

User Manual

*for REBORN**

2012 iGEM USTC-Software Team

09.2012

*main software of 2012 iGEM USTC-Software Team, stands for **R**everse **E**ngineering for **B**iological **R**egulatory **N**etworks

Introduction

The software *REBORN*, which is short for **R**everse **E**ngineering for **BiO**logical Regulatory Networks. This software aims to help biological researchers to rebuild the whole Genetic Regulatory Networks¹ in a more systematic way. Team members are *Kai Kang, Kaifeng Chen, Chao Xu, Sen Jiang, Litong Jiang, Kaishuai Yang and Yunlong Zheng*. USTC software developers spent about half year discussing the topic, writing source codes, designing wiki pages, facing challenges and having fun.

As synthetic biology becomes more and more complicated, some basic steps of designing circuits may be less familiar to biological researchers. In order to help them rebuild the whole system from the bottom to the top, a top to bottom thinking pattern should be introduced. To explain this method in an engineering way, the terminology is **Reverse Engineering**.

unlike other teams of software track this year that develop searching tools or browsers, we, 2012 iGEM USTC-Software Team determines to make a difference developing software close related to the biological essence. Integrated some basic concepts or methods such as Genetic Algorithm, Quicksort Method and Machine Learning, this all-in-one software is powerful in serious computing, simulating and modeling.

Copy rights of this software is reserved to 2012 USTC-Software team. And to know more about our team, please refer to the wiki:

<http://2012.igem.org/Team:USTC-Software>.

For more information about USTC(*University of Science and Technology of China*), you can visit the website:

<http://en.ustc.edu.cn/>

¹GRN for brief

Contents

1	Overview	4
2	Algorithms & Methods	4
2.1	Genetic Algorithm	5
2.2	Quicksort Algorithm	7
2.3	Recursion Method	7
3	Database	8
3.1	1. Regulatory Information	8
3.2	2. Genome information	9
4	Example of Oscillator	10
4.1	Input: Map	10
4.2	Computation: Console	11
4.3	Viewing Model: Sandbox	12
4.4	Representing Results: Report	13
4.5	A Brief Summary	14
5	View Source Code and Document	14

1 Overview

REBORN mainly consists of four parts, and these parts connected to each other undertaking different functions. These four parts are *Map*, *Console*, *Sandbox* and *Report*.

► **Map:**

- Inputting experimental data.
- Viewing details about the maximum value and the minimum value.
- Providing interface with **Console**.

► **Console:**

- Using algorithms and methods to do the simulation and calculation.
- Showing intermediate results.
- Providing FTP service and emailing the results.
- Providing interface with **Map** and **Sandbox**.

► **Sandbox:**

- 3-D model structure of the results.
- Providing different viewing angles.
- Having connections with **Sandbox** and **Report**.

► **Report:**

- Presenting behaviors of simulated results.
- Showing details about output cells.
- Providing connections with **Sandbox**.

REBORN uses database generated from all genome information about E.coli k-12, and the information includes regulatory relations and overall information about operons, promoters, genes, terminators, promoters, RBS, 3'UTR and 5' UTR. To make the database more accessible to researchers, we also build a clotho application based on Java platform. Details about the clotho app are provided on our team's github repository:

<https://github.com/igemsoftware/USTC-Software-Clotho-App-2012>

In developing our software, some classical algorithms and methods are used. These specific algorithms and methods will be introduced in details in following sections.

2 Algorithms & Methods

In order to make the computational process less time consuming, some advanced algorithms and classical methods are employed in the software. These algorithms and methods include Genetic Algorithm, Quicksort Algorithm and recursive method. To meet need of the common biological researchers, some innovative and original improvements have been proposed by our team to make the computing time much less and the results much more various.

2.1 Genetic Algorithm

Genetic Algorithm resembles the natural way of evolution. This algorithm is routinely used to search for optimal solutions of certain problems, belonging to the larger class of Evolutionary Algorithm. Thus, some terminologies are borrowed from natural evolution process such as generation, mutation, selection and inheritance.

A typical Genetic Algorithm should have the following requirements:

- a genetic representation of the solution domain.
- a fitness function to evaluate solution domain.

The method in classical Genetic Algorithm is to reduce the whole process into several generations. After each generation, the result will get closer to the optimum one. Every generation contains two phases: growth phase and selection phase. These two phases mimic natural growth process and selection process. At the very beginning of the Genetic Algorithm, a group of cells are initialized randomly with genes, proteins and reactions between them. Each cell represents a mathematical model, and in our case, this model is presented in Ordinary Differential Equations (ODE). These ODEs can be generated by listing all the reactions in each cell using chemical master equations. Then the evolutions begin.

Here is the introduction to classical phases.

1. Growth Phase

Assuming there are n cells in the initialization step, then in growth phase all these cells will duplicate themselves. The duplicated cells will go through randomly mutation process. In our software, these mutations include:

- 1) Changing the degradation rate of a randomly chosen protein.
- 2) Changing the kinetic constant of a randomly chosen reaction.
- 3) Adding a new gene, in the meantime the corresponding protein should be added as well, including the transcriptional reaction and the degradation reaction of the protein.
- 4) Adding a regulation. Namely adding a regulatory reaction between a gene and a protein, and the kinetic constants are randomly drawn in $(0, 1)$.
- 5) Adding a post transcriptional reaction. As for this case, there are several kinds of modifications.
 - 5.1) $A + B \rightarrow AB$: Binding between two proteins.
 - 5.2) $A + B \rightarrow A$: catalytic reaction.
 - 5.3) $A + BC \rightarrow B$: partial catalytic.
 - 5.4) $A \rightarrow A^*$: post transcriptional modification.
 - 5.5) $B \rightarrow B$: partial degradation.

After a generation, the $2 \times n$ cells will go to another phase.

2. Selection Phase

The fitting function, or score function, is used to evaluate each cell according to the mathematical model in each one. Before this phase begins, the ODE model of each cell should be firstly extracted. From all the ODEs, the software can get the simulated data. The score function here is to give each cell a score by comparing the simulated data and the input data. To make it clear in a mathematical way, the scoring function is:

$$f(x) = \sum_{i=0}^N \sum_{j=0}^M (X_{ij} - X'_{ij})^2$$

or

$$f(x) = \sum_{i=0}^N \sum_{j=0}^M |X_{ij} - X'_{ij}|$$

X_{ij} represents the input data of the i th species at the j th time point, and the X'_{ij} represents the calculated data of the i th species at the j th time point. So these two functions actually calculate the variance of simulated behavior and the input behavior.

Thus after each cell are given a score, Quicksort Algorithm is utilized in ranking all the $2n$ cells. Then the first n cells will remain for the next generation, the other cells will be eliminated.

That is a generation of the whole Genetic Algorithm. Then the group of cells will go through several hundred or thousands of generations. The final several cells are assumed to the cells with the best behaviors.

Our Improvements

By merely using classical Genetic Algorithm is proved to be slow in computational process. As for some steps in the algorithm, some cells with potentials may be eliminated due to the low score at that generation. Moreover, the original algorithm does not consider cells with different topology levels and does not sort the cells by complexity, either. Thus, to avoid these problems, we make some important changes to the algorithm, and the results prove to be better and more practical than the classical one.

1. We classify mutations into two kinds: mutation of topology and mutation of parameter. Topology mutation only contains mutation type 3—adding a new gene. In order to preserve the potential properties of each cell, for some generations in the grown phase cells can only have topology mutations, and then the cells will undergo parameter mutations. This special design of Genetic Algorithm helps preserve cells with potentials. In addition, we can thus provide cells with different complexity levels.

2. To avoid the results concentrated around the local optimum, we initialize several groups of cells with different initial conditions, and we duplicate and selection cells in each group. With this improvement, we will get cells with different behaviors, providing users with more choices.

2.2 Quicksort Algorithm

Quicksort Algorithm has been ranked among top 10 algorithms by *IEEE*. Quicksort method is mainly used in the case of sorting or retrieving. Its properties like time and computational space saving makes it outstanding among all the sort methods, and that is why it is so commonly used in simulations of nearly all the fields.

The core of Quicksort Algorithm is a recursive method which reduces the complexity of the problem step by step, and the final step should be a simple case easy to deal with. Recursion makes the problem less difficult to understand and more feasible in programming.

In simple pseudocode, the algorithm can be expressed as this:

```
function quicksort(array)
  If length(array) <=1
    return array
  select and remove a pivot value pivot from array
  create empty list less and great
  for each x in array
    if x <= pivot then appendix x to less
    else appendix x to great
  return concatenate(quicksort(less), pivot, quicksort(great))
```

In our case, the array stands for the scores of all the cells. Here we use Quicksort Algorithm to rank all the cells according to their scores, and then the first several cells will be selected waiting for the next generation.

2.3 Recursion Method

As we integrate all the regulation in a matrix, and to make the database compatible with the software, we use a recursion method to find the regulatory matrix in the database matrix. After finding the matrix, we then can used the parts in the database to rebuild the system.

This recursive method aims to find small matrix in a big matrix, and recursion makes it universal to every case. Imagine finding an $m \times m$ matrix in a $n \times n$ matrix ($n \geq m$), the finding steps are as follows:

```
FindMatrix(n*n database matrix, m*m target matrix)
If (m ==1 )
  Return target matrix
```

```
Else findMatrix( n*n database matrix, (m-1)*(m-1) target matrix)
Return choices
```

Computational tests have proved that this recursion method is efficient and time saving in finding matrix. And as for our database, two kinds of database matrix of 773*773, 549*549 are generated on users choice.

3 Database

Since this year our software has the functions of rebuilding the biological system by providing proper regulatees and regulators to users, a huge database containing comprehensive information about regulation is of great need. In order to make the calculated results experimentally feasible, we use in vivo data generated from Regulon Data Bank, a website providing all genome information about *E.coli. K-12*. Thus, our database includes all the information of *E.coli. K-12*. This is the very first time in iGEM competition that a team uses such a huge database, especially the massive regulatory information, in their software.

The database consists of two parts: first part stores regulatory information of operons to operons and genes to genes, and the second part contains genome information about operons, genes, promoters, terminators, RBS, 5 UTR and 3 UTR.

To present the unique features of our database, we name it "Regulon Lib".

3.1 1. Regulatory Information

These two matrices share some basic properties. The regulatory direction is from the regulators in the *i*th row to the regulatees in the *j*th column (*i, j* are row and column indices), and the element at (*i, j*) represents the regulatory relations. In our case, several kinds of regulatory relations are considered shown as follows:

- 1: positive regulation.
- -1: negative regulation.
- 0: no regulation.
- 2: both positive and negative regulation.
- -2: unknown regulation.

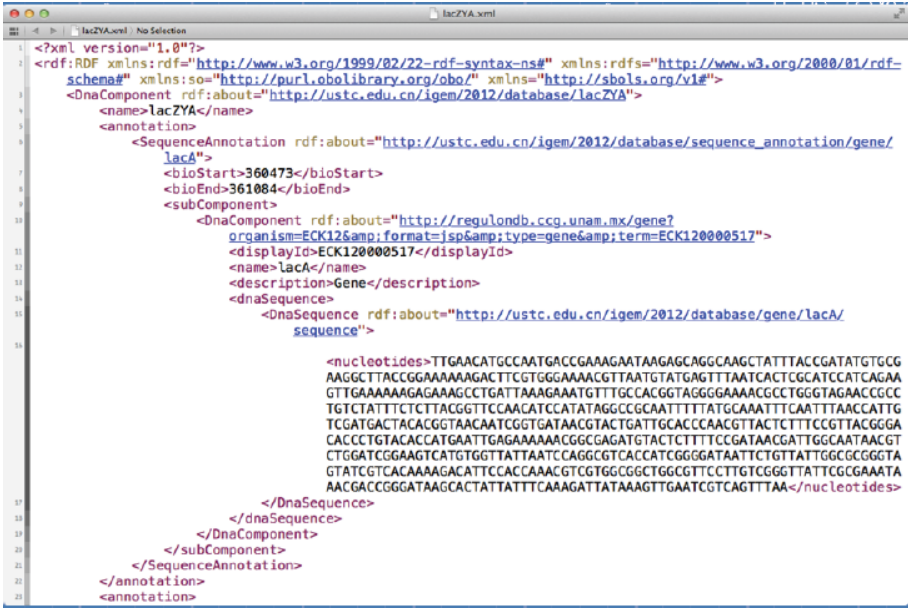
As to the operon-operon matrix, it has dimension of 549×549 and gene-promoter matrix is 773×180 . Recursive method is employed to find proper regulators and regulatees.

3.2 2. Genome information

Genome information of operons, genes, promoters, RBS, terminators, 5 UTR and 3 UTR are stored in over 10000 SBOL format files. Some details are listed below:

- 2652 Operons, along with names and components (gene, promoter, terminator) for each.
- 4554 Genes, along with name, identifier, sequence, start position, end position, start codon and end codon for each.
- 3734 Promoters, along with name, identification ,sigma factor and sequence for each.
- 207 Terminators, along with identifier, sequence and PubMed ID for each.
- 179 RBS, along with identifier, start position, end position and sequence for each.
- 179 RBS, along with identifier, start position, end position and sequence for each.

Here is a screenshot of a SBOL file: Based on this unique and comprehensive



```

1 <?xml version="1.0"?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-
3 schema#" xmlns:so="http://purl.obolibrary.org/obo/" xmlns="http://sbols.org/v1#"
4 <DnaComponent rdf:about="http://ustc.edu.cn/igem/2012/database/lacZYA">
5   <name>lacZYA</name>
6   <annotation>
7     <SequenceAnnotation rdf:about="http://ustc.edu.cn/igem/2012/database/sequence_annotation/gene/
8     lacA">
9       <bioStart>360473</bioStart>
10      <bioEnd>361084</bioEnd>
11      <subComponent>
12        <DnaComponent rdf:about="http://regulondb.ccg.unam.mx/gene?
13        organism=ECK12&format=jsp&type=gene&term=ECK12000517">
14          <displayId>ECK12000517</displayId>
15          <name>lacA</name>
16          <description>Gene</description>
17          <dnaSequence>
18            <DnaSequence rdf:about="http://ustc.edu.cn/igem/2012/database/gene/lacA/
19            sequence">
20              <nucleotides>TTGAACATGCCAATGACCGAAAGAATAAGAGCAGGCAAGCTATTTACCGATATGTGG
21              AAGGCTTACCGAAAAAGACTTCGTGGGAAAACGTTAATGTATGAGTTTAATCACTCGCATCCATCAGAA
22              GTTGA AAAAGAGAAAGCCTGATTAAAGAAATGTTTGCCACGGTAGGGGAAAACCGCTGGGTAGAACCGCC
23              TGTCTATTTCTTACGGTTCCAACATCCATATAGGCCGCAATTTTATGCAAAATTC AATTTAACCATTG
24              TCGATGACTACACGGTAACAATCGGTGATAACGTAAGTACTGATTGCACCCAACGTTACTCTTTCCGTACGGGA
25              CACCCTGTACACATGAATTGAGAAAAACGGCGAGATGTAAGTCTTTTCCGATAACGATTGGCAATAACGT
26              CTGGATCGGAAGTCATGTGTTAATTCAGGCGTCAACATCGGGGATAAATTCGTGTTATGGCGGGTA
27              GTATCGTCAAAAAGACATTCACCAACGTGTGGCGGCTGGCGTTCCTGTGCGGGTTATTCGGGAAATA
28              AACGACCGGATAAGCACTATTATTTCAAGATTATAAAGTTGAATCGTCAGTTTAA</nucleotides>
29            </DnaSequence>
30          </DnaComponent>
31        </subComponent>
32      </SequenceAnnotation>
33    </annotation>
34  </DnaComponent>
35 </annotation>
36 </rdf:RDF>

```

Figure 1: An example of SBOL file of *LacZYA*

database, we also build a clotho application with the same name Regulon Lib.

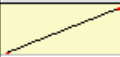
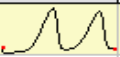
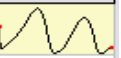
4 Example of Oscillator

In this section, a specific example of oscillator will be explained step by step. Each steps of our software will fulfill some kinds of function, and in the final, a model of oscillator will be provided to users.

4.1 Input: Map

Now an oscillator is to be the example in this user manual. Imagining a oscillating behavior of two proteins, the classical model of this is that two genes repress each other and themselves. But the kinetic constants is hard to infer. First an oscillation experimental data is imported, shown in the picture below:

The screen shot is pasted on the right side: It is easy to say the oscillation behaviors of two proteins. The input interface of the software is compatible with this kind of data, which put time points in the first column and the expression of proteins or inducers in the next columns, and before the time courses, users should tell the software how many time points are there and how many species of proteins and inducers are in the system.

	A(X)	B(Y)	C(Y)
Sparklines			
1	1	1	1
2	2	0.09173	0.21375
3	3	0.09431	0.16889
4	4	0.09737	0.18045
5	5	0.1011	0.20979
6	6	0.10554	0.24651
7	7	0.11096	0.28549
8	8	0.11756	0.32405
9	9	0.12547	0.36131
10	10	0.1348	0.39744
11	11	0.14571	0.43303
12	12	0.15867	0.46874
13	13	0.17372	0.50513
14	14	0.19162	0.5427
15	15	0.21256	0.58187
16	16	0.2374	0.62297
17	17	0.26698	0.66627
18	18	0.30227	0.712
19	19	0.34446	0.76032
20	20	0.39512	0.81133
21	21	0.45615	0.86504
22	22	0.52971	0.92142

To say more about the input interface, sometimes a system may have two distinct time courses with different protein expression amounts. To make this case also feasible in our software, we also provides multi-input interface for users to enter more than one experimental time courses for the same system.

After inputting the data, Map will show the input behavior on the right side, and if users click on each line, some details will appear at the left side. These details include the time course points, the maximum value of the curve and also the minimum value.

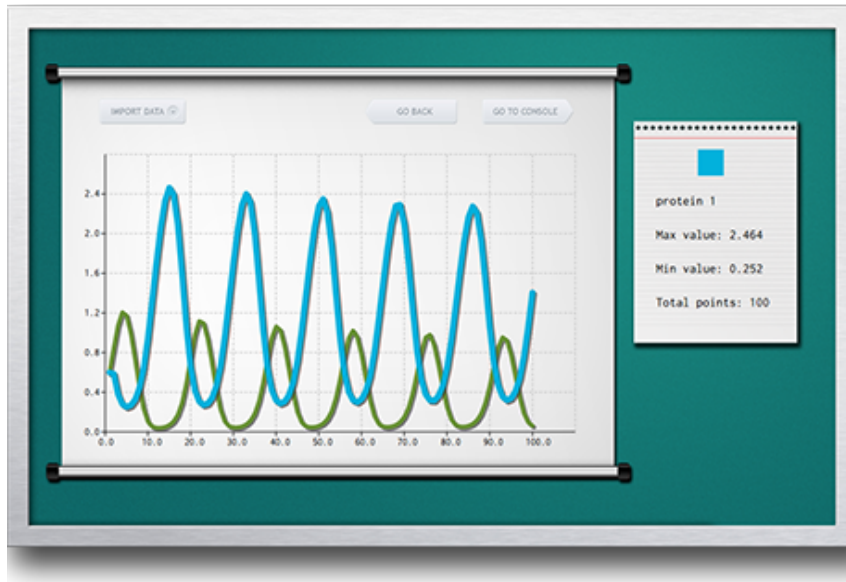


Figure 2: GUI of **Map**

4.2 Computation: Console

After inputting the data, users can use our algorithms and methods to rebuild the system. As for these computational processes, **Console** is the software integrating all the calculations. Users can click on *GO TO CONSOLE* in **Map** to do the reverse engineering processes. The GUI of **Console** is pasted below: Console will all these tasks:

- Using Genetic Algorithm to generate ODE models.
- Using ODE models of each cell to generate the GRNs.
- From the GRNs, regulatory matrix is extracted.
- Finding regulatory matrix in the database matrix.
- Waiting for outputting the results.

The right panel of **Console** is where users can choose their own preferred parameters or use the software default parameters.

Since the calculation may be really time consuming, like huge systems with thousands of cells in the initial step of Genetic Algorithm and thousands of



Figure 3: GUI of **Console**

time course points, the software **Console** also provides some other services such as FTP and email to help users save waiting time. Users can click on FTP service and when the calculation ends, output data will be sent to an FTP cloud on internet. Thus, users can view the results at any place and any time. As for the oscillation behavior, here are some tips about choosing the optimum parameters in order get the result as quick as possible.

1. The parameter of number of cells will determine the initial step of Genetic Algorithm, our experience shows this number can be several hundreds.
2. The parameter of cells survive will determine the cells which will not change topology after a generation. This number should be less than 10.
3. The whole calculation time of getting a good result of two-protein oscillation will take about 10-20 minutes in ordinary personal computers.

4.3 Viewing Model: **Sandbox**

Figure 4 is the GUI of **Sandbox**: Unlike other iGEM Software teams only concentrate on the feasibility of the software, we also include some technics in visualization that can be used in our software. To make this part the most user friendly ever, OpenGL C++ library is used in writing this part. Thus this part provides really excellent experience for users to view the whole results. By using the powerful OpenGL lib, some unique features are the first time appearing in iGEM software tracks.

- A map showing the regulatees, regulators and regulations between them.

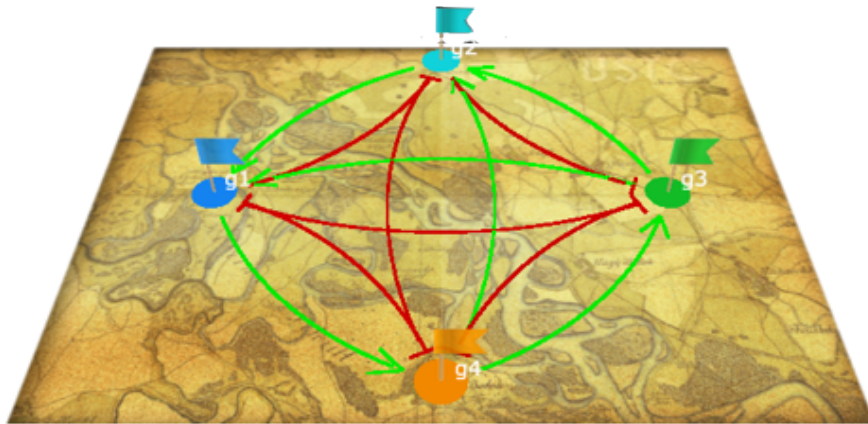


Figure 4: GUI of **Sandbox**

- Red line representing negative regulation and green line representing positive regulation.
- Scrolling the mouse to change the view angle and the perspective.
- Connecting to **Report** and **Console**.

As for the oscillation, the calculated model from our algorithms and methods are shown in Figure 4. And it is exactly the same with the classical model. This means our software works well with this problem. Some other kinds of models of oscillations will be generated as well, users can choose the tags in the bottom to change the viewpoint of the cells.

4.4 Representing Results: Report

After all the calculations in **Console**, software **Report** has the ability to show the results of each cell, namely each model. The oscillation results are shown in Figure 5. The oscillation behavior is shown at the right side, even though the calculated time courses are not exactly the same with the input data, but the properties of the model, such as the period of each oscillation, is the same with the input. Thus, the whole process ends. Only from the input oscillation behaviors we get all the models that can generate these behaviors. This is what the whole reverse engineering process in our software *REBORN* is like. Some SBML format files which include all the information in each cell will be exported to computer folds for users to check for their later use.

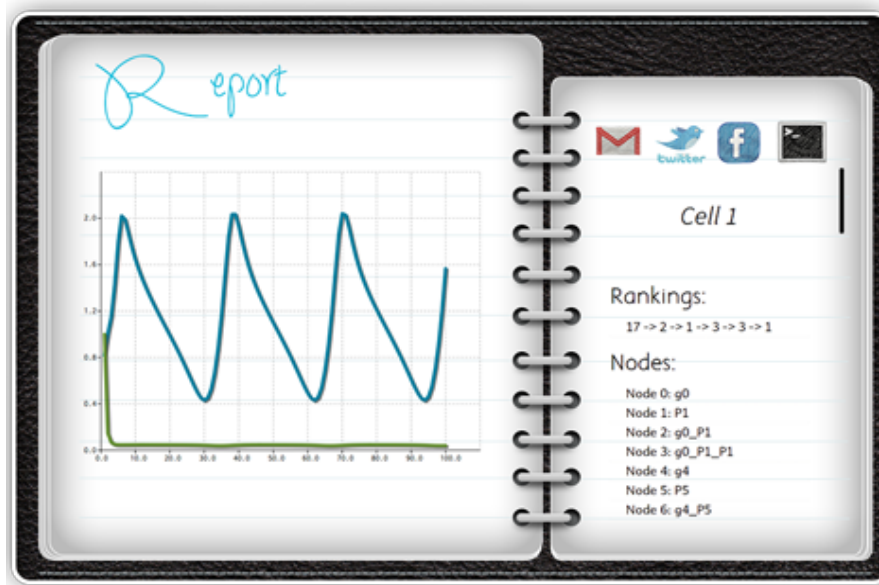


Figure 5: GUI of Report

4.5 A Brief Summary

In summary, we have fulfilled all the following tasks:

- ▶ Providing an one-in-all software dealing with reverse engineering processes.
- ▶ Powerful in use with creativity and originality.
- ▶ Working well with given examples.

We also use some standards in iGEM competitions, such as:

- ▶ Using SBML to store reaction information of each cell.
- ▶ Generating a huge and comprehensive database with all regulatory information in *E.coli.K - 12*.
- ▶ Using SBOL to store database.

5 View Source Code and Document

To meet the iGEM spirit, we make our software an open source one. And all the database, source code and executes can be downloaded from the github repository, here is the link:

https://github.com/igemsoftware/USTC-Software_2012

And for this user manual, users can find this in our team wiki page:

<http://2012.igem.org/Team:USTC-Software>

USTC has participated in iGEM competitions since 2007 and participated in Software Track since 2009. All these years USTC software developer aims to solve big problems not only in simulations in synthetic biology but also in experiments in biology. These well done jobs have been recognized by iGEM competitions, USTC software teams and other users. This year *REBORN* will be a brand new one with totally different concepts and USTC software team will never disappoint you.

All the source code and documents belong to 2012 iGEM USTC-Software, and the database should belong to **Regulon DataBank**, from which we generate the compatible database.