12-2016

Using Software Testing Techniques to Infer Biological Models

Mikaela Cashman
University of Nebraska-Lincoln, mikaela.cashman@gmail.com

USING SOFTWARE TESTING TECHNIQUES TO INFER BIOLOGICAL MODELS

by

Mikaela Cashman

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Myra B. Cohen

Lincoln, Nebraska

December, 2016

USING SOFTWARE TESTING TECHNIQUES TO INFER BIOLOGICAL MODELS

Mikaela Cashman, M.S.

University of Nebraska, 2016

Adviser: Myra B. Cohen

Years of research in software testing has given us novel ways to reason about and test the behavior of complex software systems that contain hundreds of thousands of lines of code. Many of these techniques have been inspired by nature such as genetic algorithms, swarm intelligence, and ant colony optimization. However, they use a unidirectional analogy – taking from nature without giving back.

In this thesis we invert this view and ask if we can utilize techniques from testing and modeling of highly-configurable software systems to aid in the emerging field of systems biology which aims to model and predict the behavior of biological organisms. Like configurable systems, the underlying source code (metabolic model) contains both common and variable code elements (reactions) that are executed only under particular configurations (environmental conditions), and these directly impact an organism's observable behavior. We propose the use of sampling, classification, and modeling techniques commonly used in software testing and combine them into a process called BioSIMP which can lead to simplified models and biological predictions.

We perform two case studies, the first of which explores and evaluates different classification techniques to infer influential factors in microbial organisms. We then compare several sampling methods to limit the number of experiments required in the laboratory. We show that we can reduce testing by more than two thirds without negatively impacting the quality of our models. Finally, we perform an end-to-end case study on BioSIMP using both laboratory and simulation data and show that we can find influencing environmental factors in two microbial organisms, some of which were previously unknown to biologists.

Our findings suggest that the configurable-software analogy holds, and we can identify the variable and common regions of reactions that change with respect to the environment.

DEDICATION

This thesis is dedicated to my mother Kia Cashman, partner John McDevitt, and feline children Leela and Romana Cashman.

ACKNOWLEDGMENTS

First I would like to thank my advisor and mentor Dr. Myra Cohen. I am at the University of Nebraska due to your enthusiasm and confidence in me. Thank you to the other members of my committee: Dr. Nicole Buan, Dr. Massimiliano Pierobon, and Dr. Matthew Dwyer for your interest in my work, and time in reviewing it.

I would like to thank my research team Dr. Myra Cohen, Dr. Nicole Buan, Dr. Massimiliano Pierobon, Dr. Christine Kelley, Jennie Catlett, and Zahmeeth Sakkaff. We began this work all speaking different academic languages, and have brought them together to form something new and exciting. I would like to extend a special thanks to Jennie Catlett, you have been an inspiration and a mentor to me.

I would also like to thank my ESQuaReD neighbors and those of you who helped keep me sane especially: Natasha, Mitch, Eric, Justin.

From my alma mater, Coe College, I want to thank Jeremy for always being my data hotline specialist. Thank you Dr. Jon White for your encouragement and semi-annual check-up calls that always make my day.

Finally I would like to thank my family for being so supportive. Mom your work ethic is a constant inspiration. Adam and Josh, thank you for showing me it is okay to be a nerd. Grandma, your thirst for knowledge has always been inspiring and I am so very proud to take after you. Thanks John for your endless support, encouragement, and love. Thanks Leela and Romana for reminding me to take moments for myself and being my personal heater.

This work was supported in part by National Science Foundation Grants CCF #1161767 and CNS #1205472.

Contents

Chapter 1

Introduction

Software testing research has produced many techniques to reason about and test the behavior of software systems. Software systems can be complex, having hundreds of thousands of lines of code, and can be highly-configurable. Highly-configurable software contains portions of code (features) that can be turned on or off in varying combinations depending on user preferences and environmental conditions. For example, the web browser Firefox has over 1,900 settings [29] related to security, search, sync, privacy, and plug-ins. We can underestimate the number of options by considering all preferences to be binary, and assuming no constraints this gives us over 2^{1900} configurations. We have one software system (browser) and multiple instances based on the features selected (configurations). This feature-oriented view of software [4] allows us to model, understand, and validate programs by manipulating common and variable code separately and by identifying sets of features which influence unique or undesirable behavior.

Configurability of systems adds a layer of complexity to software, and as such, a large body of research has focused on testing these systems [15,33,35,58,66,71]. Some of this research has turned to heuristics to help sample large configuration spaces [15, 25,47,48,51]. Many of these sampling techniques are derived from natural phenomena such as genetic evolution [60], intelligent swarming [68], ant colony optimization [10], etc. This is no surprise, since complex programs share traits with natural systems such as those found in biology. However, this approach of using biology to help us reason about software has been unidirectional.

In fact, recent work in systems and synthetic biology has started to look in the direction of computer science to understand how living cells can be used to perform work that is designed and programmed by humans [18, 30, 34]. Being able to control and program cells will aid in the ability to create better biofuels [21, 38], understand and control human health and disease [59, 64], increase food production on marginal lands [12], control the global climate [72], and even potentially terraform Mars [27,57]. Research in systems biology aims to model, predict and program the behavior of organisms under specific environmental conditions (food sources, media composition, light, temperature, etc.) [36]. Synthetic biology works to physically (artificially) insert code segments into DNA to effectively program biological organisms [30].

The state of the art in microbial modeling is to utilize manually-curated models which have been meticulously modified by biologists to reflect current literature and experimental results [6, 20, 32]. A model for *E. coli* can be seen in Figure 1.1. This bacterial network contains more than 1,000 reactions (nodes) and 60,000 possible pathways through the set of reactions [5], which is too complex to analyze in an ad-hoc manner. Trying to predict or understand the behavior of this organism by tracing the graph would be like trying to manually extract meaning from a program with thousands of lines of code. As a result, we are still unable to predict or control biological organisms well enough to leverage their vast capabilities. Furthermore, faults in the design and failures in the predictions can have significant ecological and health consequences [65].

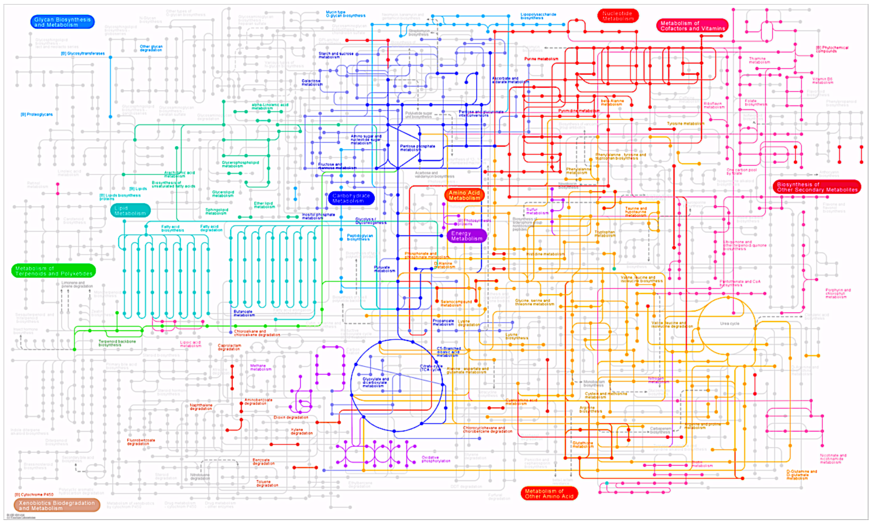
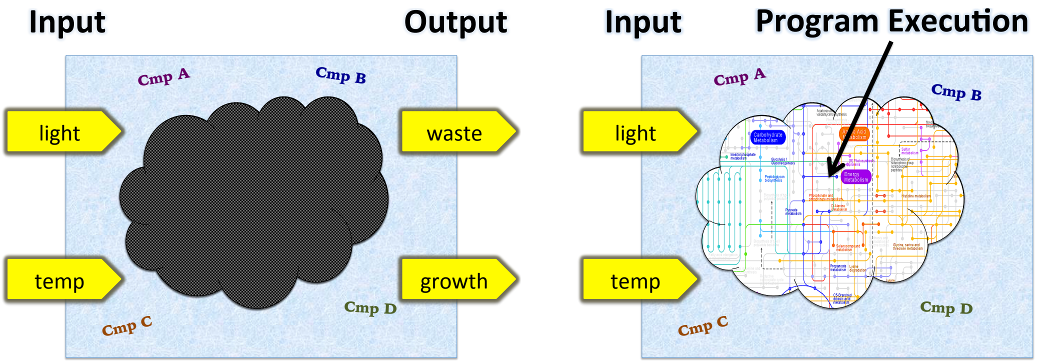

Glycan Biosynthesis and Metabolism
Nucleotide Metabolism
Metabolism of Cofactors and Vitamins
Biosynthesis of Other Secondary Metabolites
Lipid Metabolism
Carbohydrate Metabolism
Amino Acid Metabolism
Energy Metabolism
Metabolism of Terpenoids and Polyketides
Xenobiotics Biodegradation and Metabolism
Metabolism of Other Amino Acids

Figure 1.1: Metabolic Pathway Map for *E. coli*

Software testing techniques provide us with systematic methods to sample and infer across a similarly complex space. In this thesis, we utilize software testing techniques to model and understand biological systems. We view biological organisms as highly-configurable software systems. The underlying code of the organism is its *genotype* and the output (observable characteristics of the organism) is its *phenotype*. We can alter the inputs of the system, and observe how the program (organism) changes. These changes can be external (phenotypic) or internal (reaction coverage). An example of external variability can be see in Figure 1.2a, where the inputs are light, temperature and media (liquid or solid environment in which the organism lives), and the outputs are waste and growth. In this case we can view the organism itself as a blackbox, ignoring the code within. On the other hand, in internal variability (Figure 1.2b), we look at the code in the program (organism) and view how its execution changes with various inputs. Both of these methods can reveal insights



(a) External Variability

(b) Internal Variability

Figure 1.2: Biological system

about the system. We can also combine them to understand how the inputs change the reactions, and how the change in reactions affects the phenotype.

Based on this intuition, we have developed a software testing process called BioSIMP (for Biological Sampling, Inference, Modeling, and Prediction), which samples, tests, and classifies the inputs to infer *influential factors* — those factors that have an impact on the phenotypic outcomes — and model the commonality and variability of the genomes leading to these behaviors. BioSIMP then produces models to be used in future predictions. In this thesis we first investigate a selection of classification and sampling algorithms that can be used in BioSIMP. We find that classification using discrete labels and decision trees works best overall. We also evaluate BioSIMP end-to-end on two real organisms extracted from the human gut, both in a laboratory setting and via simulation, and discover some previously unknown factors that may impact their growth in a range of environmental conditions. We find that our software analogy has uncovered some subtle — yet interesting — properties of biological organisms that may lead to novel software engineering techniques and new ways to view configurable software.

The contributions of this thesis are:

1. An analogy for Highly-Configurable Biological Systems;
2. A process, BioSIMP, that samples, identifies influencing factors, and then models biological organisms so that we can predict (and eventually control) their behavior;
3. Exploration of a selection of sampling techniques and classification algorithms;
4. A case study on two biological organisms that shows our analogy holds and may lead to interesting biological predictions;
5. A reaction coverage variability model.

The rest of this thesis is organized as follows. Chapter 2 covers background material on software testing, machine learning, and systems biology, along with related work. Chapter 3 describes the process BioSIMP with a running example to guide the reader. Chapter 4 details a case study to evaluate classification and sampling algorithms. Chapter 5 contains the full case study of BioSIMP on two organisms *in silico* and *in vitro*. We end with conclusions and future work in Chapter 6.

Chapter 2

Background and Related Work

We now discuss background and related work on software testing (and its influence from nature), classification, and systems biology. We present a running example to demonstrate these topics. We end the chapter with a discussion of related work in the combination of these topics.

2.1 Software Testing

Testing software is an integral part of any software engineering project. From a report by the National Institute of Standards and Technology (NIST) [62]:

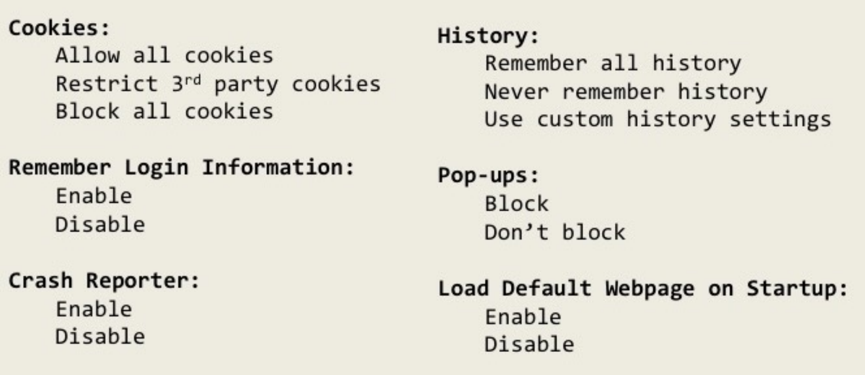
> Based on the software developer and user surveys, the national annual costs of an inadequate infrastructure for software testing is estimated to range from $22.2 to $59.5 billion. Over half of these costs are borne by software users in the form of error avoidance and mitigation activities. The remaining costs are borne by software developers and reflect the additional testing resources that are consumed due to inadequate testing tools and methods.

Software testing is clearly important to save cost and prevent major failures. Most real systems however, have too many configurations to enumerate and fully test. For instance the Linux kernel has been reported to have over 5,000 configuration options [54] leading to more than 2^{5000} configurations, while programs such as the GNU compiler, gcc have been reported to have as many as 10^{61} configurations [15]. Given the complexity of configurable software, techniques have been developed to sample and characterize faults during testing.

One sampling technique that has been used extensively for sampling configurable software — and is used in this thesis — is combinatorial interaction testing (or CIT) [14, 15, 25, 47]. CIT samples broadly and systematically across factors (features) by generating small (optimized) samples that cover all t-way combinations of factors in at least one configuration. Underlying most CIT sampling is a mathematical object called a *covering array* (CA) which defines the sample. The variable t is called the strength and determines how broadly we sample. The most common sampling used in software is $t = 2$ or pairwise testing. In pairwise testing all pairs of features appear at least once in the sample.

Many algorithms and tools have been developed to find CIT samples such as a one row at a time greedy algorithm, the Automatic Efficient Test Case Generator (AETG) [14], the In Parameter Order General (IPOG) algorithm [39] implemented in ACTS [1], or simulated annealing implemented in CASA [25]. Other variations exist which use a variety of heuristic or meta heuristic techniques such as ant-colony optimization, tabu search, and genetic algorithms. Most existing CIT algorithms will build samples of strength higher than 2, however most are optimized to work well at lower strengths (2 or 3) since the literature has shown that software tends to have mostly low order interactions [37]. Other sampling techniques have been recently proposed for performance testing [52, 55, 56], which we investigate in Chapter 4.

Let us look at an example. Suppose we have a browser that has the settings in Figure 2.1. Each setting can have a certain number of options (e.g. the setting *cookies* has three options: allow, restrict, or block). If we assume there are no constraints in this model we can calculate the number of all combinations of the possible options. Since we have three options for *cookies* and two for *remember login information* these settings together make $3 \times 2 = 6$ configurations. If we continue with this logic, we get $6 \times 2 \times 3 \times 2 \times 2 = 144$ possible configurations. This is too many test cases to run by hand efficiently. We can use CIT to sample this space instead. Since literature has shown that software tends to have mostly low order interactions, we will test this system with a strength of 2 [37]. Let us assume there is a failure if you have *all cookies blocked* and *enable loading of a default webpage on startup*. The failure could be that the default page required cookies to load. We would only see this failure occur if we tested these two options together.



Cookies:
- Allow all cookies
- Restrict 3rd party cookies
- Block all cookies

Remember Login Information:
- Enable
- Disable

Crash Reporter:
- Enable
- Disable

History:
- Remember all history
- Never remember history
- Use custom history settings

Pop-ups:
- Block
- Don’t block

Load Default Webpage on Startup:
- Enable
- Disable

Figure 2.1: A specification of settings for a web browser.

Using CASA, one possible sample can be seen in Table B.4 which contains only 9 of the 144 possible configurations. This is a much more manageable number of combinations to test. In using CIT, the size of the test suite increases logarithmically with the number of settings [14]. In this sample Tests 1 and 9 will find the failure. Note in this case that this pair is actually covered twice.

Table 2.1: A 2-way covering array for browser example.

Test	Cookies	Login	Crash	History	Pop-up	Load Default
1	Block	Enable	Disable	Custom	Block	Enable
2	Restrict	Disable	Disable	Custom	Don't block	Disable
3	Restrict	Enable	Enable	All	Block	Enable
4	Restrict	Enable	Enable	Never	Don't block	Enable
5	Allow	Disable	Disable	Never	Block	Disable
6	Allow	Disable	Enable	Custom	Block	Enable
7	Allow	Enable	Disable	All	Don't block	Disable
8	Block	Disable	Enable	Never	Don't block	Disable
9	Block	Disable	Enable	All	Don't block	Enable

2.1.1 Biology's Impact on Software Testing

We discuss an analogy between biology and software in this thesis. Examples of algorithms that were inspired by biology include *Genetic Algorithms*, and *Ant Colony Optimization*. These are examples of heuristic optimization algorithms that allow us to approximate a solution, as the best answer may be difficult to obtain or computationally infeasible.

Genetic Algorithms are based off of evolution [26]. In evolution we start with an initial population; that population breeds causing an intermix of DNA, then repeats this process. Over time, species evolve to optimize themselves to the environment. In software engineering we can use this model to try to search for a set of parameters that optimize performance, for example. In this method we can start with an initial valid set of parameters, mate them to produce a mixture of new solutions, and then mutate them to add in a factor of randomness. An example of a mutation could be randomly selecting a setting and altering it. We can then evaluate each set and choose the *best* to mate for the next iteration. This algorithm has inspired the creation of a JUnit test generation tool called EvoSuite [19].

Another algorithm inspired by biology is Ant Colony Optimization [16]. The ant colony algorithm is a technique to find the optimal path through a graph. In nature, ants will begin a search for food by randomly going down a path. If an ant finds a food source, they will bring some food back to the colony and lay down a pheromone trail. Other ants will sense this trail and follow this path (to food). The pheromones do eventually dissipate, which allows the colony to move onto another food source. This algorithm has been used in test case generation [10] and test case prioritization [45].

2.2 Classification

Classification is a technique from machine learning. Machine learning is built upon the field of statistics for data analysis bringing in logic from computational sciences and mathematics to form concrete predictions and models. Machine learning is essentially teaching computers how to learn by providing an algorithm to define the learning process and a set of *training data* to learn from [44]. Once a model has been learned, the computer can incrementally build upon its models to keep learning as new data comes in. The model can also be used on new *testing data* to make predictions.

In this work we will focus on the concept of *classification*. Classification is the problem of building a model on collected data, then using this model to make a prediction on a new observation and learn relationships between attributes. Most classification algorithms work by splitting up the training data by *attributes*. These attributes can be defined by the user or learned. Each attribute will correspond to some property of the problem. For example, Yilmaz et al. [70] use configurations of a system as the attributes, and whether or not the systems successfully completes (PASS) or results in a failure (ERROR). The relationships between attributes and between attributes and class labels can tell us a lot about the problem at hand.

2.2.1 Decision Trees

Decision Trees are one type of classification algorithm. The training data will consist of a set of attributes and class labels. Each node in the tree represents an attribute, and each edge from that node represent a possible value of that attribute. The leaves of the tree are class values. A new data instance can start at the root of the tree and follow the path that represents its attributes to predict its class value. We can evaluate a decision tree based on the accuracy (Equation 2.1) of the data reserved for testing.

$$Accuracy = \frac{\#\text{ correctly classified}}{\text{all instances}} \times 100\% \tag{2.1}$$

The algorithm to build a decision tree looks at each attribute and chooses the one with the highest *information gain*. Information gain is the change in entropy. Entropy (or disorder) means how much difference there is in the data. If by splitting the data by an attribute we create more consistent division of the data, this is good, we decrease entropy and increase information gain. The pseudocode for this algorithm can be found in Algorithm 1. We walk through the calculations on entropy and information gain below.

$$\text{Entropy}(A) = \sum_{i=1}^{c} -p_i \; log_2 \; p_i \tag{2.2}$$

Entropy of a node is formally defined in Equation 2.2 where c is number of different classes in our problem and p_i is the proportion of DATA belonging to class i. For example, in a problem with two classes, if a node has an equal number of each class the entropy would be $\sum_{i=1}^{2} -\frac{1}{2}\,\log_2\,\frac{1}{2} = 1$ which is the highest (worst) entropy we can get. This makes sense, because if we are standing on this node, the data is not distinguishable even though they belong to different classes.

To figure out the information gain of an attribute we need to calculate if the new entropy (assuming we split on that node) is less. Therefore the equation for information gain of an attribute A is defined by Equation 2.3.

$$\text{InfoGain}(A) = \text{Entropy}(A) - \sum_{v \in Values\{A\}} \frac{|A_v|}{|A|} \text{Entropy}(A_v) \tag{2.3}$$

We start with the existing entropy of attribute A, then subtract the new entropy. The new entropy is a weighted combination of the entropies for each value A could have, with the weight being how many instances fit that value. We will demonstrate with an example below.

Algorithm 1 *Decision_Tree*(DATA)

```
List $\mathcal{A}$ = all attributes
if $\forall d \in$ DATA: $C(d) = c$ then                ▷ Check for a leaf
    Create a leaf of $c$ return
for each attribute $a \in \mathcal{A}$ do               ▷ Calculate InfoGains
    Calculate InfoGain($a$)
Create a decision node for the largest InfoGain($a$)   ▷ Choose best node
for each value $v$ of $a$ do
    Decision_Tree(DATA satisfying $v$)                 ▷ repeat for each new edge
```

Let us go back to the browser example and see if we can generate a classification tree to show us what feature(s) will cause a failure, and if we can learn the nature of their interaction in the case of multiple features. We will use the 144 tests as our *training data*. Instead of using *test data*, we will analyze the model produced to infer about the cause of the failure. First we need to fill in the class labels for each of the data instances. Based on our assumption of the cause of the failure, any configuration which has *all cookies blocked* and *load default webpage enabled* would be classified as

FAIL and the rest as **PASS**. We begin by calculating the starting entropy of the entire data. In this example we have two classes in which 24 tests belong to **FAIL** and 120 belong to **PASS**. Therefore, the entropy is $-\frac{24}{144}\,log_2\,\frac{24}{144} - \frac{96}{144}\,log_2\,\frac{96}{144} \approx 0.821$.

Now let us calculate what the information gain would be if we split based on the *load default webpage* setting (Equation 2.3). We have 2 possible values of this attribute (enable, or disable).

$$\text{InfoGain(Load Default)} = 0.821 - \sum_{v \in Values\{A\}} \frac{|A_v|}{|A|}\,\text{Entropy}(S_v) \tag{2.4}$$

$$= 0.821 - \frac{|A_{enable}|}{|A|}\,\text{Entropy}(A_{enable}) - \frac{|A_{disable}|}{|A|}\,\text{Entropy}(A_{disable}) \tag{2.5}$$

$$= 0.821 - \frac{72}{144}\,\text{Entropy}(A_{enable}) - \frac{72}{144}\,\text{Entropy}(A_{disable}) \tag{2.6}$$

$$= 0.821 - \frac{72}{144} \times 0.918 - \frac{72}{144} \times 0.000 \tag{2.7}$$

$$= 0.821 - 0.539 \tag{2.8}$$

$$= 0.282 \tag{2.9}$$

If we repeat these calculations for the rest of the attributes we get the following:

Attribute	**Information Gain**
Cookies	0.654
Login	0.171
Crash	0.171
History	0.170
Pop-up	0.171
Load Default	0.282

As we can see, *cookies* gives us the highest information gain. Per the algorithm, we will make a node (the root node) for *cookies* and have three edges, one for each attribute value. On the edges for *allow all cookies* and *restrict 3rd party cookies* we can see that all the configurations lead to **PASS**, so we create leaf nodes. The decision tree looks like Figure 2.2 so far. On the edge for *block all cookies* we must recursively call Algorithm 1 again.

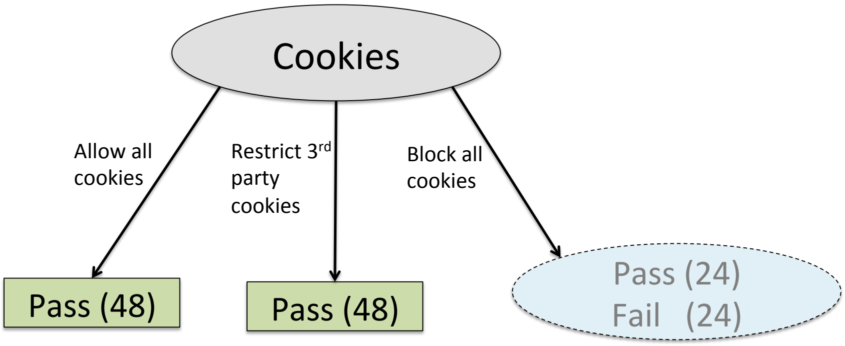


Figure 2.2: The first part of the decision tree for the browser example. The number in parenthesis indicates how many configurations fall into this category.

We repeat the calculations (left as an exercise to the reader). This time instead of looking at all 144 configurations, we only look at the ones that follow down the *block all cookies* branch, which totals 48 configurations. We will choose *load default webpage on startup* as the next attribute with the highest information gain. We create a node for it, then two edges for *enable* and *disable*. This time we see that in all cases with *load enabled* the configuration **FAILED**, and in all cases with *load disabled* the configuration **PASSED**. Thus we create two more leaves. There is now no recursive call and we have completed the algorithm. The decision tree produced by this algorithm can be seen in Figure 2.3.

We will explore real data in Chapter 5. We can use the decision tree to learn about the failure. We can see that a failure will occur if *cookies are blocked* and *load default*

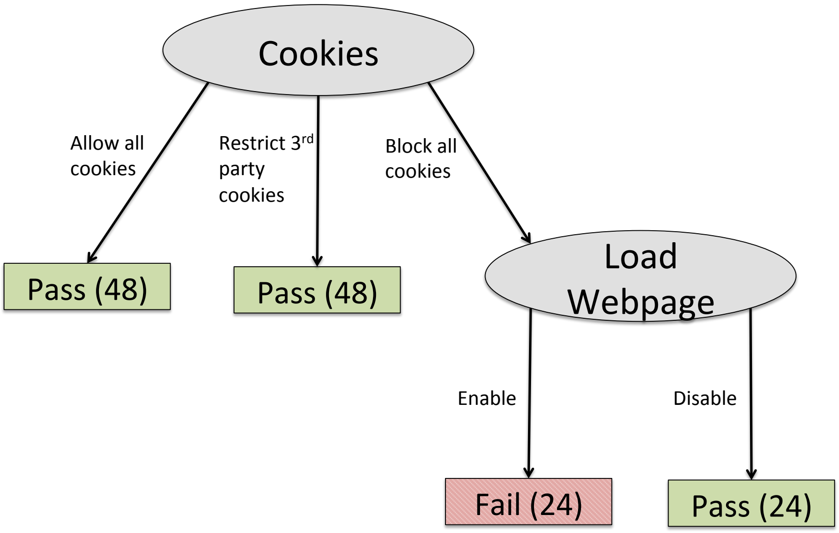


Figure 2.3: The decision tree for the browser example. The number in parenthesis indicates how many configurations fall into this category.

webpage is enabled. Therefore, we can infer that there is an interaction between these two settings leading to a failure. The computer engineer can now limit their search for the fault to these two sections of code which saves time.

2.2.2 Weka

Weka is a collection of machine learning algorithms and tools developed by the Machine Learning Group at the University of Waikato [69]. Capabilities of Weka include: pre-processing, classification, regression, clustering, association rules, and visualization.

Weka has an *explorer* environment to run single machine learning algorithms. It also has an *experimenter* environment to test and compare multiple algorithms and parameters. Weka can allow function on the command line, and its GNU General Public License allows it to plug into other applications. We utilize Weka and its capabilities in this work.

2.3 Systems Biology

Systems biology is a subfield (and often multidisciplinary field) of biology that uses mathematical modeling and engineering to study large and complex biological systems. The field studies systems as a whole, and focuses on the interactions that different subsystems (modules) may have [18]. Systems biology uses quantitative modeling to represent organisms as sets of interacting and communicating biochemical processes [42].

From a computational approach, we can view this as a source code level abstraction of the organism's behavior if we consider the organism itself as an executable program [22]. The most common modeling approach is that of a metabolic network [6, 20, 32]. A metabolic network connects chemical reactions, each representing a biological function. The models are constructed through an iterative process that collects information from manually annotated genomes, known pathway databases, inferences from similar organisms, and the body of literature to build a set of reaction equations and connect their flow [6]. This information is then integrated with data about the reaction dynamics (what compounds flow in and out of each reaction), sub-cellular localization, biomass composition, estimation of energy requirements, reaction directionality and other constraints into a detailed model of the metabolism.

Figure 2.4 shows the metabolic network of *E. coli* obtained from the Kyoto Encyclopedia of Genes and Genomes (KEGG) database [32], the standard database of network models. The nodes are compounds that are inputs and outputs to the reactions (edges). As inputs to the environment are utilized by the organism, a set of reactions creates a path through this network resulting in outputs (waste) to the environment which can be used by other organisms.

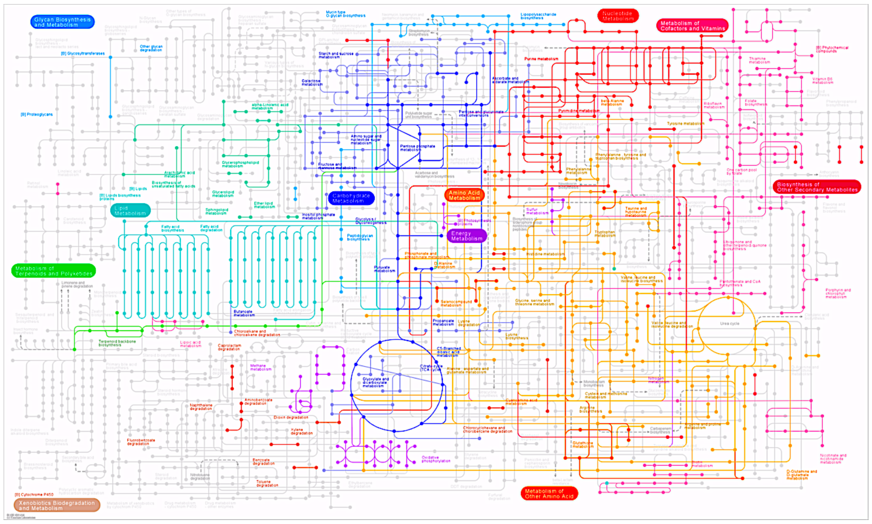

Glycan Biosynthesis and Metabolism
Nucleotide Metabolism
Metabolism of Cofactors and Vitamins
Carbohydrate Metabolism
Lipid Metabolism
Amino Acid Metabolism
Energy Metabolism
Metabolism of Terpenoids and Polyketides
Biosynthesis of Other Secondary Metabolites
Xenobiotics Biodegradation and Metabolism
Metabolism of Other Amino Acids

Figure 2.4: Metabolic model for *E. coli* from the KEGG Database.

This model can be subsequently used for detailed analysis of the metabolic potential of the organism using constraint-based modeling approaches such as a Flux Balance Analysis [28, 31]. A Flux Balance Analysis (FBA) takes environmental factors as inputs and uses a linear programming optimization methodology to maximize flow through the set of equations resulting in an output, for example, the maximum growth. FBA is a widely used method to quantitatively describe steady-state flux distributions in metabolic networks and is used to predict growth of organisms under defined conditions [28]. We use this technique in our case study.

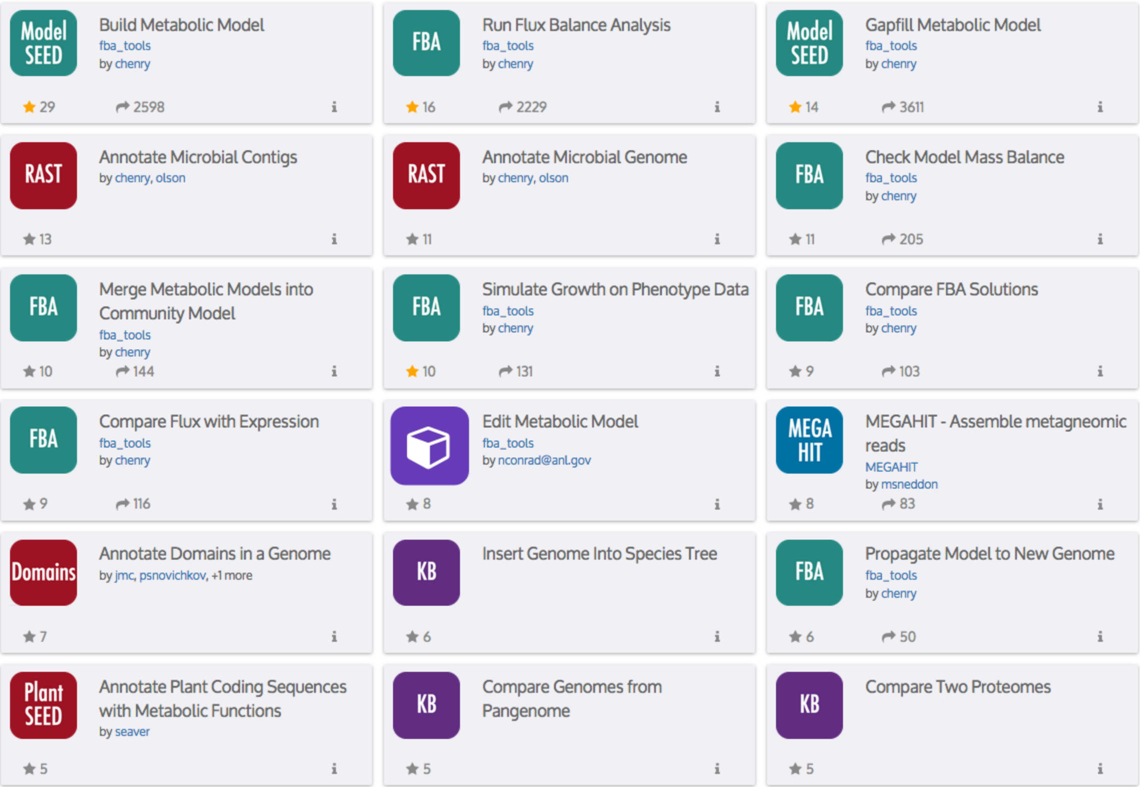
We also point out that these models are theoretical, and may not actually represent the real genomic software program. Since they are created using limited experimental data, information from similar organisms are used to fill in the gaps. As a result, some reactions and paths may be missing, or infeasible reactions and paths may be present that do not represent the original executable program (the organism). As such, we can think about this as working with an imperfect (or static) model of software, which means we need dynamic techniques to iteratively refine the model [53, 71]. However, laboratory experimentation is labor-intensive and time-consuming, therefore we must selectively use this technique.

2.3.1 KBase

The Department of Energy System Biology Knowledgeable — KBase — is an open source software hub and database designed for systems biology [31]. KBase is a highly integrated environment that combines data, tools, and results for predictive biology of microbes, plants, and their communities. A key design choice is that it is collaborative so that users can share data with other users, create new automated analyses via a scripting language, and publish their results via narrative workflows. KBase allows users to create end-to-end workflows: generate hypotheses, design experiments, develop analysis workflows, build and validate models, create visualizations, share with collaborators, and reproduce results.

Account users of KBase have access to a wide array of tools and data. Currently in KBase there are over 28,000 Genomes and 500 defined medias. There exist 54 apps (that have been fully released). Their functions range from building metabolic models to comparing proteomes to RNA-seq analysis and more. A sample of available apps can be seen in Figure 2.5.

KBase includes a graphical user interface called the *narrative interface* (Figure 2.6). In a *narrative* users can upload their own data, access the public data, run apps, comment, and share. The narrative is build within the Jupyter Notebook which gives users the ability to write custom scripts. There also exists a *Software Development Kit* which allows programmers to develop new apps to add to the system. In this



Model SEED Build Metabolic Model fba_tools by chenry ★ 29 ↪ 2598 i	FBA Run Flux Balance Analysis fba_tools by chenry ★ 16 ↪ 2229 i	Model SEED Gapfill Metabolic Model fba_tools by chenry ★ 14 ↪ 3611 i
RAST Annotate Microbial Contigs by chenry, olson ★ 13 i	RAST Annotate Microbial Genome by chenry, olson ★ 11 i	FBA Check Model Mass Balance fba_tools by chenry ★ 11 ↪ 205 i
FBA Merge Metabolic Models into Community Model fba_tools by chenry ★ 10 ↪ 144 i	FBA Simulate Growth on Phenotype Data fba_tools by chenry ★ 10 ↪ 131 i	FBA Compare FBA Solutions fba_tools by chenry ★ 9 ↪ 103 i
FBA Compare Flux with Expression fba_tools by chenry ★ 9 ↪ 116 i	Edit Metabolic Model fba_tools by nconrad@anl.gov ★ 8 i	MEGA HIT MEGAHIT - Assemble metagneomic reads MEGAHIT by msneddon ★ 8 ↪ 83 i
Domains Annotate Domains in a Genome by jmc, psnovichkov, +1 more ★ 7 i	KB Insert Genome Into Species Tree ★ 6 i	FBA Propagate Model to New Genome fba_tools by chenry ★ 6 ↪ 50 i
Plant SEED Annotate Plant Coding Sequences with Metabolic Functions by seaver ★ 5 i	KB Compare Genomes from Pangenome ★ 5 i	KB Compare Two Proteomes ★ 5 i

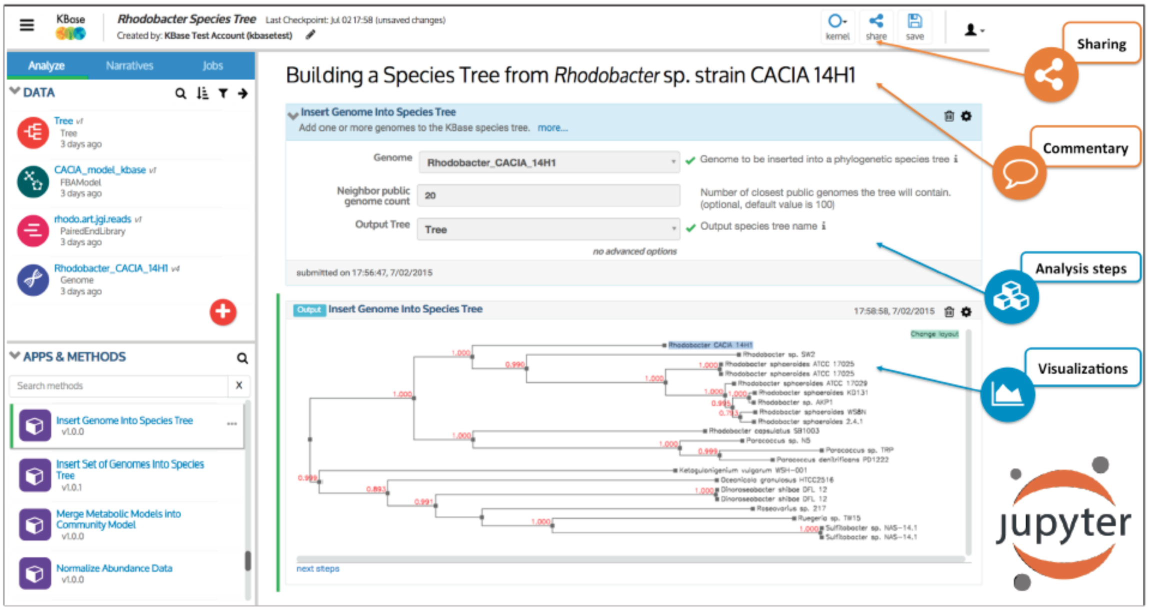
Figure 2.5: Sample of apps available in KBase.

work we prototype a classification app into KBase. All these functionalities make KBase a flexible, collaborative, and end-to-end systems biology platform. A goal of this work is to ensure our processes can be made usable for bench biologists, therefore we use KBase, which provides us the option to implement our process as an app for public use.

2.4 Other Related Work

We next present related work in combinations of the previously described fields.

There has been a large body of research on testing and analyzing configurable software [15, 33, 35, 41, 58, 66, 71]. We don't discuss all here, but instead present related work that connects systems biology with software engineering.



Rhodobacter Species Tree Last Checkpoint: Jul 02 17:58 (unsaved changes)
Created by: KBase Test Account (kbasetest)

Building a Species Tree from *Rhodobacter* sp. strain CACIA 14H1

Insert Genome Into Species Tree
Add one or more genomes to the KBase species tree. more...

Genome: Rhodobacter_CACIA_14H1 ✓ Genome to be inserted into a phylogenetic species tree

Neighbor public genome count: 20 Number of closest public genomes the tree will contain. (optional, default value is 100)

Output Tree: Tree ✓ Output species tree name

no advanced options

submitted on 17:56:47, 7/02/2015

Output Insert Genome Into Species Tree 17:58:58, 7/02/2015

KBase
kernel
share
save
Analyze
Narratives
Jobs
DATA
Tree v1
Tree
3 days ago
CACIA_model_kbase v1
FBAModel
3 days ago
rhodo.art.jgi.reads v1
PairedEndLibrary
3 days ago
Rhodobacter_CACIA_14H1 v4
Genome
3 days ago
APPS & METHODS
Search methods
X
Insert Genome Into Species Tree
v1.0.0
Insert Set of Genomes Into Species Tree
v1.0.1
Merge Metabolic Models into Community Model
v1.0.0
Normalize Abundance Data
v1.0.0
Change layout
Rhodobacter CACIA 14H1
Rhodobacter sp. SW2
Rhodobacter sphaeroides ATCC 17025
Rhodobacter sphaeroides ATCC 17025
Rhodobacter sphaeroides ATCC 17029
Rhodobacter sphaeroides KD131
Rhodobacter sp. AKP1
Rhodobacter sphaeroides WS8N
Rhodobacter sphaeroides 2.4.1
Rhodobacter capsulatus SB1003
Paracoccus sp. N5
Paracoccus sp. TRP
Paracoccus denitrificans PD1222
Ketogulonigenium vulgarum WSH-001
Oceanicola granulosus HTCC2516
Dinoroseobacter shibae DFL 12
Dinoroseobacter shibae DFL 12
Roseovarius sp. 217
Ruegeria sp. TW15
Sulfitobacter sp. NAS-14.1
Sulfitobacter sp. NAS-14.1
1.000
0.990
0.995
0.793
0.999
0.989
0.893
0.997
next steps
Sharing
Commentary
Analysis steps
Visualizations
jupyter

Figure 2.6: Sample narrative in KBase [3].

Research in systems biology aims to obtain computational and mathematical models of biological systems by applying an engineering approach [36]. For example, the concept of *executable biology* has emerged, which builds dynamic computational models of cells rather than mathematical [22]. At the same time, synthetic biology is providing novel tools for the design, realization, and control of the biological systems by programming genetic code, or DNA [30]. These tools are allowing engineers to study and access the molecular information processing of biological cells, opening the road to the control and engineering of biology. The possibility of implementing novel functionality is not only at the single cell level, but also involves communities and populations of cells exchanging information with each other [42, 46]. The future pervasive deployment of genetically engineered cells and their interaction with other bio-, micro-, and nanotechnology enabled devices through molecular communication systems and nanonetworks [49] has been recently envisioned as the novel paradigm of the Internet of Bio-Nano Things [2].

In recent years, computer science tools such as context-free attribute grammars have been applied to synthetic biology to aid in the engineering of programs based on genetic building blocks for known behavior [8]. There is also a standard, SBOL [24], that allows for new designs of biological programs. This line of research uses a software abstraction to build biological programs, but does not apply software engineering techniques and is focused on individual instances of a program.

Finally, automated software requirement analysis and model checking have been successfully applied to DNA self-assembly in the context of fabricating nano-structures with processing capabilities [17,40]. This research is the first to apply automated software engineering techniques to a biological structure (DNA), however, their purpose is to program a single function for individual pathways at the nano-level, whereas we are studying the complex system behavior of an entire organism as it interacts with its environment.

Chapter 3

BioSIMP

The laboratory sciences work to model and understand natural phenomena which can have thousands of interacting specimens. Due to this vast domain space of factors to study, they must limit the focus of studies to only a few factors. We propose utilizing sampling methods to be able to experiment on a larger domain space while minimizing information loss. The analysis on a laboratory study is typically done by hand using heat maps and restricting interactions to pair-wise. These methods can be unaware of higher order interactions, thus we might miss out on interesting and unexpected interactions between metabolites. The laboratory sciences would benefit from an end-to-end process to help sample and classify results.

We now present *BioSIMP*, a process to reason about living systems using methods from software testing. Figure 3.1 shows an overview of BioSIMP with each of the main steps: *Biological Sampling, Inference, Modeling, and Prediction.* A preliminary version of this process was called SCIM [50]. BioSIMP is also presented in [11]. We first define *biological configurable software*, then describe the details of BioSIMP. We use a running example to demonstrate each step.

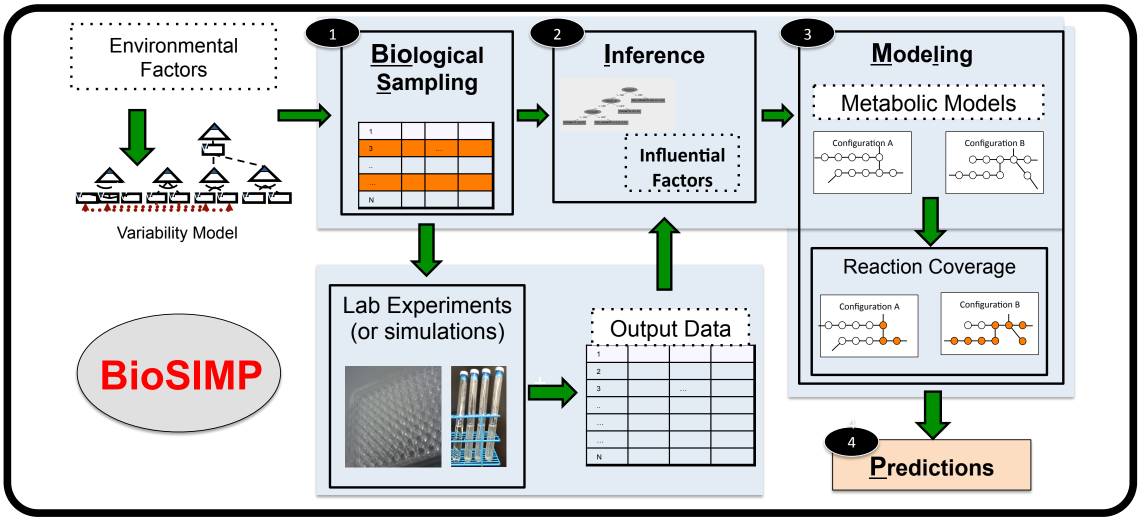


Figure 3.1: The BioSIMP Process

3.1 Biological Configurable Software

First consider the metabolic network to be a model-based abstraction of our program (organism). This model is comprised of lines of code (sets of reactions in the metabolic model). These lines of code connect up in a complex network showing all possible paths through these reactions. Each path may affect the output of the program. In the biological case, this output could be growth, compounds excreted, energy produced, and more.

Using this analogy, we can measure coverage of this model under differing configurations (environmental conditions). In the model, the reactions are the primitive elements. We can view these as methods (or statements) within the code (and can measure reaction or *code* coverage), where we evaluate the expected behavior against the observed behavior – i.e. we can use software testing.

3.2 Initalization Step

Before using BioSIMP we must identify the features that will be varied within the environment. The choice of these depends on the biological organisms under study and requires some domain knowledge. Let us start a running example where we have an organism Z with four features of interest — A, B, C, D — and assume they are all binary (ON or OFF). These features can be anything from media elements, to amount of light, to pH level. For instance, in our experiments in Chapter 5 we have modeled the nutrient components of the microorganisms' culture medium. However, we could also model light, temperature, levels of oxygen, pH, etc.

3.3 Sampling

The first step — biological sampling — selects configurations to test. One option is to exhaustively test all combinations. However, the cost of biological experiments is quite high. It can take up to a month to run an experiment with only seven factors. We can also use sampling techniques such as CIT to intelligently sample the input space.

The full combinatorial space of our running example can be seen in Table 3.1. We have four attributes each with two possible options giving us $2^4 = 16$ possible combinations. Let us suppose we ran an experiment to collect data for this problem. Sixteen tests might be a lot to run, so we use CIT sampling to intelligently choose only a portion of these tests. One possible 2-way covering array generated by CASA has only five tests: 4, 6, 8, 10, 16 (highlighted in Table 3.1).

Experiments are then performed which involve *executing* the organisms' genomic software under those configurations. This can be done in the laboratory, or via

Table 3.1: Full combinatorial set of tests for organism *Z*. A 1 means the attribute is present or ON in the system, and a 0 means the attribute was not present or OFF. A class of G means it was classified as Growth, and NG for No_Growth.

Test	***A***	***B***	***C***	***D***	Class
1	0	0	0	0	NG
2	1	0	0	0	G
3	0	1	0	0	NG
4	0	0	1	0	NG
5	0	0	0	1	NG
6	1	1	0	0	G
7	1	0	1	0	NG
8	1	0	0	1	G
9	0	1	1	0	NG
10	0	1	0	1	NG
11	0	0	1	1	NG
12	1	1	1	0	G
13	0	1	1	1	NG
14	1	1	0	1	G
15	1	0	1	1	NG
16	1	1	1	1	G

simulation; we use both in our study. In our example we are going to study whether or not the organism grows. Suppose in our example we run the sample of 5 tests and get the output in Table 3.2. This output will feed into step 2.

Table 3.2: Class values for the subset of tests in the running example.

Test	***A***	***B***	***C***	***D***	Class
4	0	0	1	0	No_Growth
6	1	1	0	0	Growth
8	1	0	0	1	Growth
10	0	1	0	1	No_Growth
16	1	1	1	1	Growth

3.4 Inference by Classification

Once experiments are complete, we move to inference. In our experiments we use traditional classification trees. We have tried a few classification techniques and have found that classification trees work well for identifying influential factors (see Chapter 4). We leave a full evaluation of alternative inference techniques for future work. Using these classification models we infer invariants at the behavioral level (e.g.

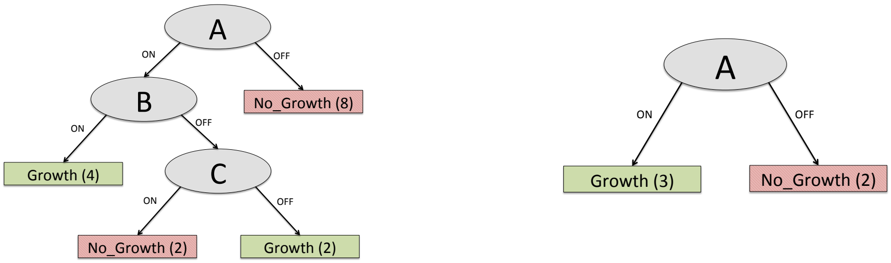
we always see growth or a particular growth threshold under a particular combination of factors).

Let us suppose we ran the full combinatorial space through weka; this will give us our *target tree*. The result for our example can be see in Figure 3.2a. If A is OFF we have No_Growth. Otherwise it depends next on B. If B is ON then we have Growth, otherwise we move to the last influencing factor C. If C is ON we have No_Growth, and Growth if C is OFF. Another interesting inference is that D does not show up; D is not an influencing factor so has no impact on the class value.

For our example the features A, B, C, and D are the attributes, and Growth or No_Growth is the class value. If we run the subset of tests though weka's implementation of the J48 classification trees with default parameters, we get the tree in Figure 3.2b. If our sampling is good, then we expect that the sampled tree will resemble the target tree. In this tree we can see that A is the influencing factor. If A is ON then we have Growth, otherwise when A is OFF we get No_Growth. This tree perfectly sorts all 5 tests.

CIT has been combined with the use of classification trees for *fault characterization* to identify patterns of interactions among the features that lead to classes of faults [41, 70, 71]. Trees are heuristically produced with confidence levels, and guide the exploration of the identification of important factors for failures [69]. Yilmaz et al. [70] have shown that CIT is effective at finding classification trees using only strength 2 and 3— all configurations may not need to be considered.

We can see that while these trees are not the same, they do share similarities such as A being the top influential factor. So while the sampled tree does not give us the complete story, the information it does reveal (that A is an influencing factor) is accurate.



(a) Classification tree for the complete set of tests in the running example.

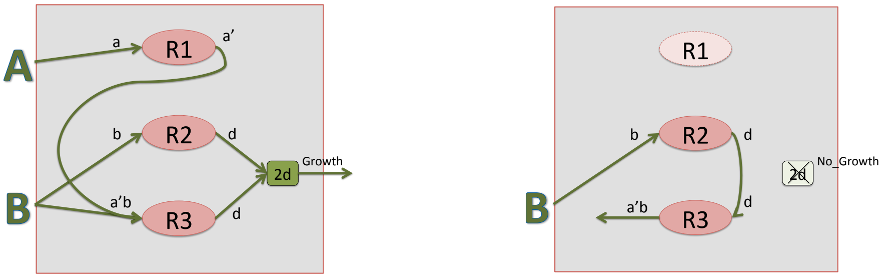
(b) Classification tree for the subset of tests in the running example.

Figure 3.2

3.5 Modeling

Once we identify the influential factors, we model these and map the behavioral (phenotypic) invariants to code level invariants (reactions). Code level invariants can be found by locating the specific reactions executed (covered) and their flow. For example we always see growth or a particular growth threshold under a particular combination of factors which translate into the influential factors. We can do this mapping using simulation (what we did in our experiments) or by *instrumenting* the organisms with markers to identify particular intermediate outputs (e.g. we can use Carbon-13 isotope labeling to trace and quantify metabolites and intermediates [61]). We then can identify the common and variable code which will allow us to focus only on the relevant code in the organisms' network model. This information can be used to iteratively focus lab experimentation only on the important factors, so that we improve the existing models and narrow the gap in representing true organism behavior.

Let us look at a set of three reactions that might occur in our example. Figure 3.3 shows two configurations, one with A and B, and one with just B. According to the classification in Figure 3.3 AB should grow and A should not grow. The next question would be to ask what the code level invariants are that cause this behavior.



(a) Combination of A and B.

(b) Only B turned ON.

Figure 3.3: Reaction Paths in Different Environments for Z

When the configuration is AB (Figure 3.3a) we see that reactions 1–3 are executed. In this model R1 creates a' from a and R2 creates d from b. Then R3 takes both a' and b and creates d. It turns out that two d's are needed for growth. So in the case of AB, growth is able to occur.

Now let us look at the configuration of just B seen in Figure 3.3b. This time R1 is not executed (or covered) at all. R2 executes as normal. Then, surprisingly, R3 executes in the reverse direction. In this pathway of execution two d's are not produced so we do not have growth.

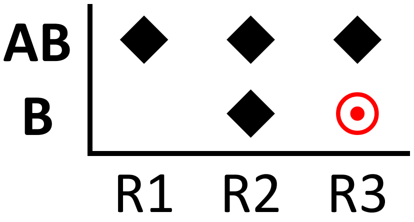


Figure 3.4: Reaction coverage variability model for Figure 3.3

We can model the variability in the execution of these reactions. We have developed a reaction coverage variability model that we show in Figure 3.4. A diamond represents forward (or positive) flow and an open circle represents reverse (or negative) flow.

3.6 Prediction

Finally, we can use our new models to predict future behavior in the organisms, and ultimately we will be able to *program* them to behave in new ways. For example we could use the model for Z we developed in this Chapter to force either growth or no_growth by fixing the inputs (A, B, C, D) to the system. We can also ask questions about what effect various environmental conditions have on the phenotype. We can also look into *why* these effects occur by finding the reactions that cause them.

The potential impact of BioSIMP is that it leads to simplified abstract models for understanding and predicting behavior in organisms, and can identify major metabolic factors from bacterial and archaeal genomic data without the need for exhaustive experimentation or manual curation. In applying BioSIMP to real organisms in our study, the biologists on this project have been able to extract new meaning from our data and are excited about the possibilities of using BioSIMP to reason in novel ways about system behavior (see Section 5.5.4 for details).

3.7 Summary

BioSIMP is an end-to-end process designed for laboratory science, inspired from software testing. This method is built upon the idea of modeling biological systems as highly-configurable software programs. BioSIMP results in reduction of experiments required, abstract model creation, and variability analysis resulting in inferences on influential factors. The insights can lead to new discoveries and predictions that otherwise might be overseen.

In the next chapter we investigate characterization and sampling techniques. In Chapter 5 we present our case study of BioSIMP on two real organisms.

Chapter 4

Evaluating Algorithms for Characterization and Sampling for the Biological Domain

In the biological sciences we can study how different environmental features interact with each other. In nature there are thousands of organisms and compounds that interact. In the human gut there are as many as 10^{12} bacteria per cm^3 [67]. Each of these interactions can have an effect on various phenotypes such as growth of organisms, methane produced into the atmosphere, and more. A set of data collected for such an experiment might include a list of configurations of the various factors and some sort of output that we choose to measure.

Once we collect this data we must analyze it. In this work we are interested in classification of the data into certain classes. Classification builds a model of the data based on the various features and sorts them into classes. For example, in a biological study the features could be pH level, amount of light, and media composition. Then the classes might be whether an organism grows or not. Classification has been used in software testing in work such as [41, 70, 71] where the classes are if a test passes or fails. We can use these classification models to infer the behavior of the system. We can easily see how the various factors interact and what outcomes they cause.

Ideally, we would like to model and understand all of these interactions, but since the number is so vast this is not feasible. We have a similar problem in software testing, and use intelligent sampling to reduce the number of experiments we have to run. The cost of biological experiments is also quite high, requiring hours of preparation and weeks of manual labor. Thus we utilize the sampling techniques we use in software testing to reduce these costs.

BioSIMP is modular in that it can apply to a number of laboratory based disciplines. We limit the focus of this study to experiments in biology. Furthermore, in each step there exist multiple approaches that can be taken. In this chapter we explore both Classification and Sampling techniques. We investigate a few sample representatives and analyze their performance on the data in our study. Characterization and sampling techniques can have different effects on different types of data, thus we limit the domain to ensure our case study on BioSIMP is optimized for our data.

4.1 Classification Techniques

Weka sorts it's classifier algorithms into categories: Bayesian, trees, rules, functions, lazy, multi-instance, and miscellaneous. Lazy learning is best for continuous data and has low readability so we do not consider them. Likewise, we do no consider multiple instance learning because it produces a collection of models rather than a single model. The algorithms in the miscellaneous category work on naive algorithms that only consider ranges of values of which we have none. Thus based on the specifics of our data problem we narrow these down to: Bayesian, trees, rules, functions. We choose a representative from each category to compare and evaluate. Respectively they are: NaiveBayes, J48, PART, MultilayerPerceptron.

Decision trees work by locating the attributes that create the largest information gain which is a measure of how evenly the data is split into its classes. The algorithm will iteratively choose and attribute and recursively go down the tree. A full description of decision trees can be found in Chapter 2.2.

Bayes produces probability estimates. In short, Bayes gives you the most probable explanation for an outcome. We can predict the probability based on the prior probability, probabilities of observing the previous data, and the new observed data. Bayes theorem in itself states $P(h|D) = \frac{P(D|h)P(h)}{P(D)}$ where $P(h)$ is the initial probability that the hypothesis h holds (or prior probability). $P(D)$ is the probability of observing the data we did. So $P(D|h)$ is the probability of observing D given that hypothesis h holds (posterior probability). Bayes chooses the most probable class value of a new observation given the previous observations. For a new test t and previous test T, we can predict the new class as: $x(t) = \mathrm{MAX}_{x_i \in C}(P(a_1, a_2, ..., a_n|x_i))P(v_j)$.

Rules algorithms generate a list of ordered logical rules to determine class values. The algorithm we use (PART) uses a divide-and-conquer approach by building partial decision trees and translating paths to leaves into rules [23].

Function algorithms generate a list of mathematical functions that can be solved to determine class values. These functions are based on the attribute variables and be either discrete or continuous; we study both. The algorithm we use is the Multilayered Perceptron, which in summary, is a neural network using the backpropegation algorithm [44].

Though we focus on classification, we also investigated regression algorithms which predict a raw value instead of a class value. We then use this predicted raw value to determine the class. We experiment with both the regression form of Multilayer Perceptron and a regression tree (M5P). M5P creates a decision tree where the leaves are linear equations of the attributes.

4.2 Sampling Techniques

We explore 3 methods of sampling: CIT, Random, and Option-Wise. We use Random as a base comparison of two sampling methods used in practice. CIT is described in Chapter 2.1.

Recent work on performance testing [52,55,56], sample configuration spaces. They argue sampling for performance differs from sampling for fault detection since performance measures are not discrete. This work is relevant to our research since we also see results (such as growth) that are real-valued. We have two ways to approach this. We can classify into discrete categories (i.e. Growth/No_Growth) or we can use the raw values. As seen in our previous section, we used both discrete labels (classification) and real values (regression). In a similar manner, we look to the literature on performance testing to aid us in finding alternatives to CIT sampling. We summarize some of this work next and then discuss two methods (option-wise and negative option-wise) that we include in our comparison study.

In performance sampling the methods of *Option-Wise Sampling* and *Negative Option-Wise Sampling* have been developed [55]. These methods allow for constraints in the system, but in our case we have no constraints so we can simplify them. Option-Wise (OW) sampling minimizes all possible interactions by only turning on one factor at a time. Negative Option-Wise (negOW) works in the opposite manner by only turning off one factor at a time. Examples of what these sampling techniques would look like on our example from Chapter 3 on organism Z can be seen in Table 4.1.

Table 4.1: Sets of tests for organism *Z*. A 1 means the attribute is present or ON in the system, and a 0 means the attribute was not present or OFF.

Table 4.2: Option-Wise

A	***B***	***C***	***D***
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Table 4.3: Negative Option-Wise

A	***B***	***C***	***D***
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

One performance testing approach looks at *progressive sampling* and *projective sampling* which iteratively builds test sets until either cost is minimized or the gradient of the learning curve is below a certain threshold [52]. For our laboratory experimentation this iterative process is not realistic given the setup cost of experiments, but this is left as future work for our simulated experiments. Another idea from this work is a heuristic for developing an initial sample set. The goals of this initial set are fundamentally different than a representative model sample set, but we still consider this as a sampling method in future work.

Another sampling method from performance testing by Siegmund et al. [56] develops a model of feature interactions using their impact on performance. They obtain raw performance values to determine the *impact* each feature has on a configuration. Although they make some additive assumptions about performance values that might not hold true in our case, we are interested in attempting this approach as future work as we consider attribute selection.

4.3 Study

We explore two research questions to investigate the effectiveness of classification and sampling algorithms on *B. theta* and *M. smithii* in the laboratory and in simulation.

RQ1: What classification technique works well for our data?

Independent Variables We experiment with six algorithms J48, PART, MultilayerPerceptron (classification), NaiveBayes, MultilayerPerceptron (regression), and M5P.

Dependent Variables We use the complete configuration space as our training set, and a 10-fold cross validation [44]. In 10-fold cross validation you split the data into 10 equal sets, train on nine sets, and test on the remaining set. This is repeated so you test on each set once. The final metrics reported are averaged over these 10 runs. A paired T-test statistically compares two populations of data. We use the experimenter in Weka (described in Chapter 2.2) to compare these approaches.

RQ2: What sampling technique works well for our data?

Independent Variables We test on CIT, Random, Option-Wise, and Negative Option-Wise. For CIT we generate 30 samples of each strength from 2-6 (as 7 would be exhaustive) for the feature model using the CASA tool [25]. We generate 30 due to the stochastic nature of CASA. Random is compared to CIT by randomly choosing the same number of tests as a CIT sample. For example, in a 2-way CIT sample we see between 4 and 6 test cases. So to compare to random we choose 100 tests of size 4, 5, and 6 for a total of 300 random tests. Option-Wise and Negative Option-Wise create only one instance each of size 7. However, both methods focus on two-way interactions so we can compare their performance to 2-way CIT samples.

Dependent Variables We use the sample data as our training set and evaluate the results against the full (128) data set to compare how well they predict the full model. We evaluate the quality of the classification trees using both the accuracy and the F-measure, two common metrics for this type of problem. The accuracy tells us the percentage the model correctly classified (4.1), while the F-measure (4.4) is a balance between the precision (4.2) and recall (4.3). Precision calculates the number of true positives (TP) divided by the sum of true positives and false positives (FP). Recall measures the ratio of the true positives, to the sum of true positives and false negatives (FN). We use the J48 classifier (an unpruned C4 decision tree) from Weka [69]. Classification trees require a training set to build the model, and a testing set to evaluate the model.

$$Accuracy = \frac{\#\text{ correctly classified}}{\text{all instances}} \times 100\% \tag{4.1}$$

$$\mathbf{P}recision = \frac{TP}{TP+FP} \tag{4.2}$$

$$\mathbf{R}ecall = \frac{TP}{TP+FN} \tag{4.3}$$

$$F - measure = \frac{2RP}{R+P} \tag{4.4}$$

4.4 Data Collection

We study two types of microbes extracted from the human gut: *Bacteroides thetaiotaomicron* (*B. theta*) and *Methanobrevibacter smithii* (*M. smithii*). These microbes share an evolutionary past and are hypothesized to interact in a synergy that benefits both organisms. We study these organisms both in the laboratory (*in vitro*) and in simulation (*in silico*). Changes in the abundance of intestinal *B. theta* and *M. smithii* have been linked to nutrition-related disorders such as anorexia and obesity [9, 43]. A greater understanding of these microbes and their interaction can help us create better models in order to enhance our understanding of human health.

As our attributes we identified a set of nutritional compounds (factors) based on the known requirements and products of each organism's metabolic system. These attributes are *Glucose*, *Hematin*, *Formate*, H_2, *Vitamin* B_{12}, *Acetate* and *Vitamin K*. Each of these compounds can either be present in the solution (ON) or not (OFF). We utilize growth of the organisms in the given media as the class value.

We use both laboratory experimentation *in vitro* and simulations *in silico* using KBase [31]. We provide only the detail necessary to understand this study. Further detail on experimentation can be found in Chapter 5.

4.4.1 Laboratory Experimentation

Our coworkers in the laboratory provide us with growth values for all 128 media configurations in replicates of eight. Since the laboratory is open to human error factors (resulting from pipetting errors, splashing, or cross contamination) we use Chauvenet's criterion for data removal [63] to eliminate data that is likely to be spurious. To make the final determination of Growth or No_Growth, we compare against 8 negative controls (media plates without any of the seven compounds, which we know will lead to no growth). We use a high stringency statistical test (5.1) over the set of eight replicates; if it satisfies the equation we classify it as Growth, otherwise No_Growth. In this equation, OD is the optical density and STD is the standard deviation. [1]

[1] In the case of regression we perform this classification on the predicted growth values.

$$OD(x_i) - STD(x_1{:}x_8) > OD_{avg}(n_1{:}n_8) + 2[STD(n_1{:}n_8)] \tag{4.5}$$

For each data point x_i we compare its optical density minus the standard deviation of the 8 duplicates to the average optical density over the negative control's $(n_1...n_8)$ duplicates. We provide some leniency by adding twice the standard deviation of the 8 duplicates back in. We use the mode of the 8 duplicates for each configuration as the result.

For *B. theta* we removed 56 of the 1024 data points as outliers using Chauvenet's criterion. For *M. smithii* 52 of 1024 data points were removed. To avoid observer bias, combinations were removed from the analysis when three or more biological replicates of the eight were significantly different from the mode. Of the 128 combinations seven from *B. theta* and 16 from *M. smithii* were removed.

4.4.2 KBase Simulations

In our simulated environment, KBase, we begin by building the metabolic model. The genomes of *B. theta* and *M. smithii* are provided in KBase. We use the *Build Metabolic Model* app which translates the organism's genome to protein sequences from a protein phylogeny database. This provides the initial model. In the first step, the model may be incomplete. We created the 128 different media configuration files and for each ran the *Gapfill Metabolic Model* app to obtain 128 Gapfilled Models. Next we run each of the 128 gapfilled models through the *Run Flux Balance Analysis* (FBA) app to obtain predicted growth values.

4.5 Results

We now present results for our two research questions on the effectiveness of various sampling and classification techniques.

RQ1: What classification technique works well for our data?

Table 4.4: Weka Experimenter results for all four data sets (rows) on 4 different algorithms (columns). Values are accuracies using 10 fold cross-validation. The classification algorithms were compared using a paired T-test. A "*" means that algorithm performed significantly worse, no symbol means there is no statical difference. The last row shows how many out of the 4 data sets was Better/Same/Worse compared to the J48 Decision Trees.

	Classification				Regression	
Dataset	**J48**	**PART**	**Multi-P**	**Bayes**	**Multi-P**	**M5P**
B. theta Lab	95.30	96.19	96.03	93.40	90.06	94.23
M. smithii Lab	89.21	86.74	86.27	72.97*	73.33	75.83
B. theta Simulation	98.45	98.45	94.38*	98.45	97.56	97.56
M. smithii Simulation	96.13	92.84*	93.99	95.97	73.59	74.87
(Better/Same/Worse)		(0/3/1)	(0/3/1)	(0/3/1)		

Table 4.4 displays the results for each algorithm tested on the four data sets we have (*B. theta* in the lab, *B. theta* in simulation, *M. smithii* in the lab, *M. smithii* in simulation). Among the classification algorithms, the J48 tree method is the only algorithm to preform statistically equal or better than the others. Although the difference between the four is not huge, trees are preferred. The regression algorithms only perform comparably on *B. theta* with both algorithms in simulation and M5P in the laboratory. In all other cases, J48 performs better.

Another factor to include is readability of results. The end-user for this work would be a biologist, therefore they need to be able to quickly and easily interpret results. All outputs for *B. theta* in the laboratory can be found in Appendix B. *Decision trees* offer the quickest explanation of the relationship between attributes. *Rules* are also quite simple to follow, but can get cumbersome to follow if there are too many. Neither *Bayes* nor *multilayer perceptrons* offer a visual representation of their models. They do however give weights to attributes which we consider using in future work to consider attribute selection. We leave a formal evaluation of the usability as future work.

Conclusion: Since Decision trees perform statistically equal or better than the others and has an added readability factor we choose this method for our study.

RQ2: What sampling technique works well for our data?

Results for *B. theta* in simulation can be seen in Table 4.5, *M. smithii* in simulation in Table 4.6, *B. theta* in the laboratory in Table 4.7, and *M. smithii* in the laboratory in Table 4.8.

Table 4.5: *B. theta* Simulation

CIT			
strength (size)	**Acc.**	**STD**	**Fm**
2-way (4-6)	70.35	0.10	0.62
3-way (10-12)	98.40	0.00	0.98
4-way (21-25)	98.40	0.00	0.98
5-way (39-42)	98.40	0.00	0.98
6-way (64)	98.40	0.00	0.98
Random			
size	**Acc.**	**STD**	**Fm**
4-6	54.96	18.51	0.44
10-12	85.99	14.53	0.83
21-25	97.73	3.81	0.98
39-42	98.42	0.23	0.98
64	98.44	0.00	0.98
OW			
size	**Acc.**		**Fm.**
7	25.00		0.10
negOW			
size	**Acc.**		**Fm.**
7	48.44		0.32

Table 4.6: *M. smithii* Simulation

CIT			
strength (size)	**Acc.**	**STD**	**Fm**
2-way (4-6)	53.36	0.12	0.45
3-way (10-12)	81.46	0.06	0.79
4-way (21-25)	94.84	0.03	0.94
5-way (39-42)	95.88	0.01	0.94
6-way (64)	96.09	0.00	0.94
Random			
size	**Acc.**	**STD**	**Fm**
4-6	42.53	14.68	0.34
10-12	67.20	13.31	0.53
21-25	82.38	6.66	0.78
39-42	91.27	5.04	0.86
64	95.28	1.94	0.92
OW			
size	**Acc.**		**Fm.**
7	37.50		0.21
negOW			
size	**Acc.**		**Fm.**
7	21.09		0.07

In Table 4.5 we can see that 2-way CIT samples do not preform well. However,

Table 4.7: *B. theta* Laboratory

CIT			
strength (size)	**Acc.**	**STD**	**Fm**
2-way (4-6)	82.96	0.18	0.81
3-way (10-12)	93.04	0.02	0.93
4-way (21-25)	92.90	0.02	0.93
5-way (39-42)	93.81	0.02	0.94
6-way (64)	93.40	0.00	0.93
Random			
size	**Acc.**	**STD**	**Fm**
4-6	66.66	18.76	0.59
10-12	90.50	9.10	0.90
21-25	92.25	3.02	0.92
39-42	93.05	1.99	0.93
64	94.11	1.95	0.94
OW			
size	**Acc.**		**Fm.**
7	57.02		0.41
negOW			
size	**Acc.**		**Fm.**
7	42.98		0.26

Table 4.8: *M. smithii* Laboratory

CIT			
strength (size)	**Acc.**	**STD**	**Fm**
2-way (4-6)	57.11	0.09	0.52
3-way (10-12)	66.96	0.06	0.66
4-way (21-25)	74.70	0.06	0.74
5-way (39-42)	83.04	0.05	0.83
6-way (64)	90.54	0.02	0.90
Random			
size	**Acc.**	**STD**	**Fm**
4-6	57.40	8.57	0.50
10-12	64.81	7.51	0.63
21-25	72.82	7.01	0.72
39-42	80.98	6.22	0.81
64	88.16	4.41	0.88
OW			
size	**Acc.**		**Fm.**
7	43.74		0.27
negOW			
size	**Acc.**		**Fm.**
7	56.25		0.41

3-way and up perform with very high accuracy (98.4) and low standard deviation (less than 0.1). For Random we do not see comparable results until at least 21 test cases. Even with 21-25 tests the accuracy is slightly less, and the standard deviation is higher (3.81). We do see Random perform slightly higher with more than 39 test cases and a low standard deviation, but this is reaching the scope of too many tests. The OptionWise algorithms do worse than all the other options.

The results for *M. smithii* in simulation are similar. In this case we do not get over 90% in a CA until a 4-way. Random does not compare at this level until sizes 39-42 and with significantly more standard deviation. Option-Wise still does not compare here.

The results are almost identical in the laboratory as seen in Tables 4.7 and 4.8.

Conclusion: Although Random can perform comparably to CIT samples in 4-way and higher, the standard deviation is much higher in most cases. In this case CIT is preferable. Similarly, OptionWise is not accurate enough to compare.

4.6 Threats to Validity

We choose only a selection of sampling and classification techniques in these experiments. We believe the algorithms chosen are a generally good representation of the state of the art and diverse.

We also evaluate the effectiveness of these methods only on the data we use for our case study (Chapter 5). However, choosing a classification method is typically dependent on the data itself. Future work would include running these experiments on more sets of data and comparing.

4.7 Summary

We evaluated six different classes of classification algorithms (Trees, Rules, Bayes, Functions, Regression Trees, Regression Functions) on all four of our data sets. Decision trees perform statistically the same and better and are the most readable. Thus we choose to use classification trees in our further studies.

We also evaluated three sampling techniques (CIT, Random, Option-Wise). We show that CIT can produce the same or better results with far less standard deviation than the other approaches. In this study we also discover we must sample at higher levels (4-6 way) to show 90% of the influential factors. While this is high for software systems, it reduces laboratory experiments by at least half, a significant gain given the cost of experiments.

Chapter 5

Evaluating BioSIMP

In this chapter we present a case study to evaluate the feasibility of using BioSIMP. While not a formal user study, we test the effectiveness with a team of biochemists at the University of Nebraska-Lincoln. We test BioSIMP on two real organisms extracted from the human gut and seven environmental factors in the form of compounds in media the organism grows in. These organisms have been a focus of study because of their role in human health. Some parts of this case study has been published in [11] and experimental artifacts can be found at [7]. We answer the following four research questions, one for each step in BioSIMP.

- **RQ1 - Inference:** Can we use classification to identify invariants in microbial metabolism from uncurated genomes?
- **RQ2 - Sampling:** How well does a sampling technique commonly used in software testing, CIT, work to identify the influencing factors?
- **RQ3 - Modeling:** Do we see both variable and common regions in the coverage of reactions in the metabolic model?

- **RQ4 - Prediction:** How well can our inferences and models help predict future behavior and guide experiments?

5.1 Configurable Biological Systems

We study the same microbes extracted from the human gut as in Chapter 4: *Bacteroides thetaiotaomicron* (*B. theta*) and *Methanobrevibacter smithii* (*M. smithii*). These microbes have evolved together in the oxygen-free environment of the human gut. The waste products of *B. theta* are hypothesized to be removed and used by *M. smithii* in a synergy that benefits both microbes. *B. theta* breaks down dietary compounds that human cells cannot utilize directly. Changes in the abundance of intestinal *B. theta* and *M. smithii* have been linked to nutrition-related disorders such as obesity [9].

Independent Variables We choose seven variable compounds: *Glucose*, *Hematin*, *Formate*, H_2, *Vitamin* B_{12}, *Acetate*, and *Vitamin K*. Each of these compounds can either be present in the solution (ON) or not (OFF). We hypothesize that these are important factors which will impact whether or not the organisms will grow. There is also a common set of compounds that all media contain. There are no known constraints on this model.

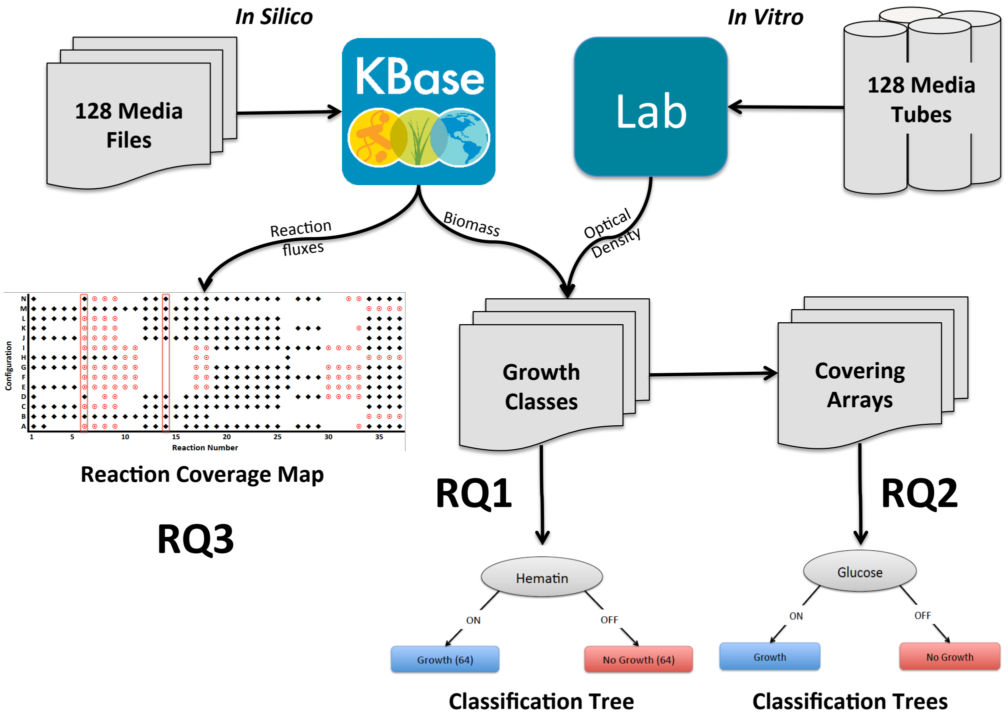
Dependent Variables We also utilize growth of the organisms in the given media as the dependent variable. For RQ1 we evaluate the quality of the classification trees using both the accuracy and the F-measure. Classification trees require a training set to build the model, and a testing set to evaluate the model. For RQ1 we use the complete configuration space as our training set, and a 10-fold cross validation [44]. This type of cross validation has also been used in [70] on configurable software. In RQ2 we use the sample data as our training set and evaluate the results against the full data set to compare how well they predict the full model.

For RQ3 we count the number of elements (reactions) that are common for all configurations and which ones vary between configurations obtained from our simulation environment. A positive flux means the reaction was executed in the forward direction, a negative flux in the reverse direction, and zero means that there was no net flux. Each reaction equation must have one of these outputs.

5.2 Case Study Workflow

The workflow of this case study can be seen in Figure 5.1.

Figure 5.1: Case Study Workflow



In silico we begin by generating the 128 media files. These files are the input to our simulation system. KBase provides us with two outputs: reaction fluxes and growth class for each 128 configurations. The reaction fluxes are used in RQ3 to generate a reaction coverage map.

In vitro we begin by creating the 128 medium and use them in the experiments described in Section 4.4.1. The lab provides us with a second set of growth classes.

For RQ1 we use the growth classifications from all 128 configurations *in vitro* and then again for the *in silico* data. We use Weka [69] to generate the classification trees.

For RQ2 we created 30 covering arrays of each strength from 2-6 for the feature model using the CASA tool [25]. We collect the same information for our analysis but use only the sampled data to build the classification trees.

5.3 Laboratory Experimentation

The laboratory experiments follow the same procedure as in Section 4.4.1 performed by coworkers at the University of Nebraska-Lincoln. Each of the 128 media configurations in replicates of eight across were placed on 32 plates (96-wells each) before inoculation with either *B. theta* or *M. smithii.* Plates were incubated in an anaerobic chamber with $N_2/CO_2/H_2S$ atmosphere (no oxygen) in either the presence or absence of 5% hydrogen gas for one week (*B. theta*) or two weeks (*M. smithii*).

Since the laboratory is open to human error factors we use Chauvenet's criterion for data removal [63] to eliminate data that is likely to be spurious. To make the final determination of growth or no growth, we compare against 8 negative controls (media plates without any of the seven compounds, which we know will lead to no growth). We use a high stringency statistical test (5.1) over the set of eight replicates. In this equation, OD is the optical density and STD is the standard deviation.

$$OD(x_i) - STD(x_1{:}x_8) > OD_{avg}(n_1{:}n_8) + 2[STD(n_1{:}n_8)] \tag{5.1}$$

For each data point x_i we compare its optical density minus the standard deviation of the 8 duplicates to the average optical density over the negative control's $(n_1 ... n_8)$ duplicates. We provide some leniency by adding twice the standard deviation of the 8 duplicates back in. We use the mode of the 8 duplicates for each configuration as the result.

For *B. theta* we removed 56 of the 1024 data points as outliers using Chauvenet's criterion. For *M. smithii* 52 of 1024 data points were removed. To avoid observer bias, combinations were removed from the analysis when three or more biological replicates of the eight were significantly different from the mode. Of the 128 combinations seven from *B. theta* and 16 from *M. smithii* were removed.

5.4 KBase Simulations

For the simulation we ran a 3-step process. Step 1 is completed only once while steps 2 and 3 are repeated for each configuration.

(1) Build Draft Model Before we can simulate a growth experiment, we need to build the metabolic model. The genomes of *B. theta* and *M. smithii* are provided in KBase. We use the *Build Metabolic Model* app in KBase which translates the organism's genome to protein sequences from a protein phylogeny database. Functions of the proteins are assigned based on a nearest-neighbor identity to proteins of known function from other microbial or eukaryotic proteins. Functions are then mapped using the known biochemical pathways in KEGG [32]. This provides the initial model.

(2) Gapfill Metabolic Model In the first step, the model may be incomplete. Not all protein sequences can be identified and some reactions that are needed to synthesize biomass will be missing. The gapfilling algorithm [28] provides a way to fill in those missing links by adding known reactions from manual curation or from a global database to the model in order to force growth, if possible. We created the 128 different media configuration files and for each ran the *Gapfill Metabolic Model* app to obtain 128 Gapfilled Models.

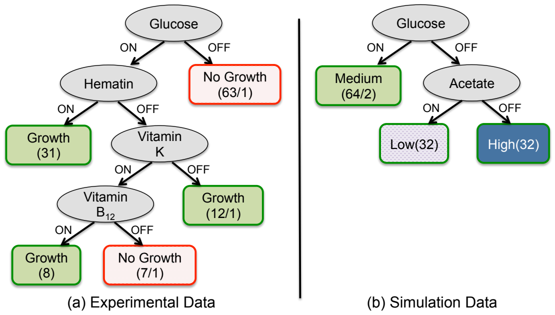
(3) Flux Balance Analysis We run each of the 128 gapfilled models through the *Run Flux Balance Analysis* (FBA) app to obtain the net flux through each metabolic node and the resulting steady-state biomass accumulation rate. This analysis works by simulating the metabolites flowing through the organism's metabolic model. We use this information to build the reaction coverage map for RQ3. We use the biomass information to build the classification trees in RQ1 and RQ2.

5.5 Results

We now present the results to our four research questions, one for each step of BioSIMP — Inference, Sampling, Modeling, Prediction.

5.5.1 RQ1: Inference

Figure 5.2 shows the classification trees for *B. theta* both *in vitro* (a) and *in silico* (b). The experimental data has two categories of outcomes (growth and no growth). The numbers in parentheses on each leaf are the tree splits for the data that has that configuration (left) or is mis-classified (right). The experimental tree has an overall accuracy of 94.22% and an F-measure of .94. The primary split is on Glucose (62 of the 63 without Glucose did not grow). Glucose is required for growth. When Hematin is present in combination with Glucose there is growth. However when Hematin is not present the organisms grow without Vitamin K, or in the presence of both Vitamin K and Vitamin B_{12}.



(a) Experimental Data

(b) Simulation Data

Figure 5.2: *B. theta* Classification Trees

Figure 5.2(b) shows the tree for the simulation data. The tree is different from the experimental tree. First, the organism always grows based on the gapfilled metabolic models (see our discussion in Section 5.4). There are three distinct clusters of the optical densities, therefore we use a tree with three output values (Low, Medium and High). The presence of Glucose leads to Medium growth. In its absence, the presence of Acetate leads to Low growth. Otherwise when neither Glucose nor Acetate is present there is High growth. This indicates that the gapfill algorithm was able to find metabolic reactions in their absence that can lead to growth. The accuracy of this tree is 98.44% with an F-measure of .98.

The classification trees for *M. smithii* are shown in Figure 5.3 and Figure 5.4. The experimental tree (Figure 5.3) has an accuracy of 88.39% and an F-measure of .88. In this organism, the factors interact in a more complex fashion. Vitamin K most often results in a lack of growth, but in the presence of both Vitamin B_{12} and Acetate the organisms can grow. When Vitamin K is absent, Formate leads to growth. Without Formate, the organism grows without Vitamin B_{12}, or with B_{12} if Acetate and H_2 are both present.

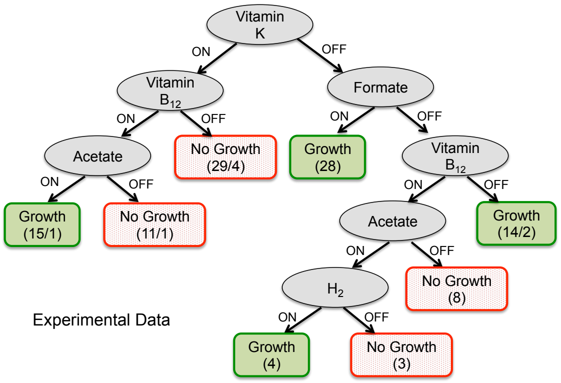


Figure 5.3: *M. smithii* Laboratory Classification Tree

The simulation data for *M. smithii* is also complex (Figure 5.4). It has a natural split (determined by a significant change using the standard deviation) into 4 classes (No Growth, Low, Medium and High). In this tree Glucose is the primary split with Acetate, Formate and Hematin interacting to inhibit or allow growth. The accuracy of this tree is 96.09% with an F-measure of .94.

Summary of RQ1. From this data we can conclude that classification trees can find the influencing factors in the configurations of our environment. We do see that simulation only finds half of the influencing factors that we find in the lab. We also find that the simulation often finds alternative routes through the organism, which may lead to different trees. Without further analysis in the laboratory, we do not know if these are feasible or not.

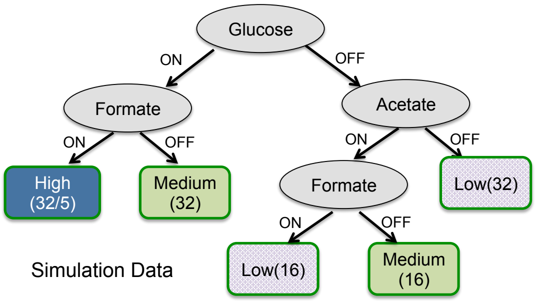


Figure 5.4: *M. smithii* KBase Classification Tree

We now present the results of each research question.

5.5.2 RQ2: Sampling

Our next question asks if we can use CIT to sample the configuration space. For each strength we show the results of 30 different covering arrays for the same model. For each sample we show both the accuracy and F-Measure. The results are averaged across samples (there is only one value for the full data set). Based on this data, Figure 5.5 shows boxplots for each strength CIT on both *B. theta* and *M. smithii*. In *B. theta* we would need to sample at strength 3 or higher to achieve an accuracy and F-measure above 90%. For *M. smithii*, however, we need to go as high as strength 5 or 6 to achieve the same result. Although the laboratory and simulation trees are different within each organism (see RQ1), we see the same pattern of interaction strength, suggesting that *M. smithii* is the more complex organism with respect to these configurations.

Summary of RQ2. CIT sampling is able to provide classification trees with good accuracy and F-measures, however, we need to sample at higher strengths (at least 3 for *B. theta* and 5 or 6 for *M. smithii*).

5.5.3 RQ3: Modeling

We next use KBase to study the variability in the reactions that are identified in the model for each of the configurations of our media. Some of these reactions can only be executed in the forward direction, some only in the reverse direction, and some can be executed in both directions. We mark positive net flux as forward, negative net flux as reverse, and a zero net flux as unexecuted.

Table 5.1 shows detailed coverage data. A + indicates forward flow and a −

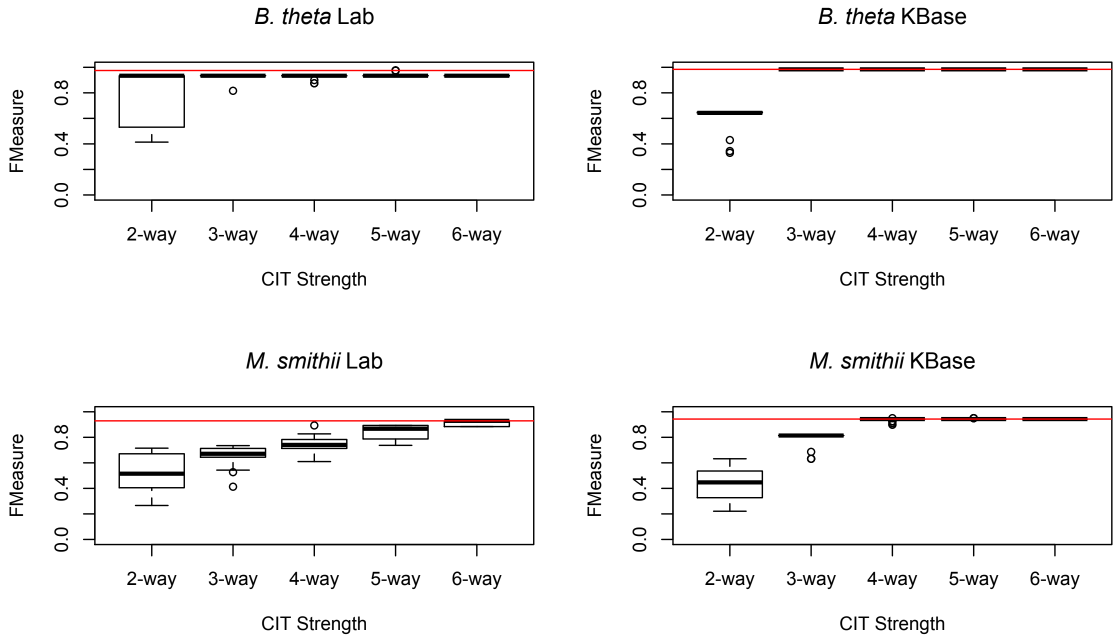


Figure 5.5: F-measures by CIT Strength, for laboratory and KBase Data. Horizontal Line is the tree based on Exhaustive Analysis

indicates reverse flow. In the aggregate model for all configurations of *B. theta* there are 950 different reactions. 37.9% are common to all configurations. 29.5% have positive flow, 8.4% have a reverse flow and 39.8% are uncovered. 212 reactions have variable coverage depending on the influencing factors in the configuration. The range of total coverage is between 459 and 477 (48.3-50.2%). We see a similar pattern for *M. smithii*.

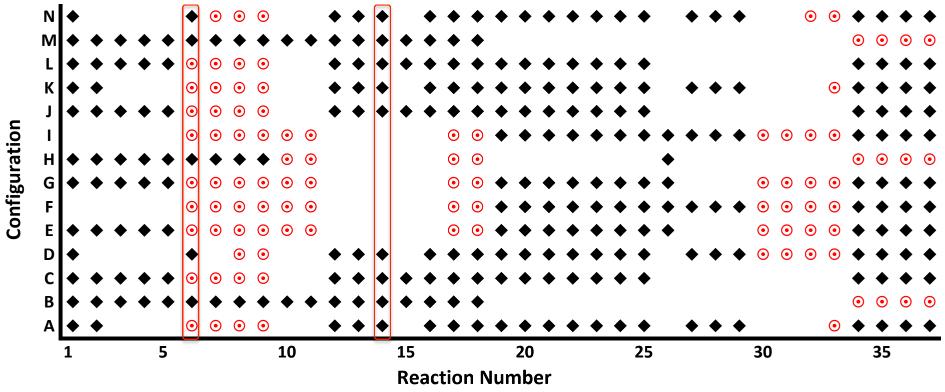


Figure 5.6: Variable Coverage Model. Sample of 37 reactions in *B. theta*.

Table 5.1: Reaction Coverage of Metabolic Model

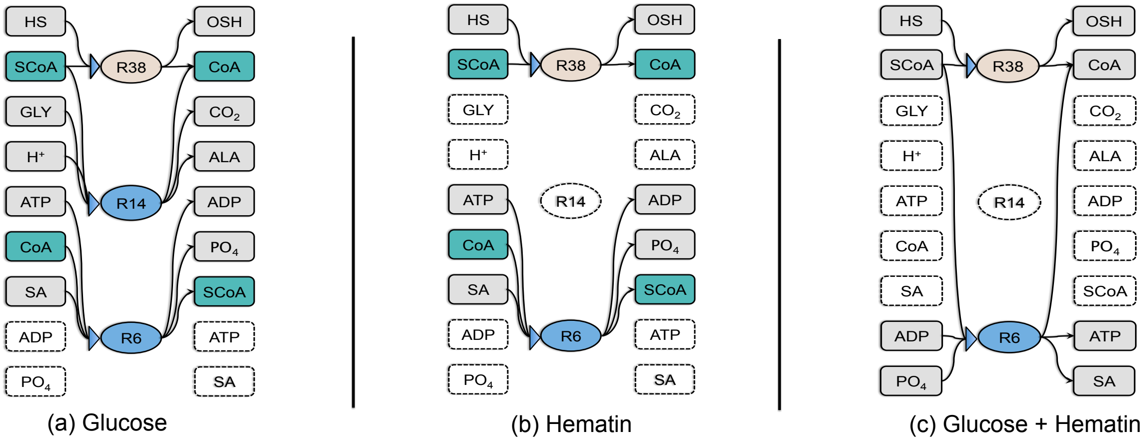
B. theta (950 Reactions)		
	Count	Percent
Common coverage+	280	29.5%
Common coverage−	80	8.4%
Common Total	360	37.9%
Uncovered	378	39.9%
Variable coverage	212	22.3%
Total Coverage Range:	459-477	48.3-50.2%
M. smithii (908 Reactions)		
	Count	Percent
Common coverage+	249	27.4%
Common coverage−	79	8.7%
Common Total	328	36.1%
Uncovered	352	38.8%
Variable coverage	228	25.1%
Total Coverage Range:	430-448	47.4-49.3%

We found 14 patterns of coverage (A-N) among the 128 configurations. These are shown in Figure 5.6 for a subset (37) of the reactions. We limit the reactions to make the graph readable, but complete data is available in Appendix (and Appendix for *M. smithii*). In this graph, the 14 configuration patterns are shown on the y-axis and the reaction number is shown on the x-axis. A diamond represents forward (or positive flow) and an open circle represents reverse (or negative flow). The white space means uncovered. As we can see the coverage pattern varies and as we often see in configurable software – we will either *c*over the reaction or not, depending on the configuration that is selected.

Discussion. We now look at an example of three reactions in *B. theta* to examine this phenomena more closely. Figure 5.7 shows a tiny section of the genome-scale metabolic model under three different environmental conditions (a-c, representing the mixture of the base growth medium). In addition to the common base set of compounds, they contain Glucose, Hematin, and both Glucose and Hematin respectively.

Each subfigure shows inputs (left) and outputs (right) and the reactions through which those inputs and outputs flow. There are three reactions denoted as ovals (#R38, #R14 and #R6). Reaction #R38 (top) behaves identically in all three conditions, but the other two change. The shaded compounds (e.g. SA) are inputs to the reactions, while the unshaded, dashed reactions (e.g. ADP) are not activated in these conditions.

For easy reference the numbered reactions in this example (#6 and #14) are highlighted in Figure 5.6. We do not see the other reaction (#38) since it is part of the common coverage and has positive flow for all combinations of the configuration model.



(a) Glucose

(b) Hematin

(c) Glucose + Hematin

Figure 5.7: Reaction Paths in Different Environments

In the presence of Glucose only (Figure 5.7(a)), all three reactions occur. We observe two compounds (SCoA and CoA represented as darker shades) that appear both as inputs and outputs (the outputs can feed into another reaction that utilizes that compound). If we now move to Figure 5.7(b), the condition in which cells only have Hematin, we find that reaction #R14 (middle) is not covered during the program execution. However, the other two reactions behave similarly to when Glucose alone is in the culture medium. Finally, in Figure 5.7(c) we have a combination of Glucose and

Hematin. Here we observe an unexpected pattern in the flux through the reactions. First, we again only observe two reactions that are executed in the model shown in Figure 5.7(c), despite the presence of both compounds (i.e. the reactions are not additive). Second, we see that #R6 (bottom) behaves in the reverse direction under the combination of these factors (the inputs and outputs are switched). This real example suggests several things.

1. There is variability in how our code executes under differing environmental conditions.

2. There is some common behavior (e.g. reaction #38).

3. The reactions can utilize different inputs and outputs under differing system configurations.

This behavior suggests to us that we can view these systems as highly-configurable software systems and perhaps leverage techniques that we have used from testing and characterizing software to help infer their behavior and to determine which factors and/or combinations of factors are influential in changing behavior. In the example it is clear that both Glucose and Hematin, as well as the interaction of the two, influence behavior. Other compounds that we have experimented with on *B. theta* do not lead to this variability and hence are not influencing factors.

Summary of RQ3. We can obtain a dynamic model of the organims that varies by different media features. From this experiment we can conclude that only a small part (less than 26%) of the reaction space varies between configurations. The implication is that we can target those reaction for further study.

5.5.4 RQ4: Prediction

In RQ4 we ask how our inferences and models can assist the biologists in simplifying the understanding of complex biological systems. These insights may also help to reduce the set of experiments they are required to run.

While not a formal user study, anecdotally the classification trees have proven very useful for the biologists for sifting through exhaustive phenotype datasets. The graphical trees clearly show which culture medium components resulted in growth and also suggest previously unknown positive and negative interactions between metabolites. These classifications hint at unknown gene regulatory networks and novel biochemical pathways that can be investigated through more invasive physiological, transcriptomic, proteomic, and metabolomic experiments. We present some new observations next.

For example, the interaction between Vitamin K and Vitamin B_{12} had not been previously observed, and has led them to run new experiments to explore the relationship and its effect on *B. theta* and *M. smithii*. This has interesting implications for obesity, as a diet rich in foods such as fish and kale is high in Vitamin K and Vitamin B_{12}.

The BioSIMP approach shows that 75% of the reactions in the models are not affected by the media configurations. These reactions may be either critical to the cell under all conditions, or are simply unrelated to the factors we tested. The remaining 25% of the reactions shift in response to the available compounds and reactions added in the gapfill, and identify significant pathway changes that may or may not be possible in the living cells.

The results of the BioSIMP approach also suggests that existing gapfilling algorithms overestimate metabolism by approximately 50% and as a result underestimate the essential metabolite factors by at least factor of 2. This result was independent of the hereditary lineage of the organism, as the same pattern was observed in a bacterium, *B. theta*, and an archaeon, *M. smithii*. The organisms have significantly different genome sizes, and completely divergent metabolisms, suggesting that gapfilling and flux minimization algorithms may uniformly overestimate metabolism across species.

BioSIMP suggests that 5-6 factor CIT will sample at least 90% of the determinant factors for growth even for an uncurated genome. While 5 or 6 is high for software testers, this reduces the laboratory experiments by at least half, a significant gain given the cost of experiments.

5.6 Summary

We show that BioSIMP is able to find influential factors in both the laboratory and in simulation on two human microbes. We also show that at most 26% of the reactions in the reaction network are variable allowing biologists to focus only on a narrow part of the network to understand behavior. We also note some differences with respect to software, such as higher strength is needed in CIT sampling and that reaction coverage is not binary as is code coverage.

Our analysis of reaction coverage has given us the following insights:

1. There is variability in how our code executes under differing environmental conditions.

2. There is some common behavior.

3. The reactions can utilize different inputs and outputs under differing system configurations.

This behavior suggests to us that we can view these systems as highly-configurable software systems and perhaps leverage techniques that we have used from testing and characterizing software to help infer their behavior and to determine which factors and/or combinations of factors are influential in changing behavior.

Chapter 6

Conclusions and Future Work

In this thesis we have presented a view of highly-configurable biological organisms in order to utilize software engineering techniques to reason about their behavior under different configurations. We present a process — BioSIMP — that models environmental configurations and then uses software testing techniques to sample, classify the results to infer influential features, and build models based on these inferences. This information can then be used to predict future behavior of the biological systems.

In a case study on classification techniques we found that J48 decision trees work as well or best out of our sample algorithms. Regression algorithms are comparable in one our of datasets, and future work lies in utilizing the weights on features hidden in the models. We also experimented on four different sampling techniques from software testing. We show CIT provides the highest accuracy under low strengths (2-4), and comparable accuracies and a lower standard deviation than Random in higher strengths (5-6).

In a case study on two human microbes, we show that BioSIMP is able to find influential factors in both the laboratory and in simulation. We also show that up to a quarter of the reactions in the reaction network are variable allowing biologists to focus only on a narrow part of the network to understand behavior.

We also note some differences with respect to software. Using CIT sampling we find that the interaction strengths needed to get quality trees in the biological systems are higher (3-6 way) compared to the usual 2-3 way in software, suggesting the need for more robust testing algorithms.

We were not expecting the notion of bi-directional coverage of the model when we started. Since reactions can be covered in two directions and we lack an analogy for this in software testing it brings up some interesting questions. We believe that this may be useful to reason about types of coverage in software that is also directed (perhaps for instance in user interface models that are represented as graphs or state-machines), or in data flow analysis. While data flow analysis has the ability to identify directional flow [13], it is not currently used due to scalability.

These differences can lead to novel software testing techniques which may also be applied to highly-configurable software.

In future work we plan to look into a larger variety of machine learning algorithms. Specifically looking at algorithms that incorporate cross terms will be useful in ranking the individual effectiveness of attributes. Along these lines, we want to explore the concept of attribute selection instead of building full models. The idea of regression models may also play into this. We would also like to continue our survey of performance testing algorithms and perhaps use their methods to weight attributes and provide raw growth values.

We also plan to explore alternative sampling techniques for this process, and develop the idea of directional code coverage. We would also like to apply BioSIMP to explore additional organisms. On the biological side, we plan to study the new factors we have identified more closely and evaluate the quality of our predictions and how we can use them more broadly.

We would also like to implement BioSIMP as an app into KBase to allow bench-biologists to utilize its capabilities.

Appendix A

Weka Algorithm Output

We present the output for the seven algorithms — J48, NaiveBayes, PART, MultilayerPerceptron (classification), MultilayerPerceptron (regression), M5P — explored in Chapter 4. Results shown are for *B. theta* in the laboratory.

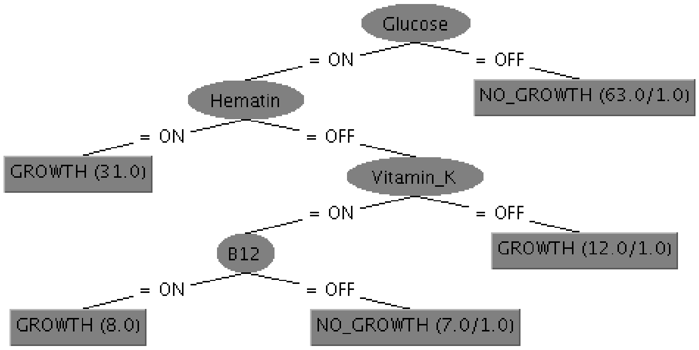
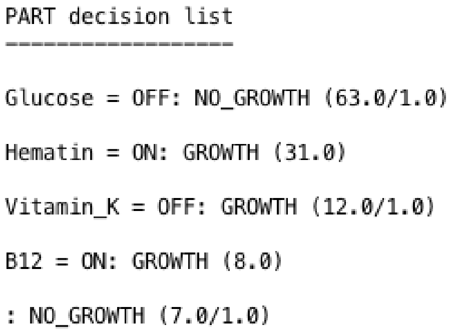
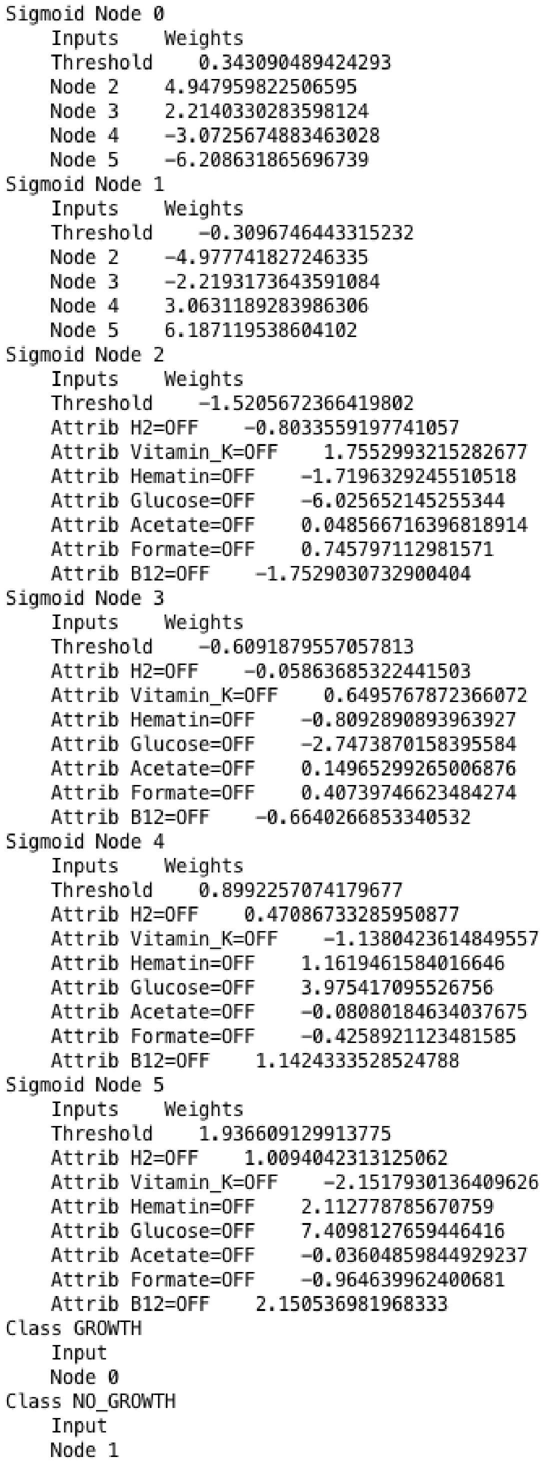


Figure A.1: J48



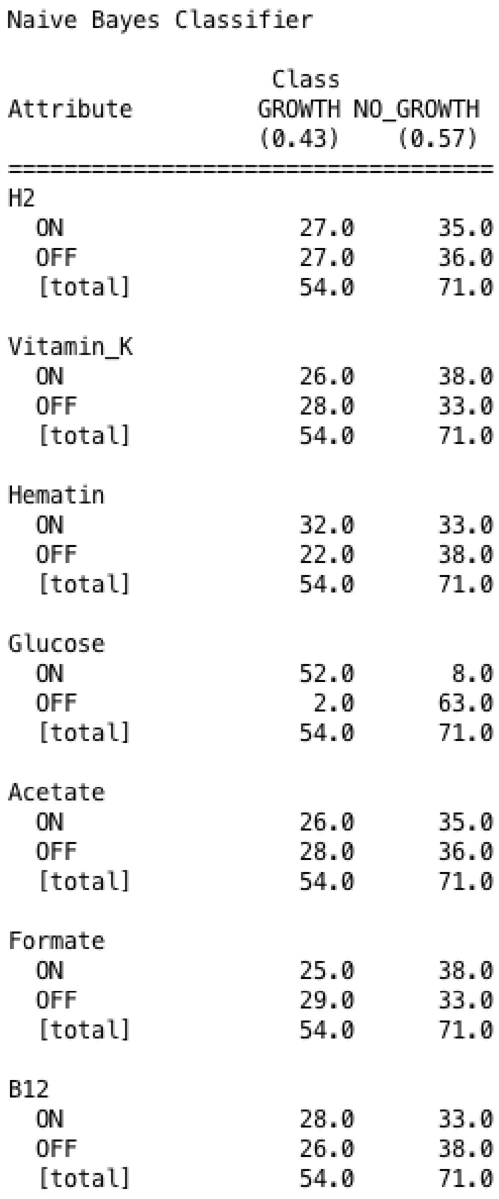
```
PART decision list
------------------

Glucose = OFF: NO_GROWTH (63.0/1.0)

Hematin = ON: GROWTH (31.0)

Vitamin_K = OFF: GROWTH (12.0/1.0)

B12 = ON: GROWTH (8.0)

: NO_GROWTH (7.0/1.0)
```

Figure A.2: PART



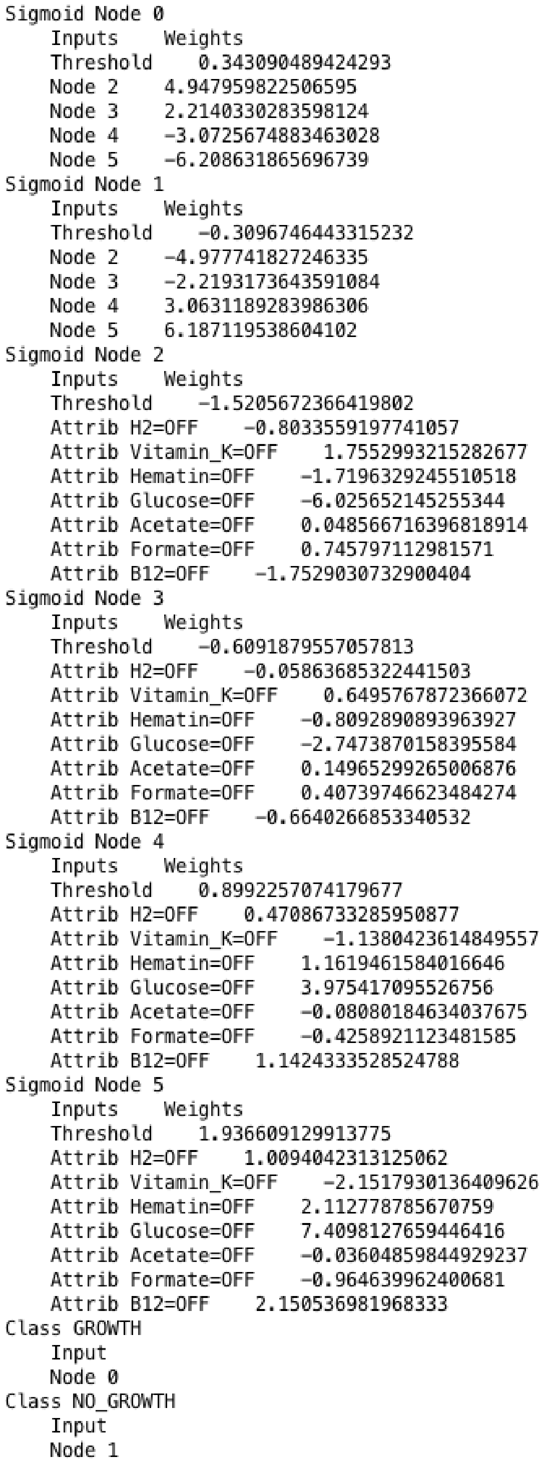
```
Sigmoid Node 0
    Inputs    Weights
    Threshold    0.343090489424293
    Node 2    4.947959822506595
    Node 3    2.2140330283598124
    Node 4    -3.0725674883463028
    Node 5    -6.208631865696739
Sigmoid Node 1
    Inputs    Weights
    Threshold    -0.3096746443315232
    Node 2    -4.977741827246335
    Node 3    -2.2193173643591084
    Node 4    3.0631189283986306
    Node 5    6.187119538604102
Sigmoid Node 2
    Inputs    Weights
    Threshold    -1.5205672366419802
    Attrib H2=OFF    -0.8033559197741057
    Attrib Vitamin_K=OFF    1.7552993215282677
    Attrib Hematin=OFF    -1.7196329245510518
    Attrib Glucose=OFF    -6.025652145255344
    Attrib Acetate=OFF    0.048566716396818914
    Attrib Formate=OFF    0.745797112981571
    Attrib B12=OFF    -1.7529030732900404
Sigmoid Node 3
    Inputs    Weights
    Threshold    -0.6091879557057813
    Attrib H2=OFF    -0.05863685322441503
    Attrib Vitamin_K=OFF    0.6495767872366072
    Attrib Hematin=OFF    -0.8092890893963927
    Attrib Glucose=OFF    -2.7473870158395584
    Attrib Acetate=OFF    0.14965299265006876
    Attrib Formate=OFF    0.40739746623484274
    Attrib B12=OFF    -0.6640266853340532
Sigmoid Node 4
    Inputs    Weights
    Threshold    0.8992257074179677
    Attrib H2=OFF    0.47086733285950877
    Attrib Vitamin_K=OFF    -1.1380423614849557
    Attrib Hematin=OFF    1.1619461584016646
    Attrib Glucose=OFF    3.975417095526756
    Attrib Acetate=OFF    -0.08080184634037675
    Attrib Formate=OFF    -0.4258921123481585
    Attrib B12=OFF    1.1424333528524788
Sigmoid Node 5
    Inputs    Weights
    Threshold    1.936609129913775
    Attrib H2=OFF    1.0094042313125062
    Attrib Vitamin_K=OFF    -2.1517930136409626
    Attrib Hematin=OFF    2.112778785670759
    Attrib Glucose=OFF    7.4098127659446416
    Attrib Acetate=OFF    -0.03604859844929237
    Attrib Formate=OFF    -0.964639962400681
    Attrib B12=OFF    2.150536981968333
Class GROWTH
    Input
    Node 0
Class NO_GROWTH
    Input
    Node 1
```

Figure A.3: Multilayer Perceptron (Classification)



```
Naive Bayes Classifier

                     Class
Attribute          GROWTH NO_GROWTH
                   (0.43)    (0.57)
====================================
H2
  ON                 27.0      35.0
  OFF                27.0      36.0
  [total]            54.0      71.0

Vitamin_K
  ON                 26.0      38.0
  OFF                28.0      33.0
  [total]            54.0      71.0

Hematin
  ON                 32.0      33.0
  OFF                22.0      38.0
  [total]            54.0      71.0

Glucose
  ON                 52.0       8.0
  OFF                 2.0      63.0
  [total]            54.0      71.0

Acetate
  ON                 26.0      35.0
  OFF                28.0      36.0
  [total]            54.0      71.0

Formate
  ON                 25.0      38.0
  OFF                29.0      33.0
  [total]            54.0      71.0

B12
  ON                 28.0      33.0
  OFF                26.0      38.0
  [total]            54.0      71.0
```

Figure A.4: Bayes



```
Sigmoid Node 0
    Inputs    Weights
    Threshold    0.343090489424293
    Node 2    4.947959822506595
    Node 3    2.2140330283598124
    Node 4    -3.0725674883463028
    Node 5    -6.208631865696739
Sigmoid Node 1
    Inputs    Weights
    Threshold    -0.3096746443315232
    Node 2    -4.977741827246335
    Node 3    -2.2193173643591084
    Node 4    3.0631189283986306
    Node 5    6.187119538604102
Sigmoid Node 2
    Inputs    Weights
    Threshold    -1.5205672366419802
    Attrib H2=OFF    -0.8033559197741057
    Attrib Vitamin_K=OFF    1.7552993215282677
    Attrib Hematin=OFF    -1.7196329245510518
    Attrib Glucose=OFF    -6.025652145255344
    Attrib Acetate=OFF    0.048566716396818914
    Attrib Formate=OFF    0.745797112981571
    Attrib B12=OFF    -1.7529030732900404
Sigmoid Node 3
    Inputs    Weights
    Threshold    -0.6091879557057813
    Attrib H2=OFF    -0.05863685322441503
    Attrib Vitamin_K=OFF    0.6495767872366072
    Attrib Hematin=OFF    -0.8092890893963927
    Attrib Glucose=OFF    -2.7473870158395584
    Attrib Acetate=OFF    0.14965299265006876
    Attrib Formate=OFF    0.40739746623484274
    Attrib B12=OFF    -0.6640266853340532
Sigmoid Node 4
    Inputs    Weights
    Threshold    0.8992257074179677
    Attrib H2=OFF    0.47086733285950877
    Attrib Vitamin_K=OFF    -1.1380423614849557
    Attrib Hematin=OFF    1.1619461584016646
    Attrib Glucose=OFF    3.975417095526756
    Attrib Acetate=OFF    -0.08080184634037675
    Attrib Formate=OFF    -0.4258921123481585
    Attrib B12=OFF    1.1424333528524788
Sigmoid Node 5
    Inputs    Weights
    Threshold    1.936609129913775
    Attrib H2=OFF    1.0094042313125062
    Attrib Vitamin_K=OFF    -2.1517930136409626
    Attrib Hematin=OFF    2.112778785670759
    Attrib Glucose=OFF    7.4098127659446416
    Attrib Acetate=OFF    -0.03604859844929237
    Attrib Formate=OFF    -0.964639962400681
    Attrib B12=OFF    2.150536981968333
Class GROWTH
    Input
    Node 0
Class NO_GROWTH
    Input
    Node 1
```

Figure A.5: Multilayer Perceptron (Regression)

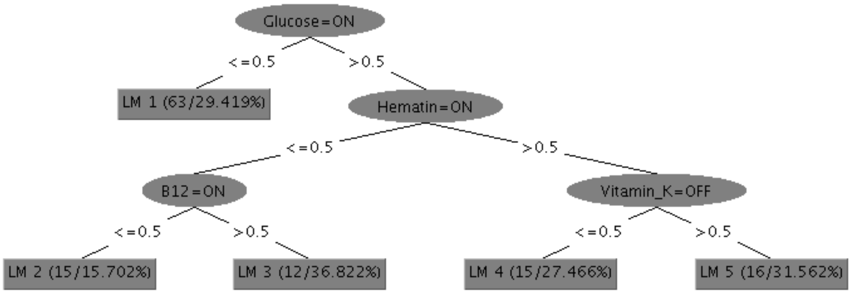
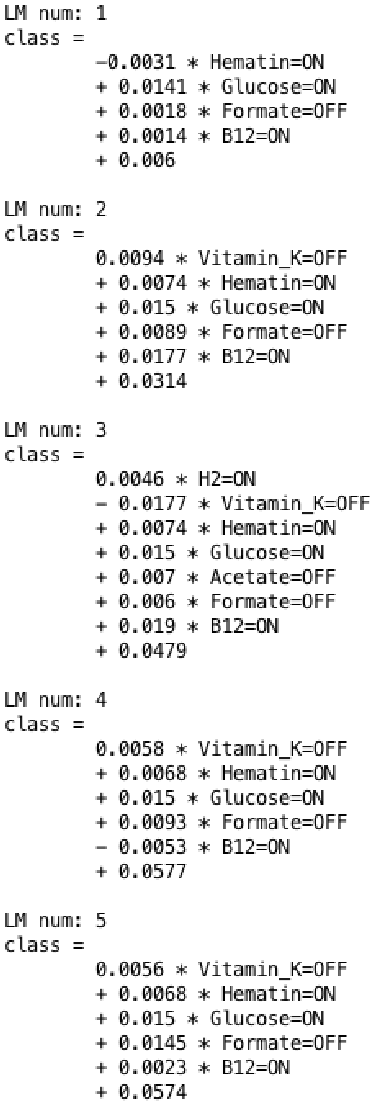


Figure A.6: M5P Tree



```
LM num: 1
class =
        -0.0031 * Hematin=ON
        + 0.0141 * Glucose=ON
        + 0.0018 * Formate=OFF
        + 0.0014 * B12=ON
        + 0.006

LM num: 2
class =
        0.0094 * Vitamin_K=OFF
        + 0.0074 * Hematin=ON
        + 0.015 * Glucose=ON
        + 0.0089 * Formate=OFF
        + 0.0177 * B12=ON
        + 0.0314

LM num: 3
class =
        0.0046 * H2=ON
        - 0.0177 * Vitamin_K=OFF
        + 0.0074 * Hematin=ON
        + 0.015 * Glucose=ON
        + 0.007 * Acetate=OFF
        + 0.006 * Formate=OFF
        + 0.019 * B12=ON
        + 0.0479

LM num: 4
class =
        0.0058 * Vitamin_K=OFF
        + 0.0068 * Hematin=ON
        + 0.015 * Glucose=ON
        + 0.0093 * Formate=OFF
        - 0.0053 * B12=ON
        + 0.0577

LM num: 5
class =
        0.0056 * Vitamin_K=OFF
        + 0.0068 * Hematin=ON
        + 0.015 * Glucose=ON
        + 0.0145 * Formate=OFF
        + 0.0023 * B12=ON
        + 0.0574
```

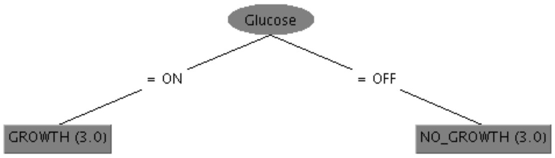
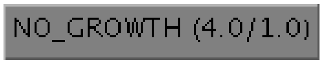
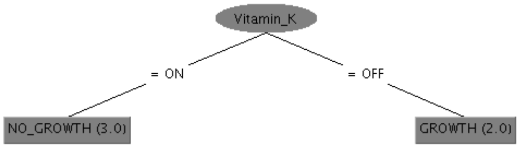
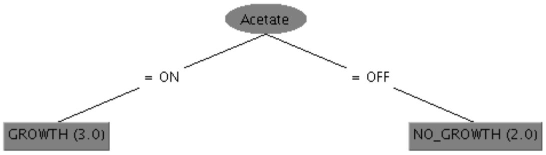
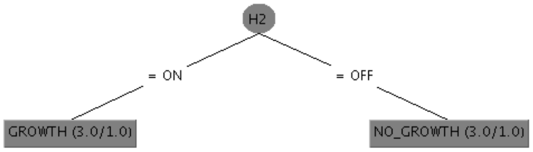
Figure A.7: M5P Linear Models

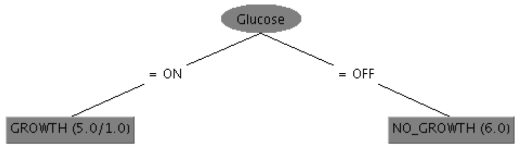
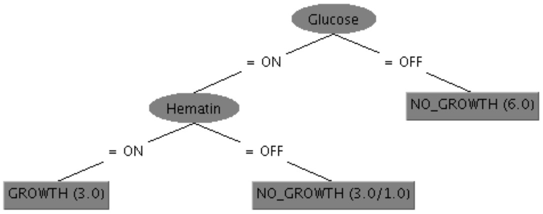
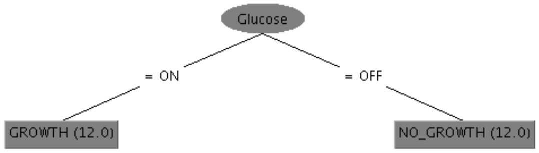
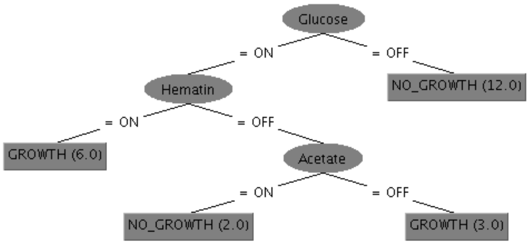
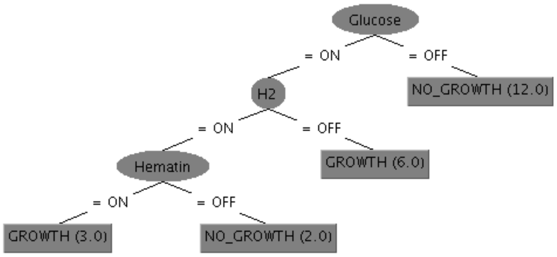
Appendix B

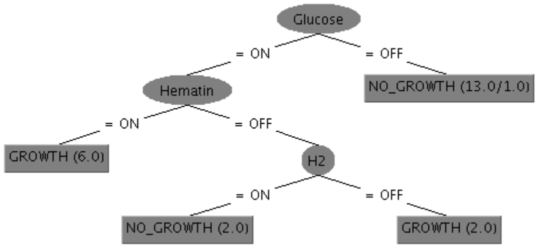
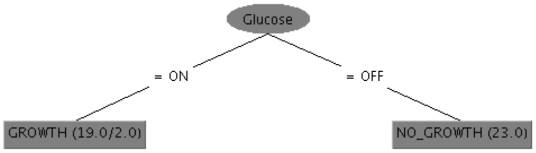
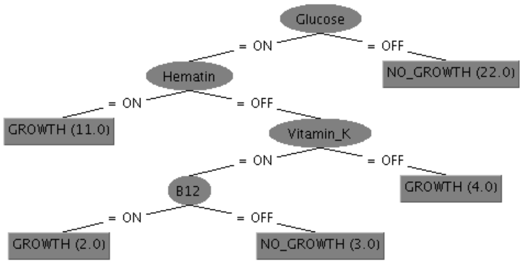
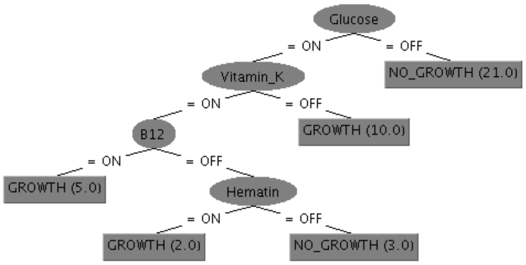
CIT Sample Results

We present tables including all classification trees generated by our sampling technique. We ran covering arrays from strength 2-6 with 30 runs of each. We present one of each structure of tree seen in the 30 samples, frequency out of the 30 runs, accuracy on the complete data set, and F-measure on the complete data set. We also note that the values in the leaves of a tree may not represent all occurrences of that tree, however the topology structure is the same.

Table B.1: *B. theta* Laboratory CIT

2-way			
Tree	Frequency	Accuracy	F-measure
	22	93.00	0.93
	4	57.00	0.41
 Vitamin_K = ON = OFF NO_GROWTH (3.0) GROWTH (2.0)	2	53.00	0.53
	1	50.00	0.50
 H2 = ON = OFF GROWTH (3.0/1.0) NO_GROWTH (3.0/1.0)	1	50.00	0.51

<table>
<tr><th colspan="4">3-way</th></tr>
<tr><th>Tree</th><th>Frequency</th><th>Accuracy</th><th>F-measure</th></tr>
<tr><td>
</td><td>29</td><td>93.00</td><td>0.93</td></tr>
<tr><td>
</td><td>1</td><td>83.00</td><td>0.82</td></tr>
<tr><th colspan="4">4-way</th></tr>
<tr><th>Tree</th><th>Frequency</th><th>Accuracy</th><th>F-measure</th></tr>
<tr><td>
</td><td>27</td><td>93.00</td><td>0.93</td></tr>
<tr><td>
</td><td>1</td><td>90.00</td><td>0.90</td></tr>
<tr><td>
</td><td>1</td><td>88.00</td><td>0.88</td></tr>
</table>

<table>
<tr><td> </td><td>1</td><td>88.00</td><td>0.88</td></tr>
<tr><td colspan="4"><b>5-way</b></td></tr>
<tr><td><b>Tree</b></td><td><b>Frequency</b></td><td><b>Accuracy</b></td><td><b>F-measure</b></td></tr>
<tr><td> </td><td>25</td><td>93.00</td><td>0.93</td></tr>
<tr><td> </td><td>4</td><td>98.00</td><td>0.98</td></tr>
<tr><td> </td><td>1</td><td>98.00</td><td>0.98</td></tr>
</table>

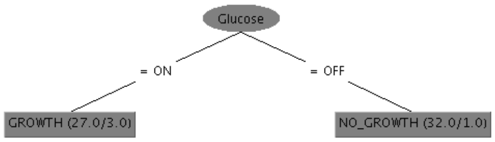
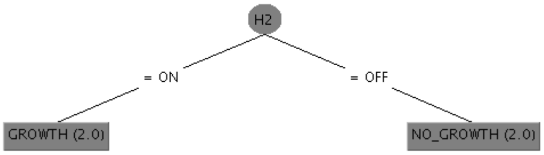
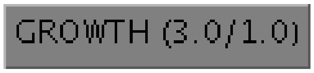
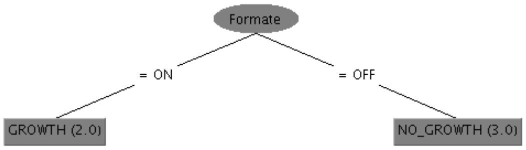
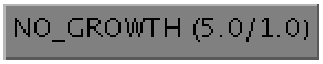
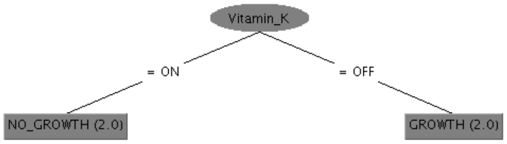
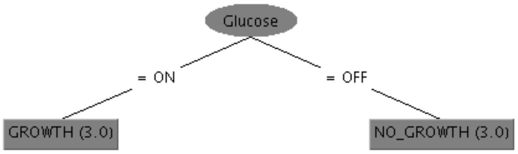
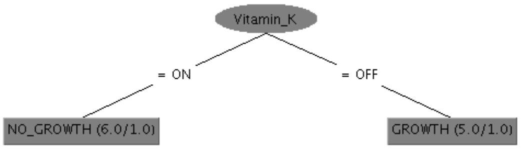
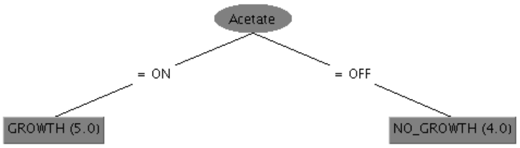
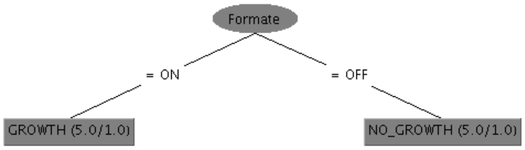
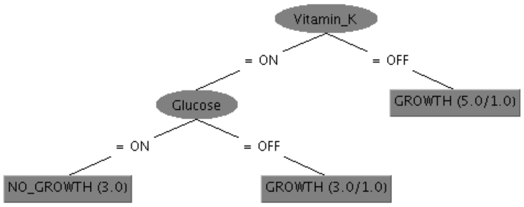
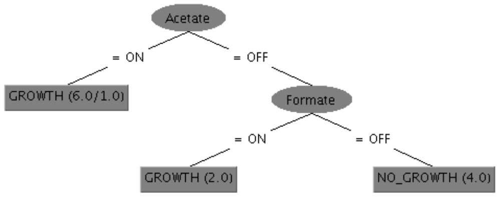
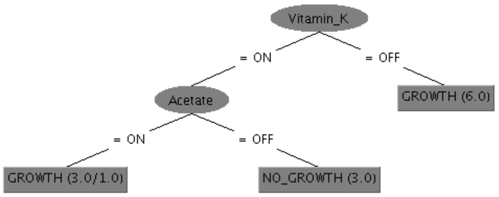
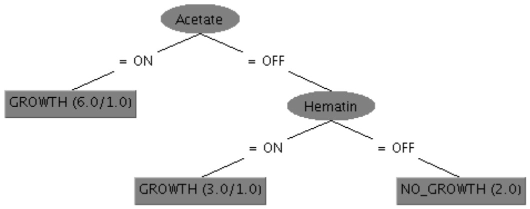
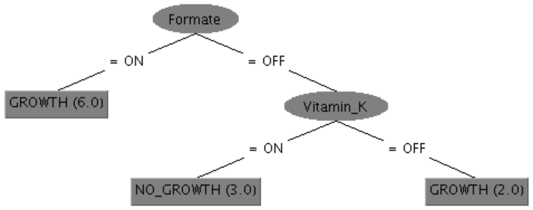
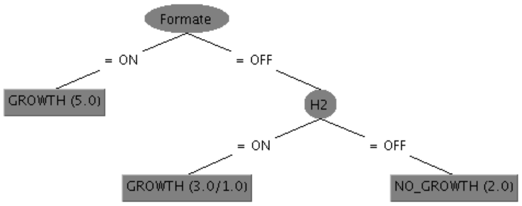
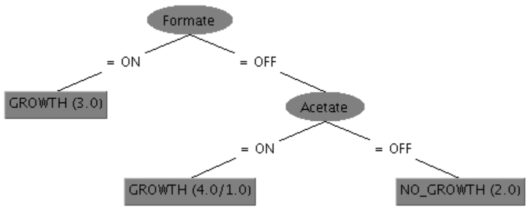
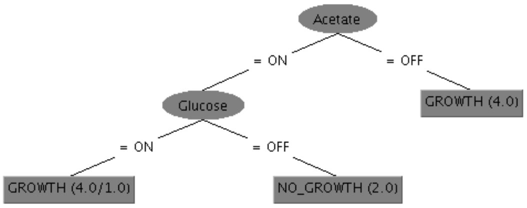
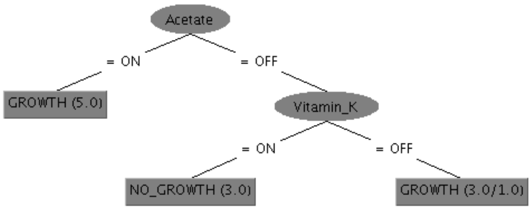
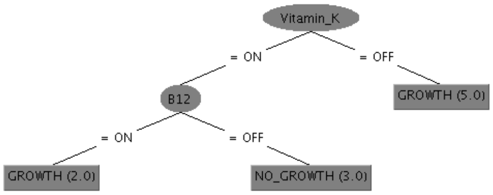
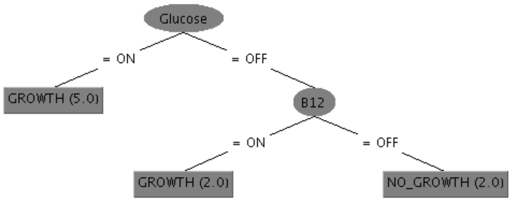
6-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
 Glucose = ON = OFF GROWTH (27.0/3.0) NO_GROWTH (32.0/1.0)	30	93.00	0.93

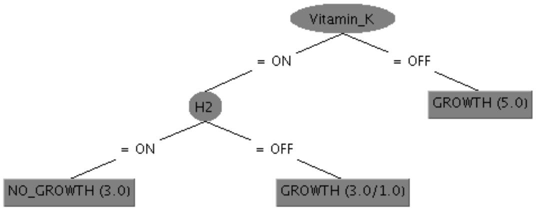
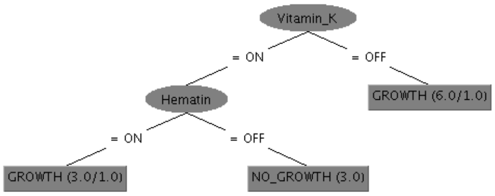
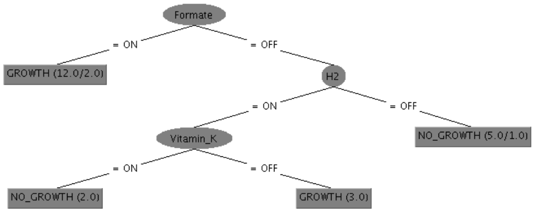
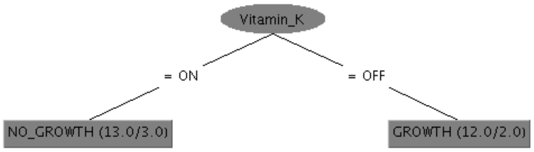
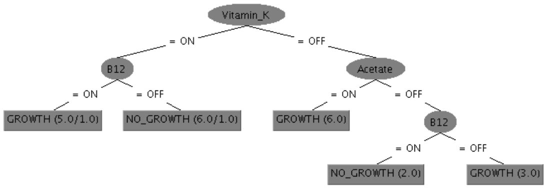
Table B.2: *M. smithii* Laboratory CIT

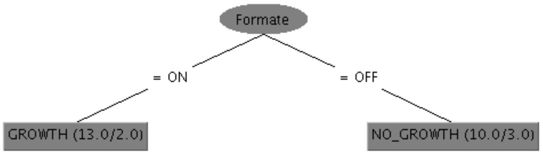
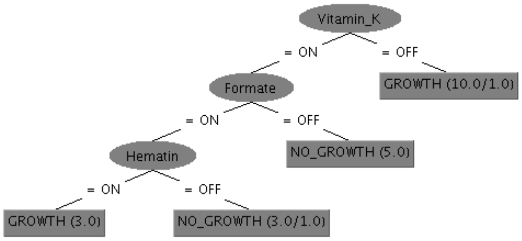
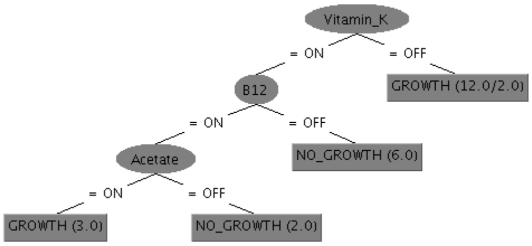
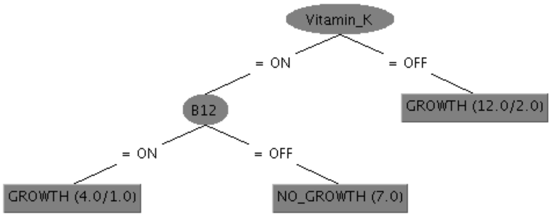
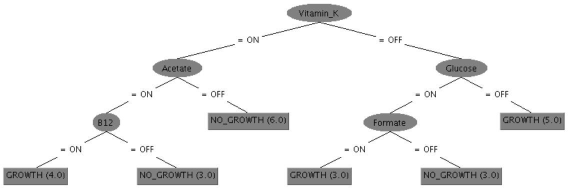
2-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
	8	53.00	0.53
	7	56.00	0.41
	7	67.00	0.67
	3	44.00	0.27
 Vitamin_K = ON = OFF NO_GROWTH (2.0) GROWTH (2.0)	3	71.00	0.71

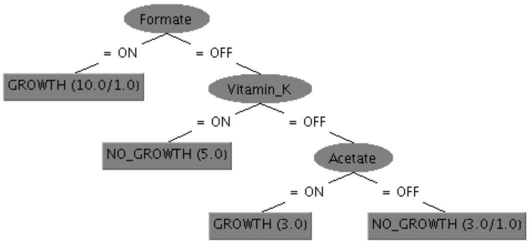
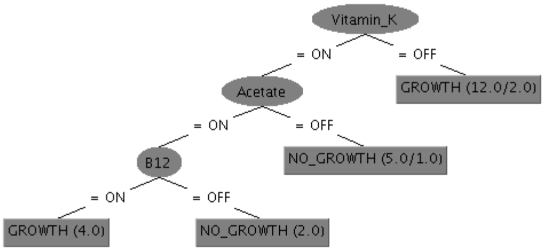
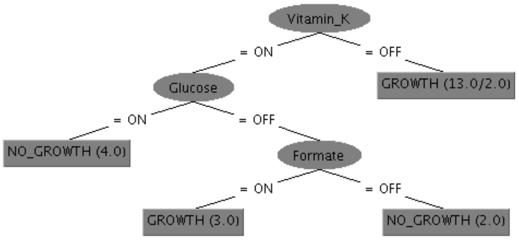
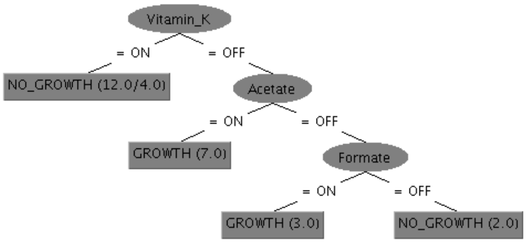
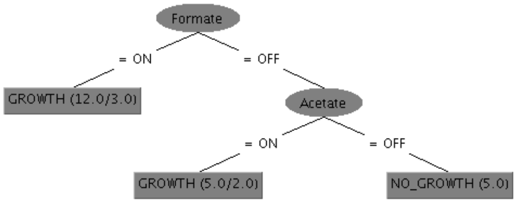
	2	50.00	0.50
3-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
 Vitamin_K = ON = OFF NO_GROWTH (6.0/1.0) GROWTH (5.0/1.0)	6	71.4	0.715
	3	65.2	0.653
 Formate = ON = OFF GROWTH (5.0/1.0) NO_GROWTH (5.0/1.0)	3	67.0	0.671
 Vitamin_K = ON = OFF Glucose GROWTH (5.0/1.0) = ON = OFF NO_GROWTH (3.0) GROWTH (3.0/1.0)	2	64.3	0.618

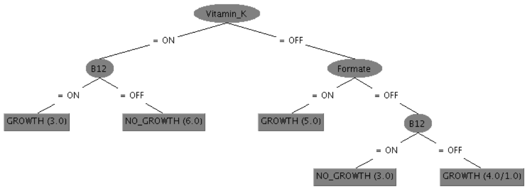
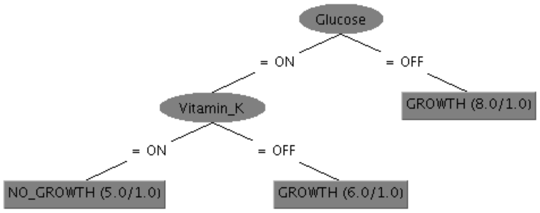
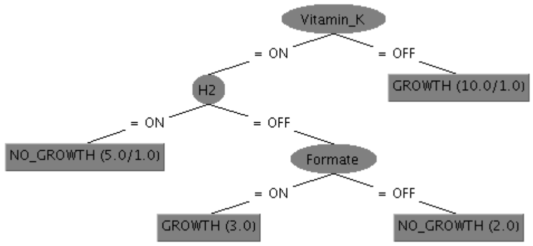
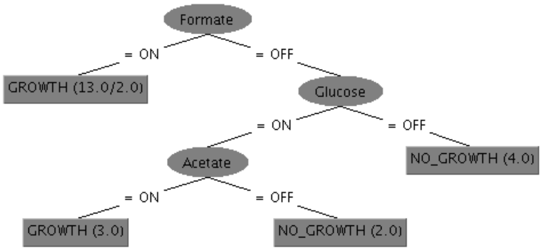
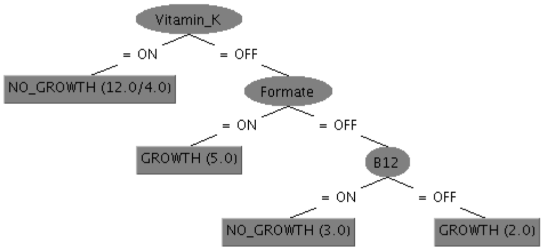
 Acetate = ON = OFF GROWTH (6.0/1.0) Formate = ON = OFF GROWTH (2.0) NO_GROWTH (4.0)	2	73.2	0.713
 Vitamin_K = ON = OFF Acetate GROWTH (6.0) = ON = OFF GROWTH (3.0/1.0) NO_GROWTH (3.0)	2	70.5	0.683
 Acetate = ON = OFF GROWTH (6.0/1.0) Hematin = ON = OFF GROWTH (3.0/1.0) NO_GROWTH (2.0)	2	63.4	0.606
 Formate = ON = OFF GROWTH (6.0) Vitamin_K = ON = OFF NO_GROWTH (3.0) GROWTH (2.0)	2	69.6	0.675
 Formate = ON = OFF GROWTH (5.0) H2 = ON = OFF GROWTH (3.0/1.0) NO_GROWTH (2.0)	1	67.0	0.644

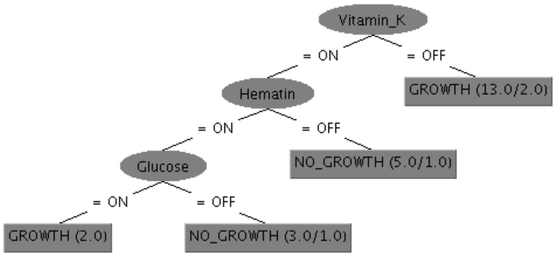
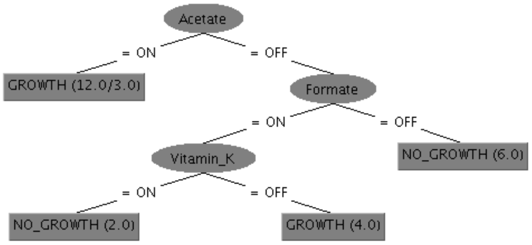
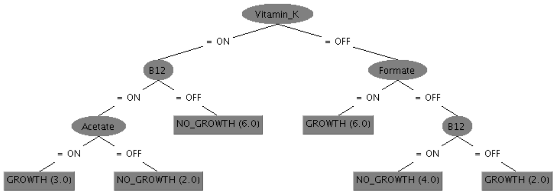
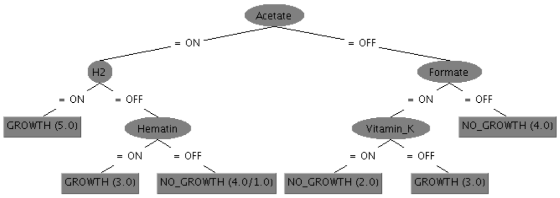
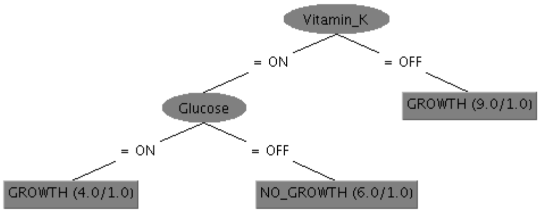
 Formate = ON = OFF GROWTH (3.0) Acetate = ON = OFF GROWTH (4.0/1.0) NO_GROWTH (2.0)	1	73.2	0.713
 Acetate = ON = OFF Glucose GROWTH (4.0) = ON = OFF GROWTH (4.0/1.0) NO_GROWTH (2.0)	1	44.6	0.413
 Acetate = ON = OFF GROWTH (5.0) Vitamin_K = ON = OFF NO_GROWTH (3.0) GROWTH (3.0/1.0)	1	70.5	0.683
 Vitamin_K = ON = OFF B12 GROWTH (5.0) = ON = OFF GROWTH (2.0) NO_GROWTH (3.0)	1	75.0	0.735
 Glucose = ON = OFF GROWTH (5.0) B12 = ON = OFF GROWTH (2.0) NO_GROWTH (2.0)	1	55.4	0.527

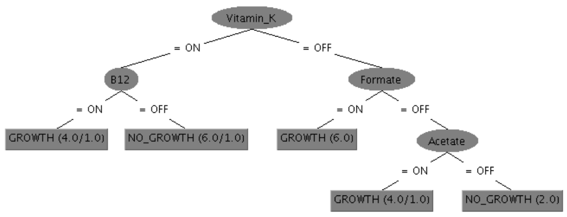
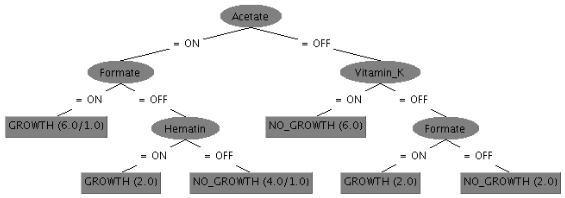
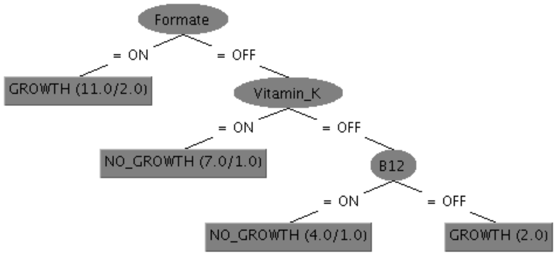
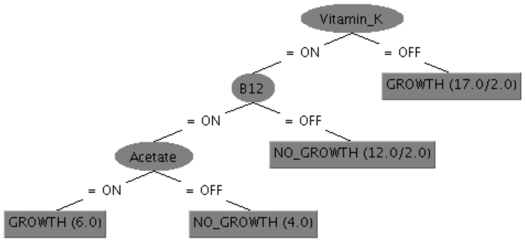
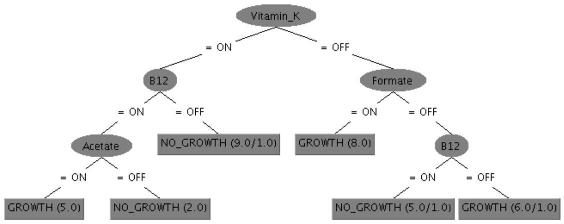
 Vitamin_K = ON = OFF H2 GROWTH (5.0) = ON = OFF NO_GROWTH (3.0) GROWTH (3.0/1.0)	1	65.2	0.633
 Vitamin_K = ON = OFF Hematin GROWTH (6.0/1.0) = ON = OFF GROWTH (3.0/1.0) NO_GROWTH (3.0)	1	67.86	0.656
4-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
 Formate = ON = OFF GROWTH (12.0/2.0) H2 = ON = OFF Vitamin_K NO_GROWTH (5.0/1.0) = ON = OFF NO_GROWTH (2.0) GROWTH (3.0)	2	73.2	0.728
 Vitamin_K = ON = OFF NO_GROWTH (13.0/3.0) GROWTH (12.0/2.0)	2	71.4	0.715
 Vitamin_K = ON = OFF B12 Acetate = ON = OFF = ON = OFF GROWTH (5.0/1.0) NO_GROWTH (6.0/1.0) GROWTH (6.0) B12 = ON = OFF NO_GROWTH (2.0) GROWTH (3.0)	2	78.6	0.783

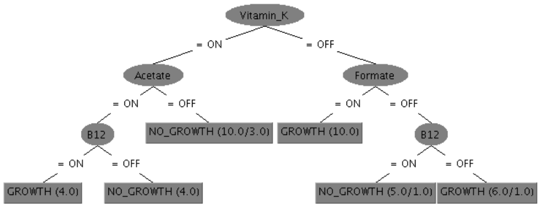
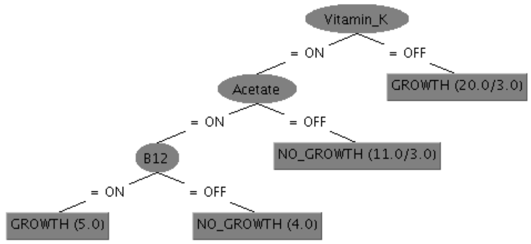
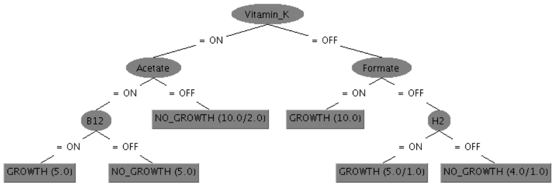
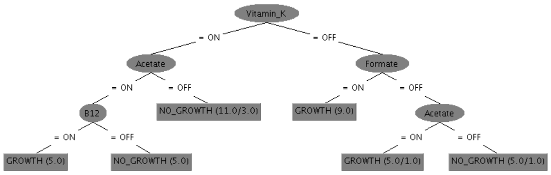
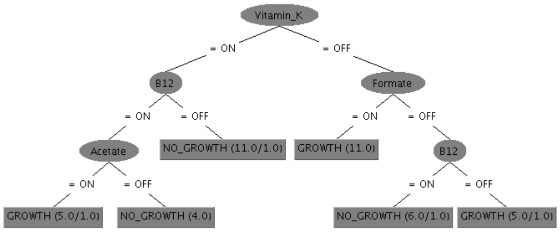
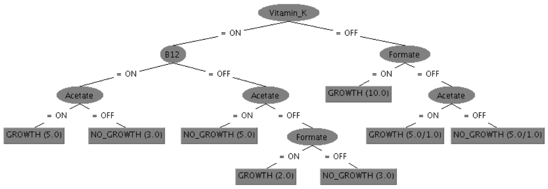
 Formate = ON = OFF GROWTH (13.0/2.0) NO_GROWTH (10.0/3.0)	2	67.0	0.671
 Vitamin_K = ON = OFF Formate GROWTH (10.0/1.0) = ON = OFF Hematin NO_GROWTH (5.0) = ON = OFF GROWTH (3.0) NO_GROWTH (3.0/1.0)	1	73.2	0.728
 Vitamin_K = ON = OFF B12 GROWTH (12.0/2.0) = ON = OFF Acetate NO_GROWTH (6.0) = ON = OFF GROWTH (3.0) NO_GROWTH (2.0)	1	83.0	0.827
 Vitamin_K = ON = OFF B12 GROWTH (12.0/2.0) = ON = OFF GROWTH (4.0/1.0) NO_GROWTH (7.0)	1	75.0	0.735
 Vitamin_K = ON = OFF Acetate Glucose = ON = OFF = ON = OFF B12 NO_GROWTH (6.0) Formate GROWTH (5.0) = ON = OFF = ON = OFF GROWTH (4.0) NO_GROWTH (3.0) GROWTH (3.0) NO_GROWTH (3.0)	1	81.3	0.813

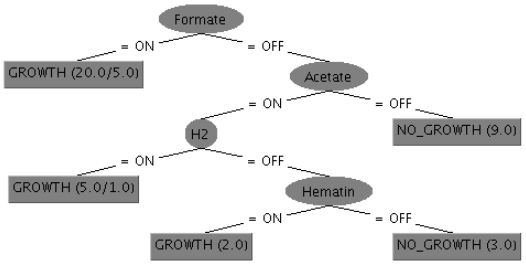
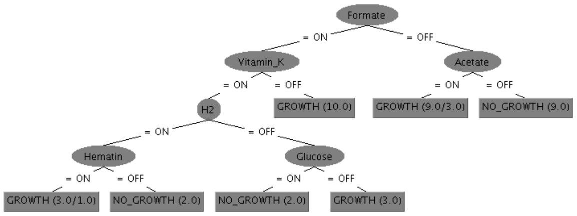
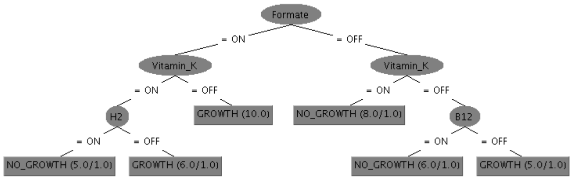
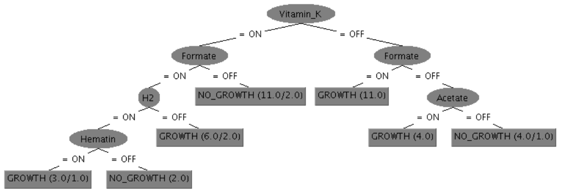
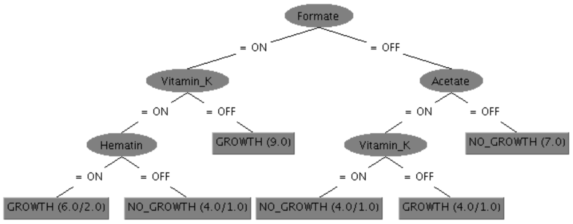
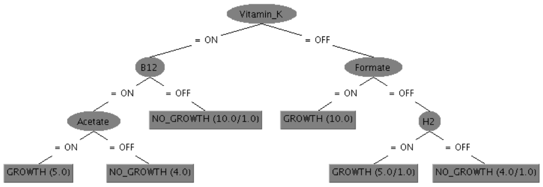
 Formate = ON = OFF GROWTH (10.0/1.0) Vitamin_K = ON = OFF NO_GROWTH (5.0) Acetate = ON = OFF GROWTH (3.0) NO_GROWTH (3.0/1.0)	1	75.0	0.746
 Vitamin_K = ON = OFF Acetate GROWTH (12.0/2.0) = ON = OFF B12 NO_GROWTH (5.0/1.0) = ON = OFF GROWTH (4.0) NO_GROWTH (2.0)	1	83.0	0.827
 Vitamin_K = ON = OFF Glucose GROWTH (13.0/2.0) = ON = OFF NO_GROWTH (4.0) Formate = ON = OFF GROWTH (3.0) NO_GROWTH (2.0)	1	70.5	0.700
 Vitamin_K = ON = OFF NO_GROWTH (12.0/4.0) Acetate = ON = OFF GROWTH (7.0) Formate = ON = OFF GROWTH (3.0) NO_GROWTH (2.0)	1	76.79	0.766
 Formate = ON = OFF GROWTH (12.0/3.0) Acetate = ON = OFF GROWTH (5.0/2.0) NO_GROWTH (5.0)	1	73.2	0.713

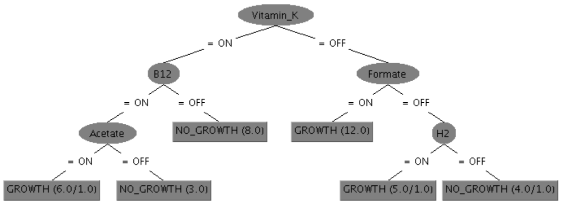
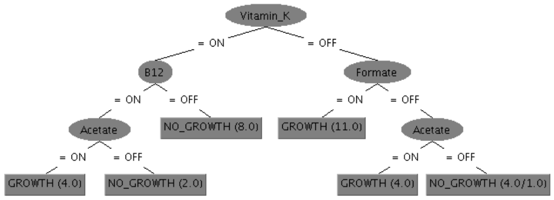
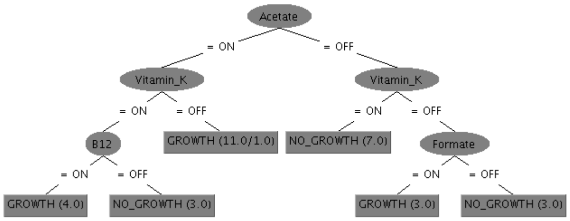
 Vitamin_K = ON = OFF B12 = ON = OFF GROWTH (3.0) NO_GROWTH (6.0) Formate = ON = OFF GROWTH (5.0) B12 = ON = OFF NO_GROWTH (3.0) GROWTH (4.0/1.0)	1	81.3	0.811
 Glucose = ON = OFF GROWTH (8.0/1.0) Vitamin_K = ON = OFF NO_GROWTH (5.0/1.0) GROWTH (6.0/1.0)	1	64.3	0.618
 Vitamin_K = ON = OFF GROWTH (10.0/1.0) H2 = ON = OFF NO_GROWTH (5.0/1.0) Formate = ON = OFF GROWTH (3.0) NO_GROWTH (2.0)	1	72.3	0.720
 Formate = ON = OFF GROWTH (13.0/2.0) Glucose = ON = OFF NO_GROWTH (4.0) Acetate = ON = OFF GROWTH (3.0) NO_GROWTH (2.0)	1	68.8	0.682
 Vitamin_K = ON = OFF NO_GROWTH (12.0/4.0) Formate = ON = OFF GROWTH (5.0) B12 = ON = OFF NO_GROWTH (3.0) GROWTH (2.0)	1	77.7	0.774

 Vitamin_K = ON = OFF Hematin GROWTH (13.0/2.0) = ON = OFF Glucose NO_GROWTH (5.0/1.0) = ON = OFF GROWTH (2.0) NO_GROWTH (3.0/1.0)	1	70.5	0.702
 Acetate = ON = OFF GROWTH (12.0/3.0) Formate = ON = OFF Vitamin_K NO_GROWTH (6.0) = ON = OFF NO_GROWTH (2.0) GROWTH (4.0)	1	75.9	0.755
 Vitamin_K = ON = OFF B12 Formate = ON = OFF = ON = OFF Acetate NO_GROWTH (6.0) GROWTH (6.0) B12 = ON = OFF = ON = OFF GROWTH (3.0) NO_GROWTH (2.0) NO_GROWTH (4.0) GROWTH (2.0)	1	89.3	0.893
 Acetate = ON = OFF H2 Formate = ON = OFF = ON = OFF GROWTH (5.0) Hematin Vitamin_K NO_GROWTH (4.0) = ON = OFF = ON = OFF GROWTH (3.0) NO_GROWTH (4.0/1.0) NO_GROWTH (2.0) GROWTH (3.0)	1	75.0	0.751
 Vitamin_K = ON = OFF Glucose GROWTH (9.0/1.0) = ON = OFF GROWTH (4.0/1.0) NO_GROWTH (6.0/1.0)	1	63.4	0.610

 Vitamin_K = ON = OFF B12 Formate = ON = OFF = ON = OFF GROWTH (4.0/1.0) NO_GROWTH (6.0/1.0) GROWTH (6.0) Acetate = ON = OFF GROWTH (4.0/1.0) NO_GROWTH (2.0)	1	80.4	0.802
 Acetate = ON = OFF Formate Vitamin_K = ON = OFF = ON = OFF GROWTH (6.0/1.0) Hematin NO_GROWTH (6.0) Formate = ON = OFF = ON = OFF GROWTH (2.0) NO_GROWTH (4.0/1.0) GROWTH (2.0) NO_GROWTH (2.0)	1	75.0	0.751
 Formate = ON = OFF GROWTH (11.0/2.0) Vitamin_K = ON = OFF NO_GROWTH (7.0/1.0) B12 = ON = OFF NO_GROWTH (4.0/1.0) GROWTH (2.0)	1	75.9	0.756
5-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
 Vitamin_K = ON = OFF B12 GROWTH (17.0/2.0) = ON = OFF Acetate NO_GROWTH (12.0/2.0) = ON = OFF GROWTH (6.0) NO_GROWTH (4.0)	4	83.04	0.827
 Vitamin_K = ON = OFF B12 Formate = ON = OFF = ON = OFF Acetate NO_GROWTH (9.0/1.0) GROWTH (8.0) B12 = ON = OFF = ON = OFF GROWTH (5.0) NO_GROWTH (2.0) NO_GROWTH (5.0/1.0) GROWTH (6.0/1.0)	4	89.29	0.893

<table>
<tr><td>
</td><td>3</td><td>75.9</td><td>0.750</td></tr>
<tr><td>
</td><td>2</td><td>83.04</td><td>0.827</td></tr>
<tr><td>
</td><td>2</td><td>86.61</td><td>0.867</td></tr>
<tr><td>
</td><td>2</td><td>88.39</td><td>0.884</td></tr>
<tr><td>
</td><td>2</td><td>89.30</td><td>0.893</td></tr>
<tr><td>
</td><td>2</td><td>89.30</td><td>0.893</td></tr>
</table>

 Formate = ON = OFF GROWTH (20.0/5.0) Acetate = ON = OFF H2 NO_GROWTH (9.0) = ON = OFF GROWTH (5.0/1.0) Hematin = ON = OFF GROWTH (2.0) NO_GROWTH (3.0)	1	74.1	0.737
 Formate = ON = OFF Vitamin_K Acetate = ON = OFF = ON = OFF H2 GROWTH (10.0) GROWTH (9.0/3.0) NO_GROWTH (9.0) = ON = OFF Hematin Glucose = ON = OFF = ON = OFF GROWTH (3.0/1.0) NO_GROWTH (2.0) NO_GROWTH (2.0) GROWTH (3.0)	1	76.79	0.764
 Formate = ON = OFF Vitamin_K Vitamin_K = ON = OFF = ON = OFF H2 GROWTH (10.0) NO_GROWTH (8.0/1.0) B12 = ON = OFF = ON = OFF NO_GROWTH (5.0/1.0) GROWTH (6.0/1.0) NO_GROWTH (6.0/1.0) GROWTH (5.0/1.0)	1	78.57	0.787
 Vitamin_K = ON = OFF Formate Formate = ON = OFF = ON = OFF H2 NO_GROWTH (11.0/2.0) GROWTH (11.0) Acetate = ON = OFF = ON = OFF Hematin GROWTH (6.0/2.0) GROWTH (4.0) NO_GROWTH (4.0/1.0) = ON = OFF GROWTH (3.0/1.0) NO_GROWTH (2.0)	1	78.60	0.786
 Formate = ON = OFF Vitamin_K Acetate = ON = OFF = ON = OFF Hematin GROWTH (9.0) Vitamin_K NO_GROWTH (7.0) = ON = OFF = ON = OFF GROWTH (6.0/2.0) NO_GROWTH (4.0/1.0) NO_GROWTH (4.0/1.0) GROWTH (4.0/1.0)	1	78.60	0.787
 Vitamin_K = ON = OFF B12 Formate = ON = OFF = ON = OFF Acetate NO_GROWTH (10.0/1.0) GROWTH (10.0) H2 = ON = OFF = ON = OFF GROWTH (5.0) NO_GROWTH (4.0) GROWTH (5.0/1.0) NO_GROWTH (4.0/1.0)	1	86.60	0.867

 Vitamin_K = ON = OFF B12 Formate = ON = OFF = ON = OFF Acetate NO_GROWTH (8.0) GROWTH (12.0) H2 = ON = OFF = ON = OFF GROWTH (6.0/1.0) NO_GROWTH (3.0) GROWTH (5.0/1.0) NO_GROWTH (4.0/1.0)	1	86.61	0.867
 Vitamin_K = ON = OFF B12 Formate = ON = OFF = ON = OFF Acetate NO_GROWTH (8.0) GROWTH (11.0) Acetate = ON = OFF = ON = OFF GROWTH (4.0) NO_GROWTH (2.0) GROWTH (4.0) NO_GROWTH (4.0/1.0)	1	88.39	0.884
 Acetate = ON = OFF Vitamin_K Vitamin_K = ON = OFF = ON = OFF B12 GROWTH (11.0/1.0) NO_GROWTH (7.0) Formate = ON = OFF = ON = OFF GROWTH (4.0) NO_GROWTH (3.0) GROWTH (3.0) NO_GROWTH (3.0)	1	88.39	0.884

6-way

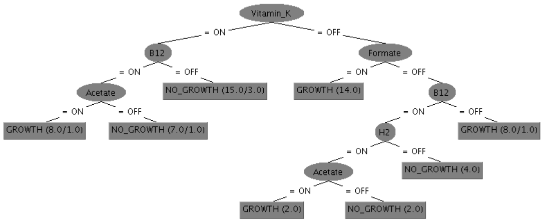
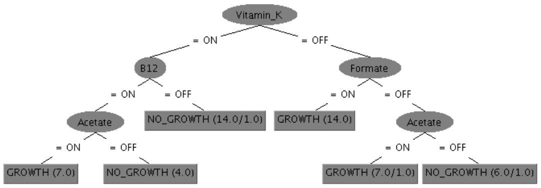
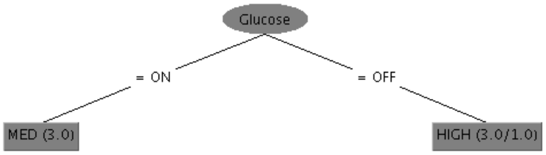
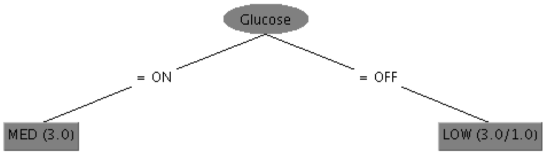
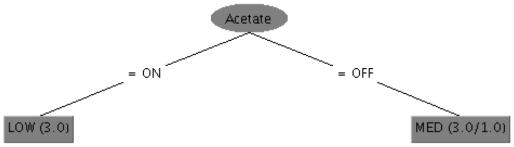
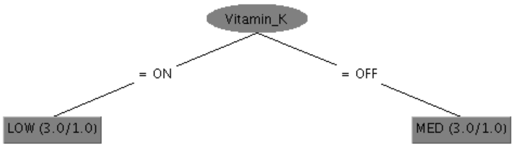
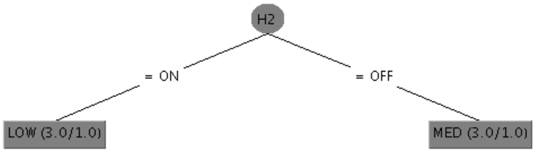
Tree	Frequency	Accuracy	F-measure
 Vitamin_K = ON = OFF B12 Formate = ON = OFF = ON = OFF Acetate NO_GROWTH (15.0/3.0) GROWTH (14.0) B12 = ON = OFF = ON = OFF GROWTH (8.0/1.0) NO_GROWTH (7.0/1.0) H2 GROWTH (8.0/1.0) = ON = OFF Acetate NO_GROWTH (4.0) = ON = OFF GROWTH (2.0) NO_GROWTH (2.0)	17	92.9	0.929
 Vitamin_K = ON = OFF B12 Formate = ON = OFF = ON = OFF Acetate NO_GROWTH (14.0/1.0) GROWTH (14.0) Acetate = ON = OFF = ON = OFF GROWTH (7.0) NO_GROWTH (4.0) GROWTH (7.0/1.0) NO_GROWTH (6.0/1.0)	13	88.4	0.884

Table B.3: *B. theta* Simulation CIT

2-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
	16	73.4	0.643
	11	73.4	0.650
	1	51.6	0.430
 Vitamin_K = ON = OFF LOW (3.0/1.0) MED (3.0/1.0)	1	39.1	0.344
	1	37.7	0.330
3-way			
Tree	**Frequency**	**Accuracy**	**F-measure**

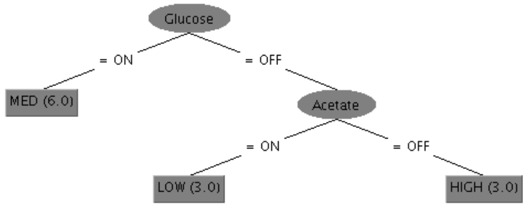
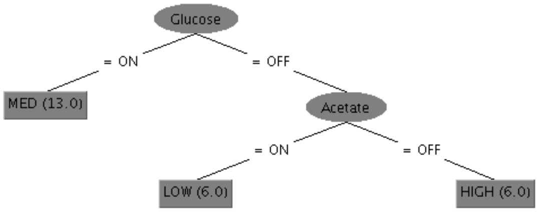
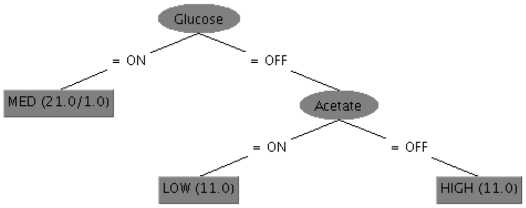
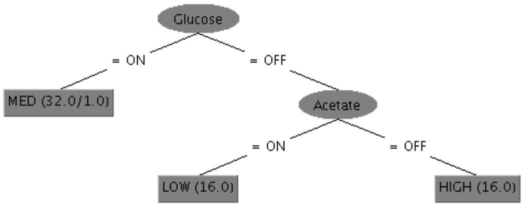
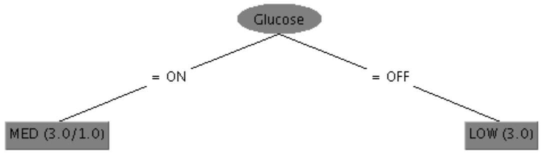
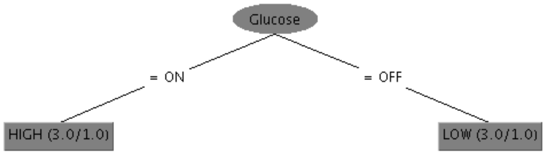
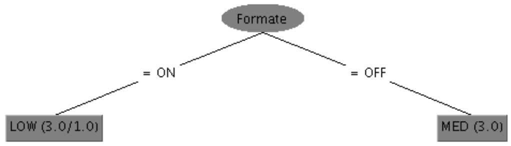
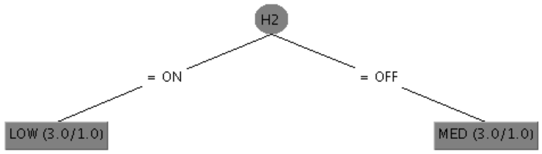
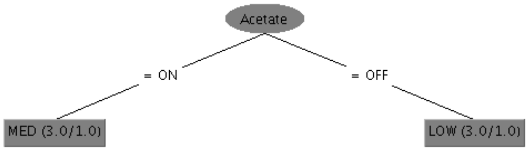
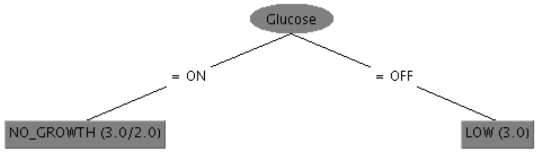
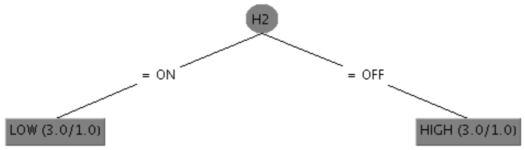
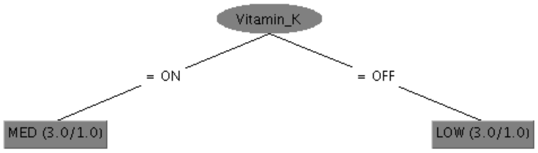
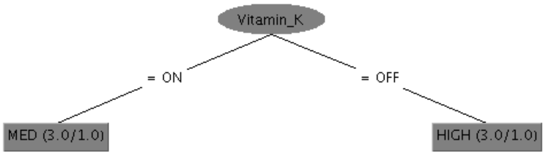
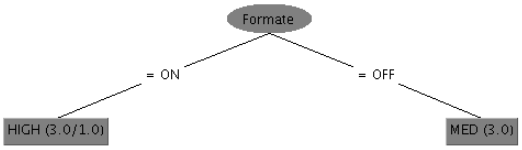
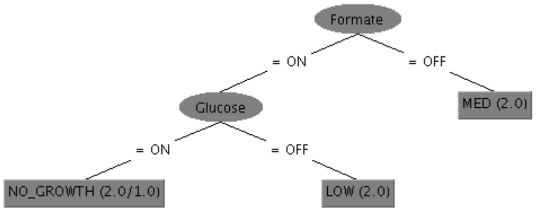
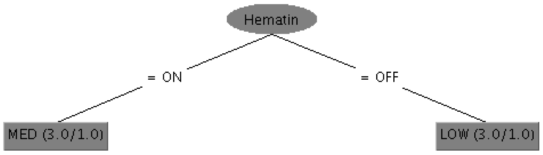
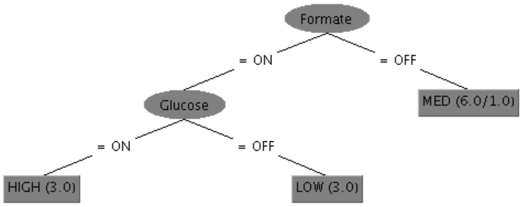
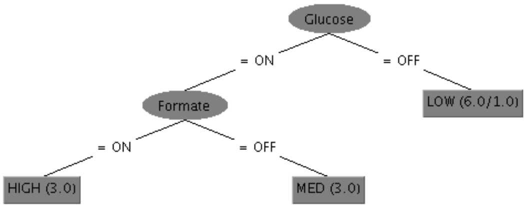
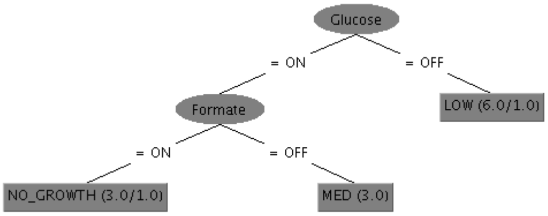
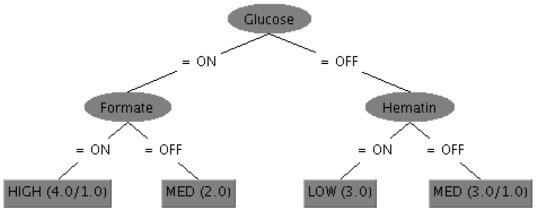
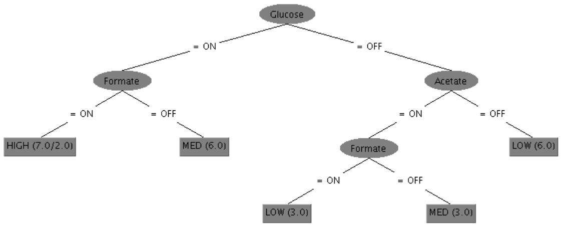
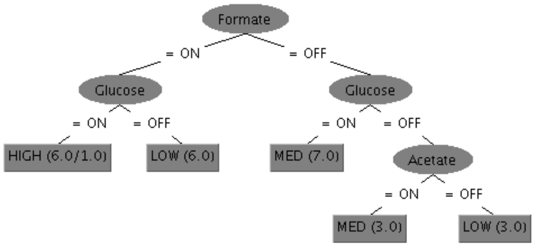
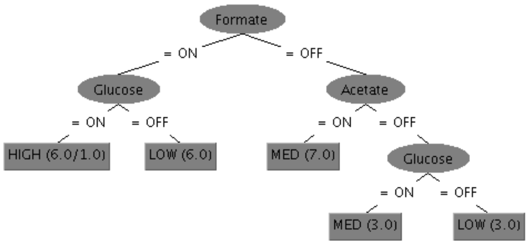
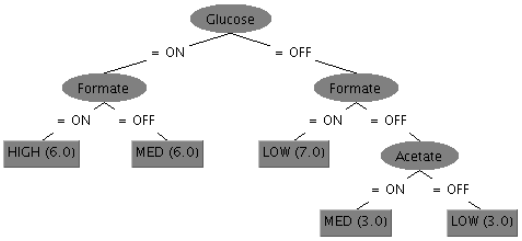
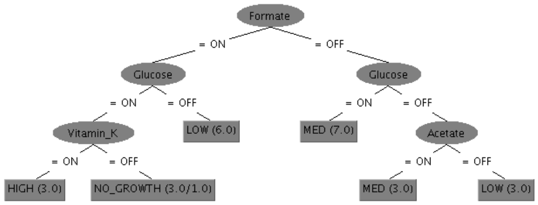
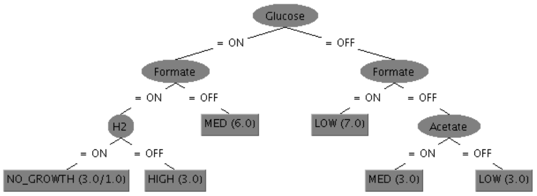
 Glucose = ON = OFF MED (6.0) Acetate = ON = OFF LOW (3.0) HIGH (3.0)	30	98.4	0.984
4-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
 Glucose = ON = OFF MED (13.0) Acetate = ON = OFF LOW (6.0) HIGH (6.0)	30	98.4	0.984
5-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
 Glucose = ON = OFF MED (21.0/1.0) Acetate = ON = OFF LOW (11.0) HIGH (11.0)	30	98.4	0.984
6-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
 Glucose = ON = OFF MED (32.0/1.0) Acetate = ON = OFF LOW (16.0) HIGH (16.0)	30	98.4	0.984

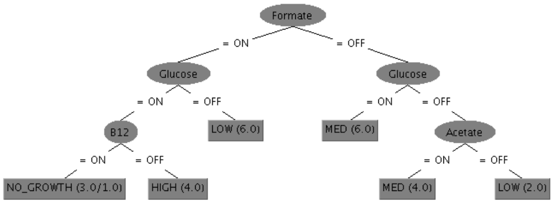
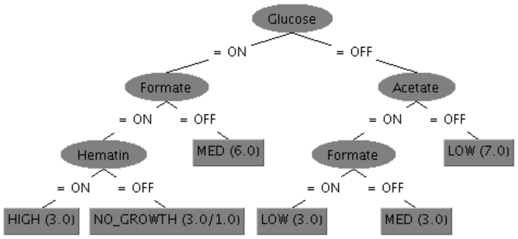
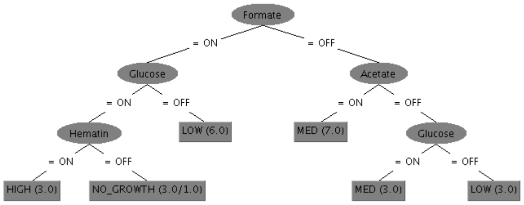
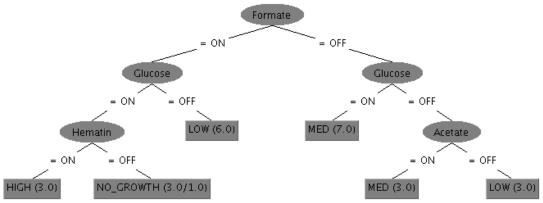
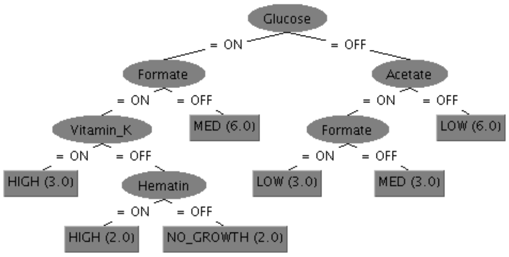
Table B.4: *M. smithii* Simulation CIT

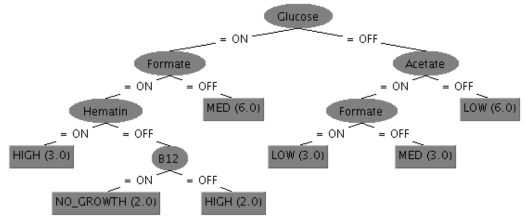
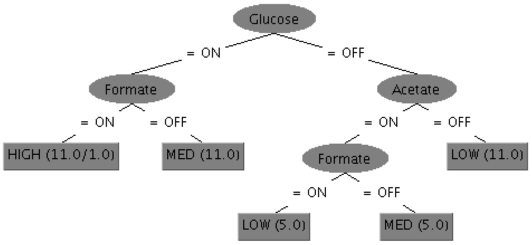
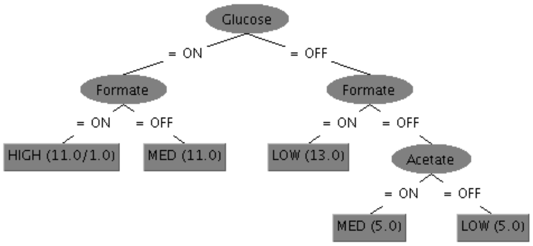
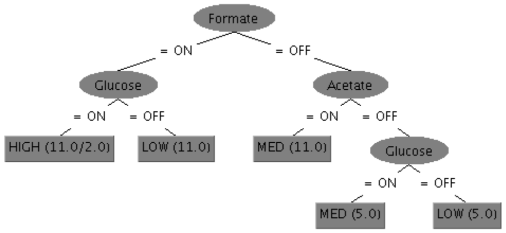
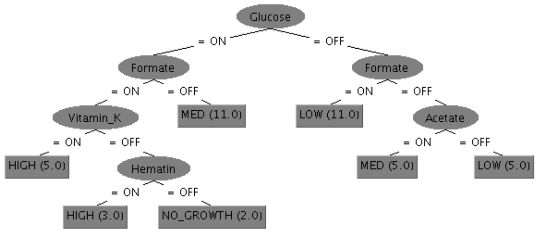
2-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
	9	62.5	0.536
	4	58.6	0.447
	4	62.5	0.536
	3	37.5	0.321
	2	50.0	0.429
 Glucose = ON = OFF NO_GROWTH (3.0/2.0) LOW (3.0)	2	41.4	0.327

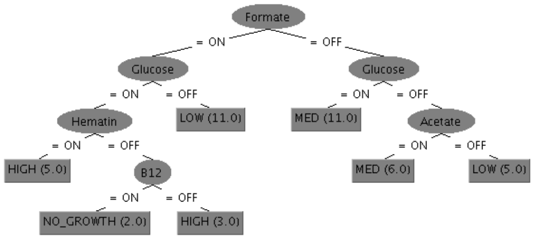
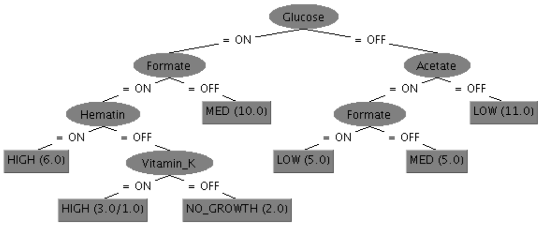
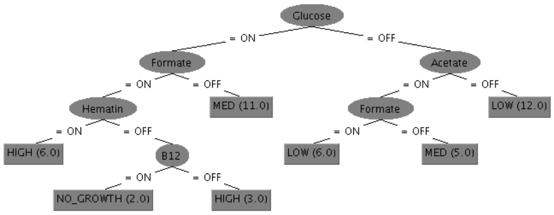
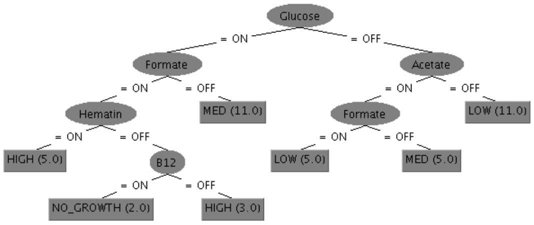
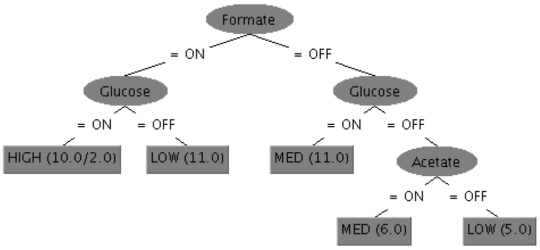
	1	29.7	0.226
 Vitamin_K = ON = OFF MED (3.0/1.0) LOW (3.0/1.0)	1	37.5	0.321
 Vitamin_K = ON = OFF MED (3.0/1.0) HIGH (3.0/1.0)	1	28.9	0.221
	1	58.6	0.447
 Formate = ON = OFF Glucose MED (2.0) = ON = OFF NO_GROWTH (2.0/1.0) LOW (2.0)	1	66.4	0.632
	1	37.5	0.321
3-way			
Tree	**Frequency**	**Accuracy**	**F-measure**

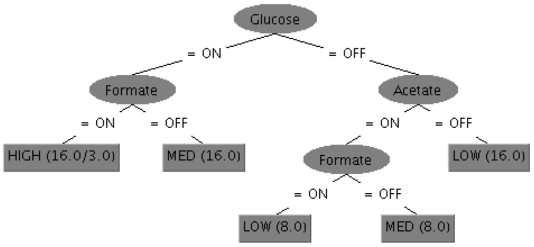
 Formate = ON = OFF Glucose MED (6.0/1.0) = ON = OFF HIGH (3.0) LOW (3.0)	15	83.6	0.814
 Glucose = ON = OFF Formate LOW (6.0/1.0) = ON = OFF HIGH (3.0) MED (3.0)	11	83.6	0.814
 Glucose = ON = OFF Formate LOW (6.0/1.0) = ON = OFF NO_GROWTH (3.0/1.0) MED (3.0)	3	66.4	0.632
 Glucose = ON = OFF Formate Hematin = ON = OFF = ON = OFF HIGH (4.0/1.0) MED (2.0) LOW (3.0) MED (3.0/1.0)	1	71.1	0.686
4-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
 Glucose = ON = OFF Formate Acetate = ON = OFF = ON = OFF HIGH (7.0/2.0) MED (6.0) Formate LOW (6.0) = ON = OFF LOW (3.0) MED (3.0)	12	96.1	0.943

 Formate = ON = OFF Glucose Glucose = ON = OFF = ON = OFF HIGH (6.0/1.0) LOW (6.0) MED (7.0) Acetate = ON = OFF MED (3.0) LOW (3.0)	5	96.1	0.943
 Formate = ON = OFF Glucose Acetate = ON = OFF = ON = OFF HIGH (6.0/1.0) LOW (6.0) MED (7.0) Glucose = ON = OFF MED (3.0) LOW (3.0)	3	96.1	0.943
 Glucose = ON = OFF Formate Formate = ON = OFF = ON = OFF HIGH (6.0) MED (6.0) LOW (7.0) Acetate = ON = OFF MED (3.0) LOW (3.0)	2	96.1	0.943
 Formate = ON = OFF Glucose Glucose = ON = OFF = ON = OFF Vitamin_K LOW (6.0) MED (7.0) Acetate = ON = OFF = ON = OFF HIGH (3.0) NO_GROWTH (3.0/1.0) MED (3.0) LOW (3.0)	1	88.3	0.899
 Glucose = ON = OFF Formate Formate = ON = OFF = ON = OFF H2 MED (6.0) LOW (7.0) Acetate = ON = OFF = ON = OFF NO_GROWTH (3.0/1.0) HIGH (3.0) MED (3.0) LOW (3.0)	1	88.3	0.899

 Formate = ON = OFF Glucose Glucose = ON = OFF = ON = OFF B12 LOW (6.0) MED (6.0) Acetate = ON = OFF = ON = OFF NO_GROWTH (3.0/1.0) HIGH (4.0) MED (4.0) LOW (2.0)	1	89.8	0.912
 Glucose = ON = OFF Formate Acetate = ON = OFF = ON = OFF Hematin MED (6.0) Formate LOW (7.0) = ON = OFF = ON = OFF HIGH (3.0) NO_GROWTH (3.0/1.0) LOW (3.0) MED (3.0)	1	91.4	0.926
 Formate = ON = OFF Glucose Acetate = ON = OFF = ON = OFF Hematin LOW (6.0) MED (7.0) Glucose = ON = OFF = ON = OFF HIGH (3.0) NO_GROWTH (3.0/1.0) MED (3.0) LOW (3.0)	1	91.4	0.926
 Formate = ON = OFF Glucose Glucose = ON = OFF = ON = OFF Hematin LOW (6.0) MED (7.0) Acetate = ON = OFF = ON = OFF HIGH (3.0) NO_GROWTH (3.0/1.0) MED (3.0) LOW (3.0)	1	91.4	0.926
 Glucose = ON = OFF Formate Acetate = ON = OFF = ON = OFF Vitamin_K MED (6.0) Formate LOW (6.0) = ON = OFF = ON = OFF HIGH (3.0) Hematin LOW (3.0) MED (3.0) = ON = OFF HIGH (2.0) NO_GROWTH (2.0)	1	94.5	0.950

 Glucose = ON = OFF Formate Acetate = ON = OFF = ON = OFF Hematin MED (6.0) Formate LOW (6.0) = ON = OFF = ON = OFF HIGH (3.0) B12 LOW (3.0) MED (3.0) = ON = OFF NO_GROWTH (2.0) HIGH (2.0)	1	96.1	0.943
5-way			
Tree	**Frequency**	**Accuracy**	**F-measure**
 Glucose = ON = OFF Formate Acetate = ON = OFF = ON = OFF HIGH (11.0/1.0) MED (11.0) Formate LOW (11.0) = ON = OFF LOW (5.0) MED (5.0)	8	96.1	0.943
 Glucose = ON = OFF Formate Formate = ON = OFF = ON = OFF HIGH (11.0/1.0) MED (11.0) LOW (13.0) Acetate = ON = OFF MED (5.0) LOW (5.0)	6	96.1	0.943
 Formate = ON = OFF Glucose Acetate = ON = OFF = ON = OFF HIGH (11.0/2.0) LOW (11.0) MED (11.0) Glucose = ON = OFF MED (5.0) LOW (5.0)	6	96.1	0.943
 Glucose = ON = OFF Formate Formate = ON = OFF = ON = OFF Vitamin_K MED (11.0) LOW (11.0) Acetate = ON = OFF = ON = OFF HIGH (5.0) Hematin MED (5.0) LOW (5.0) = ON = OFF HIGH (3.0) NO_GROWTH (2.0)	2	94.5	0.950

<table>
<tr><td>
</td><td>2</td><td>96.1</td><td>0.943</td></tr>
<tr><td>
</td><td>2</td><td>94.5</td><td>0.950</td></tr>
<tr><td>
</td><td>2</td><td>96.1</td><td>0.943</td></tr>
<tr><td>
</td><td>1</td><td>96.1</td><td>0.943</td></tr>
<tr><td>
</td><td>1</td><td>96.1</td><td>0.943</td></tr>
<tr><td colspan="4"><b>6-way</b></td></tr>
<tr><td><b>Tree</b></td><td><b>Frequency</b></td><td><b>Accuracy</b></td><td><b>F-measure</b></td></tr>
</table>

Glucose = ON = OFF Formate Acetate = ON = OFF = ON = OFF HIGH (16.0/3.0) MED (16.0) Formate LOW (16.0) = ON = OFF LOW (8.0) MED (8.0)	30	96.1	0.943

Appendix C

B. theta Coverage Model

We present the Reaction Coverage model described in Section 5.5.3. We show the flow of reactions for all 950 reactions in *B. theta* for the 14 (A-N) patterns of coverage. A + indicates forward flow, a − indicates reverse flow, and nothing indicates a net flow of zero.

N	M	L	K	J	I	H	G	F	E	D	C	B	A	
		◆												rxn00342_c0
◆	◆	◆		◆	◆	◆	◆	◆	◆	◆	◆	◆		rxn00340_c0
⊙	◆	⊙	◆	⊙	⊙	⊙	◆	⊙	⊙	⊙	⊙	◆	◆	rxn00285_c0
	◆						◆					◆		rxn00283_c0
⊙	◆	◆	⊙	◆	⊙	◆	◆	⊙	◆	⊙	◆	◆	⊙	rxn00274_c0
◆	◆	◆		◆	◆		◆			◆	◆	◆		rxn00262_c0
⊙	⊙	⊙	⊙		⊙		⊙	⊙		⊙		⊙	⊙	rxn00260_c0
⊙		⊙	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙		⊙	rxn00256_c0
						◆		⊙	◆					rxn00248_c0
	◆					◆	◆					◆	◆	rxn00247_c0
	◆	◆	◆	◆	◆	◆	◆	◆	◆		◆	◆	◆	rxn00239_c0
⊙	⊙		⊙	⊙		⊙	⊙	⊙	⊙		⊙	⊙		rxn00237_c0
◆	◆	◆	◆	⊙	⊙	⊙	⊙	⊙	◆	◆	◆	◆	◆	rxn00224_c0
⊙										⊙			⊙	rxn00216_c0
	◆	◆							◆		◆	◆		rxn00213_c0
◆		◆	◆	◆	◆	◆		◆	◆	◆	◆		◆	rxn00199_c0
		◆							◆		◆			rxn00192_c0
◆													◆	rxn00190_c0
								◆			◆			rxn00187_c0
	⊙	⊙		⊙		⊙	⊙		⊙		⊙	⊙	⊙	rxn00184_c0
⊙	⊙		⊙		⊙	⊙	⊙	⊙	⊙	⊙		⊙	⊙	rxn00182_c0
◆			◆							◆			◆	rxn00179_c0
◆			◆	◆	◆	◆		◆		◆	◆		◆	rxn00165_c0
											◆			rxn00162_c0
	◆	◆		◆				◆			◆			rxn00161_c0
⊙	⊙	⊙	◆	⊙			⊙			⊙	⊙	⊙	⊙	rxn00157_c0
◆		◆	◆	◆	◆	◆		◆	◆	◆	◆			rxn00148_c0
◆		◆	◆	◆	◆	◆		◆	◆	◆	◆		◆	rxn00146_c0
⊙		⊙				⊙				⊙				rxn00139_c0
	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆		rxn00138_c0
	◆					◆						◆		rxn00132_c0
	⊙		⊙	⊙	⊙		⊙	⊙	⊙		⊙	⊙	⊙	rxn00131_c0
			◆	◆	⊙			◆				◆	⊙	rxn00117_c0
			⊙							⊙			⊙	rxn00097_c0
◆	◆	◆	◆	◆	◆	◆	◆		◆	◆		◆	◆	rxn00085_c0
◆	◆	◆	◆	◆	◆	◆	◆	◆		◆	◆	◆	◆	rxn00077_c0
◆	◆	◆	◆						◆	◆	◆	◆	◆	rxn00060_c0
⊙	⊙	⊙	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙	⊙		rxn00056_c0
◆	◆	◆	◆						◆	◆	◆	◆	◆	rxn00029_c0
			◆			◆		◆	◆				◆	rxn00006_c0

N	M	L	K	J	I	H	G	F	E	D	C	B	A	
⊙		⊙	⊙	⊙	◆				⊙		⊙			rxn00910_c0
⊙	◆	◆	◆	◆	◆	◆	◆	◆	◆	⊙	◆	◆		rxn00907_c0
◆										◆				rxn00839_c0
					⊙			⊙						rxn00830_c0
	⊙	⊙		⊙		⊙			⊙		⊙			rxn00799_c0
⊙		⊙	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙			rxn00786_c0
◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	⊙	rxn00785_c0
⊙	◆	⊙	⊙	⊙	⊙	⊙	◆	⊙	⊙	⊙	⊙	◆	⊙	rxn00781_c0
◆	⊙	◆	◆	◆	◆	◆	⊙	◆	◆	◆	◆	⊙	◆	rxn00747_c0
	⊙			⊙		⊙		⊙		⊙				rxn00715_c0
					⊙							⊙		rxn00713_c0
			⊙											rxn00711_c0
⊙							⊙		⊙		⊙			rxn00709_c0
		◆	◆										◆	rxn00708_c0
				◆		◆				⊙				rxn00707_c0
◆	⊙	⊙	◆	⊙	◆	⊙	⊙	◆	⊙	◆	⊙	⊙	◆	rxn00692_c0
			◆											rxn00691_c0
◆	⊙	⊙		⊙	⊙	⊙	⊙	⊙	⊙		⊙	⊙	◆	rxn00690_c0
⊙	⊙		⊙	⊙		⊙	⊙		⊙	⊙	⊙	⊙	⊙	rxn00612_c0
		⊙			⊙			⊙						rxn00611_c0
⊙			⊙	⊙	⊙	⊙		⊙		⊙			⊙	rxn00606_c0
⊙			⊙	⊙	⊙	⊙	⊙	⊙		⊙			⊙	rxn00605_c0
	◆						◆	◆				◆		rxn00604_c0
◆	◆	◆	◆						◆	◆	◆	◆	◆	rxn00599_c0
◆		◆	◆	◆	◆	◆		◆	◆	◆	◆		◆	rxn00566_c0
⊙	◆	⊙				⊙	◆	⊙	⊙	⊙	⊙	◆	⊙	rxn00558_c0
			⊙		⊙			⊙						rxn00551_c0
⊙		⊙		⊙		⊙			⊙	⊙	⊙			rxn00545_c0
	⊙							⊙		⊙				rxn00515_c0
⊙		⊙	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙		⊙	rxn00500_c0
		◆												rxn00474_c0
		◆												rxn00473_c0
		◆							◆		◆			rxn00469_c0
⊙			⊙							⊙			⊙	rxn00467_c0
⊙	◆	⊙	⊙	⊙	⊙	⊙	◆	⊙	⊙	⊙	⊙	◆	⊙	rxn00459_c0
	◆	◆		◆		◆	◆		◆		◆	◆		rxn00420_c0
			◆										◆	rxn00416_c0
◆			◆							◆				rxn00412_c0
		◆		◆		◆	◆		◆		◆	◆		rxn00394_c0
◆	◆	◆	◆		◆	◆	◆	◆	◆	◆	◆	◆	◆	rxn00369_c0
⊙	⊙	⊙			⊙	⊙	⊙		⊙	⊙	⊙		⊙	rxn00368_c0
		◆												rxn00363_c0
⊙	◆		⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙		⊙	rxn00347_c0

N	M	L	K	J	I	H	G	F	E	D	C	B	A	
			◆											rxn01521_c0
◆	◆				◆		◆	◆		◆		◆		rxn01519_c0
									◆					rxn01512_c0
⊙	⊙	⊙		⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	rxn01509_c0
◆														rxn01507_c0
	◆				◆		◆	◆				◆	◆	rxn01476_c0
⊙			⊙											rxn01446_c0
◆	◆	◆		◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	rxn01445_c0
◆		◆	◆	◆	◆	◆		◆	◆	◆	◆		◆	rxn01387_c0
	◆	◆		◆	◆		◆			◆	◆	◆		rxn01360_c0
⊙	⊙	⊙		⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	rxn01353_c0
	◆	⊙	⊙	⊙		⊙	◆		⊙		⊙	◆	⊙	rxn01343_c0
	◆	⊙	⊙	⊙		⊙	◆		⊙		⊙	◆	⊙	rxn01334_c0
⊙	⊙				◆		⊙	◆		⊙		⊙	◆	rxn01333_c0
⊙	⊙		⊙	⊙		⊙	⊙		⊙	⊙	⊙	⊙	⊙	rxn01302_c0
		⊙			⊙			⊙						rxn01301_c0
◆			◆		◆			◆		◆			◆	rxn01300_c0
	◆	◆	◆		◆	◆	◆	◆	◆	◆		◆	◆	rxn01269_c0
◆				◆							◆			rxn01268_c0
			◆											rxn01226_c0
		⊙	⊙	⊙		⊙			⊙		⊙		⊙	rxn01219_c0
		◆	◆	◆		◆			◆		◆		◆	rxn01217_c0
⊙	◆	◆	◆	◆	◆	◆	◆	◆	◆	⊙	◆	◆		rxn01211_c0
◆	◆	◆	◆	◆	⊙	◆	◆	⊙	◆	◆	◆	◆	⊙	rxn01200_c0
			⊙	⊙	⊙	⊙		⊙						rxn01171_c0
			⊙	⊙	⊙	⊙		⊙						rxn01169_c0
◆			◆						⊙	◆				rxn01138_c0
⊙														rxn01127_c0
⊙	⊙	⊙	⊙	⊙		⊙	⊙		⊙	⊙	⊙	⊙	◆	rxn01116_c0
	◆				◆		◆	◆				◆	◆	rxn01115_c0
◆	⊙	◆	◆	◆	◆	◆	⊙	◆	◆	◆	◆	⊙	◆	rxn01106_c0
◆			◆		◆			◆		◆			◆	rxn01102_c0
	◆	◆		◆		◆	◆		◆		◆	◆		rxn01101_c0
◆	⊙	◆	◆	◆	◆	◆	⊙	◆	◆	◆	◆	⊙	◆	rxn01100_c0
◆			◆		◆			◆		◆			◆	rxn01069_c0
◆	⊙	⊙	◆	⊙	◆	⊙	⊙	◆	⊙	◆	⊙	⊙	◆	rxn01068_c0
⊙			⊙		⊙			⊙		⊙			⊙	rxn01014_c0
⊙			⊙		⊙			⊙		⊙			⊙	rxn01011_c0
◆		◆	◆	◆	◆	◆		◆	◆	◆	◆		◆	rxn00973_c0
		⊙		⊙										rxn00931_c0
⊙	⊙		⊙		⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	rxn00929_c0
	◆	◆		◆	◆	◆	◆	◆	◆		◆	◆	◆	rxn00927_c0
	◆			◆		◆	◆					◆		rxn00917_c0
◆		◆	◆		◆			◆	◆	◆	◆		◆	rxn00916_c0
◆			◆											rxn00913_c0

N	M	L	K	J	I	H	G	F	E	D	C	B	A	
				⊙										rxn05312_c0
◆	◆				◆		◆	◆		◆		◆		rxn05236_c0
	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	rxn05234_c0
	◆						◆					◆		rxn05226_c0
⊙	⊙	⊙		⊙		⊙	⊙		⊙	⊙	⊙	⊙		rxn05209_c0
		◆	◆								◆			rxn05150_c0
				◆	◆	◆	◆	◆						rxn05148_c0
	⊙	⊙	◆	⊙	◆	⊙	⊙	◆	⊙		⊙	⊙	◆	rxn05145_c0
⊙	⊙		⊙	⊙					⊙	⊙	⊙	⊙	⊙	rxn04996_c0
	⊙				⊙	⊙	⊙	⊙		⊙		⊙	⊙	rxn04954_c0
◆	◆	◆		◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	rxn04783_c0
◆	◆	◆	◆	◆		◆	◆		◆	◆	◆	◆	◆	rxn04113_c0
							⊙							rxn04020_c0
⊙		⊙	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙		⊙	rxn03978_c0
⊙	⊙		⊙		⊙		⊙	⊙		⊙		⊙	⊙	rxn03012_c0
			◆							⊙				rxn03004_c0
	⊙	⊙		⊙		⊙	⊙		⊙		⊙	⊙		rxn02914_c0
	◆	◆	◆	◆	◆	◆	◆	◆	◆		◆	◆		rxn02503_c0
◆	⊙	⊙	◆	⊙	◆	⊙	⊙	◆	⊙	◆	⊙	⊙	◆	rxn02484_c0
	⊙	⊙		⊙	⊙	⊙	⊙	⊙	⊙		⊙	⊙		rxn02465_c0
									◆					rxn02400_c0
			⊙	⊙	⊙	⊙		⊙					⊙	rxn02380_c0
⊙			⊙							⊙			⊙	rxn02373_c0
	◆			⊙	⊙	⊙	⊙	⊙				◆		rxn02304_c0
	◆			⊙	⊙	⊙	⊙	⊙				◆		rxn02303_c0
◆	◆	◆	◆	⊙	⊙	⊙	⊙	⊙	◆	◆	◆	◆	◆	rxn02288_c0
◆	◆	◆	◆						◆	◆	◆	◆	◆	rxn02264_c0
	◆	◆	◆	◆	◆	◆	◆	◆	◆		◆	◆		rxn02201_c0
◆							⊙			◆		⊙	◆	rxn02200_c0
⊙	⊙		⊙		⊙		⊙	⊙		⊙		⊙	⊙	rxn01991_c0
			⊙	⊙	⊙		◆							rxn01977_c0
					◆								◆	rxn01975_c0
		◆		◆		◆			◆		◆			rxn01974_c0
							⊙							rxn01967_c0
	◆	◆		◆	◆	◆	◆	◆	◆		◆	◆		rxn01917_c0
◆														rxn01859_c0
			◆											rxn01799_c0
		⊙	⊙	⊙		⊙			⊙		⊙		⊙	rxn01673_c0
⊙										⊙				rxn01649_c0
									◆					rxn01647_c0
	⊙	⊙		⊙	⊙	⊙	⊙	⊙	⊙		⊙	⊙		rxn01637_c0
	◆			◆	◆	◆	◆	◆				◆		rxn01636_c0
◆			⊙											rxn01548_c0
◆										◆				rxn01545_c0

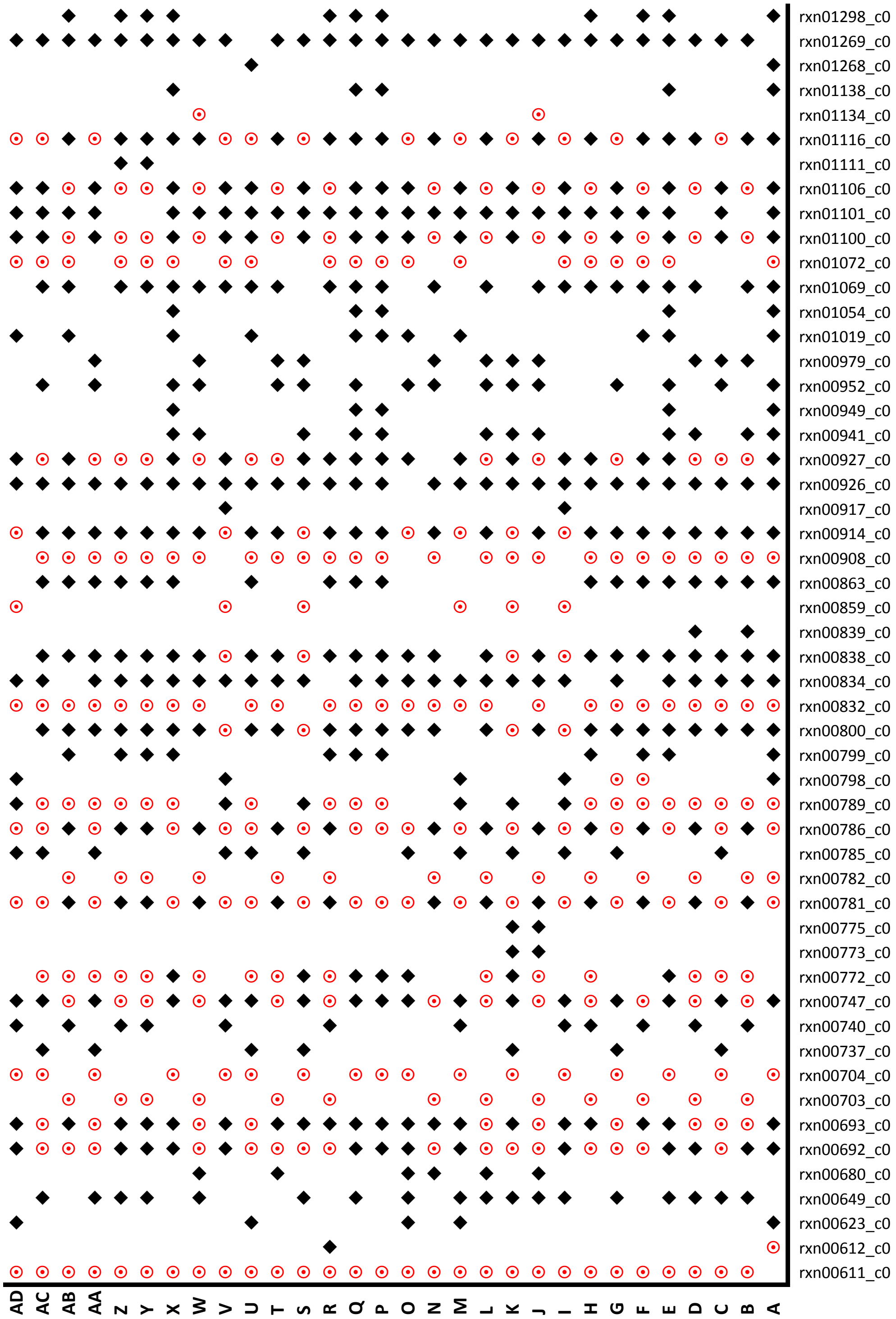
	N	M	L	K	J	I	H	G	F	E	D	C	B	A
rxn12220_c0	◆		◆	◆						◆	◆	◆		◆
rxn11268_c0								⊙					⊙	
rxn10806_c0	◆													
rxn10168_c0			◆	◆							◆	◆		
rxn10113_c0					◆									
rxn10060_c0	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	◆	⊙	⊙	⊙	⊙
rxn10052_c0	⊙	⊙	⊙		⊙	⊙	⊙		⊙	⊙		⊙		
rxn10030_c0				◆			◆		◆	◆				◆
rxn09997_c0	◆	⊙	⊙	◆	⊙	◆	⊙	⊙	◆	⊙	◆	⊙	⊙	◆
rxn09562_c0	◆										◆			
rxn09468_c0	⊙	⊙	⊙		⊙	⊙		⊙			⊙	⊙	⊙	
rxn09448_c0				◆			◆		◆	◆				◆
rxn09272_c0	◆	⊙	◆	⊙	◆	◆	◆	⊙	◆	◆	◆	◆	⊙	
rxn09188_c0	◆		◆	◆	◆	◆	◆		◆	◆	◆	◆		◆
rxn09180_c0	◆		◆	◆						◆	◆	◆		◆
rxn09167_c0	◆	◆		◆		◆		◆	◆		◆		◆	
rxn08976_c0					◆									
rxn08688_c0		⊙						⊙					⊙	
rxn08352_c0			◆			◆	◆	◆	◆					
rxn08336_c0	◆													
rxn08335_c0				◆			◆		◆	◆				◆
rxn08291_c0	⊙		⊙	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙		⊙
rxn08131_c0		◆	◆		◆		◆	◆		◆		◆	◆	
rxn07441_c0			⊙		⊙		⊙			⊙		⊙		
rxn06673_c0				◆			◆		◆	◆				◆
rxn06672_c0				◆			◆		◆	◆				◆
rxn06600_c0				◆		◆	◆		◆	◆				◆
rxn06493_c0				◆		◆	◆		◆	◆				◆
rxn06377_c0				◆		◆	◆		◆	◆				◆
rxn05939_c0	⊙	◆	⊙		⊙	⊙	⊙	◆	⊙	⊙	⊙	⊙	◆	⊙
rxn05938_c0	◆	⊙	◆	◆	◆	◆	◆	⊙	◆	◆	◆	◆	⊙	◆
rxn05893_c0	◆		◆	◆	◆	◆	◆		◆	◆	◆	◆		◆
rxn05667_c0			⊙		⊙		⊙	⊙		⊙		⊙	⊙	
rxn05596_c0	⊙	⊙			⊙	⊙	⊙	⊙	⊙	⊙			⊙	⊙
rxn05555_c0								◆						◆
rxn05488_c0			◆		◆		◆			◆		◆		
rxn05468_c0	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙
rxn05467_c0	◆	⊙	◆	◆	◆	◆	◆		◆	◆	◆	◆	⊙	◆
rxn05315_c0	⊙	⊙			⊙	⊙	⊙	⊙	⊙	⊙	⊙		⊙	⊙
rxn05313_c0	◆	◆	◆		◆		◆	◆		◆	◆	◆	◆	

Appendix D

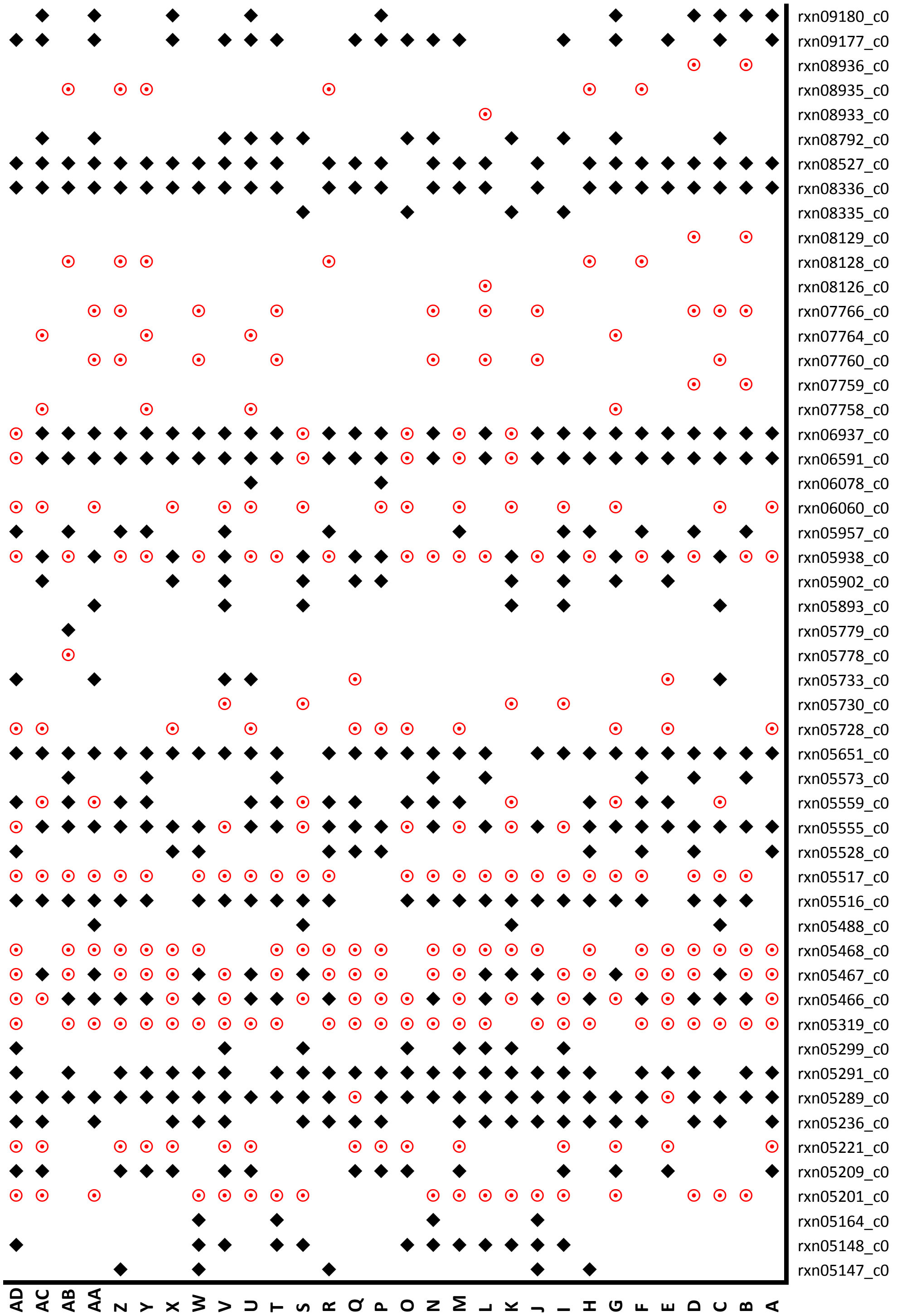
M. smithii Coverage Model

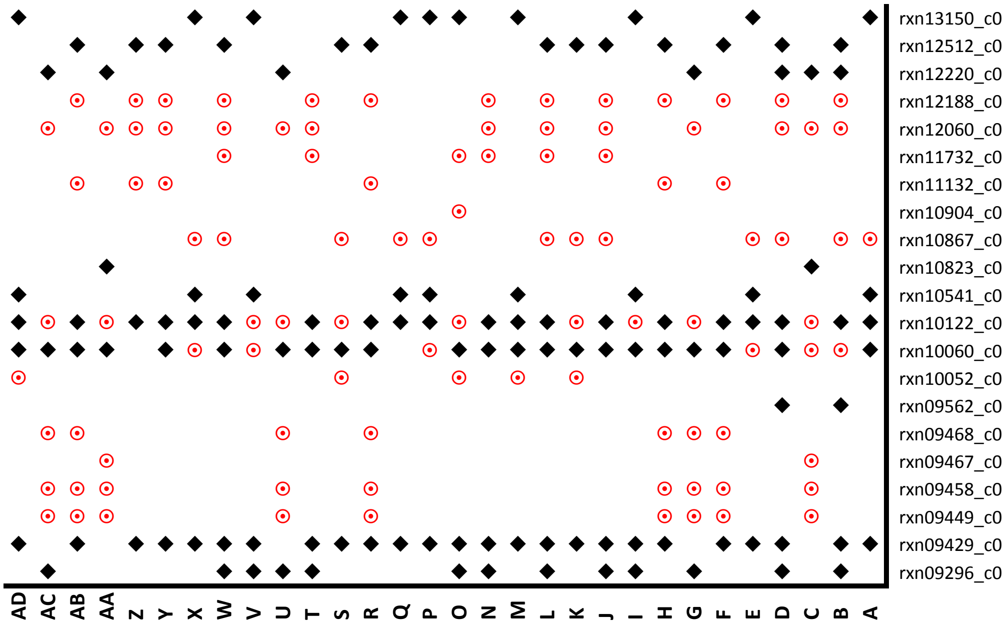
We present the Reaction Coverage model described in Section 5.5.3. We show the flow of reactions for all 908 reactions in *M. smithii* for the 30 (A-AD) patterns of coverage. A + indicates forward flow, a − indicates reverse flow, and nothing indicates a net flow of zero.

AD	AC	AB	AA	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	
⊙								⊙							⊙		⊙				⊙									rxn00599_c0
⊙	⊙		⊙			⊙		⊙	⊙		⊙		⊙	⊙	⊙		⊙		⊙		⊙		⊙		⊙		⊙		⊙	rxn00558_c0
											◆	◆														◆				rxn00549_c0
		◆		◆	◆	◆	◆			◆		◆	◆	◆		◆		◆		◆		◆		◆	◆				◆	rxn00548_c0
	⊙	⊙	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙			⊙		⊙	⊙	⊙	⊙	⊙		⊙		⊙	rxn00542_c0
⊙			⊙								⊙				⊙		⊙		⊙								⊙			rxn00541_c0
◆	◆	◆		◆	◆	◆		◆	◆			◆	◆	◆	◆		◆				◆	◆	◆	◆	◆				◆	rxn00539_c0
		⊙		⊙	⊙		⊙			⊙		⊙				⊙		⊙		⊙		⊙		⊙		⊙		⊙		rxn00519_c0
		⊙		⊙	⊙		⊙			⊙		⊙				⊙		⊙		⊙		⊙		⊙		⊙		⊙		rxn00515_c0
⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙		⊙		⊙	rxn00507_c0
	⊙		⊙	⊙	⊙		⊙	⊙		⊙	⊙	⊙				⊙		⊙	⊙	⊙	⊙	⊙	⊙			⊙	⊙	⊙		rxn00465_c0
⊙	⊙	◆	⊙	◆	◆	⊙	◆	⊙	⊙	◆	⊙	◆	⊙	⊙	⊙	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	⊙	rxn00459_c0
◆	◆		◆	◆	◆		◆	◆	◆		◆				◆		◆	◆	◆	◆	◆		◆			◆	◆	◆		rxn00423_c0
◆	◆	◆	◆			◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆		◆		◆	rxn00420_c0
	◆	◆		◆	◆		◆			◆		◆				◆		◆		◆		◆	◆	◆	◆	◆		◆		rxn00414_c0
◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆		◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	rxn00412_c0
															◆															rxn00410_c0
		◆		◆	◆				◆	◆					◆	◆	◆	◆		◆					◆			◆		rxn00407_c0
											◆								◆											rxn00383_c0
◆	◆	◆	◆	◆	◆	◆	◆	◆		◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	rxn00371_c0
			◆				◆			◆						◆		◆		◆						◆	◆	◆		rxn00333_c0
											◆								◆											rxn00324_c0
◆	◆		◆			◆		◆	◆		◆		◆	◆	◆		◆		◆		◆		◆		◆		◆		◆	rxn00305_c0
			⊙				⊙			⊙	⊙				⊙	⊙		⊙	⊙	⊙						⊙	⊙	⊙		rxn00268_c0
◆	◆	⊙	◆	⊙	⊙	◆	⊙	◆	◆	⊙	◆	⊙	◆	◆	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	rxn00250_c0
	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆		◆	◆	◆		◆		◆		◆	◆	◆	◆	◆	◆		◆		◆	rxn00239_c0
⊙	◆	◆	◆	◆	◆	◆		⊙	◆		⊙	◆	◆	◆	⊙		⊙		⊙		⊙	◆	◆	◆	◆	◆	◆	◆	◆	rxn00224_c0
		◆		◆	◆		◆			◆		◆				◆		◆		◆		◆		◆		◆		◆		rxn00223_c0
◆	◆	◆	◆				◆	◆	◆		◆	◆	◆	◆			◆			◆	◆	◆	◆	◆	◆	◆	◆		◆	rxn00187_c0
		⊙		⊙	⊙	⊙	⊙			⊙		⊙	⊙	⊙		⊙		⊙		⊙		⊙		⊙	⊙				⊙	rxn00173_c0
							◆											◆		◆										rxn00171_c0
	◆	◆	◆	◆	◆		◆			◆						◆		◆				◆	◆	◆			◆	◆		rxn00147_c0
◆	⊙	◆	⊙	⊙	⊙	◆	⊙	◆	⊙	⊙	◆	◆	◆	◆	◆		◆	⊙	◆	⊙	◆	◆	⊙	◆	◆	⊙	⊙	⊙	◆	rxn00141_c0
						◆			◆					◆															◆	rxn00140_c0
	◆	◆	◆	◆	◆	⊙	◆		◆	◆		◆	⊙	⊙	⊙	◆		◆		◆		◆	◆	◆	⊙	◆	◆	◆	⊙	rxn00139_c0
◆	◆			◆				◆	◆	◆	◆	◆	◆		◆	◆	◆	◆	◆		◆	◆	◆		◆					rxn00138_c0
◆	⊙	◆	⊙			◆	⊙	◆	⊙		◆	◆	◆	◆	◆	◆	◆	⊙	◆	⊙	◆	◆	⊙	◆	◆	⊙	⊙	⊙	◆	rxn00126_c0
◆			◆		⊙	◆		◆	◆	⊙	◆		◆	◆	◆	⊙	◆		◆		◆						◆	⊙	◆	rxn00113_c0
		◆	◆		◆	◆	◆				⊙		⊙	◆				⊙	⊙	◆				◆	⊙	◆	◆	◆	◆	rxn00105_c0
⊙	⊙	◆	⊙	◆		⊙	◆	⊙	⊙		⊙	◆	⊙	⊙	⊙		⊙	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	⊙		⊙	rxn00102_c0
◆	◆		◆					◆	◆						◆		◆				◆		◆				◆			rxn00090_c0
◆	◆					◆		◆	◆		◆			◆	◆		◆		◆		◆		◆						◆	rxn00086_c0
				◆	◆	◆				◆					◆	◆		◆	◆									◆		rxn00085_c0
				◆		◆		◆					◆	◆											◆		◆	◆		rxn00077_c0
						◆			◆					◆															◆	rxn00065_c0
⊙	◆	◆	◆	◆	◆	◆	◆	⊙	◆	◆	⊙	◆	◆	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	◆	◆	◆	◆	◆	◆	◆	rxn00060_c0
⊙	◆	◆	◆	◆	◆	◆	◆	⊙	◆	◆	⊙	◆	◆	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	◆	◆	◆	◆	◆	◆	◆	rxn00029_c0
								◆			◆								◆		◆									rxn00023_c0
										⊙						⊙														rxn00022_c0
				◆	◆																									rxn00018_c0
							⊙													⊙										rxn00007_c0


rxn01298_c0
rxn01269_c0
rxn01268_c0
rxn01138_c0
rxn01134_c0
rxn01116_c0
rxn01111_c0
rxn01106_c0
rxn01101_c0
rxn01100_c0
rxn01072_c0
rxn01069_c0
rxn01054_c0
rxn01019_c0
rxn00979_c0
rxn00952_c0
rxn00949_c0
rxn00941_c0
rxn00927_c0
rxn00926_c0
rxn00917_c0
rxn00914_c0
rxn00908_c0
rxn00863_c0
rxn00859_c0
rxn00839_c0
rxn00838_c0
rxn00834_c0
rxn00832_c0
rxn00800_c0
rxn00799_c0
rxn00798_c0
rxn00789_c0
rxn00786_c0
rxn00785_c0
rxn00782_c0
rxn00781_c0
rxn00775_c0
rxn00773_c0
rxn00772_c0
rxn00747_c0
rxn00740_c0
rxn00737_c0
rxn00704_c0
rxn00703_c0
rxn00693_c0
rxn00692_c0
rxn00680_c0
rxn00649_c0
rxn00623_c0
rxn00612_c0
rxn00611_c0
AD
AC
AB
AA
Z
Y
X
W
V
U
T
S
R
Q
P
O
N
M
L
K
J
I
H
G
F
E
D
C
B
A

AD	AC	AB	AA	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	
						◆					◆		◆	◆					◆						◆					rxn05143_c0
						⊙					⊙		⊙	⊙					⊙						⊙					rxn05142_c0
									◆																				◆	rxn05119_c0
◆	◆	◆	◆	◆	◆	◆	◆	◆		◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆	◆		rxn05117_c0
		⊙		⊙	⊙					⊙	⊙	⊙				⊙			⊙			⊙		⊙						rxn04954_c0
							⊙			⊙					⊙	⊙		⊙		⊙										rxn04794_c0
								⊙			⊙								⊙		⊙									rxn03978_c0
																						◆								rxn03971_c0
																						⊙								rxn03970_c0
		◆										◆										◆		◆		◆		◆		rxn03638_c0
⊙	◆	◆	◆	◆	◆	◆		⊙	◆		⊙	◆	◆	◆			⊙		⊙		⊙	◆	◆	◆	◆	◆	◆	◆	◆	rxn03175_c0
◆	◆	◆	◆	◆	◆	◆	◆		◆	◆		◆	◆	◆	◆	◆	◆	◆		◆		◆	◆	◆	◆	◆	◆	◆	◆	rxn03137_c0
◆	⊙	⊙	⊙	⊙	⊙	⊙		◆	⊙		◆	⊙	⊙	⊙			◆		◆		◆	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	rxn03135_c0
⊙	⊙	⊙	⊙	⊙	⊙		⊙		⊙	⊙		⊙		⊙	⊙	⊙	⊙	⊙		⊙		⊙	⊙	⊙		⊙	⊙	⊙		rxn03068_c0
			◆		◆		◆											◆		◆				◆					⊙	rxn02929_c0
⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙		rxn02928_c0
⊙	⊙	⊙	⊙			⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙		⊙		⊙	rxn02914_c0
⊙	◆	◆	◆	◆	◆	◆		⊙	◆		⊙	◆	◆	◆			⊙		⊙		⊙	◆	◆	◆	◆	◆	◆	◆	◆	rxn02835_c0
⊙	◆	◆	◆	◆	◆	◆		⊙	◆		⊙	◆	◆	◆			⊙		⊙		⊙	◆	◆	◆	◆	◆	◆	◆	◆	rxn02834_c0
⊙	◆	◆	◆	◆	◆	◆		⊙	◆		⊙	◆	◆	◆			⊙		⊙		⊙	◆	◆	◆	◆	◆	◆	◆	◆	rxn02473_c0
◆	⊙	⊙	⊙	⊙	⊙	⊙		◆	⊙		◆	⊙	⊙	⊙			◆		◆		◆	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	rxn02320_c0
⊙		◆		◆	◆			⊙			⊙	◆	◆		⊙		⊙		⊙		⊙	◆		◆	◆					rxn02304_c0
⊙		◆		◆	◆	◆		⊙			⊙	◆	◆	◆	⊙		⊙		⊙		⊙	◆		◆	◆				◆	rxn02303_c0
⊙	◆	◆	◆	◆	◆	◆		⊙	◆		⊙	◆	◆	◆	⊙		⊙		⊙		⊙	◆	◆	◆	◆	◆	◆	◆	◆	rxn02288_c0
⊙	◆	◆	◆	◆	◆	◆	◆	⊙	◆	◆	⊙	◆	◆	◆	⊙	◆	⊙	◆	⊙	◆	⊙	◆	◆	◆	◆	◆	◆	◆	◆	rxn02264_c0
◆	◆	◆	◆	◆	◆		◆		◆	◆		◆		◆	◆	◆	◆	◆		◆		◆	◆	◆		◆	◆	◆		rxn02187_c0
						◆		◆			◆		◆						◆		◆				◆				◆	rxn02186_c0
⊙	◆	◆	◆	◆	◆	◆		⊙	◆		⊙	◆	◆	◆			⊙		⊙		⊙	◆	◆	◆	◆	◆	◆	◆	◆	rxn02160_c0
	◆	◆	◆	◆	◆	◆			◆			◆	◆	◆								◆	◆	◆	◆	◆	◆	◆	◆	rxn02159_c0
◆								◆			◆		◆		◆		◆		◆		◆				◆					rxn02155_c0
	⊙			⊙					⊙	⊙		⊙				⊙		⊙				⊙	⊙							rxn02154_c0
◆	◆		◆	◆	◆	◆	◆		◆	◆	◆		◆	◆	◆	◆	◆	◆	◆	◆			◆		◆	◆	◆	◆	◆	rxn01961_c0
	⊙	⊙	⊙	⊙	⊙	⊙	⊙		⊙	⊙		⊙		⊙		⊙		⊙		⊙		⊙	⊙	⊙		⊙	⊙	⊙	⊙	rxn01792_c0
◆								◆			◆		◆		◆		◆		◆		◆				◆					rxn01791_c0
		◆		◆	◆				◆	◆					◆	◆									◆			◆		rxn01672_c0
		⊙	⊙		⊙	⊙	⊙							⊙						⊙				⊙		⊙	⊙	⊙	⊙	rxn01669_c0
						⊙							⊙	⊙											⊙				⊙	rxn01649_c0
◆	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	◆	⊙	⊙	⊙	◆	⊙	◆	⊙	◆	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	rxn01629_c0
⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	rxn01547_c0
		⊙		⊙	⊙							⊙										⊙		⊙						rxn01545_c0
◆	◆	◆	◆	◆	◆		◆		◆	◆	◆	◆	◆		◆	◆	◆	◆	◆	◆	◆	◆	◆	◆		◆			◆	rxn01512_c0
		◆		◆	◆		◆			◆		◆				◆		◆		◆		◆		◆		◆		◆		rxn01492_c0
		⊙										⊙										⊙		⊙		⊙		⊙		rxn01485_c0
◆	◆			◆						◆						◆	◆						◆			◆	◆	◆		rxn01452_c0
⊙	⊙			⊙						⊙						⊙	⊙						⊙			⊙	⊙	⊙		rxn01451_c0
																										◆		◆		rxn01343_c0
																										◆		◆		rxn01334_c0
◆		◆		◆	◆			◆				◆					◆				◆	◆		◆		◆		◆		rxn01304_c0
	◆		◆			◆	◆		◆	◆	◆		◆	◆	◆	◆		◆	◆	◆			◆		◆		◆		◆	rxn01303_c0
								◆													◆								⊙	rxn01302_c0
⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙		rxn01301_c0
	◆	◆		◆	◆	◆	◆	◆	◆	◆		◆	◆	◆		◆		◆		◆	◆	◆	◆	◆	◆	◆		◆	◆	rxn01300_c0


rxn09180_c0
rxn09177_c0
rxn08936_c0
rxn08935_c0
rxn08933_c0
rxn08792_c0
rxn08527_c0
rxn08336_c0
rxn08335_c0
rxn08129_c0
rxn08128_c0
rxn08126_c0
rxn07766_c0
rxn07764_c0
rxn07760_c0
rxn07759_c0
rxn07758_c0
rxn06937_c0
rxn06591_c0
rxn06078_c0
rxn06060_c0
rxn05957_c0
rxn05938_c0
rxn05902_c0
rxn05893_c0
rxn05779_c0
rxn05778_c0
rxn05733_c0
rxn05730_c0
rxn05728_c0
rxn05651_c0
rxn05573_c0
rxn05559_c0
rxn05555_c0
rxn05528_c0
rxn05517_c0
rxn05516_c0
rxn05488_c0
rxn05468_c0
rxn05467_c0
rxn05466_c0
rxn05319_c0
rxn05299_c0
rxn05291_c0
rxn05289_c0
rxn05236_c0
rxn05221_c0
rxn05209_c0
rxn05201_c0
rxn05164_c0
rxn05148_c0
rxn05147_c0
AD
AC
AB
AA
Z
Y
X
W
V
U
T
S
R
Q
P
O
N
M
L
K
J
I
H
G
F
E
D
C
B
A


rxn13150_c0
rxn12512_c0
rxn12220_c0
rxn12188_c0
rxn12060_c0
rxn11732_c0
rxn11132_c0
rxn10904_c0
rxn10867_c0
rxn10823_c0
rxn10541_c0
rxn10122_c0
rxn10060_c0
rxn10052_c0
rxn09562_c0
rxn09468_c0
rxn09467_c0
rxn09458_c0
rxn09449_c0
rxn09429_c0
rxn09296_c0
AD
AC
AB
AA
Z
Y
X
W
V
U
T
S
R
Q
P
O
N
M
L
K
J
I
H
G
F
E
D
C
B
A

Bibliography

[1] Acts. `http://csrc.nist.gov/groups/SNS/acts/`.

[2] I. F. Akyildiz, M. Pierobon, S. Balasubramaniam, and Y. Koucheryavy. The internet of bio-nano things. *IEEE Communications Magazine*, 53(3):32–40, March 2015.

[3] Benjamin Allen. The DOE systems biology knowledgebase: Progress toward a system for collaborative and reproducible inference and modeling of biological function. Postdoctoral Poster, 2016.

[4] Sven Apel, Don Batory, Christian Kstner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer Publishing Company, Incorporated, 2013.

[5] Masanori Arita. The metabolic world of *escherichia coli* is not small. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 101 of *PNAS*, pages 1543–1547, 2004.

[6] Gino J. E. Baart and Dirk E. Martens. Genome-scale metabolic models: reconstruction and analysis. *Methods in Molecular Biology*, 799:107–126, 2012.

[7] BioSIMP experimental artifacts. `http://cse.unl.edu/~myra/artifacts/BioSIMP/`.

[8] Yizhi Cai, Matthew W. Lux, Laura Adam, and Jean Peccoud. Modeling structure-function relationships in synthetic DNA sequences using attribute grammars. *PLoS Computational Biology*, 5(10), 2009.

[9] Franck Carbonero, Ann C Benefiel, and H Rex Gaskins. Contributions of the microbial hydrogen economy to colonic homeostasis. *Nature reviews. Gastroenterology & hepatology*, 9(9):504–18, sep 2012.

[10] Santo Carino and James H. Andrews. Dynamically testing guis using ant colony optimization (t). *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 00(undefined):138–148, 2015.

[11] Mikaela Cashman, Jennie L. Catlett, Myra B. Cohen, Nicole Buan, Zahmeeth Sakkaff, Massimiliano Pierobon, and Christine Kelley. Sampling and Inference in Configurable Biological Systems: A Software Testing Perspective. Technical Report TR-UNL-CSE-2016-0007, University of Nebraska-Lincoln, 2016.

[12] George M Church, Michael B Elowitz, Christina D Smolke, Christopher a Voigt, and Ron Weiss. Realizing the potential of synthetic biology. *Nature reviews. Molecular cell biology*, 15(4):289–94, 2014.

[13] Lori A Clarke, Andy Podgurski, Debra J. Richardson, and Steven J. Zeil. A formal evaluation of data flow path selection criteria. In *IEEE Transactions on Software Engineering*, volume 15, pages 1318–1332, 1989.

[14] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, 1997.

[15] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering*, 34(5):633–650, 2008.

[16] M. Dorigo. *Optimization, Learning and Natural Algorithms.* PhD thesis, Politecnico di Milano, Italy, 1992.

[17] Samuel J. Ellis, Eric R. Henderson, Titus H. Klinge, James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, Divita Mathur, and Andrew S. Miner. Automated requirements analysis for a molecular watchdog timer. In *International conference on Automated software engineering (ASE)*, pages 767–778, 2014.

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

[19] Automatic test suite generation for java. `http://www.evosuite.org`.

[20] Karoline Faust, Didier Croes, and Jacques van Helden. Prediction of metabolic pathways from genome-scale metabolic networks. *Elsevier*, 2(105):109–121, 2011.

[21] Michael S Ferry, Jeff Hasty, and Natalie A Cookson. Synthetic biology approaches to biofuel production. *Biofuels*, 3(1):9–12, 2012.

[22] Jasmin Fisher and Thomas A Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11):1239–1249, November 2007.

[23] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Fifteenth International Conference on Machine Learning*, pages 144–151, 1998.

[24] Michal Galdzicki, Kevin P Clancy, Ernst Oberortner, Matthew Pocock, Jacqueline Y Quinn, Cesar A Rodriguez, Nicholas Roehner, Mandy L Wilson, Laura Adam, J Christopher Anderson, Bryan A Bartley, Jacob Beal, Deepak Chandran, Joanna Chen, Douglas Densmore, Drew Endy, Raik Grünberg, Jennifer Hallinan, Nathan J Hillson, Jeffrey D Johnson, Allan Kuchinsky, Matthew Lux, Goksel Misirli, Jean Peccoud, Hector A Plahar, Evren Sirin, Guy-Bart Stan, Alan Villalobos, Anil Wipat, John H Gennari, Chris J Myers, and Herbert M Sauro. The synthetic biology open language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nature Biotechnology*, 32(6):545–550, 06 2014.

[25] Brady J. Garvin, Myra B. Cohen, and Matthew B. Dwyer. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering*, 16(1):61–102, 2011.

[26] David E. Goldberg and John H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3:95–99, 1988.

[27] James M. Graham. The biological terraforming of mars: Planetary ecosynthesis as ecological succession on a global scale. *Astrobiology*, 4(2):168–195, 2004.

[28] Christopher S Henry, Matthew DeJongh, Aaron A Best, Paul M Frybarger, Ben Linsay, and Rick L Sevens. High-throughput generation, optimization and analysis of genome-scale metabolic models. *Nature Biotechnology*, 28(9):977–982, August 2010.

[29] D. Jin, X. Qu, M.B. Cohen, and B. Robinson. Configurations everywhere: Implications for testing and debugging in practice. In *International Conference on*

Software Engineering (ICSE), Software Engineering in Practice Track (SEIP), pages 215–225, June 2014.

[30] L. J Kahl and D. Endy. A survey of enabling technologies in synthetic biology. *Journal of Biological Engineering*, 7(1):13, May 2013.

[31] The department of energy systems biology knowledgebase. `http://kbase.us`, 2016.

[32] Kyoto encyclopedia of genes and genomes. http://www.genome.jp/kegg/, 2016.

[33] Andy Kenner, Christian Kästner, Steffen Haase, and Thomas Leich. Typechef: Toward type checking #ifdef variability in c. In *Proceedings of the 2Nd International Workshop on Feature-Oriented Software Development*, FOSD '10, pages 25–32, 2010.

[34] Ahmad S Khalil and James J Collins. Synthetic biology: applications come of age. *Nature reviews. Genetics*, 11(5):367–379, 2010.

[35] Chang Hwan Peter Kim, Don S. Batory, and Sarfraz Khurshid. Reducing combinatorics in testing product lines. In *International Conference on Aspect-Oriented Software Development*, pages 57–68, 2011.

[36] Hiroaki Kitano. Computational systems biology. *Nature*, 420(6912):206–10, 2002.

[37] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering*, 30(6):418–421, 2004.

[38] Sung Kuk Lee, Howard Chou, Timothy S Ham, Taek Soon Lee, and Jay D Keasling. Metabolic engineering of microorganisms for biofuels production: from

bugs to synthetic biology to fuels. *Current Opinion in Biotechnology*, 19(6):556 – 563, 2008. Chemical biotechnology / Pharmaceutical biotechnology.

[39] Yu Lei, R. Kacker, D.R. Kuhn, V. Okun, and J. Lawrence. Ipog: A general strategy for t-way software testing. In *ECBS ’07. 14th Annual IEEE International Conference and Workshops on the*, pages 549–556, March 2007.

[40] Robyn R. Lutz, Jack H. Lutz, James I. Lathrop, Titus H. Klinge, Divita Mathur, Donald M. Stull, Taylor Bergquist, and Eric R. Henderson. Requirements analysis for a product family of DNA nanodevices. In *2012 20th IEEE International Requirements Engineering Conference (RE), Chicago, IL, USA, September 24-28, 2012*, pages 211–220, 2012.

[41] Atif Memon, Adam Porter, Cemal Yilmaz, Adithya Nagarajan, D Schmidt, and Balachandran Natarajan. Skoll: Distributed continuous quality assurance. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pages 459–468. IEEE, 2004.

[42] Diego Barcena Menendez, Vivek Raj Senthivel, and Mark Isalan. Sender-receiver systems and applying information theory for quantitative synthetic biology. *Curr Opin Biotechnol*, 31C:101–107, March 2015.

[43] M Million, E Angelakis, M Maraninchi, M Henry, R Giorgi, R Valero, B Vialettes, and D Raoult. Correlation between body mass index and gut concentrations of Lactobacillus reuteri, Bifidobacterium animalis, Methanobrevibacter smithii and Escherichia coli. *International journal of obesity (2005)*, 37(11):1460–6, nov 2013.

[44] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[45] Tadahiro Noguchi, Hironori Washizaki, Yoshiaki Fukazawa, Atsutoshi Sato, and Kenichiro Ota. History-based test case prioritization for black box testing using ant colony optimization. In *8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015*, pages 1–2, 2015.

[46] Stephen Payne and Lingchong You. Engineered cell-cell communication and its applications. *Adv Biochem Eng Biotechnol*, 146:97–121, 2014.

[47] Gilles Perrouin, Sebastian Oster, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves le Traon. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal*, 20(3-4), 2012.

[48] Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves le Traon. Automated and scalable t-wise test case generation strategies for software product lines. *IEEE Sixth International Conference on Software Testing, Verification and Validation*, 0:459–468, 2010.

[49] M Pierobon and I. F. Akyildiz. A physical end-to-end model for molecular communication in nanonetworks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 28(4):602–611, May 2010.

[50] Massimiliano Pierobon, Myra B. Cohen, Nicole Buan, and Christine Kelley. SCIM: Sampling, Characterization, Inference and Modeling of Biological Consortia. Technical Report TR-UNL-CSE-2015-0002, University of Nebraska-Lincoln, 2015.

[51] Xiao Qu, Myra B. Cohen, and Gregg Rothermel. Configuration-aware regression testing: An empirical study of sampling and prioritization. In *International Symposium on Software Testing and Analysis*, ISSTA, pages 75–86, 2008.

[52] Atri Sarkar, Jianmei Guo, Norbert Siegmund, Sven Apel, and Krzysztof Czarnecki. Cost-efficient sampling for performance prediction of configurable systems. In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 342–352, 2015.

[53] Itai Segall and Rachel Tzoref-Brill. Feedback-driven combinatorial test design and execution. In *Proceedings of the 8th ACM International Systems and Storage Conference, SYSTOR 2015, Haifa, Israel, May 26-28, 2015*, pages 12:1–12:6, 2015.

[54] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wąsowski, and Krzysztof Czarnecki. Variability model of the linux kernel. In *Fourth International Workshop on Variability Modeling of Software-intensive Systems (VaMoS 2010)*, Linz, Austria, 2010.

[55] Norbert Siegmund, Alexander Grebhahn, Christian Kästner, and Sven Apel. Performance-influence models for highly configurable systems. In *Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 284–294, New York, NY, 8 2015. ACM Press.

[56] Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting performance via automated feature-interaction detection. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 167–177, Piscataway, NJ, USA, 2012. IEEE Press.

[57] Ricard V. Solé, Raúl Montañez, and Salva Duran-Nebreda. Synthetic circuit designs for earth terraformation. *Biology Direct*, 10(37), 2015.

[58] Charles Song, Adam A. Porter, and Jeffrey S. Foster. iTree: Efficiently discovering high-coverage configurations using interaction trees. *IEEE Trans. Software Eng.*, 40(3):251–265, 2014.

[59] Justin L Sonnenburg. Microbiome engineering. *Nature*, 518(7540):S10–S10, 2015.

[60] Praveen Ranjan Srivastava. Optimization of software testing using genetic algorithm. In *Information Systems, Technology and Management - Third International Conference, ICISTM 2009, Ghaziabad, India, March 12-13, 2009. Proceedings*, pages 350–351, 2009.

[61] Gregory Stephanopoulos. Metabolic fluxes and metabolic engineering. *Metabolic Engineering*, 1(1):1 – 11, 1999.

[62] G. Tassey. The economic impacts of inadequate infrastructure for software testing, 2002.

[63] John R. Taylor. *An Introduction to Error Analysis The Study of Uncertainties in Physical Measurements*. University Science Books, 1997.

[64] Ines Thiele, Almut Heinken, and Ronan MT Fleming. A systems biology approach to studying the role of microbes in human health. *Current Opinion in Biotechnology*, 24(1):4 – 12, 2013. Analytical biotechnology.

[65] James M Tiedje, Robert K Colwell, Yaffa L Grossman, Robert E Hodson, Richard E Lenski, Richard N Mack, and Philip J Regal. The Planned Introduction of Genetically Engineered Organisms: Ecological Considerations and Recommendations. *Ecology*, 70(2):298–315, apr 1989.

[66] Engin Uzuncaova, Daniel Garcia, Sarfraz Khurshid, and Don Batory. Testing software product lines using incremental test generation. In *International Symposium on Software Reliability Engineering*, pages 249–258, 2008.

[67] Hooper L V., Littman DR, and Macpherson AJ. Interactions between the microbiota and the immune system. *Science*, 336(6086):1268?1273, 2012.

[68] Andreas Windisch, Stefan Wappler, and Joachim Wegener. Applying particle swarm optimization to software testing. In *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007*, pages 1121–1128, 2007.

[69] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining Practical Machine Learning Tools and Techniques*. Elsevier, Burlington, MA, 2011.

[70] C. Yilmaz, M. B. Cohen, and A. Porter. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering*, 31(1):20–34, Jan 2006.

[71] C. Yilmaz, E. Dumlu, M. B. Cohen, and A. A. Porter. Reducing masking effects in combinatorial interaction testing: A feedback driven adaptive approach. *IEEE Transactions on Software Engineering*, 40(1):43–66, 2014.

[72] Jizhong Zhou, Kai Xue, Jianping Xie, Ye Deng, Liyou Wu, Xiaoli Cheng, Shenfeng Fei, Shiping Deng, Zhili He, Joy D. Van Nostrand, and Yiqi Luo. Microbial mediation of carbon-cycle feedbacks to climate warming. *Nature Climate Change*, 2(2):106–110, 2011.