Master's Thesis

Logic gene network design: a CAD tool based on modularity and standardization

Thesis Committee:
Prof.dr.ir. M.J.T. Reinders
Dr.ir. D. Bellomo
Dr.ir. A. Abate
Ir. I.E. Nikerel
Dr.ir. D. de Ridder
Ir. M. Hulsman

Author	**Bastiaan A. van den Berg**
Email	b.a.vandenberg@gmail.com
Student number	1041517
Thesis supervisors	Prof.dr.ir. M.J.T. Reinders Dr.ir. D. Bellomo
Date	October 7, 2009

Information and Communication Theory Group [I,C)^T

TU Delft

Faculty of Electrical Engineering, Mathematics and Computer Science

Delft University of Technology

Preface

This document is the result of the research that has been done on the design of logic gene networks, which is a topic in the field of synthetic biology. It consists of a paper and a work document. The paper is the main part; it provides an introduction, describes the used approach, the used methods, the implementation, and the results. More detailed information is given in the work document.

The research has been done in the Delft Bioinformatics Lab at Delf University of Technology, and has been supervised by dr.ir. D. Bellomo and prof.dr.ir. M.J.T. Reinders. As a substitute of the literature study, I participated in the Delft UT 2008 iGEM[1] project, which introduced me to the field of synthetic biology. iGEM started in July 2008 and afterwards I started my thesis project in December 2008. The thesis project will be defended on October 7, 2009.

Acknowledgements

During my thesis project and the rest of my study in Delft I got a lot of help and support of many people, for which I am grateful. I cannot name them all, but I would like to mention a few.

First of all Domenico and Marcel, my supervisors. Their input was indispensable, and although it was a heavy load sometimes, the two-weekly-meetings kept me sharp and focused.

Also lots of thanks to my parents, Aad and Ans, who have always supported me and who have helped me with some tough decisions. My sisters and brothers: Susanne, Evelien, Lennard and Marijn, for their interest and appreciation. My little nephew Iddo and niece Jara, for giving me lots of joy. Wenda, for the great care during the last months of the project. My friends, for still being my friends, after all this years of me, spending too much time on my study. And finally, my DGW331 house-mates and korfball-buddies from D.S.K.V. Paal Centraal, for providing the necessary fun and relaxation.

[1] `http://2008.igem.org/Team:TUDelft`

Vol. 00 no. 00 2009
Pages 1–11

Logic gene network design: a CAD tool based on modularity and standardization

Bastiaan A. van den Berg

ABSTRACT

Motivation: Synthetic biology aims at building biological systems for useful purposes. Relatively simple gene networks have been engineered, but the design process is limited. Many papers advocate the use of engineering concepts like standardization and modular design to simplify the design process and enable the design of more complex systems. Currently, there are no tools available that implement both concepts in a practical way.

Results: We have developed a software tool to show how standardization and modular design can be used for the design of logic gene networks. We introduce gene network templates to be able to use modular design in a practical way and use a standard model to simplify the design process and enable reuse of parameters. We have designed three logic gate templates and used them to build two logic gene networks: a demultiplexer and a D-latch. Our software tool was used to turn the templates into devices and to evaluate the performance of the devices. The results show that the devices are evaluated correctly. Furthermore, the results show that for the design of a gene network our method can be used to indicate which biological parts are preferred at what location in the network.

1 INTRODUCTION

The main goal of synthetic biology is to engineer biological systems with defined function. Synthetic versions of natural systems are build to get more insight into their functioning (Sprinzak and Elowitz, 2005). Novel systems are build in order to obtain new functionalities that can be used for industrial applications, such as energy production, waste material degradation, and material construction (Endy, 2005); and medical applications, such as smart disease treatment, invasion of cancer cells (Anderson et al., 2006), and tissue engineering (Basu et al., 2005).

At a low level, synthetic biology constructs or modifies biological building blocks: DNA, RNA, proteins, and metabolites. For example, protein engineering enables the construction of non-natural proteins with different functional domains. At a higher level, synthetic biology uses the physical building blocks to create biological systems, such as gene networks, signalling pathways and metabolic networks (Heinemann and Panke, 2006; Andrianantoandro et al., 2006).

This paper focuses on the design of synthetic gene networks that implement a logic function, i.e. logic gene networks. This enables the introduction of intelligence to a cell using the principles of boolean logic, which could be useful for biotechnology applications.

Gene networks have been constructed, but they are relatively simple and the design process is limited. Elowitz and Leibler (2000) and Gardner et al. (2000) were the first to design a genetic oscillator and toggle switch respectively. More recently Stricker et al. (2008) engineered a fast, robust, and tunable oscillator and Friedland et al. (2009) constructed a gene network that can count up to three induction events.

In most cases, the design of a gene network is made by intuition only. A model is used to show that the design provides the desired qualitative behaviour. After construction, the system is fine-tuned *in vivo* until it performs well. Afterwards, the model is fitted to the measured data, which is then used for further *in silico* analysis.

With this approach, the main effort is spend on the time consuming and costly lab work. It is desirable to put more emphasis on the less expensive *in silico* design process. A reason why the design process is still limited is that we are not yet able to accurately predict the *in vivo* behaviour of a designed system. This is partly caused by the fact that different gene expression models, each with their own dedicated parameters, are used, complicating their use as building blocks for the design of other more complex systems.

We propose to make use of standard models for which the parameters are stored separately in a database. This way, parameters can be stored and reused for the design of other systems. Furthermore, it enables automatic model building, which greatly simplifies the design process. Finally, having a standard model will promote optimization of the model by simulations as to get more accurate predictions.

Another limitation in the current situation of gene network design is the complexity of the designed systems. A common engineering approach to cope with increasing complexity is the use of modular design; building a complex system out of smaller subsystems (modules). For example, an ALU, which is part of a computer processor, consists of digital circuits that can perform arithmetic operations. These digital circuits consists of logic gates, and the logic gates in turn consists of simple transistors. This way, one can go through various levels of complexity from simple transistors to a complex ALU.

Modular design within synthetic biology is still in its infancy. Massachusetts Institute of Technology has founded a registry of Standard Biological Parts: short DNA sequences with defined function[1]. Genetic parts, such as promoters, ribosome binding sites (RBS), and protein coding parts, are the most basic building blocks. The parts meet the BioBrick^{TM}standard[2], so that they can be ligated using a standardized ligation protocol (Shetty et al., 2008). The ligated parts form a device: a DNA sequence that encodes for some cellular system, for example an inverter (when the concentration of molecule A is high, then the concentration of molecule B is low, and visa versa). Finally, the devices can be used as building blocks of a more complex system (Endy, 2005). The parts registry provides a simple and standardized way to *construct* DNA sequences and build

[1] `http://partsregistry.org`

[2] `http://biobricks.org`

a more complex system *in vivo*, however, it is a registry and there are no tools that enables the *design* of complex systems. We advocate that, to accomplish this, modularity and standardization should also be incorporated into the design process.

We aim at the design of logic gene networks. In this case, boolean logic provides the theory to get from a set of simple logic operations to a complex logic function. Our goal is not to provide the theoretical basis to get complex functionality, but to show how it can be implemented using modular biological components. So what we need is a set of biological logic gates that implement a simple logic operation, in order to build a logic gene network that implements a complex logic function.

Biological logic gates have been constructed (Anderson et al., 2007; Sayut et al., 2009). It has been shown that the logic gates function correctly on their own, but it is unknown if they retain the correct function when connected to other gates, which is needed to build logic networks. For example, it could be that the number and type of gates to which it is connected affect its performance.

Del Vecchio et al. (2008) proposed a solution inspired by the field of electrical engineering to ensure that a module preserves the same function independent of the modules it is connected to. In that case, the same logic gate can be used anywhere in any logic circuit. Although this would provide modular logic gates, it will probably result in a design that is not optimally energy efficient, because it uses mechanisms that will require energy to ensure modularity. Since energy is a very costly product for a cell, it is questionable if this approach is feasible.

Instead of using energy consuming mechanisms to ensure that a module retains its function in any situation, we propose to make use of a tunable module that can be adjusted to its situation. To do so, we make use of gene network templates; abstract representations of a gene network that hide the specific information about the bioparts used to build it (Fig. 1B). The templates are modular, so that logic gate templates can be used to build a logic network template. Our software tool can turn a gene network template into devices using genetic parts from a database. This enables tuning of the overall network by varying the genetic parts.

The use of gene network templates also provides a more practical use of modularity compared to other software tools. *ProMoT* (Marchisio and Stelling, 2008; Mirschel et al., 2009), *Antimony* (Smith et al., 2009), and *Tinkercell* (Chandran et al., 2009) are all software tools that can be used to define modular models. For example, one could define a model for the OR gate in Fig. 1A, which can then be used to build logic networks. But when, for a specific network, one needs an OR gate with different transcription factors (TF) as input signals[3], another version of the same OR gate has to be defined. Having only a limited number of TFs as signal carriers already results in many different possible OR gate versions. With our approach, a user only needs to define the template in Fig. 1B, which can be turned into any possible version (instance of a template) by the software tool.

[3] In electrical engineering, the use of only one signal carrier (current) is sufficient, because each signal is directed through a wire, and thereby the different signals are physically separated. For gene networks we use TFs as signal carriers, which freely float through the cell and cannot be directed. Therefore a different TF is needed for each signal (wire) in the network.

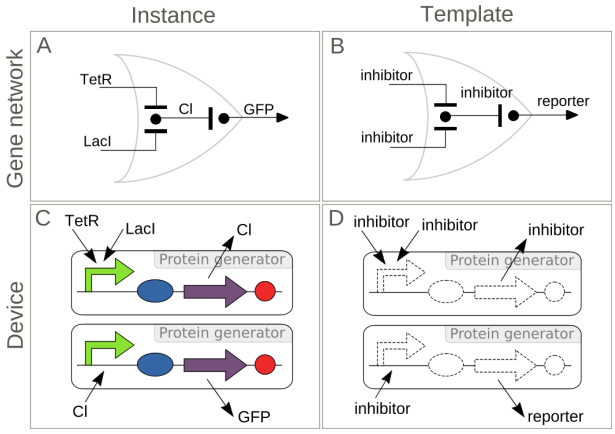


Fig. 1. Templates and instances - *A)* A logic OR gate gene network. A gene network is a graph in which the vertices are protein generators and the edges proteins. An outgoing edge is an expressed protein and an ingoing edge is a transcription factor (inhibitor or activator) that binds to the promoter of the protein generator. *C)* A device, consisting of genetic parts, that implements the gene network from *A*. The device consists of two protein generators, corresponding to the vertices in *A*, and each protein generator consists of four genetic parts; a promoter, a ribosome binding site, a protein coding part, and a terminator. *B, D)* Templates define the general structure of a gene network but not the used biological parts to implement it. *A* and *C* are a possible instance of the given template.

2 APPROACH

An overview of the design approach as we have implemented it in a software tool is shown in Fig. 2. A user has to define a gene network template, which can be done in a modular way. For the design of a logic gene network one can simply connect templates of logic gates (Fig. 2A). To be able to evaluate the performance of a logic gene network, the user also has to define one or more input signals and desired output signals. An input signal, provided as a binary timing diagram, determines the concentration of an input molecule during simulation (Fig. 2B). A desired output signal, also defined as a binary timing diagram, is compared to the simulation data for evaluation (Fig. 2C). For a gene network template, many different gene network instances can be build using bioparts from the biopart database (Fig. 2D). A gene network instance is build out of genetic parts and is therefore called a device (Fig. 2E). Each device can be turned into a model (Fig. 2F) which can be used for stochastic simulation. The simulation result (Fig. 2G) is compared to the user-defined desired output to calculate an evaluation score for a device (Fig. 2H). The evaluation score can be used to compare the performance of the different devices for a given template.

Protein generator To simplify the data transformations, a standard building block has been defined, which we call a protein generator. A protein generator is a DNA sequence able to express a protein, possibly regulated by one or more TFs. The TFs serve as interface between the protein generators and enable their (inter)connection. There are three different representations of a protein generator: as part of a gene network, as a device, and as a model.

Fig. 1A shows the use of a protein generator in a gene network. The gene network is a graph in which each vertex is a protein generator and the edges are proteins. An output edge is the protein

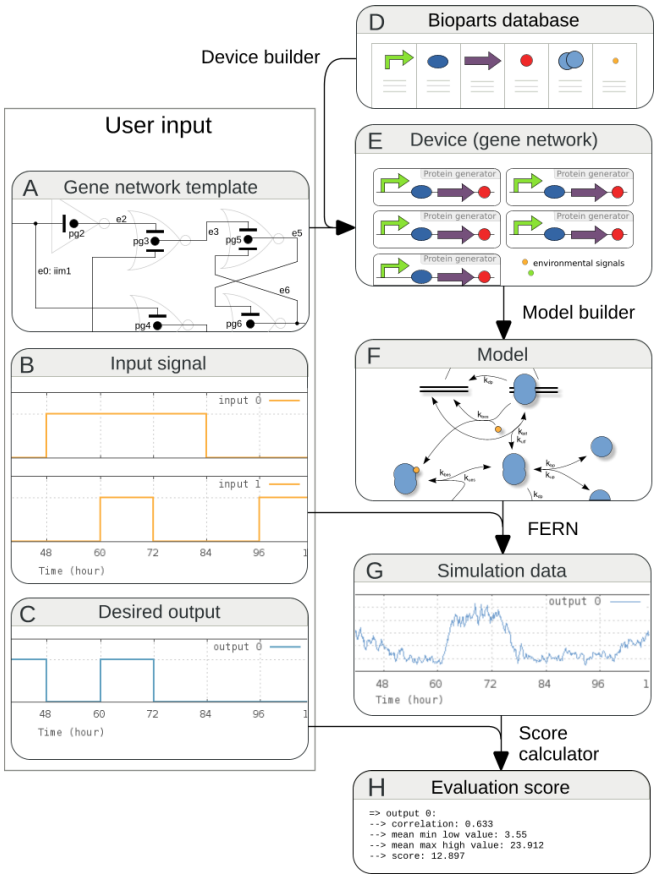


Fig. 2. Software overview - The boxes represent stored data and the arrows are data transformations. *A)* A user-defined gene network template that defines the topology of the gene network. Templates can be build in a modular way; the figure shows part of a template that is build through connecting logic gate templates. Furthermore, the user has to assign an environmental signal to each of the inputs and a reporter to each of the outputs. *B)* A binary timing diagram for each input signal, which determines the molecule count of an environmental signal during simulation. *C)* A binary timing diagram that defines the desired simulation result of an output signal. *D)* A database with bioparts that can be used to build a gene network. *E)* A device that consists of genetic parts from the biopart database and which implements the gene network in *A*. The device builder provides the functionality to turn the gene network template into devices that implement the template using bioparts from the bioparts database. *F)* A model for the device, which is build by the model builder. *G) FERN* is used to run an exact stochastic simulation algorithm on the model. The molecule count over time for a reporter is given as result, which is an average over 20 simulation runs. *H)* A device evaluation score, as a result of comparing the simulation data with the desired output.

(TF or reporter) expressed by the protein generator. The input edges are the TFs that bind to the promoter of the protein generator, and thereby regulate the expression. For example, in Fig. 1A, the left vertex takes two inhibitors, *TetR* and *LacI*, as input signal and produces one inhibitor, *Cl*, as output signal.

A device representation of a protein generator is shown in Fig. 1C. A protein generator device consists of four genetic parts: a promoter, a ribosome binding site (RBS), a protein coding part and a terminator. The promoter determines what TFs are taken as input signal and the protein coding part determines what TF is expressed as output signal.

Finally, a protein generator can be represented as a model, which captures the behaviour of the protein generator and which can be used for stochastic simulation. We return to the model in more detail in Section 3.2.

Signals We use three different signal carriers: TFs, environmental signals, and reporters. TFs are used as *internal signals* that connect protein generators. Environmental signals, such as small molecules, light, or temperature, serve as interface to the environment of a cell and are used as *input signals* of a gene network. An environmental signal can inhibit or activate a TF. For example, an inhibiting small molecule can bind to a TF, causing a conformational change that disables the TF to bind to a promoter. Finally, reporters are used as *output signals* of a gene network.

Bioparts database We used an artificial biopart database, because there was no database with kinetic constants available that we could use for our method. The database contained four types of genetic parts that can be used to build protein generator devices: promoter, RBS, protein coding part, terminator. Furthermore, the database contained the three signal carriers: TF, reporter, environmental signal.

Gene network template To avoid the definition of many versions for each logic gate, we make use of gene network templates; gene network topologies that specify how protein generators are connected and what type of TF (inhibitor or activator) is used for each signal. The gene network shown in Fig. 1B defines that there are two protein generators: one with two inhibitors as input and an inhibitor as output, and one with an inhibitor as input and a reporter as output. Fig. 1A shows a possible instance for this template. Notice that many possible instances can be build for one template, depending on the available bioparts.

We define a gene network template as a graph; a set of vertices and a set of edges. An edge has a type (inhibitor or activator) and can have multiple source and destination vertices. Furthermore, an environmental signal can be assigned as an inhibitor or activator of an edge. A vertex can have multiple inputs (the binding TFs) and has always one output (the protein it expresses).

Gene network templates are modular. For example, a NOR gate and a NOT gate can be connected to form an OR gate. This OR gate can in turn be used in more complex logic networks. For each template, one or more edges must be assigned as input edge and one or more as output edge. These edges provide the interface that enables the connection to other templates.

User input For the design of a logic gene network, the user has to define a gene network template (Fig. 2A) and it has to assign an environmental signal to each input signal and a reporter to each output signal.

Furthermore, the user has to specify a binary timing diagram for each input signal and each desired output signal. The value of a signal can be either 0, denoting a low signal, or 1, denoting a high signal. The timing diagram is divided in equal length time steps for which the length has to be defined by the user. Switching of the signal only happens at the borders of the time steps.

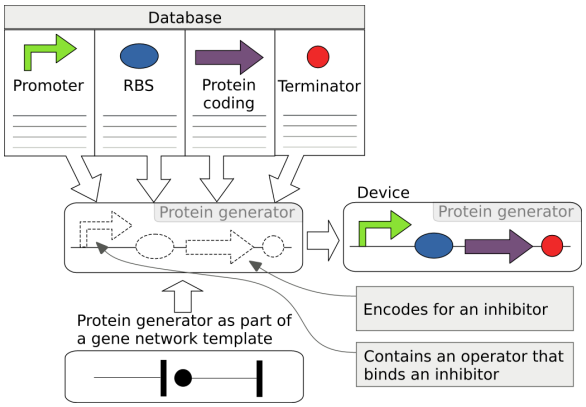


Fig. 3. From template to instance - The transformation of a protein generator that is part of a gene network template to a protein generator device happens through the assignment of genetic parts from a biopart database to the protein generator device. The gene network template puts restrictions on the parts that may be used. In this case the gene network template puts restrictions on the choice of the promoter and the protein coding part. The promoter must bind one inhibitor and the protein coding part must encode for an inhibitor. Since there can be multiple promoters that bind one inhibitor and multiple protein coding parts that encode for an inhibitor, multiple devices can be made for one gene network template.

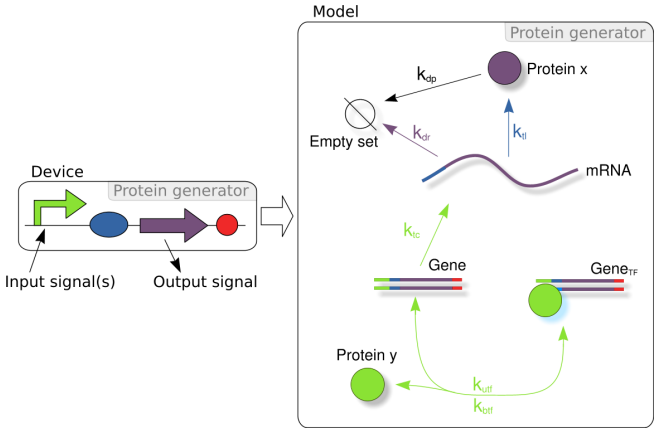


Fig. 4. From device to model - The genetic parts that make up the protein generator device store the data that is needed to turn the device into a model. A standard model for a protein generator is shown, which consists of molecular species (*gene*, *mRNA*, ...) and reactions (the arrows). The kinetic constants (k) determine the rate at which the reactions occur. More details on the model are given in Section 3.2. The colours indicate which genetic part stores what data. The promoter (green) stores the transcription rate (k_{tc}) and the binding and unbinding rates of the TF(s) that can bind to it (k_{btf}, k_{utf}). Furthermore it stores a pointer to the binding TF(s) (protein y in this case). The RBS (blue) stores the translation rate k_{tl}. The protein coding part (purple) stores the mRNA degradation rate and also stores a pointer to the protein it encodes for (protein x in this case). Finally, the terminator (red) does not determine anything in our model and therefore does not store any parameter. Next to the genetic parts, the biopart database also contains proteins, that also store some kinetic constants. For example, the protein degradation rate k_{dp}, which is therefore not coloured in this figure. But since the protein coding part stores a pointer to protein x, the model builder can still fetch this kinetic constant from the database.

The input signals (Fig. 2B) determine the molecule counts of the environmental signals during simulation. When an input signal switches from high to low, the molecule count of the corresponding environmental signal is set to zero. When it switches from low to high, the molecule count is set to a high number.

A desired output signal (Fig. 2C) specifies what the simulated data of the corresponding reporter should look like, which is used to evaluate the performance of a device. This evaluation is done through comparison of the simulation data (Fig. 2G) to the desired output signal. The goal is to have simulation data that resembles the desired output.

Device builder To get from a gene network template to a possible gene network instance, finally represented as a device (so going from Fig. 1B to 1C), the device builder assigns genetic parts from the bioparts database to each of the protein generators (vertices). The gene network template puts restrictions on the bioparts that can be used to build the device. For example, the gene network template in Fig. 3 determines that the promoter must bind one inhibitor and the protein coding part must encode for an inhibitor. To prevent crosstalk, TFs may only be used once within a network, putting further restrictions on the parts that can be used.

Model builder The data transformation from a device to a model is also done per protein generator. Each protein generator is translated into a standard model (Fig. 4). For each type of part in the biopart database, some kinetic constants are stored. For example, all RBS parts store a translation rate (k_{tl}), because the RBS determines this kinetic constant. Fig. 4 shows which kinetic constants are stored for what genetic parts. The model builder fetches the kinetic constants from the biopart database and uses them for the model-building.

The models of the different protein generators are connected as soon as the product of a translation reaction is (possibly after dimerization) also a reactant of a TF-DNA binding reaction. This way, the TF expressed by one protein generator inhibits or activates the expression of another protein generator. Combining the reactions of the protein generator models will therefore automatically form a gene network model.

Stochastic simulation The resulting model can be used for stochastic simulation (Gillespie et al., 1977). We use stochastic simulation because gene expression is a stochastic process and the stochasticity can be of influence on the behaviour of a gene network (Kaern et al., 2005). During simulation, the molecule counts of the environmental signals are determined by the user-defined input signals. When the input signal is high (1), the molecule count of the corresponding environmental signal is set to a high number, simulating an induction event. When the signal is low (0), the molecule count is set to zero.

Score calculator The score calculator compares the simulation data to the desired output in order to evaluate the performance of the device. The goal is to have an output signal that resembles the user-defined desired output. Furthermore, since we are building logic gene networks, we want to have a maximal difference between a low signal and a high signal, in order to distinguish a logic 0 from a logic 1.

3 METHODS

3.1 Stochastic Simulation

A deterministic approach is commonly used to predict the evolution of the state of a chemical system, in which the state is defined as the molecule counts of the molecular species within the system. The state changes due to chemical reactions that occur stochastically and the evolution of the system is therefore a stochastic process. A deterministic approach can accurately predict the behaviour of systems in which the number of molecules per species is high, but as soon as the molecule count of at least one of the molecular species is low, stochasticity can have a major influence and can therefore not be neglected anymore (Gillespie, 2007; Cai and Wang, 2007).

Gene expression is an inherently stochastic process, e.g. TFs randomly bind to the DNA, proteins randomly form dimers, and ribosomes randomly bind to mRNAs. In this system, the molecule count of a gene is very low and stochasticity should therefore be taken into account. Kaern et al. (2005) also showed that stochasticity can have a pronounced effect on gene expression.

We used Gillespie's exact stochastic simulation algorithm (SSA) (Gillespie et al., 1977) for the simulation of the gene networks. Although this algorithm assumes a well stirred, thermally equilibrated chemical system with a constant volume, which is not true for a cell, it has been shown that it can accurately predict gene expression (Hooshangi et al., 2005). A detailed description of stochastic simulation and an overview of available SSAs can be found in Gillespie (2007); Cai and Wang (2007).

3.2 Model

We use a stochastic model, which means that we use only two types of elemental reactions: unimolecular, with only one reactant, and bimolecular, with two reactants. Our model is based on the one proposed by Hooshangi et al. (2005), which showed that the model can accurately predict regulated gene expression for a transcriptional cascade. In contrast to Hooshangi's model, in which TFs only form dimers, we also use monomeric TFs and TFs that form tetramers.

A model is defined as a set of molecular species and a set of reactions. A reaction rate is defined for each reaction that determines the rate at which a reaction occurs. We used only five types of molecular species: *gene*, *mRNA*, *protein*, *environmental signal* (*es*), and *empty set* ($\emptyset$). Together, some of them can also form complexes. For example, an environmental signal can bind to a TF (which is a protein) forming a TF_{es} complex.

Basic gene expression is modelled with four reactions:

$$\text{gene} \xrightarrow{k_{tc}} \text{gene} + \text{mRNA}, \qquad \text{(transcription)}$$

$$\text{mRNA} \xrightarrow{k_{tl}} \text{mRNA} + \text{protein}, \qquad \text{(translation)}$$

$$\text{mRNA} \xrightarrow{k_{dr}} \emptyset, \qquad \text{(mRNA degradation)}$$

$$\text{protein} \xrightarrow{k_{dp}} \emptyset. \qquad \text{(protein degradation)}$$

The same model was also used to investigate stochasticity of gene expression (Shahrezaei and Swain, 2008; Maheshri and O'Shea, 2007; Kaern et al., 2005; Ozbudak et al., 2002). Regulation of the gene expression happens through the binding and unbinding of a TF to a gene:

$$\text{gene} + \text{TF} \underset{k_{utf}}{\overset{k_{btf}}{\longleftrightarrow}} \text{gene}_{TF}$$

Since there is no transcription reaction for $\textit{gene}_{TF}$, transcription is being repressed when a TF is bound to the gene. As soon as the TF dissociates from the gene, the system contains a *gene* and a transcription reaction can occur. The gene is not being repressed anymore. The model allows for dimer and tetramer formation using the following reactions:

$$\text{monomer} + \text{monomer} \underset{k_{up}}{\overset{k_{bp}}{\longleftrightarrow}} \text{dimer}$$

An environmental signal can bind to a TF and thereby activate or disable it. For example, a small molecule can bind to a TF causing a conformational change so that the TF cannot bind to the gene anymore. This is modeled by the following reaction:

$$\text{TF} + \text{es} \underset{k_{ues}}{\overset{k_{bes}}{\longleftrightarrow}} \text{TF}_{es}$$

When an environmental signal binding reaction occurs, a *TF* molecule disappears and a $\textit{TF}_{es}$ molecule appears. Since there is no reaction that binds a $\textit{TF}_{es}$ to a *gene*, the TF is inhibited by the environmental signal.

An inhibiting environment signal can also bind to a TF that is already bound to a gene, forcing dissociation of the TF from the gene:

$$\text{gene}_{TF} + \text{es} \xrightarrow{k_{bes}} \text{gene} + \text{TF}_{es}$$

Finally, there are two degradation reactions to make sure that TFs that are bound to a gene or an environmental signal also degrade:

$$\text{TF}_{es} \xrightarrow{k_{dp}} \emptyset + \text{es},$$

$$\text{gene}_{TF} \xrightarrow{k_{dp}} \emptyset + \text{gene}.$$

In both reactions, the TF degrades while the molecule to which it is bound is preserved. The gene is part of the DNA and does not degrade. The environmental signal is the binary input to a gene network and the molecule count is therefore either high or zero. A high signal can be seen as an induction event and during induction we want to remain a high molecule count. Therefore we do not let the environmental signal degrade.

Fig. 5 shows a model for regulated expression of a protein generator that produces a protein (green) and is regulated by a TF dimer (blue). The environmental signal (orange) can bind to the TF and thereby inhibit it. Variations of this model are also possible in which: A TF can either be a monomer, a dimer, or a tetramer; The gene can bind zero, one, or two TFs; The TF can be an inhibitor or an activator; the environmental signal can be an inhibitor or an activator.

3.3 Kinetic constants

The model uses ten different kinetic constants: transcription rate (k_{tc}), translation rate (k_{tl}), mRNA degradation rate (k_{dr}), protein degradation rate (k_{dp}), *TF-gene* binding and unbinding rate ($k_{btf},\ k_{utf}$), *protein-protein* binding and unbinding rate ($k_{bp},\ k_{up}$), and *TF-es* binding and unbinding rate ($k_{bes},\ k_{ues}$). We have defined a range for each of the parameters, which we have used to build the biopart database. Most of the parameters are based on Hooshangi et al. (2005). The ranges of the mRNA and protein degradation rates are based on Bernstein et al. (2002) and Grilly et al. (2007) respectively. The range of the protein-protein binding rate is based on Schlosshauer and Baker (2004). A table with the parameter ranges is given in the supplement.

3.4 Biopart database

The biopart database contains four types of genetic parts: promoter, RBS, protein coding part, and terminator, and two types of molecules: protein and environmental signal. For each part, a number of kinetic constants and relations to other parts are stored. A promoter stores a transcription rate, a pointer to each TF (protein) that can bind to it, and a binding and unbinding rate for each of the TFs that can bind to it. An RBS stores a translation rate. A protein coding part stores a mRNA degradation rate and a pointer to the protein for which it encodes. A terminator stores nothing. A protein stores a degradation rate. In case of a dimer or tetramer protein, it stores a protein-protein binding and unbinding rate and a pointer to the sub-proteins. When the protein binds an environmental signal, it also stores a corresponding binding and unbinding rate and a pointer to the environmental signal.

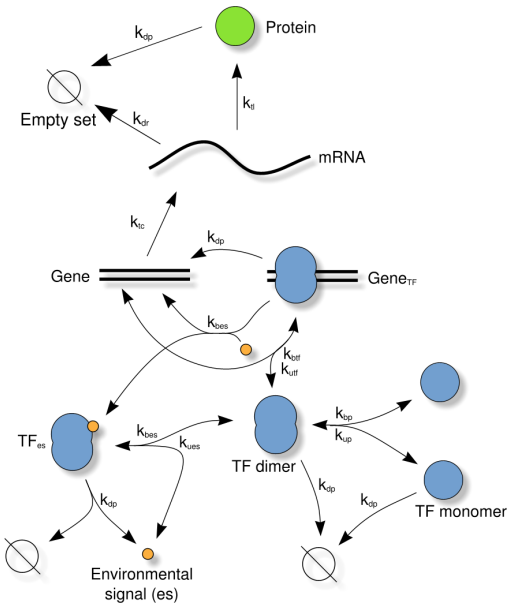


Fig. 5. Protein generator model - A stochastic model for a protein generator in which the gene, proteins and environmental signal are the molecular species and the arrows are reactions. Each reaction turns one or two reactants into one or more products. During stochastic simulation, a molecule count for each of the molecular species is stored, which is the state of the system. The molecule counts change when a reaction occurs. For example, when a TF binding reaction occurs, the molecule counts of *gene* and *TF dimer* decrease by one and the molecule count of $gene_{TF}$ increases by one. This way, the reactions can change the state of the system. The rate at which a reaction occurs depends on the reaction rate (k) and the current state of the system. Gene expression is modelled by one transcription reaction, from *gene* to *mRNA*, and one translation reaction, from *mRNA to protein*. Both proteins and mRNAs can degrade, which is modeled by a reaction that turns the molecules into a *empty set* molecule. TF monomers can form dimers and a dimer TF can bind to the gene. Since there is no transcription reaction from $gene_{TF}$ to *mRNA*, the bound TF inhibits gene expression. An environmental signal can bind to a TF dimer and thereby inhibit the TF, since there is no reaction that binds TF_{es} to a gene.

3.5 Score calculation

The performance of a device is evaluated through comparison of the simulated outputs to the desired outputs. The score is based on three measures: the correlation between the simulated and desired outputs; the high signal concentrations of the simulated outputs; and the low signal concentrations of the simulated outputs. The objective for a well performing device is maximization of the correlation, maximization of the high signal concentration, and minimization of the low signal concentration.

Fig. 6 shows an example of a desired output and a simulation result. The plots are divided into time steps, five in this case. In each of the time steps the value of a desired output can be either 0 (low) or 1 (high). The first time step is not taken into account for the score calculation, this time is needed to get to an initial steady state.

Three measures are calculated for each of the n output signals: a correlation ρ_i, a low value l_i, and a high value h_i, $i = 0, \ldots, n$. We use the sample correlation coefficient to estimate the correlation of the simulated and desired output. The low value of output 0 in Fig. 6 is the average of the orange circled values. The orange circles depict the minimum values in the time steps where the desired output is low. Similarly, the high value of output 0 in Fig. 6 is the average of the green circled values. The green circled

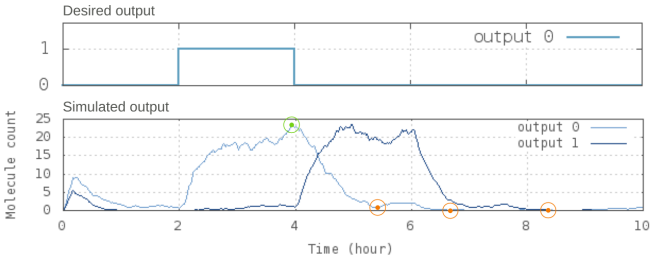


Fig. 6. Score calculation - The plots are divided in five time steps. The first time step is needed to get to an initial steady state and is not taken into account for the score calculation. For *output 0*: ρ_0 is the correlation between the desired and the simulated output; l_0 is the average of the orange circled values, which are the minimum values in the time steps where the desired output is low; and h_0 is the green circled value, which is the average of the maximum values in the time steps where the desired output is high.

value is the maximum value of the simulation output in the time step where the desired output is high. In this case, the signal has only one high output, which is therefore also the average.

The correlation ρ_i, the low value l_i, and the high value h_i for each of the n output signals are combined into one score as follows:

$$score = (\prod_{i=0}^{n} c_i) \times ((\prod_{i=0}^{n} h_i) - (\sum_{i=0}^{n} l_i))$$

The correlations scores, that range from -1 to 1, are multiplied. This way the score drops as soon as one of the output signals has a low correlation. The high values are also multiplied, to make sure that a device scores better when the high value of all outputs is high. For example, for a device with two outputs, having 6 as high value for both outputs is prefered over high values 2 and 10. Whereas with addition they would provide the same result ($6 + 6 = 2 + 10 = 12$), with multiplication the first version scores better ($6 \times 6 = 36,\ 2 \times 10 = 20$). In case of multiplication of the low values, a zero would bring the total to zero, even when another low values is high. Therefore we use addition for the low values instead of multiplication. Since we want to maximize the high value and minimize the low value, we subtract the low value from the high value.

4 IMPLEMENTATION

The software is written in *Java* using the *FERN* software package for stochastic simulation (Erhard et al., 2008), the *jdom*[4] package for XML parsing, and the *colt*[5] package for basic mathematics.

4.1 *gene_network.template* package

The *gene_network.template* package provides a data structure for a gene network template. The *GeneNetworkTemplate* class stores a network as a list of *Vertice* objects and a list of *Edge* objects that refer to each other. It can read a user defined gene network from an XML file (Listing 1). This file contains a list of vertex elements and a list of edge elements. For each edge, the source and/or destination vertices have to be defined, in order to interconnect the vertices. Finally, the user has to define which of the edge's are inputs and outputs of the network.

Gene networks can be defined in a modular way, e.g. a NOR gate template can be used as sub-network within a demultiplexer

[4] `http://www.jdom.org/`

[5] `http://acs.lbl.gov/~hoschek/colt/index.html`

```
<?xml version="1.0" encoding="UTF-8"?>
<template_network name="NORii">

  <vertice id="v0" />

  <edge id="e0" type="-">
    <dest vertice_id="v0" />
  </edge>
  <edge id="e1" type="-">
    <dest vertice_id="v0" />
  </edge>
  <edge id="e2">
    <src vertice_id="v0" />
  </edge>

  <input id="in0" edge_id="e0" />
  <input id="in1" edge_id="e1" />

  <output id="out0" edge_id="e2" />

</template_network>
```

Listing 1. An XML file that defines the gene network template of a NOR gate. The NOR gate is used multiple times a sub-network of the demultiplexer (Listing 2 and Fig. 7) and the D-latch (Fig. 10).

```
<?xml version="1.0" encoding="UTF-8"?>
<template_network name="Demultiplexer">

  <subnetwork id="NOTe0" name="NOTe" />
  <subnetwork id="NOTe1" name="NOTe" />
  <subnetwork id="NOTi" name="NOTi" />
  <subnetwork id="NORii0" name="NORii" />
  <subnetwork id="NORii1" name="NORii" />

  <connection_edge id="e0">
    <src network="NOTe0" output_id="out0" />
    <dest network="NORii0" input_id="in0" />
    <dest network="NOTi" input_id="in0" />
  </connection_edge>
  <connection_edge id="e1">
    <src network="NOTe1" output_id="out0" />
    <dest network="NORii0" input_id="in1" />
    <dest network="NORii1" input_id="in1" />
  </connection_edge>
  <connection_edge id="e2">
    <src network="NOTi" output_id="out0" />
    <dest network="NORii1" input_id="in0" />
  </connection_edge>

  <input id="in0" edge_id="e0" />
  <input id="in1" edge_id="e1" />

  <output id="out0" network="NORii1" output_id="out0" />
  <output id="out1" network="NORii0" output_id="out0" />

</template_network>
```

Listing 2. The XML file in which the gene network template of the demultiplexer from Fig. 7 is defined.

template (Listing 2). In this case, the XML file contains a list of sub-network elements and a list of connection-edge's that define which outputs are connected to what inputs.

4.2 *biopart* package

Because of the ease of implementation, the database is implemented as a directory structure with XML files. A part is stored as an XML file and parts of the same type are grouped in directories. The database contains four different types of genetic parts; promoters, RBSs, protein coding parts, and terminators. Next to the genetic parts, it also stores proteins (TFs and reporters) and environmental signals. The data stored for a part are kinetic constants and relation to other parts.

The *biopart* package provide classes for each of the bioparts and provides the functionality to interact with the database. The *BioPartDatabesBuilder* class provides a convenient way to (re)build an artificial biopart database.

4.3 *gene_network* package

The *gene_network* package provides the functionality to build devices for a given gene network template. A *GeneNetworkBuilder* class takes a *GeneNetworkTemplate* object as input an translates it into possible gene network instances (in the form of *Device* objects). It provides two different methods; *getRandomGeneNetwork* and *getAllGeneNetworks*. The first returns a random gene network instance for a given template and the latter returns all possible gene networks for a given template. Because the number of possible gene networks for a given template can grow very large, the *getAllGeneNetworks* method needs to be used careful.

4.4 *model* package

The *model* package implements the model-building functionality. There is an abstract *Model* class that takes a device as input which it can translate into a model. The model is stored as a list of *Species* objects and a list of *Reaction* objects. Other models can easily be added through extension of the abstract *Model* class. Our implemented subclass translates each protein generator in turn into a standard protein generator model (Fig. 4) and finally adds the environmental signals that are the external inputs to the gene network.

The model can be written to file with the *FernMLWriter* class which implements the *ModelWriter* interface. The resulting *fernml* file can be used by *FERN* to run a stochastic simulation. Other *ModelWriter* classes can be implemented enabling the output of other data formats.

4.5 *gene_network.logic* package

The *gene_network.logic* package provides the functionality to read the user input, run simulations and calculate the score of a device.

User input The user input can be defined in an XML file (Listing 3). The binary timing diagrams for all the input and output signals are defined as binary strings of equal length. Each value represents a high (1) or a low (0) value for a time step. The *state_time* element defines the length of the time step in seconds. When the *visual* element is set to true, a plot of the output is shown during simulation. This makes the simulation a lot slower, but is convenient to initially test a newly designed logic gene network. When running large experiments, this value should be set to false.

Simulation The *GeneNetworkSimulation* class uses the *FERN* stochastic simulation package to run Gillespie's exact SSA for a logic device. The *BinaryTimingDiagramObserver*, an extension of the abstract *Observer* class provided by *FERN*, changes the

```
<?xml version="1.0" encoding="UTF-8"?>
<logic_device_settings name="Demultiplexer0">

  <network_template>Demultiplexer</network_template>

  <input id="in0">iim0</input>
  <input id="in1">iim1</input>

  <output id="out0">reporter0</output>
  <output id="out1">reporter1</output>

  <binary_timing_diagram>
    <plot id="in0" >00110</plot>
    <plot id="in1" >01100</plot>
    <plot id="out0">01000</plot>
    <plot id="out1">00100</plot>
  </binary_timing_diagram>

  <state_time>7200</state_time>
  <visual>true</visual>

</logic_device_settings>
```

Listing 3. An XML file containing the user input. A visualisation of this user input is shown in Fig. 7

concentration of the input signals according to the user defined binary timing diagrams. The average over 20 simulation runs is the result of a device simulation. The output is stored as a tab separated file.

5 RESULTS

Two experiments were done to show that our method works as intended. The results show that the score provides a good measure for the performance of a device. Furthermore, the results show that certain biological parts are preferred at certain locations in the network. Although there is no obvious explanation for this, it provides useful information for the design of a gene network.

We used the software to generate all possible devices for two logic gene network templates; a demultiplexer, which forwards the input signal to one of the two outputs based on the select signal, and a D-latch, a logic circuit that is able to store one bit. The demultiplexer is combinatorial, the output depends on the present inputs only, and the D-latch is sequential, the output also depends on the input history. In other words, the D-latch has memory and the demultiplexer does not. For both cases we devised a bioparts database in such a way that the number of possible devices was limited, enabling the simulation of all devices within reasonable time.

Both gene network templates consist of three different logic gate networks; an external NOT gate, an internal NOT gate, and a NOR gate (external meaning that it takes an environmental signal as input, while the internal version takes a TF as input).

5.1 Demultiplexer

Fig. 7 shows the user input that was used for this experiment (Listing 3 shows how this input is defined in an XML file). The demultiplexer is defined as the gene network template, the monomer inhibitors *iim0* and *iim1* are assigned to the inputs and the reporters *reporter0* and *reporter1* are assigned to the outputs (Fig. 7A). The edges *e0*, *e1*, *e3*, and *e4* are thereby occupied by user-defined proteins. Edge

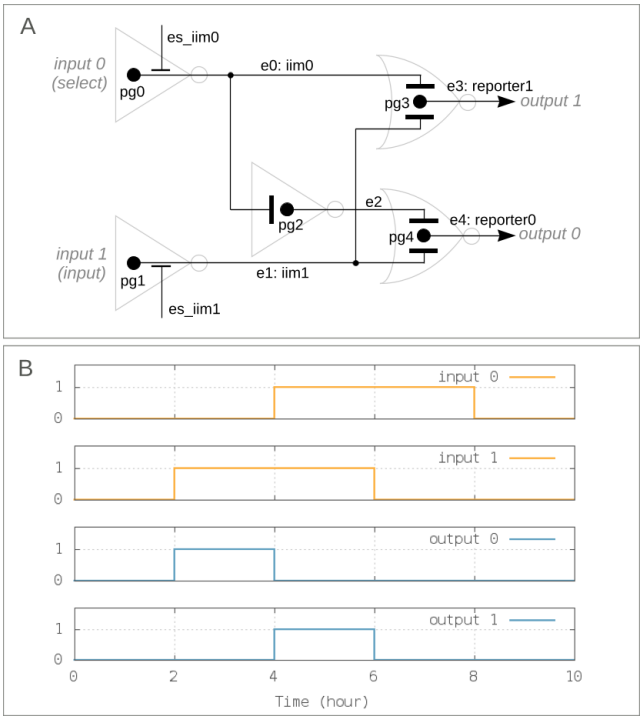


Fig. 7. User input for the design of a demultiplexer - *A)* The demultiplexer gene network template with the assigned input and output proteins. The demultiplexer network consists of three different logic gates; an external NOT gate, an internal NOT gate, and an internal NAND gate. *B)* The binary timing diagrams for each of the inputs and desired outputs.

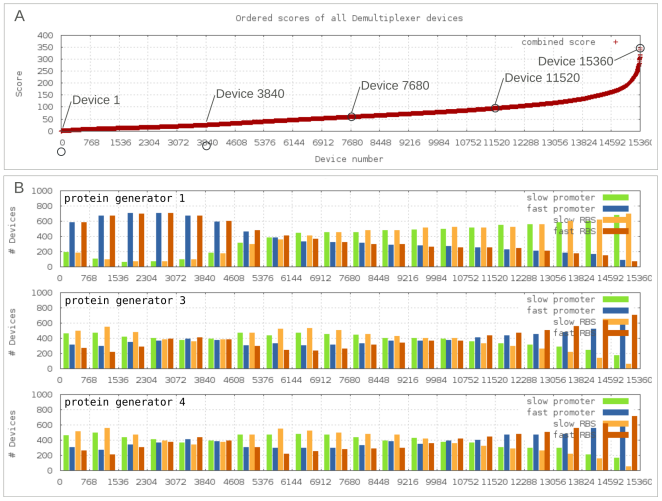


Fig. 8. Scores of all demultiplexer devices - *A)* The scores of 15360 devices ordered from low to high. *B)* The devices from the plot in *A* are subdivided into 20 bins. The histograms show the device counts of devices with slow and fast promoters and RBSs for protein generators 1, 3, and 4. It shows that most of the high scoring devices have a slow promoter and RBS for protein generator 1 and a fast promoter and RBS for protein generators 3 and 4.

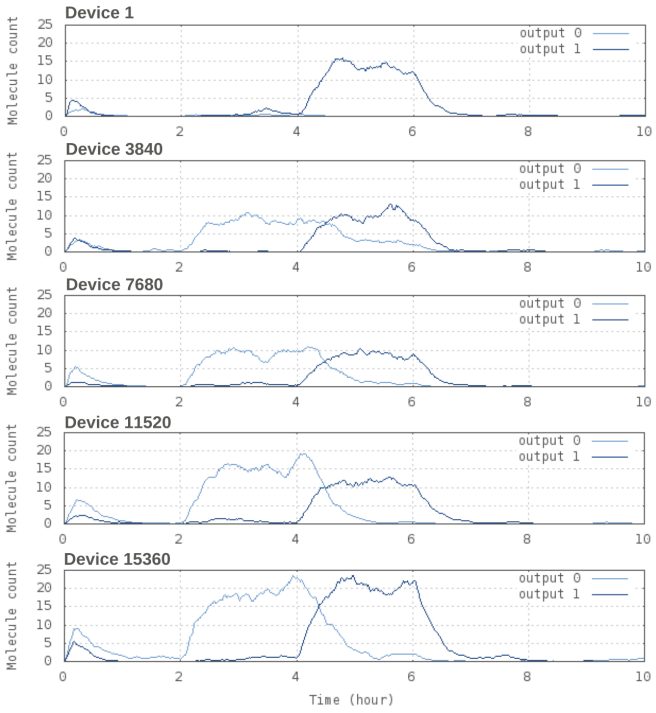


Fig. 9. Five simulation results - The averaged result over 20 simulation runs of the five circled demultiplexer devices in Fig.8A. These results show that, for this case, the score provides a good indication of the performance of the device.

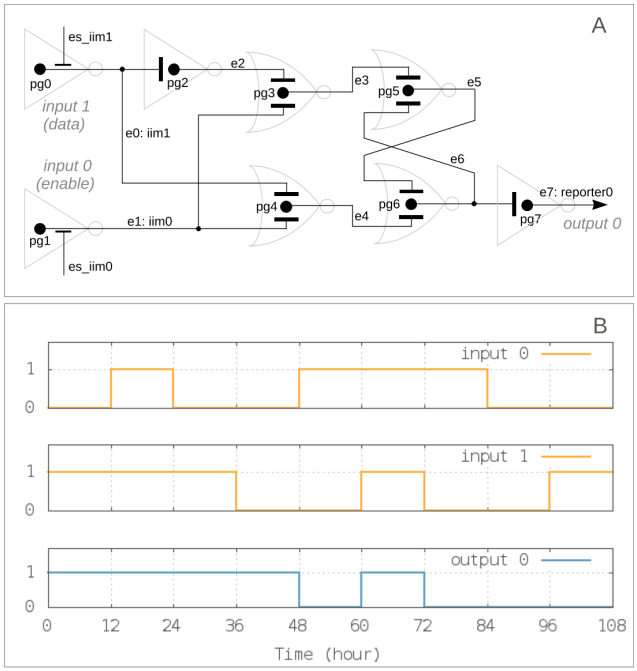


Fig. 10. User input for the design of a D-latch - *A)* The D-latch gene network template with the assigned input and output proteins. The D-latch network consists of three different logic gates; an external NOT gate, an internal NOT gate, and an internal NAND gate. *B)* The binary timing diagrams for each of the inputs and desired outputs.

e2 is the only edge left open. To go from a template to an instance, the *device builder* will have to assign an inhibitor to this edge.

In Fig. 7A it can be seen that both input TFs, the ones assigned to edge *e0* and edge *e1*, should be inhibitors that bind an inhibiting environmental signal. The TFs that we assigned to them, *iim0* and *iim1*, were both inhibitors present in the bioparts database. Also, for both it was defined (in the biopart database) that they bind an inhibiting environmental signal: *es_iim0* and *es_iim1* respectively. If we had assigned a wrong TF to an input, for example, an activator or an inhibitor that does not bind an environmental signal, then the software would have given an error message.

The binary timing diagrams for *input 0* and *input1* determine the concentrations of the environmental signals *es_iim0* and *es_iim1*. The first time step of the binary timing diagram is not taken into account for the evaluation of the device. The following time steps are all four possible logic combinations of the two inputs; 01, 11, 10, 00. A time of 2 hours was defined as the length of the time steps (the 7200 seconds as state-time in Listing 3).

The used bioparts database contained two RBSs, two promoters per promoter library and was devised in such a way that there was only one promoter library available for each protein generator. With five protein generators, this results in $2^{(5\times2)} = 1024$. Fifteen TFs (five monomers, five dimers, and five tetramers) could be assigned to edge *e2*, providing a total of $15 \times 1024 = 15360$ different possible devices for the gene network template in Fig. 7A.

Fig. 8A shows the score of all 15360 devices ordered from low to high. The simulation results of the circled devices are displayed in Fig. 9. Visual inspection of the simulation results show that the performance of the devices increases from device 1 to device 15360. This shows that the score provides a good indication for the performance of the demultiplexer device.

The ordered devices from the plot are subdivided into 20 bins of 768 devices. For each bin, the number of devices were counted that have a slow or a fast promoter for each of the protein generators. The same was also done for fast and slow RBSs. This resulted in a histogram for each of the protein generators. The histograms for protein generator 1, 3, and 4 are shown in Fig.8B. As can be seen, most of the high scoring devices have a slow promoter and a slow RBS for protein generator 1 (*pg1* in Fig. 7). For protein generators 3 and 4 (*pg3* and *pg4* in Fig. 7), most of the high scoring devices have a fast promoter and a fast RBS. The rest of the results can be found in the supplement.

5.2 D-latch

Similar to the demultiplexer, two monomeric inhibitors, *iim0* and *iim1*, that can bind the inhibiting environmental signals, *es_iim0* and *es_iim1*, were assigned to the input edges *e0* and *e1*. *Reporter0* was assigned to the output edge *e7*.

The D-latch has a data signal which is forwarded to output 0 when the enable signal is 1. When the enable signal is 0, the output stays in the current state, which can be either 0 or 1 (Fig. 10).

The bioparts database contained only 1 RBS and 1 promoter per promoter library. The TF for the edges e2, . . . , e6, is the only thing that was varied. With six monomer, six dimer, and six tetramer TFs, this provided a total of 702 possible devices.

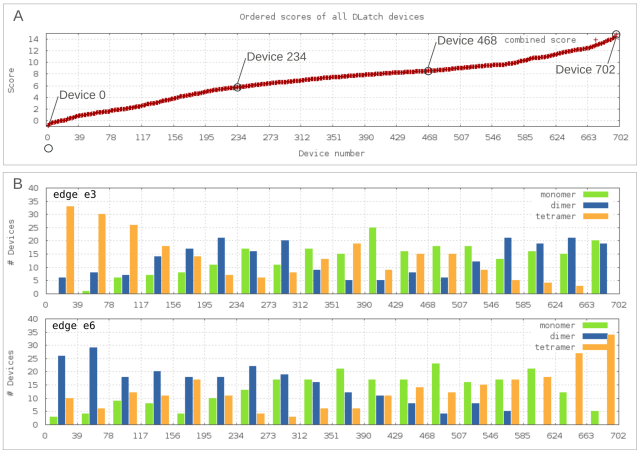


Fig. 11. Scores of all D-latch devices - *A)* The scores of 702 devices ordered from low to high. *B)* The devices from the plot in *A* are subdivided into 18 bins. The histograms show the device counts of devices with monomer, dimer, or tetramer TFs for the edge's *e3* and *e6*. It shows that most of the high scoring devices use a monomer or dimer TF for *e3* and a tetramer TF for *e6*.

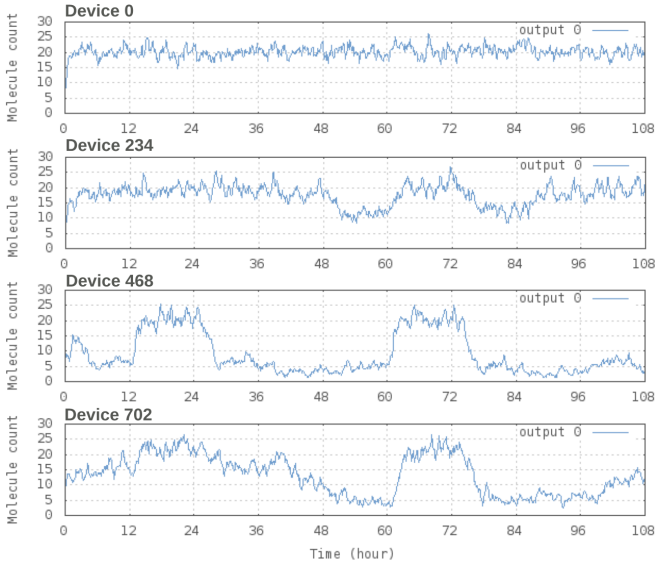


Fig. 12. Four simulation results - The averaged result over 20 simulation runs of the four circled devices in Fig. 11A. An increasing score shows an increasing performance, but the performance of the best scoring device is questionable. The signal drop that should occur at 48 hours seems to start already before that time point. Furthermore, the output signal seems to rise in the last time step while it should stay low.

Fig. 11A shows the score of all 702 devices ordered from low to high. The simulation results of the circled devices are displayed in Fig. 12. The performance of the devices seem to increase from device 1 to device 702, but the performance of the best device (702) is questionable. Whereas the output signal should drop to 0 at 48 hours, it looks as if the signal already decreases before this time point. It could be that although the system does not perform well, for this binary timing diagram it happens to show the desired result.

The ordered devices in the plot were subdivided into 18 bins. For each bin the number of devices were counted that use a monomer, a dimer, or a tetramer for an edge. This resulted in a histogram for each of the edges. The histograms of the edges *e3* and *e6* are shown in 11. As can be seen, most of the high scoring devices use a monomer or dimer for *e3* and a tetramer for *e6*. The results of the other edges can be found in the supplement.

6 DISCUSSION

The use of modularity makes it very easy to design logic gene network templates. The definition of only three different logic gates was enough to design the demultiplexer and D-latch shown in Fig. 7 and 10. Furthermore, we have also used them to design a decoder (*Supplement*). The software can be used to run a simulation on a random instance, which immediately provides some information about the functioning. For the design of gene networks this sounds appealing, but there are still a lot of hurdles to address before such a system can be used in practise.

First, the used bioparts database was artificial. The partsregistry contains a lot of genetic parts, but it provides no information about the parameters, which we need for the modeling. The majority of the parts are added by iGEM teams that design the parts they need for their system. Only a fraction of the parts is tested in the lab and just several parts are well characterised. Ellis et al. (2009) provides a method to build libraries of promoters with accompanying parameters. This method could in principle be used to build a bioparts database like ours.

Second, our method can generate possible gene network instances for a given template, but it is desirable to have an optimization algorithm that searches for the best performing gene network. This is not an easy task, since the search space, which is discrete, will in general be very large and high dimensional. Furthermore, defining an objective and evaluation measure was relatively easy in case of logic gene networks, but will be more challenging for other types of gene networks.

Third, running a stochastic simulation algorithm on a complex gene network is slow. For example, the simulation of a demultiplexer took about 0,7 seconds (it differs per device instance, higher molecule counts result in more reactions causing slower simulation). Simulation of a D-latch, a more complex network, took about 7 seconds. Such long simulation times make it infeasible to evaluate large amounts of device instances which is needed in case one needs to search for the best performing device. There is however continuous research on algorithms that provide a simulation speed-up and which are also aimed at the simulation of gene expression (Roussel and Zhu, 2006).

Finally, we do not take into account biological restrictions. For example, gene expression of an introduced gene network costs the cell energy. Energy is of great importance to a cell and it is not known how much can be used for a synthetic system. Also the other molecular species within the cell are discarded, while these can be of great influence. An introduced TF could for example cause cross-talk in an important gene network.

Although there is still a lot to achieve before a tool like ours can be used for the design of biological systems, the results show that the software can already be of use.

The 2008 University of Pavia iGEM team[6] and the 2009 University of Illinois iGEM team[7] designed a demultiplexer and a decoder respectively. Both spend a lot of time building a model from scratch, which in our case is done automatically. Furthermore, our method can be used to indicate which parts are prefered at what location in the network.

The type of TF (monomer, dimer, tetramer) to use as signal carrier seems to be of importance to the behaviour of the system. Using a different type of TF somewhere in the network results in a different model. For example, changing a monomer to a dimer results in the addition of an extra reaction. A lot of different models are thus needed to test all TF combinations within a gene network. This is no problem with our method, because the different models are generated automatically.

Furthermore, the results show interesting behaviours which could also be an inspiration for further research. The software could for example be used to investigate why, for some behaviour, a type of TF (monomer, dimer, tetramer) is prefered at a certain location within a network. The software could also be used to investigate noise propagation within gene networks similar to Hooshangi et al. (2005).

The method can also be further extended. Gene transcription regulation is the only regulation mechanism used, while biology provides much more regulation mechanisms that could be incorporated in a similar way. For example, translation regulation (riboswitches, *cis* or *trans* acting RNAs, RNA-thermometers) are regulation mechanisms that are very flexible and act on a faster time scale than transcription regulation (Davidson and Ellington, 2007; Beisel et al., 2008; Benenson, 2008).

The promoters we have used for our gene networks are relatively simple, they can bind maximally two TFs that do not overlap (so no competitive TF binding). More complex hybrid promoters (Cox et al., 2007) that implement a more complex can result in less large gene networks, which would provide a more energy efficient solution. Also the functioning of the promoters could be optimized for the use in synthetic gene networks (Setty et al., 2003; Mayo et al., 2006).

REFERENCES

Anderson, J. C., Clarke, E. J., Arkin, A. P., Voigt, C. A., 2006. Environmentally controlled invasion of cancer cells by engineered bacteria. Journal of molecular biology 355 (4), 619–627.

Anderson, J. C., Voigt, C. A., Arkin, A. P., 2007. Environmental signal integration by a modular and gate. Mol. Syst. Biol. 3, 133.

Andrianantoandro, E., Basu, S., Karig, D. K., Weiss, R., 2006. Synthetic biology: new engineering rules for an emerging discipline. Mol. Syst. Biol. 2, 2006.0028.

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

Beisel, C. L., Bayer, T. S., Hoff, K. G., Smolke, C. D., 2008. Model-guided design of ligand-regulated rnai for programmable control of gene expression. Mol. Syst. Biol. 4, 224.

Benenson, Y., 2008. Small hairpin rna as a small molecule sensor. Mol. Syst. Biol. 4, 227.

Bernstein, J. A., Khodursky, A. B., Lin, P. H., Lin-Chao, S., Cohen, S. N., 2002. Global analysis of mrna decay and abundance in escherichia coli at single-gene resolution using two-color fluorescent dna microarrays. Proceedings of the National Academy of Sciences 99 (15), 9697.

[6] `http://2008.igem.org/Team:UNIPV-Pavia`

[7] `http://2009.igem.org/Team:Illinois`

Cai, X., Wang, X., 2007. Stochastic modeling and simulation of gene networks-a review of the state-of-the-art research on stochastic simulations. IEEE Signal Processing Magazine 24 (1), 27–36.

Chandran, D., Bergmann, F. T., Sauro, H. M., 2009. Tinkercell: Modular cad tool for synthetic biology. Arxiv preprint arXiv:0907.3976.

Cox, R. S., Surette, M. G., Elowitz, M. B., 2007. Programming gene expression with combinatorial promoters. Mol. Syst. Biol. 3, 145.

Davidson, E. A., Ellington, A. D., 2007. Synthetic rna circuits. Nat. Chem. Biol. 3 (1), 23–8.

Del Vecchio, D., Ninfa, A. J., Sontag, E. D., 2008. Modular cell biology: retroactivity and insulation. Mol. Syst. Biol. 4, 161.

Ellis, T., Wang, X., Collins, J. J., 2009. Diversity-based, model-guided construction of synthetic gene networks with predicted functions. Nature Biotechnology.

Elowitz, M., Leibler, S., 2000. A synthetic oscillatory network of transcriptional regulators. Nature 403 (6767), 335–338.

Endy, D., 2005. Foundations for engineering biology. Nature 438 (7067), 449–53.

Erhard, F., Friedel, C. C., Zimmer, R., 2008. Fern - a java framework for stochastic simulation and evaluation of reaction networks. BMC Bioinformatics 9, 356.

Friedland, A. E., Lu, T. K., Wang, X., Shi, D., Church, G., Collins, J. J., 2009. Synthetic gene networks that count. Science 324 (5931), 1199–202.

Gardner, T., Cantor, C., Collins, J., 2000. Construction of a genetic toggle switch in escherichia coli. Nature 403 (6767), 339–342.

Gillespie, D., 2007. Stochastic simulation of chemical kinetics. Annual Review Of Physical Chemistry 58, 35.

Gillespie, D., et al., 1977. Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry 81 (25), 2340–2361.

Grilly, C., Stricker, J., Pang, W. L., Bennett, M. R., Hasty, J., 2007. A synthetic gene network for tuning protein degradation in saccharomyces cerevisiae. Mol. Syst. Biol. 3, 127.

Heinemann, M., Panke, S., 2006. Synthetic biology–putting engineering into biology. Bioinformatics 22 (22), 2790–9.

Hooshangi, S., Thiberge, S., Weiss, R., 2005. Ultrasensitivity and noise propagation in a synthetic transcriptional cascade. Proceedings of the National Academy of Sciences 102 (10), 3581–3586.

Kaern, M., Elston, T. C., Blake, W. J., Collins, J. J., 2005. Stochasticity in gene expression: from theories to phenotypes. Nat. Rev. Genet. 6 (6), 451–64.

Maheshri, N., O'Shea, E. K., 2007. Living with noisy genes: how cells function reliably with inherent variability in gene expression.

Marchisio, M. A., Stelling, J., 2008. Computational design of synthetic gene circuits with composable parts. Bioinformatics 24 (17), 1903–10.

Mayo, A. E., Setty, Y., Shavit, S., Zaslaver, A., Alon, U., 2006. Plasticity of the cis-regulatory input function of a gene. PLoS Biol. 4 (4), e45.

Mirschel, S., Steinmetz, K., Rempel, M., Ginkel, M., Gilles, E. D., 2009. Promot: modular modeling for systems biology. Bioinformatics 25 (5), 687.

Ozbudak, E. M., Thattai, M., Kurtser, I., Grossman, A. D., van Oudenaarden, A., 2002. Regulation of noise in the expression of a single gene. Nat. Genet. 31 (1), 69–73.

Roussel, M. R., Zhu, R., 2006. Validation of an algorithm for delay stochastic simulation of transcription and translation in prokaryotic gene expression. Phys. Biol 3, 274–284.

Sayut, D. J., Niu, Y., Sun, L., 2009. Construction and enhancement of a minimal genetic and logic gate. Appl. Environ. Microbiol. 75 (3), 637–42.

Schlosshauer, M., Baker, D., 2004. Realistic protein–protein association rates from a simple diffusional model neglecting long-range interactions, free energy barriers, and landscape ruggedness. Protein Science: A Publication of the Protein Society 13 (6), 1660.

Setty, Y., Mayo, A. E., Surette, M. G., Alon, U., 2003. Detailed map of a cis-regulatory input function. Proceedings of the National Academy of Sciences 100 (13), 7702–7707.

Shahrezaei, V., Swain, P. S., 2008. The stochastic nature of biochemical networks. Curr. Opin. Biotechnol. 19 (4), 369–74.

Shetty, R. P., Endy, D., Knight Jr, T. F., 2008. Engineering biobrick vectors from biobrick parts. Journal of Biological Engineering 2, 5.

Smith, L. P., Bergmann, F. T., Chandran, D., Sauro, H. M., 2009. Antimony: A modular model definition language. Bioinformatics 25 (18), 2452.

Sprinzak, D., Elowitz, M. B., 2005. Reconstruction of genetic circuits. Nature 438 (7067), 443–8.

Stricker, J., Cookson, S., Bennett, M. R., Mather, W. H., Tsimring, L. S., Hasty, J., 2008. A fast, robust and tunable synthetic gene oscillator. Nature 456 (7221), 516–9.

Logic gene network design: a CAD tool based on modularity and standardization
Supplementary data

B.A. van den Berg

Chapter 1

Kinetic constants

kinetic constant	min	max	
k_{tc}	0.00625	0.01	one transcript per gene each 160 . . . 100 sec
k_{tl}	0.00625	0.01	one protein per mRNA each 160 . . . 100 sec
k_{deg_r}	0.00144	0.00385	half-life 8 . . . 3 min (Bernstein et al., 2002)
k_{deg_p}	0.000183	0.00167	half-life 91 . . . 10 min (Grilly et al., 2007)
k_{btf}	0.00278	0.00417	Hooshangi et al. (2005) +/− 1 min
k_{utf}	0.00139	0.00208	Hooshangi et al. (2005) +/− 2 min
k_{bp}	0.0001	0.001	Schlosshauer and Baker (2004)
k_{up}	0.0000167		fixed, same as Hooshangi et al. (2005)
k_{bs}	0.000833		fixed, same as Hooshangi et al. (2005)
k_{us}	0.00167		fixed, same as Hooshangi et al. (2005)

Table 1.1: *Kinetic constant ranges used to build artificial biopart database.*

Chapter 2

Decoder

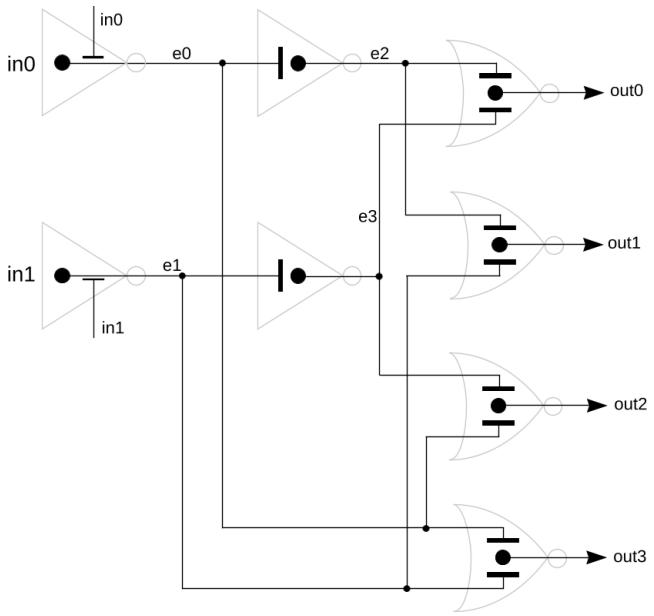


Figure 2.1: *Gene network template for a 2-to-4 line decoder.*

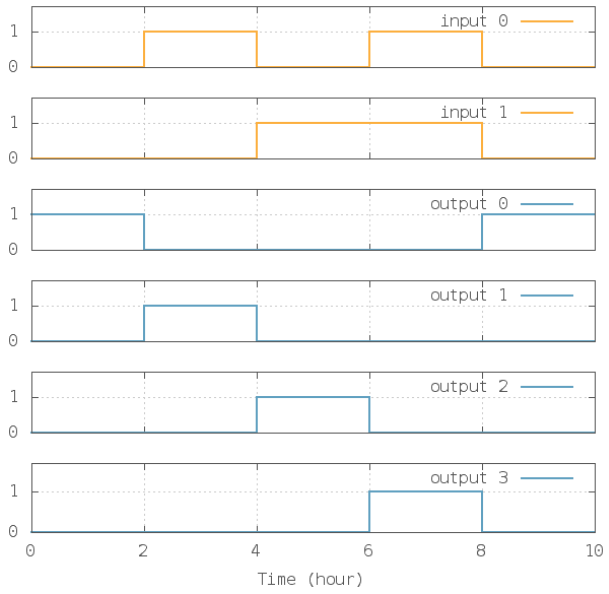


Figure 2.2: *Input signals and desired output signals for the 2-to-4 line decoder.*

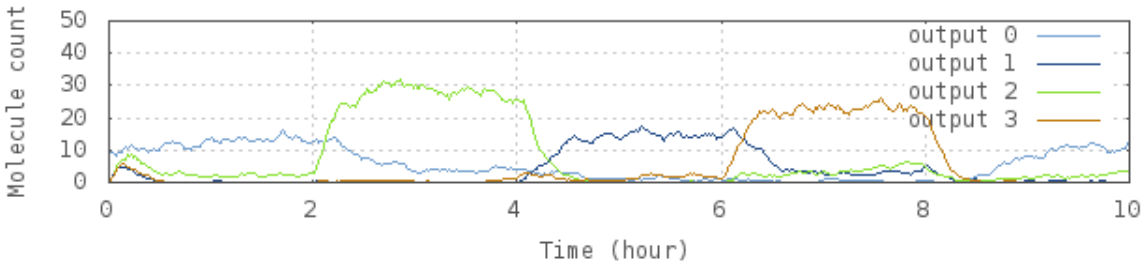


Figure 2.3: *Simulation result of a random device instance of the 2-to-4 line decoder template. The simulation result is the average over 20 simulation runs.*

Chapter 3

Results experiment 1: demultiplexer

```
<?xml version="1.0" encoding="UTF-8"?>
<logic_device_settings name="Demultiplexer0">

  <gene_network>Demultiplexer</gene_network>

  <input id="in0">iim0</input>
  <input id="in1">iim1</input>

  <output id="out0">reporter0</output>
  <output id="out1">reporter1</output>

  <binary_timing_diagram>
    <plot id="in0" >00110</plot>
    <plot id="in1" >01100</plot>
    <plot id="out0">01000</plot>
    <plot id="out1">00100</plot>
  </binary_timing_diagram>

  <state_time>7200</state_time>
  <visual>false</visual>

</logic_device_settings>
```

Listing 3.1: *Demultiplexer settings*

Figure 3.1: *Correlation output 0.*

Figure 3.2: *Correlation output 1.*

Figure 3.3: *Low value output 0.*

Figure 3.4: *Low value output 1.*

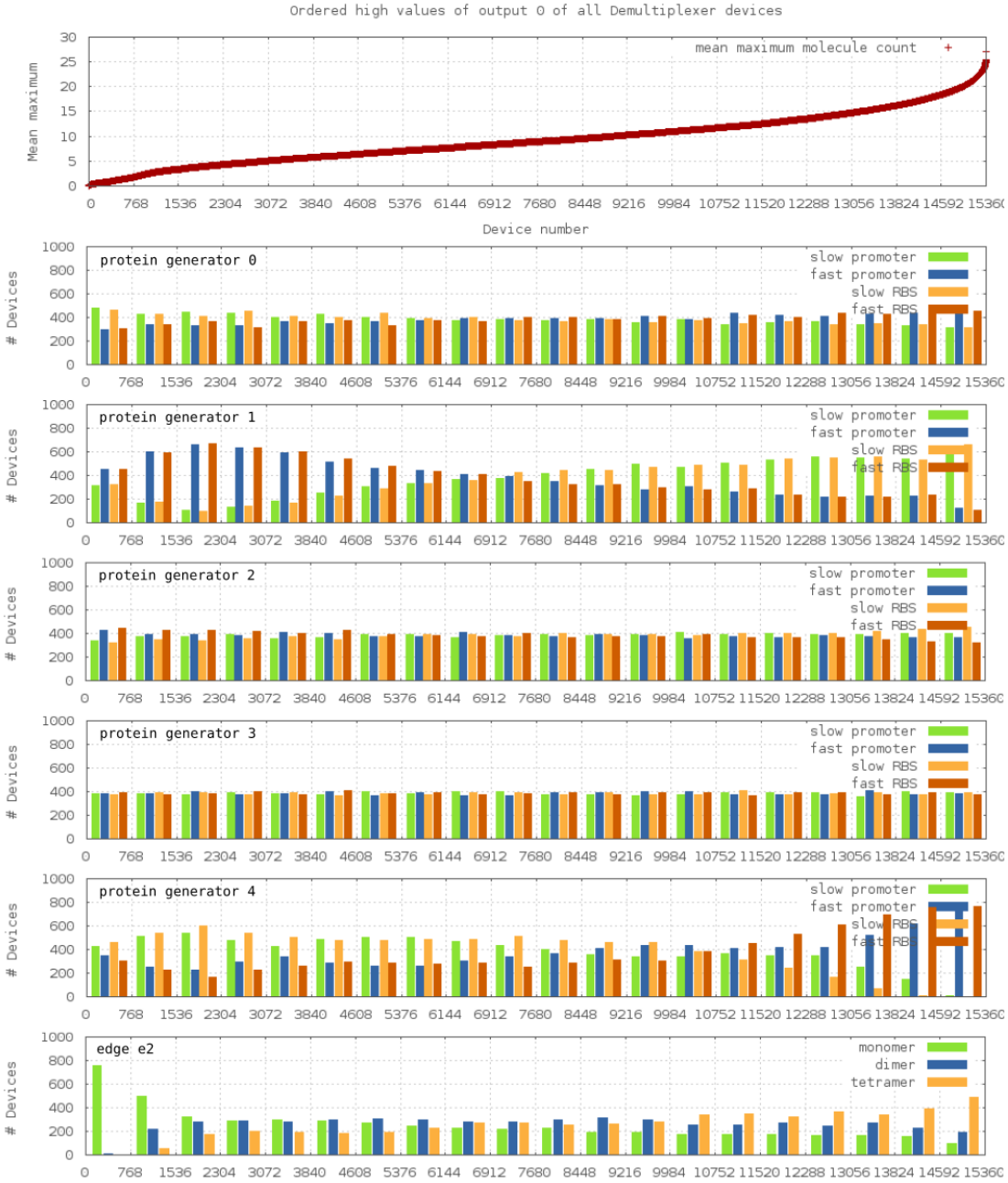


Figure 3.5: *High value output 0.*

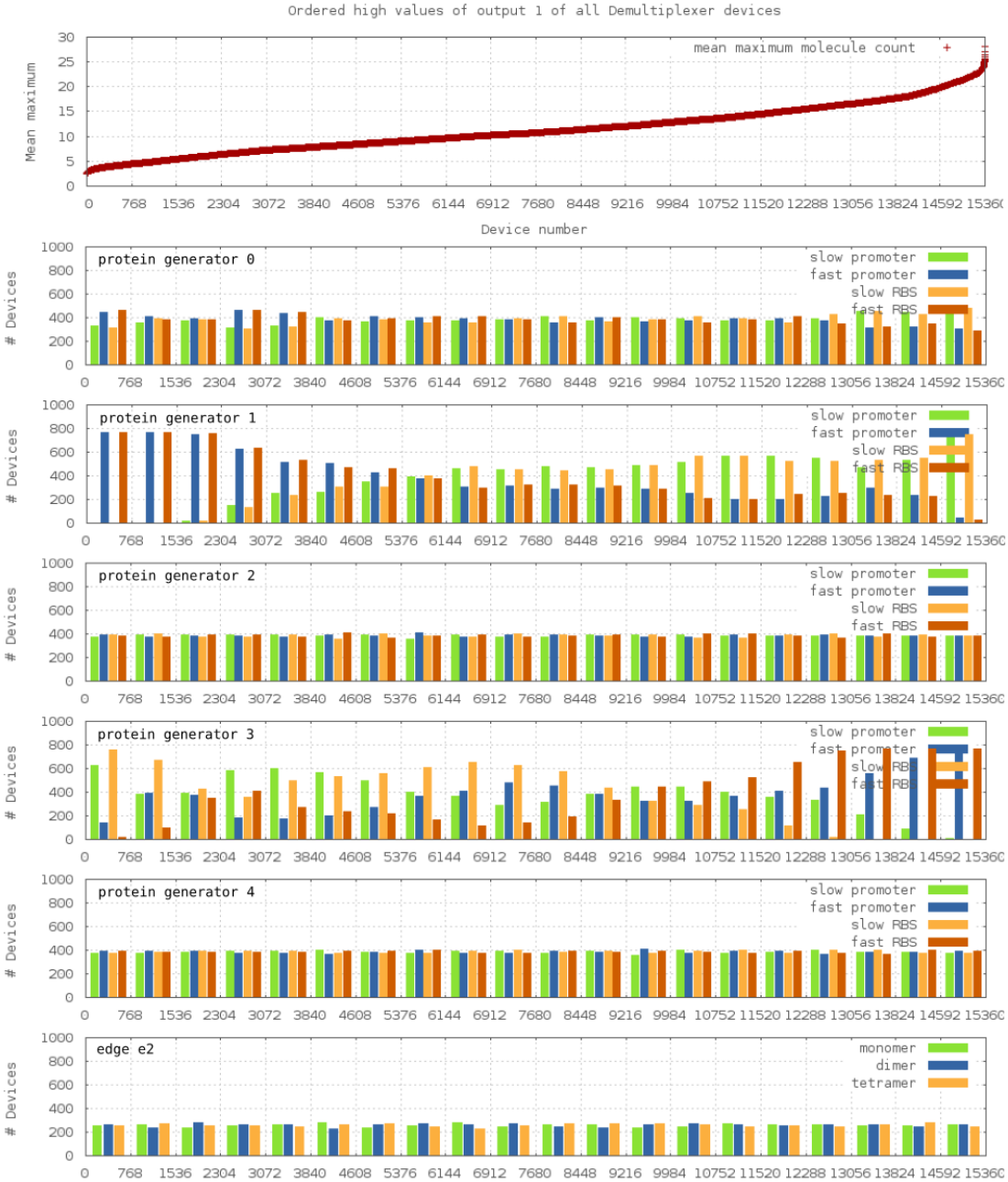


Figure 3.6: *High value output 1.*

Figure 3.7: *Score*

Chapter 4

Results experiment 2: DLatch

```
<?xml version="1.0" encoding="UTF-8"?>
<logic_device_settings name="DLatch0">

  <gene_network>DLatch</gene_network>

  <input id="in0">iim0</input>
  <input id="in1">iim1</input>

  <output id="out0">reporter0</output>

  <binary_timing_diagram>
    <plot id="in0" >010011100</plot>
    <plot id="in1" >111001001</plot>
    <plot id="out0">111101000</plot>
  </binary_timing_diagram>

  <state_time>43200</state_time>
  <visual>false</visual>

</logic_device_settings>
```

Listing 4.1: *DLatch settings*

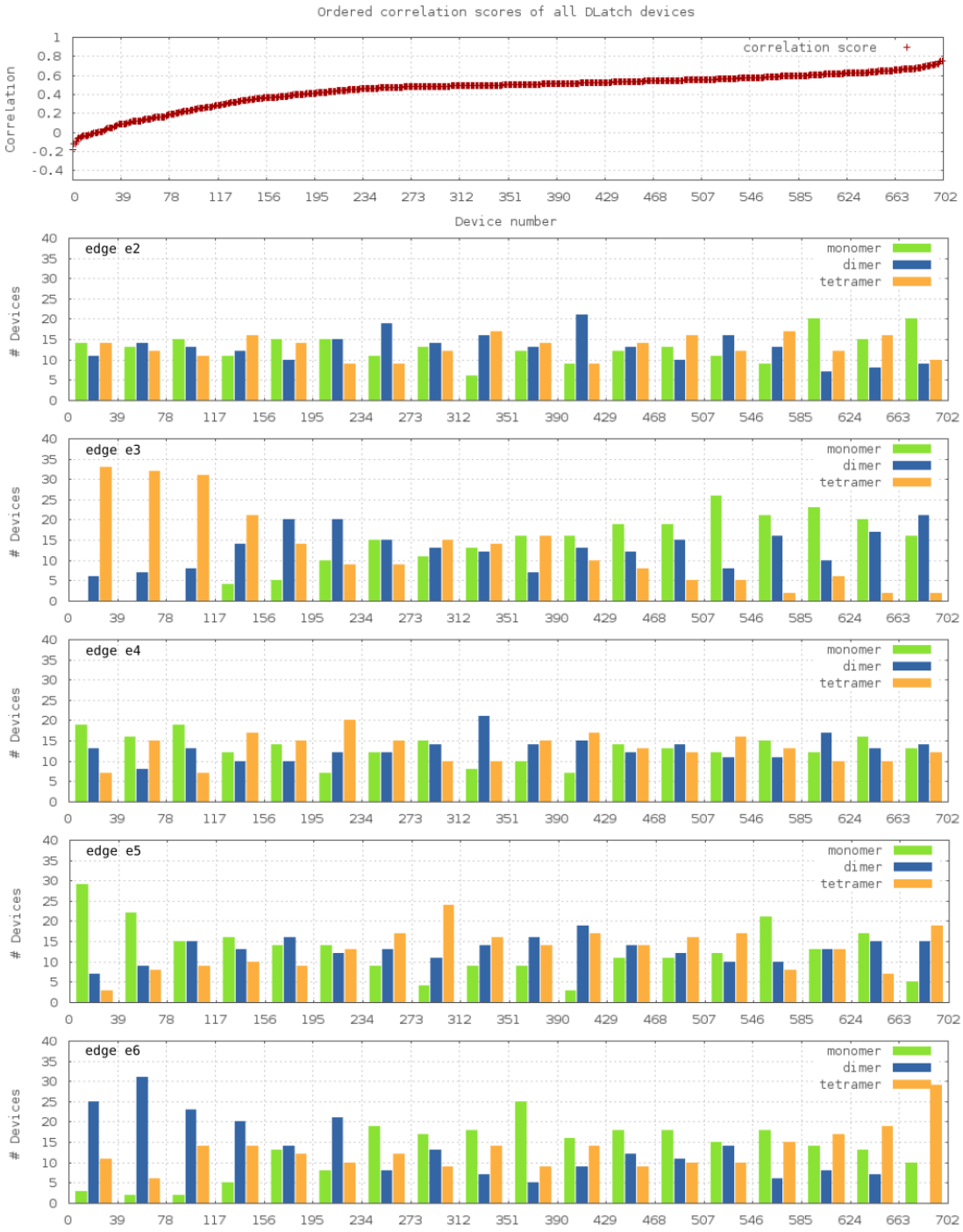


Figure 4.1: *Correlation.*

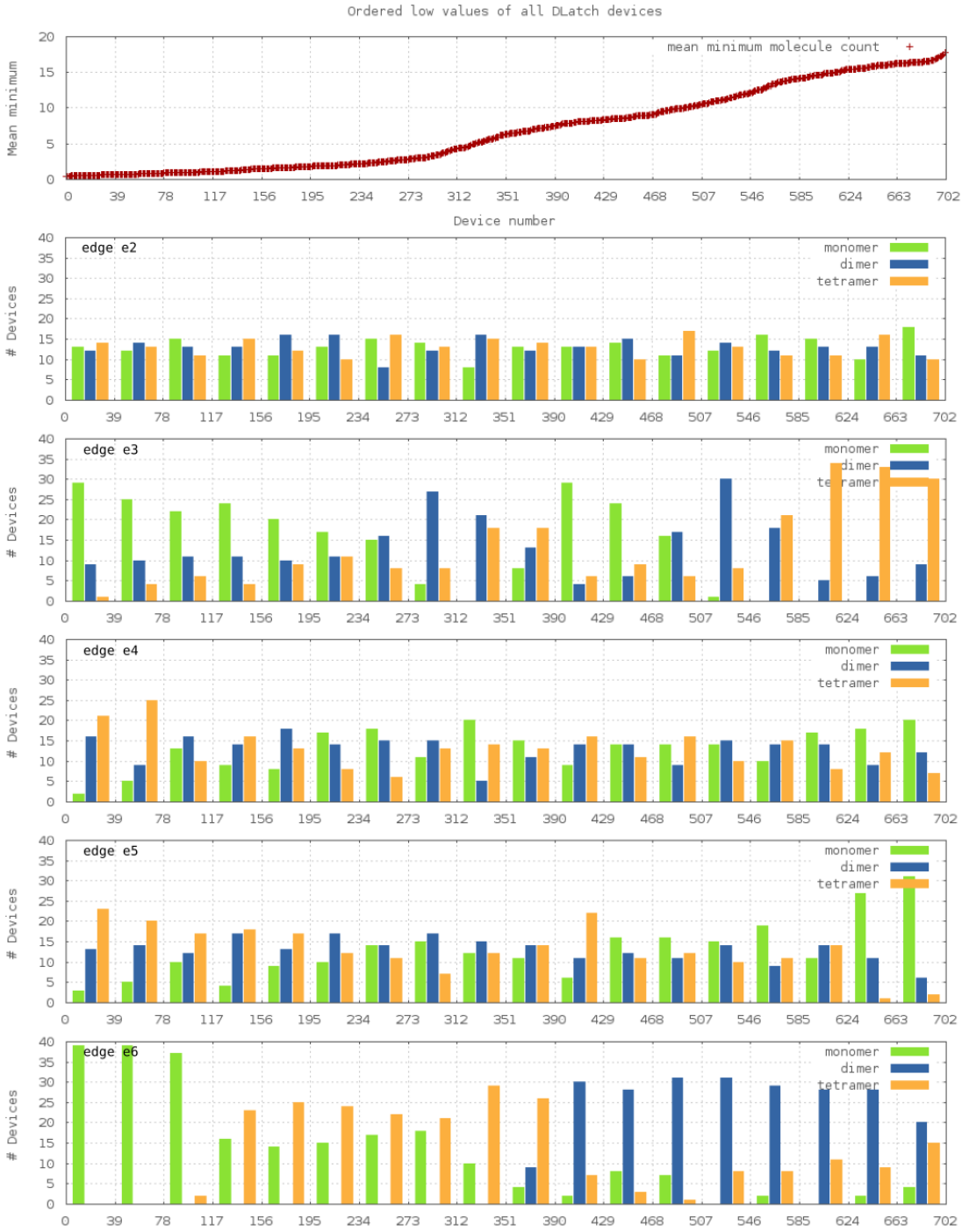


Figure 4.2: *Low value.*

Figure 4.3: *High value.*

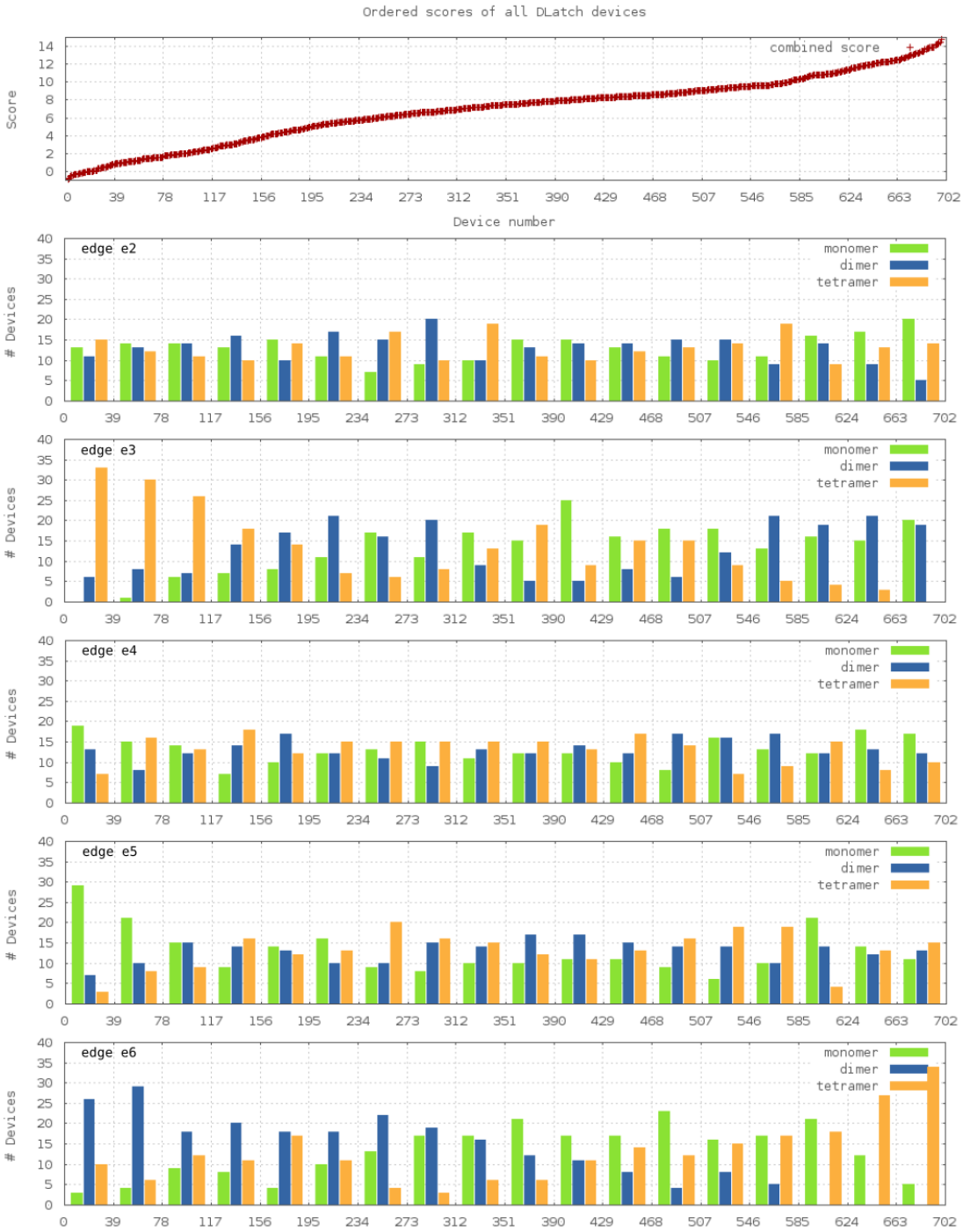


Figure 4.4: *Score.*

Bibliography

Bernstein, J. A., Khodursky, A. B., Lin, P. H., Lin-Chao, S., Cohen, S. N., 2002. Global analysis of mrna decay and abundance in escherichia coli at single-gene resolution using two-color fluorescent dna microarrays. Proceedings of the National Academy of Sciences 99 (15), 9697.

Grilly, C., Stricker, J., Pang, W. L., Bennett, M. R., Hasty, J., 2007. A synthetic gene network for tuning protein degradation in saccharomyces cerevisiae. Mol. Syst. Biol. 3, 127.

Hooshangi, S., Thiberge, S., Weiss, R., 2005. Ultrasensitivity and noise propagation in a synthetic transcriptional cascade. Proceedings of the National Academy of Sciences 102 (10), 3581–3586.

Schlosshauer, M., Baker, D., 2004. Realistic protein–protein association rates from a simple diffusional model neglecting long-range interactions, free energy barriers, and landscape ruggedness. Protein Science: A Publication of the Protein Society 13 (6), 1660.

Logic gene network design: a CAD tool based on modularity and standardization
Work document

B.A. van den Berg

Contents

Chapter 1

Research Proposal

One of the goals of synthetic biology is to engineer biological systems that add novel functionalities to a cell. From an engineering point of view this would enable us to build useful biological applications, such as the production of a chemical/drug or the metabolization of waste material.

1.1 Problem description

Multiple transcription networks performing a simple function have been engineered over the last few years. Elowitz and Leibler (2000) and Gardner et al. (2000) were the first to engineer an oscillator and a toggle switch respectively using trial-and-error. Guet et al. (2002) used a combinatorial approach to engineer three transcriptional networks that act as a logic gate.

For these simple functions it is possible to design a transcriptional network manually, using intuition only. But for a more complex function, intuition is not enough. A more complex function requires a larger network, meaning a huge increase of the solution space, which makes a combinatorial approach infeasible as well.

The main direction of synthetic biology towards the simplification of engineering complex biological systems, as proposed by Endy (2005), is standardization, decoupling, and abstraction. This basically means that we need well characterized biological building blocks that perform a defined function, also when connected to each other, so that these standardized building blocks can be used to build more complex systems.

Although there are currently numerous synthetic biological systems, these are designed to perform a specific function on itself, they are not designed to be a part of a larger system, i.e. they are not designed to be modular. So currently there are no reusable biological parts that could be used to design a complex system, which is therefor still an infeasible task.

It is also unknown if this modular approach is feasible within biology. Theoretically a modular approach makes it easier build complex systems, but biology puts some restrictions on this approach. Research is needed to provide insight into the possibilities of applying a modular approach to design biological systems.

1.2 Goal

The goal of this project is to determine if it is feasible to build a complex biological system using modular biological building blocks and if so, to determine the limits of this modular approach, taking into account biological constraints such as energy consumption.

1.3 Proposal

To test if a modular approach is feasible in biology we propose to take a similar approach as is done within electrical engineering, where logic gates are used as standard building blocks to build complex logic circuits. Similarly, biological logic gates will be designed which will be used to build biological logic circuits.

To make things less complicated we propose to initially use transcription regulation for the design of the logic gates. In other words, gene transcription networks will be designed that act as a logic gate. At a later stage, post-transcriptional regulation could be incorporated into the design as well.

Abstraction

To simplify the design of a complex system we propose to use an abstraction hierarchy as proposed by Endy (2005) (Figure 1.1). The parts (promoters, ribosome binding sites, protein coding regions, and terminators) will be used to build logic gates (devices) and these logic gates will in turn be used to build a logic circuit (system). A similar approach as used for digital electronics.

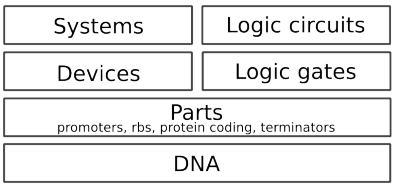


Figure 1.1: *Abstraction hierarchy as proposed by Endy (2005) with the aim of managing complexity. Parts are DNA stretches with a defined function. These parts can be used to build devices which in turn can be used to build systems. Logic gates will be designed within this project and these will be used to build logic circuits. Although the design of these interconnectable logic gates is a difficult task, having such devices greatly simplifies the design of complex systems.*

Modular logic gates (devices)

As a first step towards a biological circuit we propose to design a number of well characterized (Canton et al., 2008; Lucks et al., 2008) modular logic gates based on a number of known biological logic gates, in which modular means that the devices should be able to maintain their intrinsic properties when connected to any other device (Sauro, 2008).

Logic circuits (systems)

Logic circuits (Figure 1.2) will be build *in silico* using the designed logic gates to test if they provide the desired behavior when interconnected. In case of success, this testing will be used to explore the limits of circuit design. Otherwise the test will be used to see which problems occur and what could be done to solve these problems.

1.4 Approach

Simulation

The simulator that will be used is TABASCO (Kosuri et al., 2007). This simulator mimics gene expression at two different resolutions: a single molecule resolution for a detailed simulation of

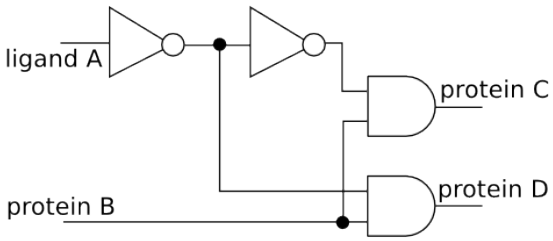


Figure 1.2: *Example of a simple logic circuit diagram that could be used to test the logic gates. In this case a demultiplexer: a high concentration of ligand A will cause protein C to mimic the 'binary' value of protein B and a low concentration of ligand A will cause protein D to mimic the 'binary' value of protein B.*

the transcription process, and to keep the method computational tractable, a species level for the translation process (Figure 1.3). The main advantages compared to other simulators are that:

- In contrast to most continuous and discrete simulators, it is scalable, able to predict the behavior of complex transcriptional networks.
- It is a stochastic simulator, providing a more accurate prediction than a continuous approach.
- The model used is closer to reality than a mathematical model, which decreases the risk of using unrealistic parameters.
- It provides more insight into what happens at the molecular scale.
- Modeling multiple regulation is much easier compared to mathematical modeling.

The organism in which all the processes will be simulated is *Escherichia coli*. This is a model organism which is well characterized and is mostly used for the implementation of synthetic biological networks. Parameters like cell volume, number of polymerases, polymerase speed, number of DNA copies, etc., need to be defined to model the host cell. The ecocyc database (`http://ecocyc.org/`), as well as available literature, can be used to retrieve such numbers.

Biological parts

A set of well characterized biological parts will be retrieved from the parts registry (`http://www.partsregistry.org`) and from literature. Cox et al. (2007) for example provides a library of well characterized multiple regulated promoters.

The parameters of the parts that are needed by the simulator, such as the strength of the ribosome binding site, need to be defined.

A set of transcription factors (protein coding parts) will be used as signal carriers which can regulate gene expression when bind to a promoter. A set of ligands will be used as input signal to the circuit which are being sensed by some of the transcription factors. These can be activated or deactivated through binding of the ligand. Finally some protein coding parts will be used as output signal, i.e. reporter.

Modular biological logic gates (devices)

Biological logic gates will be retrieved from literature (Guet et al., 2002; Mayo et al., 2006; Cox et al., 2007; Rodrigo et al., 2007; Anderson et al., 2007). The designs of these logic gates will be used as a starting point of the design of a set of logic gates which are modular, i.e. able to maintain their intrinsic properties when connected to any other logic gate (Sauro, 2008). The behavior of these gates, on their own and connected to other gates, will be predicted using the simulator.

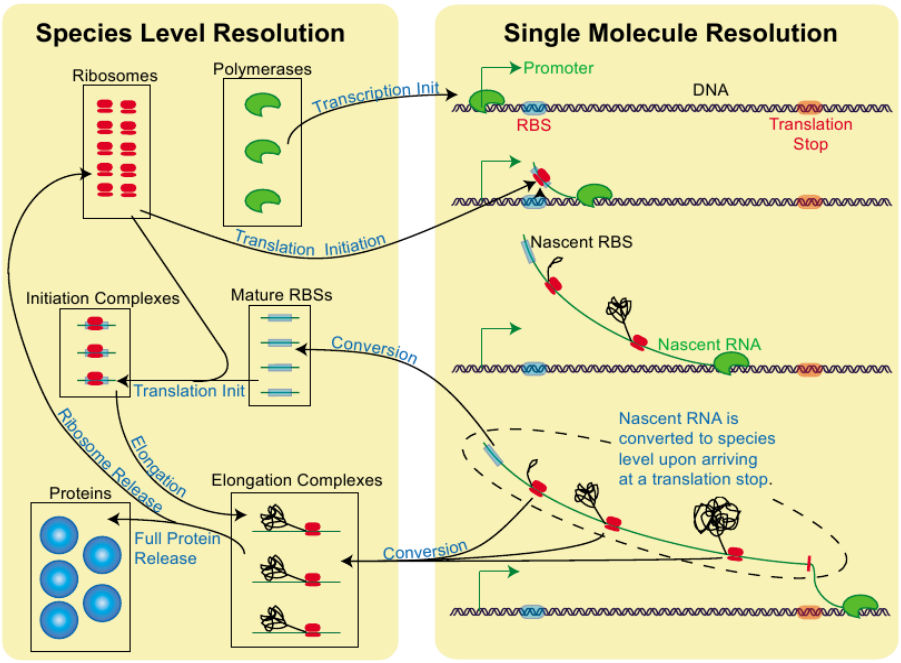


Figure 1.3: *A simplified flow diagram that shows how TABASCO transitions between two resolutions: the detailed single molecular resolution and the species level resolution (Kosuri et al., 2007).*

The goal is to find biological mechanisms that can be used to solve the encountered problems. Similar to the approach of Del Vecchio et al. (2008), electrical engineering will serve as an inspiration to find solutions.

These mechanisms will be used to design a set of modular biologic logic gates that can be used to build a biologic logic circuit.

Some expected problems when interconnecting logic gates to form a logic circuit are discussed in the following sections.

Retroactivity

One problem that also occurs in other engineering fields, as depicted by Del Vecchio et al. (2008), is retroactivity - the effect a downstream device has on an upstream device, which has to do with the modularity of a device. Minimizing the retroactivity results in a modular device. Del Vecchio et al. (2008) proposes some mechanisms to minimize the retroactivity of a device. These mechanisms will be tested in silico and used for the design of the logic gates.

Timing

Another expected problem is timing. A circuit containing a very long and a very short path can cause undesired behavior. It will take more time for a signal to propagate through the long path then the time it will take for a signal to propagate through the short path. This can lead to a situation where a signal that traverses the short path overtakes a signal that traverses the long path. Digital circuits use a clock that synchronizes the system in order to solve this problem. A similar solution could be needed for the biological case as well.

Signal propagation

The propagation of a signal is also an important issue when building circuits. For example when a signal (a transcription factor) has a very high degradation rate it could happen that the concentration of the signal does not pass the logic '1' threshold, which means that the signal will be lost. On the other hand, when the signal has a very low degradation rate, the signal could stay above the logic '1' threshold to long.

Discretization

Also the discretization of the continuous signal into a binary value could lead to some difficulties. It could be that different thresholds are needed for different signals (transcription factors) or even different thresholds in between different logic gates.

Other problems

Two approaches will be used to discover other problems. Firstly, literature about the early stages of digital logic will be used to identify other potential problems. Secondly, *in silico* simulations of biological logic circuits, using the biological logic gates, will be used to identify problems.

Logic circuits (systems)

Logic circuits will be build *in silico* in order to test the functioning of the designed logic gates. Logic circuits with different properties will be designed using available logic circuit design methods.

Next to the testing of the logic gates another goal is to explore the limitations of the biological circuits. This will be used to define a set of rules and restrictions that specify the possibilities/limitations of the biological circuits that can be designed, e.g. a logic circuit can have a maximal cascading depth of four logic gates.

If the tests do not provide any good results, this phase will be used to identify why it doesn't provide the desired results and if these problems could be resolved, i.e. if this modular approach is feasible in biology.

1.5 Validation

The only validation that will be done within this project is the testing of the logic gates as mentioned in the previous section. A number of logic circuits performing some logic function will be taken from electrical engineering and build using the logic gates to see if the biological circuit provides a similar behavior. This will hopefully provide more insight into the possibilities of a modular approach within synthetic biology.

Ideally one would like to test some designs in the lab to see if the *in silico* prediction also holds *in vivo*. Although labwork is not possible within this project, iGEM teams provide a nice opportunity to have the designed logic gates validated.

1.6 Work plan

This section provides an action plan that will be used for the project. A planning of these tasks is given in Appendix A.

Simulator - testing and configuration

- Test if TABASCO provides accurate gene expression predictions. Existing models from the biomodels database (`www.ebi.ac.uk/biomodels-main`) can be used to check if the results are comparable to other simulators.

- Configuration of TABASCO, i.e. setting the global parameters (cell volume, number of ribosomes, etc.) of the simulator. The simulations from the previous step can be used for the fine-tuning of the settings.
- A better performing simulator will have to be chosen if TABASCO does not provide accurate results.
- Add additional functionality to the simulator when needed.

Logic gates - initial design

- Define a set of biological parts that will be used to build the logic gates.
- Retrieve existing biological logic gates from literature and use these to build a set of logic gates that are composed of standard biological parts and that can be interconnected.
- Build a set of ligands, transcription factors, and reporter genes which will be used as input, internal, and output signals. Also specify interactions between ligands, transcription factors, and promoters.

Logic circuits - problem identification

- Search the literature about digital logic to identify potential problems.
- Define a set of logic circuits that will be used to investigate the behavior of the biologic logic gates, when interconnected. These circuits could be seen as a training set in order to design a number of modular logic gates.
- Use the simulator to predict the behavior of the biologic logic circuits to investigate what problems occur.
- Analyze the results and define a list of problems that need to be solved.

Logic gates - modular design

- Build a list of useful biological mechanisms, e.g. a high gain caused by a high polymerase-promoter affinity (Del Vecchio et al., 2008).
- Use electrical engineering as an inspiration to find solutions to the problems using the biological mechanisms listed in the previous step.
- Use the mechanisms to design a set of modular biological logic gates.

Logic circuits - validation

- Define a list of logic circuits which will be used to validate the functioning of the modular biologic logic gates. These circuits should be distinct from the ones used to design the modular logic gates. This set could be seen as the test set.
- Use the simulator to predict the behavior of the logic circuit and compare these results to the expected behavior.
- In case of success, explore the limits of the biologic logic circuits and create a list of design rules for the design of biological logic circuits.

Chapter 2

Approach

During the project the goal was adjusted from a more theoretical goal, testing the feasibility and exploring the possibilities of using modular design and standardization for the design of biological systems, to a more practical goal, providing a design method in which modular design and standardization are used to simplify the design process and enable the design of more complex biological systems. Also, the proposed stochastic simulator, TABASCO (Kosuri et al., 2007), appeared to be less developed as expected. Instead, we used the FERN framework (Erhard et al., 2008) to run Gillespie's exact stochastic simulations algorithm (Gillespie et al., 1977).

The developed method can be used for the design of gene networks that implement a logic function, which we will call logic gene networks. The only regulation mechanism used to build the system is transcription regulation through the binding of Transcription Factors (TF) to a gene. Environmental signals, such as small molecules, light or temperature, can inhibit or activate TFs. These signals will be called external signals, and the TFs will be called internal signals. The designed logic gene networks will take some environmental signals as input, use the internal signals for the internal logic, and produce some reporters as output.

2.1 Gene network

We represent gene networks as a graph. The vertices are gene's that express a protein. The edge's are TFs that regulate the expression of the gene's, or reporters that are used as output signal. Fig. 2.1 shows an example of a OR gate gene network. Three inhibitors (TetR, LacI, and Cl) regulate the expression of the two gene's (the vertices). The reporter GFP is the output signal.

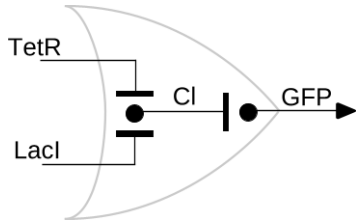


Figure 2.1: ***An OR gate gene network*** *- The gene network consists of two gene's (the vertices) that are regulated by three inhibitors (TetR, LacI, and Cl). The network produces the reporter GFP as output signal.*

An edge that regulates a gene can be either an inhibitor (edge in Fig. 2.2A) or an activator (both edge's in Fig.2.2B). An edge can have multiple source vertices, which means that the multiple gene's can produce the same protein (Fig.2.2A). Vertices can have multiple input edge's, which means that multiple TFs can regulate a gene (Fig.2.2B). An edge can also have multiple

destinations, which means that a TF can bind to multiple gene's (Fig.2.2C). Environmental signals, such as small molecules or light, can inhibit or activate a TF. This enables the introduction of environmental signals within a gene network. Fig.2.2C shows an edge (TF) inhibited by an environmental signal.

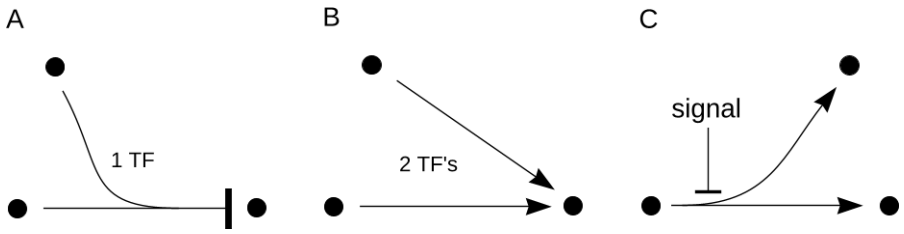


Figure 2.2: ***A*** *The TF (the edge) is an inhibitor that is produced by two gene's and inhibits one gene.* ***B*** *Both TFs are activators produced by different gene's. They regulate the same gene.* ***C*** *The produced TF regulates two different gene's. The TF can be inhibited by an environmental signal.*

2.2 Device

The Registry of standard biological parts was the first effort towards standardization in synthetic biology. A standard biological part is defined as a DNA sequence with some function. Since the foundation of the registry at MIT in 2003, many standard biological parts have been added. The parts in the registry meet the BioBrick™standard which makes it possible to ligate parts with a standard ligation protocol.

The most basic parts within the registry are the genetic parts. For our project we used four different kinds of genetic parts: promoters, ribosome binding sites (RBS), protein coding parts, and terminators. The function of a promoter is to recruit transcription machinery, i.e. polymerase. The DNA sequence will therefore contain a binding site for the transcription machinery. Binding of TFs to the operator sites can influence this. Therefore, next to the polymerase binding site, a promoter can also contain several TF binding sites, i.e. operators. The function of the RBS is to recruit translation machinery, i.e. ribosomes. The protein coding part encodes for the expressed protein. Finally. the terminator causes the polymerase to dissociate.

The genetic parts can be combined to form a cellular system. For example, an inverter, a system that produces a high output signal (high TF concentration) when the input signal is low and a low output signal when the input signal is high. Such a cellular system is called a device. Finally, a device can be used to build complex biological systems.

The next section describes the protein generator, a standard building block that we use for the design of the networks, which consist of the four genetic parts and therefore is a device. We design gene networks using the protein generator devices and the gene networks we design are thus also devices.

2.3 Protein generator

A protein generator device is build out of four genetic parts; a promoter part, a ribosome binding site part, a protein coding part and a terminator part (Fig. 2.3). Each of these parts has their own function and each of them determine some characteristics of the protein generating device. When having a database with genetic parts, different protein generators can be build.

Protein generators are used as basic building blocks to build gene networks. Fig. 2.4 shows the gene network from Fig. 2.1 together with a device that implements this it.

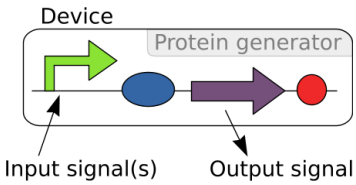


Figure 2.3: ***Protein generator*** *- This device consists of four genetic parts; a promoter (green), an RBS (blue), a protein coding part (purple), and a terminator (red). The symbols for the genetic parts are commonly used in the field of synthetic biology. The promoter binds TFs and thereby determines the input signal. The protein coding part encodes for the expressed protein and thereby determines the output signal.*

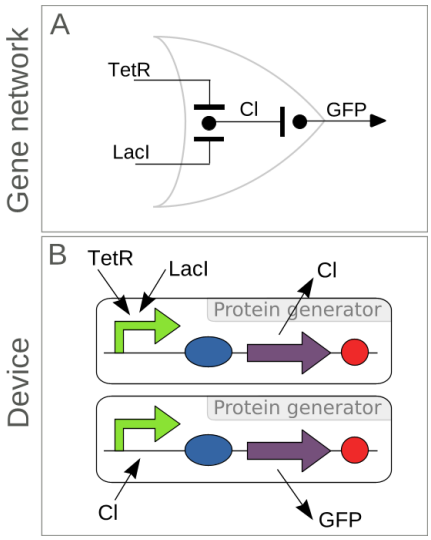


Figure 2.4: ***A*** *The gene network from Fig. 2.1.* ***B*** *The device implementation of the network in A.*

2.4 Modular gene networks

This section shows how modularity can be used to build gene networks using protein generators as basic building blocks. It shows how a AND gene network can be build using other logic gates. The example is based on the work of the 2008 Pavia iGEM team[1].

A logic NOT gate

The device shown in figure 2.5 has an operator that binds LacI tetramers and thus accepts LacI tetramer proteins as input signal. The protein coding part encodes for the LuxI protein, which is the output signal of this device. This simple device, which consists of only one protein generator, has the function of a logic NOT gate; LacI tetramer binding causes repression of LuxI expression and thus a high LacI tetramer concentration (i.e. a high input signal) causes a low LuxI concentration (i.e. a low output signal) and visa versa.

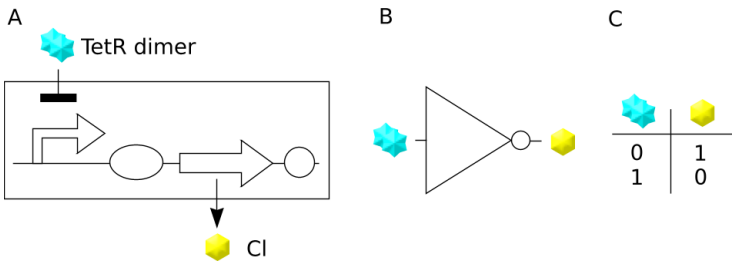


Figure 2.5: ***A biological NOT gate*** *- **A** The input signal, LacI tetramers, inhibit the expression of the output signal, LuxI. A high input signal (a high concentration of LacI tetramers) cause a low output signal (concentration of LuxI proteins) **B** The symbol of a logic gate as used in electrical engineering. **C** A truth table of the logic gate. A 1 denotes a high concentration of the signal and a 0 a low concentration.*

A logic AND gate

Figure 2.6 shows an example where protein dimerization is used to get a logic AND behaviour. Two input signals have to form heterodimers in order to bind to the operator and thereby activate the expression of the output signal. This means that there will only be a high output signal when both input signals are high, which is the typical behaviour of a logic AND gate.

A logic NAND device using the AND and NOT gates

These two logic devices can now be combined to get a more complex function. A module that acts as a logic NAND gate can for example be build using two NOT gate and one AND gate as depicted in figure 2.7. Incorporating the three logic gates as given in figure 2.7 into a cell will automatically result in the desired connection because of the 'fitting' signals. The first NOT gate produces LuxI proteins as output signal which in turn serves as input signal to the AND gate. The same applies to the second NOT gate which produces the LuxR protein which in turn serves as the second input of the AND gate.

A logic AND gate that responds to environmental signals

Finally we can add two devices that act as logic NOT gate to build an AND gate which takes two external signals, the ligands IPTG and aTc, as inputs and produces the reporter protein RFP as output (Figure 2.8). The output protein will only be expressed when both input signals are present, i.e. when both input signals are high.

[1] `http://2008.igem.org/Team:UNIV-Pavia`

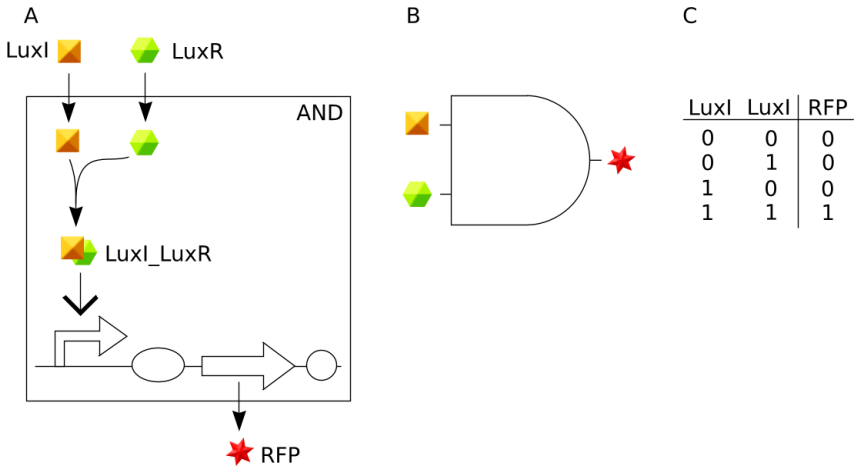


Figure 2.6: ***A biological AND gate*** *- **A** The input signals LacI and LuxI form a hetrodimer and bind as activator to the promoter. Both input signals must be high in order to get RFP expression, which is an AND behaviour. **B** A logic AND gate symbol as used in electrical engineering. **C** The truth table of an AND gate. The output is high only if both inputs are high.*

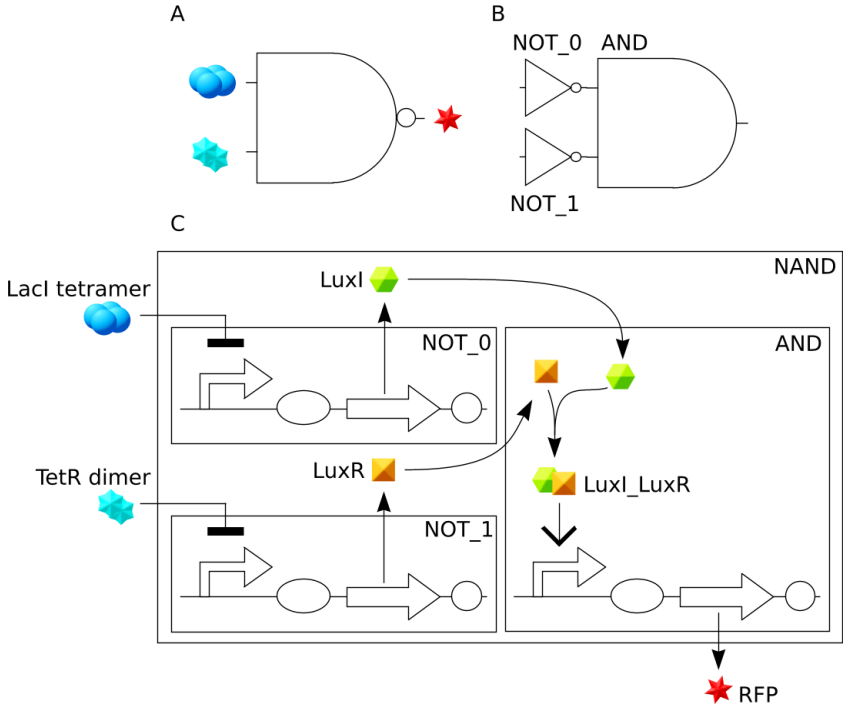


Figure 2.7: ***A biological NAND gate*** *- **A** the overall NAND symbol that takes LacI and TetR as inputs and produces RFP as output. **B** The NAND consists of two NOT gates that are connected to an AND gate. **C** The NAND gate consists of three protein generators. The produced proteins from the NOT gates bind as a TF to the promoter of the AND gate, this way the different gates are connected.*

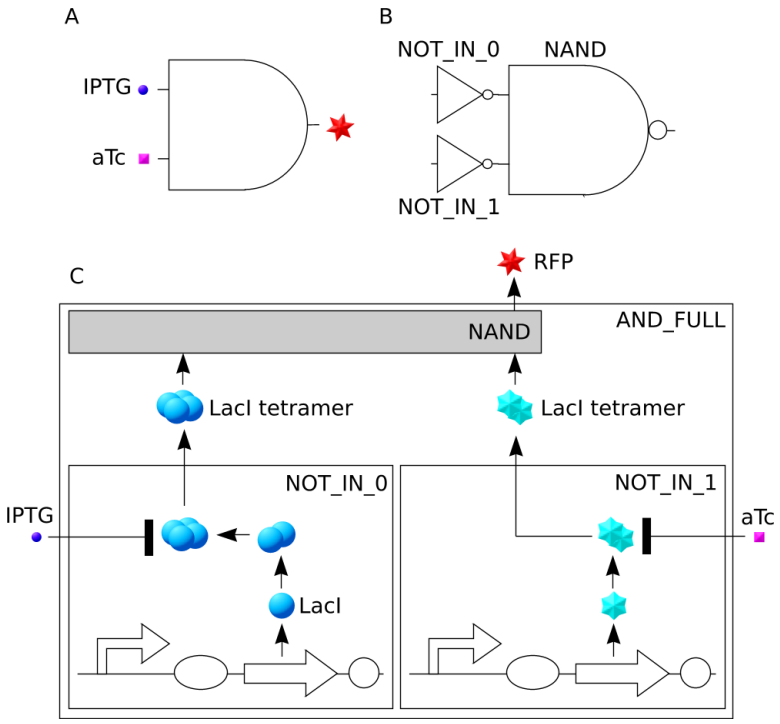


Figure 2.8: ***The final AND gate*** *- **A** the overall AND symbol that takes the small molecule IPTG and aTc as inputs and produces RFP as output. **B** The AND consists of two NOT gates that are connected to the NAND gate from Fig. 2.7. **C** The produced proteins from the NOT gates are the inputs to the NAND gate. The NOT gates take small molecules as input and the NAND produces RFP as output. This way we have a AND gate that can respond to environmental signals and produces a measurable output.*

We have now seen that protein generators can be used as basic building blocks to build gene networks that act as a logic gate. The TFs are the signal carriers within the network, small molecules can be used as external signal, and reporters as output signal.

2.5 Degree of standardization

Our aim is to design logic gene networks using modular logic gates. For the logic gates one can use different degrees of standardization.

Quantitatively characterized logic gates

A possible approach is the definition of a collection of well characterised logic gate devices. A major downside of this approach is that such a collection will soon become impractically large, because of the many different signals (TFs) that can be used. When having a device with only one input and one output and having 10 different signals available, this would already provide $10 \times 9 = 90$ different version of the same logic gate device.

When using well characterised logic gates, one needs to be sure that they also function correct when interconnected to other logic gates. Del Vecchio et al. (2008) proposes mechanisms that can be used to ensure this, but such mechanisms cost extra energy and will not result in an energetically optimal solution. This is a major downside, since energy is a costly product for a cell.

Qualitatively characterized, tunable logic gates

Another approach is the use of gene network templates, which are gene networks that provide the desired qualitative behaviour using some mechanism, but which can still be tuned by changing the genetic parts that are used to implement the network. A user can now simply combine logic gate templates and use an optimization algorithm to tune the devices so that the system provides the desired behaviour (which needs to be specified by the user). We have chosen to use this option for the design of logic gene networks.

2.6 Gene network template

A gene network template is a more general form of a gene network that hides the genetic parts that are used to build it. Fig. 2.9A shows an OR gate gene network template. It provides the network topology and indicates what kind of proteins are used as signal carriers. Fig. 2.9B shows a possible instance of this template. Notice that there are more instances possible. For example, *Cl* and *LacI* could also be the inputs and TetR the internal signal. Having more inhibitors available results in many more possible instances.

Fig. 2.9C and D show the device representation of the same situation. The template consists of two protein generators, one per vertex in the gene network. There are no genetic parts assigned yet, it is only indicated what properties the parts should have. Fig. 2.9D shows the same instance as in Fig. 2.9 B.

Gene network templates are modular. Fig. 2.10 shows that the OR gate from Fig. 2.9 is consists of a NOR gate followed by a NOT gate template. The templates for the NOR and the NOT gate can also be used for the design of other logic gene networks.

2.7 Logic gene network design method

The use of gene network templates is implemented into the design of logic gene networks as depicted in Fig. 2.11, which shows an overview of the design method as it is implemented in the developed software tool. The boxes are stored data and the arrows denote data transformations.

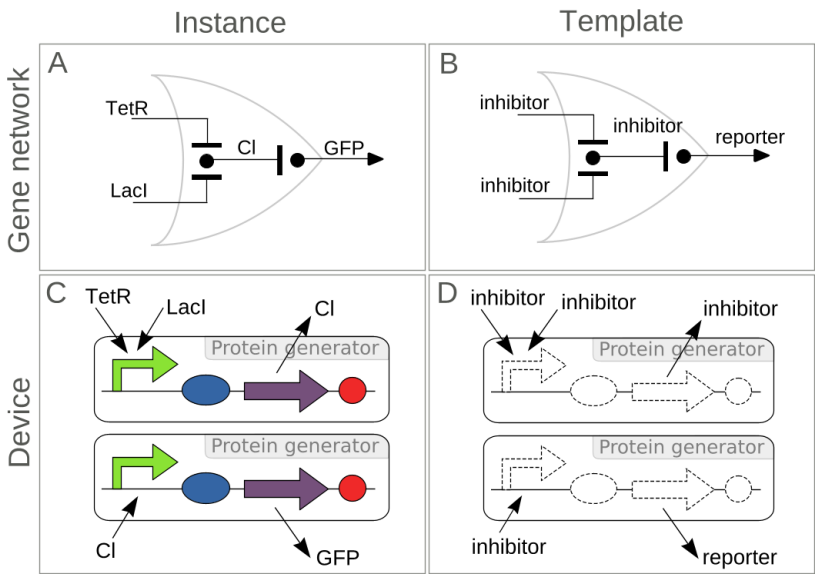


Figure 2.9: ***A and C*** *OR gate template in the form of a gene network and a device.* ***B and D*** *A possible instance of the OR gate template in the form of a gene network and a device again.*

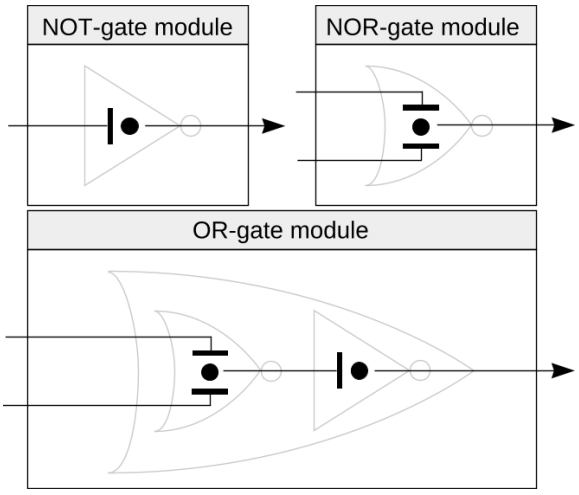


Figure 2.10: *Connection of a NOR template and a NOT template results in a OR template. This illustrates how logic gene network templates can be build in a modular way.*

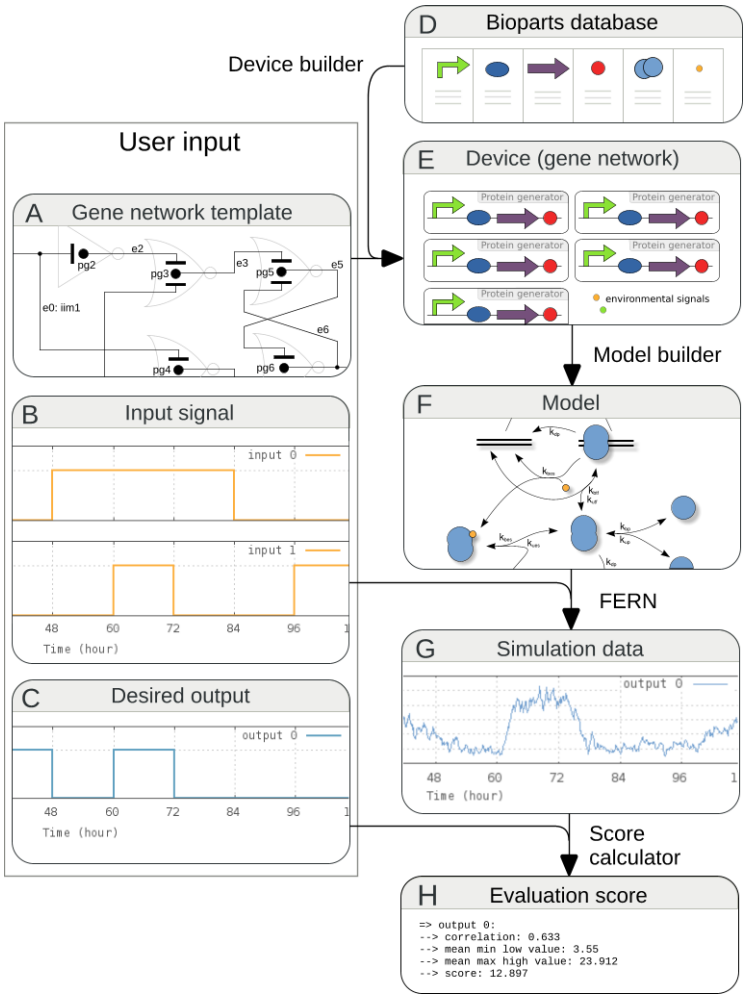


Figure 2.11: *An overview of software that implements the design method for the design of logic gene networks. The boxes are stored data and the arrows denote data transformations. The gene network builder uses the biopart database to translates gene network templates into gene network instances. A gene network instance, which is a device, can be translated to a model using the model builder. The FERN software package uses the model to run a stochastic simulation. The concentrations of the environmental signals (the input signals) are determined by the user defined inputs. Finally, the score calculator provides an evaluation score for the device based on a comparison of the simulation data to the user-defined desired output.*

The user specifies a gene network template, assigns environmental signals to the inputs and reporters to the outputs, and provides a binary timing diagram for each input signal and desired output signal. The *gene network builder* uses bioparts from a database to build gene network instances that implement the given template. The number of possible instances is in most cases very large. The *model builder* can translate a gene network instance, which is a device, into a model, which in turn can be used for stochastic simulation using the *FERN* framework. The concentration of the environmental signals during simulation is determined by the user-defined input signal. The result of the simulation is the molecule count of the output signal over time. The *score calculator* compares the simulation output with the user-defined output and turns that into a gene network evaluation score.

2.7.1 Gene network builder

To get from a gene network template to a possible gene network instance, represented as a device (so going from Fig. 2.9 A to D), the gene network builder assigns genetic parts from the bioparts database to each of the protein generators. The gene network template puts restrictions on the genetic parts that can be used to build the device. For example, the gene network template in Fig. 2.12 determines that the promoter must bind one inhibitor and protein coding part must encode for an inhibitor. Furthermore, to prevent crosstalk, transcription factors may only be used once within a network, putting further restrictions of the parts that can be used. As already mentioned, the number of possible device instantiations for a gene network template explodes already for a relatively simple network and small bioparts database.

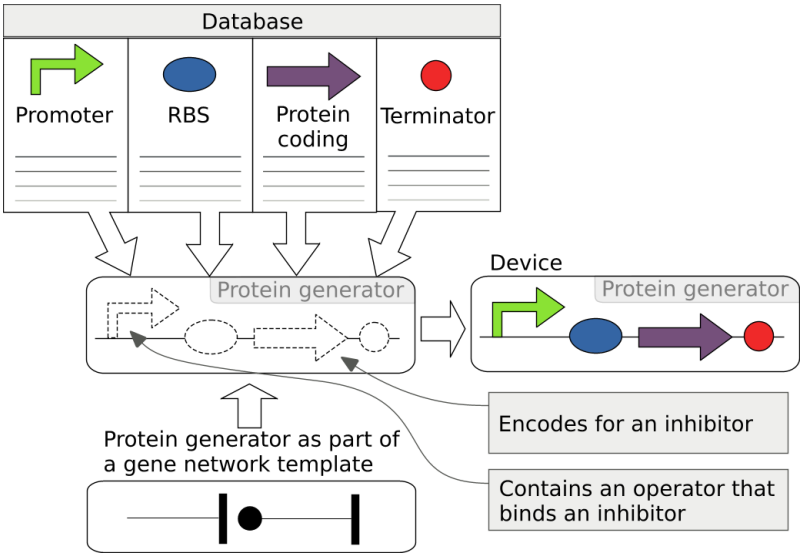


Figure 2.12: ***Gene network template to gene network instance*** *The transformation of a protein generator that is part of a gene network template to a protein generator device happens through the assignment of genetic parts from a bioparts database to the protein generator device, in which the gene network template puts restrictions on the parts that may be used. In this case the gene network template puts restrictions on the choice of the promoter and the protein coding part. The promoter must bind one inhibitor and the protein coding part must encode for an inhibitor. Since there can be multiple promoters that bind one inhibitor and multiple protein coding parts that encode for an inhibitor, multiple devices can be made for one gene network template.*

2.7.2 Model builder

The data transformation from a device to a model is done per protein generator. Each protein generator is translated into a standard model (Fig. 3.3). The models of the different protein generators are automatically connected as soon as the product of a translation reaction is also a reactant (possibly after a dimerization reactions) of a TF-DNA binding reaction (Fig. 2.13). This way, the TF expressed by one protein generator inhibits or activates the expression of another protein generator. Combining the reactions of the protein generator models will therefore automatically form a gene network model.

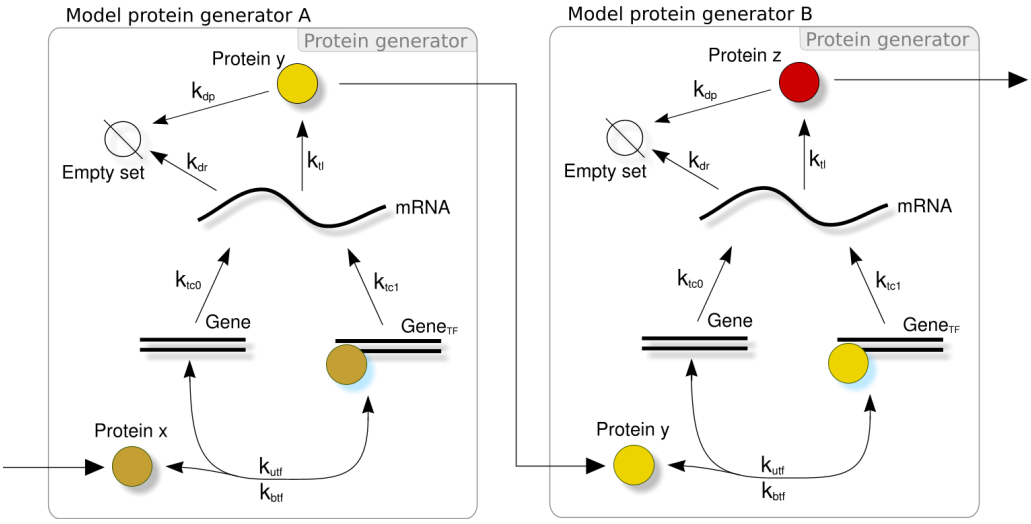


Figure 2.13: *Models of protein generators are automatically connected when a product of a translation reaction is also a reactant of TF binding reaction. In this case, protein y is a product of the translation reaction of protein generator A and also a reactant of the TF binding reaction of protein generator B. This way, combining the models of protein generator A and B automatically connects them. In other words, combining models of protein generators automatically results in a gene network model.*

The kinetic constants are stored in the bioparts database together with the parts that determine the constant. For example, a RBS determines the translation rate (k_{tl}) and therefore each RBS in the database stores an accompanying translation rate. Fig. 2.14 shows what kinetic constants are stored for which genetic parts. Next to the genetic parts, the database also contains proteins and environmental signals. Those parts also store some of the kinetic constants (Fig. 2.15). The model builder fetches the kinetic constants from the bioparts database and uses them for the model-building.

2.7.3 Stochastic simulation

The resulting model can be used for stochastic simulation. The input signal provided by the user determines at which time points the concentrations of the environmental signals are set to zero, in case of a binary '0', or a high value, in case of a binary '1'.

2.7.4 Score calculation

The score calculator compares the desired output to the simulation data in order to evaluate the performance of the device. The goal is to have an output signal that resembles the user-defined desired output. Furthermore, since we are building logic gene networks, it is desirable to have a maximal difference between a low signal and a high signal, in order to distinguish a logic 0 from a logic 1.

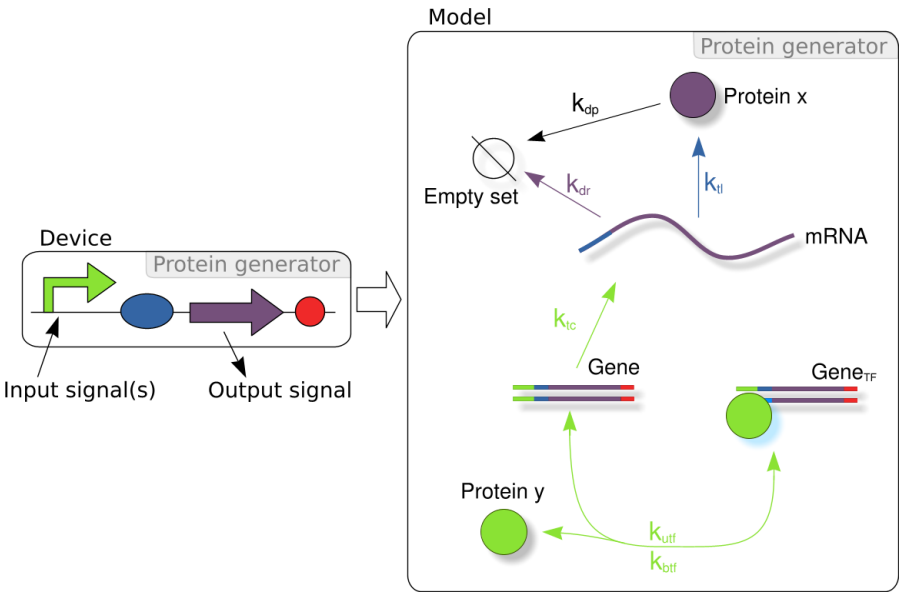


Figure 2.14: ***Device to model*** *The genetic parts that make up the protein generator device store the data that is needed to build a model for the device. The standard model for a protein generator is shown in which the colours indicate which genetic part stores what data. For example, the promoter (green) stores the transcription rates (k_{tc}) for the different gene states. Furthermore it stores the binding and unbinding rates of the TF(s) that can bind to it (k_{btf}, k_{utf}) and it stores a pointer to those TF(s). The protein degradation rate k_{dp} is not coloured, because it is not stored for a genetic part, but for a protein. The kinetic constants that are stored for a protein are shown in Fig. 2.15*

The performance is evaluated based on three measures; the correlation between the desired and simulated output, the concentration of a high signal, and the concentration of a low signal. The objective for a well performing device is maximization of the correlation, maximization of the high signal concentration, and minimization of the low signal concentration. Maximization of the correlation maximizes the resemblance of the two signals. Since the desired output signal is a block signal, this also optimizes for fast switching times. Maximization of the high signal and minimization of the low signal maximizes the separability of low and high signals.

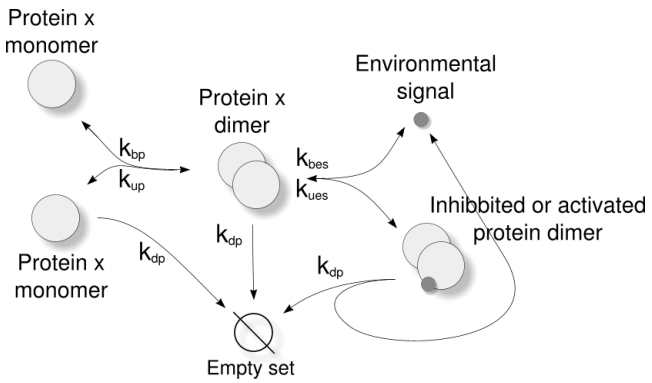


Figure 2.15: *This figure shows what data is stored for the proteins in the biopart database. Each protein stores a degradation rate k_{dp}. Dimer or tetramer proteins also store pointers to the two sub-proteins and the (un)binding rates of these proteins (k_{bp}, k_{up}). Proteins that bind an environmental signal also store a (un)binding rate of the environmental signal (k_{bes}, k_{ues}).*

Chapter 3

Methods

This chapter provides a short explanation transcription regulation, the regulation mechanism that we have used to design biological systems. It also provides an introduction to Gillespie's exact stochastic simulation algorithm, which we have used to simulate gene expression. Finally, it describes the used model, which is based on the model from Hooshangi et al. (2005).

3.1 Transcription regulation

Gene expression is the process of turning the information encoded on a gene into a synthesized functional product, which can be a protein or a functional RNA. This process can be subdivided into different steps, such as transcription, translation, and post-translational modification. The expression of a gene can be regulated using different mechanisms which can act on the different steps of the gene expression process.

The most common and most well known regulation system is transcription regulation. Proteins can bind to a gene causing an increase or decrease of the expression of the gene, i.e. the production of the functional product. Binding of the protein to the gene is called activation in the former case, and inhibition in the latter case. The location on the gene where the proteins bind is called an operator and the proteins that bind to this operators are called transcription factors (TF). A TF is more specifically called an inhibitor when the binding of the TF to the operator causes inhibition and it is called an activator when it causes activation of the expression. Multiple TF producing gene's can form a so called gene network in which the different gene's regulate each other. Some TFs can be inhibited or activated by environmental signals such as small molecules. This way a cell can respond to changing environmental conditions.

The binding of a TF to the a gene and the binding of an environmental signal to a TF are the two mechanisms that we use to build synthetic gene networks.

3.2 Stochastic simulation

Measurements have shown that gene expression is a stochastic process Elowitz et al. (2002). This stochasticity, also called noise, has multiple sources and it is not yet clear what the exact sources are and to which extend they influence the overall noise. The noise in the protein concentration can be subdivided into two components: the intrinsic noise, originating from the gene expression process itself, and the extrinsic noise, which is caused by the variation of external factors, such as the number of ribosomes or the number of transcription factors.

It is very common to use a deterministic approach, using systems of ODEs, for the simulation of gene expression. Although stochasticity is neglected and molecule numbers are represented using continuous variables instead of discrete integer numbers, this approach can accurately predict the behaviour of systems in which the number of molecules per species is high. But as soon as the molecule count of at least one of the molecular species is low, stochasticity and discreteness

can have a major influence and can therefore not be neglected anymore. It was also shown that this stochasticity could show a behaviour that can not be observed with deterministic simulation (Kaern et al., 2005). The fact of having stochastic behaviour which can be of influence to the dynamical behaviour of the systems opts for the use of a stochastic model and stochastic simulation to predict the dynamical behaviour of gene expression.

The rest of this section provides some basics about stochastic simulation, which is mainly based on a paper by Gillespie (2007).

3.2.1 Chemical system

Under the assumption of having a well stirred, thermally equilibrated chemical system with a constant volume Ω. Consider a chemical system with N molecular species $\{S_1, \ldots, S_N\}$ and X_i an integer value denoting the number of molecules of each species S_i. The dynamic state of the system is defined by $\mathbf{X}(t) = (X_1(t), \ldots, X_N(t))$ which evolutes through time due to the chemical interaction of the N molecular species through M reaction channels $R_1, \ldots, R_M$.

In order to predict the behaviour of such a system the goal is to determine the evolution of the state $\mathbf{X}$ of the system from a given initial state $\mathbf{X}(t_0)$. Such a system is known to evolve stochastically (Gillespie (2007)).

3.2.2 Deterministic approach

Most commonly, chemical kinetics are being analysed using continuous variables (molecule numbers) that evolve deterministically using the so called reaction-rate equation (RRE), which is a set of coupled ordinary differential equations (ODEs) of the form:

$$\frac{dX_i(t)}{dt} = f_i(X_1, \ldots, X_N) \; (i = 1, \ldots, N)$$

in which f_i is a combined function of all the reactions that affect X_i, which is a *continuous* variable.

Although stochasticity is neglected and molecule numbers are represented using continuous variables instead of discrete integer numbers, this approach can accurately predict the behaviour of systems in which the number of molecules per species is high. But as soon as the molecule count of at least one of the molecular species is low, stochasticity and discreteness can have a major influence and can therefore not be neglected anymore.

3.2.3 Stochastic approach

As Gillespie (2007) states it: "Stochastic chemical kinetics attempts to describe the time evolution of a well-stirred chemically reacting system in a way that takes honest account of the system's discreteness and stochasticity."

With this approach reactions R_j are characterized mathematically by two quantities: The *state-change vector* $\nu_j \equiv (\nu_{1j}, \ldots, \nu_{Nj})$, where ν_{ij} is the change in the S_i molecular population caused by one R_j reaction. The state-change vector determines the new state of the system when reaction R_j occurs. When the system is in state $\mathbf{x}$, the system will jump to state $\mathbf{x} + \nu_j$ when reaction R_j occurs.

And the *propensity function* a_j, which is defined as:

$$a_j(\mathbf{x}) = c_j h_j(\mathbf{x}) \tag{3.1}$$

in which c_j is the probability rate constant, the probability that a randomly selected combination of R_j reactant molecules in Ω react in a unit time period and h_j is defined as the number of distinct combinations of R_j reactant molecules in Ω (Cai et al. (2006)).

There are two elemental types of reactions that can be specified in a stochastic model: unimolecular reactions, with only one reactant, and bimolecular reactions, with two reactants. Higher order reactions are represented as a sequence of these elemental types of reactions. This provides

three different possible reactions, each with their own specific h_j and thus their own specific propensity function a_j:

$$\begin{aligned} S_1 \rightarrow \text{product(s)}, \qquad & a_j(\mathbf{x}) = c_j x_1, \\ S_1 + S_2 \rightarrow \text{product(s)}, \qquad & a_j(\mathbf{x}) = c_j x_1 x_2, \\ S_1 + S_1 \rightarrow \text{product(s)}, \qquad & a_j(\mathbf{x}) = c_j \tfrac{1}{2} x_1 (x_1 - 1). \end{aligned}$$

The propensity function is defined in such a way that

$$\begin{aligned} a_j(\mathbf{x})dt \quad \triangleq \quad & \text{the probability, given } \mathbf{X}(t) = \mathbf{x}, \text{ that one } R_j \text{ reaction will occur} \qquad (3.2) \\ & \text{somewhere inside } \Omega \text{ in the next infinitesimal time interval } [t, t + dt), \qquad (3.3) \end{aligned}$$

which is the fundamental premis of stochastic chemical kinetics (Gillespie (2007)). The algorithm used for stochastic simulation, as described in Appendix 3.2.5, is based on this premise.

3.2.4 Relation between stochastic and deterministic approach

It can be shown that the probability rate constant c_j of the stochastic approach has the following relation to the reaction-rate constant k_j of the deterministic approach:

$$\begin{aligned} c_j &= k_j \text{ (unimolecular reaction)} \\ c_j &= \frac{k_j}{\Omega} \text{ (bimolecular reaction with two different species)} \\ c_j &= \frac{2k_j}{\Omega} \text{ (bimolecular reaction with the same species)} \end{aligned}$$

Gillespie (2007) explicitly states that: "These results should not be taken to imply that the mathematical forms of the propensity function are just heuristic extrapolation of the reaction rates of deterministic chemical kinetics. The propensity functions are grounded in molecular physics, and the formulas of deterministic chemical kinetics are approximate consequences of the formulas of stochastic chemical kinetics, not the other way around."

3.2.5 Stochastic simulation algorithm

The probability of a chemical system being in state $\mathbf{x}$ at time t can be inferred using

$$P(\mathbf{x}, t \mid \mathbf{x}_0, t_0) \triangleq \text{Prob}\{\mathbf{X}(t) = \mathbf{x}, \text{given} \mathbf{X}(t_0 = \mathbf{x}_0)\} \qquad (3.4)$$

This equation can be turned into a time-evolution version applying the laws of probability to equation 3.2.

$$\frac{\delta P(\mathbf{x}, t \mid \mathbf{x}_0, t_0)}{\delta t} = \sum_{j=1}^{M} [a_j(\mathbf{x} - \nu_j) P(\mathbf{x} - \nu_j, t \mid \mathbf{x}_0, t_0) - a_j(\mathbf{x}) P(\mathbf{x}, t \mid \mathbf{x}_0, t_0)] \qquad (3.5)$$

Equation 3.5 is called the Chemical Master Equation (CME) and it can be shown that this is actually a set of coupled ODE's. In practice this equation can be solved analytically only in a few cases.

Solving the CME results in the probability density function of $\mathbf{X}(t)$. The stochastic simulation algorithm discussed in this sections takes random samples of $\mathbf{X}(t)$ to simulate a trajectory of $\mathbf{X}(t)$ versus t. Instead of $P(\mathbf{x}, t \mid \mathbf{x}_0, t_0)$ another probability, which is defined as,

$$p(\tau, j \mid \mathbf{x}, t)d\tau = \text{the probability, given } \mathbf{X}(t) = \mathbf{x}, \text{ that the next reaction in the system} \quad (3.6)$$
$$\text{will occur in the infinitesimal time interval } [t+\tau, t+\tau+d\tau), \text{ and will} \quad (3.7)$$
$$\text{be an } R_j \text{ reaction.} \quad (3.8)$$

is used for the random sampling, in which τ is a random variable denoting the time to the next reaction and j a random variable denoting the index of the next reaction. This equation can be turned into the following equation:

$$p(\tau, j \mid \mathbf{x}, t) = a_j(\mathbf{x})\exp(-a_0(\mathbf{x})\tau), \quad (3.9)$$

where

$$a_0(\mathbf{x}) \stackrel{\triangle}{=} \sum_{j'=1}^{M} a_{j'}(\mathbf{x}). \quad (3.10)$$

again by applying the laws of probability to equation 3.2. Equation 3.9 is the basis for the simulation algorithm.

Equation 3.9 indicates that τ is an exponential random variable and j is a statistically independent integer random variable. Samples of these two random variables can be generated using two random numbers, r_1 and r_2, from the uniform distribution in the unit interval using:

$$\tau = \frac{1}{a_0(\mathbf{x})} ln(\frac{1}{r_1}), \quad (3.11)$$

$$j = \text{the smallest integer satisfying } \sum_{j'=1}^{j} a_{j'(\mathbf{x})} > r_2 a_0(\mathbf{x}) \quad (3.12)$$

The stochastic simulation algorithm (SSA) from Gillespie, that simulates the evolution of a process, i.e. generates a numerical realization of $\mathbf{X}(t)$ works as follows:

1. Initialization, $t = t_0$, $\mathbf{x} = \mathbf{x_0}$
2. Evaluate $a_j(\mathbf{x})$ for each reaction and the sum $a_0(\mathbf{x})$
3. Generate τ and j (for example using equation 3.11 and 3.12)
4. Take a step: $t = t + \tau$ and $\mathbf{x} = \mathbf{x} + \nu_j$
5. Go to step 2, or stop the simulation.

3.2.6 Example Simulation

When taking the model from figure 3.2 with gene, mRNA, protein, and empty_set as the set of species $S = \{S_1, S_2, S_3, S_4, \}$. A set of reactions:

$$\begin{aligned} R_1 = S_1 &\xrightarrow{c_1} S_1 + S_2, \\ R_2 = S_2 &\xrightarrow{c_2} S_2 + S_3, \\ R_3 = S_2 &\xrightarrow{c_3} S_4, \\ R_4 = S_3 &\xrightarrow{c_4} S_4. \end{aligned}$$

with corresponding flux vectors:

$$\begin{aligned}\nu_1 &= [0,1,0,0]^T,\\ \nu_2 &= [0,0,1,0]^T,\\ \nu_3 &= [0,-1,0,1]^T,\\ \nu_4 &= [0,0,-1,1]^T.\end{aligned}$$

and (arbitrary) probability rate constants:

$$\begin{aligned}c_1 &= 0.03\tfrac{1}{sec},\\ c_2 &= 0.01\tfrac{1}{sec},\\ c_3 &= 0.001\tfrac{1}{sec},\\ c_4 &= 0.2\tfrac{1}{sec}.\end{aligned}$$

Notice that all the reactions are unimolecular. Therefore the propensity function of each of the reactions has the same form (section 3.2.3):

$$\begin{aligned}a_1(\mathbf{x}) &= c_1 x_1,\\ a_2(\mathbf{x}) &= c_2 x_2,\\ a_3(\mathbf{x}) &= c_3 x_2,\\ a_4(\mathbf{x}) &= c_4 x_3.\end{aligned}$$

Initialization

The system is initialized at time is 0 and in the state with one gene species and zero mRNA, protein, and empty_set species.

$$\begin{aligned}t &= 0,\\ \mathbf{x} &= [1,0,0,0].\end{aligned}$$

Iteration 1

- Evaluation of all propensity functions and the sum of them.

$$\begin{aligned}a_1 &= 0.03 * 1 = 0.03,\\ a_2 &= 0.01 * 0 = 0,\\ a_3 &= 0.001 * 0 = 0,\\ a_4 &= 0.2 * 0 = 0,\\ a_0 &= 0.03 + 0 + 0 + 0 = 0.03.\end{aligned}$$

- Generation of τ and j with $r_1 = 0.34$ and $r_2 = 0.88$.

$$\begin{aligned}\tau &= \frac{1}{0.03} ln(\frac{1}{0.34}) = 35.96,\\ j &= 1 \text{ (because } 0.03 > 0.88 * 0.03).\end{aligned}$$

- Take a step

$$\begin{aligned}t &= t + \tau = 0 + 35.96 = 35.96,\\ \mathbf{x} &= \mathbf{x} + \mathbf{v}_j = [1,0,0,0]^T + [0,1,0,0]^T = [1,1,0,0]^T.\end{aligned}$$

We are now in a new state with still one gene molecule (which will never change, since no reaction consumes or produces it) and one mRNA molecule (which is produces by reaction 1) at time $t = 35.96$ seconds.

Iteration 2

- Evaluation of all propensity functions and the sum of them.

$$\begin{aligned} a_1 &= 0.03 * 1 = 0.03, \\ a_2 &= 0.01 * 1 = 0.01, \\ a_3 &= 0.001 * 1 = 0.001, \\ a_4 &= 0.2 * 0 = 0, \\ a_0 &= 0.03 + 0.01 + 0.001 + 0 = 0.041. \end{aligned}$$

- Generation of τ and j with $r_1 = 0.55$ and $r_2 = 0.79$.

$$\begin{aligned} \tau &= \frac{1}{0.041} ln(\frac{1}{0.55}) = 14.58, \\ j &\neq 1 \text{ (because } 0.03 < 0.79 * 0.041), \\ j &= 2 \text{ (because } 0.04 > 0.79 * 0.041). \end{aligned}$$

  Because of the high r_2, the reaction chosen is R_2, while from the reaction rates (once per 33.33 seconds for reaction 1 and once per 100 seconds for reaction 2) one would expect that the first reaction would be chosen.

- Take a step

$$\begin{aligned} t &= t + \tau = 35.96 + 14.58 = 50.54, \\ \mathbf{x} &= \mathbf{x} + \mathbf{v}_j = [1, 1, 0, 0]^T + [0, 0, 1, 0]^T = [1, 1, 1, 0]^T. \end{aligned}$$

We are now in a state with one gene, one mRNA, and one protein molecule at time $t = 50.54$ seconds.

Iteration 3

- Evaluation of all propensity functions and the sum of them.

$$\begin{aligned} a_1 &= 0.03 * 1 = 0.03, \\ a_2 &= 0.01 * 1 = 0.01, \\ a_3 &= 0.001 * 1 = 0.001, \\ a_4 &= 0.2 * 1 = 0.2, \\ a_0 &= 0.03 + 0.01 + 0.001 + 0.2 = 0.241. \end{aligned}$$

- Generation of τ and j with $r_1 = 0.21$ and $r_2 = 0.25$.

$$\begin{aligned} \tau &= \frac{1}{0.241} ln(\frac{1}{0.21}) = 6.48, \\ j &\neq 1 \text{ (because } 0.03 < 0.25 * 0.241), \\ j &\neq 2 \text{ (because } 0.04 < 0.25 * 0.241), \\ j &\neq 3 \text{ (because } 0.041 < 0.25 * 0.241), \\ j &= 4 \text{ (because } 0.241 > 0.25 * 0.241). \end{aligned}$$

- Take a step

$$\begin{aligned} t &= t + \tau = 50.54 + 6.48 = 57.02, \\ \mathbf{x} &= \mathbf{x} + \mathbf{v}_j = [1, 1, 1, 0]^T + [0, 0, -1, 1]^T = [1, 1, 0, 1]^T. \end{aligned}$$

One protein is being degraded which brings us in a state with one gene, one mRNA, and no protein molecules at time $t = 57.02$ seconds.

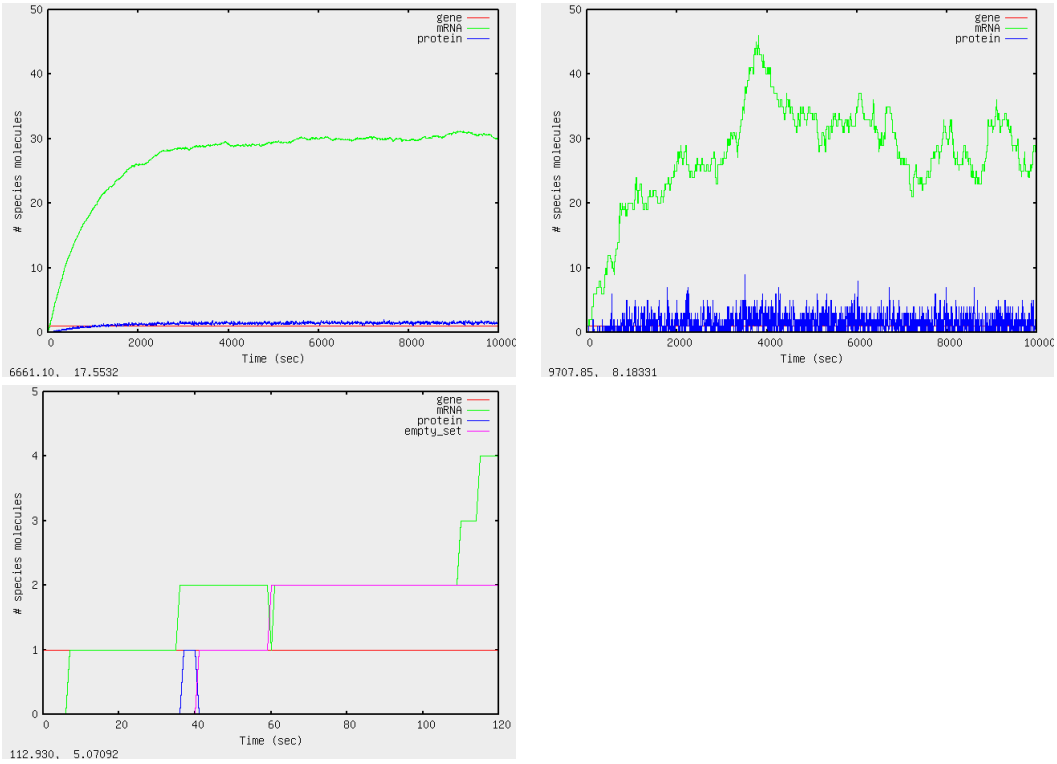


Figure 3.1: *Stochastic simulation of a simple gene expression model (figure 3.2) using arbitrary reaction rates in* $\frac{1}{s}$*. The transcription rate is* 0.03*, the translation rate is* 0.01*, the mRNA degradation rate is* 0.001*, and the protein degradation rate is* 0.2*.* ***A.*** *The average molecule count over 100 simulation runs.* ***B.*** *The result of one simulation run.* ***C.*** *Zoomed in on the first 120 seconds of B.*

full simulation

Figure 3.1 shows the results of running a simulation for the model and the parameters as given at the beginning of this section. The end time of the simulation is set to $t = 10.000$ seconds. The plot on the top left shows the average over 100 simulation runs. This result approximates the deterministic solution. The plot on the top right shows the result of running one simulation run and clearly shows the stochasticity in the production of the mRNA and protein. The plot on the bottom left is zoomed in on the first 120 seconds of the plot on the right top. Here we can clearly observe the following 8 reactions (so the first 8 iterations of the algorithm) that take place at the beginning of the simulation.

1. $j = 1, \quad t = 6.61$ (transcription)
2. $j = 1, \quad t = 35.81$ (transcription)
3. $j = 2, \quad t = 36.46$ (translation)
4. $j = 4, \quad t = 40.14$ (degradation protein)
5. $j = 3, \quad t = 59.84$ (degradation mRNA)
6. $j = 1, \quad t = 60.18$ (transcription)
7. $j = 1, \quad t = 109.88$ (transcription)
8. $j = 1, \quad t = 114.11$ (transcription)

3.3 Gene expression model

As stated in section 3.2, only two elementary types of reactions can be specified for a stochastic model: unimolecular reactions, with only one reactant, and bimolecular reactions, with two reactants.

3.3.1 Basic gene expression model

It was shown that a simple single gene expression model (figure 3.2) with a set of four species $S = \{\text{gene}, \text{mRNA}, \text{protein}, \text{empty set}\}$ and a set of four reactions $R =$

$$\text{gene} \xrightarrow{k_{tc}} \text{gene} + \text{mRNA}, \tag{3.13}$$

$$\text{mRNA} \xrightarrow{k_{tl}} \text{mRNA} + \text{protein}, \tag{3.14}$$

$$\text{mRNA} \xrightarrow{k_{dr}} \text{empty set}, \tag{3.15}$$

$$\text{protein} \xrightarrow{k_{dp}} \text{empty set}. \tag{3.16}$$

can accurately predict experimentally observed gene expression behaviour (Hooshangi et al., 2005).

The left side of each reaction specifies the reactants which are turned into the products on the right side when the reaction occurs. The rate constants determine the rate at which the reactions occur. Notice that for the reactions 3.13 and 3.14 the reactant is also one of the products.

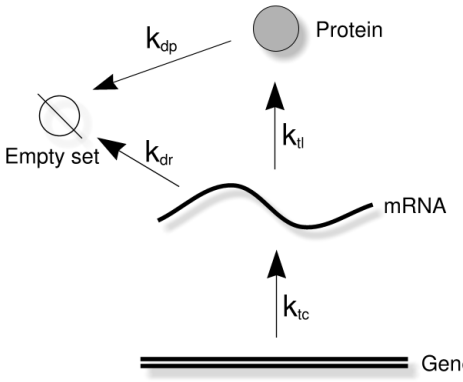


Figure 3.2: *The transcription of mRNA and the translation of mRNA to protein are both modelled as a single reaction, which means that mRNA molecules and proteins are being produced in a single time step. Also the degradation of both mRNA molecules and proteins is modeled as a single reaction. The parameters (k) are the rate constants which define the probability that a randomly selected combination of R_j reactant molecules in Ω react in a unit time period*

3.3.2 Regulated gene expression model

Hooshangi et al. (2005) has used an extended version of the just described basic gene expression model that includes also transcription regulation features. It was shown that this model can be used to accurately predict transcription regulated gene expression. Figure 3.3 shows a model that contains extra reactions that enable transcription regulation. The additional reactions are protein dimer formation, the binding of a TF to a gene, and the activation or deactivation of TFs by environmental signals, e.g. the deactivation of an inhibitor (which is a TF) through the binding of an inducer (the environmental signal). Environmental signals can be small molecules, but also light, temperature or pressure. In case of Fig. 3.3 the environmental signal inhibits the TF, while in Fig. 3.4 the environmental signal activates the TF.

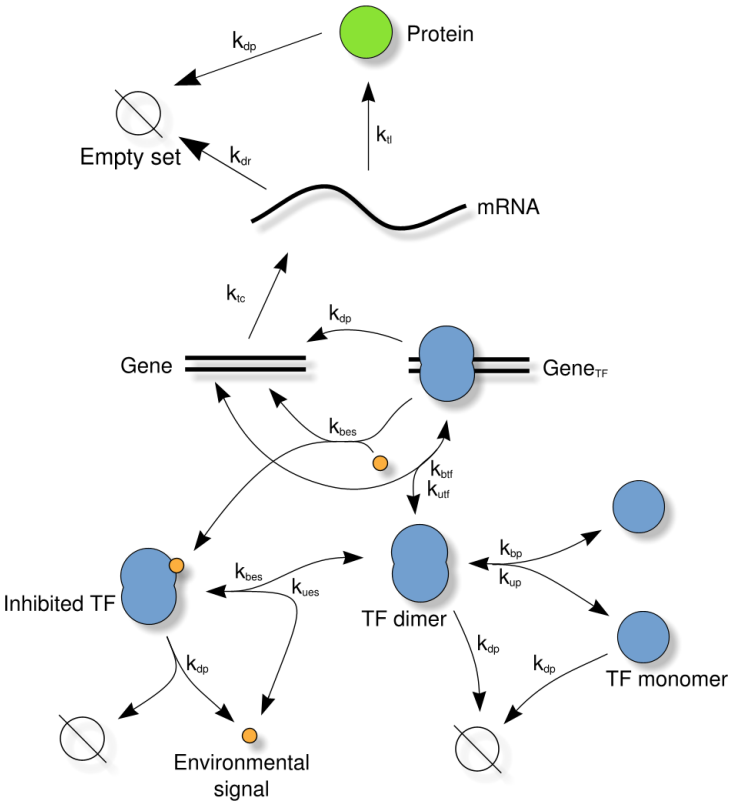


Figure 3.3: *Regulated gene expression model based on the model used by Hooshangi et al. (2005) in which the environmental signal acts as an inhibitor, disabling the TF to bind to the gene.*

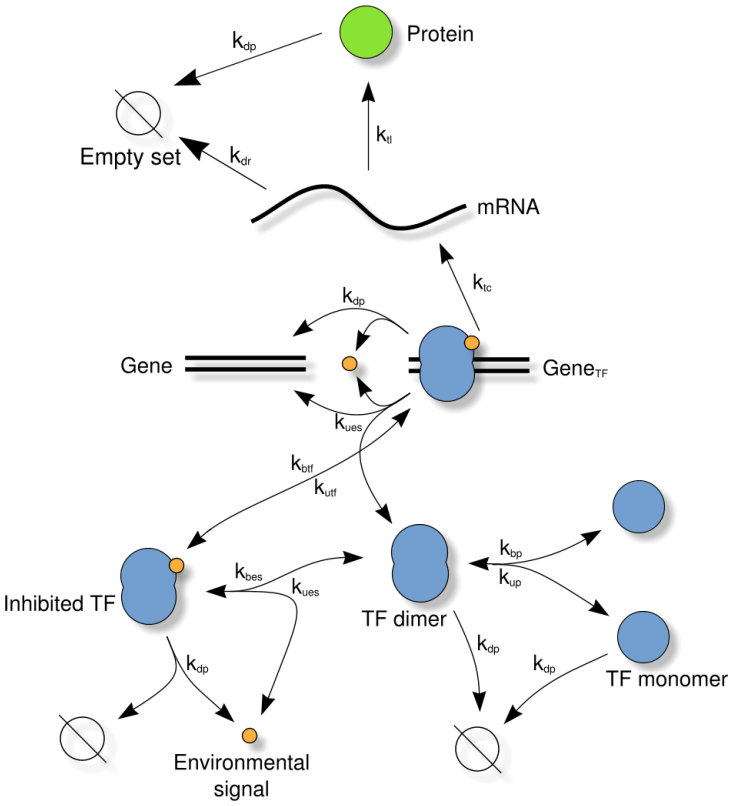


Figure 3.4: *Regulated gene expression model in which the environmental signal acts as an activator, enabling the TF to bind to the gene.*

Transcription and translation

Compared to the basic model, an extra transcription reaction is possible (Eq. 3.18). This reaction can occur when an activator is bound to the gene and thereby enables the transcription reaction. In Fig. 3.3 the TF is an inhibitor and therefore Eq. 3.17 occurs. In Fig 3.4 the TF is an activator and therefore Eq. 3.18 occurs.

$$\text{gene} \xrightarrow{k_{tc}} \text{gene} + \text{mRNA}, \tag{3.17}$$

$$\text{gene}_{TF} \xrightarrow{k_{tc}} \text{gene}_{TF} + \text{mRNA}, \tag{3.18}$$

$$\text{mRNA} \xrightarrow{k_{tl}} \text{mRNA} + \text{protein}. \tag{3.19}$$

Transcription factor binding

In contrast to the basic model, the gene can be in different states. In figure 3.3 the gene contains only one operator that can bind a TF. This gene can be in one of two states; *gene*, the state where no TF is bound to the gene, or *gene_TF*, the state where a TF is bound to it. Each state has its own transcription rate which enables the modeling of repression and activation. Note that a gene can also have multiple operators which would result in more states. In case of n different operators there will be 2^n different states, in case there is no competitive TF binding, i.e. the operators do not overlap so that each TF is able to bind simultaneously.

$$\text{gene} + \text{TF} \xrightarrow{k_{btf}} \text{gene}_{TF}, \tag{3.20}$$

$$\text{gene}_{TF} \xrightarrow{k_{utf}} \text{gene} + \text{TF}. \tag{3.21}$$

Dimer formation

Another extension is the formation of protein dimers and tetramers. This is needed because many TFs are dimers or tetramers. In both Fig. 3.3 and 3.4 the TF form dimers. But a model in which the TF act as monomer or tetramer is also possible.

$$\text{monomer} + \text{monomer} \xrightarrow{k_{bp}} \text{dimer}, \tag{3.22}$$

$$\text{dimer} \xrightarrow{k_{up}} \text{monomer} + \text{monomer}. \tag{3.23}$$

Environmental signals

Environmental signals can disable or enable the binding of a TF to an operator, i.e. the signals can activate or inhibit TFs. Note that in case of disabling TFs, a signal can also disable a TF which is bound to an operator and thereby causing the TF to dissociate from the DNA.

$$\text{TF} + \text{es} \xrightarrow{k_{bes}} \text{TF}_{es}, \tag{3.24}$$

$$\text{TF}_{es} \xrightarrow{k_{ues}} \text{TF} + \text{es}. \tag{3.25}$$

In case of an inhibiting environmental signal the following reaction can occur. The environmental signal binds to the TF, thereby forcing its dissociation from the gene.

$$\text{gene}_{TF} + \text{es} \xrightarrow{k_{bes}} \text{gene} + \text{TF}_{es}. \tag{3.26}$$

In case of an activating environmental signal the following reaction can occur. The environmental signal dissociates from the TF, thereby causing TF dissociation from the gene

$$\text{gene}_{TF} \xrightarrow{k_{bes}} \text{gene} + \text{TF} + \text{es}. \tag{3.27}$$

Degradation

Finally there are a number of degradation reactions.

$$\text{mRNA} \xrightarrow{k_{dr}} \text{empty set}, \tag{3.28}$$

$$\text{protein} \xrightarrow{k_{dp}} \text{empty set}, \tag{3.29}$$

$$\text{TF}_{es} \xrightarrow{k_{dp}} \text{empty set} + \text{es}. \tag{3.30}$$

A TF that is bound to the gene can also degrade. This provides two reactions, one in case of an inhibiting environmental signal, and one in case of an activating environmental signal.

$$\text{gene}_{TF} \xrightarrow{k_{dp}} \text{empty set} + \text{gene}, \tag{3.31}$$

$$\text{gene}_{TF_{es}} \xrightarrow{k_{dp}} \text{empty set} + \text{gene} + \text{es}. \tag{3.32}$$

3.4 Model validation

To validate the model we have modeled the first three cascading circuits from Hooshangi et al. (2005) (Fig. 3.5). The results are shown in Fig. 3.6. The switching behaviour is identical, but there is a difference in the maximum concentration. This is caused by the fact that we used a lacI tetramer, while Hooshangi used dimers for all TFs.

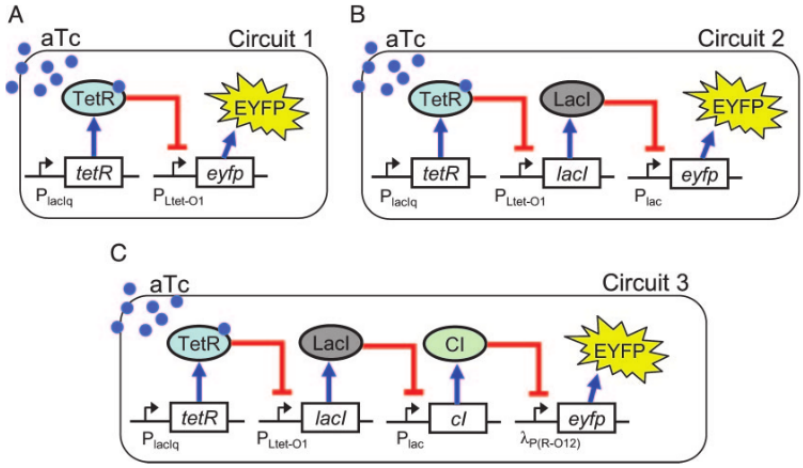


Figure 3.5: *The three cascading gene networks from Hooshangi et al. (2005).*

We have also build the repressilator from Elowitz and Leibler (2000). Fig. 3.7 shows the expected oscillating behaviour. Four different simulation runs show different results because of stochasticity.

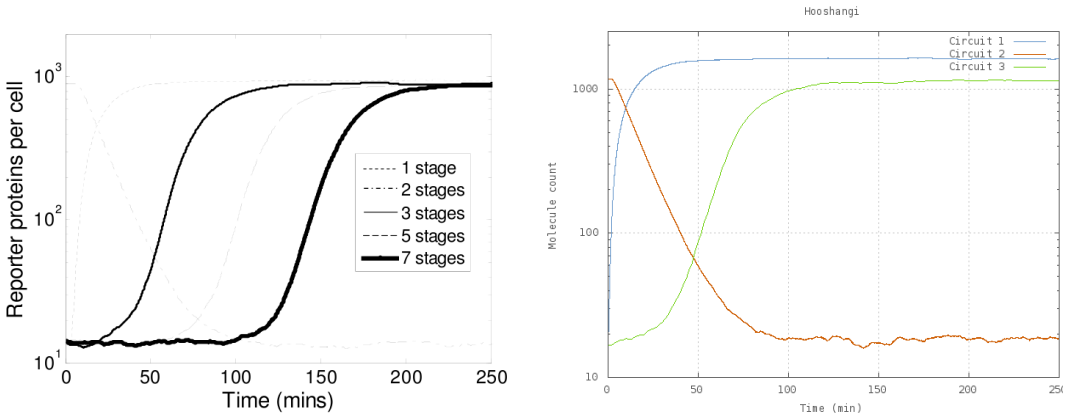


Figure 3.6: *Simulation of gene network cascades from Hooshangi et al. (2005). The simulation results of Hooshangi are on the right and the simulation results of our model on the right.*

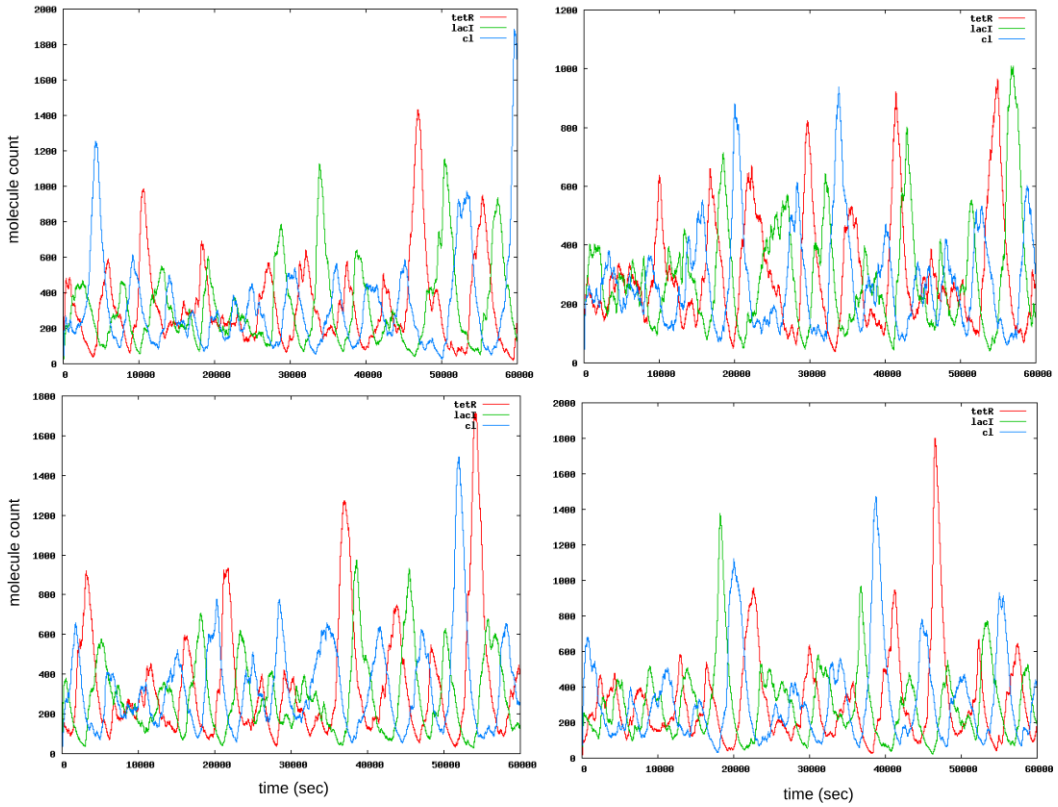


Figure 3.7: *Simulation results of the repressilator from Elowitz and Leibler (2000). The result of four different simulation runs are shon. Stochasiticity causes different results for the different runs.*

Chapter 4

Implementation

The software is written in *Java* using *eclipse* as software development tool. The software uses three *Java* libraries; *Fern* for stochastic simulation (Erhard et al., 2008), *colt*[1] for basic mathematics, and *jdom*[2] for XML parsing.

The software consists of four packages: the *biopart* package, the *gene_network* package, the *model* package, and the *global* package. The *biopart* package takes care of the interaction with the bioparts database. The *gene_network* package translates implements the translation of gene network templates to possible instances. This package has two sub-packages: The *gene_network.template* package, which provides a data structure to store gene network templates, and the *gene_network.logic* package, that can run simulations for a given logic device and provide an evaluation score. The *model* package provides a data structure to store a model and the functionality to translate a device into a model. Finally, the *global* package contains all global variables, such as relative paths to data files.

The *BiologicalDesigner* class provides a command line tool that can be used to run the following programs:

1. Run a simulation of a given device (the user should provide a string representation of the device).
2. Run a simulation for a random device instance of a given gene network template.
3. Run a simulation for all possible device instances of a given gene network template.
4. (Re)build the biopart database.

4.1 Biopart database

The partsregistry (`http://partsregistry.org`) contains a lot of genetic parts, but there are no kinetic constants available for the modeling. Therefore we have used a self made artificial database with realistic kinetic constants. Ellis et al. (2009) provides a method to build libraries of promoters with accompanying kinetic constants that can be used for modeling. A similar approach can also be used to build libraries of other genetic parts. This shows that it is feasible to realize a database similar to our artificial one.

The bioparts database contains the biological parts that can be used to build the *in silico* biological systems. It contains genetic parts (parts of DNA), proteins and environmental signals. The data that is stored for the parts are kinetic constants and relations to other biological parts. Only the data that is needed to build a model is stored. Other information, like the nucleotide sequence of the genetic parts, is not in the database.

[1] `http://acs.lbl.gov/~hoschek/colt/index.html`
[2] `http://www.jdom.org`

4.1.1 Naming convention

For simplicity we have used a naming convention biological parts. Reporter names start with *reporter* followed by an index number (*reporter0, reporter1, ...*).

TF names start with a character that indicates if it is an activator (*a*) or an inhibitor (*i*). The second character indicates if the TF can be activated by an environmental signal (*a*), inhibited by an environmental signal (*i*), or does not have an associated environmental signal (*0*). The third character indicates if the TF is a monomer (*m*), dimer (*d*), or tetramer (*t*). The fourth character is an index number. For example, the TF *iim0* is a monomer inhibitor that can be inhibited by an environmental signal. If there is another TF with the same properties it will be called *iim1*.

The names of the proteins that are part of TF tetramers and dimers starts with the name of the TF of which they are a sub-protein, followed by either *_subm*, when the sub-protein is a monomer, or *_subd*, when the sub-protein is a dimer. For example, a dimer that is part of the tetramer TF *i0t3* is called *i03t_subd*.

The names of environmental signals start with *es_* followed by the name of the TF that it either inhibits or activates. For example, an environmental signal that inhibits the TF *iim1* is called *es_iim1*.

Promoter names start with *pm_* followed by the names of all TFs that bind to it and an index number. For example, a promoter that binds the inhibitors *iim0* and *i0t1* is called *pm_iim0_i0t1_0*. Such a promoter can have different transcription speeds resulting in a library of promoters that bind the same TFs (*pm_iim0_i0t1_0, pm_iim0_i0t_1, pm_iim0_i0t_2, ...*), similar to the two promoter libraries, each containing 20 promoters, constructed by Ellis et al. (2009).

Ribosome binding site names start with *rbs* followed by an index number (*rbs0, rbs1, ...*).

Protein coding part names start with *pc_* followed by the protein for which they encode. For example, the protein coding part that encodes for the protein *i0d3_subm* has the name *pc_i0d3_subm*.

Finally there is only one terminator with the name *t*. There is only one version because the terminator does not contain any data for the model we use.

4.1.2 database implementation

The bioparts database has been implemented as a directory structure with XML files. Fig. 4.1 shows the directory structure in which each of the directories contain a number of XML files, one XML file per biopart. The *part* directory contains four different types of genetic parts: promoters, protein coding parts, ribosome binding sites, and terminators. The *pool* directory contains three types of proteins: reporters, transcription factors (subdivided into inhibitors and activators), and proteins that are part of a transcription factor dimer or tetramer, and environmental signals such as small molecules, light, temperature, or pressure.

Each biopart is stored in a separate XML file that stores kinetic rate constants ($\frac{1}{sec}$) and relations to other bioparts. As an example, listing 4.1 shows an XML file of the promoter *pm_i0m0_i0t3_0*. The element *k_transcription* determines the transcription rate for this promoter when no inhibiting or activating TFs are bound. Furthermore, multiple operators can be defined. Each operator can bind one TF defined by the *tf* element. The *type* attribute indicates if it is an activator or an inhibitor and the value is the unique name of the binding TF, which acts as a pointer to the TF in the database. The *k_bind_tf* and *k_unbind_tf* define the binding and unbinding reaction rates of the TF to the DNA.

4.1.3 *biopart* package

Fig.4.2 shows the class diagram of the biopart package. It shows what data is stored for which parts and what the relations between the different parts are. The rest of the variables and functions are not shown, these can be found in the source code as well as the *Javadoc* documentation.

Each part is stored as XML file (Fig. 4.1) and has an identifying name. There is a class for each of the parts which are all subclasses of the *AbstractDatabaseItem* class. The constructors of the biological parts take the identifying name as parameter which is used to locate the XML file

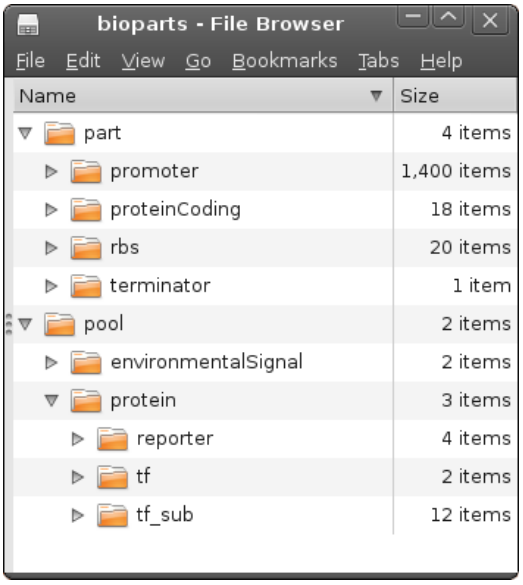


Figure 4.1: *The directory structure of the bioparts database. The data is stored in XML files within the directories.*

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <promoter name="pm_i0m0_i0t3_0">
4
5    <operators>
6      <operator>
7        <tf type="inhibitor">i0m0</tf>
8        <k_bind_tf>0.00383</k_bind_tf>
9        <k_unbind_tf>0.00145</k_unbind_tf>
10     </operator>
11     <operator>
12       <tf type="inhibitor">i0t3</tf>
13       <k_bind_tf>0.00382</k_bind_tf>
14       <k_unbind_tf>0.00165</k_unbind_tf>
15     </operator>
16   </operators>
17
18   <k_transcription>0.00635</k_transcription>
19
20 </promoter>
```

Listing 4.1: *An example of an XML file that stores the data of a the promoter pm_i0m0_i0t3_0*

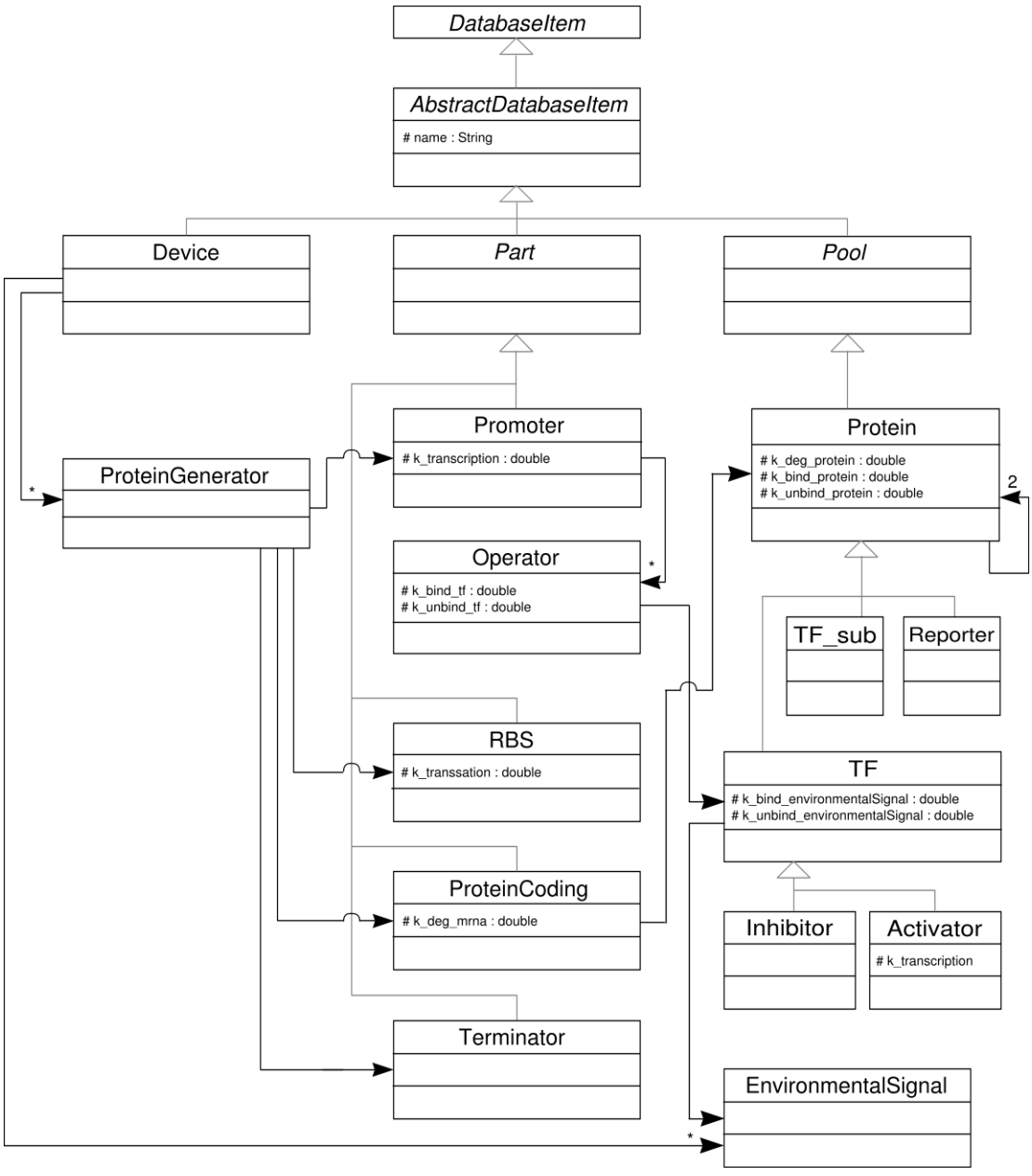


Figure 4.2: *Class diagram of the biopart package.*

that contains the data for this part. The data is then simply read from the file and stored within the class.

The package also contains a *BioPartDatabase* class that reads all the parts from the database into the memory which enables fast access to the database items. The class implements the singleton pattern so that the loading of the parts only has to be done once (which can takes a while because of the many file accesses). Since the database is not that large it is doable to completely load the database into memory.

Finally, the package contains a *BioPartDatabaseBuilder* class which provides a convenient way to build and rebuild an artificial biopart database. Parameter ranges were defined for the different kinetic constants, which enables the software to pick a random parameter from a realistic parameter range.

4.2 Gene network template

4.2.1 Graph representation

A gene network template is defined as a graph $G = (V, E)$ with V a set of vertices, representing protein generators and E a set of proteins produced by the protein generators. Each protein generator $v \in V$ must have exactly one output edge, which is the protein it produces, and can have zero, one, or more inputs, which are the proteins (TFs) that bind to the promoter of the protein generator. Each edge $e \in E$ can have zero, one or more source vertices, which are the protein generators that produce this protein. When an edge has no destination vertex, then it is an output of the system and the protein must be a reporter. When an edge has one or more destination vertices, then the edge is a TF that binds to the promoter of each destination protein generator.

An edge can be of the type -, +, or *0*, indicating that the protein used for this edge must be an inhibitor, an activator, or a reporter respectively. A signal can (but does not have to) be assigned to an edge. The value of the signal is either - or +, indicating that the edge can be inhibited or activated by an environmental signal respectively.

4.2.2 Data format

A gene network template can be defined as XML file. It contains a list of vertex with a unique identifier and a list of edges with a unique identifier. The type of the edge can be defined as an attribute of the edge element. For each edge, a number of source vertices and destination vertices can be assigned to specify the connections. Finally, some of the edge's have to be defined as input or output edge's.

As an example, Listing 4.2 shows the definition of the NOT gate from Fig. 4.3 and Listing 4.3 shows the definition of the external NOR gate from Fig. 4.4.

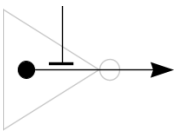


Figure 4.3: *Gene network template of an external NOT gate as defined in the XML of listing 4.2.*

A defined network can be used as a sub-network of another gene network template. In that case one needs to define a list of sub-networks with unique identifiers and a list of connection edge's, with unique identifiers as well, that interconnect the sub-networks. For example, Listing 4.4 shows the definition of the demultiplexer network in Fig. 4.5. There are five logic gate networks that are used to build this network. The connection edge's define what sub-network outputs are connected to what sub-network inputs. A number of input and output edges need to be specified as well.

```
<?xml version="1.0" encoding="UTF-8"?>
<network name="NOTe">

  <vertice id="v0"/>

  <edge id="e0" signal="-">
    <src vertice_id="v0" />
  </edge>

  <output id="out0" edge_id="e0" />

</network>
```

Listing 4.2: *An example of an XML file that stores a gene network template of an external logic NOT gate. Fig. 4.3 shows a visualization of the network.*

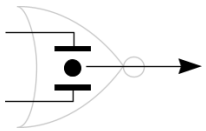


Figure 4.4: *Gene network template of a NOR gate as defined in the XML of listing 4.3.*

```
<?xml version="1.0" encoding="UTF-8"?>
<network name="NORii">

  <vertice id="v0" />

  <edge id="e0" type="-">
    <dest vertice_id="v0" />
  </edge>
  <edge id="e1" type="-">
    <dest vertice_id="v0" />
  </edge>
  <edge id="e2">
    <src vertice_id="v0" />
  </edge>

  <input id="in0" edge_id="e0" />
  <input id="in1" edge_id="e1" />

  <output id="out0" edge_id="e2" />

</network>
```

Listing 4.3: *An example of an XML file that stores a gene network template of a logic NOR gate*

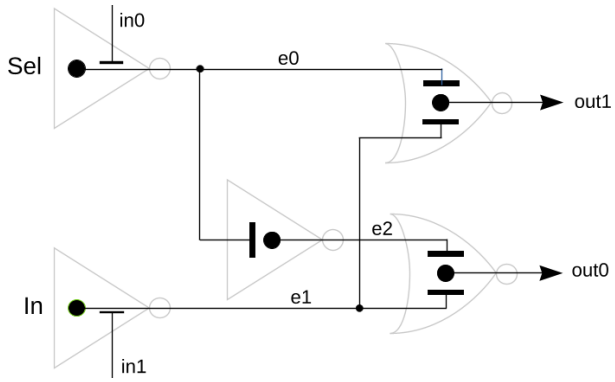


Figure 4.5: *Gene network template of a 1-to-2 line demultiplexer as defined in the XML file shown in listing 4.4.*

```
<?xml version="1.0" encoding="UTF-8"?>
<network name="Demultiplexer">

  <subnetwork id="NOTe0" name="NOTe" />
  <subnetwork id="NOTe1" name="NOTe" />
  <subnetwork id="NOTi" name="NOTi" />
  <subnetwork id="NORii0" name="NORii" />
  <subnetwork id="NORii1" name="NORii" />

  <connection_edge id="e0">
    <src network="NOTe0" output_id="out0" />
    <dest network="NORii0" input_id="in0" />
    <dest network="NOTi" input_id="in0" />
  </connection_edge>
  <connection_edge id="e1">
    <src network="NOTe1" output_id="out0" />
    <dest network="NORii0" input_id="in1" />
    <dest network="NORii1" input_id="in1" />
  </connection_edge>
  <connection_edge id="e2">
    <src network="NOTi" output_id="out0" />
    <dest network="NORii1" input_id="in0" />
  </connection_edge>

  <input id="in0" edge_id="e0" />
  <input id="in1" edge_id="e1" />

  <output id="out0" network="NORii1" output_id="out0" />
  <output id="out1" network="NORii0" output_id="out0" />

</network>
```

Listing 4.4: *An example of an XML file that stores a gene network template of a 1-to-2 line demultiplexer to show how sub-networks of logic gates can be used to build a logic network. Fig. 4.5 displays the network.*

4.2.3 *gene_network_template* package

Fig. 4.6 shows the class diagram of the *gene_network_template* package. The package can read a gene network template from file and provides a data structure to store the network. There is a *GeneNetworkTemplate* class that contains a list of vertices and a list of edges. Each *Edge* stores a number of source and destination vertices and each vertex stores a number of input edges and output edges, which of course always correspond to each other. Furthermore, a number of functions are available that are used to turn a gene network template into gene network instances.

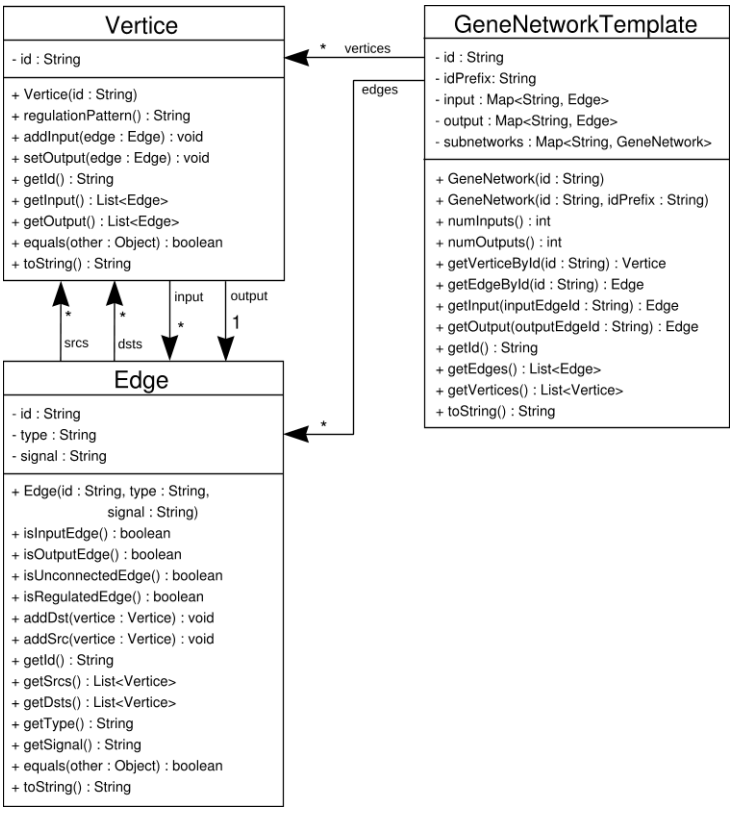


Figure 4.6: *Class diagram of the gene_network.template package.*

4.3 Gene network builder

A gene network template can be turned into many possible gene network instances using using the gene network builder.

4.3.1 *gene_network* package

The *gene_network* package contains only the *GeneNetworkBuilder* class which provides a number of methods to turn a gene network template into one or more gene network instances. The *getRandomGeneNetwork* returns a random gene network instance. The *getAllGeneNetworks* returns all possible gene network instances for the given template. Since the number of possible instances grows very large for complex networks and a large bioparts database, this method will soon result in an out of memory error. Finally, the *getGeneNetworkVariations* takes a gene network instance as input an returns all instances with varied promoters (from the same promoter library) and

varied RBSs. Variation of those parts means variation of the transcription and translation speeds at all location within the network.

4.4 Model builder

The model builder transforms a gene network device into a model.

4.4.1 *model* package

The *Model* class stores a list of *Reaction* objects and a list of *Species* objects. The *Species* class represent a molecular species which has a name and an initial amount, which is the molecule count of this species at the start of a simulation. The *Reaction* class represents a reaction from one or more reactants to one or more products. Since we are using stochastic simulation only two element types of reactions can be specified: unimolecular reactions, with only one reactant, and bimolecular, with two reactants (Section 3.2.3). A reaction rate is also defined that determines the at which rate this reaction occurs.

The *Model* class is an abstract class. This enables implementation of multiple sub-classes that define different models. For example, an extra model can be defined in which the gene transcription process is modeled different. Our sub-class *Model0* translates each of the device's protein generators into a standard model (Section 3.3), which together automatically form a model for the whole gene network (Section 2.7.2). The parameters needed for the model are fetched from the genetic parts (which are in the bioparts database) that are part of a protein generator. Finally some extra reactions are added for the binding of the environmental signals that are the input signals to the gene network.

The *ModelWriter* interface defines a method to write the model to file. Our implemented sub-class, the *FernMLWriter* class, implements this method so that the model can be written to a *fernml* file, which can be used for stochastic simulation by the *FERN* software framework.

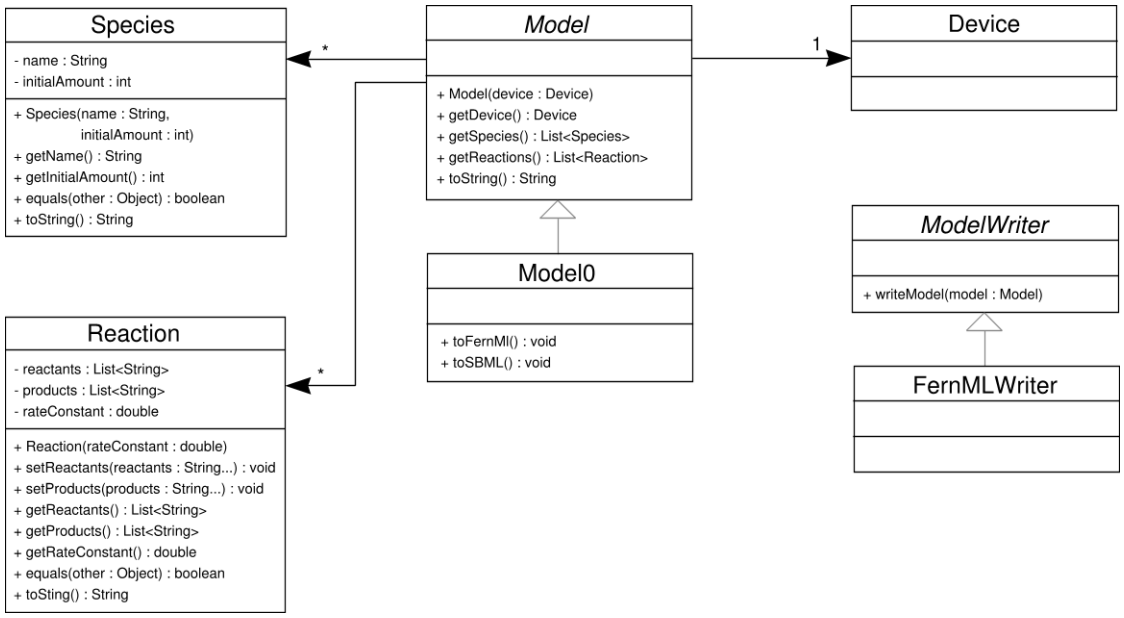


Figure 4.7: *Class diagram of the model package.*

```
<?xml version="1.0" encoding="UTF-8"?>
<logic_device_settings name="Demultiplexer0">

  <gene_network>Demultiplexer</gene_network>

  <input id="in0">iim0</input>
  <input id="in1">iim1</input>

  <output id="out0">reporter0</output>
  <output id="out1">reporter1</output>

  <binary_timing_diagram>
    <plot id="in0" >00110</plot>
    <plot id="in1" >01100</plot>
    <plot id="out0">01000</plot>
    <plot id="out1">00100</plot>
  </binary_timing_diagram>

  <state_time>7200</state_time>
  <visual>true</visual>

</logic_device_settings>
```

Listing 4.5: *User input.*

4.5 Logic gene network simulation

The gene network model can be used to run a stochastic simulation. User input is needed for the simulation settings. After simulation, a score is calculated for the simulated gene network device.

4.5.1 User input

The input to the software tool, as shown on the right of Fig. 2.11, can be defined in an XML file as the one in listing 4.5. The *gene_network* element is a pointer to the XML file that stores the gene network template that should be used. In this case, it is a pointer to the file that is displayed in listing 4.4.

The gene network template file defines two inputs and two outputs. A protein has to be assigned to each of the signals. In case of an output signal this has to be a reporter (*reporter0* and *reporter1* for this example) and in case of an input this has to be a TF. The TF should be one that binds an environmental signal, so that it can accept an external signal. When a - signal is assigned to the input edge, the TF should bind an inhibiting environmental signal, and when a + signal is assigned to the input edge, the TF should bind an activating environmental signal. In this case, both inputs are output edges from a *NOTe* network, which have an assigned - signal. The TFs assigned as inputs, *iim0* and *iim1*, therefore both bind an inhibiting environmental signal. Fig. 4.8 shows the gene network template with assigned inputs and outputs.

Fig. 4.9 shows the other input that has to be defined; a binary timing diagram for each input and output signal. The input signals will determine the concentrations of the environmental signals during simulation and the output specifies what the desired concentrations of the reporters should be. The desired output will be used after a simulation in order to evaluate the performance of the gene network. The binary timing diagrams can be specified in the XML file as a binary string. The binary strings in listing 4.5 correspond to the plots in Fig. 4.9. The *state_time* element determines the time of each of the states in seconds, which can be seen on the x-axis of the binary timing diagrams. The number of states should of course be the same for all in- and outputs. The first state is used for initialization and is not taken into account for the evaluation of the system.

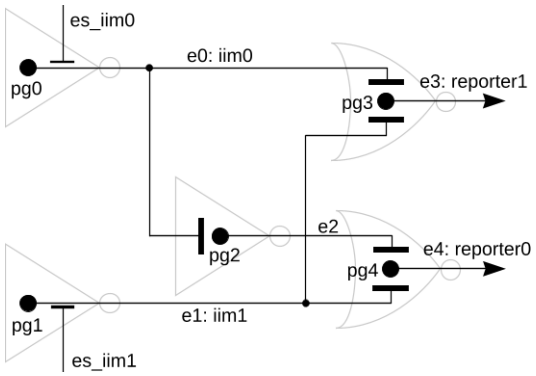


Figure 4.8: *The demultiplexer gene network template with assigned inputs and outputs.*

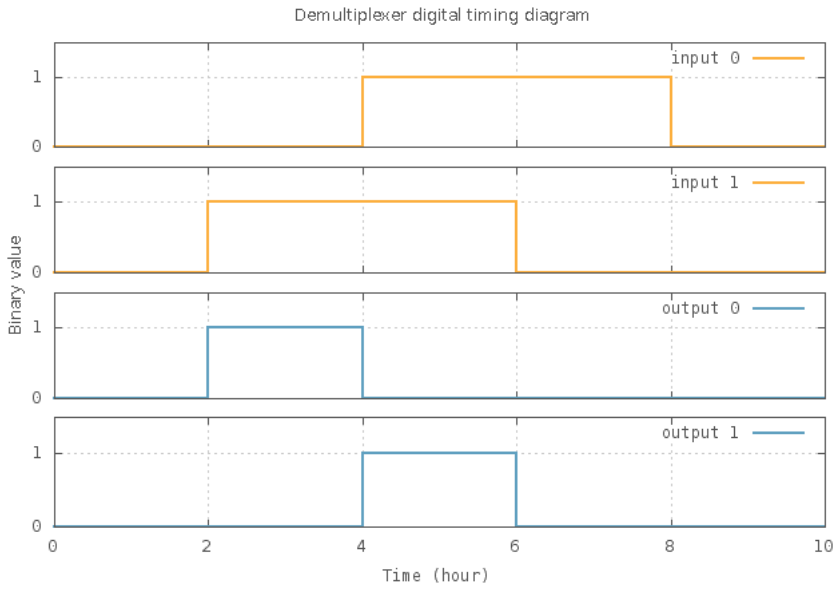


Figure 4.9: *The user defined digital timing diagram of the inputs and the desired outputs for the demultiplexer gene network.*

4.5.2 Score calculation

The first time step is used for the initialization of the system and is not taken into account with the score calculation. For each of the n outputs the correlation $c_i, i = 0, \ldots, n$, between the desired output and the simulated output is calculated.

A low value l_0 for output 0 in Fig. 4.10 is the average of the orange circled values. The orange circles are the minimum values in the time steps that have a low value in the desired output. This low value l_i is calculated for all n outputs.

A high value h_0 for output 0 in Fig. 4.10 is the average of the green circled values. The green circled value is the maximum in the time step where the desired output high. In this case, the signal has only one high output, which is therefore also the average. This high value h_i is calculated for all n outputs.

The correlations c_i, the low values l_i, and the high values h_i for all n output signals are combined into one score as follows:

$$score = (\prod_{i=0}^{n} c_i) \times ((\prod_{i=0}^{n} h_i) - (\sum_{i=0}^{n} l_i)) \tag{4.1}$$

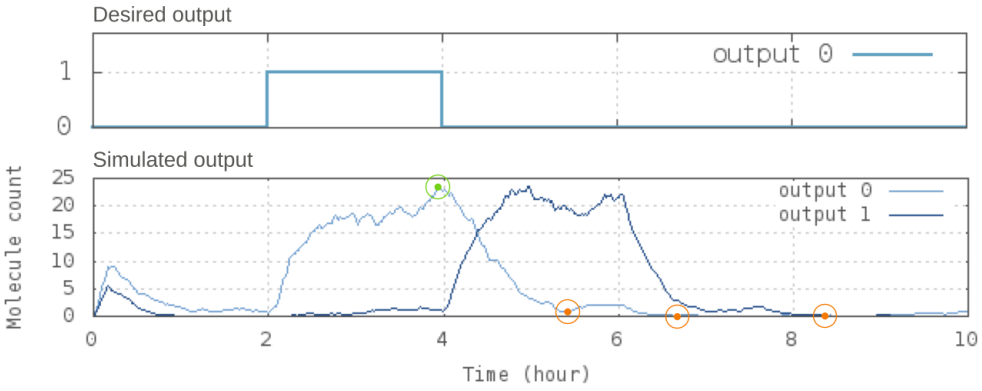


Figure 4.10:

4.5.3 *gene_network.logic* package

The *gene_network.logic* package provides a *LogicGeneNetworkSettings* class that reads the user input from file (listing 4.5). The *BinaryTimingDiagramObserver* is an extension of the *Observer* interface provided by the FERN software framework. This class changes the concentrations of the environmental signals during a simulation according to the timing diagrams provided by the user. In case of a binary '1', the molecule count of the environmental signal is set to a high value, and in case of a binary '0', the molecule count of the environmental signal is set to zero. Finally the *LogicGeneNetworkSimulation* class can be used to run a simulation for a logic gene network. When *gnuplot* is available, the plot of the inputs and outputs can be visualized during simulation. As a result, the score of a device will be returned after running a simulation.

Chapter 5

Experiments

We performed two experiments in order to test the software. Before describing the experiments, the parameters that were used are discussed first. Finally, the obtained results are discussed.

5.1 Kinetic constants

Our model uses ten different parameters: k_{tc} (transcription rate), k_{tl} (translation rate), k_{deg_r} (degradation rate mRNA), k_{deg_p} (degradation rate protein), k_{btf} (TF-DNA binding rate), k_{utf} (TF-DNA unbinding rate), (protein-protein association rate), k_{up} (protein-protein dissociation rate), k_{bs} (protein-signal association rate), k_{us} (protein-signal dissociation rate). The model is based on the one used by Hooshangi et al. (2005). Table 5.1 provides the parameters that they used with the simulations.

kinetic constant	Hooshangi et al. (2005)
k_{tc}	0.0333 sec^{-1}
k_{tc} (repressed)	0.000333 sec^{-1}
k_{tl}	0.0333 sec^{-1}
k_{deg_r}	0.0114 sec^{-1}
k_{deg_p}	0.0012 sec^{-1}
k_{btf}	0.00333 $nM^{-1}.sec^{-1}$
k_{utf}	0.00167 sec^{-1}
k_{bp}	0.0005 $nM^{-1}.sec^{-1}$
k_{up}	0.0000167 sec^{-1}
k_{bs}	0.000833 $nM^{-1}.sec^{-1}$
k_{us}	0.00167 sec^{-1}

Table 5.1: *Kinetic constants used in Hooshangi et al. (2005), with the assumption that* $1nM$ *corresponds to 1 molecule per cell.*

With our method we build genetic networks using the same model, but with varying parameters. Instead of the fixed parameters we need parameter ranges. For both the translation as well as the transcription, which are both modeled as a single step reaction, we use rather slow reaction rate ranges (1 reaction per 160 - 100 seconds, compared to the 1 reaction per 30 seconds from Hooshangi et al. (2005)). This is done in order to get low molecule counts and therefore less occuring reactions, which means faster simulation. This indicates that the stochastic simulation in combination with our pretty extensive models do not provide a practical application. To be practical, either the model should be simplified or another simulation method should be used. A simulation environment as proposed by Kosuri et al. (2007) could resolve this problem.

For the binding and unbinding of environmental signals to TFs we use fixed parameters. The parameters of the mRNA and protein degradation rates are based on Bernstein et al. (2002) and Grilly et al. (2007) respectively. Although the protein degradation rate from Hooshangi et al. (2005) lies within the given range, the mRNA rate does not. The protein binding rate range is based on Schlosshauer and Baker (2004). No publications on the binding and unbinding rates of TFs to DNA were found and therefore a range around the parameters of Hooshangi et al. (2005) was used.

kinetic constant	min	max	
k_{tc}	0.00625	0.01	one transcript per gene each 160 . . . 100 sec
k_{tl}	0.00625	0.01	one protein per mRNA each 160 . . . 100 sec
k_{deg_r}	0.00144	0.00385	half-life 8 . . . 3 min Bernstein et al. (2002)
k_{deg_p}	0.000183	0.00167	half-life 91 . . . 10 min Grilly et al. (2007)
k_{btf}	0.00278	0.00417	Hooshangi et al. (2005) +/− 1 min
k_{utf}	0.00139	0.00208	Hooshangi et al. (2005) +/− 2 min
k_{bp}	0.0001	0.001	Schlosshauer and Baker (2004)
k_{up}	0.0000167		fixed, same as Hooshangi et al. (2005)
k_{bs}	0.000833		fixed, same as Hooshangi et al. (2005)
k_{us}	0.00167		fixed, same as Hooshangi et al. (2005)

Table 5.2: *Kinetic constants used in Hooshangi et al. (2005), with the assumption that* $1nM$ *corresponds to 1 molecule per cell.*

5.2 Experiment 1: 1-to-2 line Demultiplexer

Fig. 5.1 shows the digital circuit of a 1-to-2 line demultiplexer[1] as it is used in the field of electrical engineering. The multiplexer forwards the input signal to one of the outputs, depending on the select signal. As can be seen in the truth table in Fig. 5.1, the input is forwarded to output 0 when the select signal is 0, and to output 1 when the select signal is 1.

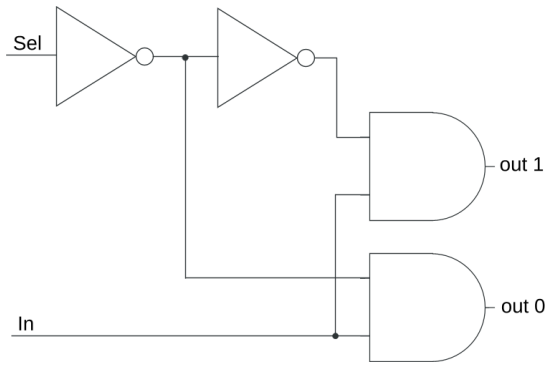
The logic circuit has been transformed into an equivalent version that has NOT gates for both inputs (Fig. 5.2). This way, we can use NOT gates that accept an environmental signal as input. The used settings are given in Lis. 5.1 and a visual representation of the settings is shown in Fig. 5.3. The used artificial biopart database contained 2 promoters per library, a slow and a fast one, and 2 ribosome binding sites, also a slow and a fast one. The database contained 5 monomer, 5 dimer, and 5 tetramer TFs that can be used for edge *e2*. With this biopart database, 15360 (15×2^{10}) possible gene network instance can be build for the demultiplexer gene network template.

Each of the possible gene networks was simulated using the *simAllPossibleGeneNetworks* program. The result was a tab separated file with the correlation, the low value (Sec. 4.5.2), the high value, and the score, for each of the 15360 gene networks.

5.3 Experiment 2: DLatch

Fig. 5.4 shows the digital circuit of a DLatch[2] as it is used in the field of electrical engineering. The DLatch forwards the data signal (D) to the output (Q) when the enable signal (E) is high. When the Enable signal is low, the data signal remains in its current state. As can be seen in the truth table in Fig. 5.4, the data signal is forwarded to Q when the enable signal is 1. When the enable signal is 0, it does not matter what the value of the data signal is, the output will stay in the current state (either 0 or 1).

[1] `http://www.play-hookey.com/digital/decoder_demux_two.html`
[2] `http://en.wikipedia.org/wiki/Latch_(electronics)#Gated_D-latch`



Select	Input	Output 0	Output 1
0	0	0	0
0	1	1	0
1	0	0	0
1	1	0	1

Figure 5.1: ***1-to-2 line demultiplexer*** *- The digital circuit is shown en the left and the truth table is shown on the right.*

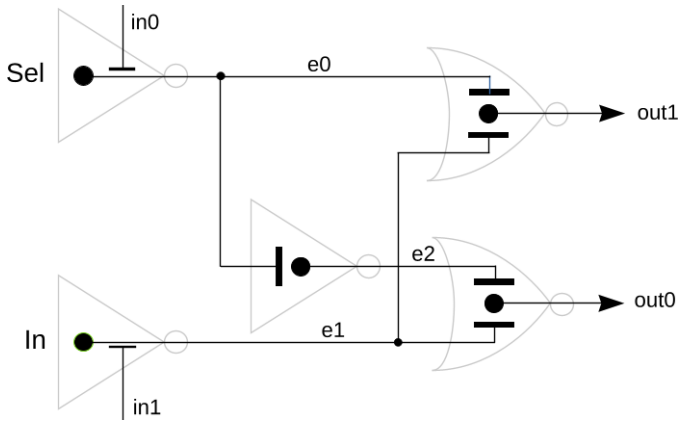


Figure 5.2: ***1-to-2-line demultiplexer gene network template*** *- The logic circuit is equivalent to the one in Fig. 5.1.*

```
<?xml version="1.0" encoding="UTF-8"?>
<logic_device_settings name="Demultiplexer0">

  <gene_network>Demultiplexer</gene_network>

  <input id="in0">iim0</input>
  <input id="in1">iim1</input>

  <output id="out0">reporter0</output>
  <output id="out1">reporter1</output>

  <binary_timing_diagram>
    <plot id="in0" >00110</plot>
    <plot id="in1" >01100</plot>
    <plot id="out0">01000</plot>
    <plot id="out1">00100</plot>
  </binary_timing_diagram>

  <state_time>7200</state_time>
  <visual>false</visual>

</logic_device_settings>
```

Listing 5.1: *Settings experiment 1.*

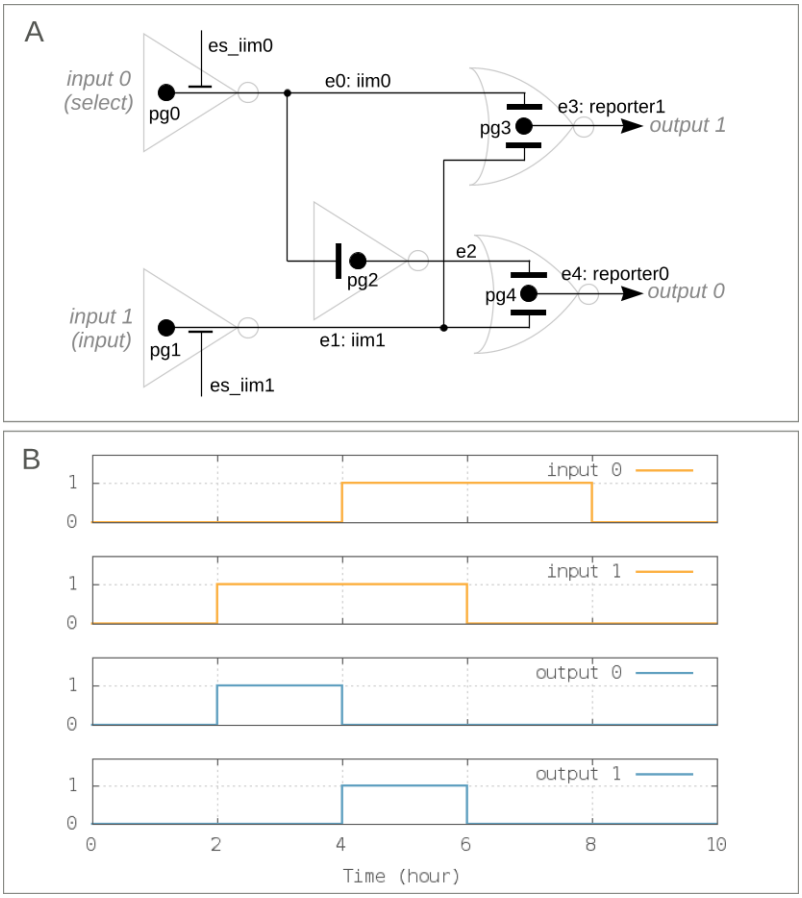
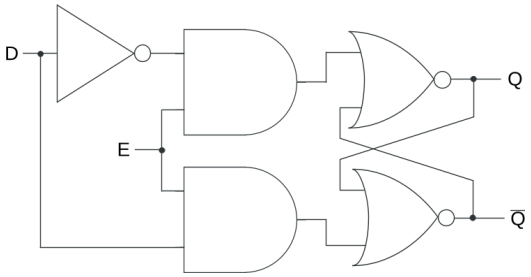


Figure 5.3: *The settings of experiment 1 as specified in Lis. 5.1.*

The logic circuit has been transformed into an equivalent version that has NOT gates for both inputs (Fig. 5.5). This way, we can use NOT gates that accept an environmental signal as input. The used settings are given in Lis. 5.2 and a visual representation of the settings is shown in Fig. 5.6. The used artificial biopart database contained 1 promoter per library and 1 ribosome binding site. The database contained 6 monomer, 6 dimer, and 5 tetramer TFs that can be used for the edges *e2*, *e3*, *e4*, *e5*, and *e6*. With this biopart database, 702 possible gene network instance can be build for the DLatch gene network template.

Each of the possible gene networks was simulated using the *simAllPossibleGeneNetworks* program. The result was a tab separated file with the correlation, the low value (Sec. 4.5.2), the high value, and the score, for each of the 702 gene networks.



Enable	Data	Q	$\overline{Q}$	
0	X	Q_{prev}	Q_{prev}	(no change)
1	0	0	1	(reset)
1	1	1	0	(set)

Figure 5.4: ***DLatch*** *- The digital circuit is shown en the left and the truth table is shown on the right.*

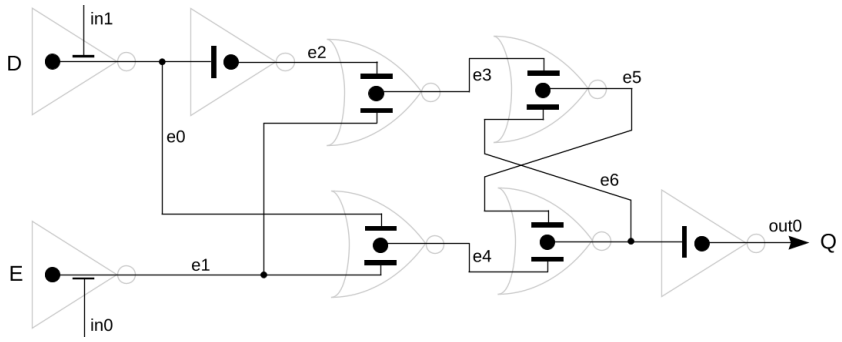


Figure 5.5: ***DLatch gene network template*** *- The logic circuit is equivalent to the one in Fig. 5.4.*

5.3.1 Results

The experiments provided for each simulated device the following measures: a correlation for each output, a low value for each output, a high value for each output, and a score (the low and high values are explained in Section 4.5.2). So for each of this measures we had a list (column in data file) of values, one per device. Other columns contained information about the device, so the device was still coupled to the result.

We then ordered the list of values from low to high. So in case of the score, the devices with a low score are at the top and the devices with a high score at the bottom of the list. We have split the list into 20, in case of experiment 1, and 18, in case of experiment 2, equal sized bins. In case of the scores, the first bin contained the devices with the lowest scores, and the last bin contained the devices with the highest scores.

In case of experiment 1, we counted the number of devices with a slow promoter and with a fast promoter for a given protein generator within a bin. We did this for each protein generator

```
<?xml version="1.0" encoding="UTF-8"?>
<logic_device_settings name="DLatch0">

  <gene_network>DLatch</gene_network>

  <input id="in0">iim0</input>
  <input id="in1">iim1</input>

  <output id="out0">reporter0</output>

  <binary_timing_diagram>
    <plot id="in0" >010011100</plot>
    <plot id="in1" >111001001</plot>
    <plot id="out0">111101000</plot>
  </binary_timing_diagram>

  <state_time>43200</state_time>
  <visual>false</visual>

</logic_device_settings>
```

Listing 5.2: *Settings experiment 2.*

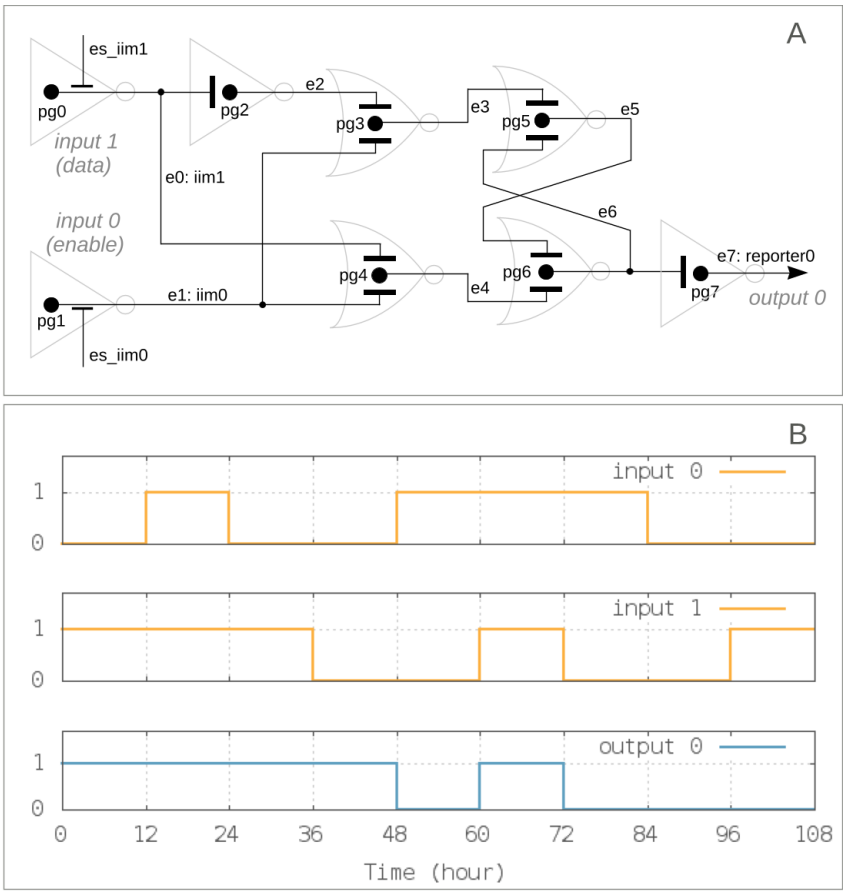


Figure 5.6: *The settings of experiment 2 as specified in Lis. 5.2.*

for all bins. The same count was also done for slow and fast RBSs. For edge *e2* we counted the number devices that used a monomer, dimer, or tetramer TF in each bin.

Fig. 5.7 shows the resulting histograms for the score measure. The plot at the top are the scores of all devices order from low to high. The y-grid lines are the boundaries of the bins. The histograms are aligned with plot and show the device counts within each bin. It can be seen that the devices with a high score mostly have a slow promoter and RBS for protein generator 1 and a fast promoter and RBS for protein generator 4. The last histogram shows that most of the devices with a low score use a monomer TF for edge *e2*. The histograms of the other measures can be found in the supplementary material of the paper.

In case of experiment 2, there was no variation of the promoter and RBS speeds. The only thing that was the use of either a monomer, a dimer, or a tetramer, for the edges textsle2, *e3*, *e4*, *e5*, and *e6*. For a given edge, we counted the number of devices that used a monomer, dimer, and tetramer within a bin. We did this for each edge for all bins.

Fig. 5.8 shows the resulting histograms for the score measure. The plot at the top are the scores of all devices order from low to high. The y-grid lines are the boundaries of the bins. The histograms are aligned with plot and show the device counts within each bin. It can be seen that the devices with a high score mostly use a tetramer TF for edge *e6* and a monomer or dimer TF for edge *e3*. The histograms of the other measures can be found in the supplementary material of the paper.

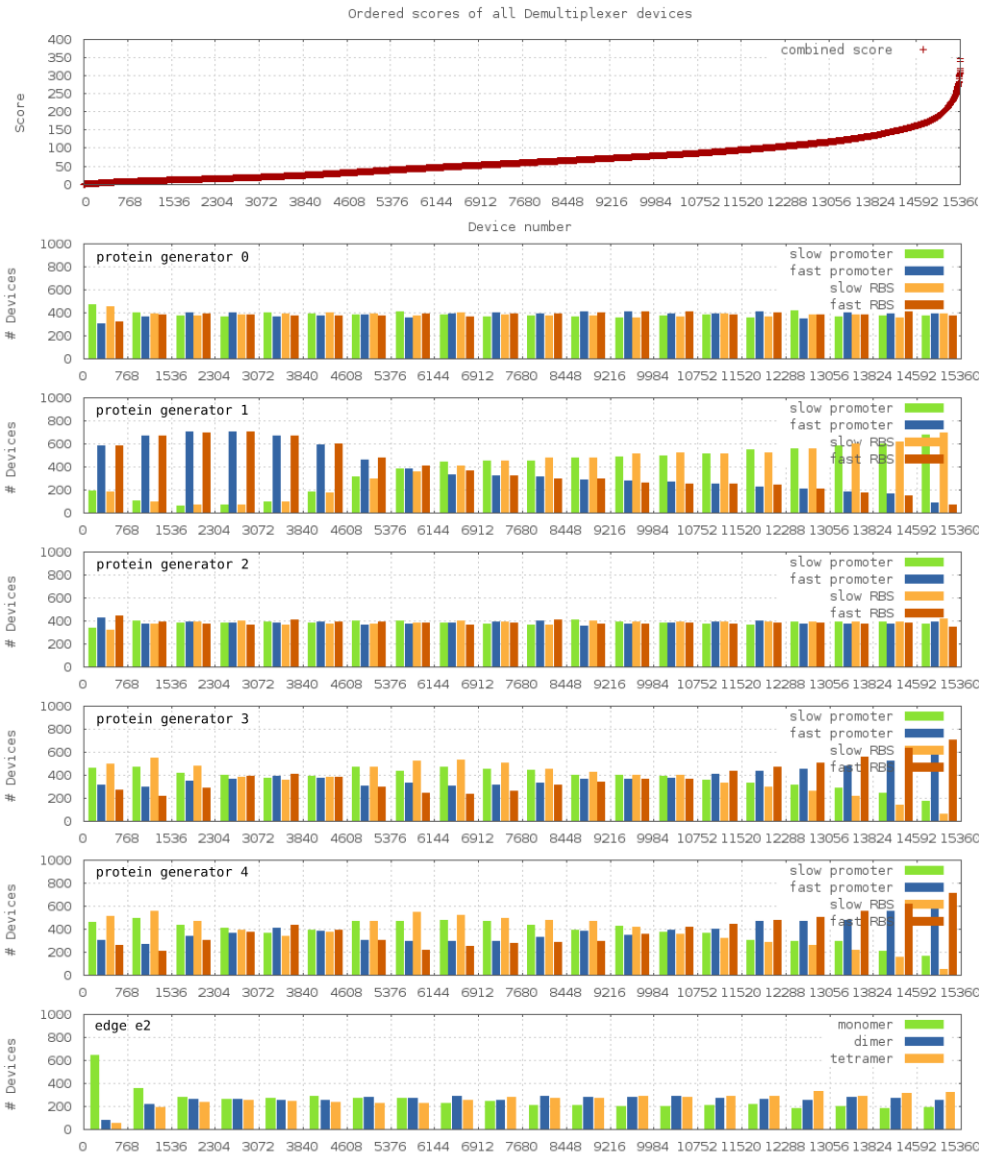


Figure 5.7: *The plot at the top are the score value of all 15360 devices order from low to high. The devices are subdivided into 20 equal sized bins. The y-axis lines are the boundaries of the bins. The histograms below show the device counts for the different bins, in which the devices are counted that use a slow promoter, fast promoter, slow RBS, and fast RBS. The last histogram shows the device count of the devices that use a monomer, dimer, and tetramer for edge e2.*

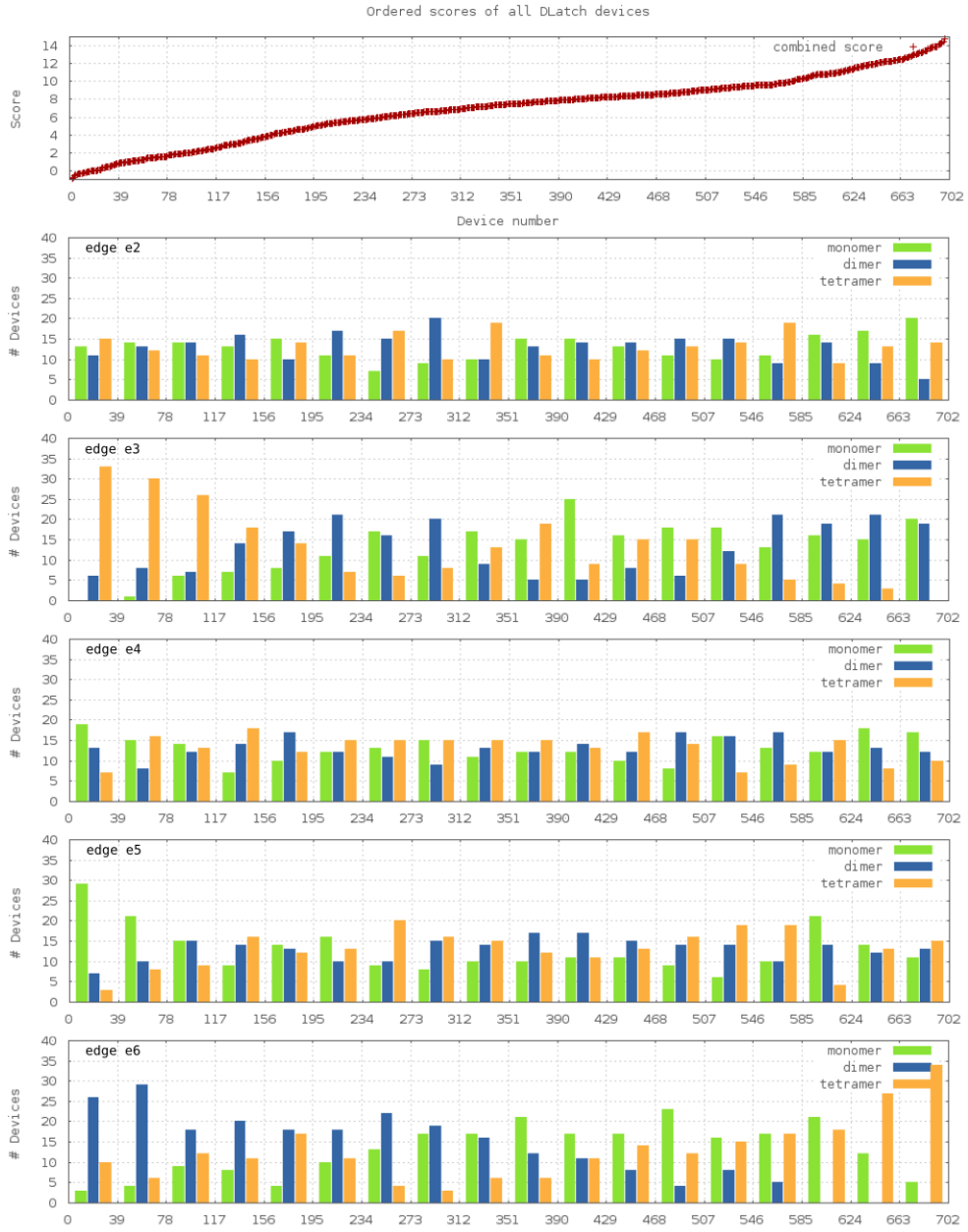
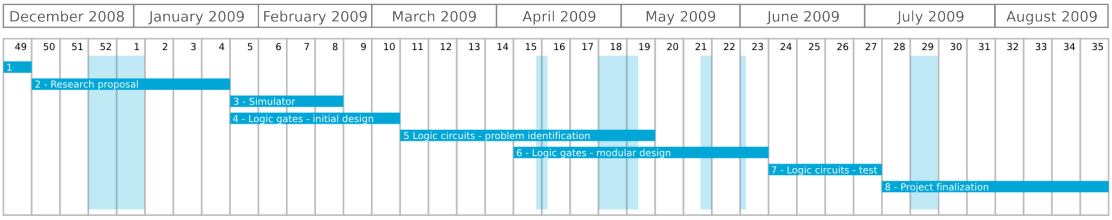


Figure 5.8: *The plot at the top are the score value of all 702 devices order from low to high. The devices are subdivided into 18 equal sized bins. The y-axis lines are the boundaries of the bins. The histograms below show the device counts for the different bins of devices that use a monomer, dimer, or tetramer for the given edge.*

Appendix A

Planning



Task	Start	Finish	Duration
1. Subject selection	Dec 1	Dec 5	5 days
2. Research proposal	Dec 8	Jan 23	25 days
3. Simulator - testing and configuration	Jan 26	Feb 20	20 days
4. Logic gates - initial design	Jan 26	Mar 6	30 days
5. Logic circuits - problem identification	Mar 9	May 8	36 days
6. Logic gates - modular design	Apr 6	Jun 5	33 days
7. Logic circuits - final tests	Jun 8	Jul 3	20 days
8. Project finalization	Jul 6	Aug 28	35 days

Bibliography

Anderson, J. C., Voigt, C. A., Arkin, A. P., 2007. Environmental signal integration by a modular and gate. Mol. Syst. Biol. 3, 133.

Bernstein, J. A., Khodursky, A. B., Lin, P. H., Lin-Chao, S., Cohen, S. N., 2002. Global analysis of mrna decay and abundance in escherichia coli at single-gene resolution using two-color fluorescent dna microarrays. Proceedings of the National Academy of Sciences 99 (15), 9697.

Cai, L., Friedman, N., Xie, X. S., 2006. Stochastic protein expression in individual cells at the single molecule level. Nature 440 (7082), 358–62.

Canton, B., Labno, A., Endy, D., 2008. Refinement and standardization of synthetic biological parts and devices. Nat. Biotechnol. 26 (7), 787–93.

Cox, R. S., Surette, M. G., Elowitz, M. B., 2007. Programming gene expression with combinatorial promoters. Mol. Syst. Biol. 3, 145.

Del Vecchio, D., Ninfa, A. J., Sontag, E. D., 2008. Modular cell biology: retroactivity and insulation. Mol. Syst. Biol. 4, 161.

Ellis, T., Wang, X., Collins, J. J., 2009. Diversity-based, model-guided construction of synthetic gene networks with predicted functions. Nature Biotechnology.

Elowitz, M., Leibler, S., 2000. A synthetic oscillatory network of transcriptional regulators. Nature 403 (6767), 335–338.

Elowitz, M., Levine, A., Siggia, E., Swain, P., 2002. Stochastic gene expression in a single cell. Science 297 (5584), 1183–1186.

Endy, D., 2005. Foundations for engineering biology. Nature 438 (7067), 449–53.

Erhard, F., Friedel, C. C., Zimmer, R., 2008. Fern - a java framework for stochastic simulation and evaluation of reaction networks. BMC Bioinformatics 9, 356.

Gardner, T., Cantor, C., Collins, J., 2000. Construction of a genetic toggle switch in escherichia coli. Nature 403 (6767), 339–342.

Gillespie, D., 2007. Stochastic simulation of chemical kinetics. Annual Review Of Physical Chemistry 58, 35.

Gillespie, D., et al., 1977. Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry 81 (25), 2340–2361.

Grilly, C., Stricker, J., Pang, W. L., Bennett, M. R., Hasty, J., 2007. A synthetic gene network for tuning protein degradation in saccharomyces cerevisiae. Mol. Syst. Biol. 3, 127.

Guet, C., Elowitz, M., Hsing, W., Leibler, S., 2002. Combinatorial synthesis of genetic networks. Science 296 (5572), 1466–1470.

Hooshangi, S., Thiberge, S., Weiss, R., 2005. Ultrasensitivity and noise propagation in a synthetic transcriptional cascade. Proceedings of the National Academy of Sciences 102 (10), 3581–3586.

Kaern, M., Elston, T. C., Blake, W. J., Collins, J. J., 2005. Stochasticity in gene expression: from theories to phenotypes. Nat. Rev. Genet. 6 (6), 451–64.

Kosuri, S., Kelly, J. R., Endy, D., 2007. Tabasco: A single molecule, base-pair resolved gene expression simulator. BMC Bioinformatics 8, 480.

Lucks, J. B., Qi, L., Whitaker, W. R., Arkin, A. P., 2008. Toward scalable parts families for predictable design of biological circuits. Curr. Opin. Microbiol. 11 (6), 567–73.

Mayo, A. E., Setty, Y., Shavit, S., Zaslaver, A., Alon, U., 2006. Plasticity of the cis-regulatory input function of a gene. PLoS Biol. 4 (4), e45.

Rodrigo, G., Carrera, J., Jaramillo, A., 2007. Genetdes: automatic design of transcriptional networks. Bioinformatics 23 (14), 1857–8.

Sauro, H. M., 2008. Modularity defined. Mol. Syst. Biol. 4, 166.

Schlosshauer, M., Baker, D., 2004. Realistic protein–protein association rates from a simple diffusional model neglecting long-range interactions, free energy barriers, and landscape ruggedness. Protein Science: A Publication of the Protein Society 13 (6), 1660.