OSAKA UNIVERSITY

PHDTHESIS

Study of Promoter Activation via Single Cell Analysis

一細胞解析によるプロモータ活性化の解析に関する研究

JIANHONG OU

Division of Advanced Science and Biotechnology
Graduate School of Engineering

Thesis Advisor: Dr. Hiroshi Shimizu

July 14, 2009

OSAKA UNIVERSITY

PHDTHESIS

Study of Promoter Activation via Single Cell Analysis

一細胞解析によるプロモータ活性化の解析に関する研究

JIANHONG OU

Division of Advanced Science and Biotechnology
Graduate School of Engineering

Thesis Advisor: Dr. Hiroshi Shimizu

July 14, 2009

Contents

Abstract

Systems biology is an approach to biology that seeks to understand and predict the quantitative features of a multicomponent biological system. It requires quantitative description of endogenous regulation networks to construct appropriate models which can make predictions about the behavior of the interacting networks. In those interacting networks, promoter activation plays a key role in driving gene transcription, which, in turn, cause fluctuation in gene expression. Such fluctuations could determine the cell fate or do harm to regulation networks. Thus not only the study of promoter activity but also the activity fluctuations become important for a quantitative understanding. However there still lack powerful technique for accurate and comparable analysis for promoter activation in complex endogenous networks. The clear background of lysine biosynthesis pathway makes the promoter activation analysis a good sample involving biological noise in systems biology. Time-dependent experiments was employed in this study because it can result in wealth of information, which makes the model construction feasible. The ability to analyze promoter activation at single-cell level will enable much more accurate studies of cell population homogeneity in their regulation. Using single-cellular technique, this study aimed at setting up a method which can be used in complex networks to conduct a transition from a qualitative to a quantitative understanding.

By single fluorescence experiments, we investigated the expression dynamics of genes involved in lysine biosynthesis in *Escherichia coli* cells to obtain a quantitative

understanding of the gene regulatory system. By constructing reporter strains expressing the green fluorescence protein gene(*gfp*) under the control of the promoter regions of those genes associated with lysine biosynthesis, time-dependent changes in gene expression in response to changes in L-lysine concentration in the medium were monitored by flow cytometry. Time-dependent gene expression data were fitted to a simple dynamical model of gene expression to estimate the parameters of the gene regulatory system. The results provide a better quantitative understanding of the promoter dynamics in the lysine biosynthesis pathway.

After that a dual-fluorescence system for promoter strength analysis was developed to involve the biological noise information. This system includes two parts, the vector pGRFP and simulation tool. By fitting the expression and intrinsic noise getting from pGRFP vector, simulation tool can easily get appropriate transition rate of the two state, λ_{on} and λ_{off}, for the target promoter based on a stochastic formulation of chemical kinetics derived by *Gillespie*. We applied this system to analyze the kinetics of promoters involved in lysine biosynthesis. By well fitting not only the expression level but also the intrinsic noise, we got the λ_{on} and λ_{off} of the promoters. We found the slow transitions between promoter states of *lysA*p, which indicates the transcriptional bursting also can be a source of noise in prokaryotic cells.

The activation of promoters involved in lysine biosynthesis were analyzed by single fluorescence experiments and dual-fluorescence system at single cell level. By placing the promoter regions of those gene associated with lysine biosynthesis upstream of *gfp*, the dynamic behavior of promoter activation was well visualized and quantitatively analyzed. This process was simulated by two fundamental ways, a simple deterministic process and a stochastic simulation algorithm. The two group parameters gotten from the different simulation methods, which used to describe the promoter activation, were validated by each other. The results provide a better quantitative understanding of the promoter dynamics in the lysine biosynthesis.

Chapter 1

Introduction

Cells are matter that dances.

Uri Alon

1.1 Systems Biology

Like a sentence we understand is not only the assembly of letters but words in a whole, a living cell is not just an assembly of genes and proteins. Its properties could not be fully understood merely by drawing diagrams of their interconnections. In addition, pathways are traditionally drawn as separated linear entities and then connected by the shared parts to form a comprehensive diagram. However, this rather reflects the history of how they were discovered than their real functional context. Therefore it is largely unknown how biological response specificity is encoded through biochemical activation kinetics among these separated entities. How does this pathway specify different biological responses from the others? How does these isolated pathways in previous study work together and embedded into a network? To answer these questions we should view a living cell as a dynamical system with integration of pathway crosstalk and the versatility of component function. That means we should understand biology at the system level(Kitano 2002).

Systems biology is a newly emerging biological field that aims to understand various complex life phenomena at a system level(Kim et al. 2008). It studies the living organisms as a network of interacting parts and seeks to understand how this network gives rise to the functional aspects of life. These networks are modular, robust and

predictable(Aderem 2005). To understand living networks at a deep level will require the transition of biology from a descriptive to a quantitative science. This quantitative process gives birth to the models which have to faithfully describe the biological system and be able to make predictions about their behavior. This predictive power can then be exploited by incorporating descriptions of perturbations of the biological system into the model and using computational techniques to predict possible behaviors of the system(Kolcha et al. 2005). In a short words, systems biology is an approach to biology that seeks to understand and predict the quantitative features of a multicomponent biological system(Kitano 2002).

In practice side, systems biology promises to personalize medicine via network-based biomarkers that predict therapeutic effectiveness. The pathogenicity of human pathogens varies from person to person, which makes the treatment and dosage for same disease should be personal (Brynildsen and Collins 2009). Goh et al. (2007) drew a network map of disorders and disease genes linked by known disorder-gene associations in order to offer a platform to indicate the common genetic origin of diseases. To acquire a deeper understanding of this graph-network structure, Chang et al. (2009) introduced a systems-based approach to break down oncogenic signaling networks into modules that predict the effectiveness of pathway-specific therapeutics.

1.2 Promoter Strength

To get the comprehensive metabolic landscape in living cells needs the quantitative understanding in global gene regulation networks. Although there are large mass of biological data by use of the high-throughput omics technologies (*e.g.* genomics, transcriptomics, proteomics, and metabolomics), there still lack the details for systems biology to generate the accurate and undubious models in the endogenous regulation network(Kim et al. 2008).

More and more details and steps involved in transcription were discussed(Zhou and Yang 2006, Mijakovic et al. 2005, CoxIII et al. 2007, Wray et al. 2003). One of the most important parts is the question how the cells initiate the transcription. The promoter, the region of DNA with specific sequences, is known to play a central role in driving gene transcription(Browning and Busby 2004b). The ability to determine the frequency of initiation of transcription is called **promoter strength**(Lu et al. 2004). To use most suitable model with appropriate parameters to simulate the initiation of transcription will do great contribution in systems biology.

1.3 Time-dependent experiments

There are many functional modules, such as enhancer, booster, activator, insulator, repressor, locus control region, upstream activating sequence, and upstream repressing sequence, that contribute to the promoter strength(Wray et al. 2003). However, most of the endogenous promoter in cell lack the information about the detail of the regulation mechanism. But frequently researchers found a given factor could activate or repress the promoter strength. Dynamic responses of promoter strength to various environmental stimuli can be easily gotten by time-dependent experiments. **Time-dependent experiments** is the experiments to monitor the change of target attribute or properties via a time series sample in a given condition.

The time-dependent experiments can result in wealth of information such as behavior changes when stimuli is modified, which makes the model construction feasible(Kitano 2002). System dynamics, one of the key properties in systems biology, resolves how a system behaves over time under various conditions. The central goals of system dynamics is to predict the dynamic behavior of a cell's genetic and metabolic networks(Mettetal et al. 2006). Living organisms require a continual input of free energy from its environment and synthesis of macromolecules for the purpose

of maintenance of life. Sum of all the chemical reactions that take place in every cell of living organism, providing energy for the processes of life and synthesizing new cellular material is referred to as metabolism. Metabolism is composed of many coupled, interconnecting reactions. How the intricate network of reactions in metabolism is coordinated is the key research topic for metabolism(Berg et al. 2002). As to systems biology, the question change to the transition from a descriptive to a quantitative understanding. Because of the complexity of biological systems, this goal requires the use of mathematical models that provide a framework for determining the outcome of numerous and simultaneous time-dependent and space-dependent processes(Jaqaman and Danuser 2006a). Dynamic systems responses to various environmental stimuli can be elucidated by systems modeling of signaling pathways. The time-dependent experiments can provide lots of available data for determining model parameters. After using maximum likelihood(ML) and least squares (LS) to regress the parameters and using the model goodness-of-fit test(GFT) to identify most suitable model, the biological processes can be described and predicted by mathematical models(Kim et al. 2008).

1.4 Reporter Genes

To determine the strength of any given promoter, a reporter gene is usually driven under its control. Classical reporter genes include *lacZ*(β-galactosidase), *gusA*(β-glucuronidase), *cat*(chloramphenicol acetyl transferase), *lux*(luciferase) and *gfp, yfp, cfp, rfp*(fluorescent protein)(Mijakovic et al. 2005, Lu et al. 2004). These can be readily adapted for comparative promoter studies. Fluorescent proteins are genetically encoded, easily imaged reporters crucial in biology and biotechnology.

Green fluorescent protein(GFP) from *Aquorea victoria* emits green light (λ_{max} 508nm) on excitation at 395nm. It has become an invaluable tool for pure and applied biological research. Mutagenesis of the wildtype yielded improved variants optimized for flow

cytometer analysis(Cormack et al. 1996, Suzuki et al. 2004). *gfpuv5* is the mutant developed by Suzuki et al. GFPuv5 emits green fluorescence at λ 511nm on exitation at λ 488nm(Suzuki et al. 2004), which can be easily used for quantitative analysis by flow cytometry.

The most useful of red fluorescent protein(RFP) for dual-color experiments with GFP is DsRed, which is derived from the coral *Discosoma.* The mutation *dsred*-T4 is a fast maturing variants of *Discosoma* red fluorescent protein developed by Bevis et al., which emits red fluorescence at λ 586nm(Bevis and Glick 2002).

Both of the GFPuv5 and DsRed have a rapid maturation, high brightness and suitable emission for flow cytometry analysis. And they are verified that their fluorescence will not lead crosstalk in dual-color experiments(Bevis and Glick 2002).

1.5 Single Cell Analysis

To conduct a transition from a descriptive to a quantitative understanding, a comprehensive set of quantitative data is required. With the development of advanced biotechnology, more and more accurate single cell level measurements were introduced into systems biology. These single-cellular analysis techniques include flow cytometry, optical well arrays, fluorescence microscopy, electrochemical detection, Raman microspectroscopy, capillary electrophoresis with laser-induced fluorescence detection (CE-LIF) biomolecules, CE-LIF organelles, matrix-assisted laser desorption/ionization mass spectrometry (MALDI-MS), laser capture microdissection (LCM) and cDNA microarray analysis, and multiplexed real-time RT-PCR(Arriaga 2009). **Flow cytometry**, a powerful technique for analyzing large populations of single cells, allows simultaneous multiparametric analysis of the physical and/or chemical characteristics of single cells flowing through an optical/electronic detection apparatus with high speed and low economic cost. In the field of promoter activation analysis it is especially useful when used

with fluorescence proteins under the control of target promoter(Ducrest et al. 2002).

Figure 1.1 showing the results of two typical experiments in which the expression level of a fluorescent reporter protein is measured in a population of isogenic bacterial cells. Traditional population-averaged measurements would summarize the entire expression histogram by its mean value, however, observation by flow cytometry shows that the expression level varies from cell to cell, with a standard deviation σ. As we know stochastic mechanisms are ubiquitous in biological systems(Ozbudak et al. 2002). Isogenic cells and organisms exhibit distinct diversity to respond to a given concentration of a stimulus. Noise, or variation, in the process of gene expression (intrinsic noise) and in cellular components (extrinsic noise) may contribute to such kind of ubiquitous phenomenon in biological systems(Elowitz et al. 2002). Analyzing the data by the technique of using flow cytometry can help us to study promoter strength more comprehensively and accurately with respect to stochastic noise as a whole.

1.6 Biological Noise

The development of live cell and biochemical analysis methods has led to an increase in our understanding of transcription profiles of genes. Researchers found even in a population of genetically identical cells experiencing the same environment, protein contents very from cell to cell. The quantitative description of such fluctuations is termed **biological noise**(Arriaga 2009).

In order to describe the stochastic or noisy process of gene expression, coefficient of variation, a conception in mathematics and statistics, was employed. In probability theory and statistics, the coefficient of variation (CV) is a normalized measure of dispersion of a probability distribution. It is defined as the ratio of the standard deviation to the mean,

$$\mathrm{CV} = \frac{\sqrt{\mathrm{D}(x)}}{\mathrm{E}(x)} \tag{1.1}$$

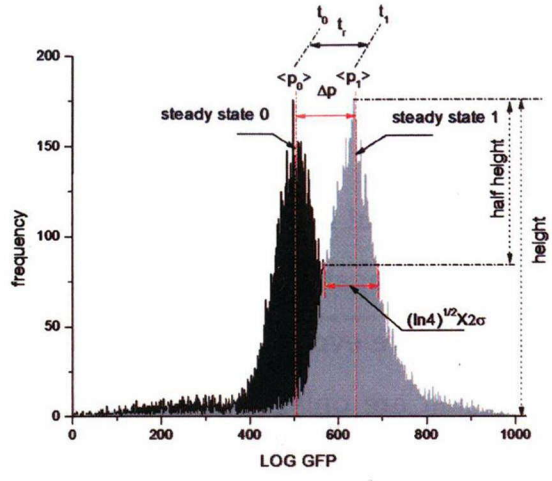


Figure 1.1: Illustration of steady-state sensitivity. The peaks in different color show two group data of LOG GFP detected by Flow cytometry. Traditional population-averaged measurements would summarize the entire histogram by the concentration of protein at peak position $\langle p \rangle$; however, our single-cell measurements show that the expression level varies from cell to cell, with a standard deviation σ. σ can be calculated by the peak width at half height if we fit the bell shape curve as normal distribution. After changing the components in the environment, the cell would respond to this change from steady state 0 to steady state 1, which can be well visualized and quantitatively analyzed by flow cytometry.

, where $\mathtt{E}(x)$ is mean value and $\mathtt{D}(x)$ is the deviation. A natural and biologically relevant measure of the magnitude of gene expression noise is thus the size of protein fluctuations compared to their mean concentration. In Figure 1.1, the mean value $\langle p(t) \rangle$ represent the gene expression at time t. Then the noise, $\eta(t)$, is given by

$$\eta^2(t) = \frac{\langle \mathtt{P}(t)^2 \rangle - \langle \mathtt{P}(t) \rangle^2}{\langle \mathtt{P}(t) \rangle^2} \tag{1.2}$$

, where the angled brackets denote an average over the probability distribution of $\mathtt{P}$ at time t.

Noise confers lots of advantages, as it brings diversity to cells. This diversity can provide a better chance at survival in uncertain environments(Fraser and Kærn 2009). Both experiments and simulations confirmed that increased gene expression noise could provide a significant selective advantage at high stress levels(Blake et al. 2006).The rapid fluctuations in gene expression noise could determine the cell fate. It is re-

ported that the competence of *Bacillus subtilis* was determined by the noise expression of ComK(Maamar et al. 2007). And it was testified that the phage λ lysis-lysogeny decision circuit has relation with biological noise(Arkin et al. 1998).

However more often it is harmful for regulation networks as it garbles cell signals, corrupts circadian clocks and propagates unstable from one gene to a downstream target. Cell regulation networks have evolved so as to minimize the disruptive effect of such fluctuations, in ways that are only now beginning to be understood(Ozbudak et al. 2002).

It is expected that control of noise in gene expression is under evolutionary pressure. Several models talking about how cell control the intrinsic noise of gene expression were reviewed(Raser and O'Shea 2004). Many simplified theoretical models were employed to show how the difference of transcription, translation, promoter activation, gene copy number and gene feedback loops contribute to controlling the noise.

However, most of the experimental studies have so far focused on noise property itself. To apply this method to analysis the strength of a promoter involved in endogenous gene networks, particularly in metabolism network, is seldom reported. A comprehensive understanding of design strategies used by endogenous transcriptional regulatory programs might require a stochastic perspective. We have barely scratched the surface of this intriguing topic, and there is a clear need to address in greater detail how gene expression responds to fluctuations in signal transduction, how gene-expression noise is transmitted through regulatory circuits and control loops, and how the architecture of regulatory networks allows cells to deal with or take advantage of unreliable, fluctuating signals(Kærn et al. 2005). We need an entrance point. The clear background of lysine biosynthesis (Figure 1.2) provides a chance of challenge in the study of stochasticity in more complex regulatory systems.

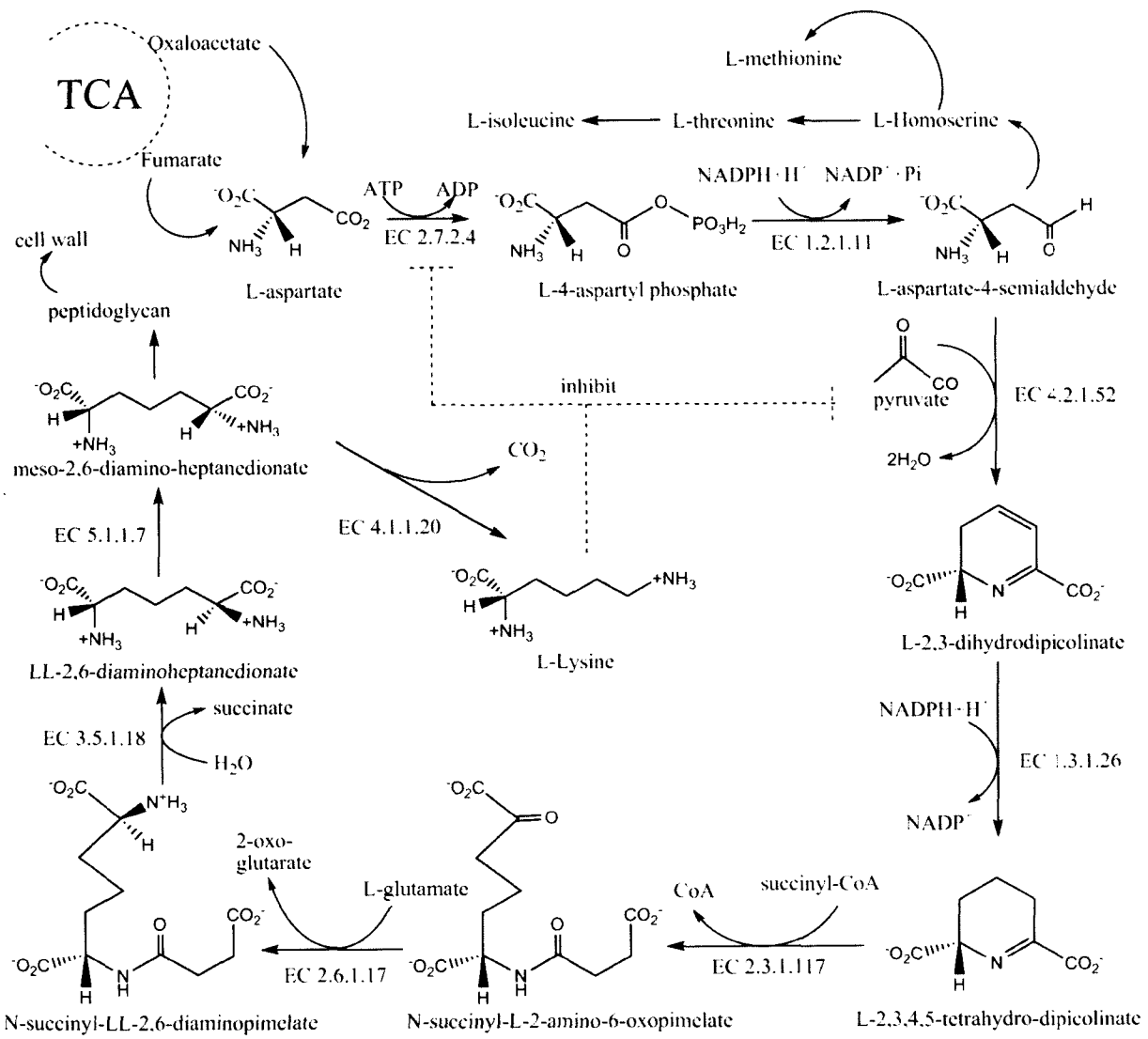


Figure 1.2: Lysine biosynthesis metabolic network involving nine enzyme reactions.

1.7 Lysine Biosynthesis

An analysis of amino acid metabolism is important for the progress of systems biology, because the role of the metabolic system, *i.e.,* to provide building blocks for the entirety of cellular dynamics, is essential for the maintenance of life, and the regulation of metabolic reactions has been thoroughly investigated. **Lysine biosynthesis**, one of the important components of metabolic networks, is a pathway starting with aspartate and runs through the diaminopimelate pathway in *E. coli*(Rodionov et al. 2003).

From previous studies(Chenais et al. 1981, Liao and Hseu 1998, Haziza et al. 1982, Richaud et al. 1986, Bouvier et al. 1992, Richaud et al. 1984, Bouvier et al. 1984, Funkhouser et al. 1974, Jin et al. 2004, stragier et al. 1983, Bouvier et al. 2008), we have obtained an elementary understanding of the lysine biosynthesis network structure (Figure 1.3). As shown in Figure 1.3, the expressions of most genes involved in the lysine biosynthesis in *E. coli* are repressed by lysine. Diaminopimelic acid (DAP) is the precursor of lysine in *E. coli*. The conversion of meso-DAP to lysine is catalyzed by DAP decarboxylase, the product of *lysA*(Chenais et al. 1981). The transcription of *lysA* is repressed by lysine. meso-DAP is synthesized from aspartic acid through the successive reactions of eight enzymes. The first two steps are catalyzed by the products of *lysC* and *asd*, and they are shared by the pathways leading to lysine, threonine, and methionine. The transcription of both *lysC*(Liao and Hseu 1998) and *asd*(Haziza et al. 1982) is repressed by lysine. The other six genes belong to the DAP pathway. The first gene of this pathway, *dapA*, is constitutively expressed, but the activity of its product, dihydrodipicolinate synthetase, is inhibited by lysine(Richaud et al. 1986). The transcriptions of *dapB*(Bouvier et al. 1984) and *dapD*(Richaud et al. 1984) are repressed by lysine. With regard to *dapC*, *dapE* and *dapF*, there is no evidence showing their expression regulation by lysine.

Although most genes involved in lysine biosynthesis in *E.coli* are repressed by ly-

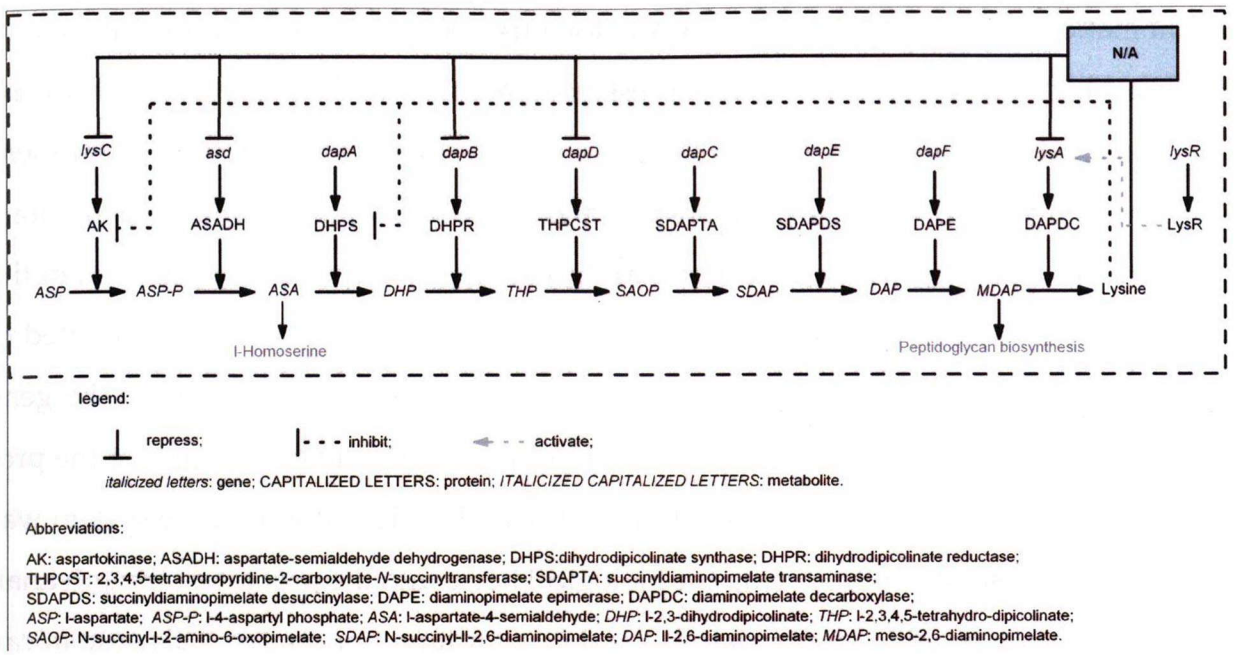


Figure 1.3: Lysine biosynthesis regulation network by L-lysine. "N/A" in the figure represents that the mechanism to regulate these genes is still unclear.

sine, little is known about their regulation mechanisms(Rodionov et al. 2003), shown as a gray box labeled as N/A in Figure 1.3. Lysine also inhibits the activities of aspartokinase III and dihydrodipicolinate synthase to regulate its synthesis by changing enzyme structures or by other mechanisms(Kotaka et al. 2006).

1.8 Object

The use of the kinetic cell model for describing cell behavior is necessary for a quantitative understanding of the metabolic regulation network, including lysine biosynthesis. Followed the deepened research in biological noise, there is a need to develop experiment in the study of promoter strength with the stochasticity property in more complex regulatory systems, particularly endogenous gene networks.

In this study, we tried to analyze the promoter strength via single cell technique. We analyzed the strength of promoters involved in lysine biosynthesis with single-fluorescence experiments and dual-fluorescence experiments by flow cytometry. The

combination of reporter strains and flow cytometry provided us with a convenient and accurate method of measuring gene expression dynamics. By the single fluorescence experiments, the dynamic response of the promoters to shift of lysine in the environment were well visualized and quantitatively analyzed by flow cytometry. Five promoters involved in lysine biosynthesis respond to the changes in L-lysine concentration in the medium. For these five promoters, time-dependent gene expression data were fitted to a simple dynamical model of gene expression to estimate the parameters of the gene regulatory system. The results provide a better quantitative understanding of the promoter dynamics in the lysine biosynthesis pathway. The dual-fluorescence system was verified to be useful in introducing the biological noise into the promoter strength analysis. And it can provide appropriate k_{on} and k_{off} for target promoter. This system can be easily used for strength analysis of grouped promoters in an endogenous regulation network. The parameters of k_{on} and k_{off} can not only describe the activation of repression by a given factor but also provide information about the biological noise for the promoter. This will help us to understand the initiation of transcription in a quantitative way and to predict the possible level of mRNA.

1.9 Outline of the thesis

This thesis consists of four chapters.

In **chapter 1**, the background of this study is represented. An overview of the thesis objective is provided.

In **chapter 2**, in order to conduct the transition from a qualitative to a quantitative understanding of the promoter involved in lysine biosynthesis in *E. coli* cells, the combination of reporter strains and flow cytometry was employed. The kinetic parameters of five genes involved in lysine biosynthesis were obtained by fitting the gene expression data to a simple dynamical model.

In **chapter 3**, a dual-fluorescence system for promoter strength analysis was developed to involve the biological noise information. This system was applied to analyze the kinetics of promoters involved in lysine biosynthesis.

In **chapter 4**, the general conclusion of this research was made.

Published as: Jianhong Ou, *et.al.* – *"Dynamic change in promoter activation during lysine biosynthesis in Escherichia coli cells."* Molecular BioSystems, vol. 4, pp. 128–134, 2008.

Chapter 2

Dynamic changes in promoter activation analyzed by single fluorescence experiments

Abstract

We investigated the expression dynamics of genes involved in lysine biosynthesis in *Escherichia coli* cells at single cell level to obtain a quantitative understanding of the gene regulatory system. By constructing reporter strains expressing the green fluorescence protein gene(*gfp*) under the control of the promoter regions of those genes associated with lysine biosynthesis, time-dependent changes in gene expression in response to changes in L-lysine concentration in the medium were monitored by flow cytometry. Five promoters involved in lysine biosynthesis respond to the changes in L-lysine concentration in the medium. For these five promoters, time-dependent gene expression data were fitted to a simple dynamical model of gene expression to estimate the parameters of the gene regulatory system. According to the fitting parameters, *dapD* shows a significantly larger coefficient of repression than the other genes in the lysine synthesis pathway, which indicates the weak binding activity of the repressor to the *dapD* promoter region. Moreover, there is a trend that the closer an enzyme is to the start of the lysine biosynthesis pathway, the smaller its maximal promoter activity is. The results provide a better quantitative understanding of the expression dynamics in the lysine biosynthesis pathway.

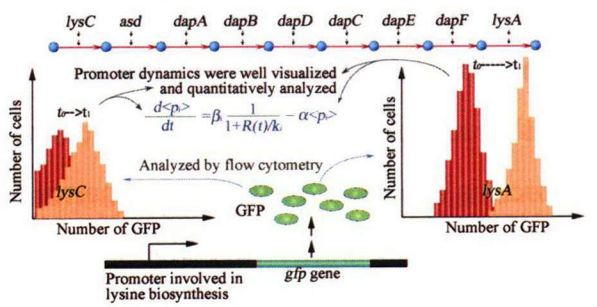


Figure 2.1: The dynamics of gene promoters involved in lysine biosynthesis in *E. coli* cells in response to changes in L-lysine concentration was investigated. The results provide a better quantitative understanding of the expression dynamics in the lysine biosynthesis pathway.

2.1 Introduction

Systems biology is the study of living organisms as a network of interacting parts and of how this network gives rise to the functional aspects of life. These networks are modular, robust and predictable(Aderem 2005). A deep understanding of these living networks requires the transition of biology from a descriptive to a quantitative science. This quantitative process gives birth to models that have to faithfully describe biological systems and can predict their behavior(Kolcha et al. 2005).

System dynamics, one of the important aspects of systems biology(Kitano 2002), resolves how a system behaves over time under various conditions. One of the central goals of systems biology is to predict the dynamic behavior of genetic and metabolic networks(Mettetal et al. 2006) in a cell. Living organisms require continuous inputs of free energy from their environment and synthesize macromolecules to maintain life. The sum of all the chemical reactions that take place in every cell of living organisms, providing energy for the processes of life and for the synthesis of new cellular material, is referred as metabolism. Metabolism involves many coupled, interconnecting reactions. The key question concerning metabolism is how it coordinates its intricate network of reactions(Berg et al. 2002). A quantitative understanding of the reaction dynamics associated with metabolism can help answer this question.

An analysis of amino acid metabolism is important for the progress of systems biology, because the role of this metabolic system, i.e., to provide building blocks for the entirety of cellular dynamics, is essential for the maintenance of life, and the regulation of metabolic reactions has been thoroughly investigated. Lysine biosynthesis, one of the important components of metabolic networks, is a pathway starting with aspartate and runs through the diaminopimelate pathway in E. coli(Rodionov et al. 2003). And its elementary understanding in network structure was obtained (Figure 1.3).

Although many studies have provided insight into the "correlations" between genes

and biological phenomena in the regulation network(Rodionov et al. 2003), little is known concerning the connection between these genes. The use of the kinetic cell model for describing cell behavior is necessary for a quantitative understanding of the metabolic regulation network, including lysine biosynthesis. The promoter has the ability to determine the frequency of initiation of transcription, which makes it always a hot spot in systems biology(Browning and Busby 2004b). This ability is denoted by the promoter strength(Lu et al. 2004). To determine the strength of any given promoter, a reporter gene is usually driven under its control(Mijakovic et al. 2005). If inserting a promoter upstream of the green fluorescent protein(*gfp*) gene, promoter activity can be monitored by the intensity of green fluorescence emitted by GFP(Lu et al. 2004).

Time-dependent experiments are commonly used to achieve a quantitative understanding, which usually involve monitoring groups of cells over their cycles or as they respond to time-dependent changes in conditions in the extracellular medium(Sayyed-Ahmad et al. 2007). Because most genes involved in lysine biosynthesis in *E. coli* are directly or indirectly repressed by L-lysine, after changing the L-lysine concentration in the medium, the expression of enzymes involved in lysine biosynthesis may shift from one steady state to another. Analyses of their time-dependent expression will enable the prediction of these phenomena associated with the dynamic variable.

In this study, we investigated the activation dynamics of gene promoters involved in the lysine biosynthesis pathway of *E. coli* cells via time-dependent experiments. We constructed nine reporter strains by cloning the promoter regions of the genes involved in lysine biosynthesis pathway upstream of the gene encoding the green fluorescence protein (*gfp*). By flow cytometry, we determined the dynamic changes in gene expressions in response to changing environmental conditions (*i.e.*, L-lysine concentration) at single cell level. We fit these experimental data of gene expression dynamics to a simple kinetic model with repression to estimate the parameters of the gene regulatory system. The results provide a better understanding of the gene expression dynamics in the ly-

sine biosynthesis pathway. But however, we failed in including noise information into the parameters.

2.2 Material and methods

2.2.1 Strains and media

Nine reporter strains were used in this study, *E. coli* DH1 (K-12 *endA1 recA1 gyrA96 thi-1 glnV44 relA1 hsdR17(rK⁻mK⁺) λ⁻*)/p*lysC*p-pGFP$_{uv5}$ (*lysC*p in short), DH1/p*asd*p-pGFP$_{uv5}$ (*asd*p), DH1/p*dapA*p-pGFP$_{uv5}$ (*dapA*p), DH1/p*dapB*p-pGFP$_{uv5}$ (*dapB*p), DH1/p*dapD*p-pGFP$_{uv5}$ (*dapD*p), DH1/p*dapC*p-pGFP$_{uv5}$ (*dapC*p), DH1/p*dapE*p-pGFP$_{uv5}$ (*dapE*p), DH1/p*dapF*p-pGFP$_{uv5}$ (*dapF*p), and DH1/p*lysA*p-pGFP$_{uv5}$ (*lysA*p) which contain the promoter regions of genes (i.e. *lysC, asd, dapA, dapB, dapD, dapC, dapE, dapF,* and *lysA*), respectively.

The promoter regions were amplified from *E. coli* DH1 genomic DNA by PCR(Table 2.1). PCR is performed as follows: 94°C for 5 min, followed by 30 cycles of 94°C for 30 s, 60°C for 30 s and 72°C for 1 min, and a final step of 72°C for 7 min. The primers listed in Table 2.22 were used to amplify the regions between two adjacent open reading frames(ORFs) with an extension of 150–200 nt upstream of the ORFs(Figure 2.2). This definition of promoter region is the usual practice in many promoter researches (Zaslaver et al. 2004). The promoter regions were cloned at *Apa*I and *Nhe*I sites upstream of gfpuv5, which is a variant of *gfp*(Ito et al. 2004)(Figure 2.3). Both of the vector and recovered promomter PCR fragments are digested for 6 h at 37°C with *Nhe*I and *Apa*I restriction enzymes(Table 2.2), and then cleaned by using Promega® gel/PCR clean-up system into 20 μL of TE(pH8.0). Before recover, the vector is performed one more dephosphorylation step that it is digested by Bacterial Alkaline Phosphatase at 37°C and 60°C one after another separately for 30 min each reaction. After that, ligation reaction is performed overnight at 16°C(Table 2.3).

Table 2.1: PCR reaction mixture for amplification of promoters involved in lysine biosynthesis

Components	Amount(μL)
Ex taq 10× buffer	10
dNTP Mixture, 25mM	10
Escherichia coli DH1 fresh culture	5
Forward primer (20pMol/μL)	5
Reverse primer (20pMol/μL)	5
TaKaRa Ex Taq™ (5 units/μL)	1
Nuclease-Free water to a final volume of	100

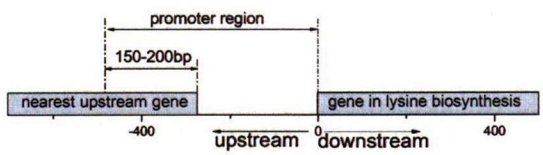


Figure 2.2: Definition of promoter region in this study

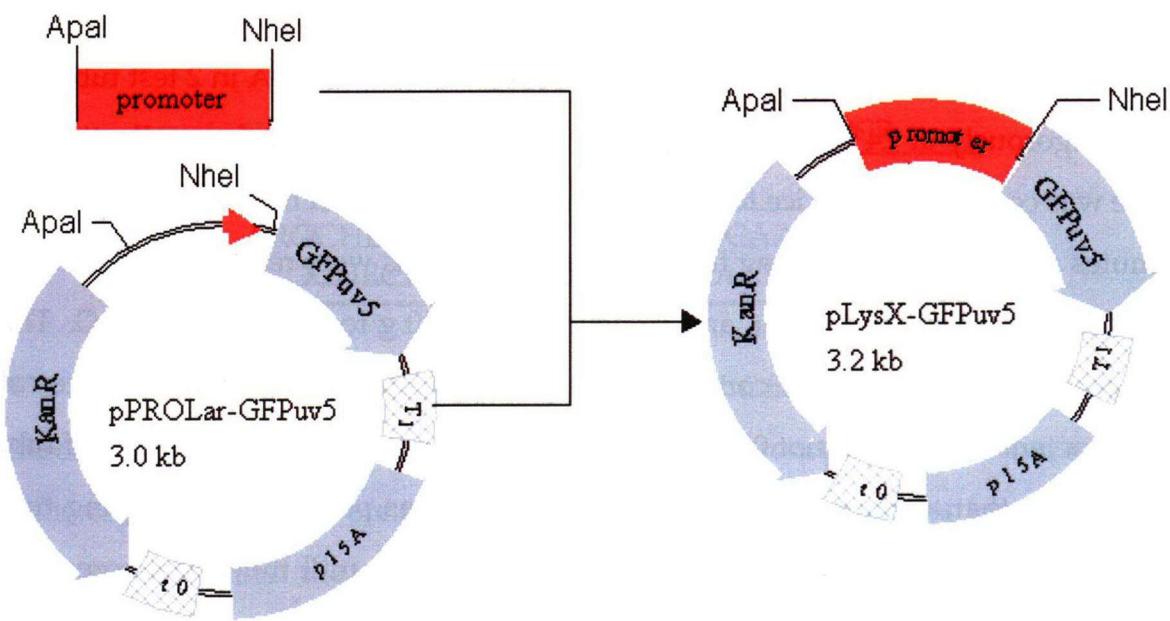


Figure 2.3: Illustration for strategy of reporter strains construction. Reporter plasmid pPROLar-GFPuv5: pPROLar contains the gene of *gfpuv5*. T1,t0: transcription termination sequence; p15A: origin of replication (20–30 copies per cell); KanR: kanamycin resistance gene; GFPuv5: green fluorescence protein gene uv5 mutant, which fluorescence excited at λ 511nm and emit at λ 488nm

Table 2.2: Enzyme digestion reaction mixture for vector and PCR fragments

Components	Amount(μL)
pPROLar-GFPuv5 or PCR products	5 or 10
L Buffer(Takara)	2
*Nhe*I(Takara)	1
*Apa*I(Takara)	1
Nuclease-Free water to a final volume of	20

Table 2.3: Ligation reaction mixture

Components	Amount(μL)
pPROLar-GFPuv5(30ng/μL)	1
PCR products(25ng/μL)	4
Takara Ligation Kit I	5

Competent *E. coli* DH1 cells were prepared by Z-competent *E. coli* transformation kit(Zymo^{TM} Research, USA) and buffer set as manual. 0.5 mL of fresh overnight *E. coli* DH1 culture grown in M9 minimal medium with an amino acid solution (M9A medium)(Ford et al. 1994)(Table 2.23) was inoculated to 10 mL M9A in 2 test tubes and shaken vigorously at 24°C until OD_{600} reaches around 0.2–0.3. Before collecting, the culture was cooled down on ice for 10 min, and then the cells were pelleted at 2500g for 6 minutes at 4°C. After removing the supernatant the cells were resuspended gently in 5mL of ice cold 1× Wash Buffer, and re-pelleted at 2500 g for 6 minutes at 4°C. Then the supernatant was removed completely and resuspended gently in 5mL of ice cold 1× Competent Buffer. Aliquot 0.1 mL of the cells was put into sterile Eppendorf tubes on ice. After that, competent cells were frozen as fast as possible by dropping tubes immediately in a small liquid nitrogen container and left until they were completely frozen and all aliquots had been made. Competent cells were stored at –80°C freezer before use.

While transformation, a tube of frozen competent cells was thawn on ice, 1 μL ligation product was added and mixed gently. The mixture was incubated on ice for 45

min. 400 μL of M9A media was added and incubated at 37°C for 30 min before spreading 50 μL on M9A plates with 25 μg/mL of kanamycin. The plate was incubated at 37°C overnight.

Positive colonies were screened by colony PCR (reaction solution as shown in Table 2.4 and picked the colony into the solution by yellow tip) using primers designed for the outer regions of the two restriction sites, *Apa*I and *Nhe*I. Primer sequences are shown in Table 2.5. PCR conditions were 94°C for 5 min, followed by 30 cycles of

Table 2.4: Colony PCR reaction solution

Components	Amount(μL)
Ex taq 10× buffer	2
dNTP Mixture, 25mM	2
PROCHECK-F (20pMol/μL)	1
PROCHECK-R (20pMol/μL)	1
TaKaRa Ex Taq™ (5 units/μL)	0.2
Nuclease-Free water to a final volume of	20

Table 2.5: Primer sequences for colony PCR

Primer name	Sequences
PROCHECK-F:	TCCTTGGCGGCAAGAAAGCC
PROCHECK-R:	CTGACAGAAAATTTGTGCCC

94°C for 30 s, 60°C for 30 s and 72°C for 1 min, and a final step of 72°C for 7 min. The size of PCR products were analyzed by agarose gels. Frozen stocks (15%(w/v) glycerol) of the reporter strains were prepared. 4 randomly selected clones for each promoter were sequenced and the reporter strains with no mutations were selected. 100 μL of plasmid prepared for sequencing by Mini Plasmid DNA Purification Kit(Labopass®) was treated by adding an equal volume of buffer-saturated phenol:chloroform (1:1) to the DNA solution, mixing for 10 s by vortex and then spinning in a microfuge for 15 min at 15,000 rpm. The aqueous layer was carefully removed to a new tube, and the

salt concentration was adjusted by adding 1/10 volume of 3M sodium acetate, pH 5.2, (final concentration of 0.3 M). The plasmids were precipitated by 2 to 2.5 volumes of cold 100% ethanol (calculated after salt addition). After cooling treatment on ice or at –20°C for over 20 min, the plasmids were collected 20 min at 15,000rpm. Supernatant was discarded and the plasmid was washed by 1 mL 70% ethanol by mixing. Pellet was dried by vacuum. After resuspending the pellet in 30 μL of TE (pH 8.0), the sample for sequence was cleaned up. A 20 μL reaction was prepared by adding the following reagents into the 5 μL of thawed Big Dye Ready Reaction Mix(Table 2.6).

Table 2.6: Sequencing reaction solution

Component	Amount
thawed Big Dye Ready Reaction Mix	5 μL
PROCHECK-F or PROCHECK-R (0.8 pmol/μL)	4 μL
Template fragment for sequencing	100~200 ng
Nuclease-Free water to a final volume of	20 μL

After well mixing by flicking tube, PCR reaction was run as follow condition: 96°C for 2 min, followed by 25 cycles of 96°C for 10 s, 50°C for 5 s and 60°C for 4 min. 2 μL of 3 M sodium acetate was added, pH 5.2 into the PCR product. The amplified DNA was precipitated by cold 100% ethanol at –20°C for 30 min. The precipitation was collected by 20 min at 15000 rpm and then washed by 70% ethanol. After briefly vacuum drying, the pellet was added into 20 μL of TSR (Template Suppression Reagent, Applied Biosystems™), mixed thoroughly on a vortex mixer and denatured for 2 min at 100°C. Then samples were chilled on ice for 5 min, vortexed to mix and spined at 13,000 rpm for 30 s to collect volume at bottom of tube. Entire volume of sample was transfered to sample tube and caped with grey rubber gasket. Sequencing was done by ABI Prism 3100 Instrument(Applied Biosystems™) within several hours. Before sequencing, the sequencing buffer and MilliQ water were changed for sequencing. All the reporter strains were confirmed by sequenceing.

Reporter strains were cultured in M9A media and kanamycin (25 μg/mL). It should be noted that the L-lysine in the amino acid solution was prepared separately, and the need to add L-lysine to the medium was decided according to the experimental condition.

2.2.2 Sample preparation

Before sample preparation, the growth rates of all the reporter strains were determined to ensure that the growth rates were stable at approximately 1.2 h^{-1}. The reporter strains were streaked on M9A minimal medium plate (1.5%[w/v] agar) from the stock and incubated at 37°C for 15 h. Preculture was carried out by picking a colony and inoculating it into 2 test tubes containing 5mL of fresh M9A medium (one test tube contained 0.3-mM L-lysine, while the other contained no L-lysine). The inoculated test tubes were then reciprocally shaken at 37°C and 160 rpm until the OD_{600} value reached 0.6–0.7. After the preculture, all the cells were collected by centrifugation at 6000 rpm for 5 min, followed by a re-suspension of the pellet in 1mL of M9A medium (with or without L-lysine, depending on the next culture step). The suspension were inoculated into test tubes containing 5 mL of M9A medium with or without L-lysine to make the initial OD_{600}=0.01 for main culture. The test tubes were reciprocally shaken at 37°C and 160 rpm. Every 15 min, samples for flow cytometry were prepared by sampling 1 mL of the culture and the samples were stored at –80°C until use.

2.2.3 Data acquisition and analysis

The samples were thawed before being analyzed using a flow cytometry (COULTER® EPICS® XL™, Beckman Coulter, Fullerton, CA) and then diluted with phosphate buffer solution (PBS, pH 7.0)(Table 2.7) to set the cell concentration at 10^7 cells/mL (OD_{600} = 0.01). Fluorescence measurements were obtained using flow cytometry. 20,000 cells from each sample were analyzed. GFP excitation was achieved using a 488-nm argon

(Ar^+) excitation laser and the fluorescence was measured with a 525±20 nm emission filter for GFP. The flow cytometry generates log-scale values using a 10-bit analog-to-digital converter, yielding integers in the range of 0 to 1,023 for each of three measurements: fluorescent intensity, forward-scattering (FSC), and side-scattering (SSC). The *lmd* files produced by the flow cytometry were converted into a Microsoft© Excel© document using EXPO© 7.0 (Beckman Coulter, Fullerton, CA). And use Equation (2.1) to convert the log scale values to fluorescent intensity. The GFP protein concentration $\langle p_i \rangle$ for each sample used in the data fitting was determined according to the peak position in the events histogram of GFP fluorescence after discrete wavelet transformation(AppendixB.1). To confirm the reproducibility of the changes in the fluorescence distribution dynamics, we performed two experiments under the same environmental conditions, and found that the measurement results were robust in these independent experiments. Thus, here we showed the results of one of these experiments.

$$\texttt{GFP-intensity} = 37.814 \times e^{(0.01 \times \texttt{LOG GFP})} \tag{2.1}$$

Table 2.7: Recipe of 1×PBS

Components	Company	Final Concentration(g/L)
K_2HPO_4	Wako	10.5
KH_2PO_4	Wako	4.5

2.2.4 Real-time reverse-transcription polymerase chain reaction (Real-time RT-PCR)

The culture of each strain and sampling were performed as described above. The RNA was isolated from these cultures using the RNeasy Mini kit (Promega™, Madison, WI) and then treated with DNase I (TaKaRa, Japan) at 37°C for 30 min (Table 2.8). The RNA purified by burrer-saturated phenol:chloroform (1:1) and precipitated by 100% ethanol.

Table 2.8: DNaseI digestion reaction solution

Components	Amount(μL)
Total RNA	30
10× DNaseI buffer	5
RNase inhibitor	0.5
DNaseI (RNase-free)	2
DEPC water	12.5

The RNA integrity was electrophoretically verified by ethidium bromide staining and by verifying that the OD_{260}/OD_{280nm} absorption ratio was greater than 2.0. After carrying out reverse transcription with random primers using the Promega® Reverse Transcription System (Promega™, Madison, WI) (Table 2.9), real-time PCR analysis was performed. In the real-time PCR analysis, the first-strand cDNA pool was mixed with SYBR© Green PCR Master Mix (Applied Biosystems™, Foster, CA) and a pair of highly purified salt-free primers, as shown in Table 2.22, to amplify the target or reference gene using the GeneAmp® 5700 Sequence Detection System (Applied Biosystems™, Foster, CA) with the following parameters: 50°C for 2 min and then 95°C for 10 min, followed by 40 cycles of 95°C for 15 s and 60°C for 1 min, and a final step starting from 60°C for generating the dissociation curve. To obtain calibration curves, seven serial two-fold dilutions of the first-strand cDNA pool were used as templates for the real-time PCR analysis. To confirm the accuracy and reproducibility of the real-time PCR analysis results, three independent experiments and three repeats within each LightCycler run were performed. The mathematical model used in the real-time PCR analysis was that of Pfaffl(Pfaffl 2001).

2.2.5 Western Blotting

The culture of each strain and sampling were performed as described above. 5 mL of cultures were collected by centrifuge and then the pellet was resuspended in 500 μL of PBS containing 1% protease inhibitor cocktail (SIGMA™, Louits, MO). The cell

Table 2.9: Reverse transcription reaction solution

Components	Amount(μL)
$MgCl_2$,25mM	4
Reverse transcription 10× buffer	2
dNTP mixture, 10mM	2
Recombinant RNasin® ribonuclease inhibitor	0.5
AMV reverse transcriptase (High Conc.)	15u
Random primers (0.5μg/μL)	1
Total RNA (200μg/μL)	5
Nuclease-Free Water to a final volume of	20

Table 2.10: 30% Acrylamide solution (Filtered and kept away from light)

concentration	reagent
29.2%	Acrylamide
0.8%	N,N'-Methylene bisacrylamide

suspension was sonicated with 10 short burst of 10 s followed by intervals of 30 s for cooling. After that cell debris were removed by centrifugation at 4°C for 20 min at 15000 rpm. The protein concentration was determined by Bradford method(Bio-Rad™). 15 g of protein in loading buffer were incubated at 95°C for 10 min, cooled and then loaded per lane. Gel electrophoresis was performed using 12.5% gel (Table 2.18) with 0.1% (w/v) SDS under a constant voltage of 100 V and then transferred to HybondTM-P® membrane (GE Healthcare Bio-Sciences™ Corp. Piscataway, NJ) under a constant current of 40 mA for 1.5 h. The membranes were blocked for 1 h at room temperature with 5% milk in PBS(Table 2.21). Membranes were incubated with primary antibody (Promega™, Madison, WI) dilution overnight at 4°C. After wash, the membranes were incubated with secondary antibody (Promega™, Madison, WI) at room temperature for 1 h. After wash, the proteins were detected with the ECL Plus™ (GE Healthcare Bio-Sciences™ Corp. Piscataway, NJ). Densitometric intensity of the exposed film was determined by ImageJ© (Wayne Rasband, National Institutes of Health, USA).

Table 2.11: 4× Running gel buffer (1.5M Tris-HCl, 0.4% SDS, pH8.8)

Tris base	36.4g
SDS	0.8g
Adjust pH to 8.8 with HCl	
MilliQ	up to 200mL

Table 2.12: 4× Stacking gel buffer (0.5M Tris-HCl, 0.4% SDS, pH6.8)

Tris base	6.8g
SDS	0.4g
Adjust pH to 6.8 with HCl	
MilliQ	up to 100mL

Table 2.13: 3× Sample buffer(Add 6% β-mercaptoethanol before use)

4× Stacking gel buffer	37.5mL
Glycerol	30mL
SDS	6g
Bromophenol blue (BPB)	1.5mg
MilliQ	up to 97mL

Table 2.14: 50mM Sodium phosphate buffer(pH7.0)

Na_2HPO_4-$12H_2O$	2.185g
NaH_2PO_4-$2H_2O$	0.608g
MilliQ	up to 200mL

Table 2.15: Running Buffer

Tris base	1.8g
Glycine	8.64g
SDS	0.3g
MilliQ	up to 300mL

Table 2.16: CBB staining solution

CBB	0.5g
Methanol	100mL
Acetate	20mL
MilliQ	80mL

Table 2.17: CBB destaining solution

Methanol	125mL
Acetate	50mL
MilliQ	325mL

Table 2.18: 12.5% SDS Running gel

30%Acrylamide solution	3.33mL
4×Running gel buffer	2mL
MilliQ	2.67mL
Total	8mL
10%Ammonium persulfate (APS)	0.096mL
TEMED*(Add just before making gel)	0.01mL

Table 2.19: Stacking gel

30%Acrylamide solution	0.5mL
4×Running gel buffer	1mL
MilliQ	2.5mL
Total	4mL
10%APS	0.05mL
TEMED*(Add just before making gel)	0.01mL

Table 2.20: Western blotting transfer buffer

Tris base	0.909g
Glycine	4.32g
MilliQ	up to 240mL
Methanol (add just before use)	60mL

Table 2.21: Western blotting phosphate buffered saline

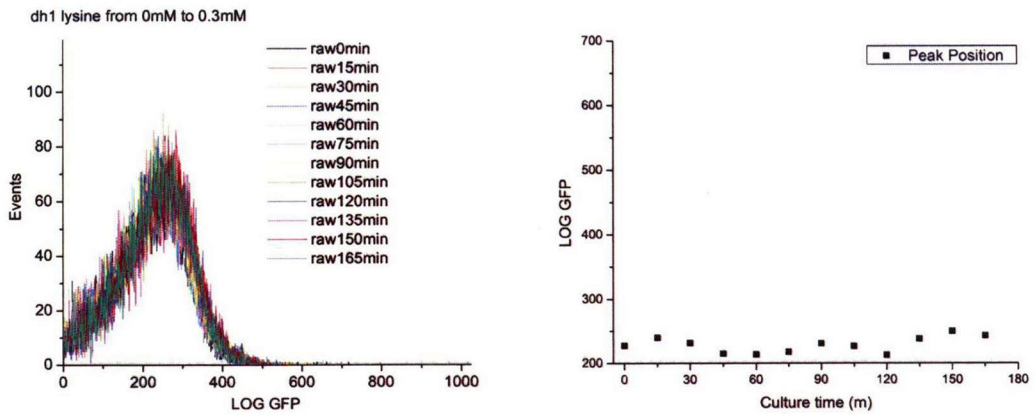
Na_2HPO_4	28.65g
NaH_2PO_4	2.40g
NaCl	5.84g
MilliQ	up to 1L

Table 2.22: Oligonucleotides used in this study

Primer name	Primer sequence (5′→3′)
ApaI-lysCp-f	CTACTAGGGCCCCAGCATCTGATCGTCGAAGG
NheI-lysCp-r	CTACTAGCTAGCCATAACTACCTCGTGTCAGGGGA
ApaI-asdp-f	CTACTAGGGCCCCACCAGGAGAGCAATAAGCA
NheI-asdp-r	CTACTAGCTAGCCATAAGCGTTTTTTTCCTGCAAA
ApaI-dapAp-f	CTACTAGGGCCCATGACGGGTGATGGTGTTCA
NheI-dapAp-r	CTACTAGCTAGCCATGGGCCATCCTCTGTGCAAAC
ApaI-dapBp-f	CTACTAGGGCCCGTGGAAACTCAGGGCGAATT
NheI-dapBp-r	CTACTAGCTAGCCATAGCTATTCTCTTTTGTTAAT
ApaI-dapDp-f	CTACTAGGGCCCCTTCATGGTGCCCGAATTAC
NheI-dapDp-r	CTACTAGCTAGCCATTGTTAAACTCTTTTCATATC
ApaI-dapCp-f	CTACTAGGGCCCGCGTGCTTTGACGTGACGGC
NheI-dapCp-r	CTACTAGCTAGCCATCTCATGATCACCCTGTTACG
ApaI-dapFp-f	CTACTAGGGCCCAGCGAAAGATTTGTCTCTTC
ApaI-dapFp-r	CTACTAGCTAGCCTTACTCCAATCACGCGGGTA
ApaI-lysAp-f	CTACTAGGGCCCAATTTCAGCCCGATCACCT
ApaI-lysAp-r	CTACTAGCTAGCCATAACAAACTCCAGATAAGTGC
ApaI-lysRp-f	CTACTAGGGCCCGCGCACCACATCAAACTGTT
ApaI-lysRp-r	CTACTAGCTAGCCATTAGCGCTCTCTCGCAATCCG
Primers for real-time RT-PCR	**Primer sequence (5′→3′)**
lysC-f	TGG CGA GCG ATT CGA AA
lysC-r	CCA GAA TGG CAA ACT GGA TGT
asd-f	CGG CTG GCG CGG TAT
asd-r	AAG TCG CGC TCT TCA ACC AT
dapA-f	GGG TTA TTT CCG TTA CGG CTA A
dapA-r	TGC TGC CAG TTT GCA CAT CT
dapB-f	ACG CTG AAC CAT CTC GCT TT
dapB-r	CCC CGT AGT GCC GAT CAC
dapD-f	TGC GTG TAG CGG AAA AAA TTG
dapD-r	GCA GCA CCG CTT TTT TCA A
dapC-f	TGG TCG CTC GCT GTT TAC C
dapC-r	CCC AAA GCC GTC GGA AT
dapF-f	GGA AAG CCA CGA GCG TTT T
dapF-r	GCT CGC GCT TAA CCA CTT G
lysA-f	CGA TCT CAC CGC CGA AAA
lysA-r	ACA CCG GGC AGC CAA A
rrsH-f	GTC GTC AGC TCG TGT TGT GAA
rrsH-r	CAC TGG CAG TCT CCT TTG AGT TC
gapA-f	AAA GGC GCT AAC TTC GAC AAA T
gapA-r	GCA GTT GGT GGT GCA GGA A
gfp-f	TCG ACA CAA TCT GCC CTT TTG
gfp-r	TCA TCC ATG CCA TGT GTA ATC C

Table 2.23: Recipe of M9A medium

Components	Company	Final conc.	Molecular weight	Weight
Amino Acid Sol. 10×		(mM)		(mg/500 mL)
L-α-Alanine	Nacalai	0.47	89.09	209.36
L-Arginine-HCl	Nacalai	0.60	210.66	631.98
L-Asparagine-H_2O	Nacalai	0.32	150.13	240.21
L-Aspartate	Nacalai	0.30	133.10	199.65
L-Cysteine-HCl-H_2O	Nacalai	0.30	175.64	263.46
L-Glutamine	Wako	5.00	146.15	3653.75
L-Glutamate-Na-H_2O	Wako	5.00	187.13	4678.25
Glysine	Wako	0.13	75.07	48.80
L-Histidine-HCl-H_2O	Nacalai	0.10	209.63	104.82
L-Isoleucine	Nacalai	0.30	131.17	196.76
L-Leucine	Wako	0.30	131.18	196.77
L-Methionine	Wako	0.30	149.21	223.82
L-Phenylalanine	Nacalai	0.30	165.19	247.79
L-Proline	Nacalai	2.00	115.13	1151.30
L-Serine	Wako	4.00	105.09	2101.80
L-Threonine	Wako	0.30	119.12	178.68
L-Trytophan	Wako	0.10	204.23	102.12
L-Tyrosine	Wako	0.10	181.19	90.60
L-Valine	Wako	0.30	117.15	175.73
Lysine Sol. 1000×				(g/50 mL)
L-Lysine-HCl	Nacalai	0.30	182.65	2.7398
M9 Salt Sol. 10×		(g/L)		(g/100 mL)
Na_2HPO_4-$12H_2O$	Nakarai	17.1	358.14	17.1
KH_2PO_4	Wako	3	136.09	3
NaCl	Nakarai	0.5	58.44	0.5
NH_4Cl	Wako	1	53.49	1
Metal Mix Sol. 100×				(g/100 mL)
$MgSO_4$-$7H_2O$	Wako	2mM	246.48	4.92
$CaCl_2$-$2H_2O$	Nacalai	0.1mM	147.01	0.147
Thiamine-HCl	Wako	10mg/L	337.27	0.1
Carbon Source 50×				(g/50 mL)
Glucose	Wako	5g/L(0.5%)	180.16	12.5
Fe^{2+} Sol. 1000×				(g/10 mL)
$FeSO_4$-$7H_2O$	Wako	10μM	278.02	0.278



Distribution of LOG GFP Peak position in different culture time

Figure 2.4: The self-fluorescent intensity of *E.coli* DH1 strain without any plasmid checked by flow cytometer

2.3 Results and discussion

2.3.1 Data acquisition and analysis

Nine reporter strains were constructed by cloning the promoter region of genes involved in lysine biosynthesis upstream of *gfp*. Before analyzing GFP expression by flow cytometry, the specific growth rates of these reporter strains were compared and confirmed to be similar (1.2±0.1/h).

Firstly, the background of flow cytometry was evaluated by *E. coli* DH1, which do not contain any plasmid. The LOG GFP value of *E. coli* DH1 is maintained at 229±12, which can be seen as background for the flow cytometry in this study.

We first studied the dynamics of the promoter activity of genes involved in the lysine biosynthesis pathway after replacing the defined medium without L-lysine with that supplemented with 0.3 mM L-lysine and vice versa. We measured the gene expressions every 15min for 3 h from the start of the medium change. Among the nine genes, the expression levels of five genes (*lysC*p, *asd*p, *dapB*p, *dapD*p, and *lysA*p) changed with the addition or removal of L-lysine from the medium (Figure 2.5). To determine how the

responses depend on the L-lysine concentration in the medium, we changed the final L-lysine concentration from 10^{-5} to 3 mM and kept all the other conditions constant. As results, we found that when the L-lysine concentration is larger than 0.3 mM, there was no difference in the results of flow cytometry analysis(Figure 2.6). Thus, we adopt two environmental conditions, *i.e.*, 0.3 mM and 0 mM L-lysine concentrations to investigate the change of expression dynamics in lysine biosynthesis pathway. We then focused on the detailed activation dynamics of genes involved in the lysine biosynthesis by changing the sampling interval from 15 to 6 min. The dynamic behavior of promoter activation was well visualized and quantitatively analyzed by flow cytometry (Figure 2.7).

2.3.2 Data fitting

We assume that each cell is well mixed system. There are sufficiently many molecules that the number of molecules can be approximated as continuously varying quantity that varies deterministically over time. And the process is fast compared with the time scale of interest. Because all the promoters have same regulation sign, lysine concentrations, we fit the time-dependent expression data to a simple Michaelis-Menten-type model developed by Ronen *et. al.*,(Rosenfeld et al. 2005) estimate the parameters of regulatory dynamics.

$$\frac{d\langle p_i \rangle}{dt} = \beta \frac{1}{1 + R(t)/k_i} - \alpha \langle p_i \rangle \tag{2.2}$$

The GFP protein concentration $\langle p_i \rangle$, which was regulated by i^{th} promoter, was balanced in terms of its expression rate and dilution by cell growth, where α represents cell growth rate. We assumed that the GFP lifetime was much longer than the cell cycle thus, we neglected the degradation of GFP protein. We adopted the peak value of the fluorescence distribution as $\langle p_i \rangle$, because the fluorescence distribution is asymmetric such that the average and median fluorescence distributions are inadequate to describe the expression dynamics. The parameters i and k_i are the maximal promoter activity and

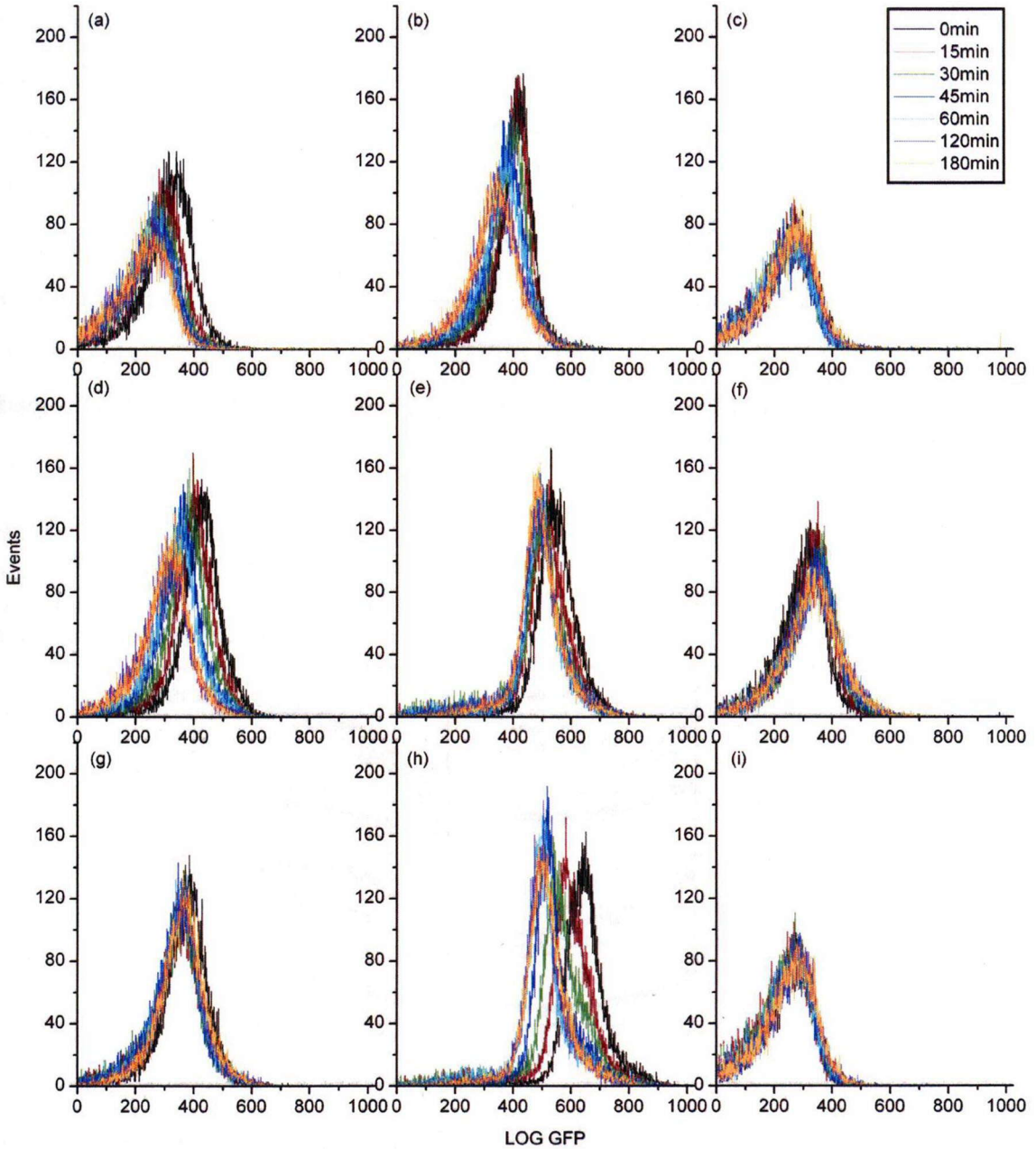


Figure 2.5: Raw gene expression data at 15min intervals when L-lysine concentration was changed from 0 to 0.3mM. (a) *lys*Cp,(b) *asd*p, (c) *dapA*p, (d) *dapB*p, (e) *dapD*p, (f) *dap*Cp, (g) *dapF*p, (h) *lysA*p and (i) *lysR*p

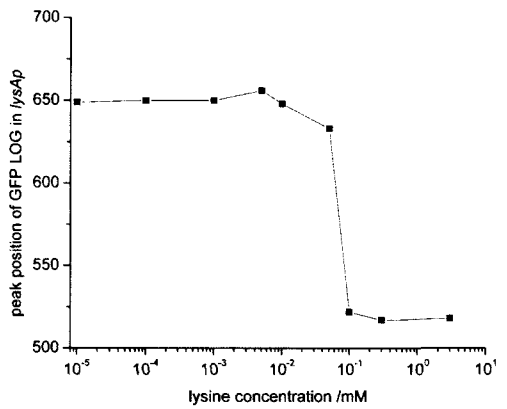


Figure 2.6: Peak position of GFP fluorescence in the reporter strain *lysA*p cultured in the medium with different L-lysine concentration

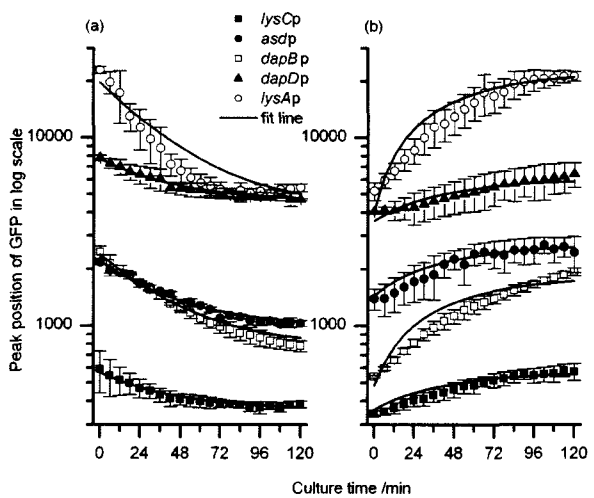


Figure 2.7: Promoter activation of genes that dynamically change their expression levels according to the change in L-lysine concentration as determined by plotting the peak position of GFP in log scale as a function of culture time. Error bars represent the standard deviations of three independent experiments. The L-lysine concentration in the medium was changed from (a) 0 to 0.3 mM and (b) 0.3 to 0 mM. The lines represent fitting results of promoter activation according to a simple dynamical model. The model parameters are shown in Table 2.24

the repression coefficient of the i^{th} promoter, respectively. The repression coefficient k_i represents the repressor concentration needed for 50% repression, which provides information on the strength of RNA polymerase binding and complex combinations of the binding affinity of the repressor to its cis-regulatory site. The $R(t)$ represents the active repressor level mediated by the intracellular L-lysine concentration. We assumed that the change in $R(t)$ induced by changing the environmental condition was faster than the change in gene expression level thus, we set $R(t)$ as a constant with respect to time and considered environmental condition as the only variable. Using the Equation (2.2), we fit the time series expression data to determine the parameters in the equations. In this fitting process, α, $R(t)$, i, and k_i were used as fitting parameters. The fitting results are shown in Figure 2.7 and the parameters used in the fitting are listed in Table 2.24.

Table 2.24: Optimal parameters of gene expression model. The parameter values of α and $R_{lys=0.3}/R_{lys=0}$ are shared in expression dynamics of all genes.
[a] $R_{lys=0.3}$ is the active repressor level when the L-lysine concentration is 0.3mM, while $R_{lys=0}$ is that in the L-lysine concentration is 0mM.

gene	α/min^{-1}	$\beta/\text{mM min}^{-1}$	k_i/mM	$R_{\text{lys=0.3}}/R^a_{\text{lys=0}}$
lysC		14.4	1.60	
asd		68.2	0.55	
dapB	0.025	46.3	0.69	0.95/0.001
dapD		152.2	2.98	
lysA		570.1	0.22	

From the fitting parameters, we found that *dapD*, one of the genes involved in the lysine biosynthesis pathway, showed a significantly larger repression coefficient (k_i) than the other genes involved in the pathway. Our experimental results show that although the absolute expression level of *dapD* is relatively high, the change of the expression between environmental conditions is small. Thus, the regulation driven by L-lysine concentration should be "weaker" than other regulated genes, as represented larger k_i value for *dapD*, which indicated the weak binding activity of the repressor to the *dapD* promoter region. The repression coefficient of *lysC* was also large. However, this result

of GFP expression controlled by the *lysC* promoter region was too close to the background level as determined by flow cytometry, and the results of RT-PCR showed a large ratio of relative expression level in the absence of L-lysine to that in the presence of L-lysine. Thus, we need to carry out more analyses before drawing any conclusion concerning the *lysC* promoter. We also found a trend that the closer an enzyme is to the start of the lysine biosynthesis pathway (Asp), the smaller its maximal promoter activity (β_i) is. This trend may suggest a design principle of the biosynthesis pathway(Zaslaver et al. 2004).

As shown in Figure 2.7, there is some deviation from fitted line and experimental data. Of course, we can adopt more complicated models with many parameters to fit our experimental data well. However, to use such complicated models makes the difference of characteristics among gene regulations obscure. The merit to use the simple kinetic model is that we can easily compare the characteristics of regulations. Although to discuss these characteristics quantitatively is rather difficult, it is possible to identify difference among promoters, as *dapD* in our study. Note that, our result that the regulatory parameters of dapD are different from other genes in lysine biosynthesis is robust with respect to changing the model to be fitted. For example, the same result can be obtained when we introduce higher Hill coefficient to the kinetic model.

This promoter dynamics include not only the steady states but also transient states. In experiments, the reporter strains were reached steady-state in preculture, and then were moved into a environment with different lysine concentration. Before they reached next steady-state in GFP expression, the changes in gene expressions are determined by the time series data in transient state. All these data in transient and steady states were used in fitting the parameters. Thus the model with parameters used in this study can predict the promoter activation not only in steady state but also in transient state. It provides biologists with tools to better understand and describe processes of lysine biosynthesis.

2.3.3 Real-time RT-PCR and Western Blotting

To validate the expression dynamics obtained by flow cytometry, we performed real-time RT-PCR and western blotting experiments. We extracted mRNA and protein samples from reporter and wild-type strains of *E.coli* cultured under different environmental conditions (*i.e.*, 0 or 0.3 mM L-lysine). Results showed that the relative changes in the gene expression levels obtained by real-time RT-PCR and western blotting experiments (Figure 2.8) are consistent with those determined by flow cytometry (Figure 2.9). We confirmed that the mRNA and protein expression levels of *lysC, asd, dapB, dapD,* and *lysA* increased with the removal of L-lysine from the medium, while those of the other 4 genes were unchanged. Of course, there are some differences in the ratio of expressions measured by flow cytometry compared to those obtained by RT-PCR and western blotting. One reason for the difference is experimental error. Furthermore, as for the difference between flow cytometry and RT-PCR analysis, it is well known that the correlation between expression levels of mRNA and proteins is not always proportional, instead, there is some deviation from proportionality due to difference in transcript efficiency. For example, in the previous report(Ghaemmaghami et al. 2003), it was shown that the amount of mRNA molecules and protein molecules per cell is well correlated, but the significant deviation from the linear relationship exists. In comparison with the previous studies, the correlations between results of flow cytometry, RT-PCR and western blotting analysis shown in our manuscript are acceptable. Our Real-time RT-PCR and western blotting results also showed that, for each reporter strain, changes in the expression level of GFP and the endogenous protein under the control of the same promoter correlated well indicating that our selection of promoter regions was adequate for investigating gene expression dynamics.

In the studies of gene expression dynamics, the combination of reporter strains and flow cytometry provided us with a convenient and accurate method of measuring gene

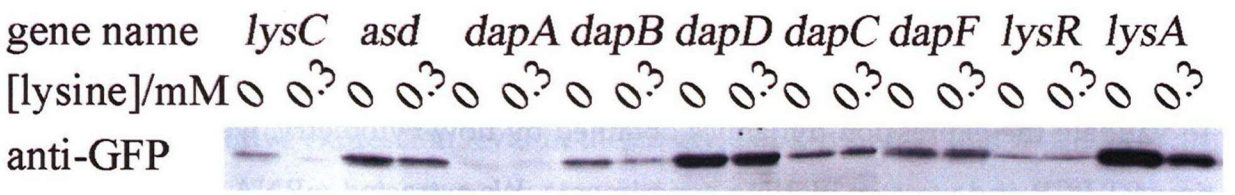


Figure 2.8: Western blotting analysis of the expression level of GFP in different strains cultured in different L-lysine concentration (0 or 0.3 mM)

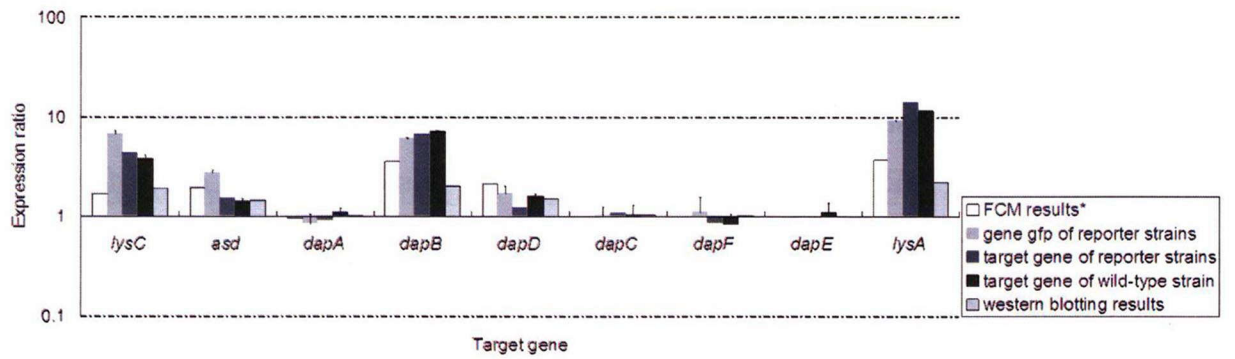


Figure 2.9: Ratio of gene expression level in the absence of L-lysine to that in the presence of L-lysine calculated using the reference gene *gapA* (results were similar to those using the reference gene rrsH; data not shown).
* calculation method: GFP expression levels of cells cultured in the medium without L-lysine divided by those of cells cultured in the medium with L-lysine as determined by flow cytometry (FCM).

expression dynamics. The reporter strain, in which fluorescent protein expression is controlled by the promoter of the target gene, is relatively easy to construct. Moreover, we found that the measurements of expression dynamics using these reporter strains were consistent with those obtained by real-time RT-PCR analysis and western blotting analysis.

One important merit of using this system is that measurements using a small number of cells (*e.g.*, 20,000 cells) are possible. In contrast, it is difficult to analyze samples obtained from such a small cell number by real-time RT-PCR analysis or using microplate reader.

Another important merit of using flow cytometry is the possibility of single-cell-level analysis, which provides information on stochastic fluctuations in gene expres-

sion dynamics. Recently, stochastic fluctuation and its propagation in cellular reaction dynamics have been studied extensively(Raser and O'Shea 2004). In these studies, artificially synthesized networks are generally used to investigate stochastic fluctuations in gene expression. In contrast, gene expression fluctuations and the propagation of these fluctuations in native regulatory and metabolic networks can be analyzed using our reporter strains. The results of the stochastic fluctuation analyses using our reporter strains will be reported elsewhere.

2.3.4 Noise Properties

By flow cytometry, not only the average expression can be determined, but also the distribution of the expression in the population can be described. By treating the distribution of expression as normal distribution, we got the phenotypic noise strength(PNS) of each promoter by Equation (1.2). There is no relationship between fluctuations and estimated parameters of expression dynamics, but negative correlation between the biological noise and their average expression instead (Figure 2.10). We failed in including noise information into the parameters.

2.4 Conclusions

Nine reporter strains were constructed by cloning the promoter region of genes involved in lysine biosynthesis upstream of *gfp*. Time-dependent changes in green fluorescence intensity under the control of the target promoter region were determined after changing the environmental condition (*i.e.*, the L-lysine concentration in the medium). By flow cytometry, the gene expression dynamics were quantitatively analyzed. The differences in the expression levels of genes involved in the lysine biosynthesis at various L-lysine concentrations were confirmed by real-time RT-PCR analysis and western blotting analysis. The expressions of five genes corresponded to the changes in L-lysine concentration in the medium. For the promoters of these five genes, time-dependent ex-

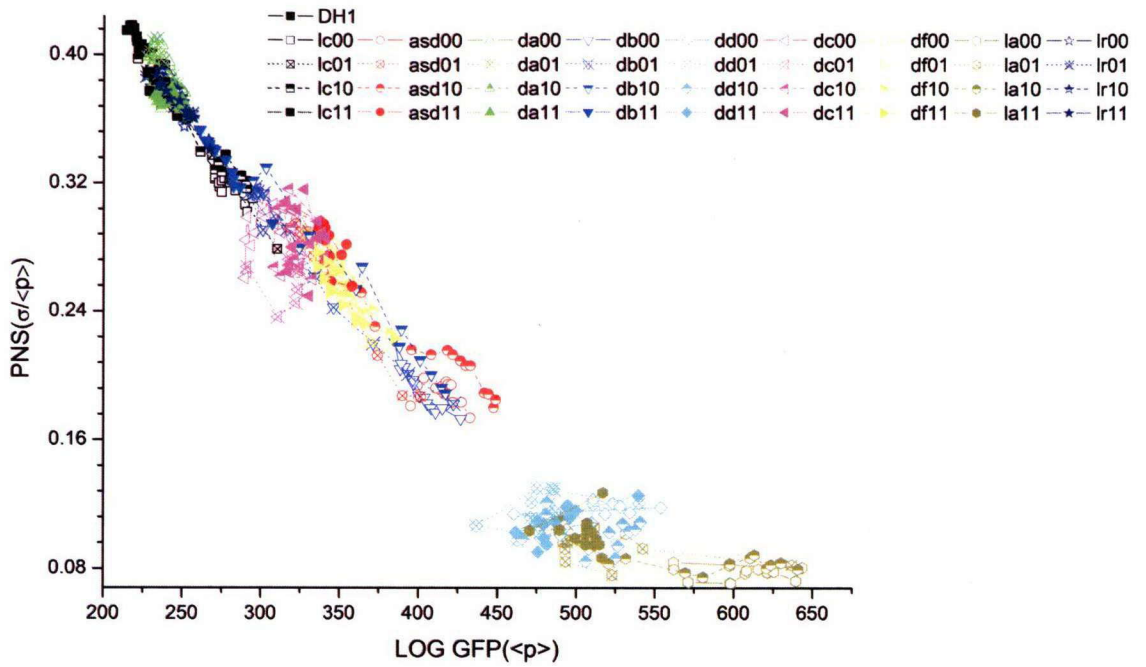


Figure 2.10: Noise properties when plot phenotypic noise strength to their average expression levels. DH1: *E.coli* DH1; lc: *lysC*p; asd: *asd*p; da: *dapA*p; db: *dapB*p; dd: *dapD*p; dc: *dapC*p; df: *dapF*p; la: *lysA*p; 00, 11: L-lysine concentration maintained at 0mM or 0.3mM; 01: L-lysine concentration shift from 0mM to 0.3mM; 10: L-lysine concentration shift from 0.3mM to 0mM.

pression data monitored by *gfp* fluorescence were fitted to a simple dynamical model of gene expression. The fitting parameters enabled an elucidation of the gene expression dynamics in the lysine biosynthesis pathway.

Published as: Jianhong Ou, *et.al.* – *"Analysis of Stochasticity in Promoter Activation by Using A Dual-Fluorescence Reporter System"* BioSystems in press

Chapter 3

Analysis of stochasticity in promoter activation by dual-fluorescence system

Abstract

Stochastic dynamics of promoter activity in bacterial cells were studied by using a dual-fluorescence reporter system of protein expression. The dual-fluorescence reporter system enabled us to calculate the amplitude of intrinsic noise generated during transcription and translation. By fitting the experimental data to a simple stochastic model of protein expression, we could estimate parameters representing the stochastic transition between the active and inactive states of a promoter. Using the system, we analyzed the stochastic dynamics of promoter activation of genes in the lysine biosynthesis pathway in *Escherichia coli*. We found that the promoter of *lysA* has a significantly slower transition rate between active and inactive states than other promoters in the lysine biosynthesis pathway. The infrequent switching between active and inactive states can be a dominant source of noise in *lysA* expression. Analysis using the dual-fluorescence reporter system provided a better understanding of stochastic dynamics in promoter activation.

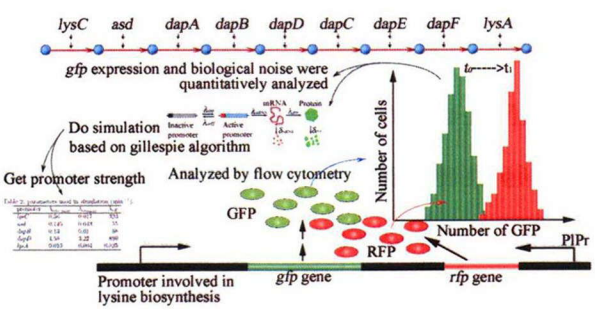


Figure 3.1: Dual-fluorescence system includes two parts, the vector pGRFP with two reporter fluorescence gene and the simulation tool used for getting appropriate λ_{on} and λ_{off} for the target promoter.

3.1 Introduction

Single fluorescence experiments provide quantitative evaluation of dynamics of promoter activation in a continuous manner. However, recent studies have revealed that changes in promoter activity do not always occur in a continuous manner. Instead, promoter activity often obeys on-off type stochastic dynamics, regarded as a major source of gene expression noise.

3.1.1 Transcriptional bursting model

In the **transcriptional bursting model**, slow promoter kinetics cause infrequent transitions between active and inactive promoter states, which, in turn, cause multiple mRNA templates to be synthesized in rapid succession at irregular intervals, if it followed with a high transcription rate(Raser and O'Shea 2004, Ozbudak et al. 2002, Blake et al. 2006). At any instant the promoter is thought to be either "switched on" by having a transcription complex bound to it, or "switched off" by not having a transcription complex bound (Brunner and Bujard 1987, Ko 1991, Kepler and Elston 2001, Pirone and Elston 2004, Lipniacki et al. 2006). There are two important parameters in these on-off stochastic dynamics: activation rate and inactivation rate; these represent the probability that an inactivated promoter will be activated and vice versa, respectively(Figure 3.2). If these two parameters are low enough and the time scale of their alternation is slower than the time needed for transcription and translation, the switching of promoter activity can be a dominant source of expression noise. The equilibrium promoter strength, which is determined by the gene expression mainly, can also be calculated from the ratio of λ_{on} and λ_{off} in the steady state(t) and shown as K_{eq}:$K_{eq} = \frac{\lambda_{on}(t)}{\lambda_{off}(t)}$ (Brunner and Bujard 1987, McClure 1980). Pairs of λ_{on} and λ_{off} can reach the same K_{eq}. For example $K_{eq} = 1$ can be gotten from any pair of λ_{on} and λ_{off} when $\lambda_{on} = \lambda_{off}$. That is to say, the same promoter strength with a different λ_{on} and λ_{off} can lead same expression

but different cell-cell variability. Only with K_{eq}, it can not include the information about the biological noise. Thus, to achieve a detailed understanding of stochastic dynamics of promoter activity, it is necessary to evaluate the parameters of switching activity.

But most of prokaryotic gene expression assume that the transition rates are so fast that the promoter states are always in steady state and the rate of transcription is constant. And the translational bursting is probably a dominant source of stochasticity in the process of prokaryotic gene expression(Kærn et al. 2005). This was mainly supported by the Ozbudak's report. They used single copy of *gfp* as reporter to monitor the effect of transcriptional and translational efficiency on biological noise(Ozbudak et al. 2002). That is the most possible reason why we fail in including noise information into the parameters by single fluorescence reporter.

3.1.2 Intrinsic noise and extrinsic noise

As reported by Elowitz et al. (2002), biological noise has two sources, intrinsic and extrinsic; promoter activity mainly contributes to intrinsic noise (Shahrezaei et al. 2008). The inherent stochasticity of biochemical processes such as transcription and translation generates "**intrinsic" noise**, denoted η_{int}, which fundamentally limits the precision of gene regulation. Such stochastic effects are set locally by the gene sequence and the properties of the protein it encodes. Living cells possess very low copy numbers of many components, including DNA, mRNA templates and regulatory molecules. The small numbers of gene copy and mRNA template results in discrete biochemical reactions involved in gene expression. Such discrete events can be explained by probability theory. The probability of each reactions in a defined interval is determined by the affinity of molecules such as regulatory proteins and polymerases, and so on, binding to the active sites, which is affected by the space structure and regulation module of the gene sequence or the protein it encodes. For example, there are many functional modules in promoter, such as enhancer, booster, activator, insulator, repressor, locus control region,

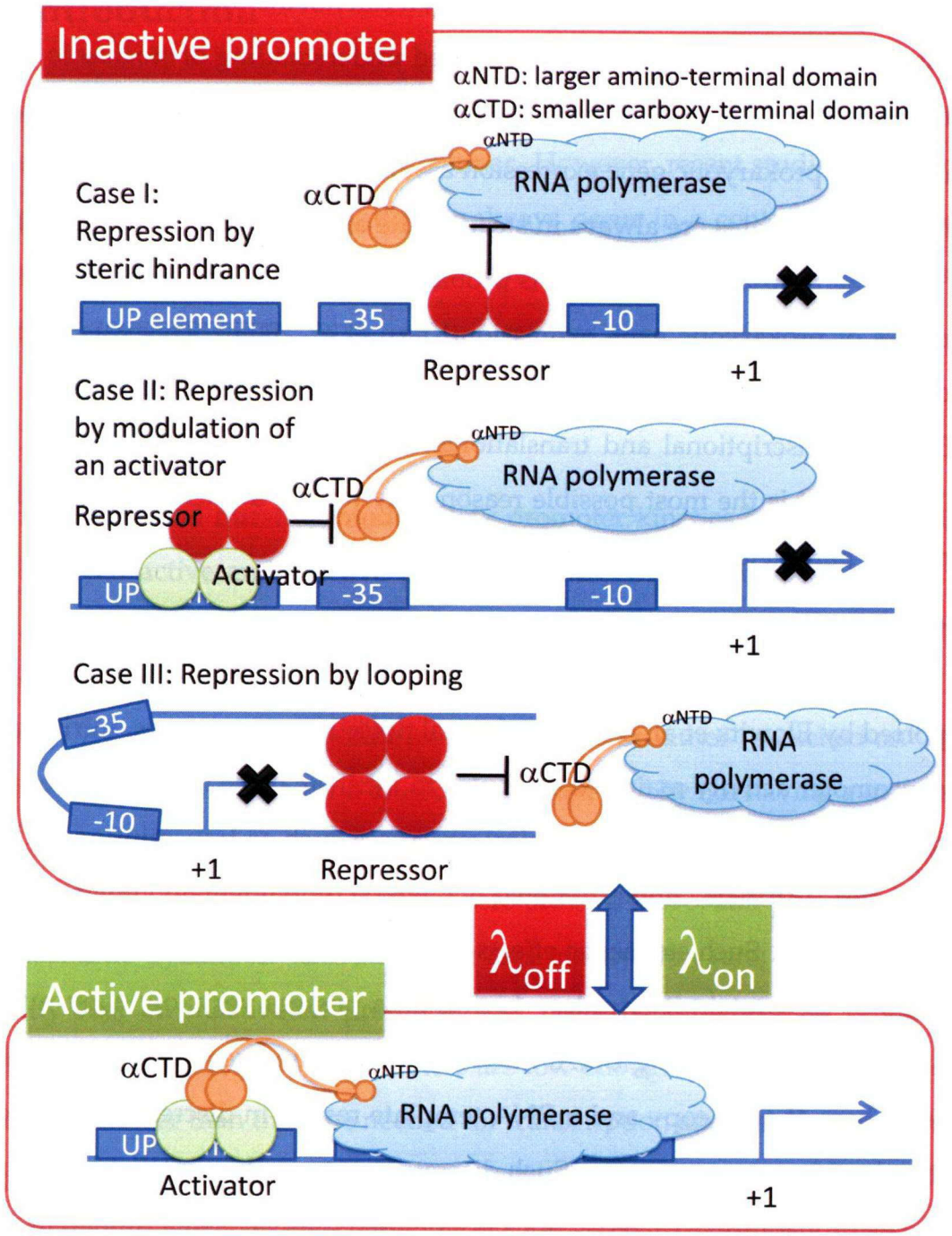


Figure 3.2: The two state of promoter. If the promoter is in active state, the transcription can start. Otherwise, it can not. There list three cases that can not start transcription. Case I, repression by steric hindrance. The repressor-binding site overlaps core promoter elements and blocks recognition of the promoter by the RNA polymerase holoenzyme. Case II, repression by modulation of an activator protein. The repressor binds to an activator and prevents the activator from functioning by blocking promoter recognition by the RNA polymerase holoenzyme. Case III, repression by looping. Repressors bind to sites and interact by looping, repressing the intervening promoter.(Browning and Busby 2004a)

upstream activating sequence and upstream repressing sequence, that contribute to the initiation of transcription. Moreover, the interaction of chromatin remodeling complex, transcription co-factors, transcription factors and chromosome can shift the states of promoter between open and close in a given probability determined by the space structure and promoter sequence. In addition, fluctuations in the amounts or states of other cellular components lead indirectly to variation in the expression of a particular gene and thus represent "**extrinsic" noise**, denoted η_{ext}. Thus, extrinsic sources of noise arise independently of the gene but act on it. Such stochastic effects are controlled by the concentrations, states, and locations of molecules such as regulatory proteins and polymerases, and so on. Although for different genes there are different regulators who will take effect on them, the extrinsic noise is defined as the environment all the genes faced. And the difference in regulation mechanism of different genes will generate the intrinsic noise. So the extrinsic noise is global to a single cell but vary from one cell to another. The extrinsic fluctuations can dominate the total noise and they are sensitive to fluctuations in the transcription and translation rate(Shahrezaei et al. 2008). To distinguish between the two noise sources, it is not sufficient to monitor the expression of a single gene using fluorescent proteins. Instead, it is necessary to monitor the expression levels of two genes at a single-cellular level, as demonstrated by Elowitz et al. (2002). If we failed in separating the intrinsic noise from the extrinsic noise, the reliability of assumption that all the prokaryotic promoters have very high λ_{on} and λ_{off} faces the challenge.

Intrinsic noise and extrinsic noise of promoter can be measured and distinguished with two report genes controlled by same copy of it(Elowitz et al. 2002). By analyzing the correlation in the expression levels of two genes, we can obtain the amplitude of extrinsic noise, and then, by subtracting the extrinsic noise from the total noise, we can estimate the amplitude of intrinsic noise. With the data regarding intrinsic noise and an appropriate model, the parameters representing infrequent stochastic transition

between active and inactive promoter states can be estimated. This method was widely used in biological noise analysis(Mettetal et al. 2006, Pedraza and van Oudenaarden 2005).

However, most of the experimental studies have so far focused on noise property itself. To apply this method to analyze the strength of a promoter involved in an endogenous gene networks, particularly in metabolism network, is seldom reported. In the transcriptional bursting model, the size of the 'burst' in transcription depends on the average number of transcripts produced between promoter activation and deactivation (the ratio $\lambda_{\mathrm{mRNA}}/\lambda_{\mathrm{off}}$), referred to as the transcriptional efficiency(Raser and O'Shea 2004, Kærn et al. 2005). To master the promoter transition rates is definitely helpful in predicting gene expression with noise property. Naturally, the gene expression noise might necessitate counteracting noise reduction mechanisms preserving the fidelity of regulatory signals. On the other hand, the probabilistic features afforded by gene expression noise lead to the evident possibility that evolution has fine-tuned noise-generating mechanisms and genetic network architectures to derive beneficial population diversity(Fraser and Kærn 2009). To get the appropriate λ_{on} and λ_{off} can help us understand the promoter strength at a deep level and be able to make predictions about promoter behavior with biological noise property.

3.1.3 Stochastic Simulation Algorithm(SSA)

In order to get the λ_{on} and λ_{off} for target promoter, a simulation tool is necessary.There are two fundamental ways to view coupled systems of chemical equations: as continuous, represented by differential equations are concentrations, or as discrete, represented by stochastic processes whose variables are numbers of molecules.

Solving the differential equations results in concentration of each substance as a function of time. There is an assumption for the differential equations is that the number of molecules can be approximated as a continuously varying quantity that varies

deterministically over time. Although this assumption holds for most systems, it does not hold in very small systems. And solving these differential equations, sometimes, one assumes that the system is in equilibrium. It is not true in general.

The stochastic processes regards the time evolution as a kind of random-walk process which is governed by a single differential-difference equation. The SSA allows one to numerically simulate the transient behavior of well-mixed systems in which many molecular species participate in many highly coupled chemical reactions(Gillespie 1977). The SSA is exact in the sense that it is rigorously based on the same microphysical premise that underlies the chemical master equation(CME)(Gillespie 1992). The SSA is widely used in biological noise analysis(Shahrezaei and Swain 2008).

In SSA, a putative time for each potential reaction in the system is calculated, and the reaction whose putative time is first is implemented. Simulation time is then incremented by this reaction time. Each putative reaction time is calculated from the propensity of the reaction: the probability of the reaction per unit time multiplied by all ways of selecting the reactants. If a fixed volume V contains a spatially uniform mixture of N chemical species which can inter-react through M specified chemical reaction channels,

$h_\mu \equiv$number of distinct R_μ molecular reactant combinations available in the state $(X_1, X_2, ..., X_N)$ (μ=1,...,M)

$a_\mu dt \equiv h_\mu c_\mu dt$=probability that an R_μ reaction will occur in $(t, t + dt)$, given that the system is in the state $(X_1, X_2, ..., X_N)$ at time t (μ=1,...,M)

The probability density function for reactions is,

$$P(\tau, \mu) = a_\mu \exp\left(-\tau \sum_{v=1}^{M} a_v\right) \tag{3.1}$$

where, τ is a uniform random number between 0 and 1, which will introduce the stochasticity into the simulation. By getting the μ and P, the questions which reaction occurs next and when does it occur are resolved.

Gillespie developed two different, but equivalent formulations; the direct method and the first reaction method(Gillespie 1977). Since it was born, there has been countless attempts to improve its computational efficiency. But the main steps to run the algorithm are not change so much. They are:

1. **Initialization**: Initialize the number of molecules in the system, reactions constants, time and random number generators.

2. **Monte Carlo step**: Generate random numbers to determine the when and which reaction to occur next.

3. **Update**: Increase the time step by the randomly generated time in Step 2. Update the molecule number based on the reaction that occurred.

4. **Iterate**: Go back to Step 2 unless the number of reactants is zero or the simulation time has been exceeded.

3.1.4 Object

In this study, we developed a dual-fluorescence system to understand the stochastic dynamics of promoter activity. This system includes two parts; the vector pGRFP, containing two fluorescent proteins, GFP and red fluorescent protein (RFP), for measuring intrinsic noise, and a simulation tool to estimate the parameters describing stochastic dynamics. In this system, the expression of GFP is controlled by the target promoter to be inspected, while the expression of RFP is controlled by the common constitutive promoter. The amplitude of extrinsic noise is obtained by the correlation between the intensities of GFP and RFP fluorescence. Thereafter, the intrinsic noise of the target promoter can be obtained by subtracting the extrinsic noise from the total noise of GFP expression. By adjusting for the parameters in the model to not only the expression level but also the intrinsic noise amplitude, the activation and inactivation rates of promoter activity can be estimated. Using this system, we investigated the characteristics

of promoters involved in lysine biosynthesis pathways in *E. coli*. We found that the activation and inactivation rates of the *lysA* promoter are significantly lower than those of other promoters involved in the lysine biosynthesis pathway, which indicates that infrequent switching can be a dominant source of noise in *lysA* expression. Analysis using the dual-fluorescence reporter system provided a better understanding of stochastic dynamics in promoter activation.

3.2 Material and Methods

3.2.1 Plasmid

The plasmid pGRFP(Figure 3.3) contains genes encoding two fluorescent proteins, GFP and RFP. The *gfpuv5* gene, a variant of *gfp*, was obtained from the plasmid pPROLar-GFPuv5(Ito et al. 2004), and *rfp*-T4 (DsRed) was obtained from the plasmid pQE31-T4 (Bevis and Glick 2002). A strong constitutive promoter P_LP_R was obtained from the plasmid pCL476 (Love et al. 1996). The kanamycin gene and the p15A origin of replication were obtained from pPROLar (Ito et al. 2004). The promoter was inserted upstream of *gfpuv5* in the *Apa*I-*Nhe*I site.

Construction of pGRFP

The whole strategy is shown as Figure 3.4. The primers used in these construction is shown as Table 3.1 and the primers used for sequencing is shown as Table 3.2.

The DsRed and stop codon were amplified by PCR with primer T4-f and T4-r. The amplification mixture is shown as Table 3.3. The amplification was performed in a DNA thermal cycler using a program set to denature at 94°C for 5min, followed by 30 cycles of 94°C for 10s, 60°C for 30s and 72°C for 1min, and a extension step of 72°C for 7min. The PCR product was recovered, digested by *Nde*I and *Nhe*I (Table 3.4) and then insert into the MCS region of pCL476 vector by *Nde*I-*Xba*I(Table 3.5) sites. The construction of pCL476 was checked by sequencing.

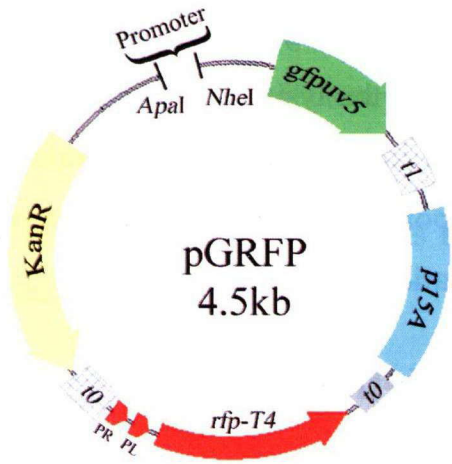


Figure 3.3: The dual-fluorescence reporter system. The plasmid pGRFP contains two genes that code for fluorescent proteins, GFP and RFP. The promoter region under investigation is inserted upstream of GFP, while the expression of RFP is controlled by the promoter $P_L P_R$.

Table 3.1: Oligonucleotides used for pGRFP construction

Primer Name	Primer sequence (5′→3′)
T4-f	GGAATTCcatatgGCCTCCTCCG
T4-r	CGgctagcTTGGATTCTCACC
Pl-T4-f	GCGcctaggCGGTGTTGACATAAATAC
Pl-T4-r	GCGcctaggacgtcTAGCTTGGATTCT
PlPr-T4-f	GCGcctaggTAACACCGTGCGTG
pSC101*-f	GCATGCaagcttGGCGTAATCATGGTCATAG
pSC101* -r	TGATAATTactagtCCTTTTcccgggagatctGGGTATCTG
par-pSC101*-f	TCCCCGCGGACAGTAAGA
par-pSC101*-r	CCTATTAATCATCTGTGCATATGGACA

Table 3.2: Oligonucleotides used for sequencing

Primer Name	Primer sequence (5′→3′)
grfp-101-f	GCTCTCCTGAGTAGGACA
grfp-101-r	GCTGACTTCAGGTGCTA
grfp-1441	CAGCTTTGAATGCACCA
grfp-1929	GCCTCGCTTATCAACCA
grfp-2928	GCTTGCGAGGGTGCTA
grfp-rfp-f	TGTAGCACCTGAAGTCA
grfp-rfp-r	GGAGGAGTCCTGGGTCA

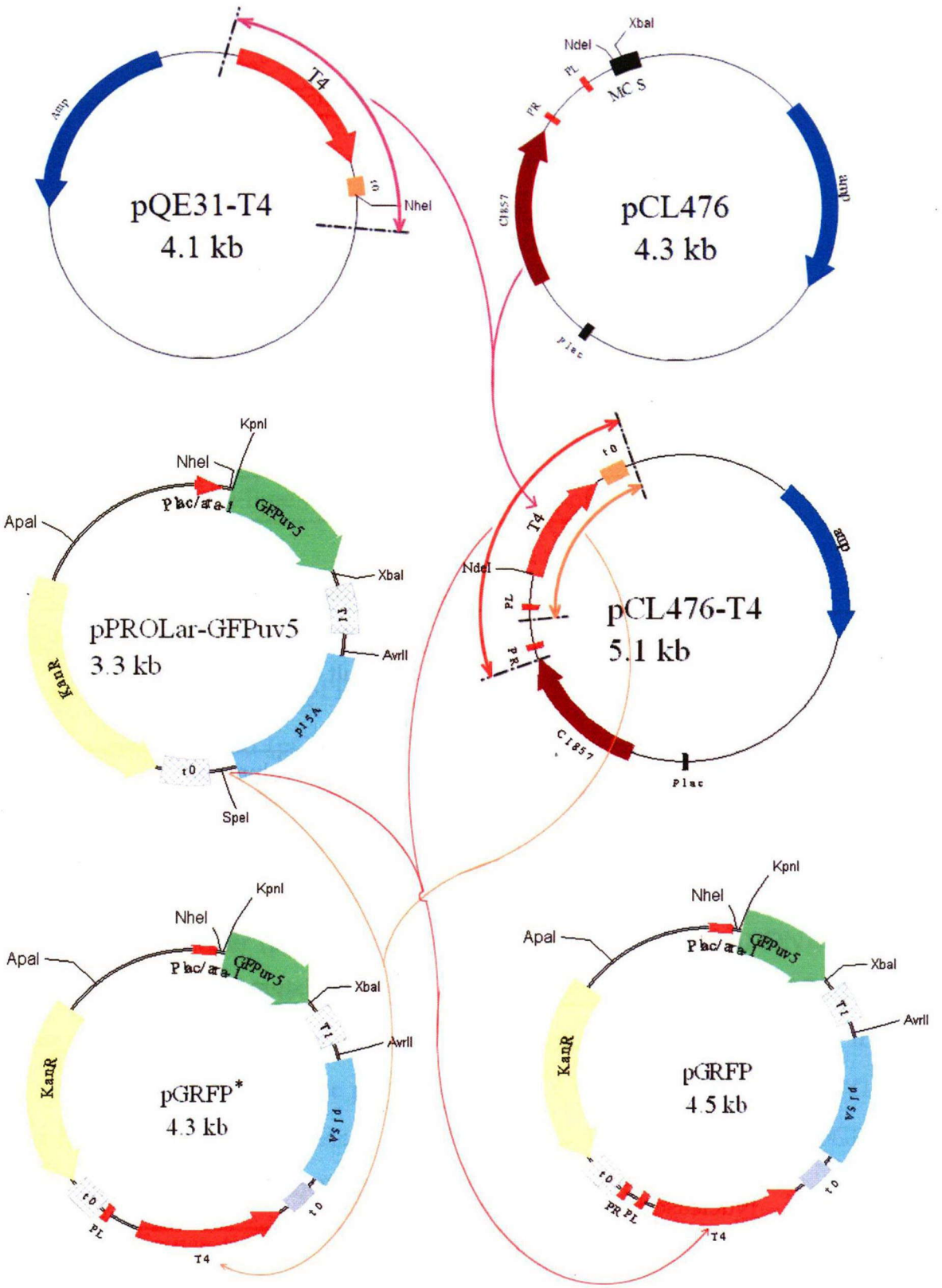


Figure 3.4: The strategy for pGRFP construction.

Table 3.3: Plasmid construction amplification mixture

Components	Amount(μL)
plasmid (10ng/μl)	5
forward primers(20pM)	5
reverse primers(20pM)	5
dNTP Mixture, 25mM	10
Ex taq 10X buffer	10
Takara Ex Taq(5 units/ul)	1
Nuclease-Free water to a final volume of	100

Table 3.4: Restriction enzyme digestion mixture for DsRed insertion

Components	Amount(μL)	Components	Amount(μL)
pQE-T4 PCR result	20	PCR result digested by *Nhe*I	38
Buffer M	5	Buffer H	5
*Nhe*I	1	*Nde*I	1
DW to a final volume of	50	DW to a final volume of	50

Table 3.5: Double restriction enzyme digestion mixture for pCL476 vector

Component	Amount (μL)
pCL476 plasmid (0.18μg/μL)	20
Buffer T	5
*Xba*I	1
*Nhe*I	1
Nuclease-Free water to a final volume of	50

Table 3.6: Restriction enzyme digestion mixture for Pl-*rfp*-t0 and PlPr-*rfp*-t0 insertion

Component	Amount (μL)
PCR amplicon	20
NEB 2	5
*Avr*II	1
Nuclease-Free water to a final volume of	50

Table 3.7: Restriction enzyme digestion mixture for pPROLar-GFPuv5 vector

Component	Amount (μL)
pPROLar-GFPuv5 plasmid	10
Buffer M	5
*Spe*I	1
Nuclease-Free water to a final volume of	50

After amplificating Pl-*rfp*-t0 and PlPr-*rfp*-t0 fragment by PCR using primer Pl-T4-f, PlPr-T4-f and Pl-T4-r(Table 3.3), the amplicon was recovered, digested by *Avr*II (Table 3.6) and then inserted into pPROLar-GFPuv5 by *Spe*I site (Table 3.7).

The low copy plasmid (pGRFP-pSC101*) was also constructed using par-pSC101* to replace the p15A. par-pSC101* was come from pMW119 with site-mutation. The primers(shown in Table 3.1) used for amplification of par-pSC101* was designed to amplify the segment of pMW119 DNA between the *Hind*III site of the MSC and the *Spe*I site. The forward primer (pSC101*-f) was complementary to pSC101 DNA origin except for five-base mismatch creating *Sma*I and *Bgl*II. This primer also contained the *Spe*I site. The original replicon was replaced by this amplicon by *Hind*III-*Spe*I site to make pMW119* with par-pSC101*. The par-pSC101* was gotten from plasmid pMW119* by enzyme digestion between the *Acc*II and *Xba*I site, and then inserted into pGRFP* by *Ase*I-*Avr*II site to construct pGRFP-pSC101*.

However, because the RFP signal of reporter strains which contain pGRFP* or pGRFP-pSC101* is not high enough from the background, only pGRFP was selected

for further research. And the promoter $P_{lac/ara-1}$ in pGRFP was replaced by P_RP_L from plasmid pCL476 to finish the construction.

3.2.2 Strains and Media

Nine reporter strains were used in this study, *E. coli* DH1 (K-12 *endA1 recA1 gyrA96 thi-1 glnV44 relA1 hsdR17(rK⁻mK⁺) λ⁻*)/p*lysC*p-pGRFP (*lysC*p in short), DH1/p*asd*p-pGRFP (*asd*p), DH1/p*dapA*p-pGRFP (*dapA*p), DH1/p*dapB*p-pGRFP (*dapB*p), DH1/p*dapD*p-pGRFP (*dapD*p), DH1/p*dapC*p-pGRFP (*dapC*p), DH1/p*dapE*p-pGRFP (*dapE*p), DH1/p*dapF*p-pGRFP (*dapF*p), and DH1/p*lysA*p-pGRFP (*lysA*p) which contain the promoter regions of genes (i.e. *lysC, asd, dapA, dapB, dapD, dapC, dapE, dapF,* and *lysA*), respectively. The promoter regions were amplified from *E. coli* DH1 genomic DNA by PCR. The primers listed in Table 2.22 were used to amplify the regions between two adjacent open reading frames (ORFs), with an extension of 150–200 bp upstream of the ORFs. The promoter regions were cloned at *Apa*I and *Nhe*I sites upstream of *gfpuv5*. Reporter strains were cultured in M9 minimal medium with an amino acid solution (M9A medium) and kanamycin (25 μg/mL) (Ford et al. 1994). It should be noted that the L-lysine in the amino acid solution was prepared separately, and the addition of L-lysine to the medium was determined according to experimental conditions. All the details please refer Charpter 2.

3.2.3 Sample preparation

The reporter strains were streaked on an M9A minimal medium plate (1.5%[w/v] agar) from the stocks and incubated at 37°C for 15 h. Pre-culture was performed by picking a colony and inoculating it into 2 test tubes containing 5 mL of fresh M9A medium (one test tube contained 0.3-mM lysine, while the other contained no lysine). The inoculated test tubes were then reciprocally shaken at 37°C and 160 rpm until the OD_{600} value reached 0.6–0.7. After the pre-culture, cells were collected by centrifugation at 6,000

rpm for 5 min, followed by re-suspension of the pellet in 1 mL of M9A medium (with or without lysine, depending on the next culture step). The suspensions were inoculated into test tubes containing 5 mL of M9A medium with or without lysine to make the initial OD_{600} = 0.01 for the main culture. The test tubes were again reciprocally shaken at 37°C and 160 rpm. Every 15 min, samples were prepared for flow cytometry by extracting 1 mL of each culture; samples were stored at –80°C until use.

3.2.4 Data acquisition and analysis

The samples were thawed before analysis with flow cytometry (COULTER® EPICS® XL™, Beckman Coulter, Fullerton, CA, USA) and then diluted with phosphate buffer solution (PBS, pH 7.0) to set the cell concentration at 10^7 cells/mL (OD_{600} = 0.01). Fluorescence measurements were conducted using flow cytometry, and 20,000 cells from each sample were analyzed. The excitation of GFP and RFP was achieved using a 488 nm argon excitation laser, and fluorescence was measured with a 525±20 nm emission filter for GFP and a 575±20 nm emission filter for RFP. The flow cytometry generated log-scale values using a 10-bit analog-to-digital converter, yielding integers in the range of 0 to 1,023 for each of four measurements: two kinds of fluorescent intensity, forward-scattering (FSC), and side-scattering (SSC).

The *lmd* files produced by the flow cytometry were analyzed by a program FCSMultiOu(AppendixB.2) developed for noise analysis by us. The program can handle hundreds of *lmd* files in the same time. In order to handle the list mode data, a class named as fcmdata was constructed. In "fcmdata" class, there are eleven public members. Member "fslog" is for FSC log-scale signal, "sslog" for SSC log-scale signal, "gfplog" for GFP log-scale signal, "rfplog" for RFP log-scale signal, and "flag" for data gating for each cell. Since the raw data include signals from non-living particles in the medium, we remove them using gates of FSC and SSC intensities. We determined the average FSC and SSC, and used these coordinates to define radial regions (gates, $r = 50$) in the 2D

scattering plots. These gates were then used to select the cells used in the GFP and RFP fluorescent intensity distributions. The gates were realized by "rFilter" method in "fcm" class. The log-scale data were converted into fluorescence intensity by "fluorescenceConvert" method in "fcm" class. The intrinsic noise was calculated by "g_intNoise" method in "fcm" class.

To obtain intrinsic noise amplitude, we adopted the following formula published in Elowitz et al. (2002):

$$\eta_{\text{ext}}^2 = \frac{\langle rg \rangle - \langle r \rangle \langle g \rangle}{\langle r \rangle \langle g \rangle}, \tag{3.2}$$

where, η_{ext} shows the amplitude of extrinsic noise; $\langle g \rangle$ and $\langle r \rangle$ represent the means of GFP and RFP intensity, respectively; and $\langle rg \rangle$ indicates the correlation between GFP and RFP over cells. The amplitude of total noise for GFP $\eta_{\text{g-tot}}$ is represented as follows:

$$\eta_{\text{g-tot}}^2 = \frac{\langle g^2 \rangle - \langle g \rangle^2}{\langle g \rangle^2}. \tag{3.3}$$

In our reporter system, GFP expression is controlled by the target promoter under investigation, while RFP expression is controlled by the constitutive promoter $\mathrm{P_L P_R}$. As these two genes are expected to share the same extrinsic noise value η_{ext} in one cell, we can distinguish between intrinsic and extrinsic noise by comparing the two. Thus, the intrinsic noise of the target promoter can be calculated by subtracting extrinsic noise from the total noise, as follows:

$$\eta_{\text{g-int}}^2 = \eta_{\text{g-tot}}^2 - \eta_{\text{ext}}^2, \tag{3.4}$$

where, $\eta_{\text{g-int}}$ represents the amplitude of intrinsic noise.

For the fluorescent measurements, the background signal (auto-fluorescence of cells) was subtracted by assuming that the background signal distribution follows a Gumbel distribution with appropriate parameters, which could represent the distribution of auto-fluorescence of cells without GFP and RFP. To confirm the reproducibility of the changes in the fluorescence distribution dynamics, we performed two experiments under the same environmental conditions and found that the measurement results were

robust in these independent experiments. In this paper, we show the results of just one of the experiments.

3.2.5 Stochastic Model

Based on Gibson' s Next Reaction Method algorithm (Gibson and Bruck 2000), which extends Gillespie' s First Reaction algorithm (Gillespie 1977), a program was built to simulate the intrinsic noise of GFP expression. This simulation allowed us to investigate the characteristics of stochastic active-inactive switching in the target promoter. Our algorithm to simulate intrinsic noise is as follows:

1. Initialize:
   (a) Set initial numbers of molecules, set $t \rightarrow 0$.
   (b) Generate a dependency graph $G_{\mathtt{GFP}}$ for GFP and $G_{\mathtt{RFP}}$ for RFP.
      i. Calculate the propensity function, a_i, for all i.
      ii. For each i, generate a putative time, t_i, according to an exponential distribution with parameter a_i.
      iii. Store the t_i values in the heap $P_{\mathtt{GFP}}$ and $P_{\mathtt{RFP}}$ for GFP and RFP, respectively.
2. Let $t_{\mathtt{shift}}$ be the time for change culture condition point. If $t = t_{\mathtt{shift}}$, change the promoter λ_{on}.
3. Compare the roots of heap $P_{\mathtt{GFP}}$ and $P_{\mathtt{RFP}}$. Let u be the reaction whose putative time, t_u, is smaller. Because the heap queue is an indexed queue sorted by putative time, t_u is the least putative time.
4. Let t be t_u.
5. Change the number of molecules to reflect execution of reaction u.
6. For each edge(u, w) in the dependency graph $G_{\mathtt{GFP}}$ or $G_{\mathtt{RFP}}$,

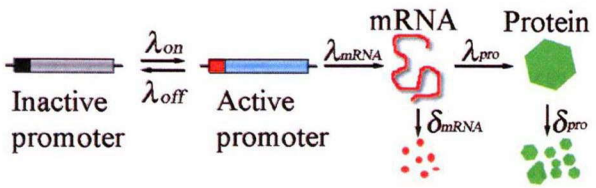


Figure 3.5: Gene expression model used in this study. Each step represents the biochemical reactions, associated with the transition between promoter states and the production and decay of mRNA and proteins. λ_{on} and λ_{off} indicate the promoter' s activation and inactivation rates. Transcription can begin only when the promoter is in the active state. λ_{mRNA} and λ_{pro} represent mRNA and protein production rates, respectively, while δ_{mRNA} and δ_{pro} show mRNA and protein degradation rates.

(a) Update a_w.

(b) Generate a random number, t', according to an exponential distribution with parameter a_u, and set $t_u \leftarrow t + t'$.

(c) Replace the old t_u value in P_{GFP} or P_{RFP} with the new value.

7. Go to Step 2.

The model of gene expression used in our simulation is illustrated in Figure 3.5. This is a three step gene expression model, which include the two states of promoter (Figure 3.2), transcription and translation. The two states of promoter are used for introducing biological noise coming from λ_{on} and λ_{off} in simulation.

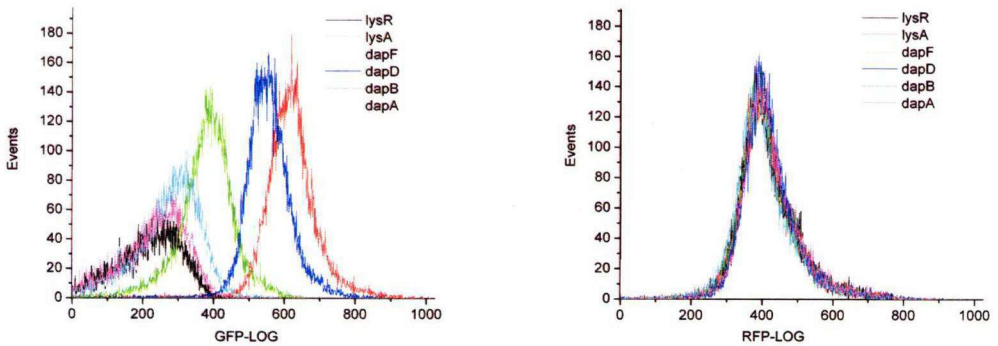
The source code is listed in AppendixB.3. Multiple thread is used to increase the simulation efficiency for such big amount particles. Binary tree is employed to store the time & reaction in pair using time as the sorting order (from low to high). The time & reaction in pair are gotten by Equation (3.1). The reaction whose putative time is first is performed. Simulation time is then incremented by this reaction time. Same as experiment, we add long enough preculture step to gain the initial condition for noise analysis. To measure the extrinsic and intrinsic noise in the simulation, we used same method as in the experimental data analysis (Equation (3.4)).

Lots of parameters can be set during initialize step, which include the growth speed, the active and inactive reaction rate of target promoter, the transcription rate, the translation rate and the degradation rate. All the noise distribution can be controlled by these values. The repression rate of degradation rate δ_{mRNA} and δ_{pro} should equal $\ln(2)/\tau_d$, where τ_d represents the half-life of the mRNA or protein. Eighty percent of mRNA half-lives are in the range of 3–8 min; for simplicity we set the average half-life of mRNA as 3 min, $\delta_{\mathrm{mRNA}} = 0.23\mathrm{min}^{-1}$ for GFP and mRNA. The GFP and RFP half-lives are long enough to ignore; thus, cell division dominates the protein degradation rate (δ_{pro}). In our independently measured generation, time $\tau_d = 40$ min, so that δ_{pro} was set to 0.02 min^{-1}. For simplicity, the constitutive synthesis rates of mRNA, λ_{mRNA}, and protein, λ_{pro}, are set by the average transcription rate and translation rate of genes in *E.coli*. If the promoter is in the active state, transcription and translation can occur at any time. The average transcription rate is approximately 25 nt/s and the average translation rate is approximately 30 aa/s (Golding et al. 2005). Thus, the parameters obtained were as follows: $\lambda_{\mathrm{mRNA_{gfp}}} = 708\ \mathrm{nt}/25\ \mathrm{nt} \cdot \mathrm{s}^{-1} = 2.1\ \mathrm{min}^{-1}, \lambda_{\mathrm{mRNA_{rfp}}} = 675\ \mathrm{nt}/25\ \mathrm{nt} \cdot \mathrm{s}^{-1} = 2.2\ \mathrm{min}^{-1}, \lambda_{\mathrm{pro_{gfp}}} = 236\ \mathrm{aa}/30\ \mathrm{aa} \cdot \mathrm{s}^{-1} = 7.5\ \mathrm{min}^{-1}, \lambda_{\mathrm{pro_{rfp}}} = 225\ \mathrm{aa}/30\ \mathrm{aa} \cdot \mathrm{s}^{-1} = 8\ \mathrm{min}^{-1}$.

3.3 Results

Nine reporter strains were constructed by cloning the promoter region of genes involved in lysine biosynthesis upstream of *gfp* of pGRFP.

Based on previous study(Ou et al. 2008), among the nine promoters the expression levels of five promoters (*lysC*p, *asd*p, *dapB*p, *dapD*p, and *lysA*p) changed with the addition or removal of L-lysine from the medium. The noise behaviors also show the same changes. We then focus on the noise behaviors of this five promoters.



Distribution of LOG GFP Distribution of LOG RFP

Figure 3.6: The event counts of GFP and RFP expression in log-scale of report strains with different promoters involved in lysine biosynthesis in late log phase.

3.3.1 Compensation in flow cytometry

The term "compensation", as it applies to flow cytometric analysis, refers to the process of correcting for fluorescence spillover, i.e., removing the signal of any given fluorechrome from all detectors except the one devoted to measuring that dye (Roederer 2002). In order to determine the GFP and RFP abundance correctly, the green and red fluorescence of reporter strains were checked by flow cytometry during late log phase without lysine in the media. As it shown in Figure 3.6, with different activation strength from different promoters, the fluorescent intensity of GFP is different, but that of RFP is similar in the same culture phase. This result indicated the interaction between GFP and RFP fluorescent intensity can be ignored.

3.3.2 Self-fluorescence background

The fluorescent intensity measured by flow-cytometry include two parts, one is the self-fluorescence of the *Escherichia coli* DH1, another part is the fluorescent intensity emitted by fluorescent protein. The background of self-fluorescence of reporter strains was evaluated by detecting GFP and RFP fluorescent intensity of *E.coli* DH1, which do not contain any plasmid. The background of the self-fluorescence was simulated as Gumbel

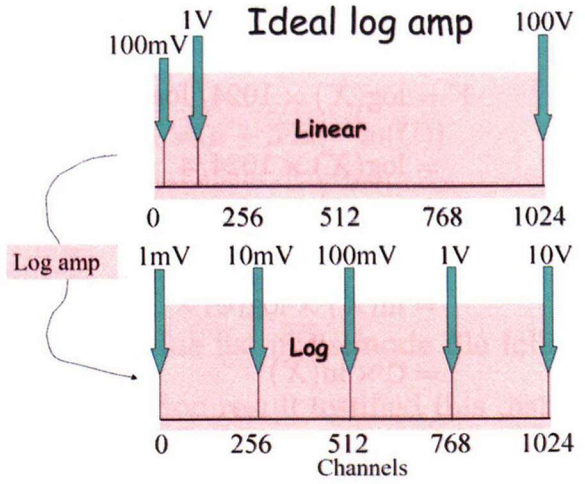


Figure 3.7: The comparision between linear scale data and log scale data

distribution (minimum).

The circuits of flow-cytometry have two kinds, one is linear circuits, whose output signal is proportion to the sum and/or difference of their input signal, and another is logarithmic circuits, whose output signal is in logarithmic scale. Usually in the output file, the signal will be saved as channels but not the pulse output. Flow-cytometry amplify signals to values ranging between 0-10V before performing a digital conversion. And then use a 10 bit ADC (Analog-to-Digital Converter) to convert the linear analog signals. So there are 1024 channels of range (2^n) ($2^{10} = 1024$) corresponding to the range 0-10V. Channels difference is 10/1024=9.8mV per channel. Sometimes we are more focus on the small-side signal, the log-scale data is required. Figure 3.7 show the ideal comparison between linear and log data in flow-cytometry.

With 4 decates, converting the linear scale to a log scale is given by the equation,

$$\begin{aligned} Y &= \log(X) \times 1024/\log(10^4) \\ &= \log(X) \times 1024/4 \\ &= \log(X) \times 256 \\ &= \ln(X) \times \log(\mathrm{e}) \times 256 \\ &= \mathrm{C} \times \ln(X) \end{aligned} \tag{3.5}$$

where Y is the log-scale value, X is the linear scale value and $\mathrm{C} = \log(\mathrm{e}) \times 256$.

Set the self-fluorescence linear analog signals in flow-cytometry follow an approximate exponential distribution,

$$f(x;\lambda) = \begin{cases} \lambda \mathrm{e}^{-\lambda x}, x \geq 0 \\ 0, x < 0 \end{cases} \tag{3.6}$$

The quantile function (inverse cumulative distribution function) for Exponential(λ) is

$$F^{-1}(p;\lambda) = \frac{-\ln(1-p)}{\lambda} \tag{3.7}$$

for $0 \leq p < 1$. So to generate exponential variates we can use,

$$\begin{aligned} T &= \frac{-\ln(1-U)}{\lambda} \\ &= \frac{-\ln(U)}{\lambda} \end{aligned} \tag{3.8}$$

,given a random variate U drawn from the uniform distribution on the unit interval (0,1). Then the log-scale values are,

$$\begin{aligned} Y &= \mathrm{C} \times \ln(T) \\ &= \mathrm{C} \times \ln(\frac{-\ln(U)}{\lambda}) \\ &= \mathrm{C} \times (\ln(-\ln(U)) - \ln(\lambda)) \\ &= \mathrm{C} \times \ln(-\ln(U)) + \mathrm{C0} \end{aligned} \tag{3.9}$$

,where $\mathtt{C0} = -\mathtt{C} \times \ln(\lambda)$. This equation is same from the quantile function for Gumbel distribution (minimum),

$$X = \mu + \beta \ln(-\ln(U)) \tag{3.10}$$

,where μ is location parameter, β is scale parameter and given a random variate U drawn from the uniform distribution on the unit interval (0,1).

That is to say the log scale value list in listmode file follow the Gumbel distribution (minimum). And the simulation result testified this distribution. The probability density function of the Gumbel (minimum) distribution is,

$$\begin{aligned} f(x) &= \frac{1}{\beta} Z \mathrm{e}^{Z} \\ Z &= \mathrm{e}^{\frac{x-\mu}{\beta}} \end{aligned} \tag{3.11}$$

where μ is the mean parameter and β is the scale parameter.

In stochastic simulation algorithm (SSA), different from systems of chemical equations as continuous, represented by differential equations are concentrations, the variables are numbers of molecules. So what I get from the algorithm based on Gillespie algorithm is the numbers of GFP and RFP. To calculate the average RFP or GFP fluorescent intensity, the following equation is employed,

$$\begin{aligned} \langle r \rangle &= \langle RFP \times f_{\mathrm{RFP}} + B_{\mathrm{RFP}} \rangle \\ \langle g \rangle &= \langle GFP \times f_{\mathrm{GFP}} + B_{\mathrm{RFP}} \rangle \end{aligned} \tag{3.12}$$

,where RFP and GFP denote the number of RFP and GFP, respectively. f_{RFP} and f_{GFP} is the intensity of single protein molecule for RFP and GFP, respectively. And B_{RFP} and B_{GFP} denote the self-fluorescence of the cell. When the reporter genes are controlled by promoter $\mathtt{P_L P_R}$, the fluorescent intensity comes from the reporter fluorescence protein is far larger than the self-fluorescence,

$$RFP \times f_{\mathrm{RFP}} \gg B_{\mathrm{RFP}} \quad \text{and} \quad RFP \times f_{\mathrm{RFP}} \gg B_{\mathrm{RFP}} \tag{3.13}$$

Equation 3.12 can be simplified as,

$$\begin{aligned} \langle r \rangle &= \langle RFP \times f_{\mathrm{RFP}} \rangle \\ \langle g \rangle &= \langle GFP \times f_{\mathrm{GFP}} \rangle \end{aligned} \tag{3.14}$$

Then, the total noise for GFP can be calculated as,

$$\begin{aligned} \eta^2_{g-tot} &= \frac{\langle g^2 \rangle - \langle g \rangle^2}{\langle g \rangle^2} \\ &= \frac{\langle (GFP \times f_{\mathrm{GFP}})^2 \rangle - \langle GFP \times f_{\mathrm{GFP}} \rangle^2}{\langle GFP \times f_{\mathrm{GFP}} \rangle^2} \\ &= \frac{\langle GFP^2 \rangle \times f^2_{\mathrm{GFP}} - \langle GFP \rangle^2 \times f^2_{\mathrm{GFP}}}{\langle GFP \rangle^2 \times f^2_{\mathrm{GFP}}} \\ &= \frac{\langle GFP^2 \rangle - \langle GFP \rangle^2}{\langle GFP \rangle^2} \end{aligned} \tag{3.15}$$

The reporter strain, whose *gfp* and *rfp* both are controlled by promoter $\mathrm{P_L P_R}$ can help us to set the appropriate f_{GFP} and f_{RFP} after the biological noise being fitted. The f_{GFP} was set as 10 and f_{RFP} is set as 3.35, which compare well with the results gotten in Sugiyama et al. (2005).

After setting the intensity of single protein molecule of GFP and RFP, the background was fitted. Because the self-fluorescent intensity detected as green fluorescence and red fluorescence have no correlation, the experiment data of the background can be fitted by two unrelated group of data in Gumbel distribution (minimum) (Figure 3.8). And the expression of *gfp* and intrinsic noise can be well fitted (Figure 3.9). But also in the experiment data we found a negative extrinsic noise. This indicated that there is negative correlation between the GFP fluorescent intensity and RFP fluorescent intensity. This may be the slight stochasticity in experiments. This background of self-fluorescent intensity was directly added to each cell when I simulated the gene expression based on Gibson's Next Reaction Method algorithm, and it followed the Gumbel distribution (minimum).

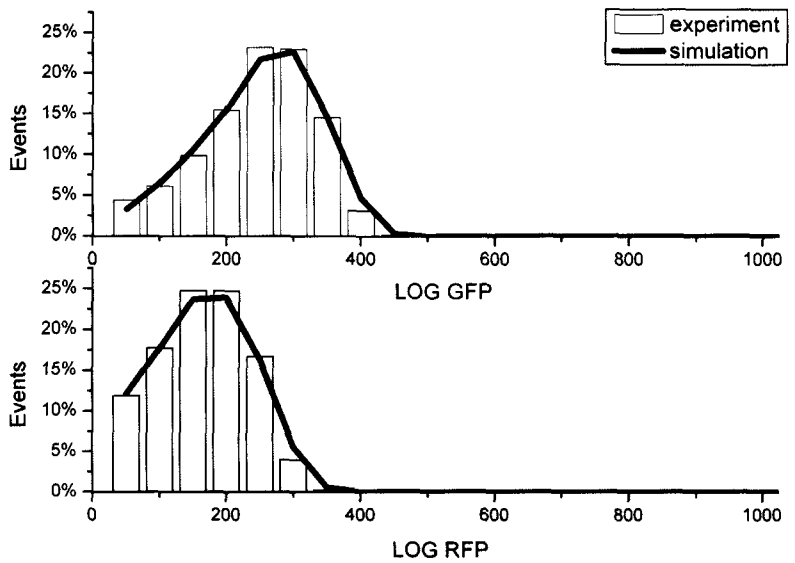


Figure 3.8: The background of *E.coli* DH1 self-fluorescent intensity was simulated as two unrelated group of data in Gumbel distribution (minimum)

3.3.3 Intrinsic noise of promoters involved in lysine biosynthesis

We measured the expression of GFP and RFP at the single-cell level using flow cytometry. Using the reporter strains, we investigated the change of GFP abundance caused by removal of L-lysine from the medium. In Figure 3.10, we plotted the protein abundance according to time after removing L-lysine from the medium. As shown, the protein levels increased when L-lysine was removed and settled into different states 120–150 min after removal. Then, as described in the materials and methods section, we calculated the amplitude of intrinsic noise in this process. In Figure 3.11, we plotted the relationship between protein abundance and the amplitude of intrinsic noise. As shown in the figure, the amplitude of the fluctuations approximately obeys $\eta_{int}^2 \propto x^{-1}$, where x represents the protein abundance. This result was consistent with previous studies (Bar-Even et al. 2006), in which the intrinsic noise caused by the stochasticity within transcription and translation generally follows a Poisson distribution; thus, the relationship $\eta_{int}^2 \propto x^{-1}$ was expected. However, it should be stressed that in the *lysA*p

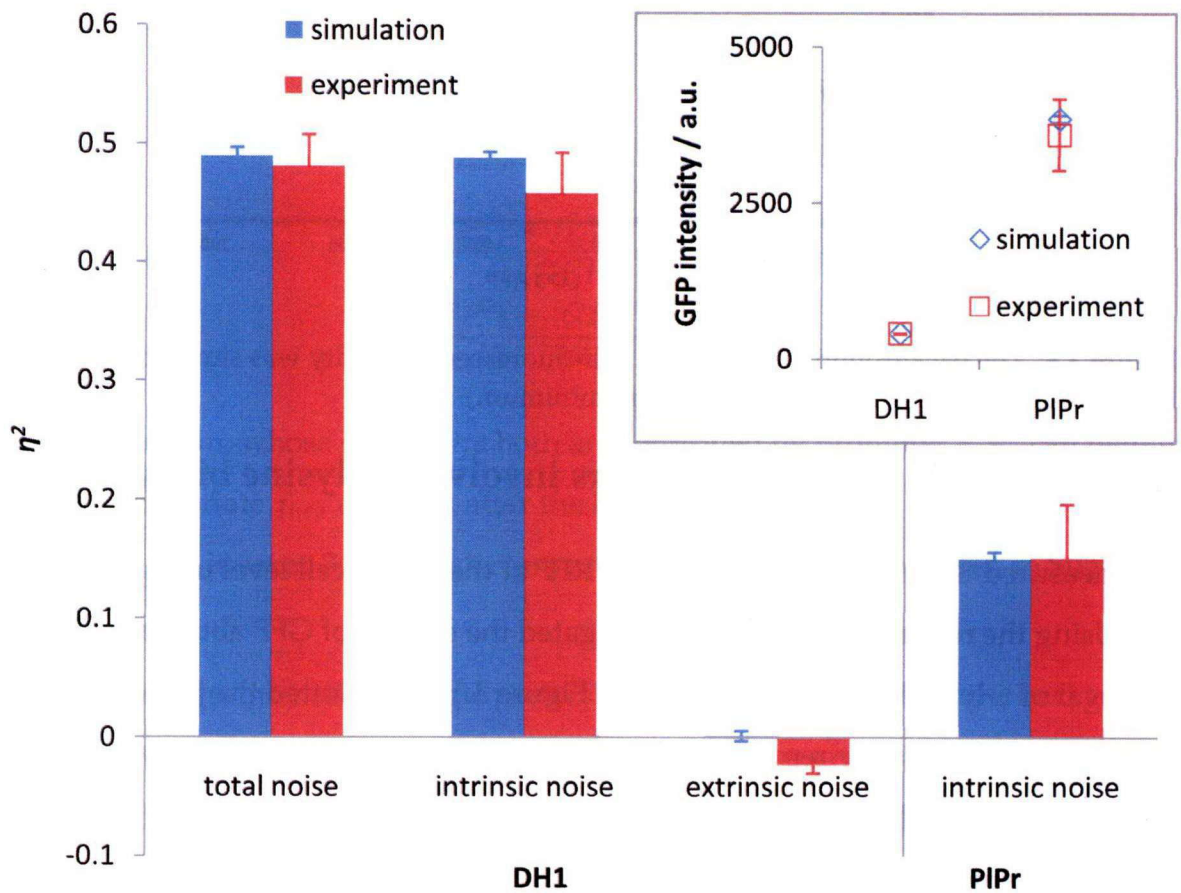


Figure 3.9: The noise of negative control. DH1: *E.coli* DH1; PlPr: both of the two fluorescent gene controlled by strong constitutive promoter $P_R P_L$

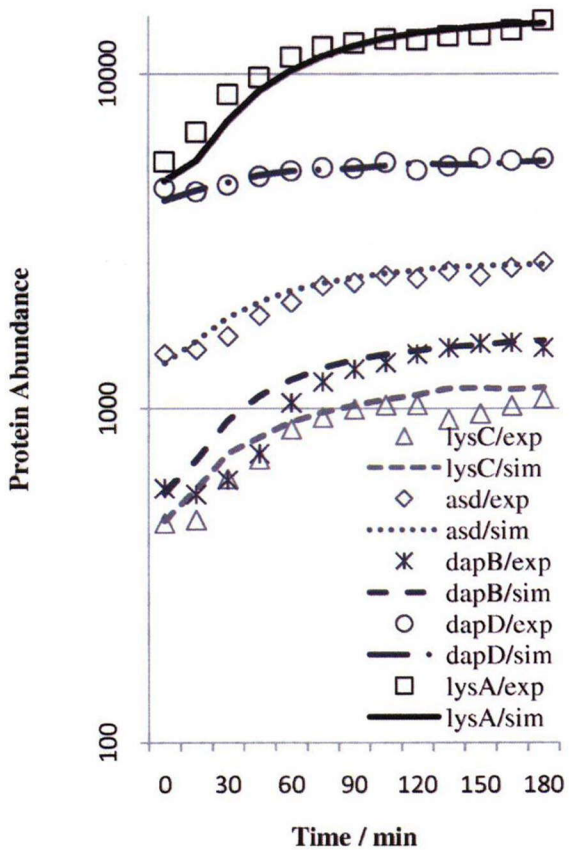


Figure 3.10: Time series showing GFP abundance after changing the environment from a medium containing lysine to one without lysine. The dots represent experimental data, while the lines represent simulation results after fitting parameters λ_{on} and λ_{off} for each promoter.

strain, the amplitude of intrinsic noise was significantly higher than that expected from the trend line. Moreover, the noise amplitude in the *dapD*p reporter strain was slightly smaller than the trend line predicted, even though the expression levels of GFP in *lysA*p and *dapD*p strains were similar. Since only the difference between these two strains is the promoter region upstream of *gfp* and thus transcription and translation kinetics were identical, the difference in the noise characteristics should be due to promoter activity dynamics. That is, this result strongly suggests that stochasticity in the promoter activation process is a major source of noise in protein expressions in prokaryotes.

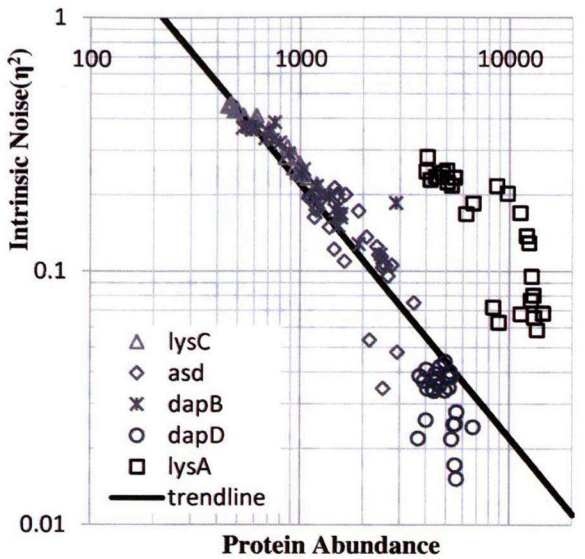


Figure 3.11: The experimentally obtained relationship between protein abundance and the intrinsic noise amplitude. The solid line represents the relationship $\eta_{int}^2 \propto x^{-1}$.

3.3.4 Correlation between $\lambda_{\mathtt{off}}$ and intrinsic noise

Next, to quantitatively evaluate stochasticity in the promoter dynamics, we performed a simple stochastic simulation of protein expressions. The schematic representation of the model is presented in Figure 3.5. As discussed above, the only difference among strains was the variation in the promoter region upstream of the *gfp* gene. Therefore, the differences in protein abundance and the amplitude of intrinsic noise should be described by the parameters $\lambda_{\mathtt{on}}$ and $\lambda_{\mathtt{off}}$, while other parameters ($\lambda_{\mathtt{mRNA}}$,$\lambda_{\mathtt{pro}}$,$\delta_{\mathtt{mRNA}}$, and $\delta_{\mathtt{pro}}$ in Figure 3.5) were set to identical values among the strains. Of course, the model presented in Figure 3.5 contains only a simplified version of the process, and we have omitted several complicated transcription and translation processes. However, this simplified model helps us make a quantitative evaluation for the essential processes, especially for the stochastic dynamics represented by parameters $\lambda_{\mathtt{on}}$ and $\lambda_{\mathtt{off}}$.

Let X_1, X_2, X_3, N be the amounts of active promoters, mRNAs, proteins and plasmid copy number, respectively. For the model in Figure 3.5, the amounts of active promoters,

mRNAs and proteins can be calculated as follows:

$$\begin{aligned}
\frac{\mathrm{d}X_1}{\mathrm{d}t} &= \lambda_{\mathrm{on}} \times (N - X_1) - \lambda_{\mathrm{off}} \times X_1, \\
\frac{\mathrm{d}X_2}{\mathrm{d}t} &= \lambda_{\mathrm{mRNA}} \times X_1 - \delta_{\mathrm{mRNA}} \times X_2, \\
\frac{\mathrm{d}X_3}{\mathrm{d}t} &= \lambda_{\mathrm{pro}} \times X_2 - \delta_{\mathrm{pro}} \times X_3
\end{aligned} \tag{3.16}$$

The average number of the molecules at stationary state is:

$$\begin{aligned}
\langle X_1 \rangle &= \frac{N \times \lambda_{\mathrm{on}}}{\lambda_{\mathrm{on}} + \lambda_{\mathrm{off}}}, \\
\langle X_2 \rangle &= \frac{\lambda_{\mathrm{mRNA}}}{\delta_{\mathrm{mRNA}}} \langle X_1 \rangle, \\
\langle X_3 \rangle &= \frac{\lambda_{\mathrm{pro}}}{\delta_{\mathrm{pro}}} \langle X_2 \rangle,
\end{aligned} \tag{3.17}$$

Thus,

$$\langle X_3 \rangle = \frac{N \times \lambda_{\mathrm{on}} \times \lambda_{\mathrm{mRNA}} \times \lambda_{\mathrm{pro}}}{(\lambda_{\mathrm{on}} + \lambda_{\mathrm{off}}) \times (\delta_{\mathrm{mRNA}} \times \delta_{\mathrm{pro}})} \tag{3.18}$$

The copy number of plasmids as set as 30(Lutz and Bujard 1997). Using the determined λ_{mRNA}, λ_{pro}, δ_{mRNA} and δ_{pro}, we simulated the correlation between expression and intrinsic noise for different λ_{off}. As it shown in the Figure 3.12, at same expression level, to get the smaller intrinsic noise needs smaller λ_{off}. We traced the number of active promoter and mRNA with low and high λ_{off}. The transcriptional bursting caused by slow promoter kinetics was well visualized (Figure 3.13).

3.3.5 Data fitting

Using this model, we fitted the parameters λ_{on} and $\lambda_{\mathtt{off}}$ of the model to reproduce the experimentally obtained abundance of proteins and the intrinsic noise amplitude shown in Figure 3.10 and Figure 3.14. In this experiment, we changed the culture condition from a M9A medium containing L-lysine (0.3 mM) to one without L-lysine after pre-culture. For simplicity, we assumed that the promoter dynamics have the same $\lambda_{\mathtt{off}}$ in the two environmental conditions, and changes in protein abundance and the intrinsic noise amplitude are represented by the change of the parameter λ_{on} between these

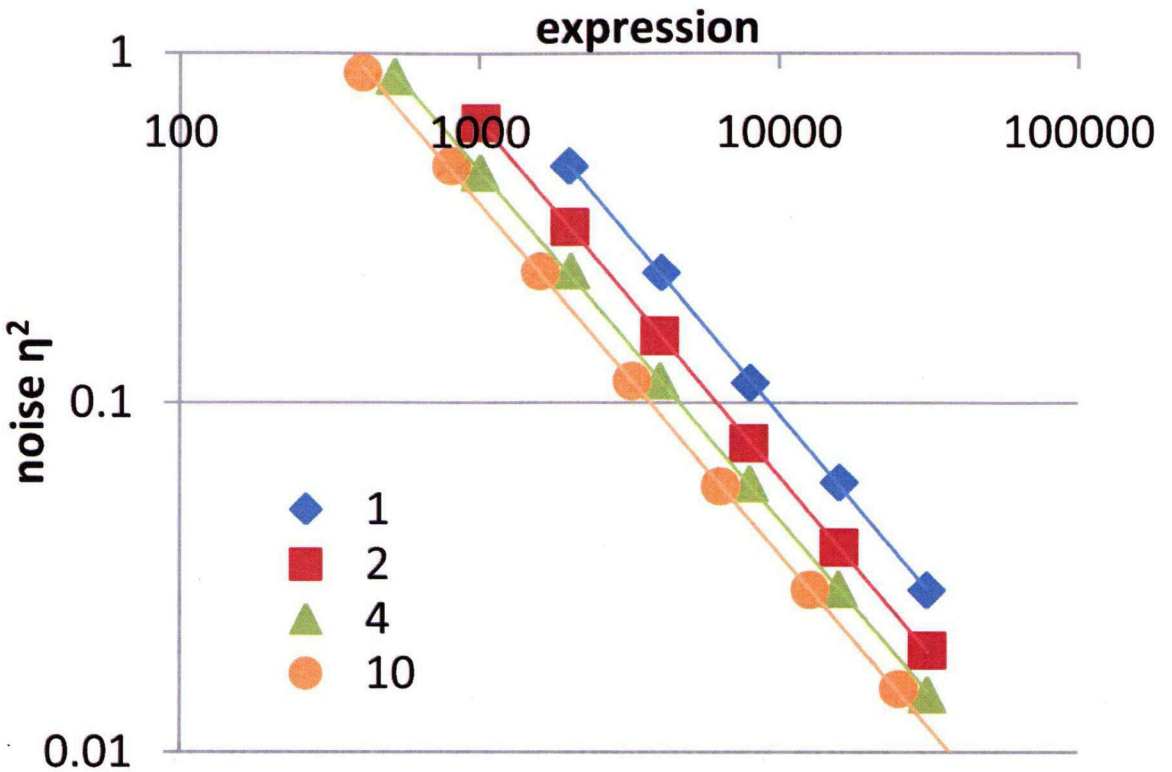


Figure 3.12: The correlation between expression and intrinsic noise for different λ_{off}.

environmental conditions. The fitting results of the protein abundance and the intrinsic noise are also shown in the Figure 3.10 and Figure 3.14. As shown, the fitted results showed good agreement with the experimentally obtained data. The fitted parameters are shown in Table 3.8 with column header as 'removal of L-lysine'. The fitted results showed that *lysC* and *dapD* have high λ_{on} and λ_{off}. In contrast, *lysA* has low λ_{on} and λ_{off}, indicating that the promoter transitions infrequently between active and inactive states. The difference in the stochastic dynamics of the promoter activity might suggest that the molecular machineries of activation and repression in the various promoter regions have different characteristics. Moreover, the variation in noise amplitude among the promoters might be linked to their different roles in the lysine biosynthesis pathway. For example, as reported by Blake et al. (2006), increase in gene expression noise could provide a significant selective advantage at high stress levels, while a strain with lower

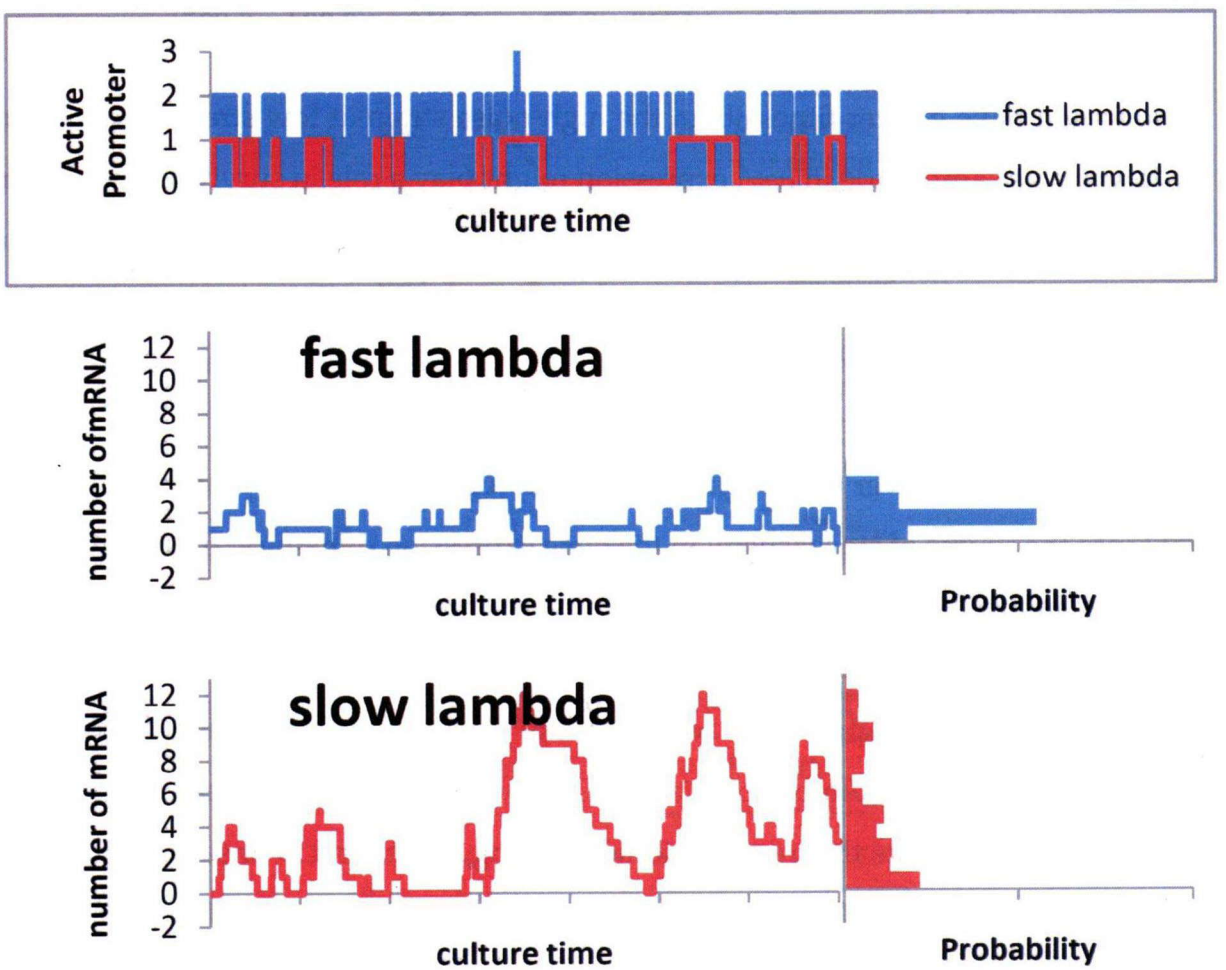


Figure 3.13: The transcriptional bursting caused by slow promoter kinetics.

Table 3.8: Parameters used in simulation (min^{-1}).

	Addition of L-lysine			Removal of L-lysine		
Promoter	$\lambda_{on_{0.3mM}}$	$\lambda_{on_{0mM}}$	λ_{off}	$\lambda_{on_{0.3mM}}$	$\lambda_{on_{0mM}}$	λ_{off}
lysC	0.28	0.018	323	0.36	0.017	323
asd	0.21	0.046	55	0.145	0.043	55
dapB	0.25	0.025	100	0.13	0.01	88
dapD	4.58	3.22	880	4.58	3.22	880
lysA	0.011	0.0025	0.70	0.013	0.004	0.825

noise would have greater fitness than the high-noise strain at low stress levels. Thus, the larger noise amplitude in *lysA* expression might play a role in survival in severe environments. As reported in our previous study (Ou et al. 2008), it should be noted that the parameters for stochastic promoter dynamics, λ_{on} and λ_{off} cannot be obtained by using the single-fluorescent reporter system, because the analysis of the intrinsic noise in the protein expression requires a multi-color reporter system.

The finding of slower transition rate for the promoter *lysA* can provide a novel insight into the source of stochasticity in the gene expression dynamics. In the expression dynamics in prokaryotic cells, it was generally assumed that the translational bursting is a dominant source of stochasticity in the process of expression (Kærn et al. 2005), while the transition of promoter activation are assumed to be enough fast and the rate of transcription is almost constant. However, our result suggested that the at least in *lysA* expression, the transcriptional bursting (Raser and O'Shea 2004, Ozbudak et al. 2002, Blake et al. 2006) can be a dominant source of intrinsic noise. This finding contributes to the further understanding of stochastic nature in the expression dynamics.

3.3.6 Validation of the parameters

To evaluate the accuracy of parameter estimation, the same parameter values were obtained by using data of an independent experiment. In that experiment, the change of lysine concentration in the environment was in opposite direction to that used in Fig-

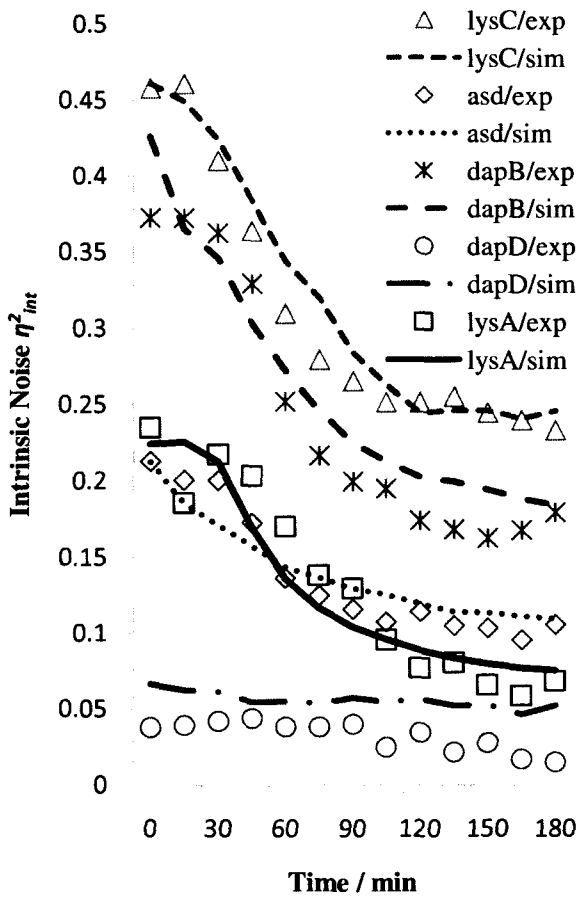


Figure 3.14: Time series showing the intrinsic noise amplitude after changing the environment from a medium containing lysine to one without lysine. The dots represent experimental data, while the lines represent simulation results using the same fitting parameters as in Figure 3.5

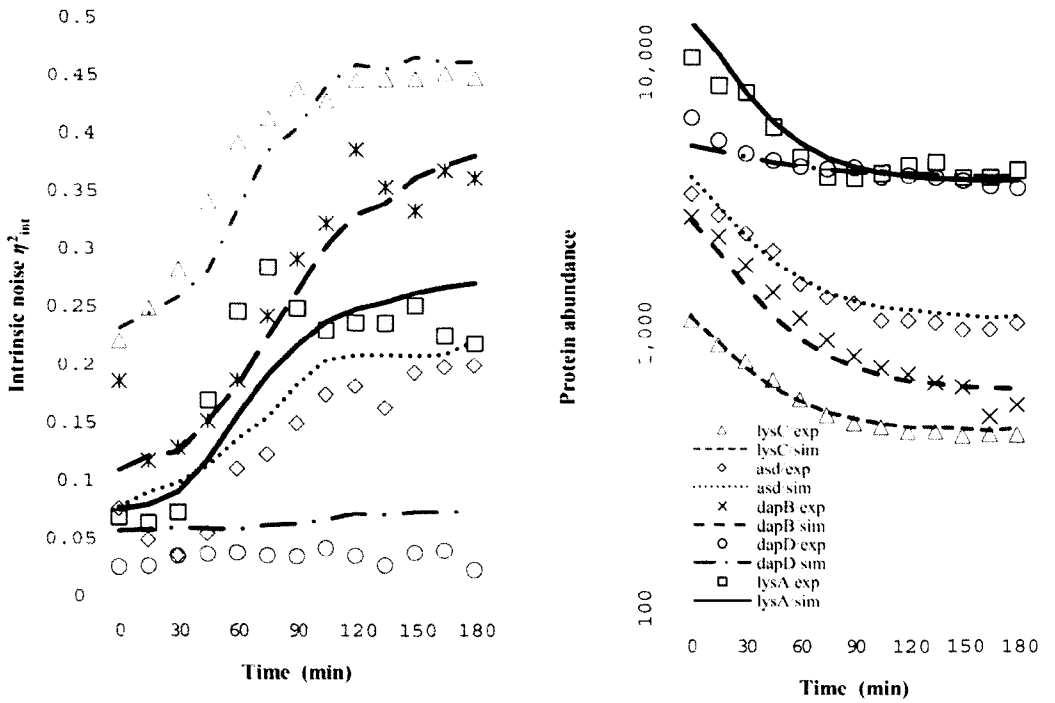


Figure 3.15: Time series showing GFP abundance and intrinsic noise amplitude after addation of L-lysine into the culture media. The dots represent experimental data, while the lines represent simulation results after fitting parameters λ_{on} and λ_{off} for each promoter.

ure 3.10 and Figure 3.14, i.e., the change of promoter activity was quantify when lysine was added to the environment. The experimental and fitting results are presented in Figure 3.15, and the parameter values estimated by using this experiment is also listed in Table 3.8. As shown in the Table 3.8, the parameter values obtained by two independent experiments showed a good agreement between each other. This result suggested that the parameter estimation used in our study was enough accurate to evaluate the stochastic dynamics of promoter activity.

3.3.7 Comparison between single and dual fluorescence experiments results

Mathematical models are an essential tool in systems biology, linking the behavior of a system to the interactions between its components. Parameters in empirical mathe-

matical models must be determined using experimental data. As models are approximations of reality, it is likely that more than one model fits the data to an acceptable degree(Jaqaman and Danuser 2006a).

As mentioned above, the promoter strength of genes involved in lysine biosynthesis pathway were described by two different model with different parameters. The model used in single fluorescence reporters are simple and that used in dual-fluorescence systems can involve the information of biological noise. Both of them can fit the expression data well. Here come the questions, is there any relationship between the two group parameters? Are they consistent with each other? Or do they contradict with each other?

In single fluorescence experiments, we used Michaelis-Menten-type model to fitting the data (Equation (2.2)). The average number of the molecules at stationary state can be obtained as

$$\begin{aligned} \frac{d\langle p\rangle}{dt} &= \beta\frac{1}{1+R(t)/k_i} - \alpha\langle p\rangle \\ 0 &= \beta\frac{1}{1+R(t)/k_i} - \alpha\langle p\rangle \\ \langle p\rangle &= \frac{\beta}{\alpha}\times\frac{1}{1+R(t)/k_i} \end{aligned} \tag{3.19}$$

We already got the average protein number at stationary state in dual-fluorescence system (Equation (3.18)).

$$\langle X_3\rangle = \frac{N\times\lambda_{\text{on}}\times\lambda_{\text{mRNA}}\times\lambda_{\text{pro}}}{(\lambda_{\text{on}}+\lambda_{\text{off}})\times(\delta_{\text{mRNA}}\times\delta_{\text{pro}})} \tag{3.20}$$

As we know, the $\alpha \approx \delta_{\text{pro}}$ and in both systems, the expression level should be similar ($\langle p\rangle \propto \langle X_3\rangle$). If we set

$$\beta = \mathrm{C}\times\varpi;\ \mathrm{C} = N\times\frac{\lambda_{\text{mRNA}}\times\lambda_{\text{pro}}}{\delta_{\text{mRNA}}} \tag{3.21}$$

Thus,

$$\langle p\rangle \propto \langle X_3\rangle \Rightarrow \frac{\varpi}{1+R(t)/k_i} \propto \frac{\lambda_{\text{on}}}{\lambda_{\text{on}}+\lambda_{\text{off}}} \tag{3.22}$$

Then we get the correlation between the two groups of parameters. We can determine the maximal promoter activity (β) and repression coefficient(k_i) by the parameters of λ_{on}

Table 3.9: repression coefficient(k_i)

gene	single-fluorescence	dual-fluorescence
lysC	1.6	0.046
asd	0.55	0.40
dapB	0.69	0.63
*dap*D	2.98	2.26
lysA	0.22	0.29

and λ_{off}. After calculation, the results are shown in Table 3.9 and Table 3.10. From the results, except *lys*Cp, all the k_i are consistent with each other and there is a constant ratio in β. As we know, the maximal we got from the single fluorescence experiments are not the real number of proteins but the green fluorescent intensity. So this constant ratio indicates the fluorescent intensity per GFP in single fluorescence experiments. However, in the dual-fluorescence simulation, the fluorescent intensity per GFP is set as 10, which is smaller than the ratio(15.43±2.26) we got here. This is caused by the difference in treating with the self-fluorescence background. In single fluorescence experiment, we directly substrate the background from the intensity data, while in dual-fluorescence systems, we simulated the background in Gumbel distribution(minimum). This difference will lead the distance of the fluorescent intensity between different culture condition larger in single fluorescence experiments. And the larger distance will increase the maximal promoter activity. The result of GFP expression controlled by *lys*Cp was too close to the background as determined by flow cytometer. This background noise may interfere with the signal. So the result of *lys*Cp still need discussion. The parameters for the same promoter we got from single fluorescent experiments and dual-fluorescent system were compared to find the correlation between them. The validity of parameters of promoter strength determined by dual-fluorescence system was verified by the results gotten from single fluorescence experiments. This result tells the parameters of promoter strength we got from dual-fluorescence system can include the information gotten from single fluorescence experiments.

Table 3.10: maximal promoter activity(β)

gene	single-fluorescence	dual-fluorescence	ratio
lysC	14.4	2.19	6.57
asd	68.2	5.18	13.17
dapB	46.3	2.90	15.94
dapD	152.2	10.19	14.93
lysA	570.1	30.55	18.66
			15.43±2.26

3.3.8 Application of dual-fluorescence system

To discuss the stochasticity in more complex regulatory systems, the first problem is that how to detect the noise in comparable system. If we construct all the regulatory circuits in one plasmid, we need lots of reporter fluorescence as much as the number of genes in the regulatory circuits. Not only multiple fluorescence proteins but also separating them by single cell analysis are problem. Another method is that we construct only one gene in the regulatory circuits in one plasmid and make a group of such kind of plasmids. By this method, the dimension of the circuits becomes unlimited. The dual-fluorescence system is based on this method. There is an endogenous control in each plasmid (the *rfp* controlled by $P_L P_R$) to make the system comparable and the distinguishing intrinsic from extrinsic noise possible. This feature provides the possibility to analyze the big complex metabolism network in system level.

The dual-fluorescence system, which includes two parts: the vector pGRFP and simulation tool, was well developed for the promoter strength analysis. The pGRFP vector contains two distinguishable fluorescent report genes. It can separate the intrinsic noise from extrinsic noise for target promoter. By fitting the expression of *gfp* and the noise of the expression, the simulation tool, which is based on a stochastic formulation of chemical kinetics derived by *Gillespie*, can easily get appropriate λ_{on} and λ_{off} for the target promoter. This system can be used for enriching the acknowledge of endogenous

promoters and complex regulation network. It is helpful for recovering the relationship between promoter sequence and promoter strength, which is impossible because of lacking data(Wray et al. 2003). Because the λ_{on} and λ_{off} also contain the information about the possibility of transcriptional bursting noise, the promoter strength gotten by this system also can be used for further noise analysis and prediction.

3.4 Conclusion

The dual-fluorescence system for promoter strength analysis was developed. This system includes two parts, the vector pGRFP and simulation tool. The reporter strains is constructed by cloning the promoter region of target genes into pGRFP vector. The green and red fluorescent intensity of the reporter strains is analyzed by flow-cytometry in order to get the expression and noise of *gfp* and *rfp*. The intrinsic noise of the promoters is determined by subtracting extrinsic noise from the total noise. After that the simulation tool is used for getting appropriate λ_{on} and λ_{off} by fitting the expression and noise based on Gibson's Next Reaction Method algorithm.

We applied this system to analyze the λ_{on} and λ_{off} of promoters involved in lysine biosynthesis. Time-dependent experiment was performed after changing the culture condition (L-lysine presence and L-lysine absence). The dynamic change in expression and noise of five promoters, which respond to the L-lysine shifting, was fitted well by a group of λ_{on} and λ_{off}. We found that *lysA*p has low λ_{on} and λ_{off}. The slow transitions between promoter states of *lysA*p indicates the transcriptional bursting also can be a source of noise in prokaryotic cells.

The dual-fluorescence system firstly introduced the biological noise into the promoter strength analysis and it can provide appropriate λ_{on} and λ_{off} for target promoter. This system can be easily used for strength analysis of grouped promoters in an endogenous regulation network. The parameters of λ_{on} and λ_{off} can not only describe the

activation of repression by a given factor but also provide information about the biological noise for the promoter. This will help us to understand the initiation of transcription in a quantitative way and to predict the possible level of mRNA.

Chapter 4

General conclusion

Systems biology is an approach to biology that seeks to understand and predict the quantitative features of a multicomponent biological system (Kitano 2002, Hartwell et al. 1999, Jaqaman and Danuser 2006b). However there is still lack of powerful technique for acurrate comparable analysis of complex endogenous networks. Followed the deepened research in biological noise, there is a need to develop experiment in the study of promoter strength with the stochasticity property in more complex regulatory systems, particularly endogenous gene networks (Kærn et al. 2005). In order to conduct a transition form a descriptive to a quantitative understanding, which can assign kinetic parameters that capture the dynamics of the network within both deterministic and stochastic model, in endogenous regulation network of *Escherichia coli*, we tried to perform the single-cellular analysis in lysine biosynthesis.

In Charpter 1, the background and significance of the research was described. Systems biology requires quantitative description of endogenous regulation networks to construct appropriate models which can make predictions about the behavior of the interacting networks. Promoter strength plays a key role in driving gene transcription, which, in turn, cause fluctuation in the interacting networks. The clear background of lysine biosynthesis pathway makes the activation analysis of promoter involved in lysine biosynthesis a good sample involving biological noise in systems biology. Time-dependent experiments was employed in this study because it can result in wealth of information, which makes the model construction feasible. Using flow cytometry, a comprehensive quantitative description of activation of promoters involved in lysine

biosynthesis was analyzed at single cell level.

In Charpter 2, by single fluorescence experiments, we investigated the expression dynamics of genes involved in lysine biosynthesis in *E. coli* cells to obtain a quantitative understanding of the gene regulatory system. By constructing reporter strains expressing the green fluorescence protein gene(*gfp*) under the control of the promoter regions of those genes associated with lysine biosynthesis, the dynamic behavior of promoter activation was well visualized and quantitatively analyzed by flow cytometry. According to the fitting parameters within a deterministic model, *dapD* shows a significantly larger coefficient of repression than the other genes in the lysine synthesis pathway, which indicates the weak binding activity of the repressor to the *dapD* promoter region. Moreover, there is a trend that the closer an enzyme is to the start of the lysine biosynthesis pathway, the smaller its maximal promoter activity is. The results provide a better quantitative understanding of the changes in promoter activation over time in the lysine biosynthesis pathway.

In Charpter 3, a dual-fluorescence system for promoter strength analysis was developed to involve the biological noise information. This system includes two parts, the vector pGRFP and simulation tool. By fitting the expression and intrinsic noise getting from pGRFP vector, simulation tool can easily get appropriate transition rate of the activation/inactivation state, λ_{on} and λ_{off}, for the target promoter based on a stochastic formulation of chemical kinetics derived by *Gillespie*. Multiple thread is used to increase the simulation efficiency for such big amount particles (e.g., 20,000 cells). A new methods in treating with self-fluorescence background was introduced. This background of the self-fluorescence was simulated as Gumbel distribution (minimum). During simulation, this background of self-fluorescent intensity was directly added to fluorescent intensity from fluorescent proteins and contributed to the noise calculation same as in experiments. Our algorithm developed Gibson's Next Reaction Method algorithm by simulating the process same as experiments, which includes the transfer point in cul-

ture condition and dual-fluorescent protein genes in one cell. We applied this system to analyze the kinetics of promoters involved in lysine biosynthesis. We found that *lysA*p has low λ_{on} and λ_{off}. The slow transitions between promoter states of *lysA*p indicates the transcriptional bursting also can be a source of noise in prokaryotic cells.

In conclusion, an experimental and data-analysis technique for quantitative and comparable analysis of promoter activation at single cell level for endogenous regulation networks is developed. The single fluorescence experiments can determine the kinetic parameters within a deterministic model of the regulation network by using accurate promoter-activity measurements. These parameters can be used to compare the difference among different promoter involved in one biosynthesis pathway. The dual-fluorescence system firstly introduced the biological noise into the analysis of promoter strength in endogenous metabolic network and it can provide appropriate λ_{on} and λ_{off} for target promoter. This system can be easily used for strength analysis of grouped promoters in an endogenous regulation network. The parameters of λ_{on} and λ_{off} can not only describe the activation of repression by a given factor but also provide information about the biological noise for the promoter. This will help us to understand the initiation of transcription in a quantitative way and to predict the possible level of mRNA. It also provides the possibility to analyze the big complex metabolic network in system level.

The transition from a qualitative to a quantitative understanding of promoter activation involved in lysine biosynthesis in *E. coli* was conducted. The results provide a better quantitative understanding of the changes in promoter activation over time in the lysine biosynthesis. Previous The parameters within a deterministic model can provide the possibility in accurate prediction of lysine biosynthesis in metabolic engineering when the cells face different concentration of lysine. The significantly higher intrinsic noise in *lysA* indicates the transcriptional bursting also can be a source of noise in prokaryotic cells. This finding contributes to the further understanding of stochastic

nature in the expression dynamics. As reported by Blake et al. (2006), increase in gene expression noise could provide a significant selective advantage at high stress levels. The further finding of transcriptional bursting by using the dual-fluorescent system in other metabolic network can help us to understand the importance of the gene in evolution.

Bibliography

Aderem, A.: 2005, Systems biology: Its practice and challenges, *Cell* **121**(4), 511–513.

Arkin, A., Ross, J. and McAdams, H. H.: 1998, Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected *Escherichia coli* cells, *Genetics* **149**, 1633–1648.

Arriaga, E. A.: 2009, Determining biological noise via single cell analysis, *Analytical and Bioanalytical Chemistry* **393**, 73–80.

Bar-Even, A., Paulsson, J., Maheshri, N., Carmi, M., O'Shea, E., Pilpel, Y. and Barkai, N.: 2006, Noise in protein expression scales with natural protein abundance, *Nat Genet* **38**(6), 636–643.

Berg, J. M., Tymoczko, J. L. and Stryer, L.: 2002, *Biochemistry*, New York: W.H. Freeman and Company.

Bevis, B. J. and Glick, B. S.: 2002, Rapidly maturing variants of the *Discosoma* red fluorescent protein(dsred), *Nature Biotechnology* **20**, 83–87.

Blake, W. J., Balázsi, G., Kohanski, M. A., Isaacs, F. J., Murphy, K. F., Kuang, Y., Cantor, C. R., Walt, D. R. and Collins, J. J.: 2006, Phenotypic consequences of promoter-mediated transcriptional noise, *Molecular Cell* **24**, 853–865.

Bouvier, J., Richaud, C., Higgins, W., Bögler, O. and Stragier, P.: 1992, Cloning, characterization, and expression of the *dapE* gene of *Escherichia coli*, *Journal of Bacteriology* **174**(16), 5265–5271.

Bouvier, J., Richaud, C., Richaud, F., Patte, J.-C. and Stragier, P.: 1984, Nucleotide sequence and expression of the *Escherichia coli dapB* gene, *The Journal of Biological Chemistry* **259**(23), 14829–14834.

Bouvier, J., Stragier, P., Morales, V., Rémy, E. and Gutierrez, C.: 2008, Lysine represses transcription of the *Escherichia coli dapB* gene by preventing its activation by the argp activator, *Journal of Bacteriology* **190**(15), 5224–5229.

Browning, D. F. and Busby, S. J.: 2004a, The regulation of bacterial transcription initiation, *Nat Rev Microbiol* **2**(1), 57–65.

Browning, D. F. and Busby, S. J. W.: 2004b, The regulation of bacterial transcription initiation, *Nature Reviews* **2**(1), 57–65.

Brunner, M. and Bujard, H.: 1987, Promoter recognitionand promoter strength in the *Escherichia coli*, *The EMBO Journal* **6**(10), 3139–3144.

Brynildsen, M. P. and Collins, J. J.: 2009, Systems biology makes it personal., *Mol Cell* **34**(2), 137–138.

Chang, J. T., Carvalho, C., Mori, S., Bild, A. H., Gatza, M. L., Wang, Q., Lucas, J. E., Potti, A., Febbo, P. G., West, M. and Nevins, J. R.: 2009, A genomic strategy to elucidate modules of oncogenic pathway signaling networks., *Mol Cell* **34**(1), 104–114.

Chenais, J., Richaud, C., Ronceray, J. and Cherest, H.: 1981, Construction of hybrid plasmids containing the *lysA* gene of *Escherichia coli*: Studies of expression in *Escherichia coli* and *Saccharomyces cerevisiae*, *Molecular & general genetics* **182**, 456–461.

Cormack, B. P., Valdivia, R. H. and Falkow, S.: 1996, Facs-optimized mutants of the green fluorescent protein (gfp), *Gene* **173**(1), 33–38.

CoxIII, R. S., Surette, M. G. and Elowitz, M. B.: 2007, Programming gene expression with combinatorial promoters, *Molecular Systems Biology* **3**, 145.

Ducrest, A.-L., Mario Amacker, a. J. L. and Nabholz, M.: 2002, Detection of promoter activity by flow cytometric analysis of gfp reporter expression, *Nucleic Acids Research* **30**(14), e65.

Elowitz, M. B., Levine, A. J., Siggia, E. D. and Swain, P. S.: 2002, Stochastic gene expression in a single cell, *Science* **297**, 1183–1186.

Ford, K. G., Whitmarsh, A. J. and Hornby, D. P.: 1994, *Methods in Molecular Biology: DNA-Protein Interactions: Overexpression and Purification of Eukaryotic Transcription Factors as Glutathione-S-Transferase Fusions in E. coli*, Humana Press.

Fraser, D. and Kærn, M.: 2009, A chance at survival: gene expression noise and phenotypic diversification strategies, *Molecular Microbiology* **71**(6), 1333–1340.

Funkhouser, J. D., Abraham, A., Smith, A. and Smith, W. G.: 1974, Kinetic and molecular properties of lysine-sensitive aspartokinase, *The Journal of Biological Chemistry* **249**(17), 5478–5484.

Ghaemmaghami, S., Huh, W.-K., Bower, K., Howson, R. W., Belle, A., Dephoure, N., O'Shea, E. K. and Weissman, J. S.: 2003, Global analysis of protein expression in yeast, *Nature* **425**, 737–741.

Gibson, M. A. and Bruck, J.: 2000, Efficient exact stochastic simulation of chemical systems with many species and many channels, *the Journal of Physical Chemistry A* **104**, 1876–1889.

Gillespie, D. T.: 1977, Exact stochastic simulation of coupled chemical reactions, *the Journal of Physical Chemistry* **81**(25), 2340–2361.

Gillespie, D. T.: 1992, A rigorous derivation of the chemical master equation, *Physica A* **188**, 404–425.

Goh, K.-I., Cusick, M. E., Valle, D., Childs, B., Vidal, M. and Barabási, A.-L.: 2007, The human disease network., *Proc Natl Acad Sci U S A* **104**(21), 8685–8690.

Golding, I., Paulsson, J., Zawilski, S. M. and Cox, E. C.: 2005, Real-time kinetics of gene activity in individual bacteria, *Cell* **123**, 1025–1036.

Hartwell, L. H., Hopfield, J. J., Leibler, S. and Murray, A. W.: 1999, From molecular to modular cell biology, *Nature* **402**(6761 Suppl), C47–C52.

Haziza, C., Stragier, P. and Patte, J.-C.: 1982, Nucleotide sequence of the asd gene of *Escherichia coli*: absence of a typical attenuation signal, *The EMBO Journal* **1**(3), 379–384.

Ito, Y., Kawama, T., Urabe, I. and Yomo, T.: 2004, Evolution of an arbitrary sequence in solubility, *Journal of Molecular Evolution* **58**(2), 196–202.

Jaqaman, K. and Danuser, G.: 2006a, Linking data to models: data regression, *Molecular Cell Biology* **7**, 813–819.

Jaqaman, K. and Danuser, G.: 2006b, Linking data to models: data regression, *Nat. Rev. Mol. Cell Biol.* **7**(11), 813–819.

Jin, J. H., Choi, K. K., Jung, U. S., In, Y. H., Lee, S. Y. and Lee, J.: 2004, Regulatory analysis of amino acid synthesis pathway in *Escherichia coli*: aspartate family, *Enzyme and Micorobial Technology* **35**, 694–706.

Kærn, M., Elston, T. C., Blake, W. J. and Collins, J. J.: 2005, Stochasticity in gene expression: from theories to phenotypes, *Nature Genetics* **6**, 451–464.

Kepler, T. B. and Elston, T. C.: 2001, Stochasticity in transcriptional regulation: origins, consequences, and mathematical representations., *Biophys J* **81**(6), 3116–3136.

Kim, D. H., Shreenivasaiah, P. K., Hong, S.-E., Kim, T. and Song, H. K.: 2008, Current research trends in systems biology, *Animal Cells and Systems* **12**, 181–191.

Kitano, H.: 2002, Systems biology: A brief overview, *Science* **295**, 1662–1664.

Ko, M. S.: 1991, A stochastic model for gene induction., *J Theor Biol* **153**(2), 181–194.

Kolcha, W., Calderc, M. and Gilbertc, D.: 2005, When kinases meet mathematics: the systems biology of mapk signalling, *Systems Biology* **579**(8), 1891–1895.

Kotaka, M., Ren, J., Lockyer, M., Hawkins, A. R. and Stammers, D. K.: 2006, Structures of r- and t-state *Escherichia coli* aspartokinase iii: mechanisms of the allosteric transition and inhibition by lysine, *The Journal of Biological Chemistry* **281**(42), 31544–31552.

Liao, H.-H. and Hseu, T.-H.: 1998, Analysis of the regulatory region of the lysc *gene* of *Escherichia coli* via the lacr/o, the tetr/o and arac/i1-i2 regulatory elements, *FEMS Microbiology Letters* **168**, 31–36.

Lipniacki, T., Paszek, P., Marciniak-Czochra, A., Brasier, A. R. and Kimmel, M.: 2006, Transcriptional stochasticity in gene expression., *J Theor Biol* **238**(2), 348–367.

Love, C. A., Lilley, P. E. and Dixon, N. E.: 1996, Stable high-copy-number bacteriophage lambda promoter vectors for overproduction of proteins in *Escherichia coli.*, *Gene* **176**(1-2), 49–53.

Lu, C., Bentley, W. E. and Rao, G.: 2004, A high-throughput approach to promoter study using green fluorescent protein, *Biotechnology Progress* **20**, 1634–1640.

Lutz, R. and Bujard, H.: 1997, Independent and tight regulation of transcriptional units in *Escherichia coli* via the lacr/o, the tetr/o and arac/i1-i2 regulatory elements, *Nucleic Acids Research* **25**(6), 1203–1210.

Maamar, H., Raj, A. and Dubnau, D.: 2007, Noise in gene expression determines cell fate in *Bacillus subtilis*, *Science* **317**, 526–529.

McClure, W. R.: 1980, Rate-limiting steps in rna chain initiation, *PNAS* **77**(10), 5634–5638.

Mettetal, J. T., Muzzey, D., Pedraza, J. M., Ozbudak, E. M. and van Oudenaarden, A.: 2006, Predicting stochastic gene expression dynamics in single cells, *PNAS* **103**, 7304–7309.

Mijakovic, I., Petranovic, D. and Jensen, P. R.: 2005, Tunable promoters in systems biology, *Current Opinion in Biotechnology* **16**, 329–335.

Ou, J., Yamada, T., Nagahisa, K., Hirasawa, T., Furusawa, C., Yomo, T. and Shimizu, H.: 2008, Dynamic change in promoter activation during lysine biosynthesis in *Escherichia coli* cells, *Molecular BioSystems* **4**, 128–134.

Ozbudak, E. M., Thattai, M., Kurtser, I., Grossman, A. D. and van Oudenaarden, A.: 2002, Regulation of noise in the expression of a single gene, *Nature Genetics* **31**, 69–73.

Pedraza, J. M. and van Oudenaarden, A.: 2005, Noise propagation in gene networks, *Science* **307**, 1965–1969.

Pfaffl, M. W.: 2001, A new mathematical model for relative quantification in real-time rt-pcr, *Nucleic Acids Research* **29**(9), 2002–2007.

Pirone, J. R. and Elston, T. C.: 2004, Fluctuations in transcription factor binding can explain the graded and binary responses observed in inducible gene expression., *J Theor Biol* **226**(1), 111–121.

Raser, J. M. and O'Shea, E. K.: 2004, Control of stochasticity in eukaryotic gene expression, *Science* **304**, 1811–1814.

Richaud, C., Richaud, F., Martin, C., Haziza, C. and Patte, J.-C.: 1984, Regulation of espression and nucleotide sequence of the *Escherichia coli dapD* gene, *The Journal of Biological Chemistry* **259**(23), 14824–14828.

Richaud, F., Richaud, C., Ratet, P. and Patte, J.-C.: 1986, Chromosomal location of and nucleotid sequence of the *Escherichia coli dapA* gene, *Journal of Bacteriology* **166**(1), 297–300.

Rodionov, D. A., Vitreschak, A. G., Mironov, A. A. and Gelfand, M. S.: 2003, Regulation of lysine biosynthesis and transport genes in bacteria: yet another rna riboswitch?, *Nucleic Acids Research* **31**(23), 6748–6757.

Roederer, M.: 2002, Compensation in flow cytometry, *Curr Protoc Cytom* **Chapter 1**, Unit 1.14.

Rosenfeld, N., Young, J. W., Alon, U., Swain, P. S. and Elowitz, M. B.: 2005, Gene regulation at the single-cell level, *Science* **307**, 1962–1965.

Sayyed-Ahmad, A., Tuncay, K. and Ortoleva, P. J.: 2007, Transcriptional regulatory network refinement and quantification through kinetic modeling, gene expression microarray data and information theory, *BMC Bioinformatics* **8**, 20.

Shahrezaei, V., Ollivier, J. F. and Swain, P. S.: 2008, Colored extrinsic fluctuations and stochastic gene expression, *Molecular Systems Biology* **4**, 196.

Shahrezaei, V. and Swain, P. S.: 2008, The stochastic nature of biochemical networks, *Current Opinion in Biotechnology* **19**, 369–374.

stragier, P., Borne, F., Richaud, F., Richaud, C. and Patte, J.-C.: 1983, Regulatory pattern of the *Escherichia coli lysA* gene: Expression of chromosomal *lysA-lacZ* fusions, *Journal of Bacteriology* **156**(3), 1198–1203.

Suzuki, M., Ito, Y., Savage, H. E., Husimi, Y. and Douglas, K. T.: 2004, Protease-sensitive signalling by chemically engineered intramolecular fluorescent resonance energy transfer mutants of green fluorescent protein, *Biochimica et Biophysica Acta* **1679**, 222–229.

Wray, G. A., Hahn, M. W., Abouheif, E., Balhoff, J. P., Pizer, M., Rockman, M. V. and Romano, L. A.: 2003, The evolution of transcriptional regulation in eukaryotes, *Molecular Biology and Evolution* **20**(9), 1377–1419.

Zaslaver, A., Mayo, A. E., Rosenberg, R., Bashkin, P., Sberro, H., Tsalyuk, M., Surette, M. G. and Alon, U.: 2004, Just-in-time transcription program in metabolic pathways, *Nature Genetics* **36**, 486–491.

Zhou, D. and Yang, R.: 2006, Global analysis of gene transcription regulation in prokaryotes, *Cellular and Molecular Life Sciences* **63**, 2260–2290.

Appendix A

Appendix: Biological Noise

A.1 What is biological noise?

Traditional population-averaged measurements would summarize the entire histogram by its mean value $\langle P(t) \rangle$ (brackets represent the population average) at time t, however, the single-cell measurements (such as flow cytometry) show that the expression level ($P(t)$) at time t varies from cell to cell. These fluctuations in the amount of protein product are the result of fluctuations in the rates of transcription and translation of its gene. In order to descript the stochastic or nosiy process of gene expression, coefficient of variation, a conception in mathematics and statistics, was employed. In probability theory and statistics, the coefficient of variation (CV) is a normalized measure of dispersion of a probability distribution. It is defined as the ratio of the standard deviation to the mean,

$$\begin{aligned} \mathrm{CV} &= \frac{\mathrm{StandardDeviation}}{\mathrm{ExpectedReturn}} \\ &= \frac{\sqrt{\mathrm{D}(X)}}{\mathrm{E}(X)} \\ &= \frac{\sqrt{\mathrm{E}(X^2) - (\mathrm{E}(X))^2}}{\mathrm{E}(X)} \end{aligned} \tag{A.1}$$

,where $\mathrm{E}(X)$ is expected return and $\mathrm{D}(X)$ is the deviation. A natural and biologically relevant measure of the magnitude of gene expression noise is thus the size of protein fluctuations compared to their mean concentration. Then the noise, $\eta(t)$, is given by

$$\eta^2(t) = \frac{\langle \mathrm{P}(t)^2 \rangle - \langle \mathrm{P}(t) \rangle^2}{\langle \mathrm{P}(t) \rangle^2} \tag{A.2}$$

, where the angled brackets denote an average over the probability distribution of $\mathrm{P}(t)$ at time t.

A.2 What are the commponents of biological noise?

Typically the source of the noise is separate into two ways, intrinsic noise and extrinsic noise. The inherent stochasticity of biochemical processes such as transcription and translation generates "intrinsic" noise, denoted η_{int}, which fundamentally limits the precision of gene regulation. Such stochastic effects are set locally by the gene sequence and the properties of the protein it encodes. In addition, fluctuations in the amounts or states of other cellular components lead indirectly to variation in the expression of a particular gene and thus represent "extrinsic" noise,

denoted η_{ext}. Thus, extrinsic sources of noise arise independently of the gene but act on it. Such stochastic effects are controlled by the concentrations, states, and locations of molecules such as regulatory proteins and polymerases, and so on. The extrinsic noise is global to a single cell but vary from one cell to another.

A.3 How to describe the two types of noise?

To examine the noise for a particular gene across a cell population, let the intrinsic and extrinsic variables for that gene be given by vectors $\vec{I}$ and $\vec{E}$, each of whose components represent a different source of noise. The expression level of the gene in one cell, as measured experimentally, is denoted P_k (with k a cell label). From a snapshot of N genetically identical cells, the P_ks can be averaged to find the moments of the protein distribution. This averaging process is equivalent to

$$\frac{1}{N}\sum_{k=1}^{N} P_k^m \approx \iint P^m(\vec{E},\vec{I})p(\vec{E}\vec{I})\mathrm{d}\vec{E}\mathrm{d}\vec{I} \tag{A.3}$$

Here $p(\vec{E}\vec{I})$ is the probability density function for the intrinsic and extrinsic variables, and $P(\vec{E},\vec{I})$ is the measured expression level for particular values of $\vec{E}$ and $\vec{I}$. Using the product rule of probabilities, this becomes

$$\begin{aligned}\frac{1}{N}\sum_{k=1}^{N} P_k^m &\approx \iint P^m(\vec{E},\vec{I})p(\vec{E}|\vec{I})p(\vec{E})\mathrm{d}\vec{E}\mathrm{d}\vec{I} \\ &= \int \mathrm{d}\vec{E}p(\vec{E}) \int \mathrm{d}\vec{I}P^m(\vec{E},\vec{I})p(\vec{I}|\vec{E})\end{aligned} \tag{A.4}$$

The second integral is an average over the intrinsic variables with the extrinsic variables held fixed and shall be denoted by angled brackets:

$$\langle P^m(\vec{E})\rangle \equiv \int \mathrm{d}\vec{I}P^m(\vec{E},\vec{I})p(\vec{I}|\vec{E}) \tag{A.5}$$

Averages over the extrinsic variables will be indicated with an overbar, so that Equation (A.4) becomes

$$\frac{1}{N}\sum_{k=1}^{N} P_k^m = \overline{\langle P_k^m\rangle} \tag{A.6}$$

That is, an average over both intrinsic and extrinsic noise sources.

Hence, the measured noise, η_{tot}, defined empirically by

$$\eta_{\text{tot}}^2 = \frac{\frac{1}{N}\sum_K P_k^2 - \left(\frac{1}{N}\sum_K P_k\right)^2}{\left(\frac{1}{N}\sum_K P_k\right)^2} \tag{A.7}$$

is equivalent to

$$\eta_{\text{tot}}^2 = \frac{\overline{\langle P^2\rangle} - \left(\overline{\langle P\rangle}\right)^2}{\left(\overline{\langle P\rangle}\right)^2} \tag{A.8}$$

This can be written as

$$\begin{aligned}\eta_{\text{tot}}^2 &= \frac{\overline{\langle P^2\rangle} - \overline{\langle P\rangle^2} + \overline{\langle P\rangle^2} - \left(\overline{\langle P\rangle}\right)^2}{\left(\overline{\langle P\rangle}\right)^2} \\ &= \frac{\overline{\langle P^2\rangle - \langle P\rangle^2} + \overline{\langle P\rangle^2} - \left(\overline{\langle P\rangle}\right)^2}{\left(\overline{\langle P\rangle}\right)^2} \\ &= \frac{\overline{\langle P^2\rangle - \langle P\rangle^2}}{\left(\overline{\langle P\rangle}\right)^2} + \frac{\overline{\langle P\rangle^2} - \left(\overline{\langle P\rangle}\right)^2}{\left(\overline{\langle P\rangle}\right)^2} \\ &\equiv \eta_{\text{int}}^2 + \eta_{\text{ext}}^2\end{aligned} \tag{A.9}$$

Here, the averages over the extrinsic variables are indicated with an overbar. In another word, there are only intrinsic variables indicated by $\overline{\langle P^2\rangle - \langle P\rangle^2} / \left(\overline{\langle P\rangle}\right)^2$. The intrinsic noise, η_{int}, is proportional to the variance of the intrinsic distribution, calculated for a particular value of the extrinsic variables and then averaged over all possible valuses of these variables. And The extrinsic noise, η_{ext}, vanishes as extrinsic distributions become more and more spiked. That is to say, the square of the experimentally measurable noise is a direct sum of the intrinsic, η_{int},and extrinsic, η_{ext}, contributions.

A.4 How to calculate the extrinsic noise and intrinsic noise in dual-fluorescence system?

As reported by M.B. Elowitz et al., intrinsic noise and extrinsic noise can be measured and distinguished with two genes controlled by identical regulatory sequences. Consider what would happen if two identical copies of the gene were present in the same (k^{th}) cell, and their protein products, labeled $P_k^{(1)}$ and $P_k^{(2)}$, were measured simultaneously. These will have different values of the intrinsic variables, but, because both are present in a single cell, they will be exposed to the same intracellular environment and so have the same value of the extrinsic variables. Therefore, by summing their product, we obtain

$$\begin{aligned}\frac{1}{N}\sum_{k=1}^{N} P_k^{(1)} P_k^{(2)} &\approx \iiint P(\vec{E},\vec{I}_1)P(\vec{E},\vec{I}_2)p(\vec{E}\vec{I}_1\vec{I}_2)\mathrm{d}\vec{E}\mathrm{d}\vec{I}_1\mathrm{d}\vec{I}_2 \\ &= \iiint P(\vec{E},\vec{I}_1)P(\vec{E},\vec{I}_2)p(\vec{I}_1\vec{I}_2|\vec{E})p(\vec{E})\mathrm{d}\vec{E}\mathrm{d}\vec{I}_1\mathrm{d}\vec{I}_2 \\ &= \int \mathrm{d}\vec{E}[\int \mathrm{d}\vec{I}P(\vec{E},\vec{I})p(\vec{I}|\vec{E})]^2 \\ &= \overline{\langle P\rangle^2}\end{aligned} \tag{A.10}$$

, precisely the average needed. And similarly we obtain

$$\begin{aligned}\frac{1}{N}\sum_{k=1}^{N}P_k^{(1)}\times\frac{1}{N}\sum_{k=1}^{N}P_k^{(2)} &\approx \iint P(\vec{E},\vec{I}_1)p(\vec{E}\vec{I}_1)\mathrm{d}\vec{E}\mathrm{d}\vec{I}_1\times\iint P(\vec{E},\vec{I}_2)p(\vec{E}\vec{I}_2)\mathrm{d}\vec{E}\mathrm{d}\vec{I}_2\\ &=[\iint P(\vec{E},\vec{I})p(\vec{E}\vec{I})\mathrm{d}\vec{E}\mathrm{d}\vec{I}]^2\\ &=\left(\overline{\langle P\rangle}\right)^2\end{aligned} \tag{A.11}$$

$$\begin{aligned}\left(\frac{1}{N}\sum_{k=1}^{N}P_k\right)^2 &\approx\left(\iint P(\vec{E},\vec{I})p(\vec{E}\vec{I})\mathrm{d}\vec{E}\mathrm{d}\vec{I}\right)^2\\ &=\left(\overline{\langle P\rangle}\right)^2\end{aligned} \tag{A.12}$$

$$\begin{aligned}\frac{1}{N}\sum_{k=1}^{N}(P_k)^2 &\approx\iint\left(P(\vec{E},\vec{I})\right)^2p(\vec{E}\vec{I})\mathrm{d}\vec{E}\mathrm{d}\vec{I}\\ &=\int\mathrm{d}\vec{E}\int\mathrm{d}\vec{I}\left(P(\vec{E},\vec{I})\right)^2p(\vec{I}|\vec{E})\\ &=\overline{\langle P^2\rangle}\end{aligned} \tag{A.13}$$

Experimentally, two distinguishable variants of fluorescent protein, corresponding to $P^{((1))}$ and $P^{((2))}$, would allow estimation of $\overline{\langle P\rangle^2}$ and $\left(\overline{\langle P\rangle}\right)^2$. We can calculate the extrinsic noise (Equation (A.9)) shared by the two protein by

$$\begin{aligned}\eta_{\text{ext}}^2 &\equiv\frac{\overline{\langle P\rangle^2}-\left(\overline{\langle P\rangle}\right)^2}{\left(\overline{\langle P\rangle}\right)^2}\\ &=\frac{\frac{1}{N}\sum_{k=1}^{N}P_k^{(1)}P_k^{(2)}-\frac{1}{N}\sum_{k=1}^{N}P_k^{(1)}\times\frac{1}{N}\sum_{k=1}^{N}P_k^{(2)}}{\frac{1}{N}\sum_{k=1}^{N}P_k^{(1)}\times\frac{1}{N}\sum_{k=1}^{N}P_k^{(2)}}\end{aligned} \tag{A.14}$$

In dual-fluorescence systems, there are two distinguishable fluorescent proteins, GFP and RFP. The two proteins are products of two fluorescent protein genes, *rfp*-T4 and *gfp*uv5, which are obtained from the plasmids pPROLar-GFPuv5 and pQE31-T4 separately. The *rfp*-T4 is controlled by a strong constitutive promoter $\mathrm{P_LP_R}$ and terminated by t0 from pCL476, where the *gfp*uv5 is controlled by the target promoter in which we are interested. If the $\mathrm{i^{th}}$ element of vectors r and g contain the average RFP or GFP intensity, respectively, of the $\mathrm{i^{th}}$ cell in the sample and angled brackets denote means over the cell population, the extrinsic noise (Equation (A.14)) in the dual-fluorescence systems can be rewritten as

$$\eta_{\text{ext}}^2=\frac{\langle rg\rangle-\langle r\rangle\langle g\rangle}{\langle r\rangle\langle g\rangle} \tag{A.15}$$

, and as the definition of noise (Equation (A.7)), the total noise for target promoter can be obtained by

$$\eta_{\text{tot}}^2=\frac{\langle g^2\rangle-\langle g\rangle^2}{\langle g\rangle^2} \tag{A.16}$$

The intrinsic noise of target promoter can be calculated by subtracting extrinsic noise from the total noise.

$$\eta_{\text{int}}^2 = \eta_{\text{tot}}^2 - \eta_{\text{ext}}^2 \tag{A.17}$$

Appendix B

Appendix: Source code

B.1 FCSGettingPeakOu

B.1.1 getPeak.cpp

```
#include <origin.h>
#include <page.h>
#include <wksheet.h>
#include <data.h>
#include <graph.h>
#include <NAG\OCN_g01.h>
#include <math.h>
#include <string.h>
#define h0 0.4829629131445341
#define h1 0.8365163037378079
#define h2 0.2241438680420134
#define h3 -0.1294095225512604
string o_fcounts()
{
        //creat a new worksheet to store all the LOG GFP
        Worksheet wks;
        wks.Create();
        string wksName="fGFP";
        wks.GetPage().Rename(wksName);
        string bName;
        int colNum=2;
        string total=",";//to set a counter for killing the noise dataset.
        Project prj;
        PageBase pg;
        Collection<PageBase>pgcoll;
        foreach (pg in pgcoll)
        {
            bName = pg.GetName();
            total=total+","+bName;
            Worksheet o_wks(bName);
            wks.AddCol("co"+bName);
            Dataset dsx;
            if(dsx.Attach(bName, 4))
            {
                Dataset dss;
              if (dss.Attach(wks.GetPage().GetName(), colNum))
              {
                dss=dsx;
                        colNum++;
            }
        }
    }
}
//creat column to store xpeak, hWidth, and area.
wks.AddCol("xc");
wks.AddCol("mean");
wks.AddCol("hWidth");
wks.AddCol("sd2");
wks.AddCol("area");
wks.AddCol("gzero");
wks.AddCol("cv");
wks.AddCol("cv1");
//create graphpage to draw the raw data.
GraphPage r_grph;
r_grph.Create(NULL, CREATE_VISIBLE_SAME);
r_grph.Rename("rdata");

int timer;
int ytemp=0;
string cName;
//      int flag=0;
//      if(wks.GetNumCols()<25)
//              flag=1;

for (int i=0;i<wks.GetNumCols()-11;i++)
{
        Worksheet c_nwks;
        c_nwks.Create();
        timer=i*6;
//      if(i<7)
//              timer=i*10+i*5*flag;
//      else
//              timer=(i-6)*15+6*10+6*5*flag;
        cName=wksName+timer;
        c_nwks.GetPage().Rename(cName);
        Dataset dsRaw(wksName,i+2);
        int rowNum=dsRaw.GetSize();

        Dataset idxData(cName,0);
        Dataset freqData(cName,1);
        double  xmax, xmin;
        int sucess;
        int num_class = 1024;
        int iclass = 0;
        int n = rowNum;
```

```
                Nag_ClassBoundary iclass_enum;
                iclass_enum = Nag_ClassBoundaryUser;
                vector<double>a = dsRaw;
                vector<double>c;
                c.SetSize(1024);
                for (int z=0;z<1024;z++){c[z]=z+1;}
                vector<int>ifreq;
                ifreq.SetSize(1024);
                sucess = nag_frequency_table(n, a, num_class, iclass_enum, c, ifreq,
                      &xmin, &xmax );
                //write the result back to the worksheets.
                freqData = ifreq;
                for (z=0;z<1024;z++){c[z]=z;}
                idxData = c;
                //plot the fGFP
                ytemp=o_plot(cName,timer,ytemp);
                //copy peak hWidth and area to worksheet fGFP.
                cdata(cName,i,timer);
        }

        return total;
}

int o_plot(string sName,int i,int ytemp)
{
                string sA=sName+"_A";
                string sB=sName+"_B";
                string sC=sName+"_Result";
                Worksheet wks(sName);
                Dataset sdata(sB);
                string timer="raw"+i+"min";
                int raw=wks.AddCol(timer);
                Dataset daRaw(sName,raw);
                daRaw=sdata;
                //do wavelet smooth.
                daub(sB);
                daub(sB);
                invDaub(sB);
                invDaub(sB);
                wks.AddCol("Result");//Create a column for results
                //do FFT smooth
                LT_execute("curve.reset()");  //initialize
                LT_execute("curve.data\$_=_"+sB);
                LT_execute("curve.result\$_=_"+sC);
                LT_execute("curve.i1_=_10");
                LT_execute("curve.i2_=_"+sdata.GetSize());
                LT_execute("curve.smoothpts_=_25");
                LT_execute("curve.derivdeg_=_1");
                LT_execute("curve.FFTSmooth()");

                int ymax=f_peak(sName);

                //draw the similation graph.
//              GraphPage       grph;
//              grph.Create(NULL, CREATE_VISIBLE_SAME);
//              grph.Rename("gr"+sName);
                Curve cc(sA, sName+"_"+timer);
//              Curve dd(sA, sC);
//              lay.AddPlot(cc, IDM_PLOT_LINE);
//              int     nPlot = lay.AddPlot(dd, IDM_PLOT_LINE);
//              lay.DataPlots(nPlot).SetColorRGB(0xFF,0,0);// make a red curve
//              resetY(ymax,"gr"+sName);

                GraphLayer rawlay("rdata");
                int     rPlot = rawlay.AddPlot(cc, IDM_PLOT_LINE);
                if (ymax>ytemp)
                        resetY(ymax,"rdata");
                else
                        ymax=ytemp;
                return ymax;
}

void daub(string sB)
{
        Dataset y(sB);
        int n=y.GetSize();
        vector<double>a;
        a.SetSize(n);
                int i, j;
         int half = n/2;
         vector<double> tmp;
         tmp.SetSize(n);
        a=y;
        a[0]=a[1];
         for (i = 0, j = 0; j < n-3; j += 2, i++) {
            tmp[i]      = a[j]*h0 + a[j+1]*h1 + a[j+2]*h2 + a[j+3]*h3;
//            tmp[i+half] = a[j]*h3 - a[j+1]*h2 + a[j+2]*h1 - a[j+3]*h0;
                        tmp[i+half] =0;
         }

         tmp[i]      = a[n-2]*h0 + a[n-1]*h1 + a[0]*h2 + a[1]*h3;
//          tmp[i+half] = a[n-2]*h3 - a[n-1]*h2 + a[0]*h1 - a[1]*h0;
                tmp[i+half] =0;
        y=tmp;
}

void invDaub(string sB) {
        Dataset y(sB);
    int n=y.GetSize();
    vector<double> a;
    a.SetSize(n);
        int i, j;
        int half = n/2;
        vector<double> tmp;
        tmp.SetSize(n);
    a=y;
        tmp[1]= a[half-1]*h2 + a[n-1]*h1 + a[0]*h0 + a[half]*h3;
        tmp[2]= a[half-1]*h3 - a[n-1]*h0 + a[0]*h1 - a[half]*h2;
        for (i = 0, j = 2; i < half-1; i++) {
           tmp[j++] = a[i]*h2 + a[i+half]*h1 + a[i+1]*h0 + a[i+half+1]*h3;
           tmp[j++] = a[i]*h3 - a[i+half]*h0 + a[i+1]*h1 - a[i+half+1]*h2;
        }
    y=tmp;
}

int f_peak(string sName)
{
                Worksheet wks(sName);
                wks.AddCol("Baseline");
                wks.AddCol("IntRes");
                wks.AddCol("Peak");

                BOOL bErr;
                Curve cvMyCurve( sName+"_A", sName+"_Result" );          // Create
                        curve object to integrate
                Curve cvMyBaseline( sName+"_A", sName+"_Baseline" );     // Create
                        curve object of integration baseline
                IntegrationResult irMyResults;                           // OriginC
                        structure to store integration results
```

```
                Dataset dsCumIntRes( sName+"_IntRes" );          // Cumulative
                    integration result
                dsCumIntRes.SetSize( cvMyCurve.GetSize() );              // Set size
                    of dsCumIntRes to size of cvMyCurve
                bErr = Curve_integrate( &cvMyCurve, &irMyResults, &cvMyBaseline, &
                    dsCumIntRes, TRUE ); // Perform integration
                Dataset res(sName+"_Peak");

                int iSize;
                Dataset dsInd(wks,0);
                Dataset dsRaw(wks,2);
                Dataset dsDataIn(wks,4);
                dsDataIn=dsRaw*dsInd;
                iSize=dsDataIn.GetSize();
                double sum1=0;
                double sum2=0;
                for (int i=0;i<iSize;i++) { sum1=sum1+dsDataIn[i];}
                for (i=1;i<iSize;i++) { sum2=sum2+dsRaw[i];}
                BasicStats bsStat;
                bsStat.mean=sum1/sum2;
                double sigma=0;
                double count=0;
                for (i=1;i<iSize;i++)
                {
                        sigma=dsInd[i]-bsStat.mean;
                        sigma=sigma*sigma;
                        sigma=dsRaw[i]*sigma;
                        count=count+sigma;
                }
                bsStat.sd=count/sum2;

                vector<double>a;
                a.SetSize(7);
                a[0]=irMyResults.xPeak;
                a[1]=irMyResults.yPeak;
                a[2]=irMyResults.Area;
                a[3]=irMyResults.dxPeak;
                a[4]=wks.Cell(0,2);
                a[5]=bsStat.mean;
                a[6]=bsStat.sd;
                res=a;

                int y=(int)irMyResults.yPeak;
                return y;
}

void resetY(int y_value, string gName)
{
                GraphLayer glay(gName);
                string str;
                str.Format("layer.x.from=\%d;layer.x.to=\%d;", 0, 1024);
                glay.LT_execute(str);
                str.Format("layer.y.from=\%d;layer.y.to=\%d;", 0, y_value+50);
                glay.LT_execute(str);
}

void cdata (string sName, int i, int timer)
{
        Dataset sData(sName+"_peak");
        Worksheet r_wks("fGFP");
        int Col=r_wks.GetNumCols()-8;
        r_wks.SetCell(i, 0, timer);
        r_wks.SetCell(i, Col++, sData[0]);
        r_wks.SetCell(i, Col++, sData[5]);
        r_wks.SetCell(i, Col++, sData[3]);
        r_wks.SetCell(i, Col++, sData[6]);
        r_wks.SetCell(i, Col++, sData[2]);
        r_wks.SetCell(i, Col++, sData[4]);
        r_wks.SetCell(i, Col++, sData[3]/sData[0]);
        r_wks.SetCell(i, Col, sData[6]/(sData[5]*sData[5]));
}

void plot_peak_width()
{
        GraphPage       grph;
        grph.Create(NULL, CREATE_VISIBLE_SAME);
        grph.Rename("PeakWidth");
        GraphLayer      lay(grph.GetName());
        string sA="fGFP_A";
        string sB="fGFP_xc";
        string sC="fGFP_hWidth";
        Curve   cc(sA, sB);
        Curve   dd(sA, sC);
        lay.AddPlot(cc, IDM_PLOT_SCATTER);
        int     nPlot = lay.AddPlot(dd, IDM_PLOT_LINE);
        lay.DataPlots(nPlot).SetColorRGB(0xFF,0,0);
        string str;
        str.Format("layer.x.from=\%d;layer.x.to=\%d;", -5, 185);
        lay.LT_execute(str);
        str.Format("layer.y.from=\%d;layer.y.to=\%d;", 100,1000);
        lay.LT_execute(str);
        lay.LT_execute("layer.y.type=2;");
}

void plot_cv_zero(string gName, int flag)
{
        GraphPage       grph;
        grph.Create(NULL, CREATE_VISIBLE_SAME);
        grph.Rename(gName);
        GraphLayer      lay(gName);
        string sA="fGFP_A";
        string sB="fGFP_"+gName;
        Curve   cc(sA, sB);
        lay.AddPlot(cc, IDM_PLOT_SCATTER);
        string str;
        str.Format("layer.x.from=\%d;layer.x.to=\%d;", -5, 185);
        lay.LT_execute(str);
        if(flag)
        {
                str.Format("layer.y.from=\%d;layer.y.to=\%d;", 0,1);
        }
        else
                str.Format("layer.y.from=\%d;layer.y.to=\%d;", 0,6000);
        lay.LT_execute(str);
}

void getstart()
{
                string sName="fGFP";
                Worksheet wks(sName);
                Dataset res(sName+"_xc");
                Dataset tag(sName+"_B");
                int n=res.GetSize();
                wks.AddCol("FFTres");

                //do FFT smooth
                LT_execute("curve.reset()");  //initialize
                LT_execute("curve.data\$=_"+sName+"_xc");
                LT_execute("curve.result\$=_"+sName+"_FFTres");
                LT_execute("curve.i1_=_0");
```

```
                LT_execute("curve.i2 = "+n);
                LT_execute("curve.smoothpts = 3");
                LT_execute("curve.derivdeg = 1");
                LT_execute("curve.FFTSmooth()");

                Dataset sdata(sName+"_FFTres");
                vector<double>a;
                a.SetSize(n);
                a[0]=0;
                for (int i=0;i<n-2;i++)
                {
                        a[i+1]=sdata[i+1]-sdata[i];
                }
                tag=a;

                GraphPage grph();
                grph.Create(NULL, CREATE_VISIBLE_SAME);
                grph.Rename("gfpshift");
                GraphLayer lay("gfpshift");
                Curve cc("fGFP_A","fGFP_B");
                lay.AddPlot(cc, IDM_PLOT_SCATTER);
                lay.LT_execute("Rescale;");
}

o_fcm()
{
        string gName="fcm";
        Project prj;
        PageBase pg;
        Collection<PageBase>pgcoll;
        foreach (pg in pgcoll)
        {
                pg.Destroy();
        }

        LT_execute("run.file(openExcel)");
        string nn=o_fcounts();
        plot_peak_width();
        plot_cv_zero("cv",1);
        plot_cv_zero("gzero",0);
        getstart();

        Page pp = Application.Pages(1);
        string strFile=pp.Label;
        if(strFile.IsFile())
        {
                string strFolder=GetFilePath(strFile);
                string strName=strFolder;
                strName.TrimRight("\\");
                string strName1=strName.Right(2);
                string con1=strName1.Left(1);
                string con2=strName1.Right(1);
                string from, to;
                if(con1.Replace('1','2'))
                        from="0.3";
                else from=con1;
                if(con2.Replace('1','2'))
                        to="0.3";
                else to=con2;
                strName.TrimRight("\\"+strName1);
                string strName2=strName.Right(4);
                string rawlabel=strName2+"_lysine_from_"+from+"mM_to_"+to+"mM";

                GraphLayer      lay("rdata");
                lay.GroupPlots(0, lay.DataPlots.Count());
                lay.LT_execute("lab -d 800 400 ("+rawlabel+")");
                lay.LT_execute("legend");
//              Project.Save(strFolder+gName+".opj");
                out_str("save "+strFolder+gName+".opj");
//              Project.LT_execute("save "+strFolder+gName+".opj");
        }

        //kill the source data because it is too large.
        for(int i=2;i<nn.GetNumTokens(',')-1;i++)
        {
                string pgName=nn.GetToken(i,',');
                Worksheet wks(pgName);
                wks.Destroy();
        }
}
```

B.1.2 openExcel.ogs

```
fdlog.ShowComment = 0;
fdlog.UseGroup(Ascii);
fdlog.UseType(XLS);
if (fdlog.MultiOpen() != 0/0)
{
    for (ii=fdlog.MultiOpen.Count; ii>=1; ii--)
    {
        win -t data Origin;
        FDlog.Get(A, ii);
        open -w %A;
    }
}
```

B.2 FCSMultiOu

B.2.1 fcmdata.h

```
#ifndef FCMDATA_H
#define FCMDATA_H
class fcmData
{
public:
        unsigned int fs;
        unsigned int ss;
        unsigned int fslog;
        unsigned int sslog;
        unsigned int gfplog;
        unsigned int gfplin;
        unsigned int rfplin;
        unsigned int rfplog;
        int flag;
        float gfp;
        float rfp;
        fcmData()
        {
                flag=1;
                gfp=0;
                rfp=0;
        }
};

#endif
```

B.2.2 fcm.h

```
#ifndef FCM_H_H
#define FCM_H_H
#pragma once

#include "fcmdata.h"
#include <math.h>

class fcm
{
public:
        fcmData *fcmdata;
        int size;
        int redEvents[1024];
        int greenEvents[1024];
        int sslogEvents[1024];
        int fslogEvents[1024];
        int i;

        fcm(void)
        {
        }

        fcm(int dataSize, unsigned int *datalist)
        {
                int j;
                size=dataSize;
                fcmdata=new fcmData[dataSize/14];
                for(i=0,j=0;i<dataSize/16;i++)
                {
                        fcmdata[i].fs=datalist[j++];
                        fcmdata[i].ss=datalist[j++];
                        fcmdata[i].fslog=datalist[j++];
                        fcmdata[i].sslog=datalist[j++];
                        fcmdata[i].gfplog=datalist[j++];
                        fcmdata[i].gfplin=datalist[j++];
                        fcmdata[i].rfplin=datalist[j++];
                        fcmdata[i].rfplog=datalist[j++];
                }
        }

        int getTotalEvents()
        {
                int events=0;
                for(i=0;i<size/16;i++)
                {
                        if(fcmdata[i].flag)
                                events++;
                }
                return events;
        }

        void redCount(int step)
        {
                for(i=0;i<1024;i++)
                {redEvents[i]=0;}
                for(i=0;i<size/16;i++)
                {
                        if(fcmdata[i].flag)
                        {
                                redEvents[(int)(fcmdata[i].rfplog)]++;
                        }
                }
                for(int i=0;i<(1024/step);i++)
                {
                        int events=0;
                        for(int j=0; j<step; j++)
                        {
                                events+=redEvents[step*i+j];
                        }
                        redEvents[i]=events;
                }
        }

        void greenCount(int step)
        {
                for(i=0;i<1024;i++)
                {greenEvents[i]=0;}
                for(i=0;i<size/16;i++)
                {
                        if(fcmdata[i].flag)
                        {
                                greenEvents[(int)(fcmdata[i].gfplog)]++;
                        }
                }
                for(int i=0;i<(1024/step);i++)
                {
                        int events=0;
                        for(int j=0; j<step; j++)
                        {
                                events+=greenEvents[step*i+j];
                        }
                        greenEvents[i]=events;
                }
        }

        void sslogCount(int step)
        {
                for(i=0;i<1024;i++)
                {sslogEvents[i]=0;}
                for(i=0;i<size/16;i++)
                {
                        if(fcmdata[i].flag)
                        {
                                sslogEvents[(int)(fcmdata[i].sslog)]++;
                        }
                }
                for(int i=0;i<(1024/step);i++)
                {
                        int events=0;
                        for(int j=0; j<step; j++)
                        {
                                events+=sslogEvents[step*i+j];
                        }
                        sslogEvents[i]=events;
                }
        }

        void fslogCount(int step)
        {
                for(i=0;i<1024;i++)
                {fslogEvents[i]=0;}
                for(i=0;i<size/16;i++)
                {
                        if(fcmdata[i].flag)
                        {
                                fslogEvents[(int)(fcmdata[i].fslog)]++;
                        }
                }
```

```
        for(int i=0;i<(1024/step);i++)
        {
                int events=0;
                for(int j=0; j<step; j++)
                {
                        events+=fslogEvents[step*i+j];
                }
                fslogEvents[i]=events;
        }
}

int getFSPeak()
{
        int peak=0;
        int peakV=0;
        fslogCount(1);
        for(i=0;i<1000;i++)
        {
                if(peakV<fslogEvents[i])
                {
                        peakV=fslogEvents[i];
                        peak=i;
                }
        }
        return peak;
}

int getSSPeak()
{
        int peak=0;
        int peakV=0;
        sslogCount(1);
        for(i=0;i<1000;i++)
        {
                if(peakV<sslogEvents[i])
                {
                        peakV=sslogEvents[i];
                        peak=i;
                }
        }
        return peak;
}

float meanred()
{
        float sum=0;
        float meanred=0;
        int counter=0;
        for(i=0;i<size/16;i++)
        {
                if(fcmdata[i].flag)
                {
                        sum+=fcmdata[i].rfp;
                        counter++;
                }
        }
        if(counter)     meanred=sum/counter;
        return meanred;
}

float meangreen()
{
        float sum=0;
        float meangreen=0;
        int counter=0;
        for(i=0;i<size/16;i++)
        {
                if(fcmdata[i].flag)
                {
                        sum+=fcmdata[i].gfp;
                        counter++;
                }
        }
        if(counter) meangreen=sum/counter;
        return meangreen;
}
float meandif()
{
        float sum=0;
        float meandif=0;
        int counter=0;
        for(i=0;i<size/16;i++)
        {
                if(fcmdata[i].flag)
                {
                        sum+=(fcmdata[i].rfp-fcmdata[i].gfp)*(fcmdata[i].rfp
                                -fcmdata[i].gfp);
                        counter++;
                }
        }
        if(counter) meandif=sum/counter;
        return meandif;
}
float meansqr()
{
        float sum=0;
        float meansqr=0;
        int counter=0;
        for(i=0;i<size/16;i++)
        {
                if(fcmdata[i].flag)
                {
                        sum+=(fcmdata[i].rfp*fcmdata[i].rfp)+(fcmdata[i].gfp
                                *fcmdata[i].gfp);
                        counter++;
                }
        }
        if(counter) meansqr=sum/counter;
        return meansqr;
}
float redmeansqr()
{
        float sum=0;
        float meansqr=0;
        int counter=0;
        for(i=0;i<size/16;i++)
        {
                if(fcmdata[i].flag)
                {
                        sum+=(fcmdata[i].rfp*fcmdata[i].rfp);
                        counter++;
                }
        }
        if(counter) meansqr=sum/counter;
        return meansqr;
}
float greenmeansqr()
{
        float sum=0;
        float meansqr=0;
```

```
        int counter=0;
        for(i=0;i<size/16;i++)
        {
                if(fcmdata[i].flag)
                {
                        sum+=(fcmdata[i].gfp*fcmdata[i].gfp);
                        counter++;
                }
        }
        if(counter) meansqr=sum/counter;
        return meansqr;
}
float meanmul()
{
        float sum=0;
        float meanmul=0;
        int counter=0;
        for(i=0;i<size/16;i++)
        {
                if(fcmdata[i].flag)
                {
                        sum+=fcmdata[i].rfp*fcmdata[i].gfp;
                        counter++;
                }
        }
        if(counter) meanmul=sum/counter;
        return meanmul;
}

float intNoise()
{
        float intNoise;
        float meandif;
        float meanred;
        float meangreen;
        meandif=fcm::meandif();
        meanred=fcm::meanred();
        meangreen=fcm::meangreen();
        return intNoise=meandif/(2*meanred*meangreen);
}

float extNoise()
{
        float extNoise;
        float meanmul;
        float meanred;
        float meangreen;
        meanmul=fcm::meanmul();
        meanred=fcm::meanred();
        meangreen=fcm::meangreen();
        return extNoise=(meanmul-meanred*meangreen)/(meanred*meangreen);
}

float total()
{
        float totalNoise;
        float meansqr;
        float meanred;
        float meangreen;
        meansqr=fcm::meansqr();
        meanred=fcm::meanred();
        meangreen=fcm::meangreen();
        return totalNoise=(meansqr-2*meanred*meangreen)/(2*meanred*meangreen
              );
}

float correlation()
{
        float correlation;
        float meangreen;
        float meanred;
        float meanmul;
        meangreen=fcm::meangreen();
        meanred=fcm::meanred();
        meanmul=fcm::meanmul();
        return correlation=(meanmul-meangreen*meanred)/(meangreen*meanred);
}
float correlationred()
{
        float correlation;
        float meanred;
        float redmeansqr;
        meanred=fcm::meanred();
        redmeansqr=fcm::redmeansqr();
        return correlation=(redmeansqr-meanred*meanred)/(meanred*meanred);
}
float correlationgreen()
{
        float correlation;
        float meangreen;
        float greenmeansqr;
        meangreen=fcm::meangreen();
        greenmeansqr=fcm::greenmeansqr();
        return correlation=(greenmeansqr-meangreen*meangreen)/(meangreen*
              meangreen);
}

float g_intNoise()
{
        float g_intNoise;
        float correlationgreen;
        float correlation;
        correlation=fcm::correlation();
        correlationgreen=fcm::correlationgreen();
        return g_intNoise=correlationgreen-correlation;
}

float meanfslog()
{
        float sum=0;
        float meanfslog=0;
        for(i=0;i<size/16;i++)
        {
                sum+=fcmdata[i].fslog;
        }
        meanfslog=(sum*16)/size;
        return meanfslog;
}
float meansslog()
{
        float sum=0;
        float meansslog=0;
        for(i=0;i<size/16;i++)
        {
                sum+=fcmdata[i].sslog;
        }
        meansslog=(sum*16)/size;
        return meansslog;
}
float meangfplog()
```

```
{
        float sum=0;
        float meangfplog=0;
        for(i=0;i<size/16;i++)
        {
                sum+=fcmdata[i].gfplog;
        }
        meangfplog=(sum*16)/size;
        return meangfplog;
}
float meanrfplog()
{
        float sum=0;
        float meanrfplog=0;
        for(i=0;i<size/16;i++)
        {
                sum+=fcmdata[i].rfplog;
        }
        meanrfplog=(sum*16)/size;
        return meanrfplog;
}
float variancefslog()
{
        float meanfslog=0;
        float variancefslog=0;
        meanfslog=fcm::meanfslog();
        for(i=0;i<size/16;i++)
        {
                variancefslog+=(fcmdata[i].fslog-meanfslog)*(fcmdata[i].
                        fslog-meanfslog);
        }
        variancefslog=(variancefslog*16)/size;
        if (variancefslog>=0) variancefslog=sqrt(variancefslog);
        return variancefslog;
}
float variancesslog()
{
        float meansslog=0;
        float variancesslog=0;
        meansslog=fcm::meansslog();
        for(i=0;i<size/16;i++)
        {
                variancesslog+=(fcmdata[i].sslog-meansslog)*(fcmdata[i].
                        sslog-meansslog);
        }
        variancesslog=(variancesslog*16)/size;
        if (variancesslog>=0) variancesslog=sqrt(variancesslog);
        return variancesslog;
}
float variancegfplog()
{
        float meangfplog=0;
        float variancegfplog=0;
        meangfplog=fcm::meangfplog();
        for(i=0;i<size/16;i++)
        {
                variancegfplog+=(fcmdata[i].gfplog-meangfplog)*(fcmdata[i].
                        gfplog-meangfplog);
        }
        variancegfplog=(variancegfplog*16)/size;
        if (variancegfplog>=0) variancegfplog=sqrt(variancegfplog);
        return variancegfplog;
}
float variancerfplog()
{
        float meanrfplog=0;
        float variancerfplog=0;
        meanrfplog=fcm::meanrfplog();
        for(i=0;i<size/16;i++)
        {
                variancerfplog+=(fcmdata[i].rfplog-meanrfplog)*(fcmdata[i].
                        rfplog-meanrfplog);
        }
        variancerfplog=(variancerfplog*16)/size;
        if (variancerfplog>=0) variancerfplog=sqrt(variancerfplog);
        return variancerfplog;
}
void resetFilter()
{
        for(i=0;i<size/16;i++)
        {
                fcmdata[i].flag=1;
        }
}

float expmeanmul()
{
        float meanred;
        float meangreen;
        float expmeanmul;
        meanred=fcm::meanred();
        meangreen=fcm::meangreen();
        return expmeanmul=meanred*meangreen;
}
float parCorrelation()
{
        float meanmul;
        float meanred;
        float meangreen;
        float greenmeansqr;
        float redmeansqr;
        float greenstd;
        float redstd;
        float parCorrelation;
        meanmul=fcm::meanmul();
        meangreen=fcm::meangreen();
        meanred=fcm::meanred();
        greenmeansqr=fcm::greenmeansqr();
        redmeansqr=fcm::redmeansqr();
        greenstd=greenmeansqr-meangreen*meangreen;
        if (greenstd>=0) greenstd=sqrt(greenstd);
        redstd=redmeansqr-meanred*meanred;
        if (redstd>=0) redstd=sqrt(redstd);
        if (greenstd>=0&&redstd>=0)
        {
                return parCorrelation=(meanmul-meanred*meangreen)/(redstd*
                        greenstd);
        }
        else
        {
                return parCorrelation=0;
        }
}

void fluoresenceConvert(bool convert=0, bool o=0, int fslogCenter=415, int
     sslogCenter=636, bool yamada=1)
{
        int fslog=0;
        float a=0;
        if(!convert)
```

```
{
        for(i=0;i<size/16;i++)
        {
                fcmdata[i].gfp=(float)fcmdata[i].gfplog;
                fcmdata[i].rfp=(float)fcmdata[i].rfplog;
        }
}
else
{
        if(yamada)
        {
                for(i=0;i<size/16;i++)
                {
                        if(fcmdata[i].gfplog>0&&fcmdata[i].rfplog>0)
                        {
                                fcmdata[i].gfp=(float)43.177*exp
                                        (0.009*fcmdata[i].gfplog);
                                fcmdata[i].rfp=(float)43.177*exp
                                        (0.009*fcmdata[i].rfplog);
                        }
                        else
                        {fcmdata[i].flag=0;}
                }
        }
        else
        {
                for(i=0;i<size/16;i++)
                {
                        a=(float)fcmdata[i].gfplog/1024;
                        a=a*4;
                        a-=1;
                        fcmdata[i].gfp=a;
                        fcmdata[i].gfp=(float)pow(10,fcmdata[i].gfp)
                                ;
                        fcmdata[i].gfp=fcmdata[i].gfp*1.024;
                        a=(float)fcmdata[i].rfplog/1024;
                        a=a*4;
                        a-=1;
                        fcmdata[i].rfp=a;
                        fcmdata[i].rfp=(float)pow(10,fcmdata[i].rfp)
                                ;
                        fcmdata[i].rfp=fcmdata[i].rfp*1.024;

                        if(o)
                        {
                                fslog=fcmdata[i].fslog-fslogCenter;
                                if (0<fslog)
                                {
                                        for (int j=0; j<fslog; j++)
                                        {
                                                fcmdata[i].gfp=
                                                        fcmdata[i].gfp
                                                        /2;
                                                fcmdata[i].rfp=
                                                        fcmdata[i].rfp
                                                        /2;
                                        }
                                }
                                else
                                {
                                        if (0>fslog)
                                        {
                                                fcmdata[i].gfp=
                                                        fcmdata[i].gfp
                                                        *2;
                                                fcmdata[i].rfp=
                                                        fcmdata[i].rfp
                                                        *2;
                                        }
                                }
                        }
                }
        }

        void rFilter(float r=12, int fslogCenter=415, int sslogCenter=636)
        {
                //how to set the center?fslog=414.787466,sslog=636.196521
                float radius;
                for(i=0;i<size/16;i++)
                {
                        if(fcmdata[i].flag)
                        {
                                radius=(fcmdata[i].fslog-fslogCenter)*(fcmdata[i].
                                        fslog-fslogCenter)
                                        +(fcmdata[i].sslog-sslogCenter)*(fcmdata[i].
                                                sslog-sslogCenter);
                                radius=sqrt(radius);
                                if(radius>r)
                                {fcmdata[i].flag=0;}
                        }
                }
        }

        ~fcm(void)
        {
        }
};
#endif
```

B.2.3 FCS2View.h

```
// FCS2View.h : interface of the CFCS2View class
//
/////////////////////////////////////////////////////////////////////////////

#pragma once
#include "fcm.h"

class CFCS2View : public CWindowImpl<CFCS2View, CRichEditCtrl>
{
public:
        unsigned int *rawdata;
        CString outPut;
        CString forSave;
        CString CounterGFPOutPut;
        CString CounterRFPOutPut;
        CString CounterFSOutPut;
        CString CounterSSOutPut;
        fcm *fcm1;
        int m_wflag;

        DECLARE_WND_SUPERCLASS(NULL, CRichEditCtrl::GetWndClassName())

        CFCS2View()
        {
                m_wflag=0;
```

```
                forSave="";
                outPut="";
                CounterGFPOutPut="";
                CounterRFPOutPut="";
                CounterFSOutPut="";
                CounterSSOutPut="";
        }

        BOOL PreTranslateMessage(MSG* pMsg);

        BEGIN_MSG_MAP(CFCS2View)
                MESSAGE_HANDLER(WM_RBUTTONDOWN, OnRButtonDown)
                COMMAND_ID_HANDLER(ID_EDIT_CUT, OnEditCut)
                COMMAND_ID_HANDLER(ID_EDIT_COPY, OnEditCopy)
                COMMAND_ID_HANDLER(ID_EDIT_PASTE, OnEditPaste)
                COMMAND_ID_HANDLER(IDM_TEST, OnTest)
        ALT_MSG_MAP(1)
                COMMAND_ID_HANDLER(ID_FILE_OPEN, OnFileOpen)
                COMMAND_ID_HANDLER(ID_FILE_SAVE, OnFileSave)
                COMMAND_ID_HANDLER(ID_EDIT_CUT, OnEditCut)
                COMMAND_ID_HANDLER(ID_EDIT_COPY, OnEditCopy)
                COMMAND_ID_HANDLER(ID_EDIT_PASTE, OnEditPaste)
                COMMAND_ID_HANDLER(ID_EDIT_UNDO, OnEditUndo)
        END_MSG_MAP()

// Handler prototypes (uncomment arguments if needed):
//       LRESULT MessageHandler(UINT /*uMsg*/, WPARAM /*wParam*/, LPARAM /*lParam*/,     //
      BOOL& /*bHandled*/)                                                                //
//       LRESULT CommandHandler(WORD /*wNotifyCode*/, WORD /*wID*/, HWND /*hWndCtl*/,
       BOOL& /*bHandled*/)
//       LRESULT NotifyHandler(int /*idCtrl*/, LPNMHDR /*pnmh*/, BOOL& /*bHandled*/)
        LRESULT OnFileOpen(WORD /*wNotifyCode*/, WORD /*wID*/, HWND /*hWndCtl*/,
              BOOL& /*bHandled*/);
        LRESULT OnCount(int counter);

};
```

B.2.4 FCS2View.cpp

```
// FCS2View.cpp : implementation of the CFCS2View class
//
/////////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "resource.h"

#include "FCS2View.h"

BOOL CFCS2View::PreTranslateMessage(MSG* pMsg)
{
        pMsg;
        return FALSE;
}


LRESULT CFCS2View::OnFileOpen(WORD /*wNotifyCode*/, WORD /*wID*/, HWND /*hWndCtl*/,
      BOOL& /*bHandled*/)
{
        CString strFolderPath="";
        CString strFileName="";
        LPSTR chBuffer;
        chBuffer=new char[65535];
        *chBuffer=0;

CFileDialog fileDlg(TRUE,0,0,OFN_ALLOWMULTISELECT|OFN_EXPLORER);
fileDlg.m_ofn.lpstrFilter="FCM_File(*.lmd)\0*.lmd\0All_Files(*.*)\0*.*\0\0";
fileDlg.m_ofn.lpstrFile=chBuffer;
fileDlg.m_ofn.nMaxFile=65535;
if(IDOK==fileDlg.DoModal())
{
        CounterGFPOutPut="";
        CounterRFPOutPut="";
        CounterFSOutPut="";
        CounterSSOutPut="";
        strFolderPath.Format("%s",chBuffer);
        int nStart=strFolderPath.GetLength()+1;
        int nEnd=nStart;
        if(GetFileAttributes(strFolderPath)==FILE_ATTRIBUTE_DIRECTORY)
        {
                CRichEditCtrl pEdit(m_hWnd);
                pEdit.AppendText("LMDFileName\t_intNoise\t_extNoise\t_
                      totalNoise\t_g_intNoise\t_g_totNoise\t_gr_extNoise\t_
                      greenMean\t_redMean\t_greenVariance\t_redVariance\t_
                      meanfslog\t_meansslog\r\n");
                for(int nIndex=nStart;nIndex<6144;nIndex++){
                        if (chBuffer[nIndex]==0)
                        {
                                nEnd=nIndex;
                                if(nStart== nEnd){break;}
                                strFileName.Format("%s",&chBuffer[nStart]);
                                pEdit.AppendText(strFileName);
                                pEdit.AppendText("\t");
                                strFileName=strFolderPath+"\\"+strFileName;
                                HANDLE hFile;
                                hFile=CreateFile(/*fileDlg.m_szFileTitle*/
                                      strFileName,
                                        GENERIC_READ,
                                        0,
                                        NULL,
                                        OPEN_EXISTING,
                                        FILE_ATTRIBUTE_NORMAL,
                                        NULL);

                                char ch[59];
                                DWORD dwReads;
                                ReadFile(hFile,ch,58,&dwReads,NULL);
                                ch[dwReads]=0;
                                CString textStart,textEnd,dataStart,dataEnd,
                                      analysisStart,analysisEnd;
                                int iTextStart,iTextEnd,iDataStart,iDataEnd,
                                      iAnalysisStart,iAnalysisEnd;
                                int i;
                                for(i=10;i<18;i++)
                                {
                                        textStart+=ch[i];
                                }
                                iTextStart=atoi(textStart);
                                for(i=18;i<26;i++)
                                {
                                        textEnd+=ch[i];
                                }
                                iTextEnd=atoi(textEnd);
                                for(i=26;i<34;i++)
                                {
                                        dataStart+=ch[i];
                                }
                                iDataStart=atoi(dataStart);
                                for(i=34;i<42;i++)
                                {
```

```
        dataEnd+=ch[i];
}
iDataEnd=atoi(dataEnd);
for(i=42;i<50;i++)
{
        analysisStart+=ch[i];
}
iAnalysisStart=atoi(analysisStart);
for(i=50;i<58;i++)
{
        analysisEnd+=ch[i];
}
iAnalysisEnd=atoi(analysisEnd);


char *textBuf;
textBuf=new char[iTextEnd+1];
SetFilePointer(hFile,0,NULL,FILE_BEGIN);
ReadFile(hFile,textBuf,iTextEnd,&dwReads,
      NULL);
textBuf[dwReads]=0;

//to find the sample name.
char *split="!";
char *sample="SRC";
char *textTemp=strstr(textBuf,sample)+strlen
      (sample)+strlen(split);
int indexOfSample=strstr(textTemp,split)-
      textTemp;
textTemp[indexOfSample]='\0';
pEdit.AppendText(textTemp);
pEdit.AppendText("\t");

CounterGFPOutPut=CounterGFPOutPut+textTemp+"
      \t";
CounterRFPOutPut=CounterRFPOutPut+textTemp+"
      \t";
CounterFSOutPut=CounterFSOutPut+textTemp+"\t
      ";
CounterSSOutPut=CounterSSOutPut+textTemp+"\t
      ";


BYTE *dataBuf;
int dataSize=iDataEnd-iDataStart+1;
dataBuf=new BYTE[dataSize];
SetFilePointer(hFile,iDataStart,NULL,
      FILE_BEGIN);
ReadFile(hFile,dataBuf,dataSize,&dwReads,
      NULL);
unsigned int *datalist;
datalist=new unsigned int[dataSize/2];
int j;
for(i=0,j=0;i<dataSize/2;i++,j+=2)
{
        datalist[i]=MAKEWORD(dataBuf[j],(
              dataBuf[j+1]&0x03));
}
rawdata=datalist;//send the rawdata to
      public

fcm1=new fcm(dataSize,datalist);
m_wflag=1;

CloseHandle(hFile);
```

```
        OnCount(1);
        OnCount(0);
        nStart = nEnd + 1;
    }
}
}
else
{

    HANDLE hFile;
    hFile=CreateFile(fileDlg.m_szFileTitle,
        GENERIC_READ,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    char ch[59];
    DWORD dwReads;
    ReadFile(hFile,ch,58,&dwReads,NULL);
    ch[dwReads]=0;
    CString textStart,textEnd,dataStart,dataEnd,
        analysisStart,analysisEnd;
    int iTextStart,iTextEnd,iDataStart,iDataEnd,
        iAnalysisStart,iAnalysisEnd;
    int i;
    for(i=10;i<18;i++)
    {
        textStart+=ch[i];
    }
    iTextStart=atoi(textStart);
    for(i=18;i<26;i++)
    {
        textEnd+=ch[i];
    }
    iTextEnd=atoi(textEnd);
    for(i=26;i<34;i++)
    {
        dataStart+=ch[i];
    }
    iDataStart=atoi(dataStart);
    for(i=34;i<42;i++)
    {
        dataEnd+=ch[i];
    }
    iDataEnd=atoi(dataEnd);
    for(i=42;i<50;i++)
    {
        analysisStart+=ch[i];
    }
    iAnalysisStart=atoi(analysisStart);
    for(i=50;i<58;i++)
    {
        analysisEnd+=ch[i];
    }
    iAnalysisEnd=atoi(analysisEnd);


    char *textBuf;
    textBuf=new char[iTextEnd+1];
    SetFilePointer(hFile,0,NULL,FILE_BEGIN);
    ReadFile(hFile,textBuf,iTextEnd,&dwReads,NULL);
    textBuf[dwReads]=0;
```

```
                        //to find the sample name.
                        char *split="!";
                        char *sample="SRC";
                        char *textTemp=strstr(textBuf,sample)+strlen
                            (sample)+strlen(split);
                        int indexOfSample=strstr(textTemp,split)-
                            textTemp;
                        textTemp[indexOfSample]='\0';

                    CRichEditCtrl pEdit(m_hWnd);
                    pEdit.AppendText(/*fileDlg.m_szFileTitle*/textTemp);
                    pEdit.AppendText("\r\n");

                    BYTE *dataBuf;
                    int dataSize=iDataEnd-iDataStart+1;
                    dataBuf=new BYTE[dataSize];
                    SetFilePointer(hFile,iDataStart,NULL,FILE_BEGIN);
                    ReadFile(hFile,dataBuf,dataSize,&dwReads,NULL);
                    unsigned int *datalist;
                    datalist=new unsigned int[dataSize/2];
                    int j;
                    for(i=0,j=0;i<dataSize/2;i++,j+=2)
                    {
                        datalist[i]=MAKEWORD(dataBuf[j],(dataBuf[j
                            +1]&0x03));
                    }
                    rawdata=datalist;//send the rawdata to public

                    fcm1=new fcm(dataSize,datalist);
                    m_wflag=1;

                    CloseHandle(hFile);

                    pEdit.AppendText("intNoise\t_extNoise\t_totalNoise\t
                        _g_intNoise\t_g_totNoise\t_gr_extNoise\t_
                        greenMean\t_redMean\t_greenVariance\t_
                        redVariance\r\n");
                    OnCount(1);
                    OnCount(0);
                }
        }
            delete  chBuffer;

            CRichEditCtrl pEdit(m_hWnd);
            pEdit.AppendText("GFP_Events_\r\n"+CounterGFPOutPut);
            pEdit.AppendText("RFP_Events_\r\n"+CounterRFPOutPut);
            pEdit.AppendText("FS_Events_\r\n"+CounterFSOutPut);
            pEdit.AppendText("SS_Events_\r\n"+CounterSSOutPut);

        return 0;
}

LRESULT CFCS2View::OnCount(int counter)
{
        if(m_wflag)
        {
            CRichEditCtrl pEdit(m_hWnd);
            if(!counter)
            {
                int step=1;
//              fcm1->resetFilter();
                CString str;
                fcm1->greenCount(step);
                fcm1->redCount(step);
                fcm1->fslogCount(step);
                fcm1->sslogCount(step);
                for(int i=1;i<(1024/step);i++)
                {
                    str.Format("%i",fcm1->fslogEvents[i]);
                    CounterFSOutPut=CounterFSOutPut+str+'\t';
                    str.Format("%i",fcm1->sslogEvents[i]);
                    CounterSSOutPut=CounterSSOutPut+str+'\t';
                    str.Format("%i",fcm1->greenEvents[i]);
                    CounterGFPOutPut=CounterGFPOutPut+str+'\t';
                    str.Format("%i",fcm1->redEvents[i]);
                    CounterRFPOutPut=CounterRFPOutPut+str+"\t";
                }
                CounterFSOutPut=CounterFSOutPut+"\r\n";
                CounterSSOutPut=CounterSSOutPut+"\r\n";
                CounterGFPOutPut=CounterGFPOutPut+"\r\n";
                CounterRFPOutPut=CounterRFPOutPut+"\r\n";
            }
            else
            {
                float intNoise;
                float extNoise;
                float totNoise;
                float correlation;
                float g_intNoise;
                float correlationgreen;
                float redmean;
                float greenmean;
                float redvariance;
                float greenvariance;
//              float parCorrelation;
                int fslogPeak;
                int sslogPeak;
                char *ch;
                ch=new char[256];
                CString str;

                fslogPeak=fcm1->getFSPeak();
                sslogPeak=fcm1->getSSPeak();

                fcm1->fluoresenceConvert(1, 0, fslogPeak, sslogPeak, 1);
                fcm1->rFilter(50, fslogPeak, sslogPeak);
                intNoise=fcm1->intNoise();
                extNoise=fcm1->extNoise();
                totNoise=fcm1->total();
                correlation=fcm1->correlation();
                g_intNoise=fcm1->g_intNoise();
                correlationgreen=fcm1->correlationgreen();
                redmean=fcm1->meanred();
                greenmean=fcm1->meangreen();
                redvariance=fcm1->variancerfplog();
                greenvariance=fcm1->variancegfplog();

                sprintf(ch,"%8f\t_%8f\t_%8f\t_%8f\t_%8f\t_%8f\t_%8f\t_%8f\t_
                    %8f\t_%8f\t_%d\t_%d\t",
                        intNoise,extNoise,totNoise,g_intNoise,
                            correlationgreen,correlation,greenmean,redmean,
                            greenvariance,redvariance,fslogPeak,sslogPeak);
                str.Format("%s",ch);
                outPut=str;
pEdit.AppendText(outPut);
                int events=0;
                events=fcm1->getTotalEvents();
                str.Format("Events_total:_%d_\r\n",events);
                pEdit.AppendText(str);
            }
```

```
        }
        else {MessageBox("please open a fcm file first before do counter","Cannot
            finish your operation.",MB_ICONWARNING);}
        return 0;
}
```

B.3 Dual-fluorescence system simulation tool

B.3.1 dualFluGil.cpp

```
////////////////////////////////////////////////////////////////////////////////
// gillespieMulti.cpp : Gillespie algorithm using the Next Reaction Method    //
//                      invented by Gibson and Bruck, for exact stochastic    //
//                      simulations of di-fluorescence system.                //
// Description:                                                               //
//            Di-fluorescence system is a system with to different fluore-     //
//            scences, one is from green fluorescence protein (GFP)and the    //
//            other is red fluorescence protein (RFP). The RFP is controled   //
//            by the PlPr promoter, which comes from lambda bacteriophage.    //
//            The GFP can be controled by any interested target promoter of   //
//            Escherichia coli. This di-fluorescence system provides the      //
//            possibility for complex gene network analysis and dynamical     //
//            process during unsteady state.                                  //
//            Use binary tree to store the time \& reaction in pair use time  //
//            as the sorting order (from low to high).                        //
//            Because the extrinsic noise is shared by all the genes in the   //
//            same cell, the GFP and RFP share the same random seed for each  //
//            particle simulation.                                            //
//            Multiple thread was used to increase the simulation efficiency  //
//            for such big amount particles.                                  //
//            The reaction rate of RFP is determined by reallity data.        //
//            Lots of parameters can be set during initialise step, which     //
//            include the growth speed, the active and inactive reaction rate //
//            of target promoter and the noise properties.                    //
//            The grwoth noise is followed the normal distribution.           //
//            The extrinsic noise for each step of the reaction is followed   //
//            log-normal distribution.                                        //
//            All the noise distribution can be controled by the lambda value,//
//            which defined same as in the normal distribution.               //
//            The cell will first incubate in one condition till steady state.//
//            Then thransfer to another condition. To reach the steady state  //
//            in preculture please make sure to set the precultrue time big   //
//            enough.                                                         //
// Reference: Gibson, Bruck, "Efficient Exact Stochastic Simulation of Chemical//
//            Systems with Many Species and Many Channels"                    //
//            Fox, Gatland, Roy, Vemuri, "Fast,accurate algorithm for numerical//
//            simulation of exponentially correlated colored noise."          //
//            Shahrezaei, Ollivier, Swain, "Colored Extrinsic Fluctuations and //
//            stochastic gene expression."                                    //
//            Sugiyama, Kawabata, Sobue, Okabe, "Determination of absolute    //
//            protein numbers in single synapses by a GFP-based calibration   //
//            technique."                                                     //
//            Golding, Paulsson, Zawilski, Cox, "Real-time kinetics of gene   //
//            Activity in individual bacteria."                               //
// Author: Jianhong Ou                                                        //
// Current Version : 1.0                                                      //
// Date: 2009/02/11                                                           //
// Class Needed in This File: BTree in BTree.h                                //
////////////////////////////////////////////////////////////////////////////////
#include "BTree.h"
#include <pthread.h>
#include <math.h>
#include <sys/time.h>
#define MAX_THREAD 50
#define RN 6 //reaction number;
#define KRM 5 //kind of reaction molecular;
#define IntensityRatio 0.335//the fluorescence intensity ratio of I(RFP):I(GFP);
      measured by negative control.
#define IntensityPerProtein 10//determined by laser power;
#define background 260//involve background;can easily increase the extrinsic noise.
#include <string>
#include <iostream>
#include <fstream>
#include <stdio.h>
using namespace std;

double *c=NULL;//reaction constants for GFP;
double *cRed=NULL;//reaction constants for RFP;
int **R=NULL;//molecular numbers for each reaction of GFP;
int **RRed=NULL;//molecular numbers for each reaction ofRFP;
int **O=NULL;//dependency graph for GFP;
int **ORed=NULL;//dependency graph for RFP;
int Iinit[5];
int IinitRed[5];
bool division=0;//the flag indicate including the cell division or not.
double cycleTime0=0;//set by the growth rate.
bool extrinsic=0;//the flag indicte including the extrinsic noise or not.
bool extFlag[6];//the noise flag for each step.
double lambda[6];//the noise strength for each step of GFP&RFP.lambda>1;
double deltaT=0.01;//time step used in gaussian.
double cActiveNoLys=-1;//reaction constants of inactive to active if there is no
      lysine;
double cActiveLys=-1;//reaction constants of inactive to active if there is lysine;
double cActiveRed=-1;//reaction constants of inactive to active of PLPR;
int Counter=1000;//set sample number, less than 12000;
int sampleNo=10;//sample number for each paticle;
double multiTime=1;//factor for changing simulation time to real time.
double ShiftTime=0;//set the preculture time.
double CultureTime=0;//set the culture time;
double tMax=0;//calculated by preculture time and culture time.
double interval=0;//the interval for sampling calculated by culture time divided by
      the sample number.
bool noLys0=0;//set as in main culture;

pthread_t thread[MAX_THREAD];//create multiple thread;
pthread_mutex_t mut,muta;//use mute to protect the data.

ofstream ofsRFP;//for recording the RFP data;
ofstream ofsGFP;//for recording the GFP data;
ofstream ofsA;//for reacording the a data;

struct A{
        int reactStep;
        double valueA;
        A(int bb=0, double cc=0){
                reactStep = bb; valueA=cc;
        }
};

void write(ofstream& out, A value){
        out.write(reinterpret_cast<char*>(&value),sizeof(A));
}
void read(ifstream& in, A& value){
```

```
        in.read(reinterpret_cast<char*>(&value),sizeof(A));
}

double propensity(int i, int **React, double *reactConst, int *Init)//calculate the
      propensity function
{
        double ai=1;
        for(int j=0;j<KRM;j++)
        {
                double h1=1;
                double h2=1;
                if(React[i][j]>0)
                {
                        for(int w=0;w<React[i][j];w++)
                        {
                                h1*=(Init[j]-w);
                                h2*=(w+1);
                        }
                        ai*=(h1/h2);
                        if(ai<0) ai=0;
                }
        }
        ai=ai*reactConst[i];
        return ai;
}


double gasdev(unsigned int *seed, double D, double E, double &prev, bool &iset,
      double &gset, bool white=0)
{
        static double epsilon=0;
        double h, fac, rsq, v1, v2;

        if(!iset)
        {
                do
                {
                        v1=2.0*rand_r(seed)/(RAND_MAX+1.0)-1.0;
                        v2=2.0*rand_r(seed)/(RAND_MAX+1.0)-1.0;
                        rsq=v1*v1+v2*v2;
                }while(rsq>=1.0 || 0==rsq);
                fac=sqrt(D*log(rsq)/rsq);
                gset=v1*fac;
                h=v2*fac;
                iset=1;
        }
        else
        {
                h=gset;
                iset=0;
        }
        if(white)
        {
                epsilon=h;
        }
        else
        {
                epsilon=prev*E+h;
                prev=epsilon;
        }
        return epsilon;
}

double gumbelMini(unsigned int *seed, double miu, double beta)
{
        double x=0;
        do
        {
                x=rand_r(seed)/(RAND_MAX+1.0);
        }while (x==0||1==x);
        double y=0;
        return y=log(log(1.0/(1.0-x)))*beta+miu;
}

double getTime(double a, double &t0, unsigned int *seed)//generate random time;
{
        double t=0;
        do
        {
                t=rand_r(seed)/(RAND_MAX+1.0);
        }while(0>=t);
        t=t0+log(1/t)/a;
        return t;
}

void initialise(BTree<double> *PP, int **React, double *reactConst, int *Init,
      double &t0, unsigned int *seed2)//initialise heap map for each reaction
{
        double *a=NULL;
        a=new double[RN];
        double *t=NULL;
        t=new double[RN];
        for(int i=0;i<RN;i++)
        {
                //calculate the propensity function, ai, for all i;
                a[i]=propensity(i,React,reactConst,Init);

                //for each i, generate a putative time, ti, according to an
                    exponential distribution with parameter ai;
                if(a[i]>0)
                {
                        t[i]=getTime(a[i],t0,seed2);
                }
                else
                {
                        do
                        {
                                t[i]=RAND_MAX/*rand_r(seed2)/(RAND_MAX+1.0)*/;
                        }while (0>=t[i]);
                }
                PP->Insert(t[i],a[i],i);
        }

        delete a;
        a=NULL;
        delete t;
        t=NULL;
}

void runReaction(int **OO, int uu, int *Init)
{
        for(int i=0; i<KRM; i++)
        {
                Init[i]-=OO[uu][i];
        }
}

void recalculatePropensity(BTree<double> *PP, int **OO, int **React, double *
      reactConst, int *Init, double &t0, unsigned int *seed2)
{
```

```
int u;
u=PP->root->reaction;
//execute the reaction;
runReaction(OO, u, Init);
//recalculate the propensity fuction;
int *pf;
pf=new int[RN];
for(int i=0;i<RN;i++)
{
        pf[i]=0;
}
//the others
for(int i=0;i<KRM;i++)
{
        if(0!=OO[u][i])
        {
                for(int j=0;j<RN;j++)
                {
                        if(0!=React[j][i])
                        {
                                pf[j]=1;
                        }
                }
        }
}
//the reaction itself
pf[u]=2;

for(int j=0;j<RN;j++)
{
        if(pf[j]>0)
        {
                double a=0;
                double t2=0;
                double pro=0;
                double t3=0;
                pro=PP->findby(j)->propensity;
                t3=PP->findby(j)->time;
                a=propensity(j,React,reactConst,Init);
                if(a>0)
                {
                        if(2==pf[j]||0==pro)
                        {
                                t2=getTime(a,t0,seed2);
                        }
                        else
                        {
                                t2=(pro/a)*(t3-t0)+t0;
                        }
                }
                else
                {
                        do
                        {
                                t2=RAND_MAX/*rand_r(seed2)/(RAND_MAX+1.0)*/;
                        }while (0>=t2);
                        t2=t2+t0;
                }

                PP->delby(j);

                PP->Insert(t2,a,j);
        }
}
delete pf;
        pf=NULL;
}

double ChangeReactionRate(double gau, double reactI, int i, BTree<double> *PP, int
     **React, double *reactConst, int *Init, unsigned int *seed2, double &t0)
{
        if(extFlag[i])
        {
                reactConst[i]=reactI*gau;
                if(reactConst[i]<0)
                {
                        cout<<"reactConst<0"<<endl;
                        exit(0);
                }
        }
        double t=0;
        double pro=PP->findby(i)->propensity;
        double t1=PP->findby(i)->time;
        double a=propensity(i,React,reactConst,Init);
        if(a>0)
        {
                if(!pro)
                {
                        t=getTime(a,t0,seed2);
                }
                else
                {
                        t=(pro/a)*(t1-t0)+t0;
                }
        }
        else
        {
                do{
                        t=RAND_MAX/*rand_r(seed2)/(RAND_MAX+1.0)*/;
                }while(0>=t);
                t+=t0;
        }
        PP->delby(i);
        PP->Insert(t,a,i);
        return reactConst[i];
}
string InitParameter()//initialise the parameter for simulation
{
        string fname;//filename for parameters saving.
        string str;
        cout<<"Please input the filename for saving simulation result: "<<endl;
        cin>>fname;
        str=fname+".xls";
        ofstream ofs(str.c_str(),ios::app);
        str="";
        char ch;
        do
        {
                cin.clear();
                cout<<"Do you want to set the cell division time (default is 40min)?
                    (y/n): "<<endl;
                cin>>ch;
                switch(ch)
                {
                case 'y':
                case 'Y':
                        division=1;
                        cin.clear(ios::goodbit);
                        break;
                case 'n':
```

```
        case 'N':
                division=0;
                cin.clear(ios::goodbit);
                break;
        default:
                cout<<"the wrong parameter!"<<endl;
                cin.clear(ios::badbit);
        }
}while(cin.fail());
ofs<<"division \t "<<division<<endl;
if (division)
{
        do
        {
                cout<<"How long it takes for one generation (min): "<<endl;
                cin>>cycleTime0;
        }while(!cycleTime0);
        ofs<<"cycleTime \t "<<cycleTime0<<endl;
}
do
{
        cin.clear();
        cout<<"Do you want to include the extrinsic noise? (y/n): "<<endl;
        cin>>ch;
        switch(ch)
        {
        case 'y':
        case 'Y':
                extrinsic=1;
                cin.clear(ios::goodbit);
                break;
        case 'n':
        case 'N':
                extrinsic=0;
                cin.clear(ios::goodbit);
                break;
        default:
                cout<<"the wrong parameter!"<<endl;
                cin.clear(ios::badbit);
        }
}while(cin.fail());
ofs<<"extrinsic \t "<<extrinsic<<endl;
for(int i=0; i<6; i++)
{
        extFlag[i]=0;
}
double extlambdaPre[6]={-1,-1,-1,-1,-1,-1};
for(int i=0; i<6; i++)
{
        lambda[i]=extlambdaPre[i];//set the lambda for generate normal
              distribution random number for GFP.
}
if (extrinsic)
{
        int setEx=1;
        while(setEx)
        {
                cout<<"which step do you want to include the extrinsic noise
                      ? please select the number"<<endl;
                cout<<"\t 1 from active plasmid to inactive plasmid;"<<endl;
                cout<<"\t 2 from inactive plasmid to active plasmid;"<<endl;
                cout<<"\t 3 transcription;"<<endl;
                cout<<"\t 4 RNA Degradation;"<<endl;
                cout<<"\t 5 translation;"<<endl;
                cout<<"\t 6 protein degradation;"<<endl;
                cout<<"\t 0 for quit;"<<endl;
                cin>>setEx;
                switch(setEx)
                {
                case 0:
                        {
                                break;
                        }
                case 1:
                        {
                                extFlag[0]=1;
                                do
                                {
                                        cout<<"Please set the lambda for
                                              generate colored noise of step
                                              1 (from active plasmid to
                                              inactive plasmid): ";
                                        cin>>lambda[0];
                                }while(lambda[0]<0);
                                break;
                        }
                case 2:
                        {
                                extFlag[1]=1;
                                do
                                {
                                        cout<<"Please set the lambda for
                                              generate colored noise of step
                                              2 (from inactive plasmid to
                                              active plasmid): ";
                                        cin>>lambda[1];
                                }while(lambda[1]<0);
                                break;
                        }
                case 3:
                        {
                                extFlag[2]=1;
                                do
                                {
                                        cout<<"Please set the lambda for
                                              generate colored noise of step
                                              3 (transcription): ";
                                        cin>>lambda[2];
                                }while(lambda[2]<0);
                                break;
                        }
                case 4:
                        {
                                extFlag[3]=1;
                                do
                                {
                                        cout<<"Please set the lambda for
                                              generate colored noise of step
                                              4 (RNA degradation): ";
                                        cin>>lambda[3];
                                }while(lambda[3]<0);
                                break;
                        }
                case 5:
                        {
                                extFlag[4]=1;
                                do
                                {
                                        cout<<"Please set the lambda for
                                              generate colored noise of step
```

```
                                        5 (translation): ";
                                cin>>lambda[4];
                        }while(lambda[4]<0);
                        break;
                }
        case 6:
                {
                        extFlag[5]=1;                                   //
                        do
                        {
                                cout<<"Please set the lambda for
                                    generate colored noise of step
                                    6 (DNA degradation): ";
                                cin>>lambda[5];
                        }while(lambda[5]<0);
                        break;
                }
        default:
                cout<<"Please select the right number:"<<endl;
        }
    }
}
for(int i=0;i<6;i++)
{
    if(extFlag[i])
    {
        ofs<<"extFlag["<<i<<"] \t "<<extFlag[i]<<endl;
        ofs<<"lambda["<<i<<"] \t "<<lambda[i]<<endl;
    }
}

c=new double[RN];
cRed=new double[RN];
do
{
    cin.clear();
    cout<<"Please input the raction constant of inactive plasmid to
        active plasmid if there is No lysine: "<<endl;
    cin>>cActiveNoLys;
}while(cin.fail()||-1==cActiveNoLys);
ofs<<"cActiveNoLys \t "<<cActiveNoLys<<endl;
do
{
    cin.clear();
    cout<<"Please input the raction constant of inactive plasmid to
        active plasmid if there is lysine: "<<endl;
    cin>>cActiveLys;
}while(cin.fail()||-1==cActiveLys);
ofs<<"cActiveLys \t "<<cActiveLys<<endl;
do
{
    cin.clear();
    cout<<"Please input the raction constant of inactive plasmid to
        active plasmid for RFP: "<<endl;
    cin>>cActiveRed;
}while(cin.fail()||-1==cActiveRed);
ofs<<"cActiveRed \t "<<cActiveRed<<endl;
double react[6]={2.4,0.0015,2.1,0.24,7.5,0.02};//active2unactive,
    unactive2active,active2rna,rnaDegrade,rna2protein,proteinDegrade.
int setInt=1;
while(setInt)
{
    cout<<"which reaction constant do you want to change? please select
        the number"<<endl;
cout<<"\t 1 from active plasmid to inactive plasmid: "<<react[0]<<
    endl;
//              cout<<"\t 2 from inactive plasmid to active
    plasmid: "<<react[1]<<endl;
cout<<"\t 3 transcription: "<<react[2]<<endl;
cout<<"\t 4 RNA Degradation: "<<react[3]<<endl;
cout<<"\t 5 translation: "<<react[4]<<endl;
cout<<"\t 6 protein degradation: "<<react[5]<<endl;
cout<<"\t 0 for quit;"<<endl;
cin>>setInt;
switch(setInt)
{
case 0:
    {
        break;
    }
case 1:
    {
        do
        {
            cin.clear();
            cout<<"Please input the reaction constant
                from active plasmid to inactive plasmid
                : ";
            cin>>react[0];
        }while(cin.fail());
        break;
    }
case 2:
    {
        /*                                          do
        {
        cout<<"Please input the reaction constant from
            inactive plasmid to active plasmid: ";
        cin>>react[1];
        }while(cin.fail());*/
        cout<<"this part defined already"<<endl;
        break;
    }
case 3:
    {
        do
        {
            cin.clear();
            cout<<"Please input the reaction constant
                for transcription: ";
            cin>>react[2];
        }while(cin.fail());
        break;
    }
case 4:
    {
        do
        {
            cin.clear();
            cout<<"Please input the reaction constant
                for RNA Degradation: ";
            cin>>react[3];
        }while(cin.fail());
        break;
    }
case 5:
    {
        do
        {
```

```
                                cin.clear();
                                cout<<"Please input the reaction constant
                                    for translation: ";
                                cin>>react[4];
                            }while(cin.fail());
                            break;
                    }
/*          case 6:
                    {
                            do
                            {
                                cin.clear();
                                cout<<"Please input the reaction constant
                                    for protein degradation: ";
                                cin>>react[5];
                            }while(cin.fail());
                            break;
                    }*/
            default:
                    cout<<"Please select the right number: "<<endl;
            }
    }
    double reactRed[6]={2.4,0.024,2.2,0.24,8,0.02};
    if(cycleTime0)
    {
            reactRed[5]=0.693/cycleTime0;
            react[5]=reactRed[5];
    }
    reactRed[1]=cActiveRed;
    for(int i=0; i<RN; i++)
    {
            ofs<<"reactionConstant["<<i<<"] \t "<<react[i]<<endl;
            c[i]=react[i];//reaction sonstants;
            cRed[i]=reactRed[i];
    }
    int init[5]={0,30,0,0,0};//active,unactive,rna,degraded rna, protein; cannot
        have 0;
    int initRed[5]={0,30,0,0,0};
    for(int i=0; i<KRM; i++)
    {
            Iinit[i]=init[i];
            IinitRed[i]=initRed[i];
    }
    R=new int*[RN];
    for(int i=0;i<RN;i++)
    {
            R[i]=new int[KRM];
    }
    RRed=new int*[RN];
    for(int i=0;i<RN;i++)
    {
            RRed[i]=new int[KRM];
    }

    int mn[6][5]=
    {
            1,0,0,0,0,
            0,1,0,0,0,
            1,0,0,0,0,
            0,0,1,0,0,
            0,0,1,0,0,
            0,0,0,0,1
    };
    int mnRed[6][5]=
    {
            1,0,0,0,0,
            0,1,0,0,0,
            1,0,0,0,0,
            0,0,1,0,0,
            0,0,1,0,0,
            0,0,0,0,1
    };
    for(int i=0;i<RN;i++)
    {
            for(int j=0;j<KRM;j++)
            {
                    R[i][j]=mn[i][j];//molecular numbers for each reaction;
                    RRed[i][j]=mnRed[i][j];
            }
    }
    O=new int*[RN];
    for(int i=0;i<RN;i++)
    {
            O[i]=new int[KRM];
    }
    ORed=new int*[RN];
    for(int i=0;i<RN;i++)
    {
            ORed[i]=new int[KRM];
    }

    int dg[6][5]=
    {
            1,-1,0,0,0,
            -1,1,0,0,0,
            0,0,-1,0,0,
            0,0,1,-1,0,
            0,0,0,0,-1,
            0,0,0,0,1
    };
    int dgRed[6][5]=
    {
            1,-1,0,0,0,
            -1,1,0,0,0,
            0,0,-1,0,0,
            0,0,1,-1,0,
            0,0,0,0,-1,
            0,0,0,0,1
    };
    for(int i=0;i<RN;i++)
    {
            for(int j=0;j<KRM;j++)
            {
                    O[i][j]=dg[i][j];//dependency graph;
                    ORed[i][j]=dgRed[i][j];
            }
    }

    do
    {
            cout<<"Please input how many particles do you want to simulate for
                one group of parameters (1~10000):";
            cin>>Counter;
    }while(Counter<=0||Counter>10000);
    ofs<<"Counter \t "<<Counter<<endl;
    do
    {
            cout<<"Please input how many samples do you want to take for each
                paticles with same interval (1~20):";
            cin>>sampleNo;
```

```
        }while(sampleNo<=0||sampleNo>20);
        ofs<<"sampleNo \t "<<sampleNo<<endl;

        do
        {
                cout<<"Please input the preculture time (min):";
                cin>>ShiftTime;
                ShiftTime*=multiTime;
        }while(!ShiftTime);
        ofs<<"ShiftTime \t "<<ShiftTime<<endl;
        do
        {
                cout<<"Please input the culture time (min):";
                cin>>CultureTime;
                CultureTime*=multiTime;
        }while(!CultureTime);
        ofs<<"CultureTime \t "<<CultureTime<<endl;
        tMax=CultureTime+ShiftTime;//use time as stop condition.
        interval=(double)CultureTime/sampleNo;

        do
        {
                cin.clear();
                cout<<"Is there lysine or not in main culture media? (y/n): "<<endl;
                cin>>ch;
                switch(ch)
                {
                case 'y':
                case 'Y':
                        noLys0=1;
                        cin.clear(ios::goodbit);
                        break;
                case 'n':
                case 'N':
                        noLys0=0;
                        cin.clear(ios::goodbit);
                        break;
                default:
                        cout<<"the wrong parameter!"<<endl;
                        cin.clear(ios::badbit);
                }
        }while(cin.fail());
        ofs<<"LysineInMainCulture \t "<<noLys0<<endl;
        ofs.close();
        return fname;
}

void *threadx(void* arg)//simulation thread.
{
        unsigned int seed1[6], seed2;//store the random seed for different use.
              seed1 for extrinsic noise, seed2 for intrinsic noise;
        timeval tim;
        gettimeofday(&tim, NULL);
        for(int i=0; i<6; i++)
        {
                seed1[i]=(unsigned)(tim.tv_usec+70*i);
        }

        bool noLys=noLys0;//set as in main culture.
        bool flag=1;//use for transfer state record.
        (noLys)?(c[1]=cActiveNoLys):(c[1]=cActiveLys);

        int init[KRM];
        int initRed[KRM];
        for(int i=0; i<KRM; i++)
        {
                init[i]=Iinit[i];
                initRed[i]=IinitRed[i];
        }
        int *I, *IRed;
        I=new int[KRM];
        IRed=new int[KRM];
        I=init;//initial molecular population numbers;
        IRed=initRed;

        double t0=0;//initial time;
        double t1=t0;//store the last time.

        int step=0;
        double interTime=ShiftTime;
        int gfpj=0;

        gettimeofday(&tim, NULL);
        seed2=(unsigned)tim.tv_usec;//initialize random array for intrinsic noise;

        double reactI[6], reactRedI[6];
        for(int i=0; i<6; i++)
        {
                reactI[i]=c[i];
                reactRedI[i]=cRed[i];
        }

        double *react, *reactRed;
        react=new double[RN];
        reactRed=new double[RN];
        for(int i=0; i<6; i++)
        {
                react[i]=c[i];
                reactRed[i]=cRed[i];
        }

        //store the ti values in an indexed priority queue P.
        BTree<double> P;
        initialise(&P,R,react,I,t0,&seed2);

        //store the ti values in an indexed priority queue P.
        BTree<double> PRed;
        initialise(&PRed,RRed,reactRed,IRed,t0,&seed2);

        bool expFlag;

        int *GFP, *RFP;
        GFP=new int[sampleNo];
        RFP=new int[sampleNo];
        for(int i=0; i<sampleNo; i++)
        {
                GFP[i]=0;
                RFP[i]=0;
        }

        double interValX=(double)CultureTime/1000;
        double interTimeX=ShiftTime;
        double deltaT=0.01;
        double interTimeO=0;
        double maxExt[6]={0};
        double minExt[6]={RAND_MAX,RAND_MAX,RAND_MAX,RAND_MAX,RAND_MAX,RAND_MAX};
        char ch[256];
        sprintf(ch,"%d.ou",tim.tv_usec);
        ofstream ofsa(ch,ios::app|ios::binary);
        double D[6]={-2};
```

```
double E[6]={0};
double white[6]={0};
double mean[6]={1};
if(extrinsic)
{
        for(int i=0; i<6; i++)
        {
                if(extFlag[i])
                {
                        if(0==lambda[i])
                        {
                                white[i]=1;
                                D[i]=-2;
                                E[i]=0;
                                mean[i]=1;
                        }
                        else
                        {
                                white[i]=0;
                                E[i]=exp(-deltaT*lambda[i]);
                                D[i]=(E[i]*E[i]-1)*2*lambda[i];
                                mean[i]=exp(E[i]*lambda[i]/2);
                        }
                }
        }
}
double prev[6]={0};
bool iset[6]={0};
double gset[6]={0};

while(t0<tMax)
{
        t1=t0;
        double gau=0;
        if(P.root&&PRed.root)
        {
                //Shift the reaction rate when there the media changed.
                if(t0>ShiftTime&&flag)
                {
                        noLys=!noLys;
                        flag=0;
                        if(noLys)
                        {
                                react[1]=cActiveNoLys;
                        }
                        else
                        {
                                react[1]=cActiveLys;
                        }
                }
                //Include the extrinsic noise.
                if(extrinsic)
                {
                        for(int i=0; i<6; i++)
                        {
                                if(extFlag[i])
                                {
                                        if(t0>interTimeO)
                                        {
                                                gau=exp(gasdev(&seed1[i], D[
                                                        i], E[i], prev[i], iset
                                                        [i], gset[i],white[i]))
                                                        /mean[i];
                                                ChangeReactionRate(gau,
                                                        reactI[i], i, &P, R,
                                                        react, I, &seed2, t0);
                                                ChangeReactionRate(gau,
                                                        reactRed1[i], i, &PRed,
                                                        RRed, reactRed, IRed,
                                                        &seed2, t0);
                                                interTimeO+=(deltaT*10);
                                        }
                                        if(t0>interTimeX&&(!flag))
                                        {
                                                A a(i,react[i]);
                                                write(ofsa,a);
                                                if(maxExt[i]<react[i])
                                                        maxExt[i]=react[i];
                                                if(minExt[i]>react[i])
                                                        minExt[i]=react[i];
                                                interTimeX+=interValX;
                                        }
                                }
                        }
                }

                (P.root->time<PRed.root->time) ? (expFlag=1) : (expFlag=0);
                if(expFlag)
                {
                        t0=P.root->time;//change time to next reaction time;
                        recalculatePropensity(&P, O, R, react, I, t0,&seed2)
                                ;
                }
                else
                {
                        t0=PRed.root->time;//change time to next reaction
                                time;
                        recalculatePropensity(&PRed, ORed, RRed, reactRed,
                                IRed, t0,&seed2);
                }

                //store protein number and time.
                if(t0>interTime&&gfpj<sampleNo)
                {
                        if(I[KRM-1]>0)
                                GFP[gfpj]=I[KRM-1];
                        if(IRed[KRM-1]>0)
                                RFP[gfpj]=IRed[KRM-1];
                        gfpj++;
                        interTime=interval*gfpj+ShiftTime;
                }
        }
        else{cout<<"root->time=0"<<"\r";}
}

pthread_mutex_lock(&mut);
for(int i=0; i<sampleNo; i++)
{
        ofsGFP<<GFP[i]<<"\t";
        ofsRFP<<RFP[i]<<"\t";
}
ofsGFP<<endl;
ofsRFP<<endl;
pthread_mutex_unlock(&mut);
delete GFP;
GFP=NULL;
delete RFP;
RFP=NULL;

ofsa.close();
```

```
        if(extrinsic)
        {
                ifstream ifsa(ch,ios::binary);
                A eve[6];
                int eventA[6][1001]={0};
                A a(0,0);
                int record[6]={0};
                for(int j=0; j<6; j++)
                {
                        eve[j].reactStep=j;
                }
                while(!ifsa.eof())
                {
                        read(ifsa,a);
                        record[a.reactStep]++;
                        eve[a.reactStep].valueA+=a.valueA;
                        eventA[a.reactStep][(int)(1000*(a.valueA-minExt[a.reactStep
                                ])/(maxExt[a.reactStep]-minExt[a.reactStep]))]+=1;
                }

                pthread_mutex_lock(&muta);
                for(int i=0; i<6; i++)
                {
                        if(record[i])
                        {
                                eve[i].valueA=eve[i].valueA/record[i];
                                ofsA<<minExt[i]<<"\t"<<maxExt[i]<<endl;
                                ofsA<<eve[i].reactStep<<"\t "<<eve[i].valueA<<"\t "
                                        <<record[i]<<endl;
                                for(int o=0; o<1001; o++)
                                {
                                        ofsA<<eventA[i][o]<<"\t ";
                                }
                                ofsA<<endl;
                        }
                }
                pthread_mutex_unlock(&muta);
        }
        remove(ch);

        pthread_exit(NULL);
}

int thread_create(int maxThread=MAX_THREAD)//create simulation thread.
{
        int counter=0;
        int temp;
        memset(&thread, 0, sizeof(thread));
        for(int i=0; i<maxThread; i++)
        {
                if (0==(temp=pthread_create(&thread[i], NULL, threadx, NULL)))
                {
                        counter++;
                }
        }
        cout<<"created "<<counter<<" simulatiion threads."<<endl;
        return counter;
}

void thread_wait(int maxThread=MAX_THREAD)//wait for each simulation thread to
    finish.
{
        for(int i=0; i<maxThread; i++)
        {
                if(0!=thread[i])
                {
                        pthread_join(thread[i],NULL);
                }
        }
}

int main()//do simulation.
{
        string fname;//filename for parameters saving.
        string str;
        fname=InitParameter();

        pthread_mutex_init(&mut,NULL);
        pthread_mutex_init(&muta,NULL);

        str=fname+"RFP.xls";
        ofsRFP.open(str.c_str(),ios::app);
        str="";
        str=fname+"GFP.xls";
        ofsGFP.open(str.c_str(),ios::app);
        str="";
        str=fname+"_Aevent.xls";
        ofsA.open(str.c_str(),ios::app);
        int maxThread=MAX_THREAD;
        int left=Counter;
        for(int gfpi=0;gfpi<Counter;)
        {
                int succeedThread=0;
                timeval start, finish;
                double duration;
                gettimeofday(&start, NULL);

                if(left<MAX_THREAD)
                {
                        maxThread=left;
                }

                succeedThread=thread_create(maxThread);
                gfpi+=succeedThread;
                thread_wait(maxThread);
                left=Counter-gfpi;

                gettimeofday(&finish, NULL);
                duration = (double)(finish.tv_sec - start.tv_sec);

                int durationHour=(int)(duration*(left)/succeedThread)/3600;
                int durationMin=(int)(duration*(left)/succeedThread-durationHour
                        *3600)/60;
                cout<<"Already simulated "<<gfpi<<" paticles.\t last loop cost "<<
                        duration<<"seconds;""<<"Time left "<<durationHour<<" h "<<
                        durationMin<<" min"<<endl;

        }
        ofsGFP.close();
        ofsRFP.close();
        ofsA.close();

        double *multiple, *multipleG, *multipleR, *r, *g;//get and evaluate
                statistics
        multiple=new double[sampleNo];
        multipleG=new double[sampleNo];
        multipleR=new double[sampleNo];
        r=new double[sampleNo];
        g=new double[sampleNo];
        int **eventGFP;
```

```
int **eventRFP;
eventGFP=new int*[sampleNo];
eventRFP=new int*[sampleNo];
for(int i=0; i<sampleNo; i++)
{
        multiple[i]=0;
        multipleG[i]=0;
        multipleR[i]=0;
        r[i]=0;
        g[i]=0;
        eventGFP[i]=new int[1001];
        eventRFP[i]=new int[1001];
        for(int j=0; j<1001; j++)
        {
                eventGFP[i][j]=0;
                eventRFP[i][j]=0;
        }
}
str=fname+"GFP.xls";
ifstream ifsGFP(str.c_str());
str="";
str=fname+"RFP.xls";
ifstream ifsRFP(str.c_str());
str="";
int GFP=0;
int RFP=0;
double DGFP=0,DRFP=0;
unsigned int *seedBck,*seedBckR;
seedBck=new unsigned int[sampleNo];
seedBckR=new unsigned int[sampleNo];
double miu=background, beta=5*background/16;
double miuR=1.8*background, betaR=background;
double bg=0,bgRed=0;
timeval tim;
gettimeofday(&tim, NULL);
for(int i=0; i<sampleNo; i++)
{
        seedBck[i]=(unsigned)(tim.tv_usec+50*i);
        seedBckR[i]=(unsigned)(tim.tv_usec+75+50*i);
}
str=fname+"G.xls";
ofstream Gtmp(str.c_str(),ios::app);
str=fname+"R.xls";
ofstream Rtmp(str.c_str(),ios::app);
double eventMax=0;//to record the max value of event as 1024.
double eventMin=RAND_MAX;//to record the min value of event as 0.
for(int i=0; i<Counter; i++)
{
        for(int j=0; j<sampleNo; j++)
        {
                ifsGFP>>GFP;
                ifsRFP>>RFP;
                if(background)
                {
                        bg=gumbelMini(&seedBck[j],miu,beta);
                        bgRed=gumbelMini(&seedBckR[j],miuR,betaR)*
                              IntensityRatio;
                        if(0>=bg)
                        {bg=0;}
                        else {bg=43.177*exp(0.009*bg);}
                        if(0>=bgRed)
                        {bgRed=0;}
                        else {bgRed=43.177*exp(0.009*bgRed);}
                        DGFP=GFP*IntensityPerProtein+bg;
                        DRFP=RFP*IntensityPerProtein*IntensityRatio+bgRed;
                }
                else
                {
                        DGFP=GFP*IntensityPerProtein;
                        DRFP=RFP*IntensityPerProtein*IntensityRatio;
                }
                if(DGFP>eventMax)
                        eventMax=DGFP;
                if(DGFP<eventMin)
                        eventMin=DGFP;
                if(DRFP>eventMax)
                        eventMax=DRFP;
                if(DRFP<eventMin)
                        eventMin=DRFP;
                Gtmp<<DGFP<<"\t";
                Rtmp<<DRFP<<"\t";
        }

}
Gtmp.close();
Rtmp.close();
ifsGFP.close();
ifsRFP.close();

str=fname+"G.xls";
ifsGFP.open(str.c_str());
str=fname+"R.xls";
ifsRFP.open(str.c_str());
int *counter0;
counter0=new int[sampleNo];
for(int i=0; i<sampleNo; i++)
{
        counter0[i]=0;
}
if(eventMax>1000)
{
        for (int i=0; i<Counter; i++)
        {
                for (int j=0; j<sampleNo; j++)
                {
                        ifsGFP>>DGFP;
                        ifsRFP>>DRFP;
                        int dgfp=(int)(1000*(DGFP-eventMin)/(eventMax-
                              eventMin));
                        int drfp=(int)(1000*(DRFP-eventMin)/(eventMax-
                              eventMin));
                        if(dgfp<0||dgfp>1000)
                        {
                                cout<<"dgfp"<<dgfp<<endl;
                                exit(0);
                        }
                        if(drfp<0||drfp>1000)
                        {
                                cout<<"drfp"<<drfp<<endl;
                                exit(0);
                        }
                        eventGFP[j][dgfp]+=1;
                        eventRFP[j][drfp]+=1;
                        if(DGFP>0&&DRFP>0)
                        {
                                counter0[j]+=1;
                                r[j]=r[j]+(DRFP-r[j])/counter0[j];
                                g[j]=g[j]+(DGFP-g[j])/counter0[j];
                                multiple[j]=multiple[j]+(DRFP*DGFP-multiple[
                                      j])/counter0[j];
```

```
                        multipleG[j]=multipleG[j]+(DGFP*DGFP-
                            multipleG[j])/counter0[j];
                        multipleR[j]=multipleR[j]+(DRFP*DRFP-
                            multipleR[j])/counter0[j];
                    }
                }
        }
}
else
{
        for (int i=0; i<Counter; i++)
        {
                for (int j=0; j<sampleNo; j++)
                {
                        ifsGFP>>DGFP;
                        ifsRFP>>DRFP;
                        int dgfp=(int)(DGFP);
                        int drfp=(int)(DRFP);
                        if(dgfp<0||dgfp>1000)
                        {
                                cout<<"dgfp"<<dgfp<<endl;
                                exit(0);
                        }
                        if(drfp<0||drfp>1000)
                        {
                                cout<<"drfp"<<drfp<<endl;
                                exit(0);
                        }
                        eventGFP[j][dgfp]+=1;
                        eventRFP[j][drfp]+=1;
                        if(DGFP>0&&DRFP>0)
                        {
                                counter0[j]+=1;
                                r[j]=r[j]+(DRFP-r[j])/counter0[j];
                                g[j]=g[j]+(DGFP-g[j])/counter0[j];
                                multiple[j]=multiple[j]+(DRFP*DGFP-multiple[
                                    j])/counter0[j];
                                multipleG[j]=multipleG[j]+(DGFP*DGFP-
                                    multipleG[j])/counter0[j];
                                multipleR[j]=multipleR[j]+(DRFP*DRFP-
                                    multipleR[j])/counter0[j];
                        }
                }
        }
}
delete counter0;
counter0=NULL;

delete seedBck;
seedBck=NULL;
delete seedBckR;
seedBckR=NULL;
ifsGFP.close();
ifsRFP.close();
remove(str.c_str());
str=fname+"G.tmp";
remove(str.c_str());

str=fname+"_event.xls";
ofstream ofs(str.c_str(),ios::app);
str="";
double intNoiseG=0;
double intNoiseR=0;
double extNoise=0;
double totalNoiseG=0;
double totalNoiseR=0;
for(int j=0; j<sampleNo; j++)
{
        ofs<<g[j]<<"\t";
}
for(int j=0; j<sampleNo; j++)
{
        ofs<<r[j]<<"\t";
}
ofs<<"mean\t"<<endl;
for(int j=0; j<sampleNo; j++)
{
        totalNoiseG=(multipleG[j]-g[j]*g[j])/(g[j]*g[j]);
        ofs<<totalNoiseG<<"\t";
}
for(int j=0; j<sampleNo; j++)
{
        totalNoiseR=(multipleR[j]-r[j]*r[j])/(r[j]*r[j]);
        ofs<<totalNoiseR<<"\t";
}
ofs<<"total_Noise\t"<<endl;
for(int j=0; j<sampleNo; j++)
{
        extNoise=(multiple[j]-r[j]*g[j])/(r[j]*g[j]);
        ofs<<extNoise<<"\t";
}
ofs<<"extrinsic_Noise\t"<<endl;
for(int j=0; j<sampleNo; j++)
{
        intNoiseG=(multipleG[j]-g[j]*g[j])/(g[j]*g[j])-(multiple[j]-r[j]*g[j
            ])/(r[j]*g[j]);
        ofs<<intNoiseG<<"\t";
}
for(int j=0; j<sampleNo; j++)
{
        intNoiseR=(multipleR[j]-r[j]*r[j])/(r[j]*r[j])-(multiple[j]-r[j]*g[j
            ])/(r[j]*g[j]);
        ofs<<intNoiseR<<"\t";
}
ofs<<"intrinsic_Noise\t"<<endl;

delete multiple;
multiple=NULL;
delete multipleG;
multipleG=NULL;
delete multipleR;
multipleR=NULL;
delete r;
r=NULL;
delete g;
g=NULL;


ofs<<"eventMax=\t"<<eventMax<<"\t"<<"eventMin=\t"<<eventMin<<endl;

for(int j=0; j<sampleNo; j++)
{
        ofs<<"GFP"<<j<<"\t";
}
for(int j=0; j<sampleNo; j++)
{
        ofs<<"RFP"<<j<<"\t";
}
ofs<<endl;
```

```
        for(int i=0; i<1001; i++)
        {
                for(int j=0; j<sampleNo; j++)
                {
                        ofs<<eventGFP[j][i]<<"\t ";
                }
                for(int j=0; j<sampleNo; j++)
                {
                        ofs<<eventRFP[j][i]<<"\t ";
                }
                ofs<<endl;
        }
        ofs.close();
        delete eventGFP;
        eventGFP=NULL;
        delete eventRFP;
        eventRFP=NULL;
        return 0;
}
```

B.3.2 BTree.h

```
/*
* FileName:BTree.h
* Description:binary tree with insert, delete, find
* Date:2008-11-4
*/
#ifndef BTree_H_H
#define BTree_H_H

#include<iostream>
#include<deque>
using namespace std;

//binary node with parent node
template<typename T>
class node
{
public:
    node(const T &v, const T &p, int &r, node<T> *L=NULL, node<T> *R=NULL, node<T> *
        P=NULL):left(L),right(R),par(P)
    {
        time = v;
                propensity = p;
                reaction = r;
    }

public:
    T time;
        T propensity;
        int reaction;
    node<T> *left, *right, *par;
};

// binary tree
template<typename T>
class BTree
{
public:
    BTree(node<T> *R=NULL):root(R)
    { }
    ~BTree()
    {
        if(root)
            delall();
    }

    node<T> *findby(int v);
    void Insert(const T &v, const T &p, int &r);
    bool delby(int v);

    node<T> *findleave(node<T> *cur);
    void delall();
    void display(node<T> *r);
        void heapify(node<T> *cur);
        void swapnode(node<T> *a, node<T> *b);

public:
    node<T> *root;
};

template<typename T>
void BTree<T>::swapnode(node<T> *a, node<T> *b)
{
        T propensity;
        int reaction;
        T tmptime;
        propensity=b->propensity;
        reaction=b->reaction;
        tmptime=b->time;
        b->propensity=a->propensity;
        b->reaction=a->reaction;
        b->time=a->time;
        a->propensity=propensity;
        a->reaction=reaction;
        a->time=tmptime;
}

template<typename T>
void BTree<T>::heapify(node<T> *cur)
{
    node<T> *left, *right, *smallest;
    if(cur)
        {
                left=cur->left;
                right=cur->right;

                if(left&&left->time<cur->time)
                        smallest=left;
                else
                        smallest=cur;
                if(right&&right->time<smallest->time)
                {
                        smallest=right;
                }

                if(smallest!=cur)
                {
                        swapnode(cur,smallest);
                        heapify(smallest);
                }
        }
}

template<typename T>
node<T> *BTree<T>::findby(int v)
{
    deque< node<T>* > Q;
```

```
    bool isfind;
    node<T> *tmp;

    if(root)
        Q.push_back(root);
    else
    {
        return NULL;
    }

    isfind = false;
    tmp = NULL;
    while(!Q.empty() && !isfind)
    {
        tmp = Q.front();
        Q.pop_front();

        if(tmp->reaction == v)
            isfind = true;
        else
        {
            if(tmp->left)
                Q.push_back(tmp->left);
            if(tmp->right)
                Q.push_back(tmp->right);
        }
    }

    if(!isfind)
        tmp = NULL;

    return tmp;
}

template<typename T>
void BTree<T>::Insert(const T &v, const T &p, int &r)
{
    deque< node<T>* > Q;
    node<T> *cur, *par;
        bool flag=1;

    if(root)
        Q.push_back(root);
    else
    {
        root = new node<T>(v, p, r, NULL, NULL, NULL);
        return;
    }

    while(!Q.empty()&&flag)
    {
        cur = Q.front();
        Q.pop_front();

        if(cur->left)
            Q.push_back(cur->left);
        else
        {
            cur->left = new node<T>(v, p, r, NULL, NULL, cur);
            flag=0;
                        par=cur;
                        cur=cur->left;
        }

                if(flag)
                {
                        if(cur->right)
                                Q.push_back(cur->right);
                        else
                        {
                                cur->right = new node<T>(v, p, r, NULL, NULL, cur);
                                flag=0;
                                par=cur;
                                cur=cur->right;
                        }
                }
    }

        while(par&&par->time>cur->time)
        {
                swapnode(cur,par);
                cur=par;
                par=par->par;
        }
}

template<typename T>
bool BTree<T>::delby(int v)
{
    node<T> *cur, *tmp, *par;
        tmp=NULL;
        par=NULL;
    bool isleave;

    isleave = false;
    cur = NULL;
    cur = findby(v);
    if(!cur)
        return false;
    else
    {
        if(cur->left && cur->right)
        {
            tmp = findleave(cur);
            tmp->left = cur->left;
            tmp->right = cur->right;

            if(cur->left)
                cur->left->par = tmp;
            if(cur->right)
                cur->right->par = tmp;
        }
        else if(cur->left)
            tmp = cur->left;
        else if(cur->right)
            tmp = cur->right;
        else
        {
            (cur == cur->par->left) ? (cur->par->left = NULL) :(cur->par->right =
                NULL);
            isleave = true;
        }

        if(!isleave)
        {
            tmp->par = cur->par;

            if(cur->par)
                (cur == cur->par->left) ? (cur->par->left = tmp) :(cur->par->right =
                    tmp);
```

```
            if(root == cur)
            {
                root = tmp;
                root->par = NULL;
            }
        }

            if(tmp) par=tmp->par;
            while(par&&par->time>tmp->time)
            {
                    swapnode(tmp,par);
                    tmp=par;
                    par=par->par;
            }

            heapify(tmp);
    }

    delete cur;

    return true;
}


template<typename T>
node<T> *BTree<T>::findleave(node<T> *cur)
{
    deque< node<T>* > Q;
    node<T> *tmp;
    bool isfind;

    if(!cur)
        return NULL;
    else
        Q.push_back(cur);

    isfind = false;
    while(!Q.empty() && !isfind)
    {
        tmp = Q.front();
        Q.pop_front();

        if(!tmp->left && !tmp->right)
            isfind = true;
        else if(tmp->left)
            Q.push_back(tmp->left);
        else
            Q.push_back(tmp->right);
    }

    if(tmp->par)
        (tmp == tmp->par->left) ? (tmp->par->left = NULL) : (tmp->par->right = NULL)
            ;

    return tmp;
}


template<typename T>
void BTree<T>::delall()
{
    deque< node<T>* > Q;


    if(root)
        Q.push_back(root);
    else
        return ;

    while(!Q.empty())
    {
        root = Q.front();
        Q.pop_front();

        if(root->left)
            Q.push_back(root->left);
        if(root->right)
            Q.push_back(root->right);

        delete root;
        root = NULL;
    }
}

template<typename T>
void BTree<T>::display(node<T> *r)
{
    if(r)
    {
        cout << r->time << '␣';
        display(r->left);
        display(r->right);
    }
}

#endif
```

List of Publications

Publications related to thesis

Jianhong Ou, Tadashi Yamada, Keisuke Nagahisa, Takashi Hirasawa, Chikara Furusawa, Tetsuya Yomo, Hiroshi Shimizu. Dynamic change in promoter activation in lysine biosynthesis of Escherichia coli cell, Molecular BioSystems, 4, 128-134 (2008 Jan)

Jianhong Ou, Takashi Hirasawa, Chikara Furusawa, Tetsuya Yomo, Hiroshi Shimizu. Analysis of stochasticity in promoter activation by using a dual-fluorescence system, BioSystems in press (2009)

Other publication

Tadashi Yamada, **Jianhong Ou**, Chikara Furusawa, Takashi Hirasawa, Tetsuya Yomo, Hiroshi Shimizu. Relationship between noise in protein expression and regulatory structure of amino acid biosynthesis pathways, IET Systems Biology in press (2009)

Acknowledgments

My foremost thank goes to my thesis adviser Dr. (of Eng.) Hiroshi Shimizu and Dr. Chikara Furusawa. Without them, this dissertation would not have been possible. I thank them for their patience and encouragement that carried me on through difficult times, and for their insights and suggestions that helped to shape my research skills. Their valuable feedback contributed greatly to this dissertation.

I am very grateful to the members of my committee, Dr. Hisao Ohtake and Dr. Masahiro Kino-oka, for their helpful comments and constructive suggestions.

I also thank Dr. Takashi Hirasawa and Dr. Naoaki Ono, who advised me and helped me in various aspects of my research. They are so kind persons that I can always count on to discuss the tiniest details of a problem.

I thank all the students in Dr. Shimizu Lab., whose presences and fun-loving spirits made the otherwise grueling experience tolerable. I enjoyed all the vivid discussions we had on various topics and had lots of fun being a member of this fantastic group.

I thank all the classmates in this special English course. We had so many memorable parties. I thank my friends in this world especially in Japan. Their names are: Sichong Chen, Guanghua He, Zhenquan Tan, Yikai Zhou, Shuoyuan Zhang, Huijian You, Xinlei Yao, Yan Li, Shaowei Huang, Siragi A. Mahmud, and Sukanda Tianniam.

Last but not least, I thank my family for always being there when I needed them most, and for supporting me through all these years.

Jianhong Ou
July 14, 2009