ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA
SECONDA FACOLTÀ DI INGEGNERIA CON SEDE A CESENA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA BIOMEDICA

TITOLO DELLA TESI

MODELING AND SIMULATION OF A SYNTHETIC GENETIC CIRCUIT THAT IMPLEMENTS A MULTICELLED BEHAVIOR IN A GROWING MICROCOLONY OF E. COLI

Tesi in:
Bioingegneria Molecolare e Cellulare LM

Relatore
Prof. Ing. Stefano Severi

Correlatori
Prof. Emanuele Domenico Giordano
Ing. Seunghee Shelly Jang
Prof. Ing. Eric Klavins

Presentata da
Andrea Samoré

Sessione III
Anno Accademico 2011/2012

Contents

1 Molecular and Synthetic Biology **12**
- 1.1 Nucleic Acids 12
- 1.2 DNA 13
- 1.3 RNA 15
- 1.4 Transcription 16
- 1.5 Translation 17
- 1.6 Synthetic Biology 18
  - 1.6.1 Gene Expression Regulation 19
  - 1.6.2 Modeling 22

2 Leader Election Project **24**
- 2.1 Introduction 24
- 2.2 Genetic Circuit 25
- 2.3 Genetic components 26

3 Mathematical Model **31**
- 3.1 Input functions 31
  - 3.1.1 Promoter regulated by a repressor 31
  - 3.1.2 Promoter regulated by an activator 33
  - 3.1.3 Binding of an inducer to a transcription factor 33
  - 3.1.4 Hybrid promoter regulated by a repressor and an activator 35
- 3.2 Enzyme Kinetics 36
- 3.3 Model of the Leader Election Circuit 38
  - 3.3.1 Assumptions 38
  - 3.3.2 Terms of the model 38
  - 3.3.3 Parameters 41

4 Simulations and Results **43**
- 4.1 gro environment 43
- 4.2 Sensitivity Analysis 45

4.2.1 Cost Function 47
4.2.2 Ranking 49
4.3 Results 51
4.4 Conclusions 60
4.5 Future application: pattern formation 61
4.5.1 Spots 63
4.5.2 Rings 66
4.5.3 Considerations 68

5 Appendix **69**
5.1 Python script to split simulations on a Linux cluster 69
5.2 gro Models 78
5.2.1 Leader Election 78
5.2.2 Spots 83
5.2.3 Rings 84
5.3 Matlab Code 85

List of Figures

1.1 Nucleotides [1] 12
1.2 DNA chemical structure [2] 14
1.3 Scanning electron micrograph of Escherichia coli [3] 18
1.4 Structure of a bacterial promoter 20

2.1 Example of the wanted behavior. Leader cells are green, followers are red and undecided are grey. 24
2.2 Proposed genetic circuit of the Leader Election project [4] . . 25
2.3 L-Arabinose 27
2.4 General structure of N-Acyl Homoserine Lactone (AHL) . . . 29

3.1 Input function of a gene regulated by a repressor with $k_d = 10$ 33
3.2 Input function of a gene regulated by an activator with $k_d = 10$ 34
3.3 Input function of an hybrid promoter with $k_{dA} = k_{dR} = 10$ and n = 2 36

4.1 Simulation of stochastic production and degradation of red fluorescent protein (RFP). The cells are genetically identical but the amount of protein differs slightly between individuals, note the different red intensity 44
4.2 Example of sensitivity plot. The parameter β_{LuxI} is varied over about two orders of magnitude around its nominal value, depicted as a yellow square. 48
4.3 Fitting of the cost function 48
4.4 Structure of the scatter plot used to rank the parameters. On the left, the information that has been used to place the β_{LuxI} dot in the right-hand graph. 49
4.5 The four zones of the scatter plot used to rank the parameters 50
4.6 Scatter plots when the goal is set to a majority of followers. . 52

4.7 The goal is set to a majority of followers. Left-hand column: cost function of the four most promising parameters. Right-hand column: the corresponding fraction of leaders, followers and undecided cells are depicted in green, red and black. . . . 54
4.8 Scatter plot when the goal is to obtain a population in which there are about as many leaders as followers. 55
4.9 The goal is set to a colony split in half between leaders and followers. Left-hand column: cost function of the four most promising parameters. Right-hand column: the corresponding fraction of leaders, followers and undecided cells are depicted in green, red and black. 56
4.10 Scatter plot when the goal is set to a majority of leaders. . . . 58
4.11 The goal is set to a majority of leaders. Left-hand column: cost function of the four most promising parameters. Right-hand column: the corresponding fraction of leaders, followers and undecided cells are depicted in green, red and black. . . . 59
4.12 Example of pattern created by a Ben-Jacob's bacterial strain. The image of P. vortex colony was created at Prof. Ben-Jacob's lab, at Tel-Aviv University, Israel 62
4.13 Simulation of the spots-forming finite state machine 65
4.14 Simulation of the rings-forming finite state machine 67

5.1 Python script diagram 72

Sommario

La Biologia Sintetica è una disciplina relativamente nuova, nata nei primi anni duemila, che porta il tipico approccio ingegneristico al campo delle biotecnologie: astrazione, modularità e standardizzazione vengono utilizzati per tentare di domare l'estrema complessità dei componenti e costruire sistemi biologici artificiali con una funzione definita. Questi sistemi, tipicamente circuiti genetici sintetici, vengono perlopiù implementati in batteri e semplici organismi eucariotici come ad esempio i lieviti. La cellula diventa quindi una macchina programmabile ed il suo linguaggio macchina è costituito da sequenze nucleotidiche.

Il lavoro di tesi è stato svolto in collaborazione con ricercatori del Department of Electrical Engineering dell'Università di Washington a Seattle e con una studentessa del Corso di Laurea in Ingegneria Biomedica dell'Università di Bologna: Marilisa Cortesi. Nell'ambito della collaborazione ho contribuito ad un progetto di Biologia Sintetica già avviato nel Klavins Lab, in particolare mi sono occupato della modellazione matematica e simulazione di un circuito genetico sintetico pensato per implementare un comportamento multicellulare in una microcolonia batterica.

Nel Primo Capitolo sono introdotte le basi della biologia molecolare, in particolare si accenna alla struttura degli acidi nucleici e vengono illustrati i processi di trascrizione e traduzione che danno luogo all'espressione genica. Sono inoltre enunciati i principali meccanismi di regolazione dell'espressione genica sia al livello trascrizionale che traduzionale. Un'introduzione alla biologia sintetica completa la sezione.

Nel Secondo capitolo è descritto il circuito genetico sintetico pensato per far emergere spontaneamente due gruppi di cellule differenti, detti leaders e followers, a partire da una colonia isogenica. Il circuito si basa sull'intrinseca stocasticità dell'espressione genica e sulla comunicazione intercellulare per mezzo di una piccola molecola segnale per rompere la simmetria nel fenotipo della microcolonia. Sono illustrati inoltre i quattro moduli di cui il circuito si compone (coin flipper, sender, receiver e follower) e le loro interazioni.

Nel Terzo Capitolo viene esposta la derivazione del modello matematico

dei singoli componenti del circuito genetico sintetico. Vengono poi esplicitate le varie assunzioni semplificative che si sono rivelate necessarie al fine di ridurne la complessità e quindi permetterne la simulazione. Trascrizione e traduzione sono modellate in un unico passo e l'espressione dei vari geni dipende dalla concentrazione intracellulare dei fattori di trascrizione che agiscono sui promotori utilizzati. Sono infine elencati i valori dei vari parametri e le fonti da cui sono stati ricavati.

Nel Quarto Capitolo sono dapprima descritte le caratteristiche principali dell'ambiente di simulazione, *gro*, sviluppato dal Self Organizing Systems Laboratory dell'Università di Washington. Viene poi dettagliata l'analisi di sensitività svolta per individuare quali caratteristiche dei diversi componenti genetici sono desiderabili per il funzionamento del circuito. In particolare, è definita una funzione costo basata sia sul numero di cellule che si trovano in ognuno dei vari stati possibili al termine della simulazione che sul risultato voluto. In base alla funzione costo e tramite un tipo particolare di scatter plot, viene stilata una classifica di parametri. A partire da una condizione iniziale in cui i parametri assumono valori in un ordine di grandezza compatibile con le informazioni attualmente disponibili nella letteratura scientifica, questa classifica suggerisce quale componente genetico conviene regolare al fine di ottenere il risultato voluto. Il comportamento per cui il circuito è stato ideato, ottenere una colonia in cui la quasi totalità di cellule siano nello stato follower e solo qualcuna nello stato leader, sembra essere il più difficile da raggiungere. Poche cellule leader non riescono a produrre abbastanza segnale per far passare il resto della colonia nello stato follower. Per ottenere una colonia in cui la maggioranza di cellule sia nello stato follower è necessario aumentare il più possible la produzione dell'enzima che genera il segnale. Ottenere una colonia in cui metà delle cellule sia nello stato leader e l'altra metà nello stato follower è più semplice. La strategia più promettente sembra essere aumentare leggermente la produzione di enzima. Per ottenere una maggioranza di cellule leader, invece, è consigliabile aumentare l'espressione basale dei geni nel modulo coin flipper. Al termine del capitolo, una possibile applicazione futura del circuito genetico sintetico, la formazione spontanea di pattern spaziali in una microcolonia, è modellata ad un alto livello di astrazione tramite il formalismo degli automi a stati finiti. La simulazione in *gro* fornisce indicazioni sui componenti genetici non ancora disponibili e che è quindi necessario sviluppare al fine di ottenere questo comportamento. In particolare, dato che entrambi gli esempi di pattern proposti si basano su una versione locale di Leader Election, è essenziale utilizzare un metodo di comunicazione intercellulare a corto raggio. Risulta inoltre fondamentale sviluppare componenti genetici che permettano di rallentare la crescita di specifiche cellule senza alterarne la capacità di espressione genica.

L'Appendice, infine, contiene il codice utilizzato per simulare il modello in *gro*, il listato di uno script Python utile per parallelizzare l'analisi di sensitività su un cluster Linux ed il codice Matlab con cui sono stati elaborati i dati provenienti dall'analisi di sensitività.

Abstract

Synthetic Biology is a relatively new discipline, born at the beginning of the New Millennium, that brings the typical engineering approach (abstraction, modularity and standardization) to biotechnology. These principles aim to tame the extreme complexity of the various components and aid the construction of artificial biological systems with specific functions, usually by means of synthetic genetic circuits implemented in bacteria or simple eukaryotes like yeast. The cell becomes a programmable machine and its low-level programming language is made of strings of DNA.

This work was performed in collaboration with researchers of the Department of Electrical Engineering of the University of Washington in Seattle and also with a student of the Corso di Laurea Magistrale in Ingegneria Biomedica at the University of Bologna: Marilisa Cortesi. During the collaboration I contributed to a Synthetic Biology project already started in the Klavins Laboratory. In particular, I modeled and subsequently simulated a synthetic genetic circuit that was ideated for the implementation of a multicelled behavior in a growing bacterial microcolony.

In the first chapter the foundations of molecular biology are introduced: structure of the nucleic acids, transcription, translation and methods to regulate gene expression. An introduction to Synthetic Biology completes the section.

In the second chapter is described the synthetic genetic circuit that was conceived to make spontaneously emerge, from an isogenic microcolony of bacteria, two different groups of cells, termed leaders and followers. The circuit exploits the intrinsic stochasticity of gene expression and intercellular communication via small molecules to break the symmetry in the phenotype of the microcolony. The four modules of the circuit (coin flipper, sender, receiver and follower) and their interactions are then illustrated.

In the third chapter is derived the mathematical representation of the various components of the circuit and the several simplifying assumptions are made explicit. Transcription and translation are modeled as a single step and gene expression is function of the intracellular concentration of the

various transcription factors that act on the different promoters of the circuit. A list of the various parameters and a justification for their value closes the chapter.

In the fourth chapter are described the main characteristics of the *gro* simulation environment, developed by the Self Organizing Systems Laboratory of the University of Washington. Then, a sensitivity analysis performed to pinpoint the desirable characteristics of the various genetic components is detailed. The sensitivity analysis makes use of a cost function that is based on the fraction of cells in each one of the different possible states at the end of the simulation and the wanted outcome. Thanks to a particular kind of scatter plot, the parameters are ranked. Starting from an initial condition in which all the parameters assume their nominal value, the ranking suggest which parameter to tune in order to reach the goal. Obtaining a microcolony in which almost all the cells are in the follower state and only a few in the leader state seems to be the most difficult task. A small number of leader cells struggle to produce enough signal to turn the rest of the microcolony in the follower state. It is possible to obtain a microcolony in which the majority of cells are followers by increasing as much as possible the production of signal. Reaching the goal of a microcolony that is split in half between leaders and followers is comparatively easy. The best strategy seems to be increasing slightly the production of the enzyme. To end up with a majority of leaders, instead, it is advisable to increase the basal expression of the coin flipper module. At the end of the chapter, a possible future application of the leader election circuit, the spontaneous formation of spatial patterns in a microcolony, is modeled with the finite state machine formalism. The *gro* simulations provide insights into the genetic components that are needed to implement the behavior. In particular, since both the examples of pattern formation rely on a local version of Leader Election, a short-range communication system is essential. Moreover, new synthetic components that allow to reliably downregulate the growth rate in specific cells without side effects need to be developed.

In the appendix are listed the *gro* code utilized to simulate the model of the circuit, a script in the Python programming language that was used to split the simulations on a Linux cluster and the Matlab code developed to analyze the data.

Ringraziamenti

Desidero qui ringraziare in primo luogo la mia famiglia, per tutto il supporto fornitomi da sempre ma in particolare in questi anni di studio universitario, senza il loro appoggio sarebbe stato tutto più difficile.

Un ringraziamento speciale va a Marilisa per il continuo incoraggiamento, le discussioni, le risate, il tempo passato insieme. Senza di lei questa tesi non sarebbe stata possibile.

Un grazie enorme al Prof. Stefano Severi, che con la sua infinita disponibilità ha reso realtà il sogno di un periodo di studio all'estero. Grazie mille anche al Prof. Emanuele Giordano e alla Dottoressa Francesca Ceroni per i vari consigli ed il loro entusiasmo. Grazie a Shelly Jang ed Eric Klavins per avermi gentilmente accolto nel loro gruppo di ricerca per un periodo di diversi mesi, durante i quali ho imparato molto ed appreso un approccio diverso ai problemi.

Grazie a Yaoyu Yang per la cena in un ristorante giapponese in cui mi ha fatto scoprire il sake! Grazie a William e Kristin, per tutto il tempo passato insieme a Seattle, le chiaccherate (metà in italiano e metà in inglese) sulle differenze fra Stati Uniti ed Italia, le birre, la casa stregata e la caccia agli zombies!

Grazie a Gianluca Selvaggio, che dal Portogallo ha sempre diffuso buon umore attraverso Skype e lo stesso ha fatto Claudio Silvani, però dall'Italia. Grazie infine a tutti gli amici che sono entrati nella mia vita ed hanno contribuito a rendermi quello che sono.

Chapter 1

Molecular and Synthetic Biology

1.1 Nucleic Acids

Nucleic Acids are linear macromolecules obtained by assembling, through covalent bond, simple building blocks named nucleotides. Each nucleotide is composed of three fundamental units: a pentose sugar, ribose o deoxiribose, a nitrogenous base in position 1', and a phosphoric acid esterified to the alcoholic group of the sugar in position 5' (Figure 1.1). Depending on the sugar utilized, the nucleotides are classified in deoxiribonucleotides and ribonucleotides. A molecule composed of the pentose sugar and a nitrogenous base, bound in position 1', is called nucleoside.

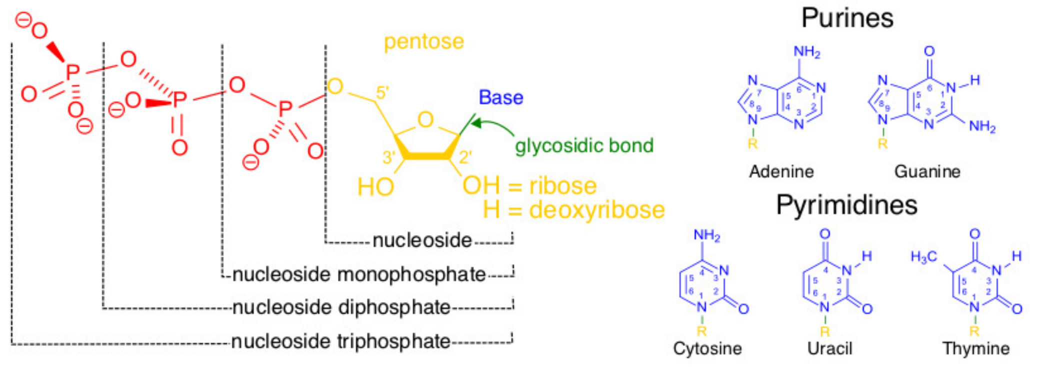


Figure 1.1: Nucleotides [1]

The DNA is formed of deoxyribonucleotides, while the RNA is composed of ribonucleotides. There are five nitrogenous bases that are most commonly used in the construction of nucleic acids: adenine (A), guanine (G), cytosine (C) and thymine (T) in the DNA, adenine (A), guanine (G), cytosine (C) and uracil (U) in the RNA, where the uracil substitutes the thymine.

1.2 DNA

The deoxyribonucleic acid (DNA) is the genetic material of the cell, it contains all the information necessary for protein synthesis and for the regulation of the cell's functions. The structure of the DNA of prokaryotic cells is very simple, it is indeed formed of a single circular chromosome free in the cytosol that is not associated with any protein nor organized in complex structures, unlike the eukaryotic ones. The DNA has a fundamental role because it allows to describe the entire cell:

- codes all the information necessary to the life of the cell;
- its structure allows for a simple and elegant transmission of all the information needed to build an organism from a generation to the next;
- directs and controls the entire vital cycle of the cell;
- can rarely mutate, change its description, in order to modify the information that it codifies and thus, generate an evolution of the functions.

The DNA is composed of two filaments with complementary orientation: for every G in a filament there's a C in the corresponding position in the complementary filament, and vice-versa. Every A of a filament is associated to a T and the other way around. The interaction between A and T and between C and G is specific and stable: the nitrogenous base guanine, with its double-ringed structure, is too big to fit in the space between the two filaments of the DNA, if coupled with the double ring of the adenine or with another guanine. Likewise, the nitrogenous base thymine, with its single-ringed structure, is too small to pair with another single-ringed base like cytosine or another thymine. Only the nitrogenous bases C and G, A and T have the appropriate spatial conformation and chemical interaction needed to form a stable base pairing (hydrogen bonds). Three hydrogen bonds form between C and G, and only two between A and T. The two DNA filaments are not just complementary but even antiparallel (Figure 1.2).

Triplets of nitrogen bases (codons) code for the twenty different amino acids that compose proteins. Most of the amino acids are coded by more than one triplet ($4^3 = 64$), three triplets don't represent any amino acid and

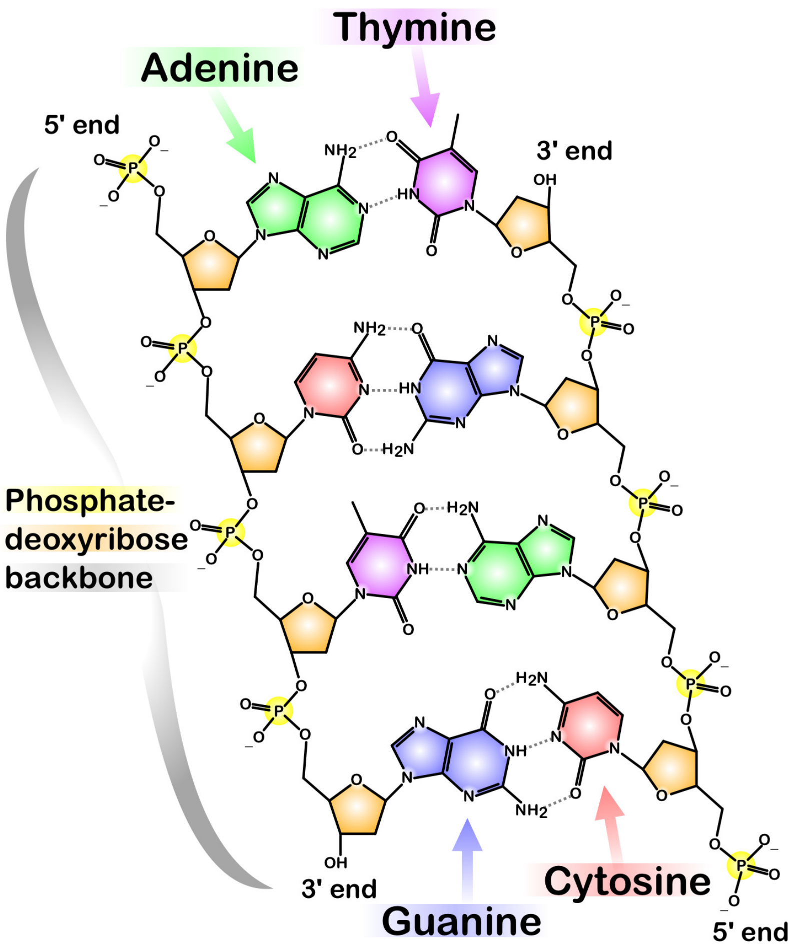


Figure 1.2: DNA chemical structure [2]

are used as a stop signal for translation. This redundant code makes the genetic information robust with respect to single nucleotide mutations.

Except for the mitochondrial DNA and the one of a small number of prokaryotes, the genetic material is universal, meaning that it follows the same rules in every living organism and virus.

1.3 RNA

The RNA has a structure that is very similar to that of DNA, in fact the genetic code of some viruses is entirely composed of RNA. However, it has assumed a totally different role in more complex organisms and so cells whose chromosomes are made of RNA do not exist.

There are three main differences between ribonucleic acid and deoxyribonucleic acid, in particular, the RNA:

- doesn't usually assume the three dimensional double helix structure typical of the DNA;
- contains ribose and not deoxyribose;
- contains the base uracil in place of thymine.

All the RNA present in the cell is synthesized from a DNA mold by particular enzymes, RNA polymerases, while its degradation is performed by another group of enzymes, the ribonucleases.

In a prokaryotic cell there are, in different quantities, three types of RNA: messenger RNA, ribosomal RNA and transfer RNA. Each kind of RNA has different functions:

- the messenger RNA (mRNA) provides to the protein synthesis apparatus a copy of the message codified in a gene of the DNA. The mRNA represent only a small fraction of the RNA present in the cell, even because a single RNA molecule can be used as a mold for many copies of the protein that it codifies;
- the ribosomal RNA (rRNA) is the type of RNA that has the highest concentration in the cell as it is part of the ribosomes, organelles that decode RNA and synthesize proteins. The ribosomes that are found in prokaryotic cells contain three different rRNAs named, after their sedimentation coefficient, 23S, 16S and 5S.

- the transfer RNA (tRNA) is composed of small molecules of ribonucleic acid that bind specifically and activate the single amino acids, while bringing them to the mRNA-ribosome complex, to become part of the forming polypeptidic chain. For every amino acid involved in protein synthesis there's, at least, a specific tRNA.

1.4 Transcription

The RNA is synthesized from molds of DNA through the activity of the RNA polymerase. The RNA polymerase that can be found in E. coli cells is a protein complex formed by four subunits, named α_2, β, β' and σ. The complex $\alpha_2\beta\beta'$ contains the catalytic site and the sites responsible to bind the DNA. The σ subunit, instead, is involved in the first steps of transcription: it is able to recognize the beginning of transcription due to the presence of a DNA region named promoter, and helps the opening of the double helix. The σ subunit is released when the RNA synthesis begins, while the remainder of the complex continues the synthesis.

In order to fulfill its function, the RNA polymerase needs triphosphate nucleotides, the DNA sequence to be copied and bivalent ions (Mg^{2+}, Mn^{2+}). The RNA polymerase scans the DNA sequence, looking for a specific sequence that marks the beginning of the region to transcribe. The σ subunit of the RNA polymerase recognizes the consensus sequences of the promoter, placed 35 and 10 nucleotides upstream of the beginning of transcription.

The higher the affinity between a particular promoter sequence and the σ subunit of the RNA polymerase, the higher will be the frequency of transcription of that gene. There are seven different σ subunits in the E. coli cell and the possibility of building RNA polymerases with significantly different σ factors allows the cell to activate or deactivate entire systems of genes with similar promoters.

The proteins produced by some genes are useful only in combination with other proteins. For this reason some groups of prokaryotic genes are under the control of the same promoter, this structure is called operon. A typical example is the lactose operon that contains three genes involved in the metabolism of this sugar. The transcription of an operon produces a long mRNA molecule that will be used by the ribosomes to synthesize the various proteins.

Some regulatory proteins, transcription factors, can precisely modulate transcription in response to external stimuli. The lactose operon, for example, has a promoter recognized by the $\sigma 70$ RNA polymerase but its expression is maximized when the environment is rich in lactose and devoid of glucose.

When the levels of lactose are low, the protein LacI binds to a specific sequence in the lac promoter just downstream of the -10 box, termed operator site. When LacI is bound to the operator, the steric bulk prevents the RNA polymerase from transcribing the downstream sequence, therefore it acts as a negative regulator (repressor). LacI is even able to bind allolactose, a lactose metabolite. When that happens the affinity of LacI for the operator site drastically diminishes thus increasing the probability of transcription of the genes of the operon. The consensus sequences of the promoter driving the lactose operon are not very similar to the ones better recognized by the RNA polymerase, so the lactose operon isn't expressed at high levels even when LacI is not bound to the promoter. The receptor protein for the cyclic AMP (CRP) that, when the levels of glucose are low, is able to bind a sequence in the lac promoter, increases the transcription rate and thus acts as a positive regulator (activator).

1.5 Translation

The translation process, that leads to protein synthesis, can be subdivided in three steps: initiation, elongation and termination. In the initial phase the ribosome finds the point at which translation starts by recognizing a particular sequence of nucleotides in the mRNA, named ribosome binding site (RBS), to which it binds. The elongation phase consist of a sequence of iterated reactions:

- combination of aminoacyl-tRNA, ribosome subunits, other proteic factors and the codon in the mRNA;
- formation of the peptide bond between the α-amminic group, of the amino acid bound to the tRNA, and the α-carboxylic one of the last amino acid of the polypeptidic chain forming on the ribosome. This causes the release of the tRNA bound to the second to last amino acid added to the chain;
- sliding of the ribosome on the mRNA till the next codon;

The termination phase, that begins when the ribosome reads one of the stop codons, causes the release of the polypeptidic chain. The correct implementation of these phases rely on the contribution of both proteic and non proteic factors.

1.6 Synthetic Biology

Synthetic biology is a relatively new discipline, founded at the beginning of the 2000s with the realization of the Repressilator [5] and the Toggle Switch [6]. It aims to engineer biology: the goal is to create a systematic engineering science, founded on the standardization of cellular chassis, the types of parts available, their manufacture, their characterization and protocols for their interconnection, analogous to those that underlie and enable the scalability of mechanical, electrical and civil engineering [7]. In order to do that, the engineers involved in this new field have started to apply some of the classical principles of engineering to biology: standardization, decoupling and abstraction.

The most used organism in synthetic biology applications is Escherichia coli (Figure 1.3), a gram negative bacteria often found in the intestine of warm blooded organisms. Most E. coli strains are not pathogens, but some of them are cause of acute food poisoning.

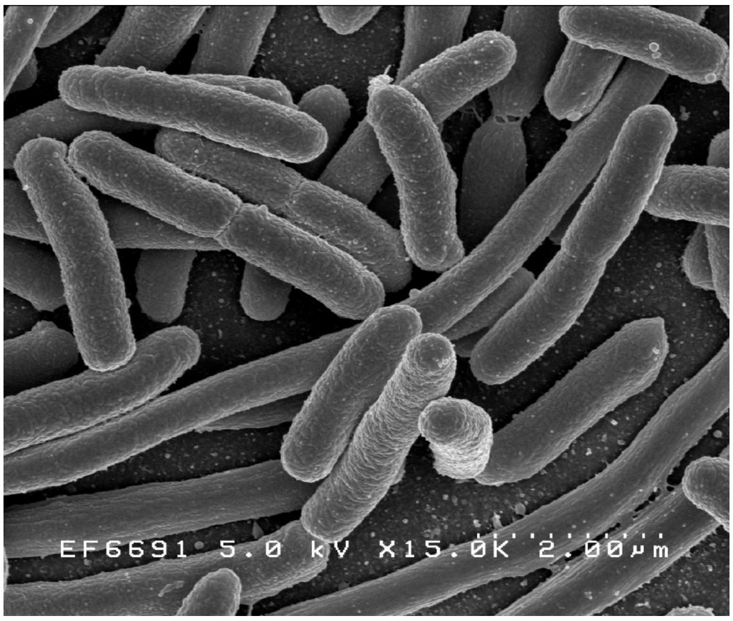


Figure 1.3: Scanning electron micrograph of Escherichia coli [3]

This bacterium plays a very important role in biotechnology and synthetic biology because it is quite easy to handle and it has been very widely used as laboratory organism. The work of Stanley Norman Cohen and Herbert Boyer, in which plasmids and restriction enzymes are used to build recombinant DNA in E. coli, is one of the foundations of biotechnology.

After being studied for over sixty years, E. coli is the organism better understood at molecular level and most of what is known about molecular processes can be ascribed to fundamental discoveries made in E. coli. Tamed strains, like K12, are well adapted to the laboratory environment and, unlike the wild type strains, have lost their capability to proliferate in the intestine and form biofilms.

The aim of a synthetic biology project is usually to build a synthetic genetic circuit that implements a particular function inside the cell. Regulation of gene expression at both the transcription and translation level is the chief way to make a group of genes solve a defined task.

1.6.1 Gene Expression Regulation

The necessity to tune gene expression and adapt it to the changes in the environment, has pushed the cell to develop mechanisms to change the rate of production of the different proteins. Synthetic biology exploits the regulatory elements of the cell to achieve a specific objective. It is important to remember that all the tuning methods that are described in the following sections can be combined in order to obtain the desired expression rate.

Regulation of Transcription

Regulation of transcription is needed to tune the amount of mRNA that is produced by the molecular machinery in a defined amount of time. It is mainly accomplished by modifying the promoter region of the considered gene or group of genes. The typical structure of a promoter is represented in Figure 1.4, the regions identified with -35 and -10 are the fundamental components of every promoter, since they are the sequences recognized by the σ subunit of the RNA polymerase. Their activity can be modulated by substituting single bases of the standard sequences for these elements, or by changing their relative distance.

A modification of the sequence of these regions usually diminishes the affinity between the RNA polymerase and the promoter, this decreases the rate of transcription and, as a consequence, the amount of mRNA available for translation. Indirectly, the concentration of the protein inside the cell will diminish. The variation in gene expression due to a different length of the core sequence is more difficult to predict. Intuitively there will be an optimal spacing, defined by the distance between the DNA-bounding regions of the RNA-polymerase, and any significant variation from that value will reduce the transcriptional strength of that promoter. If the promoter is

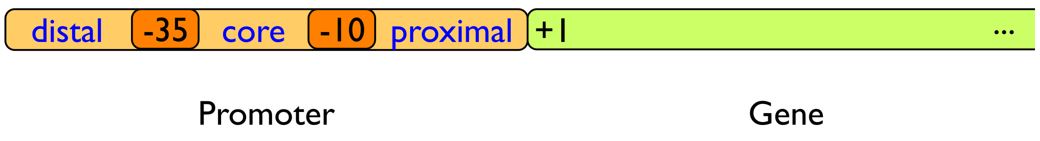


Figure 1.4: Structure of a bacterial promoter

constitutive, modifying these regions is the main way to tune gene expression at the transcriptional level.

A regulated promoter is a promoter whose action is modulated by one or more transcription factors. A transcription factor is a molecule that generally conveys an important information about an event, like a change in the environment in which the cell resides. This class of promoters is fundamental because it allows the cell to adapt the protein production to a necessity that changes in time.

The structure of a regulated promoter is the same as the one reported in Figure 1.4, the -10 and -35 boxes still have the same function and they can be used to control gene expression in the same way. In this case, though, there is an additional layer of regulation: the promoter contains even consensus sequences for the transcription factor, called operator sites. The transcription factor can bind to these sequences that are usually placed in one, or more, of the other regions of the promoter (Figure 1.4). Some consensus sequences are even found far from the promoter, both upstream and downstream, but they usually are functional only if coupled with another operator site in the promoter region. Due to the secondary importance of these sites, and the difficulty of defining clearly their activity, they are used very rarely for synthetic applications.

A transcription factor can be either an activator or a repressor, this means that there are transcription factors that promote the transcription of a gene, increasing the rate of transcription upon binding, and others that, when bound, reduce the production of mRNA, usually by preventing the RNA polymerase-promoter complex formation.

The region of the promoter in which an operator site is placed has a fundamental effect on its functionality. Repressors usually work by physically hampering the polymerase, thus preventing it from transcribing the DNA, so they will be maximally effective in the proximal or core regions. Activators, on the other hand, increase the transcription by favoring the binding between the RNA-polymerase and the promoter, so they are usually placed in distal, where they can carry out their function without unintentionally obstruct the promoter.

This kind of regulation is incredibly specific, most promoters can respond to at most two transcription factors and the sensitivity and the strength of the modulation can be tuned by changing the consensus sequence and/or its location. In order to utilize a particular regulated promoter in a synthetic circuit and avoid unwanted interactions, it is necessary to understand its functioning in the bacterial environment. The activity of several transcription factors can be modulated by the interaction with a number of small molecules, named inducers. Several examples of this class of regulatory elements will be described in the following chapter.

Regulation of Translation

Regulation of translation can be mainly achieved by acting on the Shine-Dalgarno region, also named ribosome binding site (RBS) after its function. The interaction between the ribosome and the RNA is quite well understood, computational models allow to predict the translational efficiency of a particular RBS and to design new ones with defined strength. The possibility to tune the level of protein produced is extremely important in most synthetic biology applications, and this technique is often more accurate than the ones that act at transcriptional level, since it affects directly the translation process. Modifying the RBS region can be really effective to place the protein production in the desired order of magnitude, but its action is too coarse to tune it finely.

Another technique to regulate gene expression at the translational level that has been recently devised consists in modifying the length of the spacer between the RBS and the first codon of the sequence of the protein [8]. By increasing the span of this region it is possible to down-regulate the rate of translational initiation, since the relative positions of the Shine-Dalgarno region and the first codon will not be optimal. This method has been tested in E. coli using simple sequence repeats (SSR) to alter the spacer. The use of simple sequence repeats couples the translational regulation of gene expression with an increase of the mutation rate of the spacer region, because repeated sequences have a strong bias for insertions/deletions. This second aspect of the tuning technique allows to take advantage of evolution to optimize the length of the spacer region.

The SSR used to implement this kind of regulation are composed of the repetition of one or two nucleotides and, from the characterization performed in [8], it is clear the possibility to tune gene expression over several orders of magnitude. In the same paper it is even described how the nucleotidic composition of the SSR can influence the decrease of the translational initiation rate. In particular, a SSR composed of only adenines will have the steepest decline, while a poly-thymines sequence should grant the most gradual decline.

Sequence repeats seem ideal to tune gene expression because the relation between the length of the sequence and its effect on translation is very well defined. Besides, the regulatory range achievable by coupling this method with other techniques, like promoter engineering, is very large. The choice of using repetitions of nucleotides makes it really easy to experimentally sample the expression space, through combinatorial modifications the SSR region.

A third way to regulate gene expression at the translational level utilizes small antisense RNAs that target specific mRNAs in the cell. The antisense RNA is usually a short ribonucleotidic sequence that binds to a transcribed mRNA. The steric bulk interferes with translation while double stranded RNA is targeted for degradation. This kind of regulation is faster than transcriptional regulation mediated by transcription factors because it removes from the cytoplasm genes that have been already transcribed.

1.6.2 Modeling

The application of the previously mentioned engineering principles is greatly limited by various factors [9]:

- inability to avoid or manage biological complexity;
- tedious and unreliable construction and characterization of synthetic biological systems;
- evolution.

As a consequence, even simple modules can take a significant amount of time and resources to construct from devices, often requiring multiple revisions to optimize the behavior. Modeling greatly aids in overcoming module design problems [10].

An accurate computational representation of the system can help devise reliable synthetic genetic circuits by determining, for example, which architecture is the most robust or the one that better adapts to a certain application. A mathematical model might even be fundamental for the characterization of the system, since it might be able to identify the most critical components and provide useful suggestions about the assays necessary to completely analyze the behavior of the circuit.

Simulations have two fundamental advantages over the wet laboratory: they are usually much cheaper and also quicker than experiments. While a certain minimum amount of experiments will be required for a particular study, models can help reduce their number by scanning a wide spectrum of possible components of the circuit or different experimental conditions, and allow to select only the most promising options to test in vivo. Shrinking the number of experiments means reducing the cost of the endeavor both in time and money required.

Another nice feature of the computational representation of the circuit is the possibility of having complete control over the virtual experiment and being able to extract values of quantities not actually measurable with laboratory techniques. This ability extend the usefulness of the model even to the troubleshooting phase of the construction of a genetic component. Having direct access to every intracellular part and process can be really helpful to identify the source of unexpected or undesired behavior.

In order to be useful, the model needs to faithfully represent the biological system, at least in the aspects that need to be investigated. This might mean that it is going to be necessary to build more than one model for the same circuit, to accurately capture each phenomenon. The same biological system can be described with different mathematical formalisms, but even with different parameters' sets, that define the regime in which the system operates. In order to find the better composition of mathematical representation and values for the parameters it is necessary to couple the realization of the model to the biological system. Direct or indirect measures of some characteristics of the genetic components can be used to define the system's working point in the parameter's space or, at least, define the physiological ranges of the quantities involved. Even if this phase might be very challenging, especially when it is necessary to combine data from different sources, it is clear that computational modeling is becoming a fundamental tool in synthetic biology projects.

Chapter 2

Leader Election Project

2.1 Introduction

The Leader Election project aims to engineer a multicellular behavior in a growing microcolony of E. coli, a unicellular prokaryote. The objective is the spontaneous emergence of two different groups of cells from a colony of genetically identical individuals.

Ideally, starting from a single cell in a **undecided** state that grows and divides, at a certain point we want a cell to switch to a different state, named **leader**. Then, a leader cell must be able to turn the rest of the microcolony to the **follower** state (Figure 2.1). The leader and follower states are characterized by the expression of particular genes.

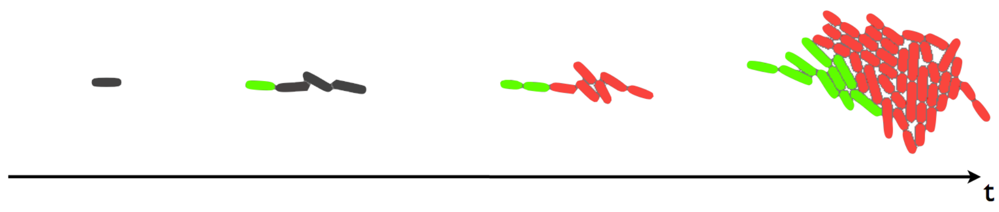


Figure 2.1: Example of the wanted behavior. Leader cells are green, followers are red and undecided are grey.

The construction of this system would provide an essential tool for the realization of cooperative behaviors in bacteria. The ability of a population of prokaryotes to work together toward a common goal by solving different aspects of a single task will make it possible to realize very complex functionalities, without risking to make the metabolic burden unsustainable for the cells.

2.2 Genetic Circuit

The genetic circuit that was conceived to elect a group of leaders in a growing microcolony of E. coli is illustrated in Figure 2.2.

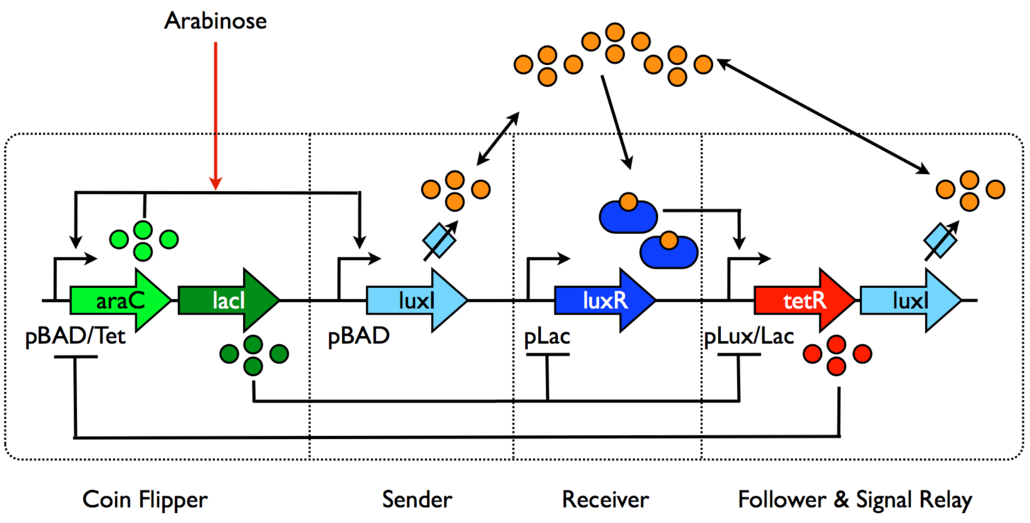


Figure 2.2: Proposed genetic circuit of the Leader Election project [4]

It can be subdivided in 4 different modules: coin flipper, sender, receiver and follower. Its modular structure was devised to allow the realization and testing of each component before the final assembly in the complete circuit. This exploits the decoupling principle, allowing to solve the issues of each module almost independently from the others.

The coin flipper module is composed of an hybrid promoter with operator sites for both the repressor TetR and another transcription factor named AraC. The hybrid promoter regulates the expression of an operon containing the coding sequences for two transcription factors: AraC and LacI.

The sender module is composed of a promoter, regulated by AraC, that controls the expression of LuxI, an enzyme that converts a couple of substrates into the messenger molecule AHL.

The receiver module is composed of a promoter, regulated by the repressor LacI, that drives the expression of LuxR. When LuxR binds AHL it becomes an activator for the hybrid promoter of the last module, the follower one, downstream of which there is a coding sequence for the repressor TetR and another copy of the LuxI gene.

AraC acts as a repressor when the environment doesn't contain arabinose, and becomes an activator in presence of that sugar. Before induction with arabinose, each cell of the microcolony is in the undecided state, in which there is negligible production of all the genes of the system, except LuxR, due to the basal expression of the various promoters. Leakiness is crucial for the pBAD/Tet promoter of the coin flipper module. With time, the stochastic leaky transcription and translation of AraC will give rise to a distribution of AraC concentration in the cells of the colony. Upon induction with arabinose, only the cells in which the concentration of AraC is above a certain threshold will be able to activate the positive feedback that leads to the substantial production of AraC and LacI that defines the leader state. Once the positive feedback is on, LacI deactivates the receiver module and AraC activates the sender module. Leaders start producing LuxI, the enzyme catalyzes the formation of a chemical signal that diffuses in the extracellular environment and causes the nearby cells to activate the follower module. This last module represses the coin flipper with a negative feedback mediated by TetR. The second copy of the LuxI gene in the follower module is needed to relay the signal. With this second copy, cells that switch to the follower state start producing both TetR and signal, so that the information that a leader is already present in the colony, and for this reason the remaining cells should stop flipping coins, is spread quickly.

All the individuals of the colony contain the same construct, but not all the modules are "on" in every cell. The coin flipper and sender modules are active in leaders, the receiver and follower ones are expressed in followers, while in undecided cells the only operating module is the receiver one.

2.3 Genetic components

In the following, a brief review of the natural function of the various molecular components of the circuit is presented.

AraC

In Nature, AraC is part of a complex system that allows the bacteria to exploit, as a source of carbon and energy, the pentose L-arabinose (Figure 2.3).

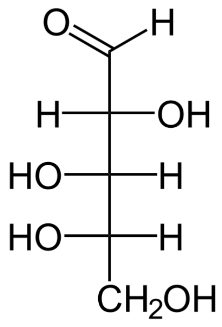


Figure 2.3: L-Arabinose

The wild type ara system is composed of various genes and promoters [11], [12]:

- araE is a gene that is needed for arabinose uptake and is controlled by the pE promoter;
- araF, araG and araH are also needed for arabinose uptake and are in an operon controlled by the pFGH promoter;
- araC encodes for a transcription factor and is under the control of the pC promoter;
- araB, araA and araD are a ribulokinase, an isomerase and an epimerase respectively and are under the control of the pBAD promoter.

AraC is a dimer that can interact with different operator sites: I1 and I2 are placed in the pBAD promoter and another one, O2, is about 200 base pairs upstream the other two. In absence of arabinose AraC binds to I1 and O2 and forms a loop in the DNA. In this conformation it is a repressor for both pBAD and pC. When arabinose is added to the environment, AraC binds to it and assumes a different 3D conformation that allows it to bind the I1 and I2 operator sites and act as an activator for the pBAD promoter.

LacI

LacI is part of another natural system involved in the utilization of a particular sugar, lactose, as carbon source. Even this module is composed of many different parts:

- lacI is a repressor for the pLac promoter, it is constitutively transcribed and prevents the expression of the other three genes;
- lacZ codes for an enzyme, named β-galactosidase, that cleaves the disaccharide lactose into glucose and galactose, prime carbon sources for E.coli;
- lacY is a gene that codes for a transport protein, β-galactoside permease, that anchors to the cell's membrane and facilitates the lactose intake;
- lacA is the third gene of the operon controlled by pLac, it produces another enzyme, β-galactoside transacetylase, whose function is still unclear.

A LacI tetramer can bind the wild type pLac promoter in two points, the O1 operator site is the main one and is placed between the promoter and the beginning of LacZ. The other two sequences that LacI can bind in addition to O1 are O2 and O3, they can be found in positions +400 and -80 with respect to the beginning of translation. When the Lac repressor binds two operators sites at the same time, it causes the DNA to form a loop that makes the promoter virtually unaccessible by the RNA polymerase. The availability of lactose in the environment causes the removal of this block. The few molecules of sugar that cross the cell membrane are degraded by the small number of enzymes produced despite the repression. A side product of the metabolism of lactose, allolactose, binds to LacI, modifying its structure and making it unable to continue its repressive action. Another level of regulation of the lac operon is realized by the cAMP-CRP protein complex, the production of cAMP is catalyzed by the absence of glucose in the environment. This molecule activates the CRP protein, that is able to bind a specific site upstream of the pLac promoter and increases the affinity of the RNA polymerase for this element [13].

When only one operator site is present, like in many synthetic applications, LacI still manages to repress transcription, despite less tightly than when all the operator sites are in the right place. LacI is easily inducible with Isopropyl β-D-1-thiogalactopyranoside (IPTG), an analog of allolactose that cannot be metabolized.

LuxI and LuxR

LuxI and LuxR are part of the luciferase enzyme complex. This system, initially identified in the marine bacteria Vibrio fischeri, controls the quorum sensing regulated luminescence production. These bacteria can establish a symbiotic relation with some marine animals that exploit the light produced by them to hunt at night or hide from predators. The light production is triggered by the increase of the cell concentration over a certain threshold that is not achievable when the bacteria are free in the ocean.

The natural luciferase enzyme complex is composed of many genes [14]

- luxI is the enzyme that produces acyl homoserine lactone (AHL), a signaling molecule (Figure 2.4), it is constitutively transcribed;
- luxR is the signal receptor, it resides in the cytoplasm and is produced continuously in Vibrio fischeri;
- luxC, luxD and luxE code for components of acid reductase that converts the long-chain fatty acid tetradecanoic acid into fatty-aldehyde substrate (tetradecanal) for the light-producing enzyme luciferase;
- luxA, luxB encode the α and β subunits of luciferase enzyme;
- luxG has still a non-identified function.

LuxI produces the AHL signal at a low rate starting from S-adenosylmethionine (SAM) and an acylated acyl carrier protein (ACP) from the fatty acid biosynthesis pathway. Apparently, the fatty acid substrate for LuxI is acquired from the pool of acyl-ACPs generated during fatty acid biosynthesis rather than from products of fatty acid degradation. This would allow for a continuous supply of the fatty acid substrate regardless of the growth conditions [15].

All the genes of this system, except LuxI and LuxR, are part of an operon regulated by the protein LuxR activated by AHL.

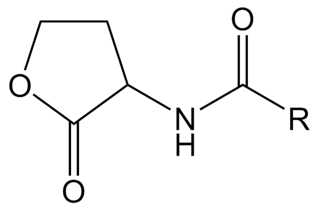


Figure 2.4: General structure of N-Acyl Homoserine Lactone (AHL)

The signal produced by the cells diffuses across their membranes and accumulates in the environment. AHL binds to the LuxR protein. The consequent conformational change seems to cause the complex LuxR-AHL to dimerize. The dimer activates the genes responsible for the production of luminous signal.

Parts of this system have been imported in E. coli and conveniently provide a means of intercellular communication.

TetR

TetR is a dimer that is part of the most abundant resistance mechanism against the antibiotic tetracycline in gram-negative bacteria [16]. In Nature, this protein binds to the operator sites TetO1 and TetO2 and represses its own production and that of TetA, a protein that is responsible for the export of the tetracycline-magnesium complex. The presence of the tetracycline-magnesium complex in the environment inactivates TetR, thus the exporter is produced and the intracellular concentration of antibiotic diminishes.

Since TetR is easily induced with anhydrotetracycline (ATc), a tetracycline analog, it is often employed in synthetic genetic circuits.

Chapter 3

Mathematical Model

The classical modeling strategy in biology and engineering makes use of ordinary differential equations (ODE). Starting from a structural model of the interactions, like the one described in the previous chapter, it is possible to map the reaction network into a system of coupled ODEs. These equations can then be solved numerically in order to track the effects over time of the simultaneously occurring reactions [17].

3.1 Input functions

Most of the information used to construct the models of the various input functions comes from [18].

3.1.1 Promoter regulated by a repressor

As previously stated, a repressor is a protein that, upon interaction with DNA at a promoter site, decreases the probability of transcription of the downstream genes.

Considering a repressor **R** that binds to a promoter **P**, the resulting complex is **R-P**. If multiple repressors are needed in order to achieve repression, it is possible to consider the simultaneous binding of more repressor molecules with the parameter **n**. The RNA polymerase manages to transcribe the coding sequences downstream the promoter only when the repressor is not bound.

The binding of the transcription factor to the promoter can be described by mass-action kinetics:

$$\frac{d[nR-P]}{dt} = k_1 \cdot [R]^n \cdot [P] - k_{-1} \cdot [nR-P] \tag{3.1}$$

Considering the equation at steady state:

$$0 = k_1 \cdot [R]^n \cdot [P] - k_{-1} \cdot [nR - P] \tag{3.2}$$

$$\frac{k_{-1}}{k_1} \cdot [nR - P] = [R]^n \cdot [P] \tag{3.3}$$

$$k_d{}^n \cdot [nR - P] = [R]^n \cdot [P] \tag{3.4}$$

Where the *dissociation constant* $\mathbf{k_d{}^n}$ [M] has been introduced. The lower k_d, the higher the strength of interaction between the transcription factor and the promoter.

It is now possible to write an equation that expresses the conservation of DNA binding sites **P**:

$$[P] + [nR - P] = [P_{tot}] \tag{3.5}$$

Where $[\mathbf{P}_{tot}]$ is the total concentration of DNA binding sites in the cell.

Combining (3.4) with (3.5) it is now easy to express the fraction of free promoter sites as an Hill function:

$$\frac{[P]}{[P_{tot}]} = \frac{k_d{}^n}{k_d{}^n + [R]^n} = \frac{1}{1 + \frac{[R]^n}{k_d{}^n}} \tag{3.6}$$

Figure 3.1 shows the monotonically decreasing input function of a promoter regulated by a repressor for different Hill coefficients n.

When the concentration of repressor equals the dissociation constant, half of the promoters are inactivated. As the Hill coefficient rises, the function tends to approximate a step and the slope of the region where the concentration of repressor is about k_d increases.

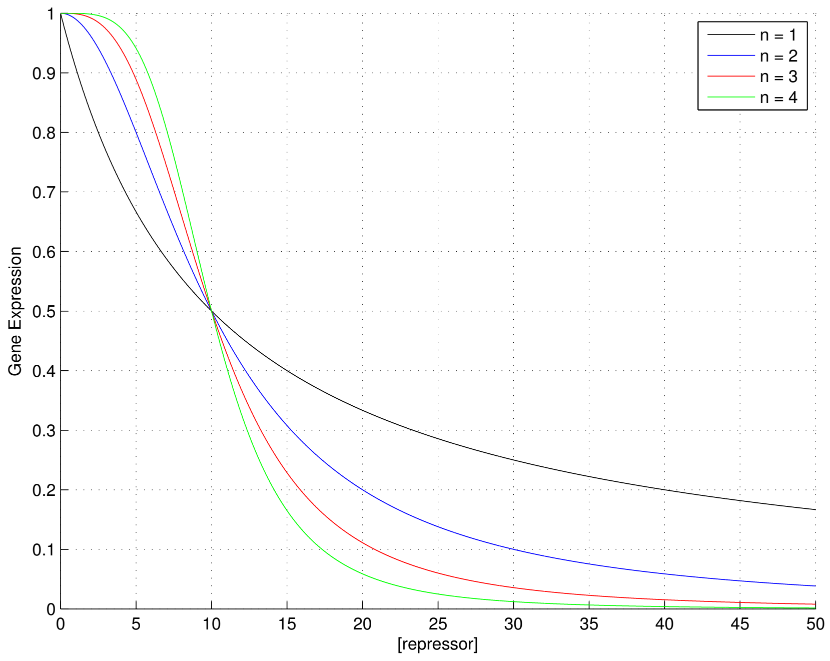


Figure 3.1: Input function of a gene regulated by a repressor with $k_d = 10$

3.1.2 Promoter regulated by an activator

If a transcriptional activator **A** is considered, it is possible to describe the fraction of bound promoter sites as:

$$\frac{[P]}{[P_{tot}]} = 1 - \frac{{k_d}^n}{{k_d}^n + [A]^n} = \frac{{k_d}^n + [A]^n - {k_d}^n}{{k_d}^n + [A]^n} = \frac{[A]^n}{{k_d}^n + [A]^n} \tag{3.7}$$

Figure 3.2 shows the monotonically increasing input function of a promoter regulated by an activator for different Hill coefficients n. Even in this case half of the promoters are inactivated (and half activated) in presence of a concentration of transcription factor that equals k_d.

3.1.3 Binding of an inducer to a transcription factor

Some transcription factors, **TF**, can bind small molecules named inducers, **I**, that alter their affinity for the DNA binding site. It is possible to describe this condition with mass-action kinetics:

$$\frac{d[TF - I]}{dt} = k_1 \cdot [TF] \cdot [I] - k_{-1} \cdot [TF - I] \tag{3.8}$$

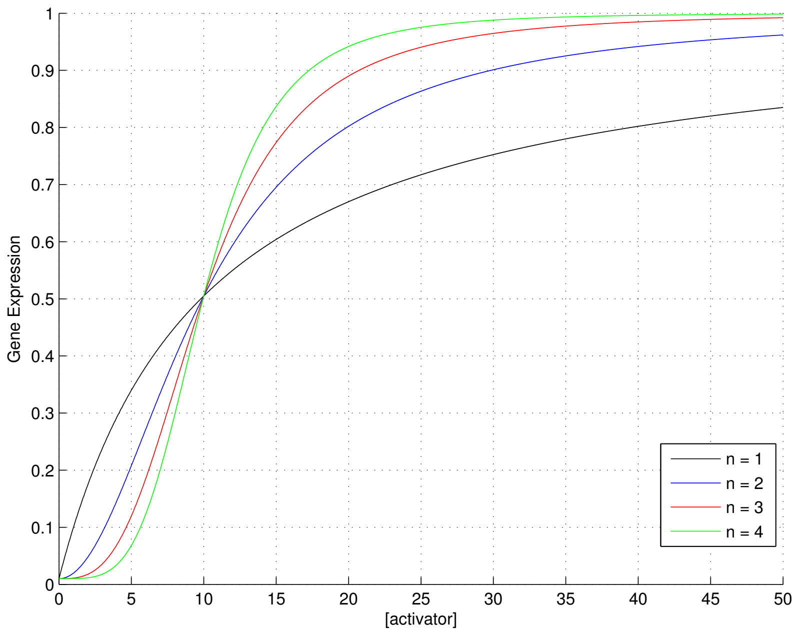


Figure 3.2: Input function of a gene regulated by an activator with $k_d = 10$

With the conservation equations:

$$[TF]_{tot} = [TF] + [TF - I] \tag{3.9}$$
$$[I]_{tot} = [I] + [TF - I] \tag{3.10}$$

At steady state equation (3.8) becomes:

$$0 = k_1 \cdot [TF] \cdot [I] - k_{-1} \cdot [TF - I] \tag{3.11}$$
$$\frac{k_{-1}}{k_1} \cdot [TF - I] = [TF] \cdot [I] \tag{3.12}$$
$$k_s \cdot [TF - I] = [TF] \cdot [I] \tag{3.13}$$

Combining (3.13) with the two conservation equations (3.9) and (3.10) we get:

$$k_s \cdot [TF - I] = ([TF_{tot}] - [TF - I]) \cdot ([I_{tot}] - [TF - I]) \tag{3.14}$$
$$k_s \cdot [TF - I] = [TF_{tot}] \cdot [I_{tot}] - [TF_{tot}] \cdot [TF - I] - [TF - I] \cdot [I_{tot}] + [TF - I]^2 \tag{3.15}$$

Reordering leads to

$$[TF - I]^2 - [TF - I] \cdot (k_s + [TF_{tot}] + [I_{tot}]) + [TF_{tot}] \cdot [I_{tot}] = 0 \tag{3.16}$$

Solving the second order equation and discarding the solution that is not physically grounded, we get:

$$[TF{-}I] = \frac{(k_s + [TF_{tot}] + [I_{tot}]) - \sqrt{(k_s + [TF_{tot}] + [I_{tot}])^2 - 4 \cdot [TF_{tot}] \cdot [I_{tot}]}}{2} \tag{3.17}$$

Equation (3.17) gives the concentration of the complex.

3.1.4 Hybrid promoter regulated by a repressor and an activator

An hybrid promoter is a promoter that can bind different transcription factors. Let's consider the case of a promoter regulated by an activator and a repressor. By multiplying the fraction of operator sites that don't bind the repressor with the fraction of operator sites that bind the activator we get equation (3.19), that constitutes a three-dimensional input function for the promoter (Figure 3.3).

$$\frac{[P_{active}]}{[P_{tot}]} = \left(\frac{[A]^n}{{k_{dA}}^n + [A]^n} \cdot \frac{1}{1 + \frac{[R]^n}{{k_{dR}}^n}}\right) \tag{3.18}$$

$$= \frac{\frac{[A]^n}{{k_{dA}}^n}}{1 + \frac{[A]^n}{{k_{dA}}^n} + \frac{[R]^n}{{k_{dR}}^n} + \frac{[A]^n \cdot [R]^n}{{k_{dA}}^n \cdot {k_{dR}}^n}} \tag{3.19}$$

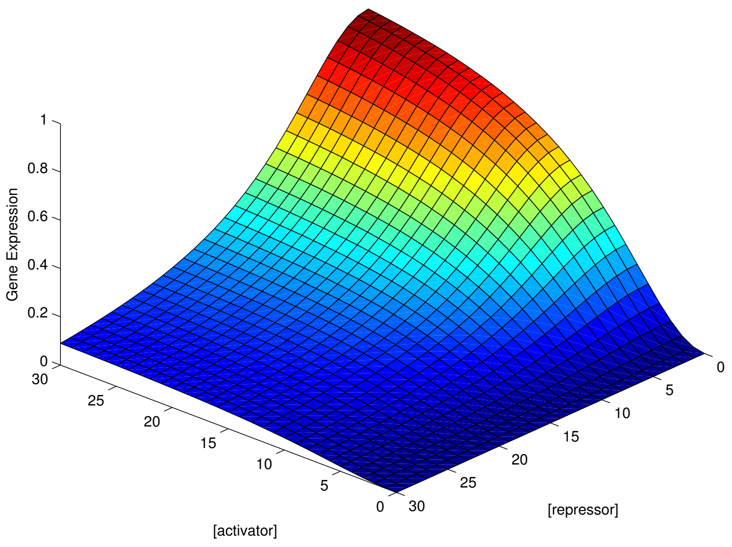


Figure 3.3: Input function of an hybrid promoter with $k_{dA} = k_{dR} = 10$ and n = 2

The hybrid promoter input function shows that gene expression is high only when there is no repressor and a lot of activator. As with the previous two-dimensional hill functions, it can be shown that the behavior becomes more switch-like when n increases.

3.2 Enzyme Kinetics

Let's consider the case of an enzyme **E** that needs two different substrates $\mathbf{S_1}$ and $\mathbf{S_2}$ in order to catalyze the reaction that leads to a product **P**. The enzyme first binds one of the two substrates and forms a complex, $\mathbf{C_1}$ (3.20). This complex is now able to bind a second substrate to form the complex $\mathbf{C_2}$. At this point, the enzyme catalyzes the reaction between the two substrates and a molecule of the product is produced (3.21).

$$E + S_1 \rightleftarrows^{k_1}_{k_{-1}} C_1 \tag{3.20}$$
$$C_1 + S_2 \rightleftarrows^{k_2}_{k_{-2}} C_2 \rightarrow_{k_3} P + E \tag{3.21}$$

The differential equations that describe this process are (3.22) to (3.27)

$$\frac{dE}{dt} = k_3 \cdot C_2 + k_{-1} \cdot C_1 - k_1 \cdot E \cdot S_1 \tag{3.22}$$
$$\frac{dS_1}{dt} = k_{-1} \cdot C_1 - k_1 \cdot E \cdot S_1 \tag{3.23}$$
$$\frac{dS_2}{dt} = k_{-2} \cdot C_2 - k_2 \cdot C_1 \cdot S_2 \tag{3.24}$$
$$\frac{dC_1}{dt} = k_1 \cdot E \cdot S_1 + k_{-2} \cdot C_2 - k_{-1} \cdot C_1 - k_2 \cdot C_1 \cdot S_2 \tag{3.25}$$
$$\frac{dC_2}{dt} = k_2 \cdot C_1 \cdot S_2 - (k_{-2} + k_3) \cdot C_2 \tag{3.26}$$
$$\frac{dP}{dt} = k_3 \cdot C_2 \tag{3.27}$$

With the conservation equations (3.28), (3.29) and (3.30).

$$E = E_0 - C_1 - C_2 \tag{3.28}$$
$$S_1 = S_{10} - C_1 - C_2 - P \tag{3.29}$$
$$S_2 = S_{20} - C_2 - P \tag{3.30}$$

Where E_0, S_{10} and S_{20} are the total concentrations of enzyme, first substrate and second substrate, respectively.

If the enzyme has a low turnover like LuxI [15], it is possible to consider (3.23) and (3.24) at steady state. In this condition it can be shown that (3.27) reduces to (3.31) and the production of P is proportional to the amount of enzyme.

$$\frac{dP}{dt} = K \cdot E_0 \tag{3.31}$$

Where K is a constant.

3.3 Model of the Leader Election Circuit

The model is at a medium level of abstraction, transcription and translation are considered in a single step. The transcription and translation rate at a given time depends on the concentration of active transcription factor that is present inside the cell.

3.3.1 Assumptions

A number of assumptions were made in order to simplify the model and reduce simulation time:

- a saturating concentration of arabinose is present at all times, so that all AraC molecules inside the cell are bound with arabinose and thus act as transcriptional activators;
- the basal expression rate of hybrid promoters and promoters regulated by an activator only is very low;
- both the substrates that LuxI needs to produce AHL are present in saturating concentrations;
- given the high diffusion constant of AHL of about $5.5 \cdot 10^{-6}[\frac{cm^2}{s}]$ [19], the small simulation volume of $160 \cdot 10^3 \mu m^3$, and the fact that AHL can freely diffuse across cell membranes [20], the concentration of AHL is considered uniform;
- interactions between transcription factors and inducers or between transcription factors and DNA are at steady state.

3.3.2 Terms of the model

It is possible to separate the contribution of three different components when the variation of the concentration of a protein in time is considered (3.32).

$$\frac{d[P]}{dt} = BE + PR - DGR \tag{3.32}$$

Where **[P]** indicates protein concentration, **BE** stands for basal expression, **PR** is the production term and finally **DGR** takes into account protein degradation. In the following, the basal expression of the coin flipper module will be termed *leakiness* and the basal expression of all the other promoters *leakiness2*.

In this model, production terms depend on the promoter input function. Since for each operator site used in the circuit at least a dimer of the corresponding transcription factor is needed to achieve activation or repression, the Hill coefficients are set to 2. The various terms for each module of the circuit are detailed in the following.

Coin flipper production term

The coin flipper module is composed of an hybrid promoter that drives the expression of two genes in an operon. The hybrid promoter contains operator sites that can bind two different transcription factors: AraC and TetR. When the sugar arabinose is present in the environment, AraC acts as a transcriptional activator. TetR, instead, always acts as a transcriptional repressor.

The first gene of the operon encodes for AraC, while second one is a coding sequence for the transcription factor LacI.

$$PR = \frac{\beta_{AraC} \cdot (\frac{[AraC]}{Kd_{AraC}})^2}{1 + (\frac{[AraC]}{Kd_{AraC}})^2 + (\frac{[TetR]}{Kd_{TetR}})^2 + (\frac{[AraC] \cdot [TetR]}{Kd_{AraC} \cdot Kd_{TetR}})^2} \tag{3.33}$$

Where β_{AraC} is a factor that defines the maximal transcription and translation rate of the genes that are under the control of this hybrid promoter when it is fully activated by AraC and not repressed by TetR.

Sender production term

The sender module consists of a promoter regulated by the activator AraC that controls the expression of a coding sequence for the enzyme LuxI. LuxI is an enzyme that converts two different substrates in a signaling molecule, AHL.

$$PR = \frac{\beta_{LuxI} \cdot [AraC]^2}{[AraC]^2 + Kd_{AraC}^2} \tag{3.34}$$

Where β_{LuxI} is a factor that defines the maximal transcription and translation rate of the LuxI gene in this module when the promoter is fully activated by AraC.

Receiver production term

The receiver module is composed of a promoter regulated by the repressor LacI. The gene downstream the promoter encodes for the protein LuxR, that acts as a signal receptor.

$$PR = \frac{\beta_{LuxR}}{1 + (\frac{[LacI]}{Kd_{LacI}})^2} \tag{3.35}$$

Where β_{LuxR} is the maximal transcription and translation rate of the LuxR gene when the promoter is not repressed by LacI.

Follower production term

The follower module contains another hybrid promoter that drives the expression of an operon. The two genes in the operon are coding sequences for the repressor TetR and the enzyme LuxI. LuxI is present even in the follower module so that cells that activate this part of the circuit start producing signal and relay the information that somewhere a leader is already present.

$$PR = \frac{\beta_{TetR} \cdot (\frac{[AHL-LuxR]}{Kd_{AHL-LuxR}})^2}{1 + (\frac{[AHL-LuxR]}{Kd_{AHL-LuxR}})^2 + (\frac{[LacI]}{Kd_{LacI}})^2 + (\frac{[AHL-LuxR] \cdot [LacI]}{Kd_{AHL-LuxR} \cdot Kd_{LacI}})^2} \tag{3.36}$$

Where $[AHL - LuxR]$ is calculated with (3.17) and β_{TetR} is a factor that defines the maximal transcription and translation rate of the genes that constitute the operon when the hybrid promoter is fully activated by AHL-LuxR and not repressed by LacI.

Protein degradation

The degradation terms are protein-specific and can be modeled with equation (3.37) for each protein i of the circuit, when appropriate degradation constants are considered.

$$DGRi = Kdgr_{Pi} \cdot [Pi] \tag{3.37}$$

Where Pi is the considered protein.

Signal production and degradation

LuxI is an enzyme that in presence of two different substrates, SAM and hexanoyl-ACP, produces a small signaling molecule named AHL. The two substrates of the enzyme are assumed to be present in saturating concentrations, so that the contribution of each cell to the production term is proportional to the intracellular concentration of the enzyme.

$$\frac{d[AHL]}{dt} = \sum_{i=1}^{N_c} (v_{max} \cdot [LuxI]_i) - Kdgr_{AHL} \cdot [AHL] \tag{3.38}$$

Where $[LuxI]_i$ is the concentration of enzyme in the simulation volume due to cell i, N_c is the number of cells in the microcolony and [AHL] is the concentration of signaling molecule in the volume of simulation.

3.3.3 Parameters

The circuit is assumed to be assembled on a low copy number plasmid, and the parameters that describe this condition are reported in Table 3.1.

Converting the concentration of the various molecules in their copy numbers is straightforward: considering that the volume of an E. coli cell is about $2\mu m^3$ it follows that if the cell contains only 1 molecule of a specific substance, then its concentration is $\frac{1molecule}{2\mu m^3} = \frac{1molecule}{2fl} \approx 1nM$.

Parameter	**Value**	**Unit**
growth rate	0.02	$[min^{-1}]$
v_{max}	1.1	$[min^{-1}]$
leakiness2	0.01	$[min^{-1}]$
leakiness	0.01	$[min^{-1}]$
β_{LuxR}	40	$[min^{-1}]$
β_{AraC}	5	$[min^{-1}]$
β_{LuxI}	40	$[min^{-1}]$
β_{TetR}	40	$[min^{-1}]$
$Kdgr_{LuxR}$	0.02	$[min^{-1}]$
$Kdgr_{AraC}$	0.01	$[min^{-1}]$
$Kdgr_{LacI}$	0.02	$[min^{-1}]$
$Kdgr_{LuxI}$	0.01	$[min^{-1}]$
$Kdgr_{TetR}$	0.02	$[min^{-1}]$
$Kdgr_{AHL}$	0.0018	$[min^{-1}]$
$Ks_{AHL-LuxR}$	100	$[molecules \cdot fl^{-1}]$
$Kd_{AHL-LuxR}$	10	$[molecules \cdot fl^{-1}]$
Kd_{AraC}	5	$[molecules \cdot fl^{-1}]$
Kd_{LacI}	6	$[molecules \cdot fl^{-1}]$
Kd_{TetR}	5	$[molecules \cdot fl^{-1}]$

Table 3.1: Table of parameters

The growth rate of 0.02 corresponds to a doubling time of 50 minutes, this value is not much higher than 20 minutes, that is widely considered the

bottom limit [21]. It is possible to vary the doubling time by changing the growth medium or the temperature of the local environment.
Since the substrate of LuxI is assumed to be present in saturating concentrations, the enzyme produces AHL at the rate of about 1.1 AHL molecule per LuxI molecule each minute [15].
The values assigned to the betas are in a realistic range according to [22], where the maximal production rate of LacI and GFP was set to about $20\frac{proteins}{gene \cdot min}$.
β_{AraC} has a lower value than the others because preliminary simulations made it clear that, with the actual dissociation constants, the production of 40 proteins per minute of AraC and LacI would have biased the system toward the leader state. This low rate of protein synthesis can be achieved by tuning gene expression, for example by varying the RBS spacer sequence [8].
Even the degradation constants of AHL, LacI, LuxR-AHL and AraC were obtained from literature [23], [24], [25], while the TetR one was assumed to be in the same order of magnitude as that of the other proteins.
$Ks_{AHL-LuxR}$ is in the same order of magnitude as in [26], moreover the activation threshold is consistent with [20]. The value for Kd_{LacI} was obtained considering that a single operator site in the promoter can reduce the protein production about 20 fold [27], in presence of about 40 monomers of LacI [28]. In these conditions, using equation (3.6) it is possible to estimate a Kd_{LacI} between 10^{-8} and 10^{-9}. The same reasoning led to the choice of Kd_{AraC}, because 40 monomers are enough to activate gene expression in rapidly growing cells [12]. The *in vivo* affinity between TetR and its operator site tetO is assumed to be in the same order of magnitude of the one between LacI and O_1. That explains the similar value of Kd_{TetR} and Kd_{LacI}.
From *in vitro* experiments, it was shown that the binding constant between LuxR and its operator site is about twice the one between TetR and tetO [29], [30].

Chapter 4

Simulations and Results

4.1 gro environment

gro is a specification and simulation language developed by the Klavins Laboratory at the University of Washington. This section describes the main features of the simulation environment and its peculiar characteristics [31].

With gro, it is possible to simulate a mathematical model of a synthetic genetic circuit at different levels of abstraction in each cell, and observe the emergent behavior of the growing microcolony. gro combines a distributed systems and parallel computing approach. With gro it is possible to simulate the growth of a microcolony in a monolayer, visualize it as would be viewed with a fluorescence microscope (Figure 4.1) and export data like the copy number of each molecule in each cell.

gro models growth, division, contact forces between cells and small molecule diffusion. Cells are assumed to be approximately cylindrical, with radius $r = 0.5\mu m$ and initial length of $l = 2\mu m$. The time resolution of each subprocess of the simulation is controlled by the time step parameter dt that can be adjusted to reduce numerical errors.

The initial volume of the cell is $V = \pi r^2 \cdot l \approx 1.57 fL$ and the growth of each cell is modeled according to the differential equation $\frac{dV}{dt} = k \cdot V$ where the growth rate k can be varied. Each cell grows until it has approximately doubled in size, at which point it divides approximately in two. The mean and variance of the division size are also parameters that can be set in gro. Although the volume of the microcolony grows smoothly, the number of cells increases according to a discrete stochastic process.

The cells are constrained in a single layer so the contact forces can be modeled using a simple two-dimensional physics engine, the one that is implemented in gro was originally developed to simulate physics in computer

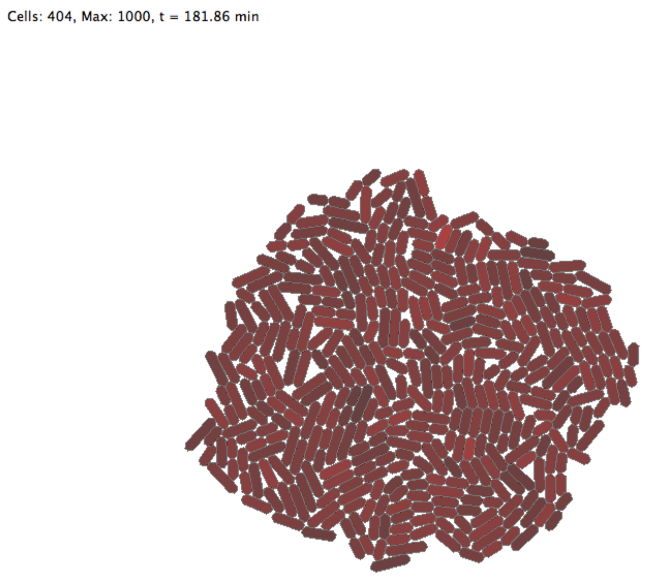


Figure 4.1: Simulation of stochastic production and degradation of red fluorescent protein (RFP). The cells are genetically identical but the amount of protein differs slightly between individuals, note the different red intensity

games. The effect is intended to be only qualitatively similar to the actual process.

Cell to cell communication via small molecules is simulated using the finite difference method with a 2D grid of square elements, the resolution of which can be specified when a model is implemented. The dynamics are simulated using Euler integration. When a new signaling molecule is declared, its diffusion and degradation rates can be specified. Cells emit, sense and absorb small molecules.

Each cell in the simulation runs a program written in the gro programming language. It is possible to specify the behavior at the most appropriate level of abstraction for the current design phase. For example, it is equally possible to specify the production of a particular molecule at a particular rate or model in detail the processes of transcription and translation.

gro is a strongly typed, interpreted programming language. To model parallelism, gro programs consist of sets of unordered guarded commands of the form g:c where g, the guard, is a boolean expression and c, the command, is a list of statements that can be either assignments or function calls. In each step of the simulation, each guard is evaluated: if it evaluates to true, then the associated command is evaluated. Each guarded command specifies a distinct process in the cell and all such processes occur effectively simultaneously. Following a standard approach in modeling parallelism, guarded commands are executed in an unspecified order despite being listed in a particular order in the code.

To model stochastic events in the cell, gro provides a special function, rate(), that takes one argument and returns true or false randomly. In particular, rate(r) returns true upon a given evaluation with probability $r \cdot dt$ and false the rest of the time, where dt is the simulation time step. The rate function allows gro to approximate the Master Equation with Euler integration.

4.2 Sensitivity Analysis

To take into account the stochasticity of gene expression, the differential equations that constitute the mathematical model of the Leader Election circuit, described in the previous chapter, were approximated by using the *rate()* function of the gro language.

Starting from a condition in which the parameters of the model have their nominal value, reported in Table 3.1, a sensitivity analysis by varying one factor at a time around that central point in the parameters' space was performed.

Nine parameters were varied in a range that spans two orders of magnitude: the four **betas**, the four **Kds** and the **leakiness** of the coin flipper module.

The parameters to be varied were chosen on the base of their relation to the genetic components of the circuit and the possibility to physically tune them:

- The betas define the maximal protein production per minute, they can be tuned by modifying the RBS sequence, by choosing an appropriate RBS spacer or by modifying the promoter region;
- The Kds are related to the binding affinity of the transcription factor to the promoter, thus it should be possible to tune them by varying the sequence of the operator site in the promoter region;
- The leakiness of the coin flipper is the basal transcription and translation rate of the hybrid promoter when no transcription factor is acting on it, it can be tuned by modifying the sequence of the promoter region;

The size of the range of variation was chosen to be of about two orders of magnitude because it is deemed to be an achievable range in the physical tuning of the different genetic parts.

A simulation starts with a single cell that grows and divides and stops when there are more than 50 cells in the simulation volume. The threshold of 50 cells was chosen because, by the time that the colony reaches that population size with the nominal set of parameters, almost all the cells manage to end up in either the leader or the follower state.

A cell is classified as a leader if its intracellular concentration of AraC is above 20 $[\frac{molecules}{fl}]$ and its intracellular concentration of TetR is below 20 $[\frac{molecules}{fl}]$; it is considered a follower if its intracellular concentration of AraC is below 20 $[\frac{molecules}{fl}]$ and the one of TetR is above 20 $[\frac{molecules}{fl}]$; otherwise the cell is undecided. Given the previous criterion, a cell can be undecided for two different reasons: both AraC and TetR above the threshold or both below it.

In each simulation there are various sources of stochasticity: protein production, the size at which a cell divides and even the order of the various reactions. Since the outcome critically depends on an initial stochastic event that makes leader cells emerge, one hundred repetitions of the simulations were performed in order to obtain a meaningful statistic and draw conclusions. Each simulation is independent from the others, so it is possible to speed up the sampling of the parameter's space by running multiple instances at the same time. To this end, a Python script to send simulations to the nodes of a Linux cluster via ssh was developed. The script is described in the appendix.

4.2.1 Cost Function

To grade the outcome of each simulation, a cost function based on the difference between the fraction of cells in each of the three states and the wanted outcome was defined (4.1).

$$Cost = \left\| \begin{bmatrix} \%Leaders - \%Leaders_{opt} \\ \%Followers - \%Followers_{opt} \\ \%Undecided - \%Undecided_{opt} \end{bmatrix} \right\|_2 \tag{4.1}$$

Where $||\cdot||_2$ is the 2 norm and the square brackets indicate a column vector. The cost function reaches the value zero when the outcome of the simulation is exactly the wanted one all the time, it gets to $\sqrt{2} \approx 1.4$ when the outcome is the furthest from the goal.

The sensitivity analysis consists of a series of plots, one for each parameter that is varied, where on the x axis is present the value of the parameter while on the y axis is represented the corresponding value of the cost function. Since each simulation is stochastic, dots represent the mean of the cost function over all repetitions and bars represent the standard deviation (Figure 4.2).

With the sensitivity plot, it is possible to calculate the average slope that separates the central value of the considered parameter and the value where the cost function is minimized. A linear fitting in semilogaritmic space was computed using the least squares method and the result is the red line in Figure 4.3.

The higher the slope of the red line, the quicker it will be possible to reach the optimal value of the parameter. An high slope also means that even a small variation in the physical characteristics of the genetic component that correspond to the considered parameter, will improve the outcome by a great extent.

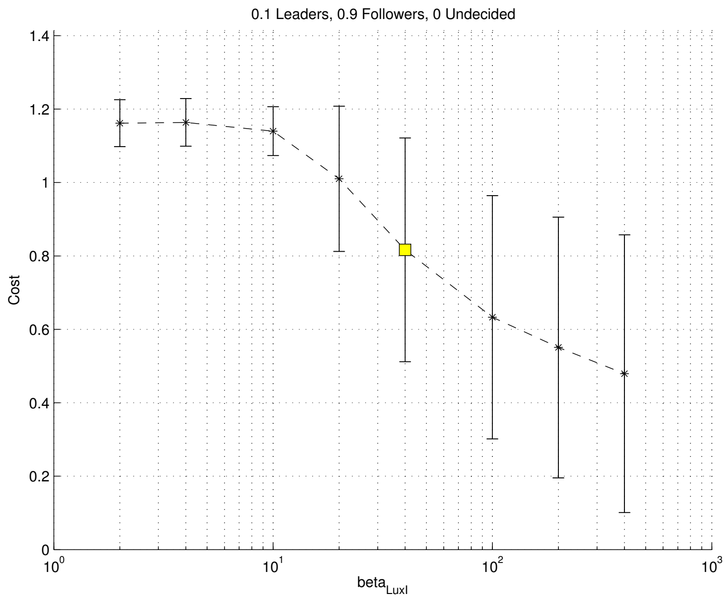


Figure 4.2: Example of sensitivity plot. The parameter β_{LuxI} is varied over about two orders of magnitude around its nominal value, depicted as a yellow square.

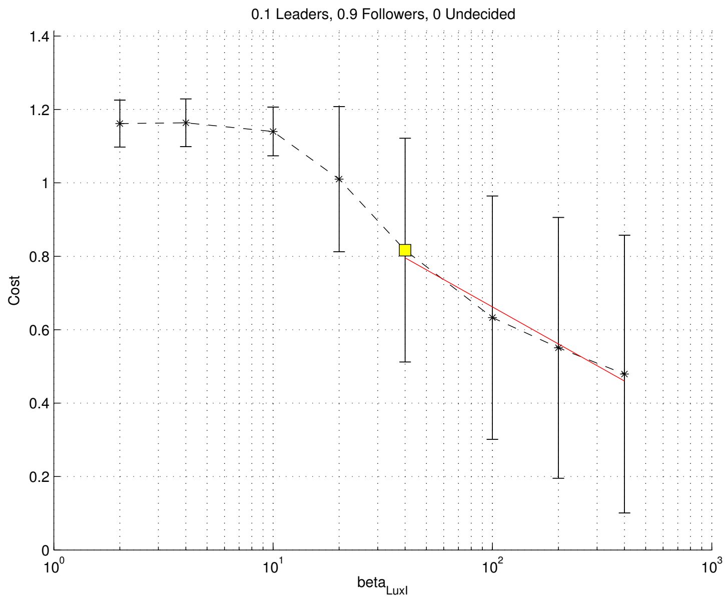


Figure 4.3: Fitting of the cost function

4.2.2 Ranking

To rank the parameters on the base of both the previously mentioned slope, and the minimum of the cost function that it is possible to obtain by varying each parameter in the considered range, a special version of scatter plot was ideated. The structure of the plot is illustrated in Figure 4.4.

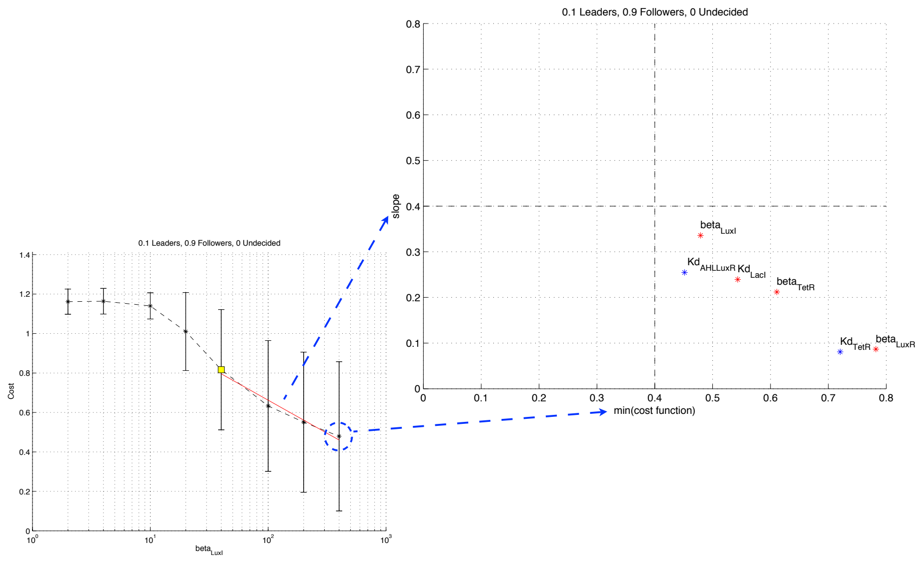


Figure 4.4: Structure of the scatter plot used to rank the parameters. On the left, the information that has been used to place the β_{LuxI} dot in the right-hand graph.

On the x axis is indicated the minimum of the cost function that is possible to reach by varying the considered parameter, while on the y axis is indicated the absolute value of the average slope between the central value and the one that minimizes the cost function. The various parameters are indicated as color-coded dots in the graph. If the dot is red, then the slope of the fitting is negative and the analysis suggest to increase the value of the parameter in order to approach the goal, see the case of β_{LuxI} in Figure 4.4 as an example. Otherwise, if the dot is blue, that means that the average slope is positive, thus the parameter needs to be decreased in order to get close to the desired behavior.

The scatter plot can be divided in the four different zones of Figure 4.5.

In the top-left corner (red), the slope is high while it is also possible to

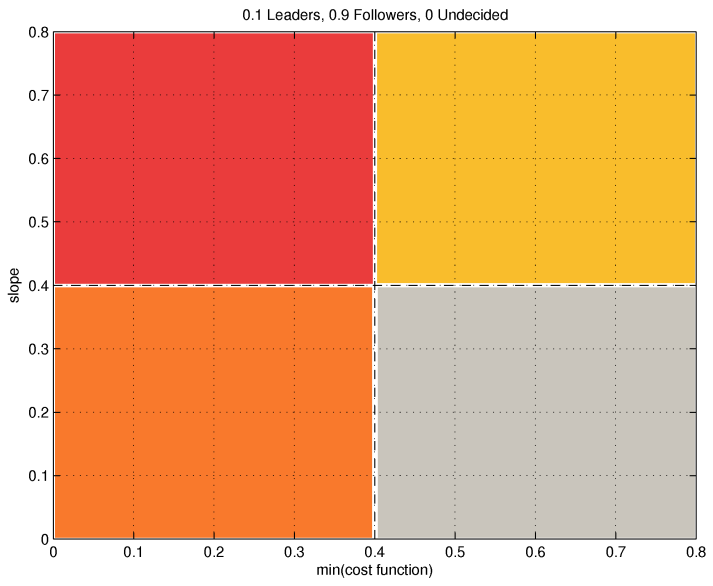


Figure 4.5: The four zones of the scatter plot used to rank the parameters

reach a low value of the cost function, there the most important parameters reside, because, by varying one of them, it is possible to improve quickly the outcome and get very close to the goal. The bottom-left corner (orange) is an important zone too, because by tuning one of the parameters in there it is possible to obtain the wanted behavior. The low slope tells that the optimal outcome is not very far from that of the circuit with the nominal components.

The right-hand half of the graph encompasses a region in which the cost function is high, varying a parameter that is in that zone will push the system toward the goal but it will not be possible to reach it. The top-right corner (yellow) is just slightly more important than the bottom-right one (grey) because, on top, the slope of the cost function is high and at least the cost can be lowered quickly.

Parameters that minimize the cost function when their value is the central one are not shown in the scatter plots because modifying them would make the system perform worse than before.

It is important to note that, since the sensitivity analysis was performed by varying one factor at a time, what happens when more than one parameter is varied at the same time cannot be predicted.

4.3 Results

By setting different combinations of optimal values in the definition of the cost function, it is possible to explore a number of behaviors that the circuit can implement just by modifying the quantitative characteristics of its components (e.g. level of affinity between transcription factors and operator sites) and not their qualitative nature, that defines the structure of the circuit.

In fact, modifying the optimum makes the shape of the sensitivity plots change. This reflects itself in a different ranking of the various parameters in function of the behavior that is set as the optimal one.

In the following, three different optimal outcomes are considered.

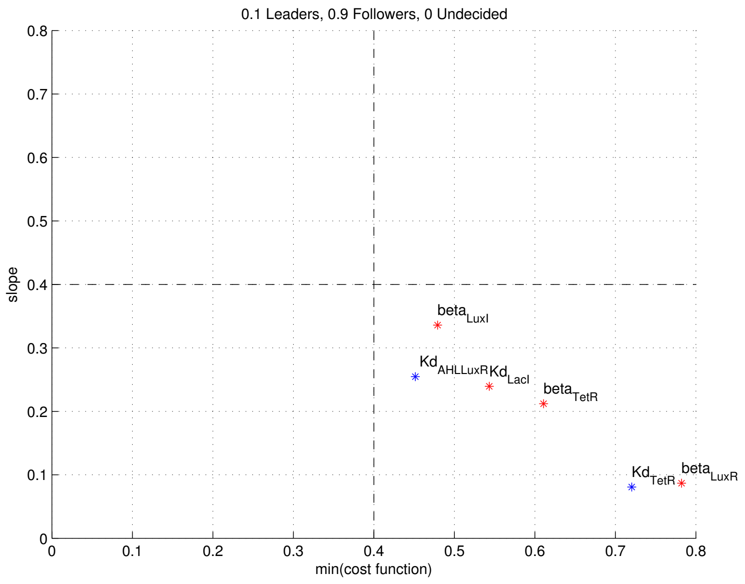
Goal 1: Majority of followers and minority of leaders

The top scatter plot in Figure 4.6 considers as a target a colony with 90% of cells in the follower state, 10% of cells in the leader state and none undecided. This is the goal the circuit was ideated for.

As we can see, all the parameters fall in the grey region, so it is not possible to reduce significantly the cost. Anyway, as it is evident from the bottom plot in Figure 4.6, it seems to be possible to obtain a microcolony in which the majority of cells is in the follower state.

To reach this condition, the parameters that the analysis suggest to tune are either $Kd_{AHLLuxR}$, that is related to the affinity of the LuxR-AHL complex to the lux operator site in the promoter of the follower module, or β_{LuxI} that is the maximal transcription and translation rate of the promoter that drives the production of enzyme in the sender module of the circuit. $Kd_{AHLLuxR}$ would need to be decreased, that means that, in order to get more followers than leaders, the promoter of the follower module needs to be very sensitive to the activator. The less activator is needed to promote transcription, the better. Since the relation between operator sequences and affinity for transcription factors is still not well understood, it might be better to focus on increasing the transcription and translation rate of the enzyme and thus increasing β_{LuxI}.

Figure 4.7 shows a series of plots: in the left-hand column is displayed the cost function of the top-4 parameters, ranked from top to bottom in increasing order on the base of the minimum that is possible to reach by varying them. The right-hand column contains the fraction of leaders, followers and undecided cells at the end of the simulation in function of the parameter's value, for the same parameters of the left-hand column. In the



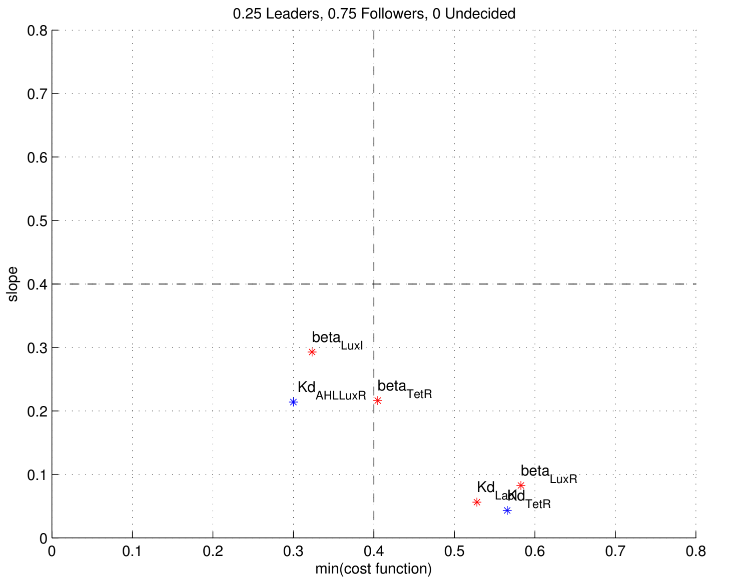


Figure 4.6: Scatter plots when the goal is set to a majority of followers.

right-hand plots, dots represent the average fraction of cells in each state, over one hundred repetitions of the simulations, and bars represent two standard deviations in total.

It is easy to see that, to obtain the best outcome, the parameter that should be varied is one of the two described before. By varying either β_{LuxI} or $Kd_{AHLLuxR}$ it is possible to reach most of the time a majority of followers in the microcolony, while tuning the third or fourth best candidate leads to a mix of followers and undecided cells in one case and an equal fraction of leaders and followers in the other.

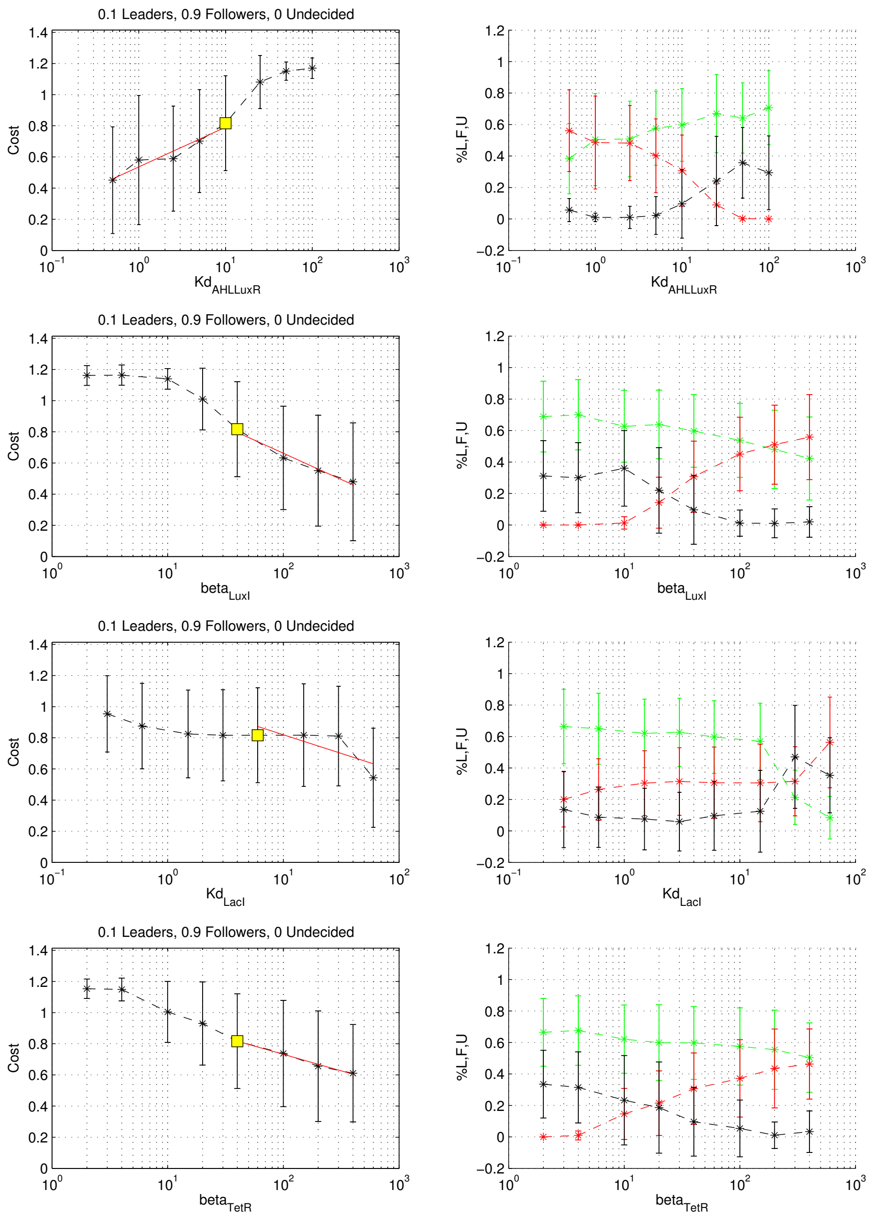


Figure 4.7: The goal is set to a majority of followers. Left-hand column: cost function of the four most promising parameters. Right-hand column: the corresponding fraction of leaders, followers and undecided cells are depicted in green, red and black.

Goal 2: Balanced population of leaders and followers

Considering the case in which the aim is to obtain a microcolony that is split in half between leaders and followers, all the parameters fall in the orange region of the scatter plot (Figure 4.8). This means that reaching this goal is relatively easy, because the variation of almost every parameter in the right direction can be effective.

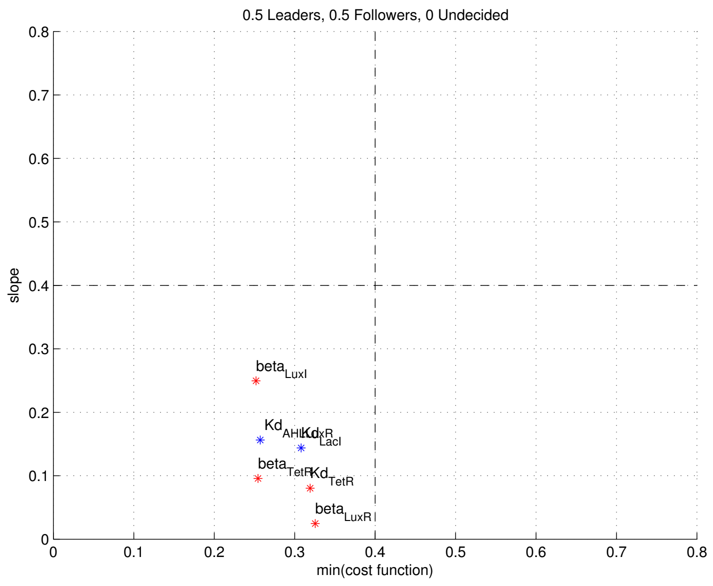


Figure 4.8: Scatter plot when the goal is to obtain a population in which there are about as many leaders as followers.

As before, the four parameters that can can push the system closer to the optimum than the others are ranked in Figure 4.9, where the plot has the same structure as the previous one. Note that in the first and fourth row, the optimum that the cost function suggests is not the one that is easily guessed by looking at the corresponding plot on the right. This happens because the mean value of the cost function is calculated by averaging each cost of the one hundred simulations. Since the cost is calculated via a 2-norm, outliers weigh a lot.

Even if either one the first three parameters analyzed in Figure 4.9 appears to be a good choice, when the suggested direction is taken into account the easiest to tune in the laboratory are the betas. Between them, the best one seems to be β_{LuxI}, because it needs a smaller increase than β_{TetR} and also

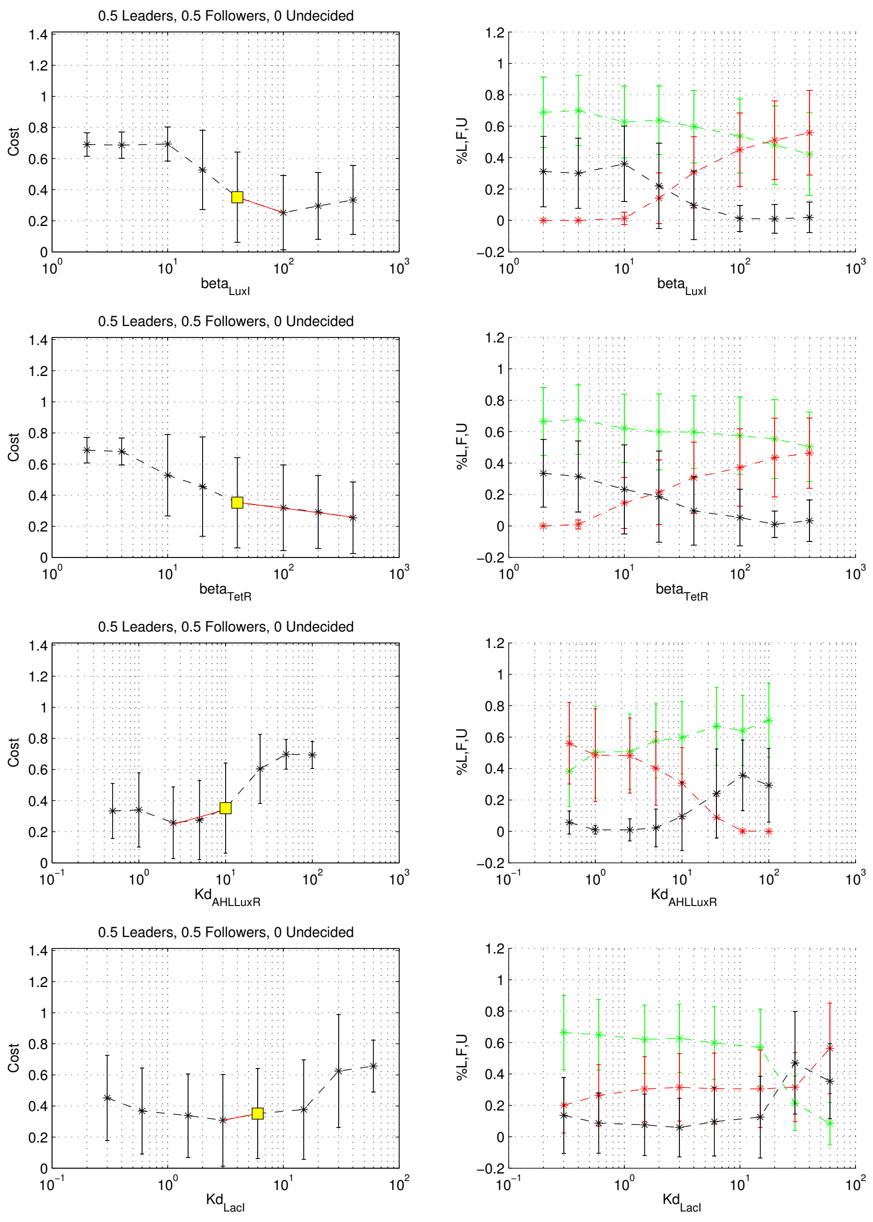


Figure 4.9: The goal is set to a colony split in half between leaders and followers. Left-hand column: cost function of the four most promising parameters. Right-hand column: the corresponding fraction of leaders, followers and undecided cells are depicted in green, red and black.

the standard deviation of undecided cells looks smaller when the maximal production rate of the enzyme is high.

Goal 3: Majority of leaders and minority of followers

The third and last objective analyzed is sort of the complementary of the first one: a microcolony in which the number of leaders dominate that of the followers. In the top scatter plot of Figure 4.10, the target is set to three quarters of leaders and one quarter of followers. All the parameters reside in the left-hand half of the graph, this indicates that it is quite easy to obtain this behavior. Considering the bottom scatter plot of Figure 4.10 it seems to be relatively easy even to get a microcolony composed of almost all leaders and a few followers. In both plots the four most important parameters are the $leakiness$, Kd_{AraC}, β_{AraC} and Kd_{TetR}.

As in the previous cases, Figure 4.11 ranks them in increasing order on the base of the minimum that is possible to reach by varying them one at a time. In this case the goal is set to 90% leaders, 10% followers and none undecided.

Considering that increasing the affinity of AraC to its operator site might prove to be difficult in the laboratory, the two parameters that can easily bias the system towards a majority of leaders are the leakiness of the coin flipper module and β_{AraC}, the maximal transcription and translation rate of the genes of the same module. In fact, while even increasing Kd_{TetR} makes the number of leaders dominate the one of the followers, the high standard deviations of all the three different states indicates that the outcome would be quite unpredictable.

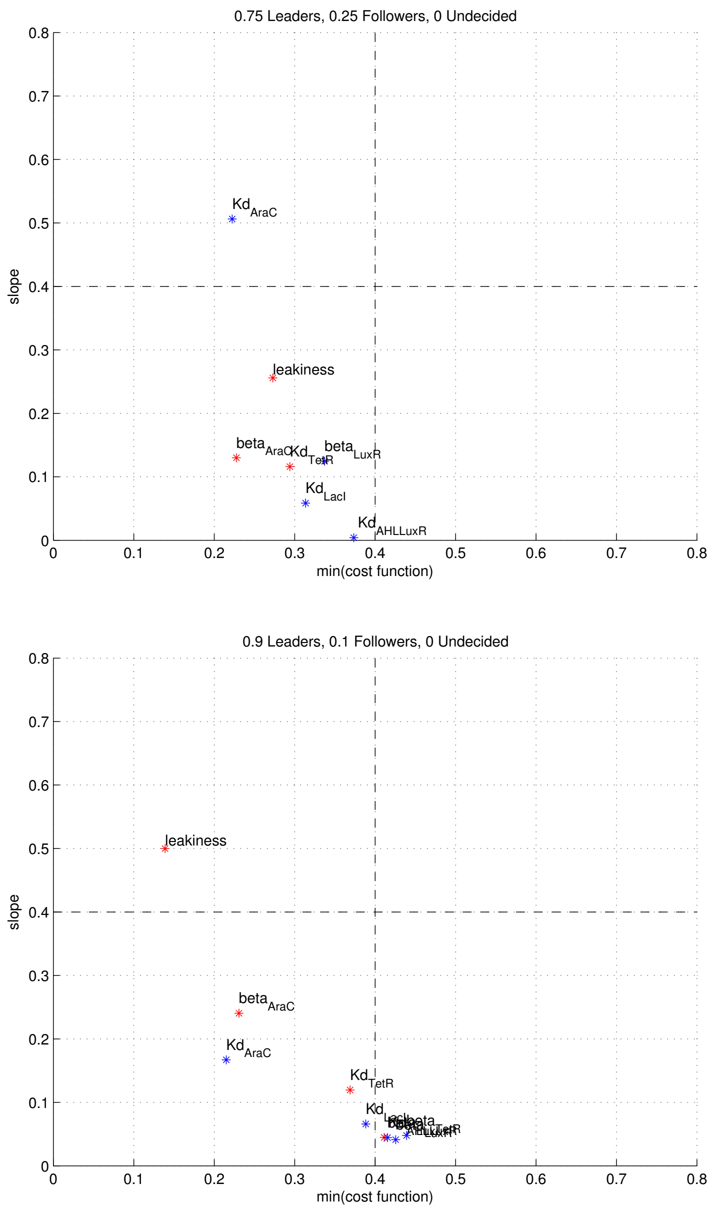


Figure 4.10: Scatter plot when the goal is set to a majority of leaders.

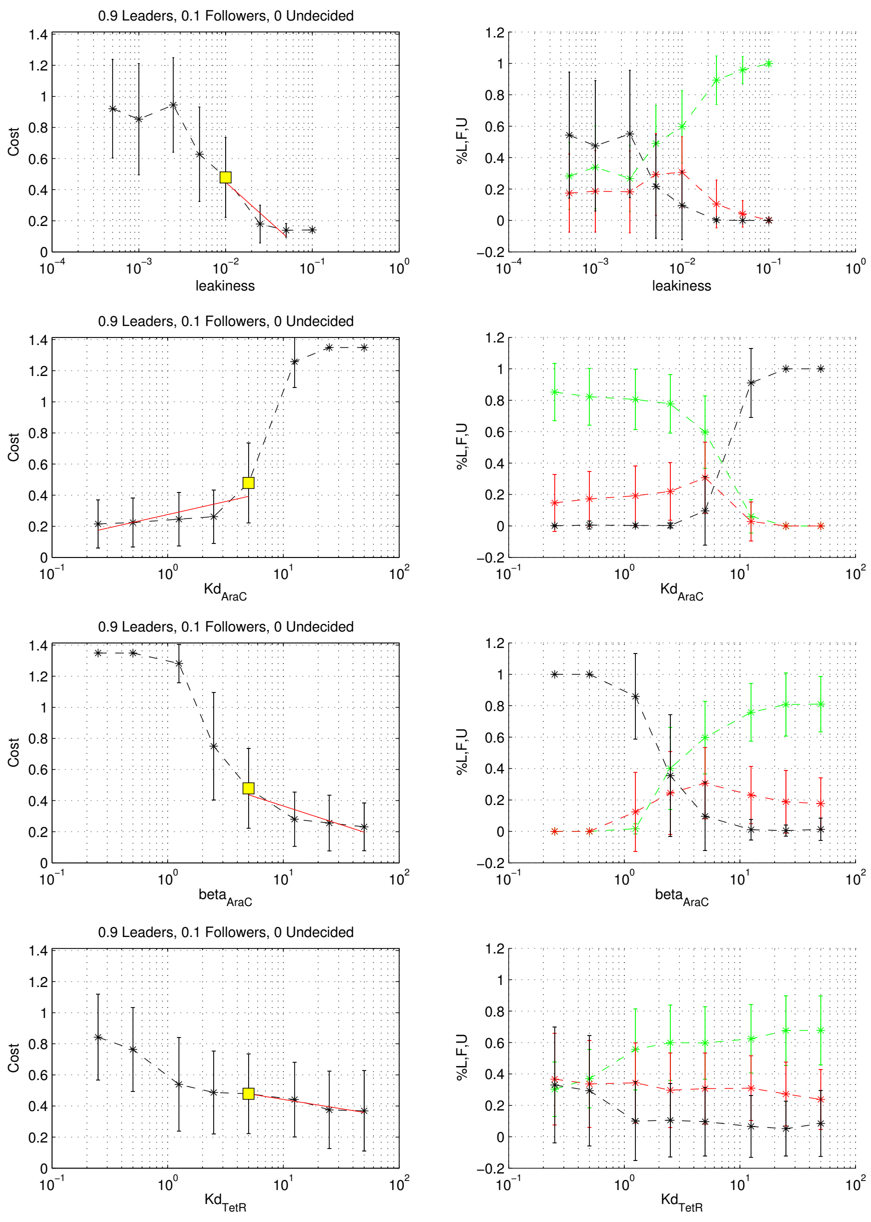


Figure 4.11: The goal is set to a majority of leaders. Left-hand column: cost function of the four most promising parameters. Right-hand column: the corresponding fraction of leaders, followers and undecided cells are depicted in green, red and black.

4.4 Conclusions

The Leader Election circuit is a complex structure of non-linearly interacting genetic components. Modeling and simulation provide a quick and inexpensive mean to analyze the relation between parts and determine the required characteristics for a correct behavior of the system.

The analysis of the different goals shows that obtaining a majority of follower cells in a microcolony is the most difficult task. Since the analysis suggests to increase the production of enzyme or increase the sensitivity of the promoter in the follower module, the limiting factor seems to be the amount of signal in the environment. A small number of leader cells in a microcolony struggle to produce quickly the volume of signal that is needed to turn the rest of the colony in the follower state. As time passes, the probability that new leader cells will emerge increases. When the production of enzyme is quick enough, and thus more signal is produced early in the simulation, the outcome is most of the time a microcolony that contains a majority of cells in the follower state.

Achieving a microcolony in which the cells are equally split between the leader and follower state is comparatively easy. The best strategy to reach this goal is to slightly increase the transcription and translation rate of the enzyme in the sender module. Alternative approaches consist in increasing either the affinity of the promoter in the folower module for the complex AHL-LuxR or increasing the transcription and translation rate of the genes in the same module.

Finally, it is possible to attain reliably a majority of leader cells in a number of different ways. The first approach, increasing the basal expression rate of the coin flipper module, can push the system to the extreme condition in which all the colony switches to the leader state. A smaller increment, instead, leaves a few followers at the end of the simulation. The second strategy, instead, involves either the increase of the expression of the genes in the coin flipper module or the increase in affinity between the activator AraC-Arabinose and its corresponding operator sequence. Both the modifications make the number of leaders dominate that of followers and also minimize the number of undecided cells.

4.5 Future application: pattern formation

Pattern formation is a common phenomenon in the natural world and its best exemplification is morphogenesis. The ability to produce artificially spatial patterns in a microcolony of growing cells might have implications in the fields of tissue engineering and biomaterials production. In future, the technology might allow the growth of complex structures in response to external stimuli or the biological fabrication of valuable products only in defined regions of space.

The seminal paper about modeling the emergence of a spatial pattern in a system of reacting and diffusing substances was published in the 1950s by Alan Turing [32]. In order to develop a definite, stationary pattern, the two chemicals need to be able to diffuse and interact non-linearly. [33].

In the early 1990s, it was discovered that particular strains of bacteria, collectively called Ben-Jacob's bacteria, are able to grow in a macroscopic colony that displays a definite pattern of cell concentration like the one in Figure 4.12 [34].

In the recent past, synthetic genetic circuits that form macroscopical patterns in Petri dishes have been devised. In [24] the authors were able to realize a concentration bandpass filter by placing a disk of sender cells in the center of a Petri dish on which two different strains of receiver cells were uniformly spread. The sender cells were engineered to emit a signal that, diffusing into the extracellular environment, developed a concentration gradient. After a number of hours of incubation, it was possible to visualize the bull's eye pattern formed by the receiver cells. The two strains were sensitive to different concentrations of the signal and so, if the concentration in that region of the dish was in their detecting range, a fluorescent protein was produced. More complex patterns were obtained by accurately positioning disks of sender cells in the experimental environment.

This system is a very important step toward the engineering of a synthetic pattern formation system in vivo, but it is just a proof of principle: different groups of cells receive and send the signal and the sender cells are artificially placed in the most convenient position. Furthermore, some natural organisms form pattern at a smaller spatial scale than the one considered in this application.

The Leader Election circuit is a perfect example of symmetry breaking, since, starting from an uniform population of bacteria that developed from a single cell, it aims to obtain two distinct classes of cells that display phenotypically different characteristics. This behavior can be exploited for the

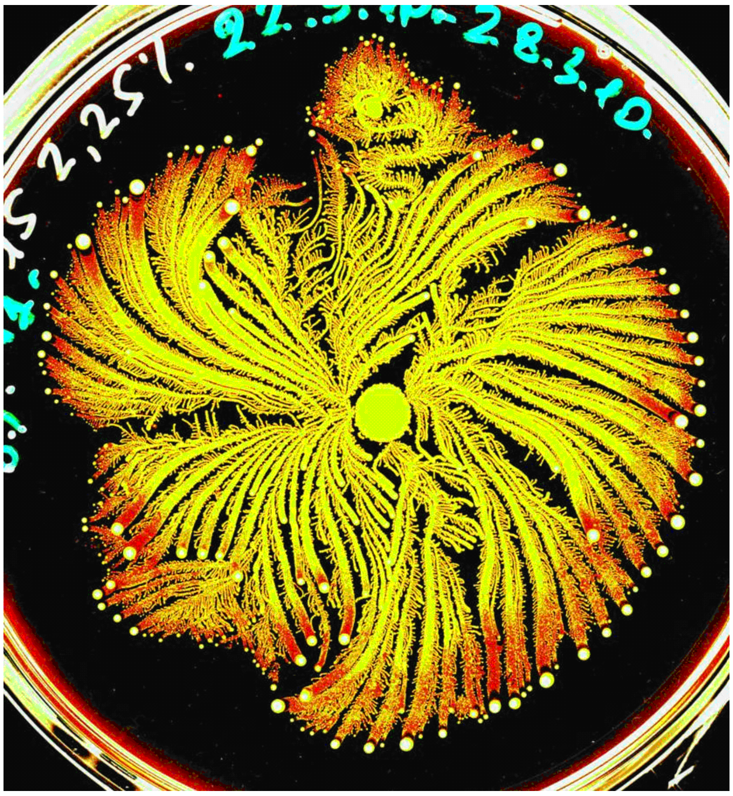


Figure 4.12: Example of pattern created by a Ben-Jacob's bacterial strain. The image of P. vortex colony was created at Prof. Ben-Jacob's lab, at Tel-Aviv University, Israel

generation of patterns because it allows to make a structure emerge in a predictable way, without manually positioning the different components. For this reason, the realization of spatial patterns was analyzed, using the finite state machines formalism, as a future application of the Leader Election system.

A finite state machine is a computational model that describes a particular system or problem. Each machine has a defined number of states connected through arcs, each state represent a configuration of the system, while the arcs define the allowed transitions between the states. A transition is triggered by an external event or a particular condition. This representation is particularly convenient to reason on a system without worrying about the details of the physical realization.

With gro, it is easy to implement a finite state machine in cells. In contrast to the deterministic ODE models proposed in the past, each cell is genetically identical, the environment is uniform and the symmetry is broken due to the presence of stochasticity in the process of switching states, that in a more detailed model would be due to stochastic gene expression.

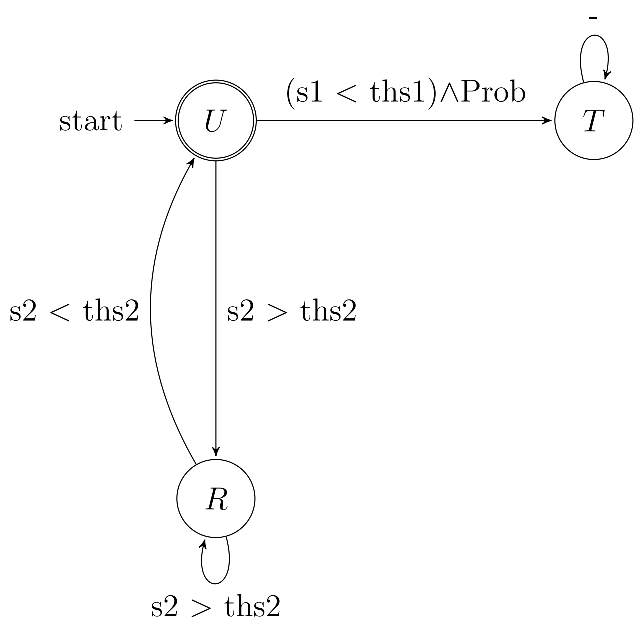
4.5.1 Spots

In the following, a finite state machine that gives rise to a spotty pattern is illustrated. It can be described by three states, two different signals and two thresholds. The thresholds define the conditions that must be satisfied in order to move from one state to the other.

A single cell starts in the undecided state **U**, it grows and divides. If the signal **s1** is below a certain threshold, a cell in the undecided state **U** can switch, with a certain probability, to the transmitter state **T** in which it remains trapped.

Once in the transmitter state, the cell decreases its growth rate, produces a green fluorescent protein (GFP) and emits two different signals: **s1** is a long range signal (cyan in Figure 4.13) while **s2** is a short range signal (purple in Figure 4.13).

Cells that didn't switch to the transmitter state **T** and detect that the amount of **s2** is above a certain threshold switch to the **R** state and produce a red fluorescent protein (RFP). A cell in the **R** state can switch back to the undecided state if **s2** goes below the previous threshold.



Since new transmitter cells can emerge only where **s1** is not present, the distance between these cells is characterized by a lower bound.

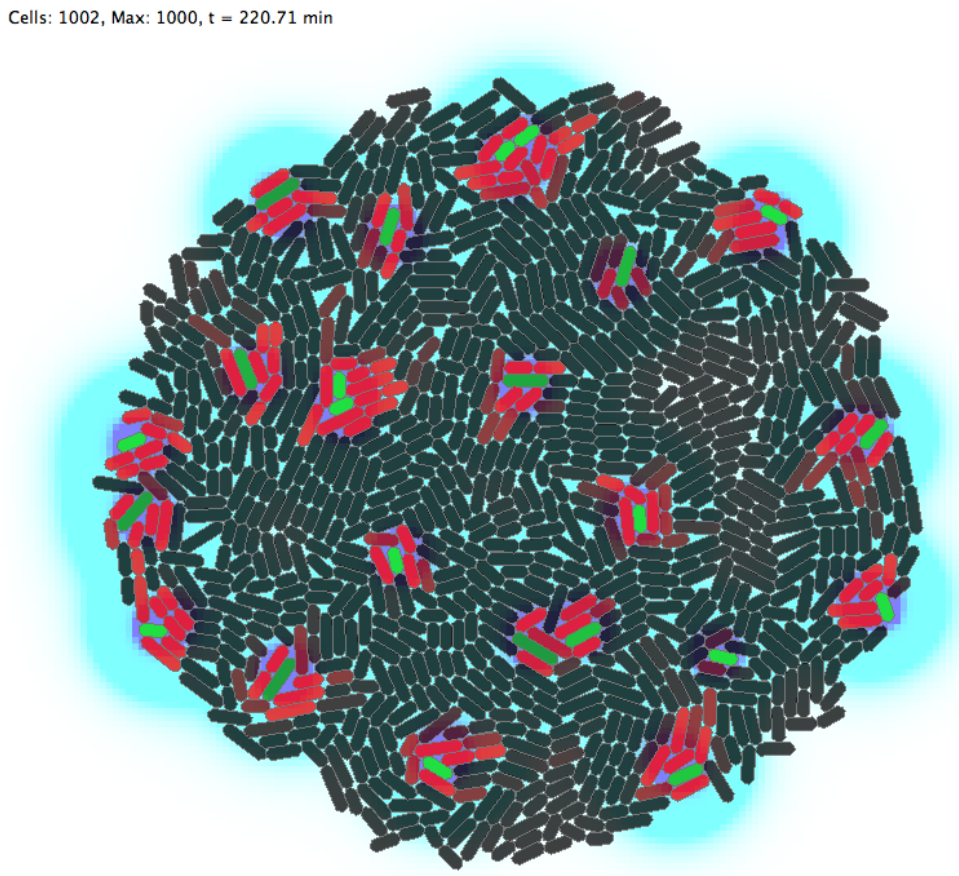
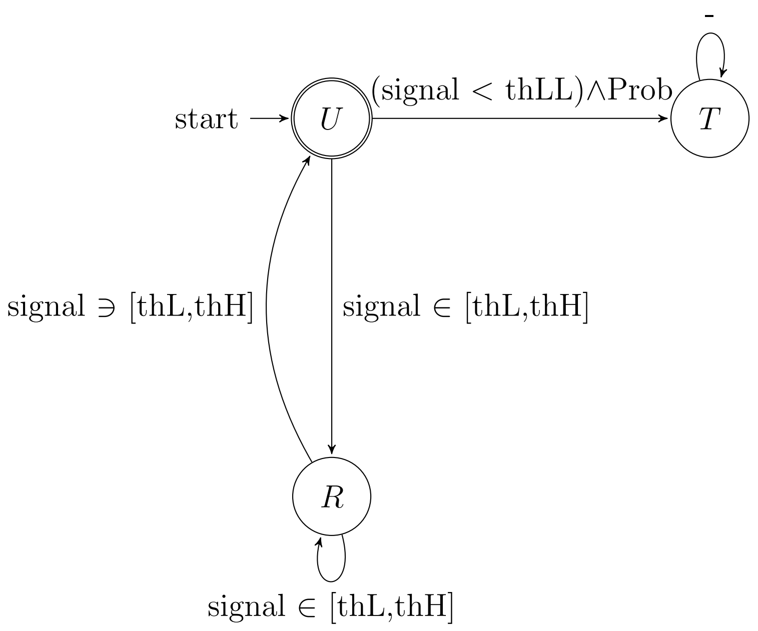


Figure 4.13: Simulation of the spots-forming finite state machine

4.5.2 Rings

A slightly different finite state machine, composed of three states, one signal and three thresholds, implements a pattern of rings. A single cell starts in the undecided state **U** then grows and divides. If the signal is below a certain threshold, a cell in the undecided state **U** can switch with a certain probability to the transmitter state **T**.

A cell in the transmitter state **T** decreases its growth rate, produces GFP and emits the signal. Cells in the undecided state **U** can detect the signal and, if it is in a specified range of concentrations, respond by switching to the **R** state and produce RFP.



The result of the simulation is a microcolony in which rings of RFP-producing cells emerge (Figure 4.14).

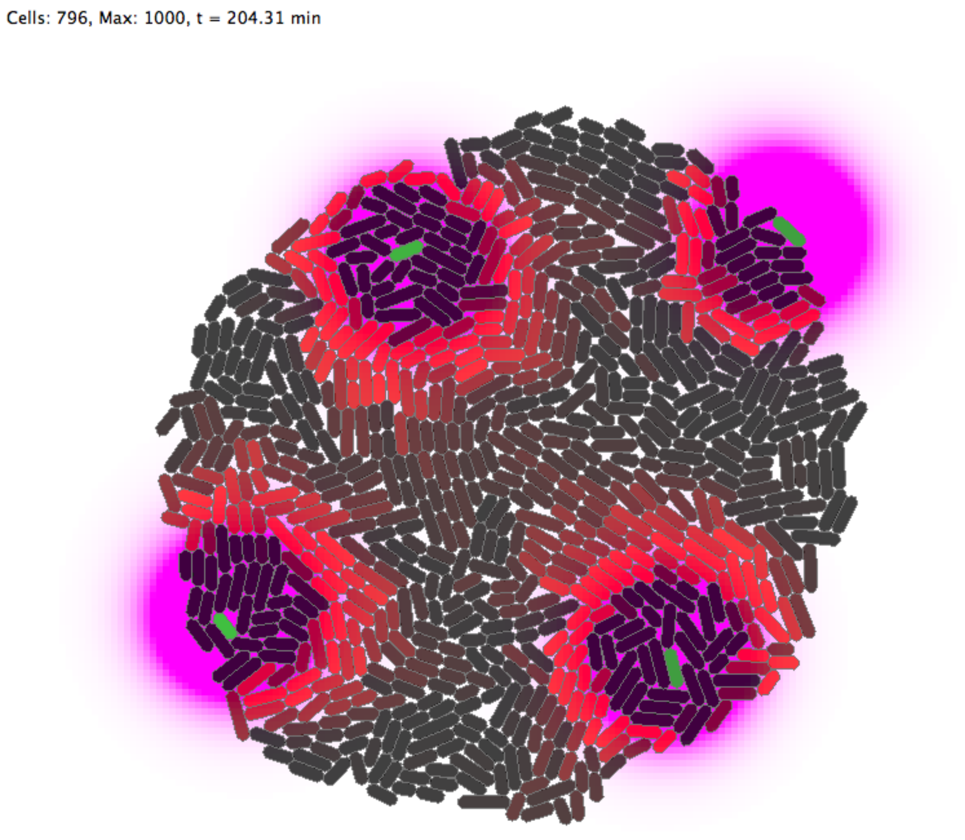


Figure 4.14: Simulation of the rings-forming finite state machine

4.5.3 Considerations

Both patterns rely on a short-range form of intercellular communication. At present, the available well characterized genetic components that implement a communication system in bacteria rely on the diffusion of small molecules like AHL. These signals are not adequate because their diffusion coefficient is usually high, this makes it very difficult to generate a steep enough signal gradient around the transmitter cell. A possible solution might come from the use of nitric oxide as a signaling molecule because it was shown that its range of action can be very short [35]. E. coli cells can sense and produce nitric oxide [36], but its use in a synthetic circuit will require a thorough characterization of the various components involved.

Another key requirement for the emergence of the spatial pattern in the previous examples is that transmitter cells need to decrease their growth rate without hampering protein production. Known toxin-antitoxin systems can regulate the rate of cellular division but usually they also affect the amount of protein that the cell can produce in a given amount of time [37]. Further research is needed to invent a satisfactory mechanism to precisely tune the growth rate of single baterial cells with minimal side effects.

Chapter 5

Appendix

5.1 Python script to split simulations on a Linux cluster

In a sensitivity analysis, the parameters' space is sampled. Since each simulation of this study is independent, it is possible to speed up the computation by running a lot of different instances at the same time.

The Department of Electrical Engineering at the University of Washington has a cluster that can be accessed remotely by researchers and students. The cluster is composed of 30 nodes, located in a server room, that run the Red Hat Enterprise distribution of the Linux operating system. Taken together, the machines total 88 processors and more than 90 GB of RAM.

To take advantage of that commodity, a Python script to split the various simulations on the cluster and retrieve the results was developed.

The client on which the script runs, that can also be one of the nodes of the cluster, needs to be a Unix system (like Linux, BSD, Mac OSX, etc..) with Python 3 installed. Client and cluster communicate via secure shell (ssh). Passwordless access to the various nodes is needed and easily set up.

The script takes as inputs:

- the name of the gro file containing the model that needs to be simulated;
- a parameter's text file in which it is defined the central point in the parameters' space and the range of variation of each parameter for the sensitivity analysis;
- the number of times that each point needs to be sampled to obtain a statistic.

Each point in the parameters' space has to be sampled more than once because gro simulations embed various sources of stochasticity.

A gro model that needs to be simulated on the cluster by means of the python script must accept the values of the parameters that are varied from command line and return the result of the simulation to the standard output via the *print()* function. To avoid problems, a stop condition that terminates the simulation with the *exit()* function when a certain event happens (e.g. simulation time above a threshold or number of cells above a limit) must be placed in the gro code.

The parameters' file is a simple text file that must start with a line containing either the word "sum" or "mult" and must end with an empty line. This file can have one of two alternative structures that are useful for different kinds of sensitivity analyses.

The first one is handy for a local sensitivity analysis: each parameter is varied from a lower value to an upper value by iteratively adding a step. The structure is illustrated in Table 5.1, where each row represent a different parameter and the columns contain, in order: lower value, upper value, step and central point of the considered parameter.

Parameter	From	To	Step	Central
P1	0.01	2	0.02	1
P2	2	40	1	18
P3	0.5	50	0.25	30
⋮	⋮	⋮	⋮	⋮

Table 5.1: Example of the first structure of the parameters' file.

To sample broadly the parameters' space, it is better to use the second structure. Each parameter is varied by multiplying the central value to a series of factors. The structure is illustrated in Table 5.2, where the last column contains the central point of the analysis and the others consist of the various factors by which the central point is multiplied, while the rows have the same meaning as before.

This script is general and not model-specific, so it can be used to simulate every gro model as long as it satisfies the requisites stated before.

Upon launch, the script requests some information to the user, then reads the parameters' file and generates a list of simulations (Figure 5.1). After that, the instances are sent to the nodes. Each node receives a number of instances that equals the number of cores on that machine. Once all the

Parameter	F1	F2	F3	...	Central
P1	0.01	0.1	1	10	5
P2	0.1	0.5	1	5	400
P3	0.005	0.05	0.5	5	5.5
⋮	⋮	⋮	⋮	⋮	

Table 5.2: Example of the second structure of the parameters' file

simulations of the first repetition are sent, the script waits for all the nodes to complete the computation. Then, the results of the first repetition are compressed, sent to the client and removed from the cluster's filesystem. The cycle repeats itself until all the repetitions have been simulated.

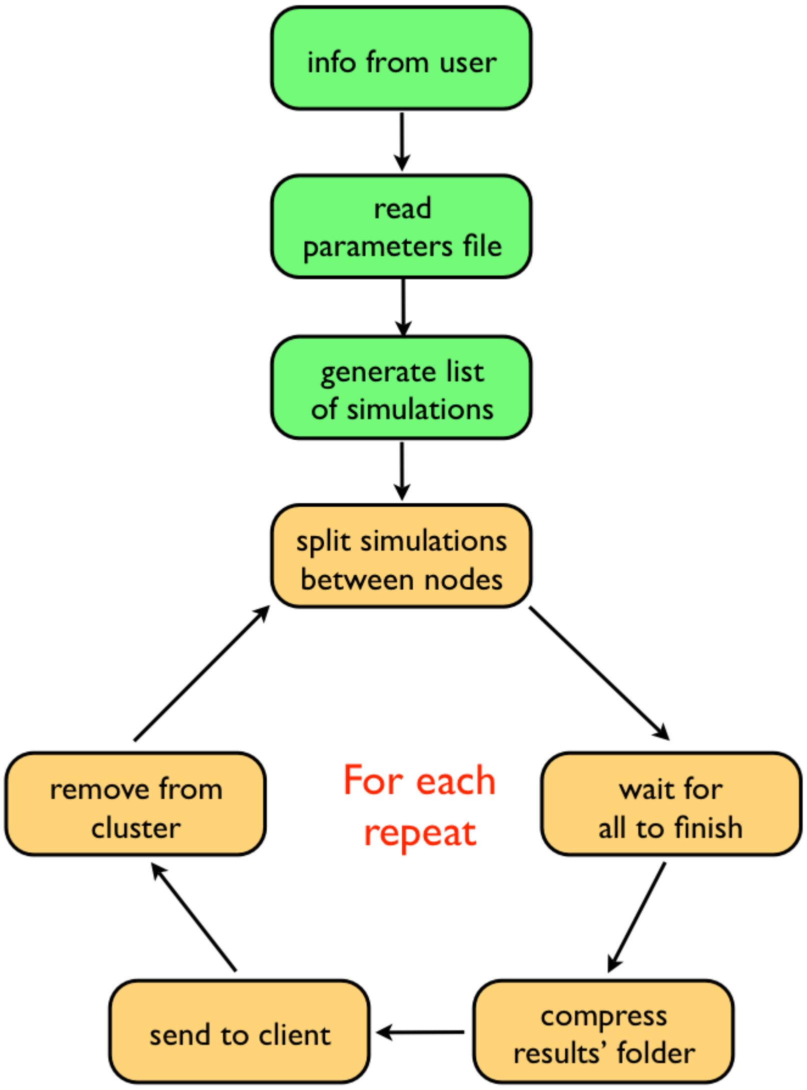


Figure 5.1: Python script diagram

```python
#!/usr/local/bin/python3

# Coded by Andrea Samore'

import os
import time
import subprocess

### Get info from the user ###
username = input('User name: ')
# Name of the gro program on the cluster
groFile = input('gro program name: ')
# Name of the parameters file on the client, it must be in the same folder where this script is
parFileName = input('Parameters file: ')
parFile = open('./'+parFileName,'r')

# Number of times that each simulation needs to be repeated in order to obtain a statistic
repeats = int(input('Repeats: '))
### Read the parameters file and set the number of workstations ###
lines = parFile.readlines()
numPar = len(lines)-1 # The first line defines how to vary the parameters
numWorkstations = 31    # There are 31 nodes in the UW EE server room

print('Your model has ' + str(numPar) + ' parameters!')

### Parameters are varied in a "linear" way ###
if str(lines[0]) == 'sum\n':

        # This list will contain the lowest value for each parameter
        fromList = []
        # This list wil contain the highest value for each parameter
        toList = []
        # This list will contain the step by which each parameter is varied
        stepList = []
        # This list contains the nominal value of all the parameters (InitialGuessList)
        IGList = []

        # Extract the information from the parameters' file and fill the previous 4 lists
        for i in range(numPar):
                linea = lines[i+1]
                print(linea)
                lun = len(linea)
                lista = []
                lastIndex = -1
                for j in range(lun):
                        if (linea[j] == ',') | (linea[j] == '\n'):
                                lista.append(linea[(lastIndex+1):j])
                                lastIndex = j
                print(lista)
                fromList.append(float(lista[0]))
                toList.append(float(lista[1]))
                stepList.append(float(lista[2]))
                IGList.append(float(lista[3]))

        print(fromList)
        print(toList)
        print(stepList)
        print(IGList)
```

```python
    # Build a big list with each set of parameters that needs to be simulated and calculate the number of simulations needed
    bigList = []
    currentList = []
    # Let's start in the nominal condition
    currentList[:] = IGList[:];
    nSim = 0

    # For each parameter..
    for i in range(numPar):
        if (i>=1):
            currentList[i-1] = IGList[i-1]
        # Starting from the lowest value, increase it by the step and add the set of parameters to the big list containing all sets that need to be simulated
        currentList[i] = fromList[i]
        if (stepList[i] != 0):
            simPar = int(round((toList[i] - fromList[i])/stepList[i],3))
            print('simPar: ' + str(simPar))
        for j in range(simPar+1):
            bigList.append(currentList[:])
            nSim = nSim +1
            currentList[i] = round(currentList[i] + stepList[i],5)

### Parameters are varied by multiplication with constants ###
elif str(lines[0]) == 'mult\n':

    # This list will contain all the factors that will multiply the considered parameters
    factorList = []
    # This list contains the nominal value of all the parameters (InitialGuessList)
    IGList = []

    simPar = []

    # For each parameter..
    for i in range(numPar):
        linea = lines[i+1]
        print(linea)
        lun = len(linea)
        lista = []
        lastIndex = -1
        for j in range(lun):
            if (linea[j] == ',') | (linea[j] == '\n'):
                lista.append(linea[(lastIndex+1):j])
                lastIndex = j
        print(lista)
        Llista = len(lista)
        simPar.append(int(Llista -1))
        print(Llista)
        for k in range(Llista -1):
            factorList.append(float(lista[k]))
        IGList.append(float(lista[Llista -1]))

    print(factorList)
    print(len(factorList))
    print(IGList)
    print(simPar)
    bigList = [];
    currentList = [];
    currentList[:] = IGList[:];
    nSim = 0;

    numPar = len(IGList)

    for i in range(numPar):
        if (i>=1):
            currentList[i-1] = IGList[i-1]
        for j in range(simPar[i]):

            currentList[i] = round(factorList[nSim]*IGList[i],5)
            nSim = nSim +1
            print(currentList)
            bigList.append(currentList[:])
    print(str(nSim) + ' simulations!')
    print(bigList)

# Build a list containing the number of cores of each of the 31 workstations
CW = []
for n in range(numWorkstations):
    # Find the number of cores of the considered workstation
    commandCores =  'ssh -l '+username+' -q linux'+str(n+12)+'.ee.
          washington.edu "cat /proc/cpuinfo | grep processor | wc -l"'
    numCores = int(subprocess.check_output([commandCores], shell=True,
          universal_newlines=True))
    CW.append(int(numCores))

print('CoreList: '+str(CW))


for umm in range(repeats):

    # Make a new folder for each repeat
    resultsFolder = 'results'+str(umm)
    subprocess.call(['ssh -l '+username+' -q linux13.ee.washington.edu "
          cd grong && mkdir '+resultsFolder+' && exit"'],shell=True)

    # Split the simulations on the 31 workstations that are in the
          server room
    indiceBigList = 0
    smallList = []

    stop = False
    while stop == False:
        for k in range(numWorkstations):
            if stop == True:
                break
            # Send to each workstation a number of simulations
                  that equals the number of cores
            for nc in range(int(CW[k])):
                comando = 'ssh -l '+username+' -q linux'+str
                      (k+12)+'.ee.washington.edu "cd grong ;
                       '
                print(str(indiceBigList))
                smallList[:] = bigList[indiceBigList][:]
                comando = comando + './grong ./'+groFile
                for i in range(numPar):
                    comando = comando +' '+str(smallList
                          [i])
                comando = comando + ' > '+resultsFolder+'/'
                for i in range(numPar):
                    if i == 0:
                        comando = comando + str("
                            {0:.3f}".format(
                            smallList[i]))
                    else:
                        comando = comando +'_'+str("
                            {0:.3f}".format(
                            smallList[i]))
                comando = comando + '.txt && '
                indiceBigList = indiceBigList+1
                if indiceBigList == nSim:
                    comando = comando + ' exit" &'
                    subprocess.call([comando],shell=True
                        )
                    stop = True
                else:
                    comando = comando + ' exit" &'
                    subprocess.call([comando],shell=True
                        )
                    time.sleep(1)
                print('stop:' +str(stop))
                if stop == True:
                    break

### Polling... Once there is no instance of gro running I can compress the
        results folder, send it back locally and remove the folder and the
        compressed folder from the cluster ###

    stopCond = False
    while stopCond == False:
        activeGro = 0
        print('Polling..')
        # Check if there is any instance of grong running on the
            cluster
        for i in range(numWorkstations):
            commandProcess =  'ssh -l '+username+' -q linux'+str
                (i+12)+'.ee.washington.edu "ps -f -C grong |
                grep '+username+' | wc -l ; exit"'
            activeGro = activeGro + int(subprocess.check_output
                ([commandProcess], shell=True,
                universal_newlines=True))
            time.sleep(1)
        if activeGro == 0:
            stopCond = True
            # Compress the current results and remove the
                corresponding folder from the cluster
            print('Compressing..')
            zipCommand =  'ssh -l '+username+' -q linux14.ee.
                washington.edu "cd grong && tar -cjf '+
                resultsFolder+'.tar.bz2 '+resultsFolder+' &&
                rm -rf '+resultsFolder+' && exit"'
            subprocess.call([zipCommand],shell=True)
            # Copy the compressed file that contains the current
                 result folder locally
            copyCommand = 'scp '+username+'@linux14.ee.
                washington.edu:grong/'+resultsFolder+'.tar.bz2
                 ./'
            subprocess.call([copyCommand],shell=True)
            # Remove the current result from the cluster
            removeCommand = 'ssh -l '+username+' -q linux14.ee.
```

```
                        washington.edu "cd grong && rm '+resultsFolder
                        +'.tar.bz2 && exit"'
                    subprocess.call([removeCommand],shell=True)
                    break
            else:
                    print('activeGro: '+ str(activeGro))
                    time.sleep(5)
```

5.2 gro Models

This section contains the *gro code* that has been used to simulate the previously described models. For more information on the syntax of the code, consult [38]

5.2.1 Leader Election

```
// Coded by Andrea Samoré

include gro

srand(-3);

// Set time step and growth rate
set ("dt",0.0025);
set ( "ecoli_growth_rate", 0.02 );

// Global variables
numCells := 1;
t := 0;
s := 0;
leaders := 0;
followers := 0;
undecided := 1;

// The signal has a constant concentration due to the fast diffusion of AHL
// I am supposing that the bacteria are in a box with sides 400µm x 400µm
// and height 1 µm.
globalSignal := 0;

// Input functions of the various promoters and binding of AHL to LuxR
fun repressor beta X Kd . beta / (1+ (X/Kd)^2);
fun activator beta X Kd . (beta * X^2) / (X^2 + Kd^2);
fun michment S Xt Ks . ((Ks + Xt + S) - sqrt((Ks + Xt + S)^2 - 4*(Xt*S)))/2;
fun actrepr beta X Y Kx Ky . (beta*(X/Kx)^2) / (1 + (X/Kx)^2 + (Y/Ky)^2 + ((X*Y)/(Kx*Ky))^2 );

// Parameters
// Maximal transcription and translation rate of the sender module
betaLuxR := atof(ARGV[2]);

// LuxR protein degradation constant
KdgrLuxR := 0.02;

// Equilibrium constant of AHL-LuxR binding
KsAhlLuxR := 100;

// Equilibrium constant of AHL-LuxR binding the promoter
KdAhlLuxR := atof(ARGV[3]);

// Maximal transcription and translation rate of the operon in the coin-flipper module
betaAraC := atof(ARGV[4]);

// Equilibrium constant of AraC-Arabinose binding the promoter
KdAraC := atof(ARGV[5]);

// AraC protein degradation rate
KdgrAraC := 0.01;
```

```
// Equilibrium constant of LacI binding the promoter
KdLacI := atof(ARGV[6]);

// LacI protein degradation rate
KdgrLacI := 0.02;

// Maximal transcription and translation rate of the sender module
betaLuxI := atof(ARGV[7]);

// LuxI protein degradation constant
KdgrLuxI := 0.01;

// Maximal transcription and translation rate of the follower module
betaTetR := atof(ARGV[8]);

// Equilibrium constant of TetR binding the promoter
KdTetR := atof(ARGV[9]);

// TetR protein degradation constant
KdgrTetR := 0.01;

// Basal expression of the coin-flipper module
leakiness := atof(ARGV[10]);

// Basal expression of all the other modules
leakiness2 := 0.01;

// Degradation constant of the signal
KdgrAHL := 0.0018;


program le(leakiness,leakiness2,betaLuxR,KdLacI,KdgrLuxR,betaLuxI,KdAraC,KdgrLuxI, ...
betaAraC,KdTetR,KdgrAraC,KdgrLacI,betaTetR,KsAhlLuxR,KdAhlLuxR,KdgrTetR,KdgrAHL) := {

// Initialization
rfp := 0;
gfp := 0;
araC := 0;
lacI := 0;
luxI := 0;
luxR := 0;
tetR := 0;

state := [value:= 0];

daughter : {
numCells := numCells + 1;
state.value := 0;
undecided := undecided +1;
};

// Degradation terms
rate (KdgrAraC * araC/volume) : { araC := araC - 1};
rate (KdgrLacI * lacI/volume) : { lacI := lacI - 1};
rate (KdgrLuxI * luxI/volume) : { luxI := luxI - 1};
rate (KdgrLuxR * luxR/volume) : { luxR := luxR - 1};
rate (KdgrTetR * tetR/volume) : { tetR := tetR - 1};

// GFP and RFP are fixed to the amount of AraC and tetR for visualization purposes
true : {
gfp := araC;
rfp := tetR;
};

// Coin-flipper module production term
rate(actrepr betaAraC (araC/volume) (tetR/volume) KdAraC KdTetR) : {
araC := araC + 1;
lacI := lacI + 1;
};

// Coin-flipper module basal expression
rate(leakiness): {
araC := araC + 1;
lacI := lacI + 1;
};

// Sender module production term
rate(activator betaLuxI (araC/volume) KdAraC) : {
luxI := luxI +1;
};

// Sender module basal expression
rate(leakiness2) : {
luxI := luxI + 1;
};

// Receiver module production term
rate(repressor betaLuxR (lacI/volume) KdLacI) : {
luxR := luxR +1;
};

// Receiver module basal expression
rate(leakiness2) : {
luxR := luxR + 1;
};

// Follower module production term
rate(actrepr betaTetR (michment ( globalSignal) (luxR/volume) KsAhlLuxR) (lacI/volume) KdAhlLuxR KdLacI) : {
tetR := tetR +1;
luxI := luxI + 1;  // Signal relay!
};

// Follower module basal expression
rate(leakiness2) : {
tetR := tetR +1;
luxI := luxI + 1;  // Signal relay!
};

// Signal production term, each cell contributes with its intracellular concentration of LuxI
 true :  {
  globalSignal := globalSignal + (1.1*luxI/(160000) * dt);
 };

// Classification

// Undecided --> Leader
(state.value = 0) & (araC/volume > 20 ) & (tetR/volume < 20 ) : {
state.value := 1; // Leader!
leaders := leaders +1;
undecided := undecided -1;
};

// Follower --> Leader
```

```
(state.value = 2) & (araC/volume > 20 ) & (tetR/volume < 20 ) : {
state.value := 1; // Leader!
leaders := leaders +1;
followers := followers -1;
};

// Undecided --> Follower
(state.value = 0) & (araC/volume < 20 ) & (tetR/volume > 20 ) : {
state.value := 2; // Follower!
followers := followers +1;
undecided := undecided -1;
};

// Leader --> Follower
(state.value = 1) & (araC/volume < 20 ) & (tetR/volume > 20 ) : {
state.value := 2; // Follower!
followers := followers +1;
leaders := leaders -1;
};

//Leader --> Undecided
(state.value = 1) & (araC/volume > 20) & (tetR/volume >20) : {
state.value := 0; // undecided
undecided := undecided +1;
leaders := leaders -1;
};
(state.value = 1) & (araC/volume < 20) & (tetR/volume < 20): {
state.value := 0; //undecided
undecided := undecided +1;
leaders := leaders -1;
};

// follower --> undecided
(state.value = 2) & (araC/volume > 20 ) & (tetR/volume > 20 ) : {
                state.value := 0; // undecided!
                followers := followers -1;
                undecided := undecided +1;
        };
(state.value = 2) & (araC/volume < 20 ) & (tetR/volume < 20 ) : {
                state.value := 0; // undecided
                followers := followers -1;
                undecided := undecided +1;
        };

};

program report() := {
needs lacI;
needs araC;
needs luxI;
needs tetR;
needs luxR;
needs p;
selected : {message(1,tostring(id) <> " [luxI]: " <> tostring(luxI) <> " [ahl]: " ...
 <> tostring( globalSignal ) <> " [tetR]: " <> tostring( tetR/volume ))};
selected : {message(2,tostring(id) <> " [araC]: " <> tostring(araC/volume) <> ...
" [lacI]: " <> tostring(lacI/volume) <> " [luxR]: " <> tostring(luxR/volume) )};
};

program p(leakiness,leakiness2,betaLuxR,KdLacI,KdgrLuxR,betaLuxI,KdAraC,KdgrLuxI, ...
betaAraC,KdTetR,KdgrAraC,KdgrLacI,betaTetR,KsAhlLuxR,KdAhlLuxR,KdgrTetR,KdgrAHL) :=
 le(leakiness,leakiness2,betaLuxR,KdLacI,KdgrLuxR,betaLuxI,KdAraC,KdgrLuxI,betaAraC,KdTetR, ...
KdgrAraC,KdgrLacI,betaTetR,KsAhlLuxR,KdAhlLuxR,KdgrTetR,KdgrAHL)
 + report() sharing lacI , araC, luxI, tetR, luxR, gfp, rfp, p;

program main() := {

// Signal degradation term
    true : {globalSignal := globalSignal - KdgrAHL * globalSignal * dt};

true : {t := t + dt};
true : {s := s + dt};

// Stop condition
numCells >= 50 : {
exit();
};

// Output every minute
s>1 : {
print(t,",",numCells,",",leaders,",",followers,",",undecided,"\n");
s := 0;
};

};

// Place a single E. coli cell in the center of the volume and start the simulation
ecoli ([x := 0, y := 0], program p(leakiness,leakiness2,betaLuxR,KdLacI,KdgrLuxR,betaLuxI,KdAraC, ...
KdgrLuxI,betaAraC,KdTetR,KdgrAraC,KdgrLacI,betaTetR,KsAhlLuxR,KdAhlLuxR,KdgrTetR,KdgrAHL));

start();
```

5.2.2 Spots

```
// Coded by Andrea Samoré

include gro
set ("dt",0.05);
// Short range signal definition
SR := signal (0.3,1);

// Long range signal definition
LR := signal (3,0.5);

program g1() := {

// Initialization
rfp := 0;
gfp := 0;
p :=[state := 0, thSR := 1, thLR := 0.5];

// RFP degradation
rate(0.05*rfp) : {rfp := rfp-1};

// Switch with a certain probability from the undecided state to the leader state if
// the long range signal is below a specified threshold
(p.state = 0) & rate(0.01) & (get_signal(LR) < p.thLR) : {
p.state := 1;
};

// Once in the leader state, emit the two signals, decrease the growth rate and
// become green
(p.state = 1) : {
emit_signal (SR,70);
emit_signal (LR,100);
set ("ecoli_growth_rate",0.001);
gfp := 100;
};

// If in the undecided state and the short range signal emitted by a leader is above a
// certain threshold then switch to the "red" state
(p.state = 0) & ( get_signal (SR) > p.thSR ) : {
p.state := 3;
};

 // In the "red" state the cell produces rfp. It gets out of the "red" state if the short
 // range signal emitted by the leader is below a certain threshold
 (p.state = 3) : {
  rfp := rfp + 1;
  p.state := if ( get_signal (SR) < p.thSR) then 0 else 3 end;
 };
};

program report() := {
needs rfp;
selected : {
message(1,tostring(id) <> " rfp: " <> tostring(rfp) <> " SR:" <> tostring(get_signal(SR)) <> ...
 " LR:" <> tostring(get_signal(LR)))
};
};

program p() := g1() + report() sharing gfp,rfp;
ecoli([x := 0,y:=0], program p());
start();
```

5.2.3 Rings

```
// Coded by Andrea Samoré

include gro
// Set the integration step.
set ("dt",0.05);

// Initialization of the random number generator, a negative number means that t is
// initialized each time with a different seed.
srand(-3);

// Declare the signal's diffusion and degradation rates
aS := signal (4,0.5);

program g1() := {

// Initialization of the copy number of the fluorescent markers.
rfp := 0;
gfp := 0;

// RFP degradation rate.
rate(0.05*rfp) : {rfp := rfp-1};

// N.B. GFP does not degrade because it is used only to highlight the cell that emits the
// signal.

// A record protects these variables from being split upon cell division.
p :=[state := 0, thLL := 0.01, thL := 0.5, thH := 1.5];

// If the cell is in the undecided state 0 and the signal is below a certain threshold
// then it can switch to the transmitter state with a specified probability and emit a
// burst of signal in the transition.
(p.state = 0) & rate(0.001) & (get_signal(aS)<p.thLL) : {
emit_signal (aS,1000);
p.state := 1;
};

// Once in the transmitter state, the cell keeps emitting signal and slows its growth
// rate.
(p.state = 1) : {
emit_signal (aS,100);
gfp := 100;
set ("ecoli_growth_rate",0.001);
};

// A cell in the undecided state switches to the band state if it detects that the signal
// is in a specified range of concentrations.
 (p.state = 0) & ( (get_signal(aS) > p.thL) & (get_signal(aS) < p.thH) ) : {
  p.state := 4;
  };

// Once in the T state, it produces RFP. If the detected signal is now outside the specified
// range, the cell switches back to the undecided state.
(p.state = 4) : {
p.state := if (get_signal(aS) < p.thL) | (get_signal(aS) > p.thH) then 0 else 4 end;
rfp := rfp+1;
};

};
 // This code show information about the content of RFP and the level of signal detected
 // by a cell when it is selected from the simulator.
```

```
program report() := {
needs rfp;
selected : {
message(1,tostring(id) <> " rfp: " <> tostring(rfp) <> " aS:" <> tostring(get_signal(aS)))
};
};

// Program composition.
program p() := g1() + report() sharing gfp,cfp,rfp;

// Initialization of a single E. coli cell in the center of the field of view.
ecoli([x:=0,y:=0], program p());
start();
```

5.3 Matlab Code

The data analysis was performed using MATLAB (r2012a, The MathWorks, Natick, MA). This section contains the code.

```
% Coded by Andrea Samore'

%% Load results

clear
close all
clc

[a b] = system('ls -al results*.tar.bz2 | wc -l');

% Number of repeats
b= = str2double(b);

% Name of each parameter
namePar = {'beta_{LuxR}';' Kd_{AHLLuxR}';'beta_{AraC}';'Kd_{AraC}'; 'Kd_{
      LacI}'...
           ; 'beta_{LuxI}'; 'beta_{TetR}'; 'Kd_{TetR}'; 'leakiness'};

numPar = length(namePar);

%Extract from tarball
fprintf('%d results to extract!\n',b);

for i = 0:(b-1)
    fprintf('Extracting result %d of %d\n',i+1,b);
    name = sprintf('./results%d.tar.bz2',i);
    system(sprintf('tar -jxf %s',name));
end

fprintf('\nDone!\n')

% Central point in the parameters space.
centralPoint = [40 10 5 5 6 40 40 5 0.01];

% Factors by which the parameters are multiplied. This script works only if
% the number of factors is the same for all the parameters.
factorList = [0.05 0.1 0.25 0.5 1 2.5 5 10];

% Big Matrix containing all the combinations of parameters (filled in the
      following cycles)
```

```
bigM = [];

% All values for the considered parameter
parlist = [];

indice = 1;
for j = 1:numPar

    for gg = 1:length(factorList)
       parlist(gg) = factorList(gg)*centralPoint(j);
    end

    for i = 1:length(parlist)

        bigM(indice,:) = centralPoint;
        bigM(indice,j) = parlist(i);

        % For each repeat
        for k = 0:(b-1)
            % Print what you want to open
            fprintf('./results%d/%.5f_%.5f_%.5f_%.5f_%.5f_%.5f_%.5f_%.5f_%.5
                  f.txt\n',k,bigM(indice,1),bigM(indice,2),bigM(indice,3),
                  bigM(indice,4),bigM(indice,5),bigM(indice,6),bigM(indice
                  ,7),bigM(indice,8),bigM(indice,9));
            % Then open it and put it in a matrix
            part = csvread(sprintf('./results%d/%.5f_%.5f_%.5f_%.5f_%.5f_%.5
                  f_%.5f_%.5f_%.5f.txt',k,bigM(indice,1),bigM(indice,2),bigM
                  (indice,3),bigM(indice,4),bigM(indice,5),bigM(indice,6),
                  bigM(indice,7),bigM(indice,8),bigM(indice,9)));
            % Calculate the fraction of leaders, followers and undecided
            % and if everything goes smoothly put them in a cell
            % array
            if ~isnan(part(end,3:end)./sum(part(end,3:end)))
                % j: number of parameters;
                % i: number of steps of the considered parameter;
                % k: number of repeats - 1
                % Consider only the fraction of cells in the various states
                % of the last step of the simulation.
                res{j,i}(k+1,:) = part(end,3:end)./sum(part(end,3:end));
            end
            clear part
        end
        indice = indice + 1;
    end

end

% a: number of parameters;
% l: maximum number of steps.

[a l] = size(res);

for i = 1:a
   for j = 1:l
        if ~isempty(res{i,j})
          MS{i}(j,1:3) = mean(res{i,j});
          MS{i}(j,4:6) = std(res{i,j});
        end
   end
end
```

```
% matrix of "x axes"
matrice = zeros(length(centralPoint),length(factorList));

for w = 1:length(centralPoint)
    for y = 1:length(factorList)
        matrice(w,y) = factorList(y)*centralPoint(w);
    end
end


%% Plot raw results

close all

for i = 1:a
    h = figure(i);
    hold on
    nameFig = sprintf('./figsDots/fig%d.eps',i);
    indice = 8;
    for k = 1:indice
        plot(factorList(k)*centralPoint(i),res{i,k}(:,1),'gx');
        plot(factorList(k)*centralPoint(i),res{i,k}(:,2),'r*');
        plot(factorList(k)*centralPoint(i),res{i,k}(:,3),'ko');

    end
    ylabel('%L,F,U');
    xlabel(sprintf('%s',namePar{i,:}));
    set(gca, 'xscale', 'log')
    grid
    print(h,'-depsc2',nameFig);
end

%% Plot of the mean and standard deviation of the fractions over all repeats
    .

close all

for i = 1:a
    h = figure(i);
    hold on
    nameFig = sprintf('./figsMean/fig%d.eps',i);
    errorbar(matrice(i,:),MS{i}(:,1),MS{i}(:,4),'g--')
    errorbar(matrice(i,:),MS{i}(:,1),MS{i}(:,4),'*g')
    errorbar(matrice(i,:),MS{i}(:,2),MS{i}(:,5),'r--')
    errorbar(matrice(i,:),MS{i}(:,2),MS{i}(:,5),'*r')
    errorbar(matrice(i,:),MS{i}(:,3),MS{i}(:,6),'k--')
    errorbar(matrice(i,:),MS{i}(:,3),MS{i}(:,6),'*k')
    ylabel('%L,F,U');
    xlabel(sprintf('%s',namePar{i,:}));
    set(gca, 'xscale', 'log')
    grid
    print(h,'-depsc2',nameFig);
end

%% All Costs

close all
% Optimal fraction of leaders, followers and undecided
opt = [0.1 0.9 0];
%opt = [0.5 0.5 0];
%opt = [0.9 0.1 0];

```

```
% Calculate mean and ds of the cost function

for i = 1:a
    for j = 1:l
        for k = 1:length(res{i,j}(:,1))
            if ~isempty(res{i,j})
                allCosts{i,j}(k,1) = norm([res{i,j}(k,1)-opt(1),res{i,j}(k
                      ,2)-opt(2),res{i,j}(k,3)-opt(3)],2);
            end
        end
        cost{i}(j,1) = mean(allCosts{i,j}(:,1));
        cost{i}(j,2) = std(allCosts{i,j}(:,1));
    end
end

for i = 1:a
    h = figure(i);
    hold on
    errorbar(matrice(i,:),cost{i}(:,1),cost{i}(:,2),'k--')
    errorbar(matrice(i,:),cost{i}(:,1),cost{i}(:,2),'*k')
    [minCost, Ind]= min(cost{i}(:,1));
    if Ind < 5
        fitting(i,:) = polyfit(log10(matrice(i,Ind:5)),cost{i}(Ind:5,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,Ind:5)));
        plot(matrice(i,Ind:5),line,'r')
    end
    if Ind > 5
        fitting(i,:) = polyfit(log10(matrice(i,5:Ind)),cost{i}(5:Ind,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,5:Ind)));
        plot(matrice(i,5:Ind),line,'r')
    end
    if Ind == 5
        fitting(i,:) = polyfit(log10(matrice(i,:)),cost{i}(:,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,:)));
        plot(matrice(i,:),line,'r')
    end
    nameFig = sprintf('./figsCost/fig%d.eps',i);
    ylabel('Cost');
    xlabel(sprintf('%s',namePar{i,:}));
    ylim([0 sqrt(2)])
    plot(centralPoint(i),cost{i}(5,1),'s','MarkerEdgeColor','k','
          MarkerFaceColor','y','MarkerSize',10)
    set(gca, 'xscale', 'log')
    set(gca, 'xscale', 'log')
    title(sprintf('%g Leaders, %g Followers, %g Undecided',opt(1),opt(2),opt
          (3)))
    grid
    print(h,'-depsc2',nameFig);
end

%% subplot 0.1L 0.9F 0U

close all

% Optimal fraction of leaders, followers and undecided
opt = [0.1 0.9 0];

% Calculate mean and ds of the cost function
for i = 1:a
    for j = 1:l
        for k = 1:length(res{i,j}(:,1))
            if ~isempty(res{i,j})
```

```
                allCosts{i,j}(k,1) = norm([res{i,j}(k,1)-opt(1),res{i,j}(k
                    ,2)-opt(2),res{i,j}(k,3)-opt(3)],2);
            end
        end
        cost{i}(j,1) = mean(allCosts{i,j}(:,1));
        cost{i}(j,2) = std(allCosts{i,j}(:,1));
    end
end

indiceplot = 1;
for i = [2 6 5 7]
    subplot(4,2,indiceplot)
    errorbar(matrice(i,:),cost{i}(:,1),cost{i}(:,2),'k--')
    hold on
    errorbar(matrice(i,:),cost{i}(:,1),cost{i}(:,2),'*k')
    [minCost, Ind]= min(cost{i}(:,1));
    if Ind < 5
        fitting(i,:) = polyfit(log10(matrice(i,Ind:5)),cost{i}(Ind:5,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,Ind:5)));
        plot(matrice(i,Ind:5),line,'r')
    end
    if Ind > 5
        fitting(i,:) = polyfit(log10(matrice(i,5:Ind)),cost{i}(5:Ind,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,5:Ind)));
        plot(matrice(i,5:Ind),line,'r')
    end
    if Ind == 5
        fitting(i,:) = polyfit(log10(matrice(i,:)),cost{i}(:,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,:)));
        plot(matrice(i,:),line,'r')
    end
    ylabel('Cost');
    xlabel(sprintf('%s',namePar{i,:}));
    ylim([0 sqrt(2)])
    plot(centralPoint(i),cost{i}(5,1),'s','MarkerEdgeColor','k','
        MarkerFaceColor','y','MarkerSize',10)
    set(gca, 'xscale', 'log')
    title(sprintf('%g Leaders, %g Followers, %g Undecided',opt(1),opt(2),opt
        (3)))
    grid
    indiceplot = indiceplot +2;
end

indiceplot = 2;
for i = [2 6 5 7]
    subplot(4,2,indiceplot)
    hold on
    errorbar(matrice(i,:),MS{i}(:,1),MS{i}(:,4),'g--')
    errorbar(matrice(i,:),MS{i}(:,1),MS{i}(:,4),'*g')
    errorbar(matrice(i,:),MS{i}(:,2),MS{i}(:,5),'r--')
    errorbar(matrice(i,:),MS{i}(:,2),MS{i}(:,5),'*r')
    errorbar(matrice(i,:),MS{i}(:,3),MS{i}(:,6),'k--')
    errorbar(matrice(i,:),MS{i}(:,3),MS{i}(:,6),'*k')
    ylabel('%L,F,U');
    xlabel(sprintf('%s',namePar{i,:}));
    set(gca, 'xscale', 'log')
    grid
    indiceplot = indiceplot + 2;
end

%% subplot 0.5L 0.5F 0U
close all
```

```
% Optimal fraction of leaders, followers and undecided
opt = [0.5 0.5 0];

% Calculate mean and ds of the cost function

for i = 1:a
    for j = 1:l
        for k = 1:length(res{i,j}(:,1))
            if ~isempty(res{i,j})
                allCosts{i,j}(k,1) = norm([res{i,j}(k,1)-opt(1),res{i,j}(k
                      ,2)-opt(2),res{i,j}(k,3)-opt(3)],2);
            end
        end
        cost{i}(j,1) = mean(allCosts{i,j}(:,1));
        cost{i}(j,2) = std(allCosts{i,j}(:,1));
    end
end

indiceplot = 1;
for i = [6 7 2 5]
    subplot(4,2,indiceplot)
    errorbar(matrice(i,:),cost{i}(:,1),cost{i}(:,2),'k--')
    hold on
    errorbar(matrice(i,:),cost{i}(:,1),cost{i}(:,2),'*k')
    [minCost, Ind]= min(cost{i}(:,1));
    if Ind < 5
        fitting(i,:) = polyfit(log10(matrice(i,Ind:5)),cost{i}(Ind:5,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,Ind:5)));
        plot(matrice(i,Ind:5),line,'r')
    end
    if Ind > 5
        fitting(i,:) = polyfit(log10(matrice(i,5:Ind)),cost{i}(5:Ind,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,5:Ind)));
        plot(matrice(i,5:Ind),line,'r')
    end
    if Ind == 5
        fitting(i,:) = polyfit(log10(matrice(i,:)),cost{i}(:,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,:)));
        plot(matrice(i,:),line,'r')
    end
    ylabel('Cost');
    xlabel(sprintf('%s',namePar{i,:}));
    ylim([0 sqrt(2)])
    plot(centralPoint(i),cost{i}(5,1),'s','MarkerEdgeColor','k','
          MarkerFaceColor','y','MarkerSize',10)
    set(gca, 'xscale', 'log')
    title(sprintf('%g Leaders, %g Followers, %g Undecided',opt(1),opt(2),opt
          (3)))
    grid
    indiceplot = indiceplot +2;
end

indiceplot = 2;
for i = [6 7 2 5]
    subplot(4,2,indiceplot)
    hold on
    errorbar(matrice(i,:),MS{i}(:,1),MS{i}(:,4),'g--')
    errorbar(matrice(i,:),MS{i}(:,1),MS{i}(:,4),'*g')
    errorbar(matrice(i,:),MS{i}(:,2),MS{i}(:,5),'r--')
    errorbar(matrice(i,:),MS{i}(:,2),MS{i}(:,5),'*r')
    errorbar(matrice(i,:),MS{i}(:,3),MS{i}(:,6),'k--')
    errorbar(matrice(i,:),MS{i}(:,3),MS{i}(:,6),'*k')
    ylabel('%L,F,U');
    xlabel(sprintf('%s',namePar{i,:}));
    set(gca, 'xscale', 'log')
    grid
    indiceplot = indiceplot + 2;
end

%% subplot 0.9L 0.1F 0U

close all

% Optimal fraction of leaders, followers and undecided
opt = [0.9 0.1 0];

% Calculate mean and ds of the cost function

for i = 1:a
    for j = 1:l
        for k = 1:length(res{i,j}(:,1))
            if ~isempty(res{i,j})
                allCosts{i,j}(k,1) = norm([res{i,j}(k,1)-opt(1),res{i,j}(k
                      ,2)-opt(2),res{i,j}(k,3)-opt(3)],2);
            end
        end
        cost{i}(j,1) = mean(allCosts{i,j}(:,1));
        cost{i}(j,2) = std(allCosts{i,j}(:,1));
    end
end

indiceplot = 1;
for i = [9 4 3 8]
    subplot(4,2,indiceplot)
    errorbar(matrice(i,:),cost{i}(:,1),cost{i}(:,2),'k--')
    hold on
    errorbar(matrice(i,:),cost{i}(:,1),cost{i}(:,2),'*k')
    [minCost, Ind]= min(cost{i}(:,1));
    if Ind < 5
        fitting(i,:) = polyfit(log10(matrice(i,Ind:5)),cost{i}(Ind:5,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,Ind:5)));
        plot(matrice(i,Ind:5),line,'r')
    end
    if Ind > 5
        fitting(i,:) = polyfit(log10(matrice(i,5:Ind)),cost{i}(5:Ind,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,5:Ind)));
        plot(matrice(i,5:Ind),line,'r')
    end
    if Ind == 5
        fitting(i,:) = polyfit(log10(matrice(i,:)),cost{i}(:,1)',1);
        line = polyval(fitting(i,:),log10(matrice(i,:)));
        plot(matrice(i,:),line,'r')
    end
    ylabel('Cost');
    xlabel(sprintf('%s',namePar{i,:}));
    ylim([0 sqrt(2)])
    plot(centralPoint(i),cost{i}(5,1),'s','MarkerEdgeColor','k','
          MarkerFaceColor','y','MarkerSize',10)
    set(gca, 'xscale', 'log')
    title(sprintf('%g Leaders, %g Followers, %g Undecided',opt(1),opt(2),opt
          (3)))
    grid
    indiceplot = indiceplot +2;
```

```
end

indiceplot = 2;
for i = [9 4 3 8]
    subplot(4,2,indiceplot)
    hold on
    errorbar(matrice(i,:),MS{i}(:,1),MS{i}(:,4),'g--')
    errorbar(matrice(i,:),MS{i}(:,1),MS{i}(:,4),'*g')
    errorbar(matrice(i,:),MS{i}(:,2),MS{i}(:,5),'r--')
    errorbar(matrice(i,:),MS{i}(:,2),MS{i}(:,5),'*r')
    errorbar(matrice(i,:),MS{i}(:,3),MS{i}(:,6),'k--')
    errorbar(matrice(i,:),MS{i}(:,3),MS{i}(:,6),'*k')
    ylabel('%L,F,U');
    xlabel(sprintf('%s',namePar{i,:}));
    set(gca, 'xscale', 'log')
    grid
    indiceplot = indiceplot + 2;
end

%% Scatter plots: dCost/d(log(parameter))

close all

% Matrix of optimal values
opt = [0.9 0.1 0;0.75 0.25 0;0.5 0.5 0;0.25 0.75 0;0.1 0.9 0];
color = ['b','g','r','c'];

% Calculate mean and ds of the cost function
for ndi = 1:length(opt(:,1))
    for i = 1:a
        for j = 1:l
            for k = 1:length(res{i,j}(:,1))
                if ~isempty(res{i,j})
                    allCosts{i,j}(k,1) = norm([res{i,j}(k,1)-opt(ndi,1),res{
                        i,j}(k,2)-opt(ndi,2),res{i,j}(k,3)-opt(ndi,3)],2);
                end
            end
            cost{i}(j,1) = mean(allCosts{i,j}(:,1));
            cost{i}(j,2) = std(allCosts{i,j}(:,1));
        end
    end

    h = figure(ndi);
    hold on
    nameFig = sprintf('./figsSamo/fig%d.eps',ndi);

    for i = 1:a
        [minCost2(ndi,i), Ind]= min(cost{i}(:,1));
        if Ind < 5
            fitting(i,:) = polyfit(log10(matrice(i,Ind:5)),cost{i}(Ind:5,1)
                ',1);
        end
        if Ind > 5
            fitting(i,:) = polyfit(log10(matrice(i,5:Ind)),cost{i}(5:Ind,1)
                ',1);
        end
        if Ind == 5
            fitting(i,:) = polyfit(log10(matrice(i,:)),cost{i}(:,1)',1);
        end
        if Ind ~= 5
            if fitting(i,1)>0
                plot(minCost2(ndi,i),abs(fitting(i,1)),'b*')
                text(minCost2(ndi,i),abs(fitting(i,1)),namePar(i),'
                      VerticalAlignment','bottom');
            end
            if fitting(i,1)<0
                plot(minCost2(ndi,i),abs(fitting(i,1)),'r*')
                text(minCost2(ndi,i),abs(fitting(i,1)),namePar(i),'
                      VerticalAlignment','bottom');
            end
        end
    end

    plot([0 .8],[0.4 0.4],'k--')
    plot([0.4 0.4],[0 .8],'k--')
    xlim([0 .8])
    ylim([0 .8])
    xlabel('min(cost function)')
    ylabel('slope')
    title(sprintf('%g Leaders, %g Followers, %g Undecided',opt(ndi,1),opt(
          ndi,2),opt(ndi,3)))
    grid
    print(h,'-depsc2',nameFig);

end
```

Bibliography

[1] Mankind. `http://en.wikipedia.org/wiki/Nucleotides`.

[2] Mankind. `http://en.wikipedia.org/wiki/DNA`.

[3] Mankind. `http://en.wikipedia.org/wiki/Escherichia_coli`.

[4] S. Jang and E. Klavins. Manuscript in preparation. 2013.

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

[6] T.S Gardner, C.R Cantor, and J.J Collins. Construction of a genetic toggle switch in escherichia coli supplementary information.

[7] Adam P Arkin and Daniel A Fletcher. somewhat in control. *Genome Biol*, 7(8):114, Jan 2006.

[8] R Egbert and E Klavins. Fine-tuning gene networks using simple sequence repeats. *Proceedings of the National Academy of Sciences*, Jan 2012.

[9] Drew Endy. Foundations for engineering biology. *Nature*, 438(7067):449–453, Nov 2005.

[10] Ernesto Andrianantoandro, Subhayu Basu, David K Karig, and Ron Weiss. Synthetic biology: new engineering rules for an emerging discipline. *Molecular Systems Biology*, 2:1–14, May 2006.

[11] R Schleif. Regulation of the-arabinose operon of escherichia coli. *Trends in Genetics*, 16(12):559–565, 2000.

[12] R Schleif. Arac protein, regulation of the l-arabinose operon in escherichia coli, and the light switch mechanism of arac action. *FEMS microbiology reviews*, 34(5):779–796, 2010.

[13] O Díaz-Hernández and M Santillán. Bistable behavior of the lac operon in e. coli when induced with a mixture of lactose and tmg. *Frontiers in Physiology*, 1, 2010.

[14] C Fuqua and E Greenberg. Listening in on bacteria: acyl-homoserine lactone signalling. *Nature Reviews: Molecular ...*, Jan 2002.

[15] A.L Schaefer, D.L Val, B.L Hanzelka, J.E Cronan Jr, and EP Greenberg. Generation of cell-to-cell signals in quorum sensing: acyl homoserine lactone synthase activity of a purified vibrio fischeri luxi protein. *Proceedings of the National Academy of Sciences*, 93(18):9505–9509, 1996.

[16] P Orth, D Schnappinger, W Hillen, W Saenger, and W Hinrichs. Structural basis of gene regulation by the tetracycline inducible tet repressor-operator system. *Nature structural biology*, 7(3):215–219, 2000.

[17] Z Szallasi, J Stelling, and V Periwal. *System Modeling in Cell Biology: From Concepts to Nuts and Bolts*. Jan 2006.

[18] Uri Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. 2006.

[19] G Dilanji, J Langebrake, P Deleenheer, and S.J Hagen. Quorum activation at a distance: spatiotemporal patterns of gene regulation from diffusion of an autoinducer signal. *Bulletin of the American Physical Society*, 57, 2012.

[20] H.B Kaplan and EP Greenberg. Diffusion of autoinducer is involved in regulation of the vibrio fischeri luminescence system. *J Bacteriol*, 163(3):1210–1214, 1985.

[21] A.G Marr. Growth rate of escherichia coli. *Microbiological reviews*, 55(2):316–333, 1991.

[22] Francesca Ceroni, Simone Furini, Emanuele Giordano, and Silvio Cavalcanti. Rational design of modular circuits for gene transcription: A test of the bottom-up approach. *Journal of Biological Engineering*, 4(1):14, Nov 2010.

[23] Yu Tanouchi, Dennis Tu, Jungsang Kim, and Lingchong You. Noise reduction by diffusional dissipation in a minimal quorum sensing motif. *PLoS computational biology*, 4(8):e1000167, Aug 2008.

[24] S Basu, Y Gerchman, C.H Collins, F.H Arnold, and R Weiss. A synthetic multicellular system for programmed pattern formation. *Nature*, 434(7037):1130–1134, 2005.

[25] D Kolodrubetz and R Schleif. Identification of arac protein on two-dimensional gels, its¡ i¿ in vivo¡/i¿ instability and normal level. *Journal of Molecular Biology*, 149(1):133–139, 1981.

[26] ML Urbanowski, CP Lostroh, and EP Greenberg. Reversible acyl-homoserine lactone binding to purified vibrio fischeri luxr protein. *J Bacteriol*, 186(3):631–637, 2004.

[27] S Oehler, E.R Eismann, H Krämer, and B Müller-Hill. The three operators of the lac operon cooperate in repression. *The EMBO journal*, 9(4):973, 1990.

[28] J Elf, G.-W Li, and X. S Xie. Probing transcription factor dynamics at the single-molecule level in a living cell. *Science*, 316(5828):1191–1194, May 2007.

[29] N Qin, S. M Callahan, P. V Dunlap, and A. M Stevens. Analysis of luxr regulon gene expression during quorum sensing in vibrio fischeri. *J Bacteriol*, 189(11):4127–4134, Jun 2007.

[30] A Kamionka. Two mutations in the tetracycline repressor change the inducer anhydrotetracycline to a corepressor. *Nucleic Acids Research*, 32(2):842–847, Jan 2004.

[31] S Jang, K Oishi, R Egbert, and E Klavins. Specification and simulation of synthetic multi-celled behaviors. *ACS Synthetic Biology*, Jan 2012.

[32] A.M Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72, 1952.

[33] AJ Koch and H Meinhardt. Biological pattern formation: from basic mechanisms to complex structures. *Reviews of Modern Physics*, 66(4):1481–1507, 1994.

[34] E Ben-Jacob. Bacterial self–organization: co–enhancement of complexification and adaptability in a dynamic environment. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1807):1283–1312, 2003.

[35] Wright WW Vanderkooi JM and Erecinska M. Nitric oxide diffusion coefficients in solutions, proteins and membranes determined by phosphorescence. *Biochimica et biophysica acta*, 1994.

[36] S Spiro. Nitric oxide-sensing mechanisms in escherichia coli. *Biochemical Society Transactions*, 34:200–202, 2006.

[37] Yoshihiro Yamaguchi and Masayori Inouye. Regulation of growth and death in escherichia coli by toxin–antitoxin systems. *Nature Reviews Microbiology*, 9(11):779–790, Sep 2011.

[38] Eric Klavins. `http://depts.washington.edu/soslab/gro/`.