in the sciences, including the optimisation of chemical reactions [107] and in deriving optimal treatment strategies for HIV [142] and sepsis [143]. A partially supervised reinforcement learning algorithm has also been applied to a model of a fed-batch bioreactor containing a yeast monoculture [144].

Here I develop a control scenario in which the growth of two microbial species in a chemostat is regulated through the addition of nutrients C_1 and C_2 for which each species is independently auxotrophic (Figure 5.1B, 5.1C). The influx of each nutrient is controlled in a simple, on-off manner (bang-bang control). At each time point, the agent decides, for each auxotrophic nutrient, whether to supply this nutrient to the environment at the fixed inflow rate over the subsequent inter-sample interval. This constitutes the set of possible actions. A constant amount of carbon source, C_0, is

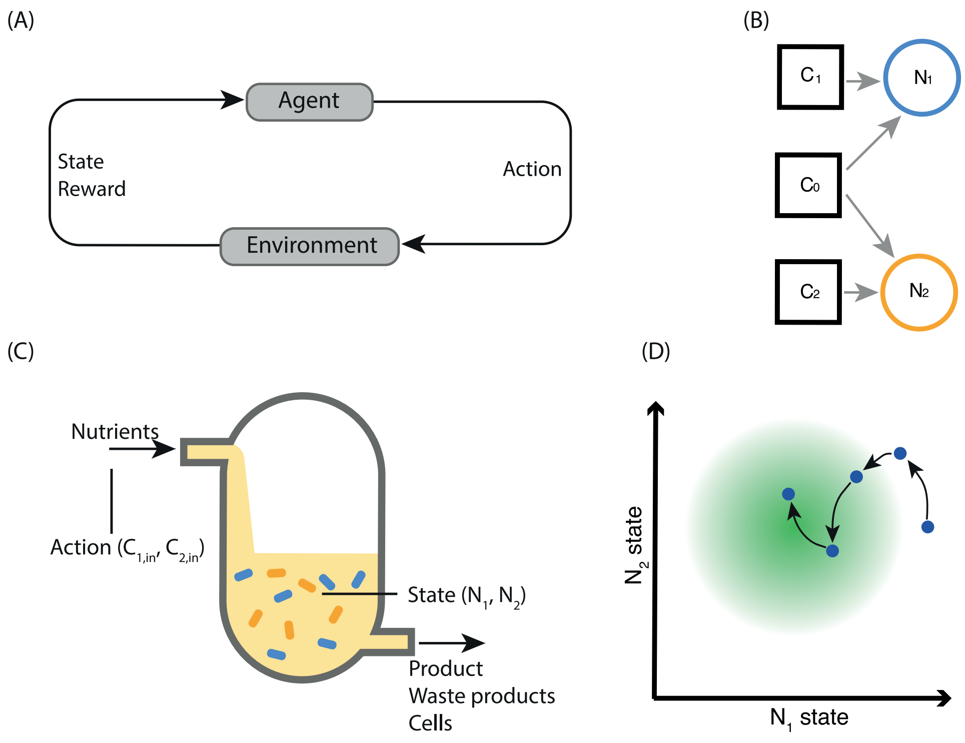


Figure 5.1: Reinforcement learning for the control of two auxotrophic species in a chemostat. (A) The basic reinforcement learning loop; the agent interacts with its environment through actions and observes the state of the environment along with a reward. The agent acts to maximise the total reward it receives (the return). (B) System of two auxotrophs dependent on two different nutrients, with competition over a common carbon source. (C) Diagram of a chemostat. The state observed by the reinforcement learning agent is composed of the populations of two strains of bacteria; the actions taken by the agent control the concentration of auxotrophic nutrients flowing into the reactor. (D) Representative system trajectory. The agent's actions, taken at discrete time-points (circles), influence the state dynamics (black arrows), with the aim of fulfilling the reward condition (moving to the centre of the green circle). The state is comprised of the (continuously-defined) abundance of two microbial populations, $\mathbf{N}_1$ and $\mathbf{N}_2$. The agent's actions dictate the rate at which auxotrophic nutrients flow into the reactor. At each time-step, the agent's reward is dependent on the distance between the current state from the target state.

supplied to the co-culture. I define the environmental state as the population levels of each population in the chemostat (assumed to be measured using fluorescence techniques). The objective is either to maintain specific population levels or to maximize product output. A corresponding reward is given, which depends on the distance of the population levels from the target value or as a function of product output. The populations evolve continuously, and the reward is likewise a continuous function of the state. In contrast, the agent's actions are discrete (bang-bang), and are implemented in a sample-and-hold strategy over a set of discrete sampling times. A visual representation of a two-population case is shown in Figure 5.1D.

Below, I illustrate that an agent can successfully learn to control the bioreactor system in the customary episodic manner and is robust to differing initial conditions and target set-points. Secondly, we compare a reinforcement learning approach to proportional integral control, both working in a model free way on simulated data, and show that the learning approach performs better in situations where sampling is infrequent. We then show that the agent can learn a good policy in a feasible twenty four hour experiment. Finally, I demonstrate that reinforcement learning can be used to optimise productivity from direct observations of the microbial community. Traditional proportional integral control could only be applied to such a case via an accurate model of the system, or with additional measurement data from further online sensors.

5.2 A mathematical model of interacting bacterial populations in a chemostat

I develop a general model of m auxotrophs growing and competing in a chemostat. An auxotroph is a bacterial strain that has is reliant on a specific nutrient, often an amino acid, for growth. The model captures the dynamics of the abundance of each species (m-vector $\mathbf{N}$), the concentration of each auxotrophic nutrient (m-vector $\mathbf{C}$), and the concentration of the shared carbon source (scalar C_0). A sketch of the two-species case is shown in Figure 5.1B-C.

Parameter	Description	Value	Unit	Source
$\mathbf{C}_{0,in}$	Reservoir concentration of carbon source	1	g L^{-1}	Experimentally controllable
q	Flow rate	0.5	h^{-1}	Experimentally controllable
γ_0	Yield coefficient for common substrate	4.8×10^{11}	cells g^{-1}	[145]
γ_1	Yield coefficient for arginine	5.2×10^{11}	cells g^{-1}	[146]
γ_2	Yield coefficient for tryptophan	4.4×10^{11}	cells g^{-1}	[146]
$\mu_{max,1}$	Maximum growth rate	1	h^{-1}	[147]
$\mu_{max,2}$	Maximum growth rate	1.1	h^{-1}	[147]
$\mathbf{K}_{s,0}$	Saturation constant for the carbon source	6.8×10^{-5}	g L^{-1}	[146]
$\mathbf{K}_{s,1}$	Saturation constant for arginine	4.9×10^{-4}	g L^{-1}	[146]
$\mathbf{K}_{s,2}$	Saturation constant for tryptophan	$1.02-6.9\times10^{-6}$	g L^{-1}	[146]

Table 5.1: Parameters for the double auxotroph system. Parameter values used for simulations of a system consisting of two auxotrophic populations of bacteria with competition for nutrients. μ_{max} values were chosen using values from the literature [147] as a guide.

The rate of change of the concentration of the shared carbon source is given by:

$$\frac{d}{dt}C_0(t) = q(C_{0,in} - C_0(t)) - \sum_{i=1}^{m} \frac{1}{\gamma_{0,i}}\mu_i(t)\mathbf{N}_i(t) \tag{5.1}$$

where γ_0 is a vector of the bacterial yield coefficients for each species, $C_{0,in}$ is the concentration of the carbon source flowing into the bioreactor, μ is the vector of the growth rates for each species, and q is the flow rate. The parameters are found in Table 5.1.

The concentration of each auxotrophic nutrient, $\mathbf{C}_i$, is given by:

$$\frac{d}{dt}\mathbf{C}_i(t) = q(\mathbf{C}_{i,in}(t) - \mathbf{C}_i(t)) - \frac{1}{\gamma_i}\mu_i(t)\mathbf{N}_i(t), \tag{5.2}$$

where γ is a vector of bacterial yield for each auxotrophic species with respect to their nutrient and $\mathbf{C}_{in}$ is a vector of the concentration of each nutrient flowing into the reactor (which is the quantity controlled by the reinforcement learning agent). Note that I assume all the auxotrophs are independent, i.e. each auxotrophic nutrient is only used by one population.

The growth rate of each population is modelled using the Monod equation:

$$\mu_i = \mu_{max,i}\frac{\mathbf{C}_i}{\mathbf{K}_{s,i}+\mathbf{C}_i}\frac{C_0}{\mathbf{K}_{s0,i}+C_0}, \tag{5.3}$$

where μ_{max} is a vector of the maximum growth rate for each species, $\mathbf{K}_s$ is a vector of half-maximal auxotrophic nutrient concentrations and $\mathbf{K}_{s0}$ is a vector of half-maximal concentrations, C_0, for the shared carbon source. Finally, the growth rate for each population is determined as:

$$\frac{d}{dt}\mathbf{N}_i(t) = (\mu_i(t) - q)\mathbf{N}_i(t). \tag{5.4}$$

5.3 Controlling interacting bacterial populations in a chemostat

Episodic Fitted Q-learning (Algorithm 2, Section 2.2.1) was applied to a model of the system shown in Figure 5.1 using the parameters in Table 5.1. This model was parametrised to simulate the growth of two distinct *E. coli* strains in a continuous bioreactor, with glucose as the shared carbon source, C_0, and arginine and tryptophan as the auxotrophic nutrients, C_1 and C_2. The reward was selected to penalize deviation from target populations of $[N_1, N_2] = [20, 30] \times 10^9$ cells L^{-1}. Specifically, the reward function was: $r = \frac{1}{10}(1 - \frac{1}{2}(\frac{|N_1 - N_1^{target}|}{20\times10^9} + \frac{|N_2 - N_2^{target}|}{30\times10^9})$. The scaling was of $\frac{1}{10}$ was selected to ensure a maximum possible reward of 0.1, which helped prevent network instability. (Negative rewards below -0.1 are possible; however due to the system dynamics they rarely occurred and did not effect training performance). The contribution of each population was scaled according to its target value so that each contributed proportionally to the reward. This prevented the contribution to the reward function from one strain becoming insignificant if its target value was considerably smaller than the other. The absolute error was chosen because it is continuous and differentiable (except when populations are at the target value) and has a unique minimum, all properties that are favourable for reinforcement learning in continuous state spaces. Absolute error was chosen over the squared error so that the reward gradient didn't diminish in the region near the target. The reward function is based on target population levels because we have already assumed that these are measurable through, for example, fluorescence measurement. Selection of a target set-point in state space is also an approach widely used with more traditional control techniques and so allows for direct comparison to these. The state variables considered by the algorithm were the continuous populations of each species of microbe. The agent acted as a bang-bang controller with respect to each input nutrient, giving 2^n possible actions, where n is the number of nutrients (in this work, $n = 2$). The neural network that was used to estimate the value function consisted of two hidden layers of 20 neurons, following the approach in previous work [105]. Each neuron in the hidden layers used the ReLU activation function. The input layer had n neurons, one for each microbial strain; the linear output layer had 2^n neurons one for each available action. The populations levels were scaled by a factor of 10^{-5} before being entered into the neural network; this generated values between 0 and 1 (with units 10^6 cells L^{-1}) and prevented network instability. The odeint function of SciPy (version 1.3.1) [148] was used to numerically solve all differential equations in this chapter. The code and examples are available on GitHub: https://github.com/ucl-cssb/ROCC.

First an investigation is done to find the minimum inter-sampling period which can be used in conjunction with reinforcement learning. This is important as it is not guaranteed that reinforcement learning can be applied to this partially observed problem without increasing the inter-sample period to a practically unsuitable time. Then investigations are done to find suitable values for an important hyper parameter, the number of Fitted Q-iterations done after each episode. This hyper parameter needs to be chosen such that there is a balance between enough iterations to ensure value convergence, but not so many that overfitting can occur. Finally, I demonstrate the application of Fitted Q-learning to the model of interacting bacteria in a chemostat and show that it is robust to different initial conditions and targets.

5.3.1 Minimum inter-sampling period

The theoretical convergence guarantees of reinforcement learning assume that it is applied to a Markov decision process [103]. The two-strain chemostat system we use here has five state variables: two auxotrophic nutrient concentrations, the concentration of carbon source and the two microbial population levels. Only the population levels are known to the agent, meaning the system is only partially observed and is hence not a Markov decision process. There are methods to extend reinforcement learning to partially observed Markov decision processes, including incorporating time series information using a recurrent network [10], keeping track of approximate belief states of the hidden variables [149] or using Monte-Carlo methods [150]. To assess whether these computationally expensive methods would be required, I determined the minimum sample-and-hold interval that allowed the agent to accurately predict the reward resulting from a chosen action. (Intuitively, this can be thought of as the minimum sample-and-hold interval in which an action has time to have an observable effect on the the population levels.) To determine this minimum interval, I first generated system trajectories of (s_t, a_t, r_t) resulting from random actions. The agent was trained on these sequences to predict the reward, r_t, from the state action pairs (s_t, a_t). This was done with the rationale that the ability to correctly predict the instantaneous reward from a state-action pair is a requirement for the ability to predict the value from a state-action pair. I repeated this process one hundred times for each of the following sampling times: [1,2,3,4,5,10] minutes. The results, shown in Figure 5.2A, indicate that at time steps lower than four minutes the agents are unable to accurately predict the reward received from the state and action, meaning reinforcement learning cannot be effective. However at four minutes and above the reward prediction is accurate. I concluded that by using intervals of four

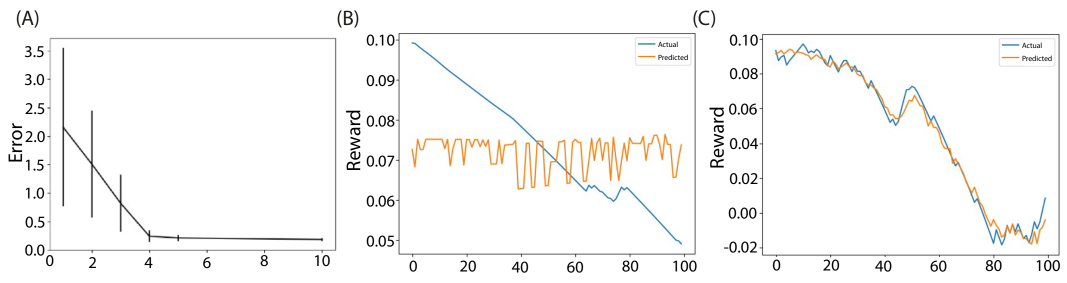


Figure 5.2: Identifying the minimum timestep above which the chemostat system behaves effectively as a Markov decision process. (A) The error in reward prediction is negligible for time steps above four minutes. Error bars represent one standard deviation over one hundred repeats. The predicted vs actual reward for one minute (B) and five minute (C) timesteps. Markov decision-based learning is not possible for the short one-minute intervals, but performs well for five-minute intervals.

minutes or longer, the sophisticated non-Markovian methods mentioned above would not be required for this application. Figure 5.2 B and C show the reward prediction for both one- and five-minute time steps, showing that the agent performs well for five minutes intervals and poorly for one minute intervals. For the remaining results in this section an inter-sampling period of five minutes is used and we assume that this is short enough that it would not present a practical problem for implementation.

5.3.2 Number of Fitted Q-iterations to avoid over fitting

To determine how many Fitted Q-iterations to implement, I generated sequences of (s_t, a_t, r_t) of varying lengths by interacting with the chemostat system using randomly chosen actions. Fitted Q-agents were again trained to predict the instantaneous reward r_t from the state-action pair (s_t, a_t). I determined the training and testing error for each Fitted Q-iteration, with a maximum of 40 iterations. Figure 5.3 shows the results of repeating this process 100 times for each sequence length. The data reveal clear overfitting for the datasets shorter than 200 time steps long and a reduction in testing error as the sequence length increases (i.e. with more training data). For each sequence length, the training process with 4 Fitted Q-iterations gave the smallest testing error (except for 100 training timesteps, where 5 iterations performed marginally better). With a training set of 200 time steps, no significant overfitting occurred.

5.3.3 Number of Fitted Q-iterations for value convergence

Another consideration is how many Fitted Q-iterations are required for the values to converge via bootstrapping. For this analysis, I generated 100 sequences of (s_t, a_t, r_t), each one thousand time steps long, in the same manner as the previous section. For each sequence, the actual values were calculated and Fitted Q-iteration was used to obtain predicted values. After each Fitted Q-iteration, the error between the predicted and actual values was recorded. As shown in Figure 5.4, the values converge after about ten iterations. Using this and the information from the previous section, the number of Fitted Q-iterations was chosen depending on the length of the agent's memory to both prevent overfitting and to allow convergence via bootstrapping.

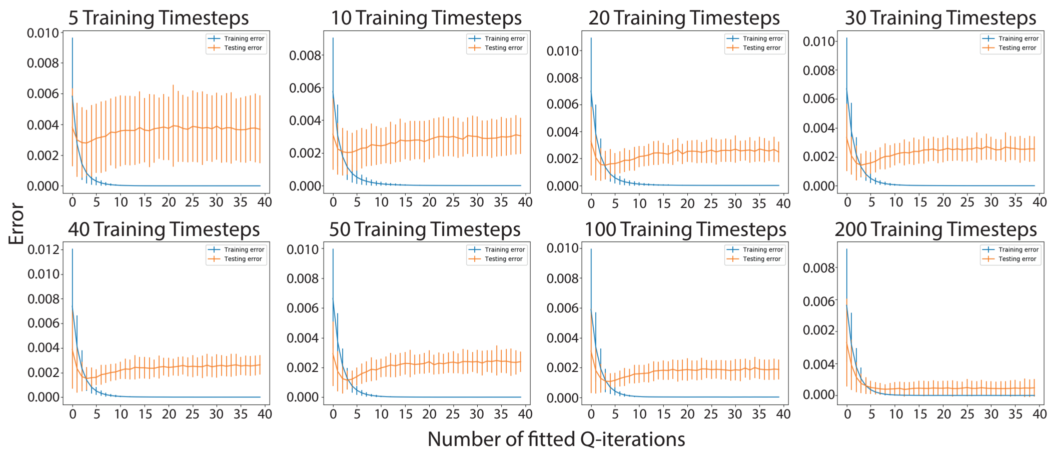


Figure 5.3: Overfitting of Fitted Q-iteration for different dataset sizes. The training (blue) and testing (orange) accuracy of a Fitted Q-agent to predict rewards from states and actions was tested after every Fitted Q-iteration. Overfitting is seen for number of transitions less than 200. Error bars represent one standard deviation.

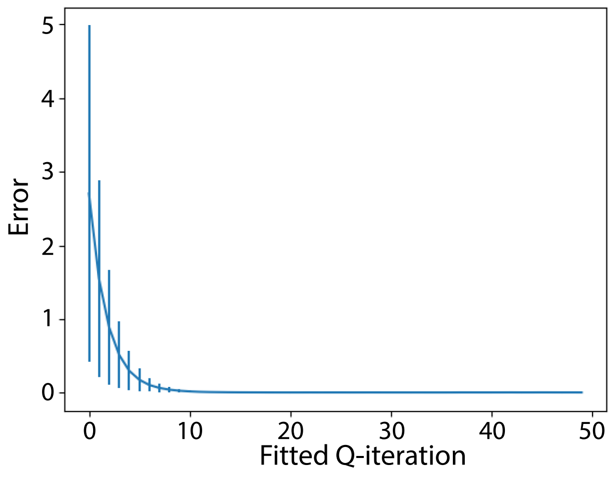


Figure 5.4: Convergence of Fitted Q-iteration. The scaled error between actual and predicted values as Fitted Q-iterations are completed

For all Episodic Q-learning the number of Fitted Q-iterations was set to 10 as the agent's memory always contains at least one episode of 288 transitions. For online Q-learning the number of Fitted Q-iterations was set to 4 if there were less than 100 transitions in the agent's memory, 5 if there were 100-199 transitions and 10 if there were 200 or more transitions.

5.3.4 Reinforcement learning can be used to control the bioreactor system

The agent was trained on thirty sequential episodes, this provided enough data for the agent to learn while not being prohibitive in terms of computational time. Each episode was twenty four hours long with sampling and subsequent action choice every five minutes. The initial system variables of the chemostat for each episode were $[N_1, N_2] = [20, 30] \times 10^9$ cells L^{-1} and $[C_0, C_1, C_2] = [1, 0, 0]$ $g\ L^{-1}$. The explore rate was initially set to $\varepsilon = 1$ and decayed as $\varepsilon = 1 - \log_{10}(aE)$ where E is the episode number, starting at 0, and $a = 0.3$ is a constant that dictates the rate of decay. A minimum explore rate of $\varepsilon = 0$ was set and was reached by the end of training. Figure 5.5A shows the training performance of twenty replicate agents, each trained over thirty episodes. The twenty agents converged to a mean final return of 27.4 with a standard deviation of 0.33. The theoretical maximum return is 28.8; all twenty agents were thus able to learn near optimal policies despite being restricted to bang-bang control. The population curve in Fig 5.5B shows the system behaviour when under control of a representative agent trained in one of the replicates (for all twenty replicates see Figure A.1). The population levels track the targets, with some jitter as expected with a bang-bang controller. Fig 5.5C shows the value function learned by this representative agent at the end of training, indicating its prediction of the total return from each point in state space. As expected, the value peaks at the target point. The corresponding state-action plot, Fig 5.5D, shows that the agent has adopted a simple, intuitive feedback law: add the specific nutrient needed by a strain when its population level is below the target and refrain from adding the nutrient if it is above the target. From these results, we conclude that reinforcement learning can be successfully applied to the chemostat system with a practical inter-sampling period of five minutes, as predicted .

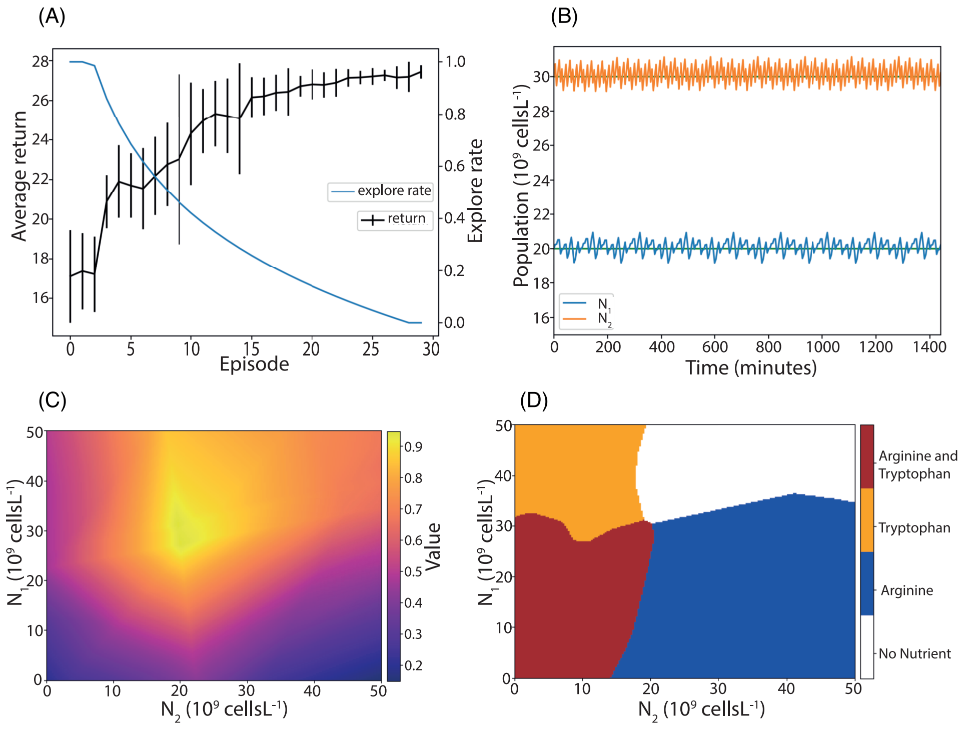


Figure 5.5: Reinforcement learning applied to the bioreactor system. (A) Performance of the agent improves and the explore rate decreases during training. The average of the return over twenty training replicates is plotted; error bars represent one standard deviation. (B) System behaviour under control of a trained agent for a twenty four hour period. Populations are maintained near target values (green lines). (C) Heatmap of the learned state value function; values are maximal at the target. (D) A learned state action plot, showing the agent's learned action (coloured regions) over the state space.

5.3.5 Reinforcement learning is robust to different initial conditions and targets

To verify that the algorithm is robust to different initial conditions and different target populations I began by choosing a range of initial population values: $[5,10,40,50] \times 10^9$ cells L^{-1} and two different targets: $[20,30] \times 10^9$ cells L^{-1} and $[30,20] \times 10^9$ cells L^{-1}. For every combination of initial populations and target, a Fitted Q-agent was trained in the same manner as in the previous section. This was repeated three times. One of the three population curves from each experiment is shown in Figure 5.6, the corresponding actions for the first 600 minutes of the simulation are shown in Figure 5.7 and the average return across each of the three repeats is shown in Figure 5.8. On 2 of the 96 total repeats the agent failed to maintain the populations at the target levels. This happened with the target: $[30,20] \times 10^9$ cells L^{-1} with initial conditions $[N_1, N_2] = [5,5] \times 10^9$ cells L^{-1} and $[N_1, N_2] = [5,10] \times 10^9$ cells L^{-1}. Intuitively these represent two of the most challenging combinations, where the target has the slower growing strain (N_1) above the faster growing strain (N_2) and in which both populations are at low initial values.

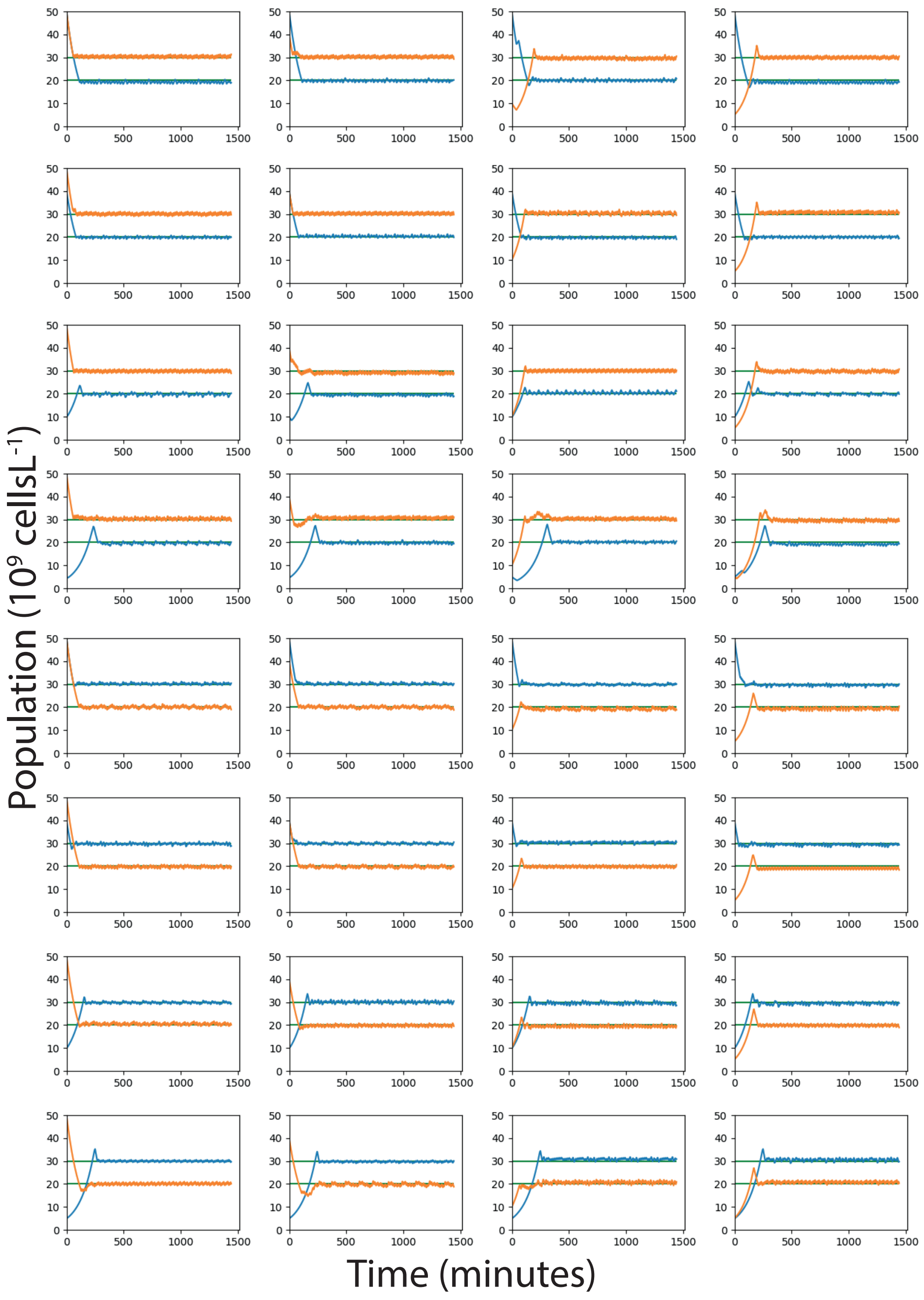


Figure 5.6: Population curves for different initial conditions and targets. Populations of one of the three replicates for each of the different initial conditions and targets.

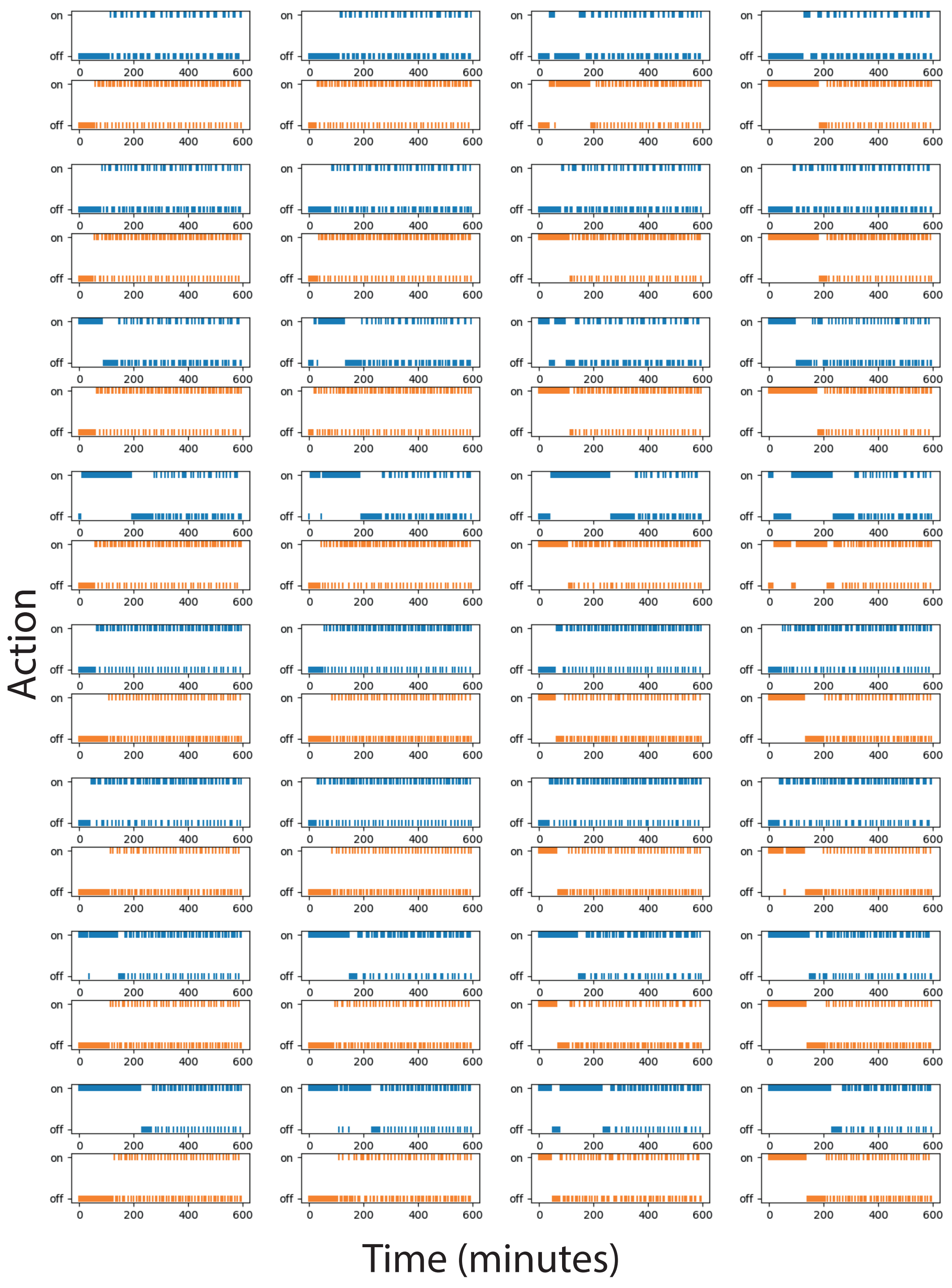


Figure 5.7: Actions for different initial conditions and targets. The actions taken by the agent for the first 600 minutes of one of the three replicates for each of the different initial conditions and targets. The top graph of each panel shows the agent's actions with respect to the addition of the nutrient that N_1 is dependent on, the bottom graph shows the same for N_2.

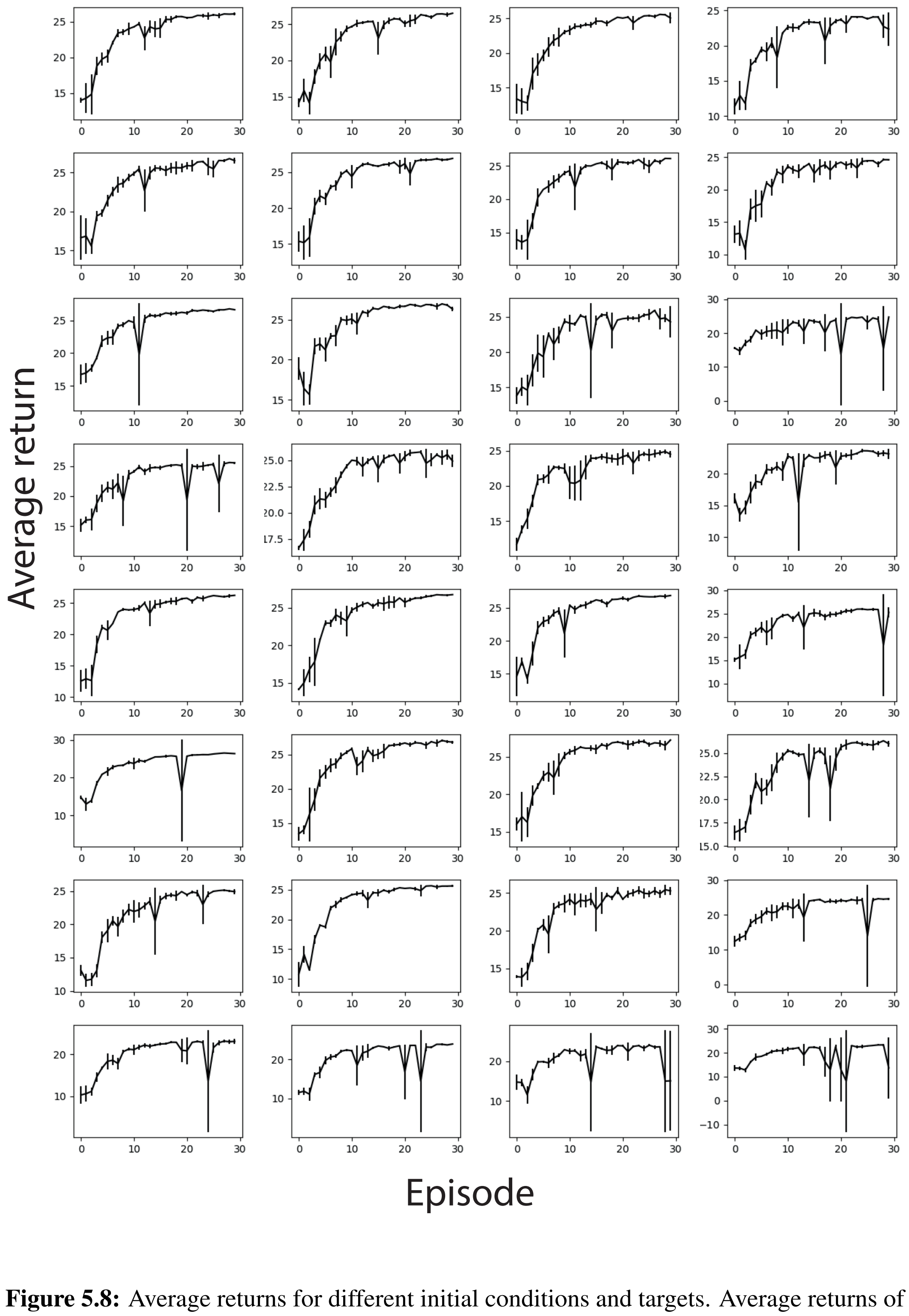


Figure 5.8: Average returns for different initial conditions and targets. Average returns of the three replicates for each of the different initial conditions and targets. Error bars represent one standard deviation.

5.4 Comparison of reinforcement learning with proportional integral control

A potential advantage of the reinforcement learning method developed in this chapter is the improved performance on systems that are hard to measure or interact with. In all the results in this chapter the Fitted Q-agent is restricted to bang-bang control. This means that for each controlled input only the implementation of discrete on and off states would be required, significantly simplifying the interaction with the system. In this section I also investigate how the Fitted Q-agent handles systems that are hard to measure. This was done by comparing the performance of the agent against the industry standard technique, proportional integral (PI) control (see Section 2.3 for details) at different inter-sampling periods.

5.4.1 Proportional integral controller tuning

For the comparison between reinforcement learning and PI control, I tested a range of sample-and-hold intervals ($[5,10,20,30,40,50,60]$ mins). For each choice of sampling interval, I generated thirty, twenty-four hour long episodes, each starting with initial system variables $[N_1,N_2] = [20,30] \times 10^9$ cells L^{-1} and $[C_0,C_1,C_2] = [1,0,0]$ $g\ L^{-1}$ and selecting actions randomly from $[0,0.1]$ $g\ L^{-1}$. These thirty episodes were used as training data for the Fitted Q-agent. From each dataset, an input-output model was constructed using the plant identification function in the PID tuner app of MATLAB's Simulink toolbox [115], which allows the identification of an input-output model for any input-output dataset. Here, the randomly chosen actions were used as input and the resulting populations (scaled by a factor of 10^{-10}) were taken as output. The model was a state space model, of order chosen by the system identification app to best fit the data. The Akaike's Final Prediction Error (FPE) of the model fits was of the order 10^{-2} for the 5 min sample-and-hold intervals, rising to a maximum of almost 1 for 60 min intervals. An independent input-output model was derived for each microbial population. These were used to tune two independent PI controllers, one controlling each population. I used independent controllers because the PI tuner app is only compatible with single

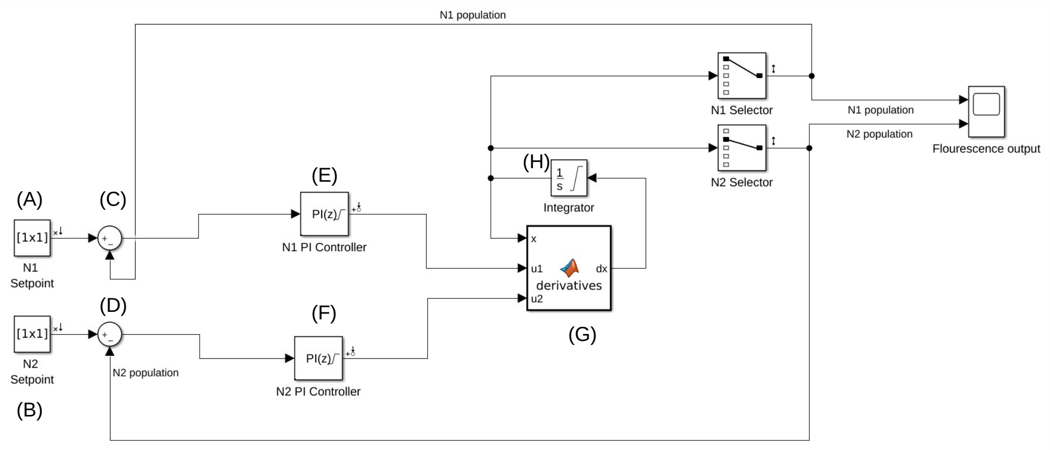


Figure 5.9: The Simulink diagram of the system and PI controllers. (A,B) the setpoints or target population levels, from which the error is calculated (C,D) and used by the PI controllers (E,F) to adjust nutrient levels. The system of ODEs (G) is solved by a continuous time integrator (H).

input, single output systems. I considered a range of tuning objectives to assess the merits of tuning to minimise settling time, rise time or overshoot percentage. I found that minimising rise time led to high overshoot errors, while minimising overshoot percentage also led to high errors because the controller would be slow to reach the target. Tuning the controller to minimise settling time worked best for all cases tested and can be seen as a compromise between speed of response and robustness. Hence, for all results presented, the PI controllers were tuned to minimise settling time. The Simulink diagram of the system is shown in Figure 5.9.

5.4.2 Reinforcement learning outperforms proportional integral control for long inter-sampling periods

As a comparison to a standard control approach, the reinforcement learning controller was compared to a traditional PI controller. The controllers differ in that the proportional integral controller implements feedback over a continuous action space, whereas the reinforcement learning controller uses bang-bang control. For both controllers thirty episodes of data were generated, each twenty-four hours long, for a range of sampling-and-hold intervals: $t_s = [5, 10, 20, 30, 40, 50, 60]$ mins by starting with initial system variables $[N_1, N_2] = [20, 30] \times 10^9$ cells L^{-1} and $[C_0, C_1, C_2] = [1, 0, 0]\ g\ L^{-1}$ and sampling random input concentrations C_1, C_2 from $[0, 0.1] gL^{-1}$. For each choice of sampling frequency, the reinforcement learning agent was trained using Fitted Q-iteration (Section 2.2.1, Algorithm 1) on the dataset of thirty randomly generated episodes, while the proportional integral controller was tuned on an input-output model of the system derived from the same dataset. The performance of the two controllers is illustrated in Figure 5.10, which shows how the performance depends on the choice of sampling frequency. For inter-sampling intervals longer than five minutes, the reinforcement learning controller outperforms the proportional integral controller and the difference in performance increases as the inter-sampling interval increases. I conclude that reinforcement learning can produce comparable and even better performance, with the potential added advantage of a simpler implementation (the proportional integral controller employs continuous actions, whereas the reinforcement learning controller uses only bang-bang control).

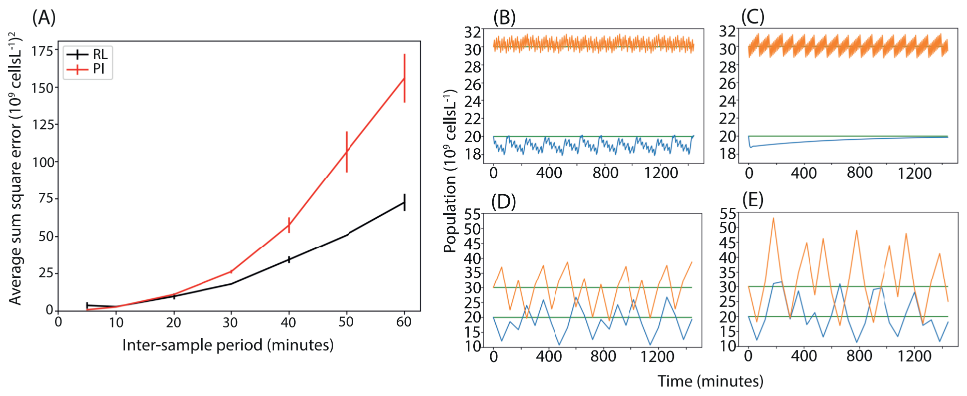


Figure 5.10: Comparison of reinforcement learning and proportional integral controllers. (A) The scaled average sum square error between the system state and the target. For long inter-sample periods, the reinforcement learning controller outperforms the proportional integral controller. The sum square error was calculated from population values that were scaled by a factor of 10^{-9}. (B-C) Population time-courses under the reinforcement learning and proportional integral controllers respectively, with a five minute inter-sampling time. (D-E) Population time-courses under the reinforcement learning and proportional integral controllers respectively, with a sixty minute inter-sampling time. Here the proportional integral controller allows the populations to stray further from their target values.

Moreover, for microbial chemostat systems that are difficult or expensive to sample at high frequency, reinforcement learning could be the preferred option.

5.5 A good policy can be learned online using parallel bioreactors

A potential barrier to the use of reinforcement learning in real world applications is the amount of data required. Experimental systems do exist that would allow one to gather the necessary data to train an agent in the manner demonstrated above [151]. However, I aim to lower the barrier of entry so that our method can be implemented in low cost bioreactors. For example, the development of a low cost turbidostat capable of running eight cell culture experiments in parallel, demonstrated on time periods up to 40 hours [152] presents a realistic scenario for a cell biology lab. I next show that Online Fitted Q-learning, a variant of Fitted Q-learning adapted to run in an online manner (Section 2.2.1, Algorithm 3), can learn to control the chemostat system using an amount of data realistically obtainable in a single experiment. I trained an agent online on five chemostat models running in parallel. Each modelled the system described in Figure 5.1B and was run for the equivalent of twenty-four hours of real time. The agent took an action every five minutes, making an independent decision for each of the five chemostats from a single policy learned from experience gathered from all models. The reward was observed and the value function updated by the agent every ten time steps, using all experience gathered up to that time (Figure 5.11A). As in the previous sections, the initial microbial populations were set to the target value of $[N_1, N_2] = [20, 30] \times 10^9$ cells L^{-1} and the initial concentrations of the nutrients were $[C_0, C_1, C_2] = [1, 0, 0]\ g\ L^{-1}$. Figure 5.11B shows the online reward the agent received from the five chemostats. The initial reward was high, due to the initial populations being set to the target values. As the agent explored, the reward decreased and the standard deviation between the parallel chemostats increased because the agent took independent exploratory actions in each chemostat and drove them into different regions of state space. As time progressed, the reward from all five chemostats increased and the standard deviation decreased because the

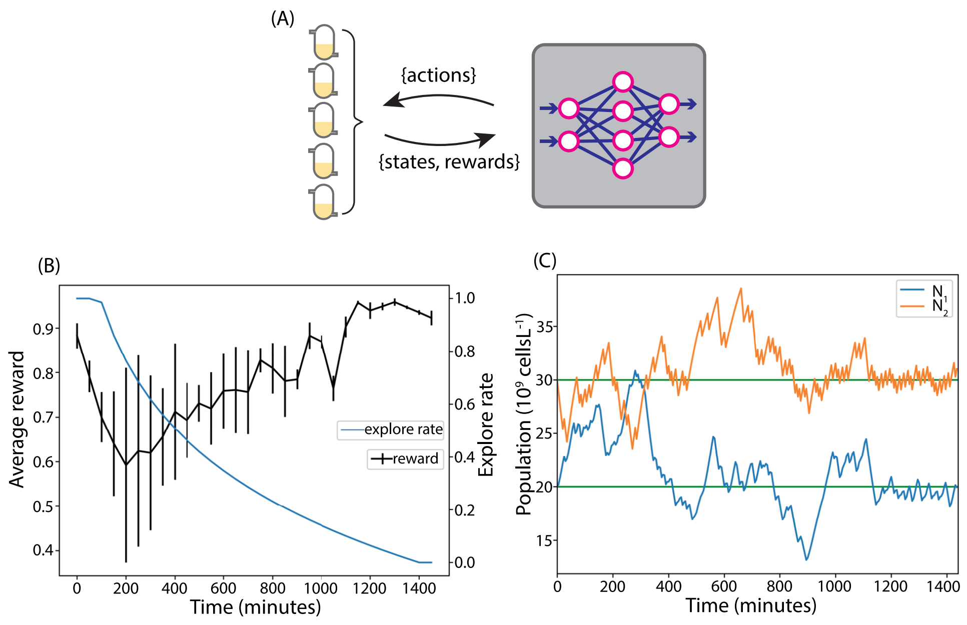


Figure 5.11: Learning a policy in twenty four hours. (A) A reinforcement learning agent was trained online on models of five parallel chemostats for twenty four hours. (B) The average reward received from the environments. By the end of the simulation all five chemostats were moved to the target population levels with very little standard deviation in reward. (C) The population curve from one of the chemostats. During the exploration phase the population levels vary and random actions are taken, as the explore rate decreases they move to the target values.

agent learned and moved all populations closer to the target. A pair of representative population time-courses is shown in Figure 5.11C (all five are shown in Appendix A.2). From these results, I conclude that Online Fitted Q-learning can be used to learn a policy in a data-efficient, online manner.

5.6 The yield of a community-based product can be directly optimised

Another advantage of reinforcement learning is the flexibility to optimise different reward functions, while still engaging in model free control with no knowledge of the underlying system dynamics. To demonstrate the ability of reinforcement learning to directly optimise the output of a community-based bioprocess, the system in Figure 5.12A was modelled. Here, each microbial strain produces an intermediate product; N_1 produces A and N_2 produces B, each at a rate of 1 molecule per cell per hour. Factors A and B react to a product P, via the reaction $2A + B \rightarrow P$, which is presumed rapid. Consequently, the optimal state of the system has population ratio $N_1 : N_2 = 2 : 1$, with the populations at the maximum levels that the chemostat can support, which in our model means that all the carbon source, C_0, is being consumed. In this case, I set the agent's reward to be proportional to the instantaneous production rate of the bioreactor. I again take the observed state and the available actions to be the population levels and the bang-bang auxotrophic nutrient inflow rates, respectively. I set the initial populations to $[N_1, N_2] = [20, 30] \times 10^9$ cells L^{-1} and initial nutrient concentrations to $[C_0, C_1, C_2] = [1, 0, 0]$ $g\ L^{-1}$ as before. The initial levels of A, B and P were all 0. Ten replicate agents were trained using Episodic Fitted Q-learning (Section 2.2.1, Algorithm 2). The sample-and-hold interval was increased to ten minutes, which improved learning performance by giving sufficient time for the agents actions to effect production rate. Performance in terms of the return is shown in Figure 5.12B. The average ratio of the population levels in steady state (the last 440 minutes of the simulation), over all agents, was 1.99 (with s.d. 0.08), showing convergence to near optimal populations in all replicates. A representative population time-course is shown in Figure 5.12C, for all time-courses see Appendix A.3. Likewise, the average final concentration of the carbon source was 0.11% (s.d. 0.015%) of the source concentration, showing that in all cases the total population was close to the carrying capacity of the chemostat. As shown in Figure 5.12D, the replicates showed very little deviation in the final product output. However, in the initial phase of moving and stabilising the populations to the optimal levels, there is significant deviation. This suggests that most of the deviation in return shown in Figure 5.12B is due to this initial stabilising phase and not to the final phase the agents reached. From this analysis, I conclude that the reinforcement learning agent can learn to move the system to, – and keep it at – the near optimal state for product formation in a model free way.

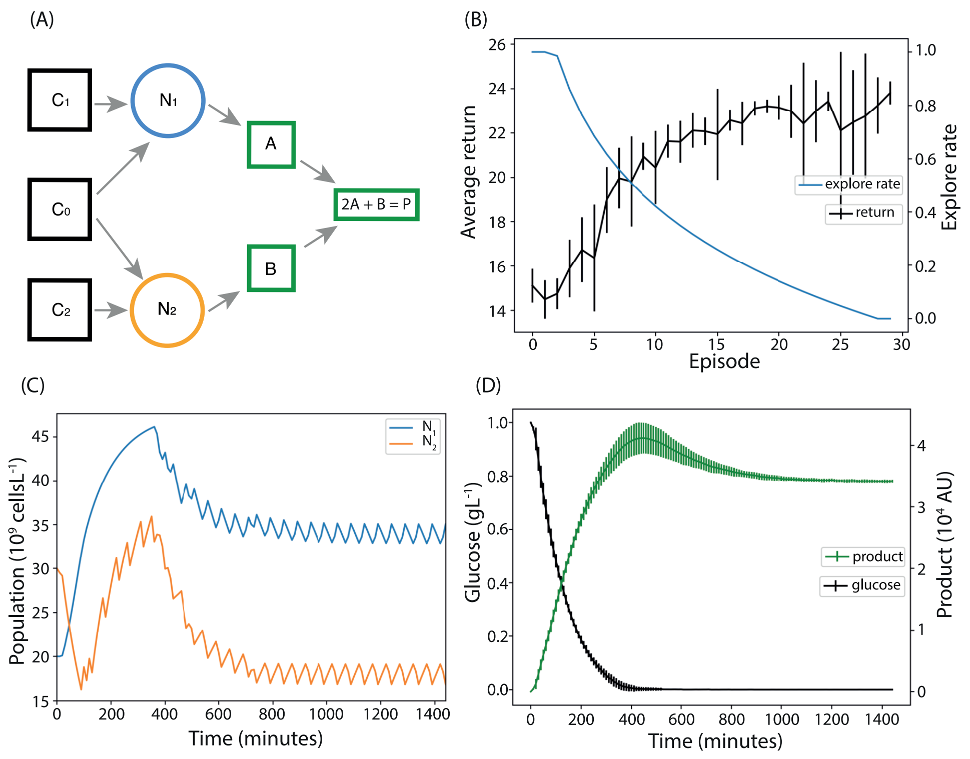


Figure 5.12: Using reinforcement learning to optimise product output. (A) Each microbial population produces an intermediate; these react to produce the desired product. (B) Training performance of ten reinforcement learning agents trained to optimise product output. (C) The resulting population curves of the system under control of a representative agent. The populations reach and then are maintained at the optimal level for product production. (D) The levels of carbon and product inside the chemostat. After the initial phase all carbon is being consumed. The levels of product peak as all initial carbon is used, then reach a level supported by the carbon supply to the reactor.

5.7 Discussion

I have applied deep reinforcement learning, specifically Neural Fitted Q-learning, to the control of a model of a microbial co-culture, thus demonstrating its efficacy as a model-free control method that has the potential to complement existing techniques. I have shown that reinforcement learning can perform better than the industry standard, PI control, when faced with long sample-and-hold intervals. In addition, I showed that the data efficiency of Neural Fitted Q-learning can be used to learn a control policy in a practically feasible, twenty-four hour experiment. Reinforcement learning is most often used in environments where data is cheap and effectively infinitely available. Importantly, this result shows that it can also be realistically used to control microbial co-cultures. Finally it is shown that the output of a bacterial community can be optimised in a model free way using only knowledge of microbial population levels and the rate of product output, showing that industrial bioprocess optimisation is a natural application of this technique.

In this work I have applied our method to a chemostat model. It could easily be applied in a range of other culture environments. Over the past several years, a number of low-cost bioreactors have been developed that can operate as both turbidostats or as chemostats [153, 154, 151, 152]. One such system has 78 chambers running in parallel; easily producing enough data to train the RL agent [151]. Another system incorporates online measurement of multiple fluorescence channels, facilitating state measurements at faster intervals than human sampling would allow [154]. Similar devices have been made at a smaller scale, using microfluidics capable of running batch, chemostat and turbidostat cell cultures [155, 156]. These have been applied to high-throughput gene analysis [155, 156], elucidating the relationship between population density and antibiotic effectiveness [157]. The development of a morbidostat facilitated the investigation into the evolution of resistance to antibiotics [158]. As these devices become more widely available, intelligent control methods could be used to explore these important topics while enabling additional layers of complexity, such as multiple competing species or environmental variation.

Here I adopted auxotrophy as our mechanism for control. The utility of this approach has been highlighted by previous studies of microbial communities [159, 160, 161, 63]. However, this is an unlikely approach to be used in industry due to the high cost of amino acids. Other methods of controlling strain growth or competitiveness could also be used as long as they can be externally controlled by the agent e.g. independent carbon sources [162], induced lysis [163] or growth arrest [164]. More traditional control methods such as the control of feedstocks, temperature, stirring rate or flow rate could also be used to apply reinforcement learning to an industrial application.

It should be noted that any attempt to control microbial populations may give rise to mutations. Because reinforcement learning approaches involve continual updates of the agent's policy, our method has the capacity to adapt to evolutionary changes in the growth dynamics. Understanding how to control populations of evolving species is crucial for preventing the development of antibiotic resistance [165] and the design of chemotherapy regimens [166]. Dynamic programming, the model-based analogue of reinforcement learning, has been used to solve the optimal control problem in both of these scenarios.

Overall, I have demonstrated the potential for control of multi-species communities using deep reinforcement learning. As synthetic biology and industrial biotechnology continue to adopt more complex processes for the generation of products from fine chemicals to biofuels, engineering of communities will become increasingly important. This work suggests that leveraging new developments in artificial intelligence may be highly suited to the control of these valuable and complex systems.

Chapter 6

Deep reinforcement learning for optimal experimental design in synthetic biology

6.1 Introduction

Synthetic biology has the aim of applying engineering principles to build biological systems from the ground up. A key part is the model based prediction and design of genetic circuits and other cellular systems. Biological systems are often complex, highly non-linear and noisy which makes obtaining accurate models challenging. Furthermore, characterisation experiments can be resource intensive and time consuming, making the optimisation of experimental effort important. The field of optimal experimental design (OED) uses mathematical techniques to identify experiments that will provide maximally useful characterisation data given a finite experimental capacity. This can be used to reduce the experimental effort required for accurate parameter inference when applied to complex biological systems. The difficulty with OED in biology is well known and has been tackled using Bayesian methods [167, 168] and methods based on the optimisation of the Fisher information [169, 170]. I aim to use reinforcement learning, a branch of machine learning, to develop a novel OED method for model parametrisation of biological systems. Deep reinforcement learning has been successfully applied to a similar class of problems within machine learning, called active learning. During active learning a learning agent can sequentially choose training examples to maximise its efficacy with the aim of data efficiency [171, 172]. The success of reinforcement learning to be applied to active learning motivates my application of it to OED and I develop a method complimentary to existing OED approaches. Previous work has shown the potential utility of the Fisher information even on non-linear systems [169, 170] and I build on this to use deep reinforcement learning to optimise the expected Fisher information given a model of a system. Like Bayesian methods we can learn on distributions over parameters. However, I exploit the ability of reinforcement learning to learn from samples to negate the need to approximate complex probability distributions. In place of this approximation is a, potentially lengthy, training process. However the training takes place prior to the experiment and results in an experimental controller that can be rapidly queried for experimental inputs. Here I focus on applications within synthetic biology, but the method is general and can be applied to any OED problem.

In this chapter I focus on time course experiments. Experiments are assumed to be constrained in both the total time and the number of experimental inputs that can be applied. Experiments are divided into a number of time intervals of equal duration, which I refer to as sample and hold intervals. At the beginning of each sample and hold interval an experimental measurement is taken and an experimental input is chosen that is applied to the system for the duration of the interval. An experimental design consists of a sequence of experimental inputs of length equal to the number of sample and hold intervals that constitute the experiment. I develop a control scenario in which the experimental inputs are controlled online by a reinforcement learning agent. For each sample and hold interval, the agent decides which input to apply and this constitutes the action space. I assume that potentially noisy measurements of a subset of the system variables can be taken and the agent makes decisions based on these measurements. The objective is to maximise the information gained with respect to obtaining accurate parameter estimates of a dynamical model of the system. A corresponding reward is given, such that the agent aims to maximise the determinant of the Fisher information matrix over an experiment.

6.2 Formulation of the optimal experimental design problem

Optimal experiments will be designed on systems which can be described by a set of non-linear differential equations:

$$\frac{d\mathbf{X}}{dt} = \mathbf{F}(\mathbf{X}, \theta, \mathbf{u}), \tag{6.1}$$

where $\mathbf{X}$ is a vector of system variables, θ is a vector of parameters and $\mathbf{u}$ is a vector of experimental inputs. It is assumed that system measurements $\mathbf{Y} = \mathbf{g}(\mathbf{X}) + \varepsilon$ can be taken, where $\mathbf{g}$ is a measurement function and ε is Gaussian noise. In general only a subset of the system variables may been observable. We intend to find an experiment that allows us to subsequently obtain an accurate estimate of the unknown system parameters, θ. We define our objective as the determinant of the Fisher Information matrix, I, this is called D-optimal design. Although the theory of D-optimal design only holds exactly for linear systems with Gaussian errors, previous work has shown that it can be successfully applied to non-linear systems [169, 170] and we follow their approach to obtain I from the system equations (Equation 6.1). First we obtain time derivatives for the sensitivity of each of the outputs with respect to each of the parameters:

$$\frac{d}{dt}\frac{\partial \mathbf{X}}{\partial \theta_j} = \frac{\partial \mathbf{F}}{\partial \theta_j} + \frac{\partial \mathbf{F}}{\partial \mathbf{X}}\frac{\partial \mathbf{X}}{\partial \theta_j}. \tag{6.2}$$

The scale of parameters can vary, which can lead to poor conditioning of the Fisher information matrix [31], to remedy this, the sensitivities are scaled by the parameter values, which is equivalent to using logarithmic sensitivities. The scaled sensitivities are

$$\bar{\mathbf{X}}_{\theta_j} = \frac{\partial \mathbf{X}}{\partial \log(\theta_j)} = \theta_j \frac{\partial \mathbf{X}}{\partial \theta_j}.$$

Writing Equation 6.2 in terms of the scaled sensitivities results in

$$\frac{d\bar{\mathbf{X}}_{\theta_j}}{dt} = \theta_j \frac{\partial \mathbf{F}}{\partial \theta_j} + \frac{\partial \mathbf{F}}{\partial \mathbf{X}} \bar{\mathbf{X}}_{\theta_j}.$$

We assume that measurement error, ε, is normally distributed with variance equal to 5% of the measured quantity: $\sigma^2 = 0.05\mathbf{X}$ $\varepsilon = \mathcal{N}(\mathbf{X}, \sigma^2)$ and we assume no covariance between measurements. The time derivative of the scaled FIM can be written as [33]

$$\frac{d}{dt} I_{jk}(\mathbf{u}, \theta, t) = \bar{\mathbf{X}}_{\theta_j} \Sigma^{-1}(t) \bar{\mathbf{X}}_{\theta_k},$$

where

$$\Sigma(t) = \begin{bmatrix} \sigma_1^2 X_1 & & \\ & \ddots & \\ & & \sigma_n^2 X_n \end{bmatrix}$$

is a diagonal matrix. The Fisher information can then be integrated over a whole experiment, assuming that $I(t=0) = 0$

$$I_{jk}(\mathbf{u}, \theta, 0, t_f) = \int_0^{t_f} \bar{X}_{\theta_j} \Sigma^{-1}(t) \bar{X}_{\theta_k} dt.$$

Because the Fisher information can vary over many orders of magnitude the overall optimality score (D optimality) is taken to be

$$\Theta_D(\mathbf{u}, \theta, 0, t_f) = \log(|I|). \tag{6.3}$$

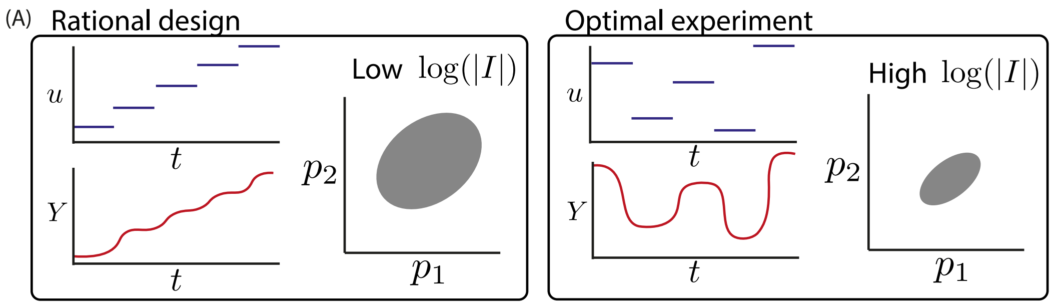
The Python API of the CasADi library was used to solve all differential equations in this chapter [118].

6.3 Fitted Q-learning for optimal experimental design

I imagine a scenario in which a reinforcement learning agent is trained on a dynamical model of a system of interest for a number of episodes. Each episode constitutes a single experiment and is constrained in terms of the total experimental time and partitioned into a number of equal time intervals. For each interval an experimental input can be applied. Using the model the reinforcement learning agent can train over repeated episodes with the goal of maximising the logarithm of the determinant of the FIM over an experiment (Equation 6.3). After training, the resulting agent can be used as an experimental controller, referred to as the RL controller, and applied to the real system to dynamically control an experiment. D-optimal design is equivalent to minimising the confidence ellipsoid of the resulting parameter estimates. Figure 6.1A shows the difference between a rationally designed experiment with low D-optimality and a large confidence ellipsoid for the parameters (left) and an optimal experiment with a high D-optimality score (right). The optimal experiment will reduce the uncertainty of the parameters with no additional experimental effort.

To formulate the OED problem in the reinforcement learning framework we need to consider the agent's environment, the states observed and actions taken by the agent, as well as the reward function that guides the agent towards optimal behaviour. The environment the agent interacts with is a model of an experimental system governed by a set of differential equations (Equation 6.1). The state that the agent observes is composed of the current system observables, $\mathbf{Y}(t)$, an estimate of the current unique elements of the FIM, $I(t)$, and the current sample and hold interval, t. The action taken by the agent at each time step is the experimental input that will be applied over the next sample and hold interval $\mathbf{u}(t)$. The reward given to the agent is the change in the logarithm of the determinant of the Fisher information matrix over the last sample and hold interval $\Delta\log(|I|)$. This means that over an experiment, the return is equal to the log of the determinant of the Fisher information matrix and that the agent's optimisation objective is equivalent to D-optimal design. This overall picture is summarised in Figure 6.1B.

First I apply the Fitted Q-learning algorithm developed in Section 2.2.1 to OED on two systems to validate the approach. These systems constitute a simple model described by Monod growth dynamics and a biologically realistic system of gene transcription. I then propose and develop two modifications to the algorithm. First,



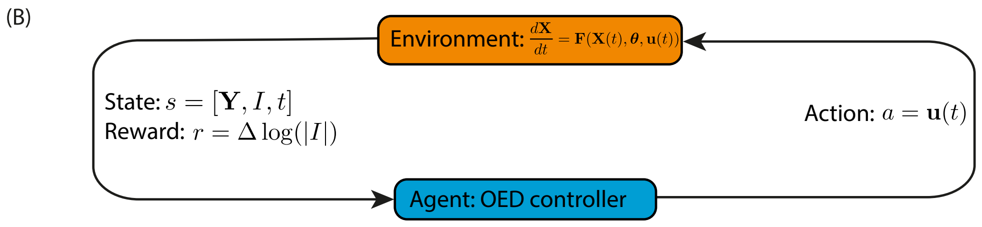

(B)
Environment: $\frac{d\mathbf{X}}{dt} = \mathbf{F}(\mathbf{X}(t), \boldsymbol{\theta}, \mathbf{u}(t))$
State: $s = [\mathbf{Y}, I, t]$
Reward: $r = \Delta \log(|I|)$
Action: $a = \mathbf{u}(t)$
Agent: OED controller

Figure 6.1: Reinforcement learning for optimal experimental design. A) An optimal design that maximises the determinant of the Fisher information matrix will minimise the confidence ellipsoid of the resulting parameter estimates. B) Optimal experimental design formulated as a reinforcement learning problem.

the use of a recurrent neural network to make decisions using the entire experimental trajectory and, secondly, the extension into continuous action spaces. Applying these two modifications results in the Recurrent T3D algorithm. I then show the effectiveness of this algorithm on the design of experiments for the growth of a bacterial culture inside a chemostat, a system of interest for the synthetic biology community and for industrial bioprocessing applications.

6.3.1 Reinforcement learning for optimal experimental design on a simple Monod growth system

To test the potential of using reinforcement learning for OED I first apply the Fitted Q-learning algorithm to a simple non-linear system with Monod dynamics. I also compare the performance of the RL controller with a one step ahead optimiser (OSAO) (see Section 2.4 for details). The OSAO uses the model of the system to predict the experimental input which would greedily maximise the determinant of the FIM over the next sample and hold interval.

In this system we have one state variable, x, and no measurement noise so that output $\mathbf{Y} = x$. There is one control variable u that is controlled by the OSAO or the

RL controller. The dynamics of the system are given by a simple Monod relationship between $\frac{dx}{dt}$ and u:

$$\frac{dx}{dt} = \frac{p_1 u}{p_2 + u} x,$$

where p_1 and p_2 are the parameters to be estimated.

In the following $p_1 = p_2 = 1$, both the RL and OSAO implementations start from initial condition $x_0 = 1$, $u_0 = 0.5$ and are able to choose u between $0 \leq u \leq 0.1$. The Fitted-Q agent works in a discrete action space and therefore has ten equally spaced discrete actions to choose from. The choice of discretisation scheme and the upper and lower bounds must be made for each system and a bad choice could have a negative effect on performance. In a practical application the upper and lower bounds are likely to be dependent on the experimental setup and sometimes the discretisation scheme will be also. If the experimental setup allows for continuous inputs then it must be discretised according to the experimenters knowledge. In this case I equally space the possible input choices linearly, which I suggest is a good default choice, but other methods such as equally spaced inputs on the log scale will likely be appropriate for other systems and I make use of this in later sections. In Figure 6.2A we can see that the experimental input profiles chosen by the RL controller and OSAO are similar, choosing between the highest u available and a value about half way between the maximum and minimum u value. The RL controller picks two u values either side of the optimum found by the OSAO. This is likely due to the discretisation of its action space which means that it can't precisely choose the optimum value. Figure 6.2B shows the system trajectories of both controllers, as expected these are very similar. Figure 6.2C shows the performance of the RL controller as it was trained for 500 episodes, compared to the performance of the trajectory found by the OSAO. This shows that their performance is extremely similar, at the end of training the RL controller is performing slightly better than the OSAO.

To understand these results and further verify the efficacy of the RL controller I derived an analytical expression for the determinant of the FIM resulting from

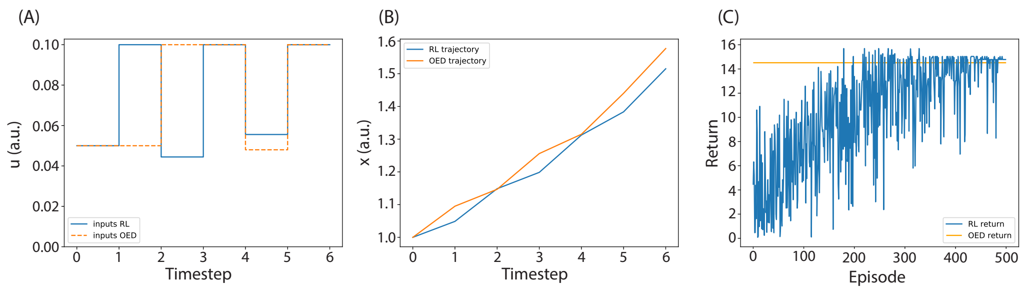


Figure 6.2: Reinforcement learning for optimal experimental design on a simple non-linear system. (A) The input profile of the two agents, (B) The system trajectory (C) The training performance of the RL agent compared to the performance achieved by the OSAO

an experiment on the Monod system (see Appendix B for the full derivation). An expression for the determinant of the FIM can be written as

$$|I| = \int_0^{t_i} \frac{u(t)^2}{(p_2+u(t))^2} x^2 dt \int_0^{t_i} \frac{p_1^2 u(t)^2}{(p_2+u(t))^4} x^2 dt - \left(\int_0^{t_i} \frac{p_1 u(t)^2}{(p_2+u(t))^3} x^2 dt \right)^2 . \quad (6.4)$$

For the ease of visualisation, an approximate expression for an experiment composed of two sample and hold intervals and therefore two experimental inputs (u_1 and u_2) can be written as

$$\begin{aligned} |I|(u_1,u_2) \approx \\ p_1^2 u_1^2 u_2^2 \Bigg(& \frac{1}{(p_2+u_1)^4(1+u_2)^2} + \\ & \frac{1}{(p_2+u_1)^2(p_2+u_2)^4} - \\ & \frac{2}{(p_2+u_1)^3(p_2+u_2)^3} \Bigg) . \end{aligned} \quad (6.5)$$

This expression is an approximation as I have ignored the x^2 that all the integral terms depend on, which allows us to easily calculate $|I|$ for different values of u_1 and u_2. The effect of this approximation is to alter the scaling between u_1 and u_2 as x increases monotonically in our simulations. Equation 6.5 was plotted for ranges of values for u_1 and u_2 between 0 and 0.1 (Figure 6.3A). Here we see that the experiment

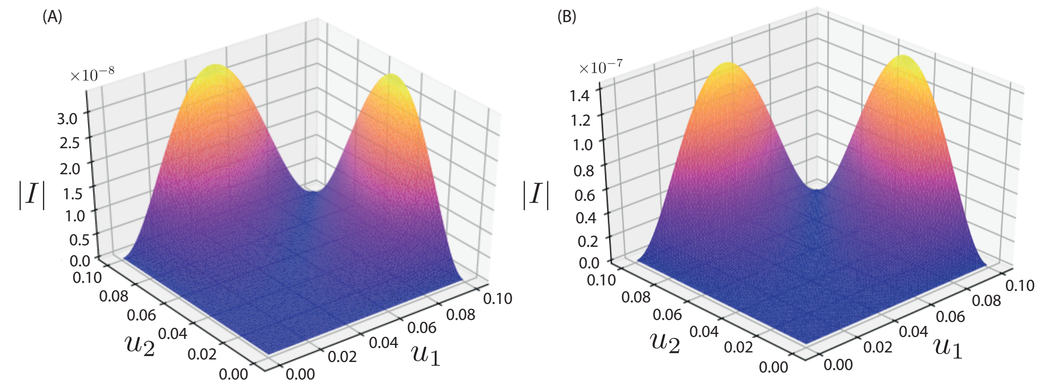


Figure 6.3: Analytical derivation of Fisher information. Determinant of I according to Equation 6.5 (A) and Equation 6.4 (B) as a function of the two inputs u_1 and u_2

that maximises $|I|$ has one u at its maximum and one u at an intermediate value at roughly half of the upper bound on u. This result corroborates the behaviour of the OSAO and RL controllers. To check that this result was not significantly changed by the removal of the x^2 factors the full system was simulated and I according to Equation 6.4 was calculated for the same region in u space (Figure 6.3B). This shows a very similar result, with the expected difference in scaling between u_1 and u_2. These results validate the solutions found by both agents and that the optimal experimental design setup is working as intended (Figure 6.3A was made independently of any of this code). Importantly, this shows that enough information is provided to the agent through the state and the reward to choose optimum experimental inputs.

6.3.2 Reinforcement learning for optimal experimental design on a model of gene transcription

As an example of a system of realistic biological complexity that has been used in previous OED research [169], the method was tested on the inference of the ‘intrinsic’ parameters of a genetic construct. The model equations are:

$$\frac{d}{dt}\frac{X_{rna}}{V} = \alpha\frac{g}{V}\frac{\frac{P_a}{\eta G}K_r + \frac{P_a K_{rt}}{(\eta G)^2}u(t)}{1+\frac{P_a}{\eta G}K_r + (\frac{K_t}{\eta G}+\frac{P_a K_{rt}}{(\eta G)^2})u(t)} - \delta\xi\frac{X_{rna}}{V}$$

$$\frac{d}{dt}\frac{X_{prot}}{V} = \frac{\beta\frac{R_f}{V}}{K_M + \frac{R_f}{V}}\frac{X_{rna}}{V} - \lambda\frac{X_{prot}}{V}$$

where X_{rna} and X_{prot} are the concentration of RNA and protein, $\alpha, K_r, K_t, K_{rt}, \delta, \beta$ and K_M are intrinsic parameters, V, g, P_a, G, R_f and λ are growth rate dependent parameters and η and ξ are fixed. The distinction is made between these intrinsic parameters, which govern the genes induction behaviour, and the physiological parameters which are growth-rate-dependent and reflect the state of the host cell. In this work I focus on the inference of the intrinsic parameters, omitting K_r from the set of parameters to be inferred as it has previously been found practically unidentifiable [169]. The control inputs are the transcription factor copy number, u, and the levels of both protein and mRNA, X_{rna} and X_{prot}, are measured.

Experiments were designed using rational design, the OSAO and RL controller. The rational design represents an experiment that might be chosen by a human designer. An experiment lasts 600 minutes and is divided into 6 intervals of 100 minutes, for each interval a different u can be chosen. The OSAO could choose u continuously from within the range $0 < u < 1000$, the RL agent could choose between 12 discrete values for u equally spaced on the log scale where $-3 < \log(u) < 3$. The RL controller was trained on 20000 episodes. Because the optimality of an experiment designed using the D-optimality score is only guaranteed for linear systems we follow the approach taken in previous work [169] to test the correspondence of a high optimality score with low uncertainty in resulting parameter estimates. To test an experimental design the co-variance of 30 resulting independent parameter estimates was calculated. To obtain the parameter estimates, 30 independent trajectories were generated using each experimental design with measurement noise equal to $\sigma^2 = 0.05\mathbf{X}$, where $\varepsilon = \mathcal{N}(\mathbf{X}, \sigma^2)$. Using the CasADi library [118] each noisy experimental trajectory was used to infer the intrinsic system parameters, where the

Intrinsic parameter	Value	Minimum	Maximum	Unit
α	20	1	30	min^{-1}
K_t	5×10^5	2×10^3	1×10^6	AU
K_{rt}	1.09×10^9	4.02×10^5	5.93×10^{10}	AU
δ	2.57×10^{-4}	7.7×10^{-5}	7.7×10^{-4}	$\mu m^{-3}\text{min}^{-1}$
β	4.0	1	10	min^{-1}
K_M	750	750	1500	μm^{-3}
Physiological parameter	Value			Unit
g	5.7			AU
P_a	4000			AU
G	4.3			AU
η	900			$\mu m^{-3}\text{min}^{-1}$
R_f	7000			AU
V	2.24			μm^{-3}
λ	3.5×10^{-2}			min^{-1}

Table 6.1: The intrinsic parameters of the gene transcription system. Parameter values used for simulations of an inducible gene transcription system and their minimum and maximum bounds when sampling from a distribution.

	Rational	OSAO	RL
Optimality score	67.73	71.00	73.26
$\log\|cov(\theta)\|$	66.96	34.17	25.72

Table 6.2: Summary of the performance metrics of different methods for optimal experimental design on the gene transcription model

initial values of the parameters were sampled from a uniform range (Table 6.1). The covariance of these resulting parameter estimates was calculated, where experiments with higher optimality scores should lead to a lower parameter covariance.

The optimality score and the resulting logarithm of the determinant of the covariance matrix of the parameters is shown in Table 6.2. We can see that the RL controller outperformed both the rational design and the OSAO in terms of the optimality score. Furthermore, there is good agreement between the optimality scores and the co-variance in the parameter estimates, which confirms that maximising the D-optimality reduces the uncertainty of the parameter estimates on this non-linear system. The experimental inputs and resulting trajectories of the three experimental designs are shown in Figure 6.4. The experiments designed by the OSAO and the RL controller show similarities, which is not unexpected when optimising the same

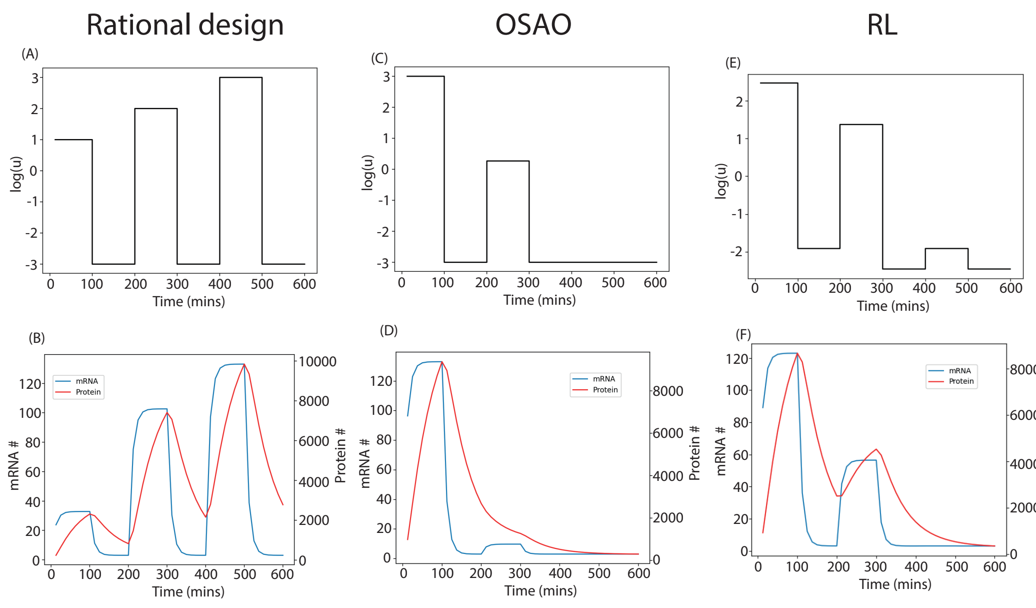


Figure 6.4: Optimal experimental design for parameter inference on a gene transcription system. Experiments design using rational design (A), one step ahead optimisation (C) and reinforcement learning (E) and the corresponding system trajectories (B), (D), (F) respectively.

objective using two different methods. Figure 6.5 shows the training performance of 3 independent RL controllers. We see that the resulting performance at the end of training is consistently high compared with the rational design and OSAO. However there is some difference in the performance in the intermediate portions of training. One of the repeats adopted a bad policy at around episode 14000 before converging to a good policy at the end of training.

6.4 A continuous, recurrent RL controller for OED for bacterial growth

The results in the previous sections show that reinforcement learning has promise for the optimal design of experiments on systems with realistic biological complexity. In the subsequent sections I develop the reinforcement learning algorithm further and propose a method of practical application.

Inserting new genes into a bacterial strain can alter the growth dynamics by inflicting a metabolic burden on the cells, or other unintended effects such as cross

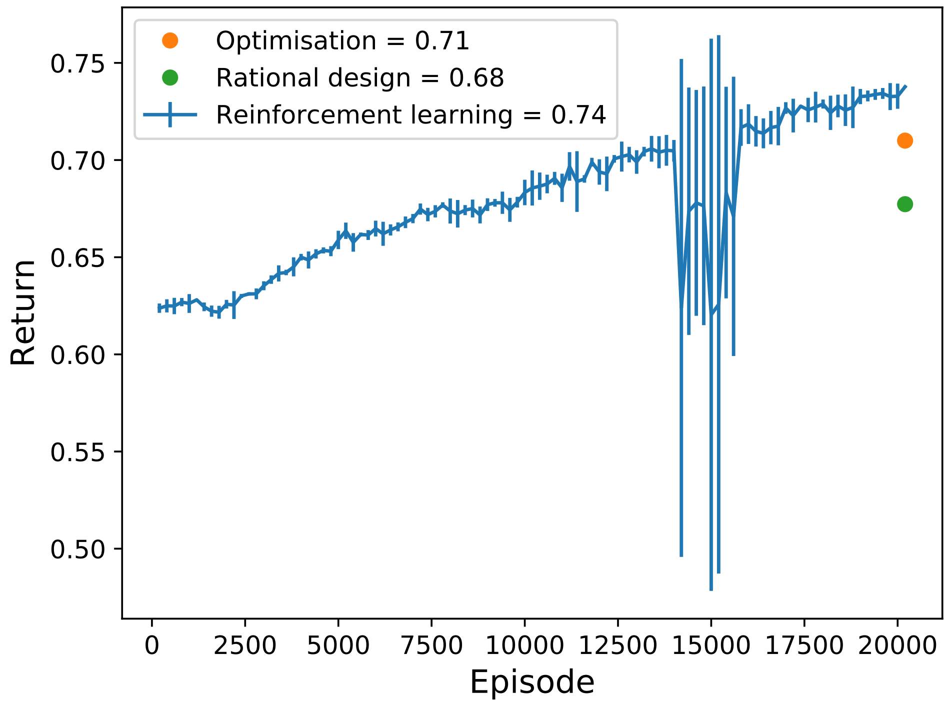


Figure 6.5: Average return of three Fitted Q-agents over 20000 episodes and the scores of the one step ahead optimisation and rational design applied to a gene transcription system. For interpretability the returns for each of the 20000 episodes were split into bins of 200 and the average of each bin was taken. This was then averaged across the 3 repeats.

talk between regulatory pathways. This makes the characterisation of the growth characteristics of a, potentially engineered, strain of bacteria in liquid culture of interest to the field of synthetic biology. One way this can be done is the monitoring of bacterial growth under different nutrient conditions in a bioreactor. Using cheap tabletop bioreactors this is a setup that is feasible for most synthetic biology labs. I investigate the parameter inference of bacteria growing in a chemostat where the bacteria has been engineered to be an auxotroph. An auxotroph is a bacterial strain that has is reliant on a specific nutrient, often an amino acid, for growth.

I model the system where the agent controls the concentration of carbon source, C_0, and auxotrophic nutrient, C_1, for an auxotroph growing in a chemostat (Figure 6.6 A,B). This model is the same the one used in Chapter 5 used to model microbial communities, but in this chapter only a single bacterial strain is considered. This

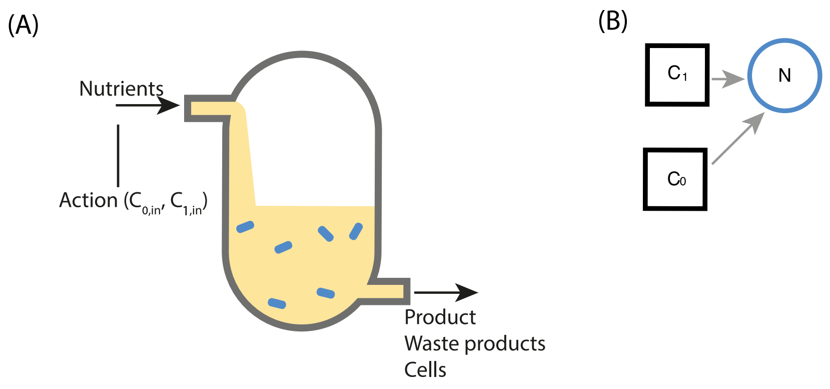


Figure 6.6: Description of the chemostat system. (A) Diagram of a chemostat. The actions taken by the agent control the concentration of nutrients flowing into the reactor. (B) System of an auxotroph dependent a carbon source and an amino acid.

model simulates the growth of one *E. coli* strain in a continuous bioreactor. The strain has been engineered so that it is an auxotroph and is reliant on the amino acid arginine to grow, in addition to a carbon source. Both of the nutrients can be controlled during an experiment by controlling the flow of nutrients into the chemostat, C_{in} and $C_{0,in}$, so $\mathbf{u}(t) = [C_{in}(t), C_{0,in}(t)]$. The population of the bacteria, N, is the only measured system variable and C_1 and C_0 are hidden variables, so $\mathbf{Y}(t) = N(t)$. The system equations are:

$$\frac{d}{dt}C_0 = q(C_{0,in} - C_0) - \frac{1}{\gamma_0}\mu N$$
$$\frac{d}{dt}C_1 = q(C_{1,in} - C_1) - \frac{1}{\gamma}\mu N$$
$$\mu = \mu_{max}\frac{C_1}{K_s + C_1}\frac{C_0}{K_{s0} + C_0}$$
$$\frac{d}{dt}N = (\mu - q)N$$

where μ_{max} is the maximum growth rate, K_s is the half-maximal auxotrophic nutrient concentration, K_{s0} is the half-maximal concentration for the carbon source and N is the concentration of bacterial cells in the chemostat which is assumed to be measurable using optical density or fluorescence techniques.

The aim of this section is to design an experiment to infer the growth parameters

Parameter	Value	Minimum	Maximum	Unit	Source
q	0.5			h^{-1}	Experimentally controllable
γ_0	4.8×10^{11}			cells g^{-1}	[145]
γ_1	5.2×10^{11}			cells g^{-1}	[146]
μ_{max}	1	0.5	2	h^{-1}	[147]
$\mathbf{K}_{s,0}$	6.8×10^{-5}	10^{-5}	10^{-4}	g L^{-1}	[146]
$\mathbf{K}_{s}$	4.9×10^{-4}	10^{-4}	10^{-3}	g L^{-1}	[146]

Table 6.3: Parameters for the single auxotroph system. Parameter values used for simulations of a system consisting of a single population of bacteria and their minimum and maximum bounds when sampling from a distribution.

of this strain. Each episode consists of one twenty-hour experiment in which experimental inputs are chosen every two hours. This means that the controllers choose ten different inputs over the course of an experiment. The initial system variables of the chemostat for each episode were $N = 20 \times 10^9$ cells L^{-1} and $[C_0, C_1] = [1, 0]\ g\ L^{-1}$. The parameters in Table 6.3 were used. It was found that the parameters γ_0 and γ were practically unidentifiable (Appendix C) and so these were set to their true values throughout. This means that in this section we are designing experiments to infer 3 parameters: K_{s0}, K_{s1} and μ_{max}. Throughout, the experimental inputs C_{in} and $C_{0,in}$ have the minimum and maximum bounds of $0.01 g/l$ and $1 g/L$ respectively.

6.4.1 Removing the dependence on a priori parameter estimates

The FIM is a function of the parameters of the system and so the experiment that maximises the determinant of the FIM is dependent on these parameters. So far all experimental designs have been done with a priori knowledge of the true parameter values. The OSAO has access to the true model to make predictions and the RL controller uses the true parameter values to calculate the elements of the FIM that make up part of its observed state. This is obviously an unrealistic scenario and in this section I investigate methods to remove this dependence for the RL controller. Designing and analysing experiments using true parameter values is common in the field of OED [169, 170]. The effectiveness of experiments experiments designed with parameters drawn from a distribution sufficiently close to the true values is sometimes subsequently verified [169]. However, there is no guarantee that the experiment designed using any given sample from a parameter distribution will perform well for the true parameters, especially with limited prior knowledge, and I view this as a limitation of existing approaches.

The first potential solution is taken from the OED literature [173, 174, 175]. It is to use an initial parameter guess and iteratively improve this guess as the experiment progresses. Using this method an initial guess of the parameters would be taken and after each sample and hold interval the parameter guess would be updated by fitting the model to the data gathered so far. This technique has previously been discussed in literature [173] and used as part of online OED algorithms on fed-batch bioreactors [174, 175]. I found that using CasADi to do this accurately estimates the parameters. However, it was prohibitive in terms of computation time and integrating this into training would increase training time by several orders of magnitude. I also tried various methods from the Scipy library, these are faster, however the majority of the time they converge to local parameter optima, which give low error in terms of predicting the data but do not actually correspond to the true parameters. Furthermore, the performance of the resulting experimental design would likely be dependent on the initial parameter guess, which could lead to inconsistent results. These factors lead me to conclude that this is not a viable option.

The second potential method leverages the capabilities of deep reinforcement learning so that the agent can learn to adapt its experiment based on previous system behaviour. This involves changing the state observed by the agent by removing the elements of the FIM, which require parameter estimates to calculate, and replacing them with the timeseries of previous experimental measurements and inputs. The resulting timeseries for any given sequence of actions depends on the true parameters of the system. Therefore, I hypothesise that the agent should be able to infer where it is in parameter space given the full experimental trajectory of states and actions leading up to the current sample and hold interval and adapt its future actions accordingly. To verify this I tested the ability of three different agents to learn a value function. The three agents choose from the same set of actions and are rewarded in the same way as throughout, however they differ in the state information that is available to them. The first agent uses the original state composed of the system observables, elements of the FIM and the current timestep. The second agent's state is composed of only the current observables and timestep, removing the dependence on parameter guesses but not giving the agent any additional information. The third agent observes the current observables, the full trajectory of past observables and past actions taken during the current experiment and the current timestep. The third agent has substituted the FIM elements for a full experimental trajectory in the hope that it can infer where it is in parameter space using the full trajectory. Each agent was evaluated by testing its ability to fit its value function to effectively predict the returns from a corresponding sequence of states and actions. The value of a state-action pair is defined as the expected return after taking the given action while in the given state and so a good value function will shown low, but not necessarily zero, error in predicting returns. First, a training set of 1000 experiments was generated by sampling 1000 different chemostat environments from a uniform distribution over parameters (see Table 6.3 for the minimum and maximum bounds). For each environment 10 random experimental inputs were applied, sampled uniformly from within the bounds on C_{in} and $C_{0,in}$. An independent testing set was also generated in the same way. Each experiment was simulated and the return was calculated for each state action pair and the value functions of each of the three agents were fitted to the training data. This process is illustrated in Figure 6.7A. Effectively this is testing each agent's ability to learn the value function of the random policy applied to chemostat environments sampled from the uniform distribution over parameters. Because the parameters are different for each experiment the agent must learn to infer where it is in parameter space to accurately predict the returns of experimental inputs. The results, in terms of mean square error of return estimates over the training and testing set, are shown in Figure 6.7. Figure 6.7B shows the performance of the first agent that has the dependence on the true parameter values and acts as a baseline for comparison. Figure 6.7C shows the performance of the second agent with only knowledge of the current state and timestep, we can see in this case that the agent struggles to learn the values accurately, likely because it has no way of inferring where it is in parameter space for any given parameter sample. Figure 6.7D shows the third agent with knowledge of the full system trajectory. We can see that at the end of training performance is only slightly worse than the agent with prior knowledge of the true parameter values. This implies that, using the full system trajectory, the agent can infer where it is in parameter space and alter its value predictions accordingly. The result of this is that we can sidestep the dependence on parameter guesses during decision making by changing the agent's state. It should be noted that the reward signal is still dependent on the unknown parameter values. However, unlike the state, the calculation of the reward is not required when deploying a trained agent on a real system and I develop this idea further in Section 6.4.4.

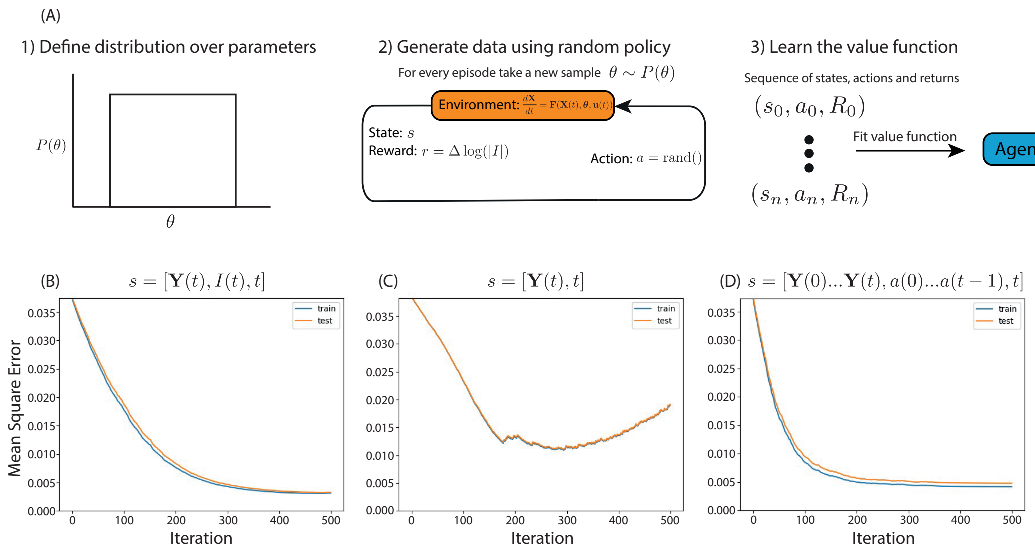


Figure 6.7: The performance of agents using three different states to predict values of the random policy. (A) Outline of the testing method; 1) the distribution of parameters was chosen as a wide flat distribution, 2) data was generated on chemostats sampled from the distribution using the random policy, 3) agents were trained to predict the observed returns with three different states. (B) State composed of the current observables, the true elements of the Fisher information matrix and the current timestep (C) State composed of the current observables and the current timestep (D) State composed of the current observables, the full system trajectory of states and actions so far and the current timestep. The agent in (D) is able to accurately predict values with no a priori knowledge of the parameters.

6.4.2 Continuous, recurrent reinforcement learning to design experiments

Further to the addition of the full experimental timeseries I also anticipate that learning in a continuous action space will be beneficial. There are three reasons for this. Firstly, in the experimental setups we consider here the inputs are continuous, therefore an agent with continuous actions will have more precise control over the experimental inputs it can apply. This could improve the overall quality of our experimental designs. Secondly, there will be no need for the experimenter to choose a discretisation scheme, which could lower the agent's effectiveness if done poorly. Thirdly, in a discrete action space the number of available actions grows to the power of the number of different experimental inputs. For example, in the case previously considered we have a single experimental inputs with 12 possible values. Adding a second experimental input also with 12 possible values would give $12 \times 12 = 144$ possible input combinations to choose from. A third experimental input would give 1728 and so on. Q-learning in such large action spaces results in a difficult learning problem [176]. In a continuous action space only a single continuous value is required to be learned for each experimental input, meaning that the method could easily be applied to systems with many experimental inputs. If an experimental setup that uses discrete inputs was required the original discrete Q-learning method could be used or, if the resulting discrete action space was too large to learn effectively on, the continuous experimental input from the agent could be binned into a discrete value. Incorporating the binning into training would mean that the agent could be 'aware' of the binning while it learns and I anticipate that this should not effect performance.

To incorporate the recurrent neural network and the transition to a continuous action space I developed a recurrent version of the continuous reinforcement learning algorithm Twin Delayed Deep Deterministic Policy Gradient (T3D) [108] (see Section 2.2.2 for details). The internal structure of the agent is shown in Figure 6.8A. The main components of the agent are the memory, value network and policy network. During each sample and hold interval states, s, and rewards, r, are observed

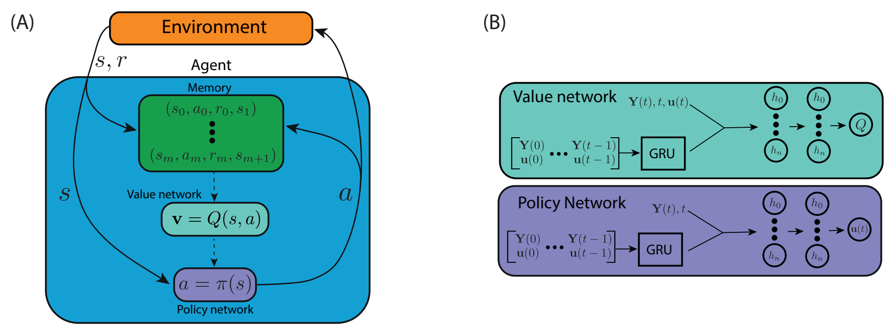


Figure 6.8: The Recurrent T3D algorithm. A) Structure of the T3D agent. B) Network diagrams of the artificial neural networks used to approximate the value function (top) and policy (bottom)

from the environment and stored in memory. The state also acts as input for the policy network which chooses the action, a, to apply to the environment and this action is also stored in memory. The agent's value and policy functions are periodically updated by training on the experience stored in the memory (dotted arrows). The value function is approximated using a deep neural network (Figure 6.8B - top). This is composed of a gated recurrent unit (GRU) layer which takes as input all system measurements, $\mathbf{Y}$, and experimental inputs, $\mathbf{u}$, seen so far in the current experiment. The GRU cell is a type of recurrent layer which excels at processing sequences such as time series data. The output of the GRU is concatenated with the current system measurements, $\mathbf{Y}(t)$, the current time step, t, and the current action, $\mathbf{u}(t)$. These are fed into a feed forward network composed of two hidden layers, each with 100 neurons. The output of the value network is an estimate of the value of the supplied experimental input, $\mathbf{u}(t)$, in the given system state. The policy network is shown in network (Figure 6.8B - bottom). This is composed of a recurrent GRU layer which takes as input all observable states and experimental inputs seen so far in the current experiment. The output of the GRU is concatenated with the current state, composed of the current observable system measurements and the current time step, and fed into a feed forward network composed of two hidden layers, each with 100 neurons. The output of the policy network is the experimental input that is estimated to have the highest value for the given system state.

Because training a RL controller takes place over a large number of independent episodes, it is possible to gather experience from multiple episodes in parallel. I took advantage of this by running experimental simulations in parallel using the functionality provided by the CasADi library. This means that all of the computationally demanding aspects of the training process are parallelised. The training of the neural networks using TensorFlow is implemented in parallel and can be run on a GPU if available or a multicore CPU. The simulation of the experimental system using CasADi is parallelised and can take advantage of a multicore CPU. In the following section I ran experimental simulations in batches of 10 parallel simulations, but this number could be increased to take advantage of more computing resources. The average training time was 11.73 hours with standard deviation 0.91 hours over 20 total training runs on a computing cluster with 40 CPU cores and a GeForce GTX 1080 Ti. To asses the training time on a smaller scale personal computer, a single agent was trained on a 13-inch MacBook pro with a 2GHz Quad core Intel i5 processor with no GPU and the training time was 15.48hrs.

6.4.3 Optimal experimental design for a single auxotroph in a chemostat

OED was done for the single auxotroph system with the full Recurrent T3D algorithm using the true parameters in Table 6.3. The RL controllers were trained for 17500 episodes. Because the RL controller now makes decisions based on a full timeseries I also developed a Model Predictive Controller (MPC) which works similarly to the OSAO in that it uses the model of the system to predict which experimental input would maximise the determinant of the FIM. However, the MPC optimises over the full experimental trajectory (see Section 2.4 for details).

The experimental input profiles and resulting system trajectories for the rational design, OSAO, MPC and RL controllers are shown in Figure 6.9. There is less similarity in the experiments designed using the different controllers, the MPC and RL controllers have achieved similar optimality scores with very different experiments. Figure 6.10 shows the training performance of ten RL controllers and the equivalent performance of the optimisation and rational designs. The training performance is

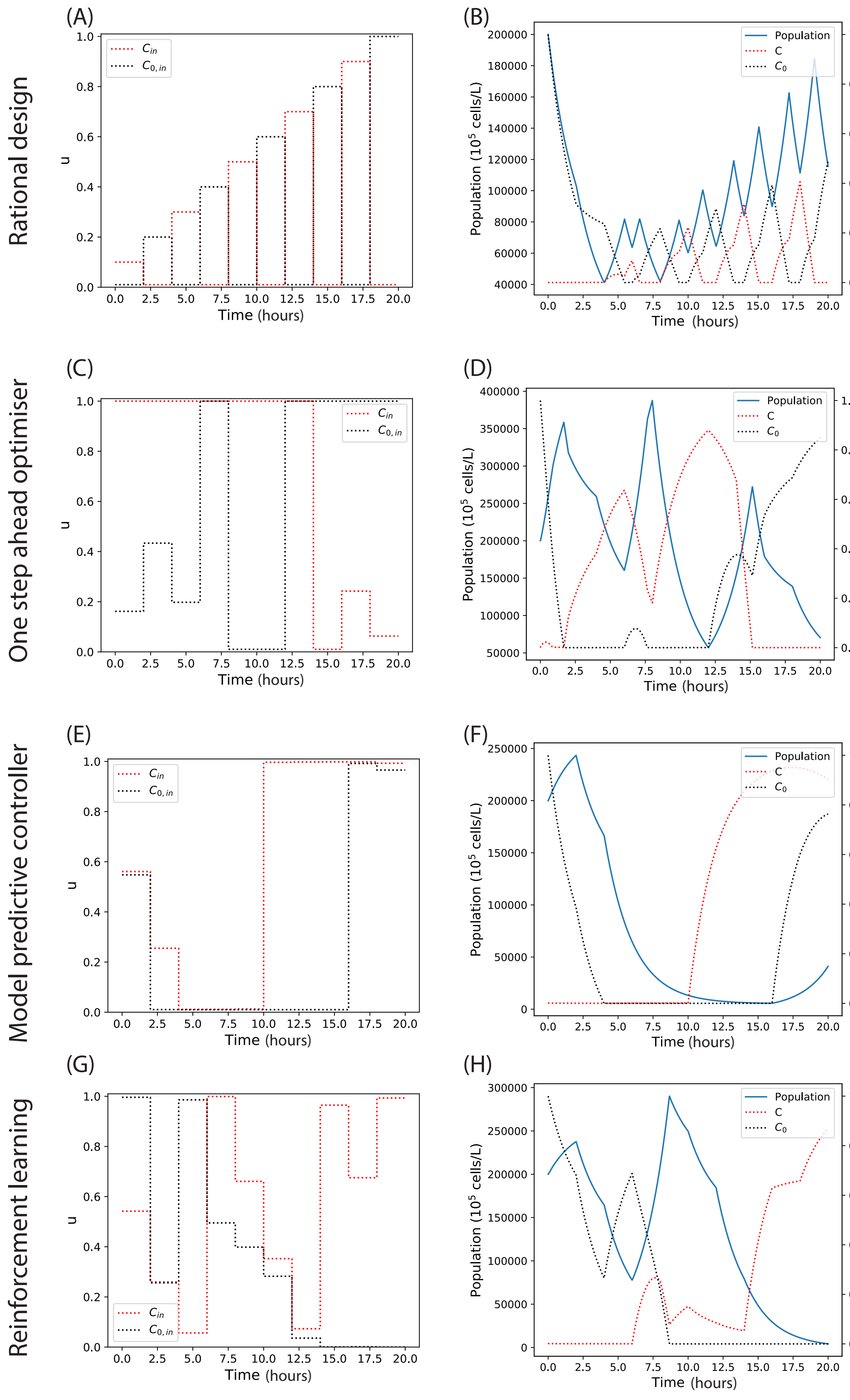


Figure 6.9: Optimal experimental design to infer parameters of a single auxotrophic bacteria growing in a chemostat. Control inputs chosen by rational design (A), one step ahead optimisation (C), model predictive control (E) and reinforcement learning (G) and the corresponding system trajectories (B), (D), (F) and (H) respectively.

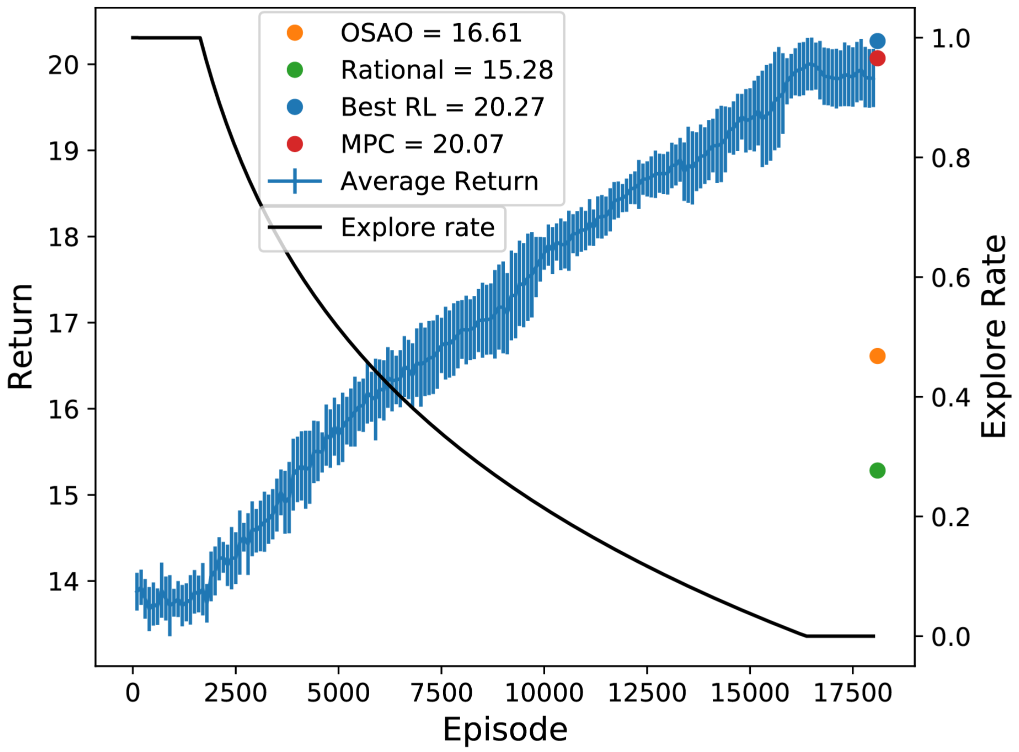


Figure 6.10: Training performance of reinforcement learning for optimal experimental design on the chemostat system. Average training progress of ten Recurrent T3D agents over 17500 episodes and the scores of the MPC, OSAO and rational design. For interpretability the returns for each of the 17500 episodes were split into bins of 100 and the average of each bin was taken. This was then averaged across the 10 repeats.

stable with only minor differences between each of the 10 independently trained RL controllers. We can see that the MPC and RL controllers significantly outperform the other controllers and that the MPC and RL perform very similarly to each other. The MPC reaches a slightly higher optimality score than the mean of the RL controllers, but the best RL controller performs better than the MPC. From these results I can conclude that the recurrent RL controllers are performing at a similar level to an MPC controller that has explicit prior knowledge of the true parameters. The methods are further compared by repeated parameter fittings to noisy data generated by each experimental design. For each method, 30 experiments are carried out and 30 independent parameter estimates are done. The total sum square error in the parameter estimates, normalised by the true parameter values, and the logarithm

	Rational	OSAO	MPC	Best RL
Optimality score	8.43	16.6	20.07	20.27
Total normalised error	33.0	26.7	6.1	3.4
$\log\lvert cov(\theta)\rvert$	-3.76	-5.47	-8.72	-11.85

Table 6.4: Summary of the performance metrics of different methods for optimal experimental design on the single auxotroph model

of the determinant of the covariance matrix of the parameter estimates is shown (Table 6.4). For this the best performing RL agent was used. Again these results show that a high optimality score is a good predictor of a low determinant of the covariance matrix and here we see that this also corresponds to low error in the inferred parameters. As expected, according to the two additional metrics the MPC and RL controllers have greatly outperformed the rational design and the OSAO and the performance of the best RL agent is slightly better than the MPC.

6.4.4 Reinforcement learning can be used to optimise over a parameter distribution

One of the key advantages of the reinforcement learning approach is the trivial extension to optimising over a distribution of parameters. Reinforcement learning can train by simply drawing a new sample from a distribution for each episode. This is similar to a technique used in reinforcement learning called domain randomisation [177]. Domain randomisation involves perturbing the behaviour of a simulated system during training so that the learned policy is less brittle with respect to any potential error in the simulation model when applied in a real situation. Here I employ a very similar method, but I use the distribution to capture any prior knowledge of the parameters in question. If there is significant prior knowledge of the system this can be reflected by sampling parameters from a narrow distribution such as a Gaussian with low standard deviation. Equally, if there is little prior knowledge parameters can be sampled from a wide flat range. Furthermore, there is no restriction on the shape of the distribution beyond the requirement that it can be sampled. This also enables a method of real world implementation that doesn't require prior knowledge of the true parameter values (summarised in Figure 6.11):

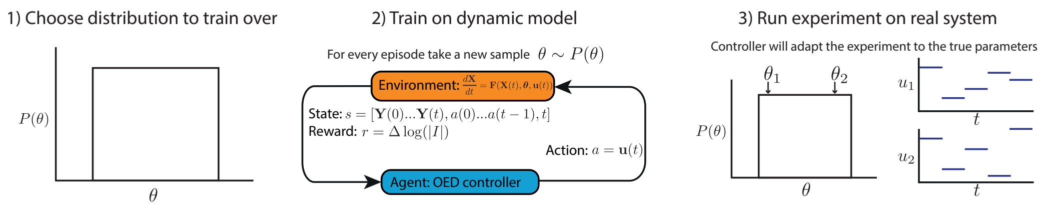

1) Choose distribution to train over
$P(\theta)$
θ
2) Train on dynamic model
For every episode take a new sample $\theta \sim P(\theta)$
Environment: $\frac{d\mathbf{X}}{dt} = \mathbf{F}(\mathbf{X}(t), \boldsymbol{\theta}, \mathbf{u}(t))$
State: $s = [\mathbf{Y}(0)...\mathbf{Y}(t), a(0)...a(t-1), t]$
Reward: $r = \Delta \log(|I|)$
Action: $a = \mathbf{u}(t)$
Agent: OED controller
3) Run experiment on real system
Controller will adapt the experiment to the true parameters
θ_1
θ_2
$P(\theta)$
θ
u_1
t
u_2
t

Figure 6.11: Training over a parameter distribution. 1) The distribution of parameters was chosen as a wide flat distribution, 2) the RL controller is trained where each episode a new chemostat system was sampled from the distribution, 3) the RL controller can design near optimal experiments across the parameter distribution.

1. A distribution is chosen to train over, which can be used to incorporate any prior knowledge. Here I use a wide flat distribution.
2. The RL controller is trained on the dynamical model of the system, for each episode it takes a new parameter sample from the distribution
3. This trained agent can then be used to experiment on the real system with the ability to tailor the experiment depending on the system behaviour

A RL controller trained in this manner will learn to maximise the expected return over the specified distribution. As evidenced by Section 6.4.1 the agent should be able to infer its position in parameter space using the trajectory of experimental inputs and measurements. This means that as the agent observes how the system behaves it should be able to adapt the experiment to maximise the optimality score

To investigate this further, the RL controller was trained in this manner on the single auxotroph system, where each episode uses a different parameter set sampled from a uniform distribution over the three parameters to be optimised (see Table 6.3 for the allowable parameter ranges). The training performance in Figure 6.12 shows that RL controllers can successfully learn to optimise the objective

	Rational	OSAO	MPC	RL
Total normalised error	30.29	39.9	23.5	7.8

Table 6.5: Total parameter error of different methods for optimal experimental design on the single auxotroph model over a parameter distribution

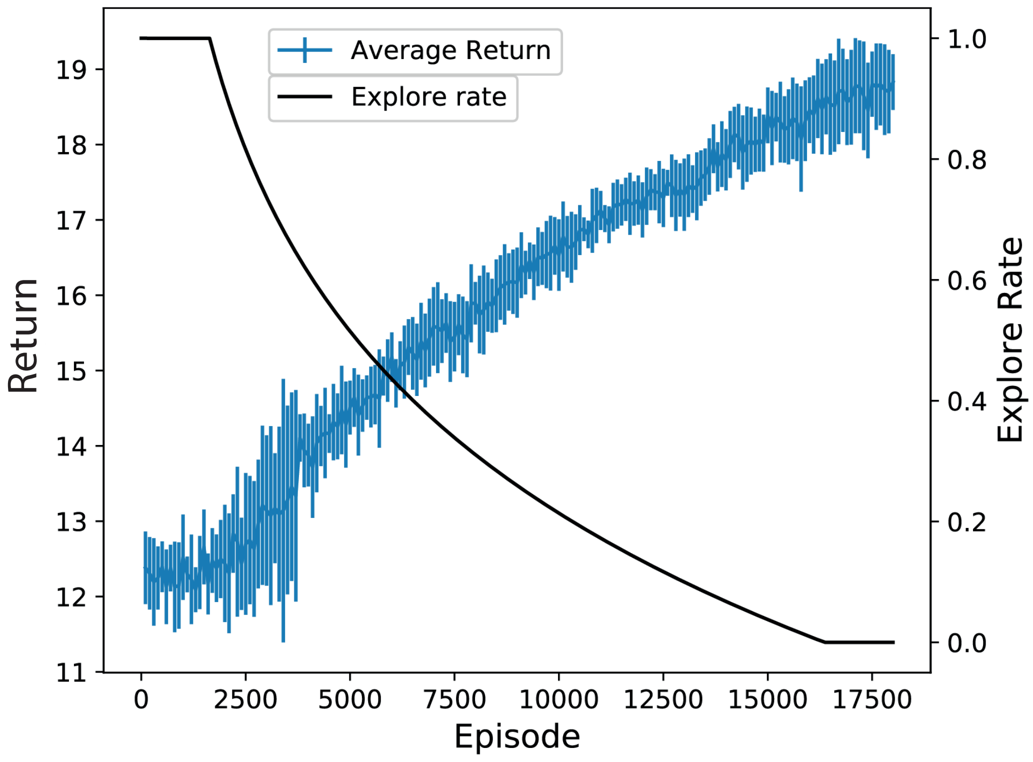


Figure 6.12: Training performance of reinforcement learning for optimal experimental design on the chemostat system over a parameter distribution. Average training progress of ten Recurrent T3D agents over 17500 episodes where the system for each episode is sampled from a distribution over parameters. For interpretability the returns for each of the 17500 episodes were split into bins of 100 and the average of each bin was taken. This was then averaged across the 10 repeats.

over the distribution of parameters. As a comparison to the other methods, 30 noisy experimental trajectories were generated and the rational design, OSAO, MPC and RL controllers were compared by finding the error in repeated fittings of the parameters, where the best performing repeat from Figure 6.12 was used as the RL controller. Here the true system parameters were sampled from the uniform distribution before each repeat and the OSAO and MPC used parameters in the centre of the distribution to optimise with respect to the experimental inputs. Because the true system parameters are different for each experiment, we discard the determinant of the covariance matrix of parameters and the optimality score, as these will both be highly dependent on the samples taken, and focus on the normalised sum square parameter error in the resulting inferred parameters, these results are shown in Table 6.5. These results show that in comparison to the previous section the performance of the MPC has drastically reduced. This is expected because it has no mechanism to change its experiment depending on where in the parameter distribution the true system happens to be. In contrast, the RL controller has maintained its performance well, with a much smaller increase in the total parameter error. This is further verification that the RL controller can adapt the experiment online and that this leads to more informative experiments.

I then further investigated the quality of the resulting experiments designed by the best performing RL controller by comparing them to those designed by an MPC with complete a priori knowledge of the system. This was done by comparing the performance, in terms of the optimality score, of the RL controller against an MPC at different parameters sampled from the distribution. For each parameter sample an experiment was designed using both the RL controller and the MPC. In this scenario the MPC was given the sampled parameters to predict the optimum experimental inputs, whereas the RL controller only had the information encoded in the parameter distribution it was trained on. In total eight samples were investigated. Four of these were chosen rationally to understand the behaviour of the RL controller across the parameter distribution, these were the true parameter values from the literature, the lower bounds, upper bounds and centre of the distribution. The remaining four were sampled randomly from the distribution. The results are summarised in Table 6.6. This shows the optimality score of experiments produced by the MPC and RL controllers. We can see that for every sample the RL controller trained on the parameter distribution has performed nearly as well as an MPC that has total system knowledge. This implies that by using a parameter distribution we can train an RL controller to perform near optimal experiments across the whole distribution. The samples at the upper and lower bounds of the distribution show more deviation between the two controllers. This is expected as the random sampling during training will be less likely to sample points in the vicinity of these positions and the RL controller will likely have less access to relevant training data for these points. For this reason

Parameter value $[\mu_{max}, K_m, K_{m0}]$	MPC	RL
$[0.5, 0.0001, 0.00001]$ (Lower bound)	18.01	16.78
$[1.25, 0.00055, 0.000055]$ (Centre)	20.79	20.31
$[1, 0.00048776, 0.00006845928]$ (Actual params)	20.07	20.11
$[2, 0.001, 0.0001]$ (Upper bound)	21.86	20.15
$[0.552564, 0.000400962, 0.0000775143]$	18.85	17.63
$[0.708972, 0.000500437, 0.0000490196]$	20.05	19.03
$[0.500478, 0.00041873, 0.000029529]$	18.02	16.81
$[1.45073, 0.000810734, 0, 0000961402]$	21.52	20.41

Table 6.6: Comparison of RL OED controller trained over a parameter distribution compared with an MPC with perfect system knowledge. The optimality score of the experiments produced by each controller is shown for different samples within the distribution

the extremities of the prior likely represent a pathological worst case for the RL controller. To understand the experiments the RL agent is performing at different parameter samples the experimental designs and resulting system trajectories of the four rationally chosen parameter samples were plotted (Figure 6.13). The four experiments (Figure 6.13 A, C, E, and G) and the corresponding timeseries (Figure 6.13 B, D, F, and H) are similar, but they do show differences. The experimental inputs for the initial 3 sample and hold intervals are the same for all parameter samples. It is expected that the initial portion of the experiment will be the same for any sample, this is because the agent has little or no information with which to infer the system behaviour at the beginning of the experiment. After this initial stage however, the experiments diverge and there are subtle changes in the input profiles for the different parameter samples. The differences in the experiments are not large however, and this implies that there is a 'core' experimental design which works well over the distribution and the agent deviates from this slightly to maximise its effectiveness for different regions of the parameter distribution.

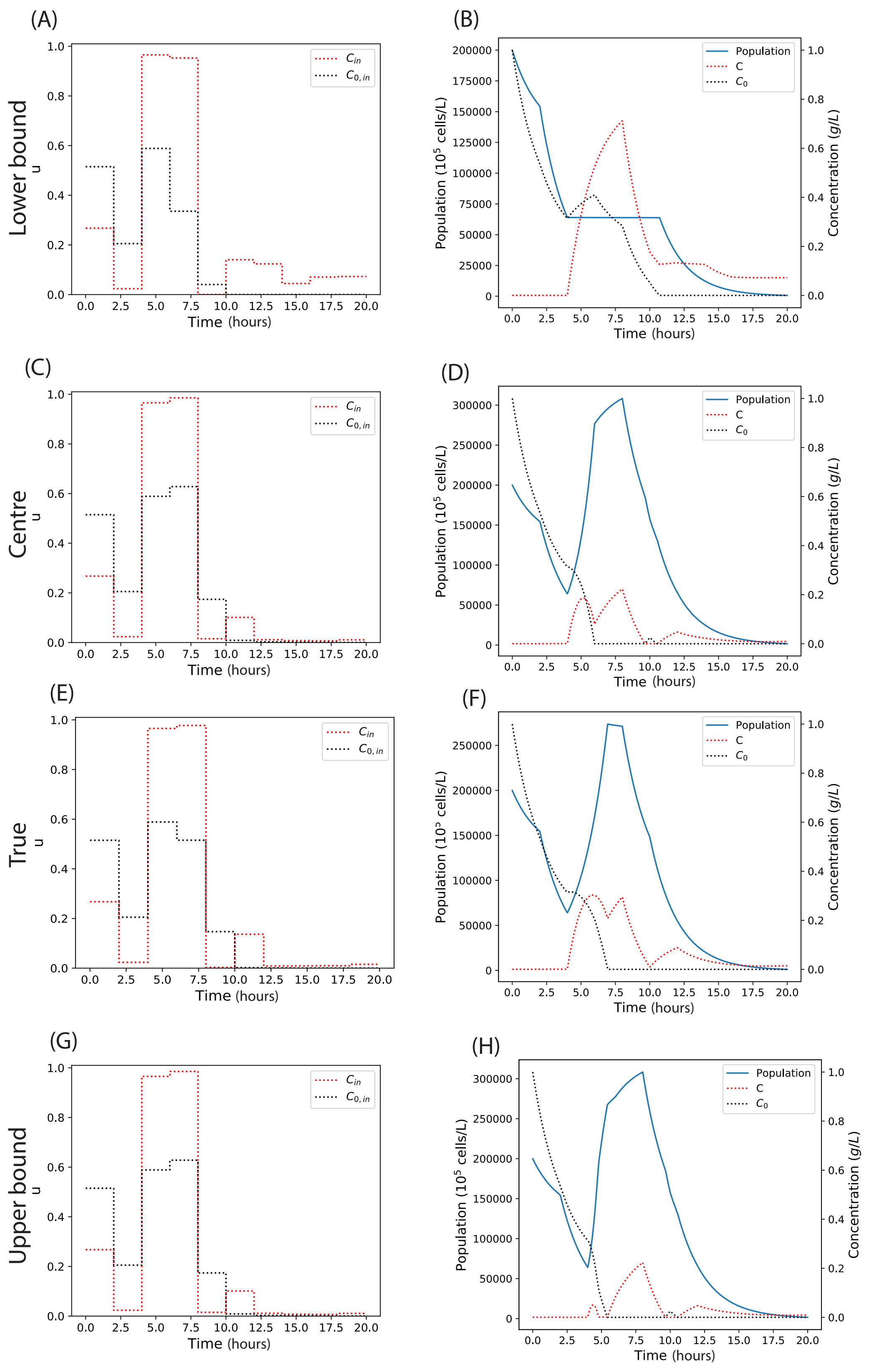


Figure 6.13: The experimental designs of the RL controller at different parameter samples. Here the RL controller is trained on a distribution over parameters and adapts its experimental design for different points in parameter space. The experimental inputs (left) and resulting system trajectories (right) are shown. For each experimental design the system is initialised with different parameter values: the lower bound of the parameter distribution (A,B), the centre (C,D), the true parameter values from literature (E,F) and the upper bound of the distribution (G,H)

6.5 Discussion

In this chapter I have investigated and demonstrated a novel application of reinforcement learning; the optimal design of experiments within biology. The optimal experimental design problem was formulated as the maximisation of the determinant of the Fisher information (D-optimal design). First, the efficacy of the approach was tested using the Fitted Q-learning algorithm (Section 2.2.1). Fitted Q-learning was applied to a simple non-linear Monod growth model and the resulting experimental design was investigated and verified mathematically and by comparison with a OSAO. Next, Fitted Q-learning was applied to a model of gene transcription as an example of a system of realistic biological complexity. The RL controller was compared with a OSAO and the resulting experimental designs were further confirmed by the covariance of repeated parameter estimates fitted to independent noisy experimental trajectories. These initial results with Fitted Q-learning showed that reinforcement learning has the potential to design optimal experiments and my subsequent work focussed on removing the unrealistic dependence on the true parameter values and the restriction of the Fitted Q-agent to discrete action spaces. This dependence on the true parameter values is a limitation of other OED works [169, 170, 173, 174, 175], which require ad hoc verification [169] or other work arounds [173, 174, 175]. To decouple the RL controller from the true parameters I used a recurrent neural network to enable it to make experimental decisions based on a full experimental trajectory of measurements and experimental inputs. It was shown that this allowed an agent to effectively predict the optimality of experiments on systems sampled from a distribution over parameters, performing similarly to an agent with explicit knowledge of the parameters. To move from a discrete to a continuous action space an implementation of the continuous reinforcement learning algorithm T3D was used. This simplifies the learning problem when a large number of different experimental inputs are required and allows the precise control of experimental inputs in a continuous space. The result of these two modifications is the Recurrent T3D algorithm, which I demonstrate by designing optimal experiments to infer the growth parameters of an auxotroph growing in a chemostat. To mirror the RL controller's knowledge of the full experimental trajectory an MPC was developed that optimised over the full experimental trajectory. The RL controller was found to perform well in comparison to the OSAO and MPC despite not having explicit knowledge of the true parameters. Lastly, the ability of the RL controller to infer where it is in parameter space was further shown by training the agent on a distribution over parameters. The resulting trained agent was compared with the MPC and OSAO which used the mean parameter values to make predictions. In this case we see drastically better performance in terms of the error in the inferred parameters for the RL controller than the other methods. Furthermore, it was shown that an RL controller trained on a parameter distribution can design near optimal experiments throughout the distribution by comparison with an MPC with complete system knowledge. This demonstrates how the recurrent neural network allows the same agent to perform well even for different parameter samples and enables a natural method for implementation on a real system.

Overall this work shows the potential to do optimal experimental design with deep reinforcement learning and that this could compliment existing methods. OED work has often been categorised into Bayesian and Fisher information based methods [178, 179]. Fisher information based techniques have been limited to local analysis around a nominal parameter guess. Bayesian approaches allow global optimisation and the incorporation of prior knowledge, but can be very computationally expensive. Here I build on previous work that has demonstrated the applicability of Fisher information based methods to non-linear biological systems [169, 170], including online OED of fed batch bioreactors [174, 175]. I have shown the capability to encode prior parameter knowledge and therefore globally optimise with respect to the Fisher information by training an RL controller over samples from a parameter distribution. The same technique could be used in the future to encompass measurement noise and stochasticity in the model dynamics. This mirrors some of the statistical capabilities of Bayesian approaches. However, Bayesian approaches can require complex integrals over multiple probability distributions, which for online experimental design needs to be solved during the experiment using methods such as Monte Carlo simulation [167, 180]. This can be computationally expensive, potentially limiting the speed at which experiments can be done. Previous Bayesian approaches have been limited to greedily optimising over the next time step only [167] or choosing from a limited set of experimental designs [180], likely to reduce the computational effort required. In this approach, a trained RL controller can be rapidly queried for inputs on the order of a second. However, the price for this is a potentially lengthy training process which must be done prior to the experiment. The longest training times in this chapter were around 12 hours of simulation time on a computing cluster and around 15 hours on a personal laptop.

In this chapter I have focussed on D-optimal design by maximising the determinant of the FIM. However, there is no reason other metrics could not be used. These could include other Fisher information based metrics including maximising the trace of the FIM (A-optimality) or maximising the minimum eigenvalue of the FIM (E-optimality). Furthermore, we are not limited to Fisher information based metrics and future work could explore the possibility of metrics such as maximising the uncertainty of the posterior predictive distribution [180].

I demonstrated the capability to learn over a parameter distribution with a uniform distribution. Future work could be done to evaluate the performance of an RL controller when trained over more complex distributions that are non-symmetric or multi-modal. As the RL controller learns by sampling the distribution this is trivial to implement and the flexibility to learn over any distribution presents a compelling reason to use reinforcement learning for OED. Additionally, work in this area could involve the extension to model selection. This could be done using D_s optimal design, in which experimental effort is focussed on a subset of the parameters of a nested model [181] or by introducing an extra term in the reward function to encourage the divergence of different model predictions [182]. The flexibility of the reinforcement learning framework means that it is possible to incorporate prior knowledge by sampling from distributions over parameters and models and to learn on stochastic models by sampling system trajectories. Further investigation could be done into the effectiveness of this approach to do simultaneous model selection and parameter inference on highly stochastic systems as an alternative to Bayesian methods. The effect of training over a distribution of structurally distinct models on the resulting experiments would be an interesting result. The flexibility of reinforcement learning means that training effort can easily be focussed to regions in model space by increasing their weighting in the distribution training episodes are sampled from. Combining this with techniques such as D_s optimal design could lead to a flexible method to explore large model spaces. Often in synthetic biology there is uncertainty in the structure of the model we are investigating and this can lead to model mismatch. Using a distribution over model space and D_s optimal design it should be possible to investigate a large model space and find the model that best fits the available data. If required, this could be done iteratively over multiple experiments. The first experiment would have a wide focus, where training is done over the whole model space. Subsequent training runs and experiments could then narrow the focus regions of model space found to be relevant.

Overall in this chapter I have developed and demonstrated the potential for reinforcement learning to be used for OED. As the systems we build and characterise in biology continue to increase in complexity, automated experimental design tools will become ever more important. Furthermore this method is general and could be applied in many areas of science to understand natural systems and shows potential to be complimentary to existing tools.

Chapter 7

General conclusions

In the last two decades the field of synthetic biology has made rapid progress by applying the principles of engineering to the construction of novel biological systems. As the complexity of our engineered systems increases we will find increasing need to escape the limitations of using a single cell type and create engineered distributed systems. Synthetic biology is a highly interdisciplinary field and a key characteristic of the field is the use of mathematical and computational design tools. In this thesis ideas from mathematics, physics and artificial intelligence have been applied in the development of novel design, control and experimentation algorithms which will facilitate the construction of distributed biological systems.

In Chapters 3 and 4 I develop novel methods for the design of distributed biological computers composed of patterns of bacterial colonies communicating with diffusible molecules. I develop a new optimisation algorithm that facilitates a key capability from electronic circuit design, the optimal construction of digital functions, within synthetic biology. I also create an algorithm for the training of bacterial neural networks, this leverages the analogue computing capabilities of artificial neural networks that have been the driving force behind the widespread success of deep learning. I anticipate that as synthetic biology advances as a field the limitations of bio-computing within a single cell will become increasingly restrictive and distributed methods of computing like the ones investigated in this thesis will become ever more important. These methods could be applied in the future to the automated construction of biocomputers with complex information processing capabilities. In combination with biosensors, which can be engineered to detect a myriad of biological molecules, these could be used to analyse biological samples cheaply and provide an easy to read fluorescence output. An important example of this is medical diagnosis of the gut microbiome disease states. This design methodology could, in the future, be combined with methods for the printing of cells on paper devices which can be stored for later use [55] to construct cheap diagnostic devices for use in hospitals, and eventually as home testing kits. Furthermore, the methods developed represent general theoretical frameworks for digital and analogue spatial computers. These frameworks could be applied in the future for understanding how patterning is used for natural biocomputing, for example the role of spatial structure in the information processing capabilities of biofilms.

The second theme investigated was the control of distributed biological systems. In Chapter 5 I use a technique from artificial intelligence, reinforcement learning, to control microbial communities. Microbial communities engineered for bioprocessing have been shown to be more productive than single strains [82, 83, 86] and are not limited in the complexity of the metabolic pathway by the metabolic capability of a single cell [87]. These factors make the effective utilisation of communities an important consideration for the future of industrial bioprocessing and I show that reinforcement learning is a compelling compliment to existing control techniques. As the complexity of the microbial communities used for industrial processes increases I anticipate that more complex control methodologies will become ever more important.

The final avenue of work, Chapter 6, considered the use of reinforcement learning for optimal experimental design in synthetic biology. Distributed biological systems have the potential to be governed by complex and highly non-linear dynamics. This further increases the need for accurate models within synthetic biology, while also making the rational design of experiments challenging. I show that the flexibility of reinforcement learning allows a natural way to optimise over statistical distributions resulting in a method that could be implemented in the laboratory. There are several potential advantages of a general reinforcement learning algorithm for optimal experimental design over existing mathematical methods. These include ease of use for non-specialists, greater flexibility of implementation, removal of restrictions on the probability distributions that can be optimised over, and parallel implementation to reduce the computational time required. I believe these advantages make it a compelling option for the future of experimental design in synthetic biology and other scientific fields.

In summary, the methods developed here will have impact in distributed biological computing, distributed bioprocessing and optimal experimental design in synthetic biology. Like the field as a whole this thesis is an interdisciplinary effort to apply techniques from other disciplines to biological systems and I believe that continuing interdisciplinary work will be key to the future success of synthetic biology. Biological systems are often complicated and highly non-linear and many engineering challenges such as noise, unexpected downstream effects and interference between communication channels are compounded. Furthermore, cells are reproducing organisms subject to mutation and the forces of selection, which means that any engineered alteration might be discarded if it hinders reproductive success. These factors can make the reductionist, bottom up approaches from traditional engineering challenging and much of the future success of synthetic biology will be dictated by how we deal with these challenges. The obvious approach is to mitigate or build resistance to these effects into designs and I think that the near future work into the implementations of synthetic biocomputers should continue in this vein. Additionally, the continued advancement of high throughput experimental techniques means that the quantity of data we have access to is always increasing. This quantity of data facilitates the, often data hungry, methods from machine learning and data science and means a key future challenge in biology is the application of these methods to best exploit this data. Within synthetic biology, I think that it should continue to be used to create new machine learning based methods to design emergent systems which exploit the effects of noise, interference and selection in their designs. This could mean that a large part of the future of synthetic biology looks much different from the traditional engineering disciplines.

Appendix A

Reinforcement learning control repeats

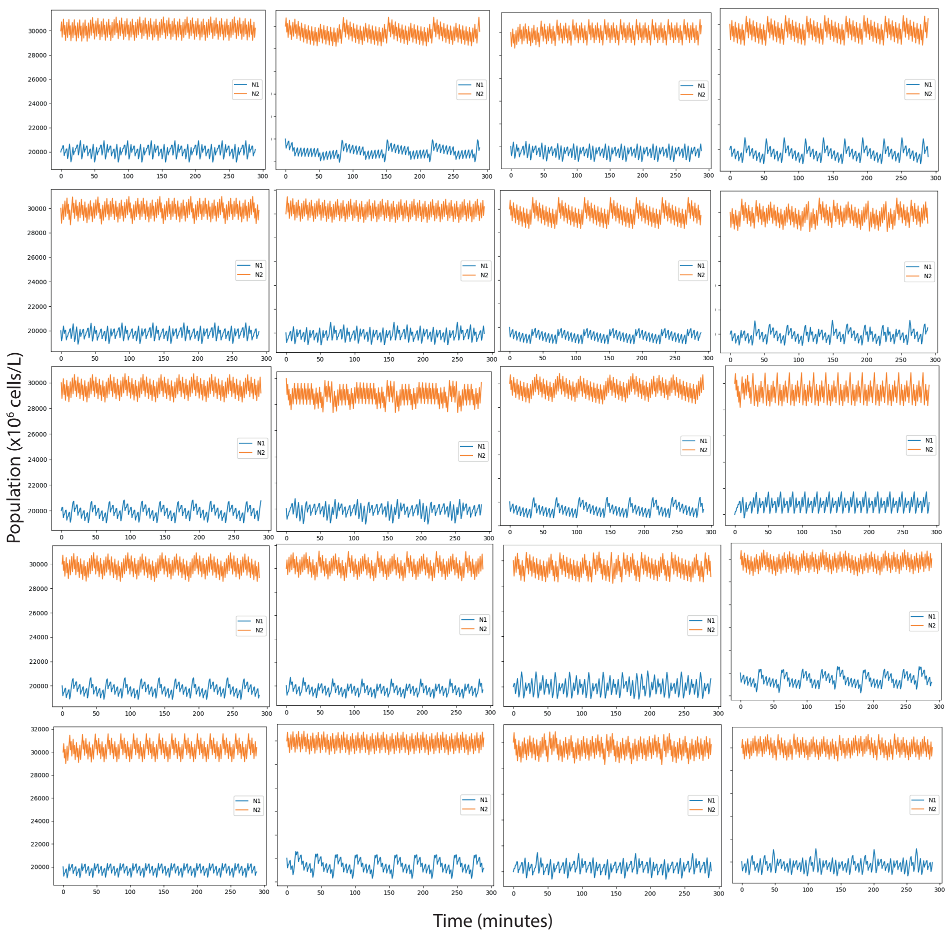


Figure A.1: Episodic Fitted Q-iteration repeats. Population curves of twenty trained agents controlling the chemostat system.

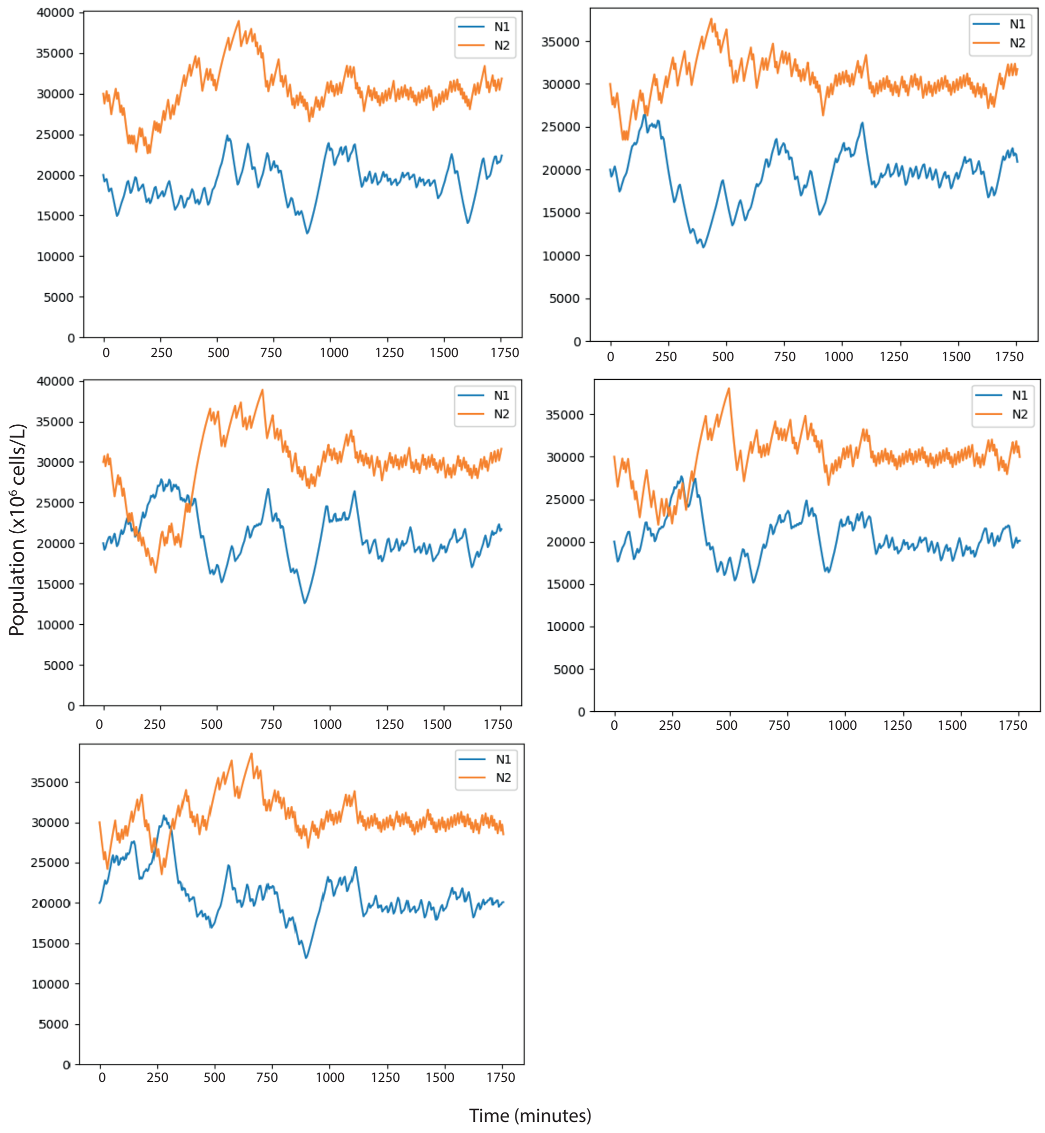


Figure A.2: All population curves from Online Fitted Q-iteration running in parallel. Populations curves of five chemostats running in parallel while under online control of a single agent. Here the agent is trained for 1440 minutes (twenty-four hours) and then allowed to control the system for a further 310 minutes to show that the target system behaviour is maintained.

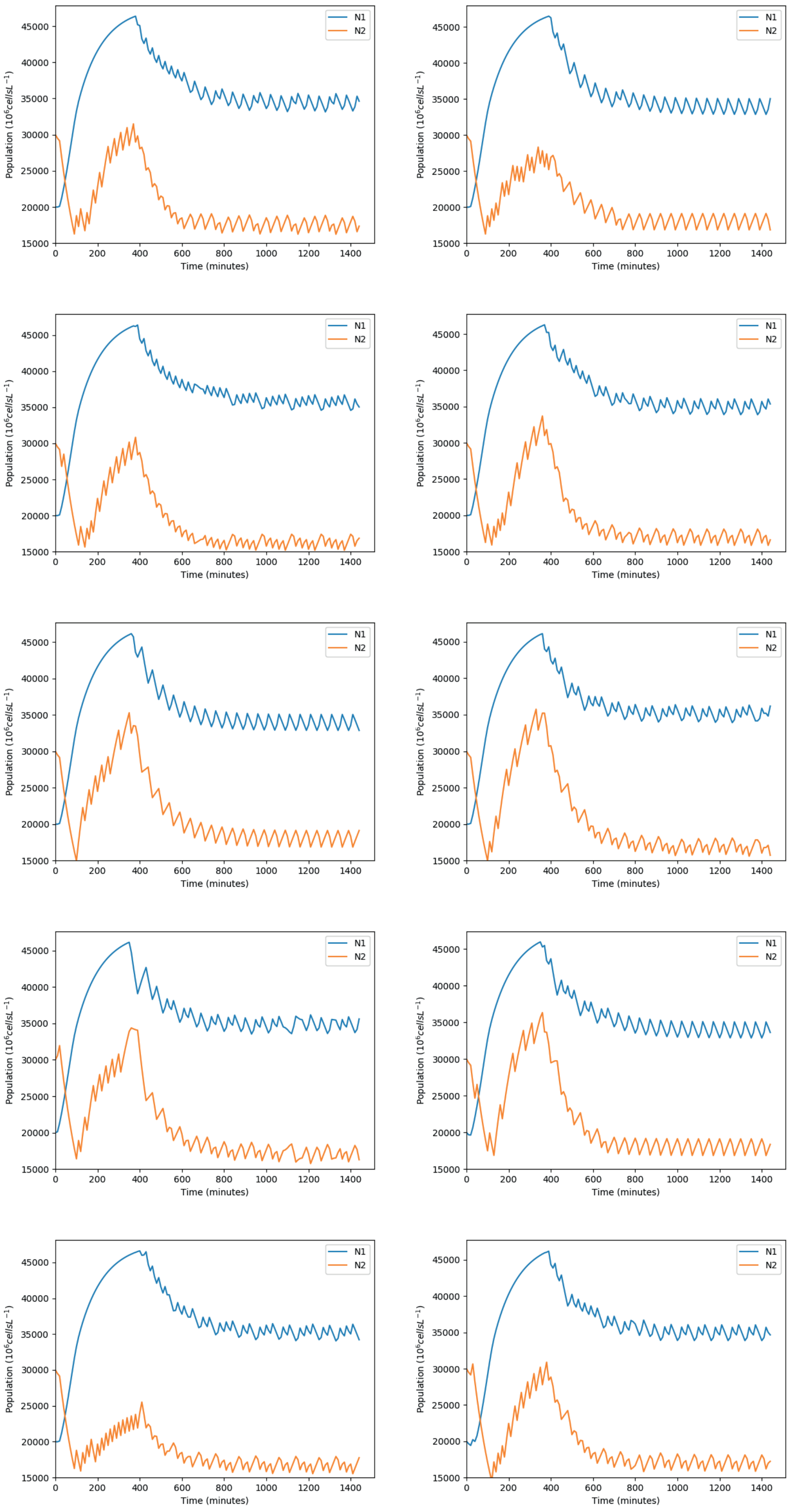


Figure A.3: Optimising product output repeats. Population curves of ten trained agents controlling the chemostat system with the goal of optimising product output.

Appendix B

Mathematics of optimal experimental design on the Monod system

Firstly, we will show that to get information about the parameters u must vary with time.

We start from

$$\frac{dx}{dt} = \frac{p_1 u}{p_2 + u} x$$

First calculate the sensitivity time derivatives:

$$\frac{d\dot{x}}{dp_1} = \frac{u}{p_2 + u} x = \dot{x}_{p_1}$$

$$\frac{d\dot{x}}{dp_2} = -\frac{p_1 u}{(p_2 + u)^2} x = \dot{x}_{p_2}$$

We know that

$$I = \begin{bmatrix} \int_0^{t_i} \dot{x}_{p_1}^2 dt & \int_0^{t_i} \dot{x}_{p_1} \dot{x}_{p_2} dt \\ \int_0^{t_i} \dot{x}_{p_1} \dot{x}_{p_2} dt & \int_0^{t_i} \dot{x}_{p_2}^2 dt \end{bmatrix} = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

Where the full fisher information matrix. The determinant of F is given by:

$$|I| = ac - b^2 = \int_0^{t_i} \frac{u^2}{(p_2 + u)^2} x^2 dt \int_0^{t_i} \frac{p_1^2 u^2}{(p_2 + u)^4} x^2 dt - \left(\int_0^{t_i} \frac{p_1 u^2}{(p_2 + u)^3} x^2 dt \right)^2$$

If we assume u is constant, we can take it out of the integrals:

$$|I| = \frac{u^2}{(p_2+u)^2}\int_0^{t_i} x^2 dt \frac{p_1^2u^2}{(p_2+u)^4}\int_0^{t_i} x^2 dt - \left(\frac{p_1u^2}{(p_2+u)^3}\int_0^{t_i} x^2 dt\right)^2$$

$$|I| = \frac{p_1^2u^4}{(p_2+u)^6}\left(\int_0^{t_i} x^2\right)^2 dt - \frac{p_1^2u^4}{(p_2+u)^6}\left(\int_0^{t_i} x^2\right)^2 dt = 0$$

Showing that if u doesn't vary then the determinant of the fisher information matrix is 0.

Now we will consider a simple experiment with two timesteps of time 1, each with different input u; u_1 and u_2, where u is a step function. For this special case $\int_0^2 f(u)dt = f(u_1) + f(u_2)$. Now look at

$$|I| = \int_0^{t_i} \frac{u^2}{(p_2+u)^2} x^2 dt \int_0^{t_i} \frac{p_1^2u^2}{(p_2+u)^4} x^2 dt - \left(\int_0^{t_i} \frac{p_1u^2}{(p_2+u)^3} x^2 dt\right)^2 \tag{B.1}$$

we will, for now, ignore the x^2 that all the integral terms depend on to simplify the analysis. The effect this will have is to alter the scaling between u_1 and u_2 as x increases monotonically in our simulations.

$$|I| \approx \int_0^{t_i} \frac{u^2}{(p_2+u)^2} dt \int_0^{t_i} \frac{p_1^2u^2}{(p_2+u)^4} dt - \left(\int_0^{t_i} \frac{p_1u^2}{(p_2+u)^3} dt\right)^2$$

For our simple experiment with two u values:

$$\begin{aligned} |I| \approx& \\ &\left(\frac{u_1^2}{(p_2+u_1)^2} + \frac{u_2^2}{(p_2+u_2)^2}\right)\left(\frac{p_1^2u_1^2}{(p_2+u_1)^4} + \frac{p_1^2u_2^2}{(p_2+u_2)^4}\right) \\ &-\left(\frac{p_1u_1^2}{(p_2+u_1)^3} + \frac{p_1u_2^2}{(p_2+u_2)^3}\right)^2. \end{aligned} \tag{B.2}$$

Note that if $u_1 = u_2$ then $|I| = 0$ as expected. If we expand the brackets and simplify we are left with:

$$
\begin{aligned}
&|I|(u_1,u_2) \approx \\
&\quad p_1^2u_1^2u_2^2\left(\frac{1}{(p_2+u_1)^4(1+u_2)^2}+\right. \\
&\qquad \frac{1}{(p_2+u_1)^2(p_2+u_2)^4}- \\
&\qquad \left.\frac{2}{(p_2+u_1)^3(p_2+u_2)^3}\right).
\end{aligned} \tag{B.3}
$$

Appendix C

Parameter identifiability of the single chemostat system

First we will look at the structural identifiability of the parameters of a single auxotroph grown in a chemostat. In this model a single bacterial strain grows in a chemostat, its growth is reliant a carbon source and an auxotrophic nutrient, both of which can be controlled during an experiment by controlling the flow of nutrients into the chemostat (C_{in} and $C_{0,in}$). The population of the bacteria (N) is the only measured system variable and $\mathbf{C}$ and C_0 are hidden variables. The system equations are:

$$\frac{d}{dt}C_0 = q(C_{0,in} - C_0) - \frac{1}{\gamma_0}\mu\mathbf{N} \tag{C.1}$$

$$\frac{d}{dt}\mathbf{C} = q(\mathbf{C}_{in} - \mathbf{C}) - \frac{1}{\gamma}\mu\mathbf{N} \tag{C.2}$$

$$\mu = \mu_{max}\frac{\mathbf{C}}{\mathbf{K}_s + \mathbf{C}}\frac{C_0}{\mathbf{K}_{s0} + C_0} \tag{C.3}$$

where μ_{max} is a vector of the maximum growth rate for each species, $\mathbf{K}_s$ is a vector of half-maximal auxotrophic nutrient concentrations and $\mathbf{K}_{s0}$ is a vector of half-maximal concentrations C_0 for the shared carbon source.

The growth rate of the bacterial population is:

$$\frac{d}{dt}\mathbf{N} = (\mu - q)\mathbf{N} \tag{C.4}$$

First substitute Equation C.3 into Equations C.1, C.2 and C.4:

$$\frac{d}{dt}C_0 = q(C_{0,in} - C_0) - \frac{\mu_{max}}{\gamma_0}\frac{\mathbf{C}}{\mathbf{K}_s + \mathbf{C}}\frac{C_0}{\mathbf{K}_{s0} + C_0}\mathbf{N}$$

$$\frac{d}{dt}\mathbf{C} = q(\mathbf{C}_{in} - \mathbf{C}) - \frac{\mu_{max}}{\gamma}\frac{\mathbf{C}}{\mathbf{K}_s + \mathbf{C}}\frac{C_0}{\mathbf{K}_{s0} + C_0}\mathbf{N}$$

$$\frac{d}{dt}\mathbf{N} = (\mu_{max}\frac{\mathbf{C}}{\mathbf{K}_s + \mathbf{C}}\frac{C_0}{\mathbf{K}_{s0} + C_0} - q)\mathbf{N}.$$

We will non-dimensionalise the hidden variables by substituting $C = a\tilde{C}$ $C_0 = b\tilde{C_0}$ where $\tilde{C}$ and $\tilde{C_0}$ are the dimensionless variables.

This gives us, after some rearranging:

$$\frac{d}{dt}\tilde{C_0} = q(\frac{C_{0,in}}{b} - \tilde{C_0}) - \frac{\mu_{max}}{b\gamma_0}\frac{\tilde{C}}{\frac{\mathbf{K}_s}{a} + \tilde{C}}\frac{\tilde{C_0}}{\frac{\mathbf{K}_{s0}}{b} + \tilde{C_0}}\mathbf{N}$$

$$\frac{d}{dt}\tilde{C} = q(\frac{\mathbf{C}_{in}}{a} - \tilde{C}) - \frac{\mu_{max}}{a\gamma}\frac{\tilde{C}}{\frac{\mathbf{K}_s}{a} + \tilde{C}}\frac{\tilde{C_0}}{\frac{\mathbf{K}_{s0}}{b} + \tilde{C_0}}\mathbf{N}$$

$$\frac{d}{dt}\mathbf{N} = (\mu_{max}\frac{\tilde{C}}{\frac{\mathbf{K}_s}{a} + \tilde{C}}\frac{\tilde{C_0}}{\frac{\mathbf{K}_{s0}}{b} + \tilde{C_0}} - q)\mathbf{N}.$$

Now set

$$\frac{\mu_{max}}{b\gamma_0} = \frac{\mu_{max}}{a\gamma} = 1$$

meaning $b = \frac{\mu_{max}}{\gamma_0}$ and $a = \frac{\mu_{max}}{\gamma}$

so that

$$\frac{d}{dt}\tilde{C_0} = q(\frac{C_{0,in}\gamma_0}{\mu_{max}} - \tilde{C_0}) - \frac{\tilde{C}}{\frac{\mathbf{K}_s\gamma}{\mu_{max}} + \tilde{C}}\frac{\tilde{C_0}}{\frac{\mathbf{K}_{s0}\gamma_0}{\mu_{max}} + \tilde{C_0}}\mathbf{N}$$

$$\frac{d}{dt}\tilde{C} = q(\frac{C_{in}\gamma}{\mu_{max}} - \tilde{C}) - \frac{\tilde{C}}{\frac{\mathbf{K}_s\gamma}{\mu_{max}} + \tilde{C}} \frac{\tilde{C_0}}{\frac{\mathbf{K}_{s0}\gamma_0}{\mu_{max}} + \tilde{C_0}} \mathbf{N}$$

$$\frac{d}{dt}\mathbf{N} = (\mu_{max} \frac{\tilde{C}}{\frac{\mathbf{K}_s\gamma}{\mu_{max}} + \tilde{C}} \frac{\tilde{C_0}}{\frac{\mathbf{K}_{s0}\gamma_0}{\mu_{max}} + \tilde{C_0}} \mathbf{N} - q)\mathbf{N}.$$

Here some of the parameters are lumped, however we have 5 unique lumped parameters and 5 real parameters. I wasn't able to simplify any further than this, leading me to believe that the model is structurally identifiable. To verify this I used the MATLAB [183] package STRIKE-GOLD [184]. STRIKE-GOLD can determine the structural identifiability of the parameters of dynamic systems models and confirmed the structural identifiability of all parameters.

To investigate the practical identifiability I looked a the values for the Fisher information matrix over 25000 experiments. For each experiment a different sample of the uniform prior between $[100000, 100000, 0.5, 0.0001, 0.00001] \leq [\gamma, \gamma_0, \mu_{max}, \mathbf{K}_s, \mathbf{K}_{s0}] \leq [1000000, 1000000, 2, 0.001, 0.0001]$ was taken and 10 random experimental inputs were applied to the resulting system. Included in these experimental inputs was the dilution rate, q, to ensure that the identifiability problems were not due to our choice of q. For every experiment the Fisher information was always 0 for both γ_0 and γ, leading me to believe they are practically unidentifiable.

Bibliography

[1] Maurice Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599, 1953.

[2] Robert Brayton, G.D. Hatchel, R. Hemanchandra, and A. Sangiovanni-Vincentelli. A comparison of logic minimization strategies using espresso: An apl program package for partitioned logic minimization. pages 42–48, 1982.

[3] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[4] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[5] Chen Debao. Degree of approximation by superpositions of a sigmoidal function. *Approximation Theory and its Applications*, 9(3):17–28, 1993.

[6] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[7] Xueheng Qiu, Le Zhang, Ye Ren, Ponnuthurai N Suganthan, and Gehan Amaratunga. Ensemble deep learning for regression and time series forecasting.

In *2014 IEEE symposium on computational intelligence in ensemble learning (CIEL)*, pages 1–6. IEEE, 2014.

[8] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[10] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[11] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[12] Lewis Grozinger, Martyn Amos, Thomas E Gorochowski, Pablo Carbonell, Diego A Oyarzún, Ruud Stoof, Harold Fellermann, Paolo Zuliani, Huseyin Tas, and Angel Goñi-Moreno. Pathways to cellular supremacy in biocomputing. *Nature communications*, 10(1):1–11, 2019.

[13] Leonard M Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[14] Milan N Stojanovic, Tiffany Elizabeth Mitchell, and Darko Stefanovic. Deoxyribozyme-based logic gates. *Journal of the American Chemical Society*, 124(14):3555–3561, 2002.

[15] Milan N Stojanović and Darko Stefanović. Deoxyribozyme-based half-adder. *Journal of the American Chemical Society*, 125(22):6673–6676, 2003.

[16] Harvey Lederman, Joanne Macdonald, Darko Stefanovic, and Milan N Stojanovic. Deoxyribozyme-based three-input logic gates and construction of a molecular full adder. *Biochemistry*, 45(4):1194–1199, 2006.

[17] Joanne Macdonald, Yang Li, Marko Sutovic, Harvey Lederman, Kiran Pendri, Wanhong Lu, Benjamin L Andrews, Darko Stefanovic, and Milan N Stojanovic. Medium scale integration of molecular logic gates in an automaton. *Nano letters*, 6(11):2598–2603, 2006.

[18] Renjun Pei, Elizabeth Matamoros, Manhong Liu, Darko Stefanovic, and Milan N Stojanovic. Training a molecular automaton to play a game. *Nature nanotechnology*, 5(11):773, 2010.

[19] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with dna strand displacement cascades. *Nature*, 475(7356):368, 2011.

[20] Andrew Currin, Konstantin Korovin, Maria Ababi, Katherine Roper, Douglas B Kell, Philip J Day, and Ross D King. Computing exponentially faster: implementing a non-deterministic universal turing machine using dna. *Journal of The Royal Society Interface*, 14(128):20160990, 2017.

[21] Zhen Xie, Liliana Wroblewska, Laura Prochazka, Ron Weiss, and Yaakov Benenson. Multi-input rnai-based logic circuit for identification of specific cancer cells. *Science*, 333(6047):1307–1311, 2011.

[22] Satoshi Matsuura, Hiroki Ono, Shunsuke Kawasaki, Yi Kuang, Yoshihiko Fujita, and Hirohide Saito. Synthetic rna-based logic computation in mammalian cells. *Nature communications*, 9(1):1–8, 2018.

[23] Alexander A Green, Jongmin Kim, Duo Ma, Pamela A Silver, James J Collins, and Peng Yin. Complex cellular logic computation using ribocomputing devices. *Nature*, 548(7665):117–121, 2017.

[24] Guillermo Rodrigo, Thomas E Landrain, Eszter Majer, José-Antonio Daròs, and Alfonso Jaramillo. Full design automation of multi-state rna devices to program gene expression using energy-based optimization. *PLoS computational biology*, 9(8):e1003172, 2013.

[25] Pejman Mohammadi, Niko Beerenwinkel, and Yaakov Benenson. Automated design of synthetic cell classifier circuits using a two-step optimization strategy. *Cell systems*, 4(2):207–218, 2017.

[26] Ronan Baron, Oleg Lioubashevski, Eugenii Katz, Tamara Niazov, and Itamar Willner. Logic gates and elementary computing by enzymes. *The Journal of Physical Chemistry A*, 110(27):8548–8553, 2006.

[27] Guinevere Strack, Marcos Pita, Maryna Ornatska, and Evgeny Katz. Boolean logic gates that use enzymes as input signals. *ChemBioChem*, 9(8):1260–1266, 2008.

[28] Jian Zhou, Mary A Arugula, Jan Halamek, Marcos Pita, and Evgeny Katz. Enzyme-based nand and nor logic gates with modular design. *The Journal of Physical Chemistry B*, 113(49):16065–16070, 2009.

[29] Vladimir Privman, Oleksandr Zavalov, Lenka Halámková, Fiona Moseley, Jan Halámek, and Evgeny Katz. Networked enzymatic logic gates with filtering: new theoretical modeling expressions and their experimental application. *The Journal of Physical Chemistry B*, 117(48):14928–14939, 2013.

[30] Jan Halamek, Vera Bocharova, Soujanya Chinnapareddy, Joshua Ray Windmiller, Guinevere Strack, Min-Chieh Chuang, Jian Zhou, Padmanabhan Santhosh, Gabriela V Ramirez, Mary A Arugula, et al. Multi-enzyme logic network architectures for assessing injuries: digital processing of biomarkers. *Molecular BioSystems*, 6(12):2554–2560, 2010.

[31] Dan V Nicolau, Mercy Lard, Till Korten, Falco CMJM van Delft, Malin Persson, Elina Bengtsson, Alf Månsson, Stefan Diez, and Heiner Linke.

Parallel computation with molecular-motor-propelled agents in nanofabricated networks. *Proceedings of the National Academy of Sciences*, page 201510825, 2016.

[32] Timothy S Gardner, Charles R Cantor, and James J Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767):339–342, 2000.

[33] Michael B Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[34] Attila Becskei and Luis Serrano. Engineering stability in gene networks by autoregulation. *Nature*, 405(6786):590–593, 2000.

[35] Neil Dalchau, Gregory Szép, Rosa Hernansaiz-Ballesteros, Chris P Barnes, Luca Cardelli, Andrew Phillips, and Attila Csikász-Nagy. Computing with biological switches and clocks. *Natural computing*, 17(4):761–779, 2018.

[36] J Christopher Anderson, Christopher A Voigt, and Adam P Arkin. Environmental signal integration by a modular and gate. *Molecular systems biology*, 3(1):133, 2007.

[37] Tae Seok Moon, Chunbo Lou, Alvin Tamsir, Brynne C Stanton, and Christopher A Voigt. Genetic programs constructed from layered logic gates in single cells. *Nature*, 491(7423):249–253, 2012.

[38] Brynne C Stanton, Alec AK Nielsen, Alvin Tamsir, Kevin Clancy, Todd Peterson, and Christopher A Voigt. Genomic mining of prokaryotic repressors for orthogonal logic gates. *Nature chemical biology*, 10(2):99–105, 2014.

[39] Shuyi Zhang and Christopher A Voigt. Engineered dcas9 with reduced toxicity in bacteria: implications for genetic circuit design. *Nucleic acids research*, 46(20):11115–11125, 2018.

[40] Alec AK Nielsen, Bryan S Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A Strychalski, David Ross, Douglas Densmore,

and Christopher A Voigt. Genetic circuit design automation. *Science*, 352(6281), 2016.

[41] Joseph H Davis, Adam J Rubin, and Robert T Sauer. Design, construction and characterization of a set of insulated bacterial promoters. *Nucleic acids research*, 39(3):1131–1141, 2011.

[42] Chunbo Lou, Brynne Stanton, Ying-Ja Chen, Brian Munsky, and Christopher A Voigt. Ribozyme-based insulator parts buffer synthetic circuits from genetic context. *Nature biotechnology*, 30(11):1137–1142, 2012.

[43] Vivek K Mutalik, Joao C Guimaraes, Guillaume Cambray, Colin Lam, Marc Juul Christoffersen, Quynh-Anh Mai, Andrew B Tran, Morgan Paull, Jay D Keasling, Adam P Arkin, et al. Precise and reliable gene expression via standard transcription and translation initiation elements. *Nature methods*, 10(4):354–360, 2013.

[44] Mao Taketani, Jianbo Zhang, Shuyi Zhang, Alexander J Triassi, Yu-Ja Huang, Linda G Griffith, and Christopher A Voigt. Genetic circuit design automation for the gut resident species bacteroides thetaiotaomicron. *Nature Biotechnology*, 38(8):962–969, 2020.

[45] R Sarpeshkar. Analog synthetic biology. *Phil. Trans. R. Soc. A*, 372(2012):20130110, 2014.

[46] Lulu Qian and Erik Winfree. Scaling up digital circuit computation with dna strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

[47] Ramiz Daniel, Jacob R Rubens, Rahul Sarpeshkar, and Timothy K Lu. Synthetic analog computation in living cells. *Nature*, 497(7451):619, 2013.

[48] Amir Pandi, Mathilde Koch, Peter L Voyvodic, Paul Soudier, Jerome Bonnet, Manish Kushwaha, and Jean-Loup Faulon. Metabolic perceptrons for neural computing in biological systems. *Nature communications*, 10(1):1–13, 2019.

[49] Alvin Tamsir, Jeffrey J Tabor, and Christopher A Voigt. Robust multicellular computing using genetically encoded nor gates and chemical 'wires'. *Nature*, 469(7329):212–215, 2011.

[50] Sergi Regot, Javier Macia, Núria Conde, Kentaro Furukawa, Jimmy Kjellén, Tom Peeters, Stefan Hohmann, Eulãlia De Nadal, Francesc Posas, and Ricard Solé. Distributed biological computation with multicellular engineered networks. *Nature*, 469(7329):207–211, 2011.

[51] M Ali Al-Radhawi, Anh Phong Tran, Elizabeth A Ernst, Tianchi Chen, Christopher A Voigt, and Eduardo D Sontag. Distributed implementation of boolean functions by transcriptional synthetic circuits. *ACS Synthetic Biology*, 9(8):2172–2187, 2020.

[52] Sarah Guiziou, Federico Ulliana, Violaine Moreau, Michel Leclere, and Jerome Bonnet. An automated design framework for multicellular recombinase logic. *ACS synthetic biology*, 7(5):1406–1412, 2018.

[53] Sarah Guiziou, Pauline Mayonove, and Jerome Bonnet. Hierarchical composition of reliable recombinase logic devices. *Nature communications*, 10(1):1–7, 2019.

[54] Javier Macia, Romilde Manzoni, Núria Conde, Arturo Urrios, Eulàlia de Nadal, Ricard Solé, and Francesc Posas. Implementation of complex biological logic circuits using spatially distributed multicellular consortia. *PLoS computational biology*, 12(2):e1004685, 2016.

[55] Sira Mogas-Díez, Eva Gonzalez-Flo, and Javier Macía. 2d printed multicellular devices performing digital and analogue computation. *Nature communications*, 12(1):1–10, 2021.

[56] T Chen, C Voigt, E Sontag, et al. A synthetic distributed genetic multi-bit counter. 2021.

[57] Hideki Kobayashi, Mads Kaern, Michihiro Araki, Kristy Chung, Timothy S Gardner, Charles R Cantor, and James J Collins. Programmable cells: interfacing natural and engineered gene networks. *Proceedings of the National Academy of Sciences*, 101(22):8414–8419, 2004.

[58] Victoria Hsiao, Yutaka Hori, Paul WK Rothemund, and Richard M Murray. A population-based temporal logic gate for timing and recording chemical events. *Molecular systems biology*, 12(5):869, 2016.

[59] Arturo Urrios, Javier Macia, Romilde Manzoni, Núria Conde, Adriano Bonforti, Eulàlia de Nadal, Francesc Posas, and Ricard Solé. A synthetic multicellular memory device. *ACS synthetic biology*, 5(8):862–873, 2016.

[60] Josep Sardanyés, Adriano Bonforti, Nuria Conde, Ricard Solé, and Javier Macia. Computational implementation of a tunable multicellular memory circuit for engineered eukaryotic consortia. *Frontiers in physiology*, 6:281, 2015.

[61] Javier Macia, Blai Vidiella, and Ricard V Solé. Synthetic associative learning in engineered multicellular consortia. *Journal of The Royal Society Interface*, 14(129):20170158, 2017.

[62] Yolanda Schaerli, Andreea Munteanu, Magüi Gili, James Cotterell, James Sharpe, and Mark Isalan. A unified design space of synthetic stripe-forming networks. *Nature communications*, 5(1):1–10, 2014.

[63] Alissa Kerner, Jihyang Park, Audra Williams, and Xiaoxia Nina Lin. A programmable escherichia coli consortium via tunable symbiosis. *PLoS One*, 7(3), 2012.

[64] Oleg Kanakov, Roman Kotelnikov, Ahmed Alsaedi, Lev Tsimring, Ramon Huerta, Alexey Zaikin, and Mikhail Ivanchenko. Multi-input distributed classifiers for synthetic genetic circuits. *PloS one*, 10(5):e0125144, 2015.

[65] Behzad D Karkaria, Neythen J Treloar, Chris P Barnes, and Alex JH Fedorec. From microbial communities to distributed computing systems. *Frontiers in Bioengineering and Biotechnology*, 8:834, 2020.

[66] Adam G Krieger, Jiahao Zhang, and Xiaoxia N Lin. Temperature regulation as a tool to program synthetic microbial community composition. *Biotechnology and Bioengineering*, 118(3):1381–1392, 2021.

[67] Christina V Dinh, Xingyu Chen, and Kristala LJ Prather. Development of a quorum-sensing based circuit for control of coculture population composition in a naringenin production system. *ACS synthetic biology*, 9(3):590–597, 2020.

[68] Angel Goñi-Moreno and Martyn Amos. A reconfigurable nand/nor genetic logic gate. *BMC systems biology*, 6(1):1–11, 2012.

[69] Michael J Liao, M Omar Din, Lev Tsimring, and Jeff Hasty. Rock-paper-scissors: engineered population dynamics increase genetic stability. *Science*, 365(6457):1045–1049, 2019.

[70] Angel Goñi-Moreno, Fernando de la Cruz, Alfonso Rodríguez-Patón, and Martyn Amos. Dynamical task switching in cellular computers. *Life*, 9(1):14, 2019.

[71] Andriy Didovyk, Oleg I Kanakov, Mikhail V Ivanchenko, Jeff Hasty, Ramón Huerta, and Lev Tsimring. Distributed classifier based on genetically engineered bacterial cell cultures. *ACS synthetic biology*, 4(1):72–82, 2015.

[72] 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, 2005.

[73] Subhayu Basu, Rishabh Mehreja, Stephan Thiberge, Ming-Tang Chen, and Ron Weiss. Spatiotemporal control of gene expression with pulse-generating

networks. *Proceedings of the National Academy of Sciences*, 101(17):6355–6360, 2004.

[74] Andreia Carvalho, Diego Barcena Menendez, Vivek Raj Senthivel, Timo Zimmermann, Luis Diambra, and Mark Isalan. Genetically encoded sender–receiver system in 3d mammalian cell culture. *ACS synthetic biology*, 3(5):264–272, 2014.

[75] 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, 2012.

[76] Martín Gutiérrez, Paula Gregorio-Godoy, Guillermo Perez del Pulgar, Luis E Muñoz, Sandra Sáez, and Alfonso Rodríguez-Patón. A new improved and extended version of the multicell bacterial simulator gro. *ACS synthetic biology*, 6(8):1496–1508, 2017.

[77] Jonathan Pascalie, Martin Potier, Taras Kowaliw, Jean-Louis Giavitto, Olivier Michel, Antoine Spicher, and René Doursat. Developmental design of synthetic bacterial architectures by morphogenetic engineering. *ACS synthetic biology*, 5(8):842–861, 2016.

[78] Yangxiaolu Cao, Marc D Ryser, Stephen Payne, Bochong Li, Christopher V Rao, and Lingchong You. Collective space-sensing coordinates pattern scaling in engineered bacteria. *Cell*, 165(3):620–630, 2016.

[79] Isaac N Nuñez, Tamara F Matute, Ilenne D Del Valle, Anton Kan, Atri Choksi, Drew Endy, Jim Haseloff, Timothy J Rudge, and Fernan Federici. Artificial symmetry-breaking for morphogenetic engineering bacterial colonies. *ACS synthetic biology*, 6(2):256–265, 2017.

[80] Liyang Xiong, Yuansheng Cao, Robert Cooper, Wouter-Jan Rappel, Jeff Hasty, and Lev Tsimring. Flower-like patterns in multi-species bacterial colonies. *Elife*, 9:e48885, 2020.

[81] Lee R Lynd, Mark S Laser, David Bransby, Bruce E Dale, Brian Davison, Richard Hamilton, Michael Himmel, Martin Keller, James D McMillan, and John Sheehan. How biotech can transform biofuels. *Nat. Biotechnol.*, 26(2):169, 2008.

[82] Hyun-Dong Shin, Shara McClendon, Trinh Vo, and Rachel R Chen. Escherichia coli binary culture engineered for direct fermentation of hemicellulose to a biofuel. *Appl. Environ. Microbiol*, 76(24):8150–8159, 2010.

[83] Garima Goyal, Shen-Long Tsai, Bhawna Madan, Nancy A DaSilva, and Wilfred Chen. Simultaneous cell growth and ethanol production from cellulose by an engineered yeast consortium displaying a functional mini-cellulosome. *Microb. Cell Fact.*, 10(1):89, 2011.

[84] Chris J Paddon and Jay D Keasling. Semi-synthetic artemisinin: a model for the use of synthetic biology in pharmaceutical development. *Nat. Rev. Microbiol.*, 12(5):355, 2014.

[85] M Fujita, M Ike, and S Hashimoto. Feasibility of wastewater treatment using genetically engineered microorganisms. *Water Research*, 25(8):979–984, 1991.

[86] Mark A Eiteman, Sarah A Lee, and Elliot Altman. A co-fermentation strategy to consume sugar mixtures effectively. *J. Biol. Eng.*, 2(1):3, 2008.

[87] Kang Zhou, Kangjian Qiao, Steven Edgar, and Gregory Stephanopoulos. Distributing a metabolic pathway among a microbial consortium enhances production of natural products. *Nat. Biotechnol.*, 33(4):377, 2015.

[88] Katie Brenner, Lingchong You, and Frances H Arnold. Engineering microbial consortia: a new frontier in synthetic biology. *Trends Biotechnol.*, 26(9):483–489, 2008.

[89] Ahmad A Zeidan, Peter Rådström, and Ed WJ van Niel. Stable coexistence of two caldicellulosiruptor species in a de novo constructed hydrogen-producing co-culture. *Microb. Cell Fact.*, 9(1):102, 2010.

[90] Frederick K Balagaddé, Hao Song, Jun Ozaki, Cynthia H Collins, Matthew Barnet, Frances H Arnold, Stephen R Quake, and Lingchong You. A synthetic escherichia coli predator–prey ecosystem. *Mol. Syst. Biol.*, 4(1):187, 2008.

[91] Wenying Shou, Sri Ram, and Jose MG Vilar. Synthetic cooperation in engineered yeast populations. *Proc. Natl. Acad. Sci*, 104(6):1877–1882, 2007.

[92] Alex JH Fedorec, Behzad D Karkaria, Michael Sulu, and Chris P Barnes. Single strain control of microbial consortia. *Nature communications*, 12(1):1–12, 2021.

[93] Gang Wu, Qiang Yan, J Andrew Jones, Yinjie J Tang, Stephen S Fong, and Mattheos AG Koffas. Metabolic burden: cornerstones in synthetic biology and metabolic engineering applications. *Trends Biotechnol.*, 34(8):652–664, 2016.

[94] Rimvydas Simutis and Andreas Lübbert. Bioreactor control improves bioprocess performance. *Biotechnol. J.*, 10(8):1115–1130, 2015.

[95] J Prakash and K Srinivasan. Design of nonlinear pid controller and nonlinear model predictive controller for a continuous stirred tank reactor. *ISA Trans.*, 48(3):273–282, 2009.

[96] Guang-Yan Zhu, Abdelqader Zamamiri, Michael A Henson, and Martin A Hjortsø. Model predictive control of continuous yeast bioreactors using cell population balance models. *Chem. Eng. Sci.*, 55(24):6155–6167, 2000.

[97] S Ramaswamy, TJ Cutright, and HK Qammar. Control of a continuous bioreactor using model predictive control. *Process Biochem.*, 40(8):2763–2770, 2005.

[98] Iasson Karafyllis, Georgios Savvoglidis, Lemonia Syrou, Katerina Stamatelatou, Costas Kravaris, and Gerasimos Lyberatos. Global stabilization of continuous bioreactors. In *American Institute of Chemical Engineers-Annual Meeting, Sn. Francisco, USA*, 2006.

[99] Hernán De Battista, Martín Jamilis, Fabricio Garelli, and Jesús Picó. Global stabilisation of continuous bioreactors: Tools for analysis and design of feeding laws. *Automatica*, 89:340–348, 2018.

[100] Frédéric Mazenc, Jérôme Harmand, and Michael Malisoff. Stabilization in a chemostat with sampled and delayed measurements and uncertain growth functions. *Automatica*, 78:241–249, 2017.

[101] Karlene A Hoo and Jeffrey C Kantor. Global linearization and control of a mixed-culture bioreactor with competition and external inhibition. *Math. Biosci.*, 82(1):43–62, 1986.

[102] Neythen J Treloar, Alex JH Fedorec, Brian Ingalls, and Chris P Barnes. Deep reinforcement learning for the control of microbial co-cultures in bioreactors. *PLoS computational biology*, 16(4):e1007783, 2020.

[103] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[104] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[105] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[106] Sabino Gadaleta and Gerhard Dangelmayr. Learning to control a complex multistable system. *Phys. Rev. E*, 63(3):036217, 2001.

[107] Zhenpeng Zhou, Xiaocheng Li, and Richard N Zare. Optimizing chemical reactions with deep reinforcement learning. *ACS Cent. Sci.*, 2017.

[108] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[109] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[110] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[111] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv.org*, 2014.

[112] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv.org*, 7(1), 2015.

[113] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[114] Michael A Johnson and Mohammad H Moradi. *PID control*. Springer, 2005.

[115] MATLAB. *version 9.60.0 (R2019a)*. The MathWorks Inc., Natick, Massachusetts, 2019.

[116] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

[117] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[118] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[119] Jillian M Couto, Anne McGarrity, Julie Russell, and William T Sloan. The effect of metabolic stress on genome stability of a synthetic biology chassis escherichia coli k12 strain. *Microbial cell factories*, 17(1):8, 2018.

[120] Peter Rugbjerg, Nils Myling-Petersen, Andreas Porse, Kira Sarup-Lytzen, and Morten OA Sommer. Diverse genetic error modes constrain large-scale bio-based production. *Nature communications*, 9(1):1–14, 2018.

[121] Javier Macia and Ricard Sole. How to make a synthetic multicellular computer. *PLoS One*, 9(2):e81248, 2014.

[122] Yeqing Zong, Haoqian M Zhang, Cheng Lyu, Xiangyu Ji, Junran Hou, Xian Guo, Qi Ouyang, and Chunbo Lou. Insulated transcriptional elements enable precise design of genetic circuits. *Nature communications*, 8(1):1–13, 2017.

[123] Joy Doong, James Parkin, and Richard M Murray. Length and time scales of cell-cell signaling circuits in agar. 2017.

[124] Kathakali Sarkar, Deepro Bonnerjee, Rajkamal Srivastava, and Sangram Bagh. A single layer artificial neural network type architecture with molecular engineered bacteria for reversible and irreversible computing. *Chemical science*, 12(48):15821–15832, 2021.

[125] Jürgen Branke. Evolutionary algorithms for neural network design and training. In *In Proceedings of the first Nordic workshop on genetic algorithms and its applications*. Citeseer, 1995.

[126] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[127] Wolfram Research, Inc. Mathematica, Version 12.3.1. Champaign, IL, 2021.

[128] Joy Doong, James Parkin, and Richard M. Murray. Length and time scales of cell-cell signaling circuits in agar. *bioRxiv*, 2017.

[129] James E Baker et al. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms*, volume 206, pages 14–21, 1987.

[130] Stephen B Hanauer. Inflammatory bowel disease: epidemiology, pathogenesis, and therapeutic opportunities. *Inflammatory bowel diseases*, 12(suppl_1):S3–S9, 2006.

[131] Magdy El-Salhy. Irritable bowel syndrome: diagnosis and pathogenesis. *World journal of gastroenterology: WJG*, 18(37):5151, 2012.

[132] Julia MW Wong, Russell De Souza, Cyril WC Kendall, Azadeh Emam, and David JA Jenkins. Colonic health: fermentation and short chain fatty acids. *Journal of clinical gastroenterology*, 40(3):235–243, 2006.

[133] Kristina N-M Daeffler, Jeffrey D Galley, Ravi U Sheth, Laura C Ortiz-Velez, Christopher O Bibb, Noah F Shroyer, Robert A Britton, and Jeffrey J Tabor. Engineering bacterial thiosulfate and tetrathionate sensors for detecting gut inflammation. *Molecular systems biology*, 13(4):923, 2017.

[134] Christoph Küper and Kirsten Jung. Cadc-mediated activation of the cadba promoter in escherichia coli. *Journal of molecular microbiology and biotechnology*, 10(1):26–39, 2005.

[135] Tanel Ozdemir. *Design and construction of therapeutic bacterial sensors in Escherichia coli Nissle 1917*. PhD thesis, UCL (University College London), 2018.

[136] Olivier Bernard, Zakaria Hadj-Sadok, Denis Dochain, Antoine Genovesi, and Jean-Philippe Steyer. Dynamical model development and parameter identification for an anaerobic wastewater treatment process. *Biotechnology and bioengineering*, 75(4):424–438, 2001.

[137] Amanda N Payne, Annina Zihler, Christophe Chassard, and Christophe Lacroix. Advances and perspectives in in vitro human gut fermentation modeling. *Trends in biotechnology*, 30(1):17–25, 2012.

[138] Pascal Cougnon, Denis Dochain, Martin Guay, and Michel Perrier. On-line optimization of fedbatch bioreactors by adaptive extremum seeking control. *Journal of Process Control*, 21(10):1526–1532, 2011.

[139] L Syrou, I Karafyllis, K Stamatelatou, G Lyberatos, and C Kravaris. Robust global stabilization of continuous bioreactors. *IFAC Proceedings Volumes*, 37(9):995–1000, 2004.

[140] Aivar Sootla, Natalja Strelkowa, Damien Ernst, Mauricio Barahona, and Guy-Bart Stan. Toggling a genetic switch using reinforcement learning. *arXiv.org*, 2013.

[141] Thomas Lampe and Martin Riedmiller. Approximate model-assisted neural fitted q-iteration. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 2698–2704. IEEE, 2014.

[142] Damien Ernst, Guy-Bart Stan, Jorge Goncalves, and Louis Wehenkel. Clinical data based optimal sti strategies for hiv: a reinforcement learning approach. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 667–672. IEEE, 2006.

[143] Xuefeng Peng, Yi Ding, David Wihl, Omer Gottesman, Matthieu Komorowski, Li-wei H Lehman, Andrew Ross, Aldo Faisal, and Finale Doshi-Velez. Improving sepsis treatment strategies by combining deep and kernel-based reinforcement learning. In *AMIA Annual Symposium Proceedings*, volume 2018, page 887. American Medical Informatics Association, 2018.

[144] B Jaganatha Pandian and Mathew Mithra Noel. Control of a bioreactor using a new partially supervised reinforcement learning algorithm. *Journal of Process Control*, 69:16–29, 2018.

[145] MASAYUKI Seto and MARTIN Alexander. Effect of bacterial density and substrate concentration on yield coefficients. *Appl. Environ. Microbiol.*, 50(5):1132–1136, 1985.

[146] JD Owens and JD Legan. Determination of the monod substrate saturation constant for microbial growth. *FEMS Microbiol. Rev.*, 3(4):419–432, 1987.

[147] Robert A Cox. Quantitative relationships for specific growth rates and macromolecular compositions of mycobacterium tuberculosis, streptomyces coelicolor a3 (2) and escherichia coli b/r: an integrative theoretical approach. *Microbiology*, 150(5):1413–1426, 2004.

[148] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001.

[149] Andres C Rodriguez, Ronald Parr, and Daphne Koller. Reinforcement learning using approximate belief states. In *Advances in Neural Information Processing Systems*, pages 1036–1042, 2000.

[150] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[151] Brandon G Wong, Christopher P Mancuso, Szilvia Kiriakov, Caleb J Bashor, and Ahmad S Khalil. Precise, automated control of conditions for high-throughput growth of yeast and bacteria with evolver. *Nature biotechnology*, 36(7):614–623, 2018.

[152] Chris N Takahashi, Aaron W Miller, Felix Ekness, Maitreya J Dunham, and Eric Klavins. A low cost, customizable turbidostat for use in synthetic circuit characterization. *ACS synthetic biology*, 4(1):32–38, 2015.

[153] Stefan A Hoffmann, Christian Wohltat, Kristian M Müller, and Katja M Arndt. A user-friendly, low-cost turbidostat with versatile growth rate estimation based on an extended kalman filter. *PloS one*, 12(7), 2017.

[154] Harrison Steel, Robert Habgood, Ciarán Kelly, and Antonis Papachristodoulou. Chi. bio: An open-source automated experimental platform for biological science research. *bioRxiv*, page 796516, 2019.

[155] Kevin S Lee, Paolo Boccazzi, Anthony J Sinskey, and Rajeev J Ram. Microfluidic chemostat and turbidostat with flow rate, oxygen, and temperature control for dynamic continuous culture. *Lab on a Chip*, 11(10):1730–1739, 2011.

[156] Gustaf Ullman, Mats Wallden, Erik G Marklund, Anel Mahmutovic, Ivan Razinkov, and Johan Elf. High-throughput gene expression analysis at the level of single proteins using a microfluidic turbidostat and automated cell tracking. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 368(1611):20120025, 2013.

[157] Jason Karslake, Jeff Maltas, Peter Brumm, and Kevin B Wood. Population density modulates drug inhibition and gives rise to potential bistability of treatment outcomes for bacterial infections. *PLoS computational biology*, 12(10), 2016.

[158] Erdal Toprak, Adrian Veres, Jean-Baptiste Michel, Remy Chait, Daniel L Hartl, and Roy Kishony. Evolutionary paths to antibiotic resistance under dynamically sustained drug selection. *Nature genetics*, 44(1):101, 2012.

[159] Samay Pande, Holger Merker, Katrin Bohl, Michael Reichelt, Stefan Schuster, Luís F De Figueiredo, Christoph Kaleta, and Christian Kost. Fitness and stability of obligate cross-feeding interactions that emerge upon gene loss in bacteria. *The ISME journal*, 8(5):953–962, 2014.

[160] Colton J Lloyd, Zachary A King, Troy E Sandberg, Ying Hefner, Connor A Olson, Patrick V Phaneuf, Edward J O'Brien, Jon G Sanders, Rodolfo A Salido,

Karenina Sanders, et al. The genetic basis for adaptation of model-designed syntrophic co-cultures. *PLoS computational biology*, 15(3):e1006213, 2019.

[161] Xiaolin Zhang and Jennifer L Reed. Adaptive evolution of synthetic cooperating communities improves growth performance. *PloS one*, 9(10), 2014.

[162] Haoran Zhang, Brian Pereira, Zhengjun Li, and Gregory Stephanopoulos. Engineering escherichia coli coculture systems for the production of biochemical products. *Proceedings of the National Academy of Sciences*, 112(27):8266–8271, 2015.

[163] M Omar Din, Tal Danino, Arthur Prindle, Matt Skalak, Jangir Selimkhanov, Kaitlin Allen, Ellixis Julio, Eta Atolia, Lev S Tsimring, Sangeeta N Bhatia, et al. Synchronized cycles of bacterial lysis for in vivo delivery. *Nature*, 536(7614):81–85, 2016.

[164] Jérôme Izard, Cindy DC Gomez Balderas, Delphine Ropers, Stephan Lacour, Xiaohu Song, Yifan Yang, Ariel B Lindner, Johannes Geiselmann, and Hidde de Jong. A synthetic growth switch based on controlled expression of rna polymerase. *Molecular systems biology*, 11(11), 2015.

[165] Jeff Maltas and Kevin B Wood. Pervasive and diverse collateral sensitivity profiles inform optimal strategies to limit antibiotic resistance. *PLoS biology*, 17(10), 2019.

[166] Andrej Fischer, Ignacio Vázquez-García, and Ville Mustonen. The value of monitoring to control evolving populations. *Proceedings of the National Academy of Sciences*, 112(4):1007–1012, 2015.

[167] Edouard Pauwels, Christian Lajaunie, and Jean-Philippe Vert. A bayesian active learning strategy for sequential experimental design in systems biology. *BMC Systems Biology*, 8(1):1–11, 2014.

[168] Juliane Liepe, Paul Kirk, Sarah Filippi, Tina Toni, Chris P Barnes, and Michael PH Stumpf. A framework for parameter estimation and model selec-

tion from experimental data in systems biology using approximate bayesian computation. *Nature protocols*, 9(2):439–456, 2014.

[169] Nathan Braniff, Matthew Scott, and Brian Ingalls. Component characterization in a growth-dependent physiological context: optimal experimental design. *Processes*, 7(1):52, 2019.

[170] Nathan Braniff, Addison Richards, and Brian Ingalls. Optimal experimental design for a bistable gene regulatory network. *IFAC-PapersOnLine*, 52(26):255–261, 2019.

[171] Meng Fang, Yuan Li, and Trevor Cohn. Learning how to active learn: A deep reinforcement learning approach. *arXiv preprint arXiv:1708.02383*, 2017.

[172] Adam Yala. *Improving information extraction by acquiring external evidence with reinforcement learning*. PhD thesis, Massachusetts Institute of Technology, 2017.

[173] Tilman Barz, Diana C López Cárdenas, Harvey Arellano-Garcia, and Günter Wozny. Experimental evaluation of an approach to online redesign of experiments for parameter determination. *AIChE Journal*, 59(6):1981–1995, 2013.

[174] MN Cruz Bournazou, T Barz, DB Nickel, DC Lopez Cárdenas, F Glauche, A Knepper, and P Neubauer. Online optimal experimental re-design in robotic parallel fed-batch cultivation facilities. *Biotechnology and bioengineering*, 114(3):610–619, 2017.

[175] David Benjamin Nickel, Mariano Nicolas Cruz-Bournazou, Terrance Wilms, Peter Neubauer, and Andreas Knepper. Online bioprocess data generation, analysis, and optimization for parallel fed-batch fermentations in milliliter scale. Technical report, Wiley Online Library, 2017.

[176] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas

Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[177] Fabio Muratore, Felix Treede, Michael Gienger, and Jan Peters. Domain randomization for simulation-based policy optimization with transferability assessment. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 700–713. PMLR, 29–31 Oct 2018.

[178] Ankush Chakrabarty, Gregery T Buzzard, and Ann E Rundell. Model-based design of experiments for cellular processes. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 5(2):181–203, 2013.

[179] Nathan Braniff and Brian Ingalls. New opportunities for optimal design of dynamic experiments in systems and synthetic biology. *Current Opinion in Systems Biology*, 9:42–48, 2018.

[180] Juliane Liepe, Sarah Filippi, Michał Komorowski, and Michael PH Stumpf. Maximizing the information content of experiments in systems biology. *PLoS computational biology*, 9(1):e1002888, 2013.

[181] Anthony Atkinson, Alexander Donev, and Randall Tobias. *Optimum experimental designs, with SAS*, volume 34. Oxford University Press, 2007.

[182] Lucia Bandiera, David Gomez-Cabeza, James Gilman, Eva Balsa-Canto, and Filippo Menolascina. Optimally designed model selection for synthetic biology. *ACS Synthetic Biology*, 9(11):3134–3144, 2020.

[183] MATLAB. *9.7.0.1190202 (R2019b).* The MathWorks Inc., Natick, Massachusetts, 2018.

[184] Alejandro F Villaverde, Antonio Barreiro, and Antonis Papachristodoulou. Structural identifiability of dynamic systems biology models. *PLoS computational biology*, 12(10):e1005153, 2016.