Neural Network Design via Evolutionary Programming

Discovering Relevant Inputs and Optimal Configuration

BY

JONATHAN P LESSNER
B.S., Northern Illinois University, 2004

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Sciences
in the Graduate College of the
University of Illinois at Chicago, 2014

Chicago, Illinois

Defense Committee:

Piotr Gmytrasiewicz, Computer Science, Chairman
Bing Lie, Computer Science
Yang Dai, BioEngineering

This thesis is dedicated to my wife, Sarah, who stayed by my side through my surgeries and encouraged me to transform my struggles into something of personal and public value.

ACKNOWLEDGMENTS

I would like to offer my heartfelt thanks first and foremost to the team of doctors on whom I relied for treatment and support during my long illness. My gastroenterologists, Ramamoorthy Sundaresan (Illinois Masonic Hospital), Terrence Barrett (Northwestern Medical Facility) and Pradeep Kumar (Austin Gastroenterology) helped immeasurably to control my disease with medical intervention and personal care, allowing me to live and persevere through even the most trying episodes of my Ulcerative Colitis. When medical intervention was no longer tenable, my expert colorectal surgeon, David Fleeger (Central Texas Colon & Rectal Surgeons) and his team cured me of my disease and guided me through recovery, to a healthful and satisfying life.

I would also like to express my appreciation to all of the engaging and entertaining teachers at UIC who made learning difficult concepts an adventure. In particular, I would like to thank Dr. Daniel Graupe for teaching me many of the core concepts in this paper, and encouraging me to expand on the theories and methods in its predecessor, a smaller project done for his Neural Networks class. I would also like to thank Dr. Piotr Gmytrasiewicz for teaching three of my favorite classes and guiding me though the thesis and defense process.

JPL

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	1
	A. Purpose	1
	1. Ulcerative Colitis	1
	2. Personalized Medicine	2
	B. Problem Space	4
	C. Solution Space	5
	D. Overview of Theory and Methods	6
	1. Use of Evolutionary Design to Derive Relevant Inputs	6
	2. Maximizing Input Relevance	10
	3. Optimizing Body Topology	12
II.	THEORY AND CONCEPTUAL BACKGROUND	13
	A. Back Propagation Neural Networks	13
	B. Evolutionary Programming	17
	C. Synthesis	19
III.	METHODS AND ALGORITHMS	20
	A. Data	20
	1. Collection	20
	a. Ulcerative Colitis Symptoms	20
	b. Medication	21
	c. Dietary Intake	22
	d. Exercise Regimen	22
	2. Preprocessing	22
	B. Back Propagation Neural Network	24
	1. Body Configuration	24
	2. Output Node Configuration	25
	3. Input Node Configuration	26
	4. Training and Prediction	29
	C. Evolutionary Programming	32
	1. Scoring Algorithm	33
	2. Initialization and Generation Propagation	33
	3. Crossbreeding Method	36
	4. Mutation Methods	37
	5. Ending the Experiment	38
	6. Miscellaneous Optimizations	38
	a. Generation Shuffling	38
	b. Crossbreeding/Mutation Substitution	39

TABLE OF CONTENTS (continued)

D. Output Processing…… 39
1. Optimizer……... 39
a. Diagnostic Setting…….. 40
b. Prescriptive Setting…… 41
2. Human Readability…….. 41
3. Statistics Graphs…….. 42

IV. RESULTS AND DISCUSSION……..... 43
A. BPNN and EP Performance……... 43
1. Training Threshold…….. 43
2. Average MSE……... 44
3. Average Training Passes……... 46
4. Average Inputs…… 48
5. Average Category Inputs…… 49
6. Average Hidden Layers and Nodes……... 50
7. Average Learning Rate and Momentum…….. 51
B. Discovered Relevant Inputs…… 52
C. Optimizer Program……... 54

V. CONCLUSIONS…….. 57
A. Primary Goal: Determining Relevant Factors via BPNN and EP…… 57
B. Secondary Goal: Determining Expressive and Suppressive Factors…….. 57
C. Tertiary Goal: Medical Treatment Plan…… 58

VI. FURTHER STUDY……..... 60
A. Better Optimizer Algorithm…….. 60
B. Eliminating False Positives…… 61
C. Better Data Classification…… 63

VII. APPENDICES…….. 64
A. BPNN and EP Performance Graphs……. 64
B. Input-Specific Optimizer Results…… 82
C. Collected Data Tables…… 92
D. Code……... 290

VIII. CITED LITERATURE……. 391

IX. VITA…….. 393

LIST OF FIGURES

FIGURE	PAGE
II.A.1 Basic BPNN Design……..	13
II.A.2 Sigmoid Filter Curve……	14
II.A.3 Output Error Formula……	14
II.A.4 Back Propagation Algorithm……	15
II.A.5 Link Weight Adjustment Formula……	16
III.B.1 BPNN Parameters……	24
III.B.2 Output Layer Configuration……	26
III.B.3 Input Node Parameters……...	27
III.C.1 Genome Subsections……..	32
III.C.2 Scoring Algorithm…….	33
III.C.3 Selection Method…….	34
III.C.4 Crossbreeding Method……	34
III.C.5 Mutation Method……..	35
III.C.6 Generation (n+1) Composition……..	35
III.C.7 Crossbreeding Permutations……..	37
IV.A.1 Training Threshold……...	43
IV.A.2 Average MSE……	44
IV.A.3 Average Training Passes……	46
IV.A.4 Average Input Nodes……..	48
IV.A.5 Category Input Nodes…….	49
IV.A.6 Average Hidden Layers and Nodes……..	50
IV.A.7 Average Learning Rate and Momentum……..	51
IV.B.1 Symptom Factor Frequencies……	52
IV.B.2 Non-Symptom Factor Frequencies……...	53
IV.C.1 Optimizer Key……...	54
IV.C.2 Optimization Example…….	55
VII.A.1-18 Experiment Run Performance Graphs……..	64
VII.B.1-8 Symptom Optimizer Assessments…….	82
VII.B.9-22 Non-symptom Optimizer Assessments ……	86

LIST OF TABLES

TABLE	PAGE
III.B.4 Experiment Settings…….	31
III.C.8 Mutation Types…….	37
III.D.1 Optimizer Input Permutations…….	40

SUMMARY

Neural Networks are a powerful tool for predicting quantifiable events based on sensory input or sampled data. However, while our ability to collect and organize this data is increasing exponentially, the task of selecting and curating relevant data among mountains of irrelevant noise quickly outpaces our ability to capture predictive value by even the most clever intentional design of input data streams. The non-rational assessment of correlational links that makes NNs so powerful as predictors often make efficient and accurate NN design and configuration challenging at best and confounding at worst. This is especially true in problems spaces where the available data is extensive in scope and diverse in detail.

For this reason, my thesis proposes to leverage another powerful non-rational approach to design: Genetic Programming. The goal is to use the methods of natural selection to breed a population of Back Propagation Neural Networks that are both accurate and efficient, while using selection pressure to find a minimal number of input nodes (which sample data from the provided set).

In the first generation, a population of BPNNs is created, each with a set of randomized input configurations (data category, data type, and sample window). Each of these BPNNs is then trained and tested in the typical fashion (i.e. no GA or EP to set the link weights, just regular Back Propagation). Then the selection pressure is applied by a scoring algorithm, designed to reward accuracy and efficiency. The top performers of each generation are retained into the next, and fill up the rest of the population slots for the next generation by the processes of crossbreeding, mutation, and randomization. Through this method of rewarding BPNNs for high accuracy (elicited by high-relevance input node configurations) and punishing them for the opposite (low-relevance input node configuration), it is hypothesized that the process will derive a population of BPNNs with inputs that represent only the most-relevant correlational data in the data set.

SUMMARY (continued)

This is especially important in problem spaces whose data sets are complex, massively multivariate, and often inscrutable. For this reason, I have chosen the domain of Personal Medicine, specifically the expression and suppression of Ulcerative Colitis in a single patient (myself). By exhaustively tracking medical treatment, dietary intake, exercise, and symptomatic expression over 6 months, I attempted to apply this method to gain insight into the complex biological systems at work, and how they might be influenced by different treatment regimen.

I. INTRODUCTION

A. **Purpose**

1. **Ulcerative Colitis**

Ulcerative Colitis is a chronic inflammatory bowel disease that is highly influenced by diet, exercise and various other factors beyond medical treatment. Individual dietary and environmental triggers are often confounding, preventing the study and treatment of some chronic diseases like Ulcerative Colitis from being amenable to controlled population studies, the cornerstone of most current medical investigation.

Ulcerative Colitis is recognized as having two major classifications. The aforementioned classification arises from a chronic and recurring infection by the bacterium *Clostridium difficile*[11]. Due to the highly personal effects of dietary intake, causative and suppressive factors are often very difficult to isolate and understand. During the course of treatment, it is often suggested by medical professionals that patients should undertake a controlled individual dietary assay.

A second form of UC arises from an immunological malfunction, wherein the patient's own antibodies mistake the tissue of the mucosal membrane of the colon for a foreign invader, and begin to attack it. This form is genetically influenced and is a heritable trait[5]. Since this form is not triggered by the presence of toxic bacteria in the colon, but rather by the presence of the patient's own natural tissue, the symptoms usually cannot be alleviated by the application of special diets. For these patients, the only reliable options for sustained remission are powerful anti-inflammatory and immunosuppressant medication. Failing that, the only known permanent cure for genetic UC is surgery, removing the entire colon and hence the tissue that triggers the immune malfunction.

Strong medications, such as Prednisone, Imuran, Humira, and Methotrexate can induce maintain remission of symptoms, but carry with them heavy side effects, such as diabetes,

osteoporosis, frequent infections, and compromised immune systems. While *c. diff* infections may be managed by careful dietary regimen, allowing those patients to retain their colons and return to relatively normal gastrointestinal function, the genetic form of the disease is often progressive, responsive only to ever-increasing doses of the harsh medications, and usually ends only when surgery is performed to remove the colon.

For this reason, it is important to know whether a change in diet might induce prolonged remission without drugs, or whether surgery is the only reliable option for continuing health. Because both forms of ulcerative colitis respond well to prescription medications, but only the infection-triggered form is responsive to dietary regulation, it is desirable for patients suffering from UC symptoms to know which form they are afflicted with, so a proper course of treatment can be planned.

2. **Personalized Medicine**

One current diagnostic approach for assessing the effects of various balances of dietary intake is the "Elimination Diet", in which a sufferer eliminates all possible food triggers, and after achieving remission re-adds them to their diet one at a time to determine which ingredients trigger a UC flare. This is often very difficult to perform and is sometimes completely ineffective, as the very restrictive initial diet suggestions sometimes themselves contain ingredients that have been found to cause distress, such as rice or poultry. Another diagnostic approach is the use of a food-specific histamine panel, in which the sufferer is tested for specific food allergies. This may identify dietary influences that are relevant to symptoms and general flare-ups. However, if the patient has not eaten a specific ingredient for a long period of time because of their own suspicions (such as nuts/seeds or lactose), the histamine for that irritant will not be detected by the panel, and a false negative for allergy/sensitivity to that ingredient will be determined by the test.

For these and other reasons, most current diagnostic methods for individuals suffering with UC are woefully ineffective and devilishly frustrating. A personalized, data-driven method of dietary diagnosis is needed. Statistical approaches of pattern recognition could be fruitfully applied to this problem, to determine experimentally and empirically whether the form of the disease from which the patient is suffering is reliably affected by dietary management.

The traditional form of medical assessment of treatment has been the “population study”, in which large groups of patients are randomly assigned to either a “treatment group”, which receives the experimental treatment (usually a medication), and a “control group”, which does not. Patients check back in with their supervising physicians, typically in regular intervals ranging from weeks to months to check overall response to treatment. Differences in response to treatment and control conditions are then assessed and medical efficacy is based on the average difference of change in disease expression between treatment and control groups.

As mentioned above, because the treatment in the case of dietary restriction methods is so variable from patient to patient, based on individual sensitivities to specific foods, a general population study for UC reaction to something like lactose or gluten might produce highly variable or conflicting results. For this reason, a population study of dietary effect on UC would likely be inconclusive.

Another reason that dietary assays are not amenable to population studies is that the reaction time to dietary intake is often on the order of hours to days, while the check-in intervals with managing physicians is much longer. It would be very difficult for a patient to correctly link every flare-up or abatement of their symptoms to a specific food without exhaustive tracking of personal data, as well as a powerful means of interpreting statistical correlation between individual dietary ingredients and subsequent symptom expression. Management of long-term symptom expression and statistically noisy dietary trends would do very little to illuminate how large a role diet plays in flare-ups, and hence is not much help in determining whether the patient is suffering from infectious UC or from genetic UC.

B. **Problem Space**

Ulcerative Colitis is an incredibly difficult disease to predict and control. Its many variables and chaotically interacting biological systems make accurate prediction beyond a matter of hours difficult and prediction beyond a few days nearly impossible. Constructing algorithms from data as complex as prescription dosage, dietary intake, and exercise regimen can be a daunting if not insurmountable task. Mathematicians, statisticians, doctors, and patients are simply not up to the task of rationalizing and predicting this massively multivariate and highly personal system of interactions[10].

This is not a problem of shortage of data or processing resources. The problem is the inability of analytical minds to take nonlinear data sets and craft rational equations which are capable of describing biological systems and disease pathways in a comprehensible way.

In an age where resources such as personal bioinformatics trackers are becoming ever more popular and user-friendly, and computational resources double in performance per dollar every twelve to eighteen months, eliminating the bottleneck of human interpretation from our predictive tools is not just useful for the fact that it frees up intellectual resources and halts the endless confusion and frustration. It can also produce much better prediction and knowledge discovery tools in areas where we simply do not have the capacity to create sufficiently accurate rational descriptions of the systems and diseases which we are observing.

Statistical and correlative predictive systems learn the relationships between associated events by repeated exposure to massive amounts of sensory input. Like a specialized organ which needs no conscious or rational understanding of the system, they can react with the correct prediction by trained reflex.

Yet the very strength of these neural networks (their ability to draw associations between events in a non-rational way) can also be the biggest obstacle to optimal implementation. Often, the designers themselves are unaware of which design parameters have the most influence over the performance of their networks. Optimization is little more than guesswork and lucky breaks. Much man hour-intensive development time is wasted on finding the best parameters for a network, in the same way that this man hour-intensive time was wasted on distilling mathematical equations and algorithms before the shift to using neural networks.

C. **Solution Space**

Because much of the effort in optimizing the performance of a network is simple randomization and tinkering, I believe that another powerful non-rational method can be used to replace the tedious and often frustrating hands-on work. This method is referred to as Evolutionary Programming.

The theory of Evolutionary Programming is based on and approach which mirrors Darwin's Theory of Evolution by Natural Selection. In this case, rather than organisms forced to compete in a natural environment, researchers model solutions to their problem and rank them based on performance defined by an appropriate scoring algorithm. Top performers survive, crossbreed, and mutate for hundreds or thousands of generations, until a stable population of optimal solutions is reached.

Like natural selection, this is a non-rational approach that requires time and pressure. However, it does not require human intervention or rational understanding to produce accurate, powerful, and efficient solutions. Once again, we can remove the rational designer from the design process, and rely on raw data and computation to distil the solutions that our rational minds cannot intentionally construct.

Once a sufficiently accurate and efficient pool of solutions is created, engineers can then create programs that leverage these solutions to discover most-relevant factors, and create diagnostic and prescriptive reports. These reports are human-readable, and distil the most important insights about the solution space. They may also offer suggestions on how to better influence the system to achieve desired results. Distilling these fundamental insights about complex systems is the goal of all rational thought. With the power of Machine Learning and Evolutionary Programming, engineers and scientists can discover knowledge beyond what has previously been imaginable by rational deduction and design.

D. **Overview of Theory and Methods**

1. **Use of Evolutionary Design to Derive Relevant Inputs**

Implicit in the manual design of Back Propagation Neural Networks are two basic assumptions about input nodes. The first is that some data among the data is more valuable and some is less valuable in regards to eliciting predictive accuracy from the output nodes. Much effort is expended in data collection and curation as well as input node configuration in order to produce accurate prediction output. The second assumption is that processing inputs and propagating signals through the network takes a non-trivial amount of time. A smaller, faster accurate network is generally assumed to be better than a large, slow network, given approximately equivalent predictive accuracy. Much effort is expended to produce network topology that results in acceptable accuracy with minimal training time.

Since this design work is largely reliant on the personal experience and expertise of individual engineers, whose relative rarity in the information industry can be seen as a bottleneck to the facilitation of more accurate and efficient predictive tools available to the public, it can be seen as desirable to create an automatic method to construct these prediction tools.

This project in particular has a primary and a secondary goal. The primary goal is to use the process of Evolutionary Programming to establish a method for creating BPNNs optimized for massively multidimensional data sets. The EP guides the evolution of a population of BPNNs over a preselected number of generations (1000) through the use of processes modeled after observed biological selection, crossbreeding, and mutation. The result of these processes is a set of BPNNs with a small set of inputs (5-20) that have been bred to represent only the most relevant and valuable sample windows, as well as body topology optimized for accuracy and training speed.

For the purposes of encoding BPNN configurations (including input selection and body topology), a Genome data structure was created. This Genome is split into two arbitrary halves, the first half (A) recording the configuration of all input nodes (see Section III.B.3), and the second half (B) recording the configuration of the body topology (see Section III.B.1). The Genome was split into two parts, as opposed to three or more parts, to simplify the crossbreeding process and to limit combinatorial multiplication of offspring to a reasonable number (in this case 5^2 rather than 5^3 or more).

For the first generation, a population of Genomes (50) was created. This represents "spontaneous generation", which doesn't happen in nature, but was necessary and practical for the simulation. For each Genome, a BPNN was constructed according to the configuration data contained in the Genome. The BPNNs were trained and tested in the standard method. It's important to note here that nothing regarding Evolutionary Programming or Genetic Algorithms was utilized during the supervised learning of each individual BPNN. Previous work has been done, using the theory of Genetic Algorithms to adjust the link-weights in individual NNs as an alternative learning method to back propagation[16], but these methods were *not* used in this project.

After training an individual BPNN, it was scored in regards to its training speed, efficiency, and prediction accuracy (see Section III.C.2). Each Genome in the first generation was trained, tested, and scored in this manner, then ranked by these scores.

From these ranked Genomes, generation 2 was created. Because each generation contained 50 members, it was important to employ a balanced mix of methods when selecting and creating new members of proceeding generations. The first method used was that of "survival of the fittest". A small portion the highest ranking members of the parent generation (top 5) was retained and carried to the next generation, while the rest (bottom 45) were discarded. This fulfills the basic premise of the Evolutionary Programming theory; that the most optimal members of a generation propagate more prolifically then the less optimal members. In this case, the difference in propagation was binary: survive and propagate, or die and become extinct.

The second method used was "crossbreeding". This method works to allow similar, but only marginally optimal Genomes the chance to stumble upon configurations that may perform more optimally in a local area of the solution space. As mentioned earlier, each Genome was divided into A/B sections, and for this method, a set of new members in the child generation was created by crossbreeding each of the top 5 retained Genomes with each of the other retained Genomes, thus producing 20 new Genomes comprised of swapped combinations of crossed A and B sections.

The third method used was "mutation". This method allows Genomes that are marginally optimal a chance to stumble upon an incremental improvement in their input or body configuration that allows them to perform better among the Genomes of the next generation. This is roughly equivalent in theory to other stochastic methods of "hill climbing". Each of the 5 retained Genomes from the parent generation produced 5 offspring of its own via the mutation method, creating a total of 25 more Genomes for the child generation.

The fourth method used was “spontaneous generation” once again. One major problem regarding stochastic methods of optimization is tendency for solutions to get stuck in a “local maximum”. While spontaneous generation isn’t strictly allowed in engineering methods that employ bio-mimicry, its usage can go a long way to avoid the problem of local maxima. The final 5 Genomes of the child generation were created completely at random, by the methods and parameters used to generate all of the Genomes in the first generation.

For every generation (n), generation (n+1) was created using training, scoring, ranking and the four evolutionary methods above. By this process, optimized Genomes and their associated BPNNs were created over the course of 1000 generations.

The secondary goal of this project is to use the assumptions about input nodes and body topology to leverage the Evolutionary Program into discovering the most valuable and efficient input configurations, hence illuminating the most relevant factors in the data in regards to the predictive accuracy of the output nodes.

Since the selection pressure of the Evolutionary Program was defined as an algorithm that rewards predictive accuracy and training efficiency, and the processes of selection and evolution defined above were successful, the resultant input configurations then represent the most relevant predictive factors among the input data, in regards to predictive accuracy.

By this method, applied to the exhaustive data pertaining to one patient’s diet, medication, exercise and symptoms collected over 6 months, we can tell the relative impact of each of these inputs on the expression of UC symptoms. If the numbers and consistency of inputs regarding specific dietary ingredients are large, then it can be deduced that those ingredients have a large positive or negative effect on disease expression, and that their increased use or elimination might be used as a treatment. If the numbers and consistency of inputs regarding specific dietary ingredients are small, while those regarding medication are

large, then it can be deduced that the influence of dietary intake is small, and that treatments such as surgery must be pursued in lieu of less invasive treatments like dietary management.

Any number of engineering projects can be found which demonstrate the concept of Evolutionary Programming, from a stick figure that evolves a method for walking[9], to a 2D cart racer that evolves effective geometries to traverse hilly terrain[2].

2. **Maximizing Input Relevance**

Earlier work concerning the automated optimization of relevant inputs can be found in the work of Dr. Graupe in his work on the LAMSTAR network[7]. This network is constructed as a discrete classifier, utilizing Self Organizing Maps in combination with randomized input configurations that are added and removed during training, based on performance and relevance measures (denoted by strength of each input node's link weights).

A basic application of the LAMSTAR network may be that of character recognition. An input for this network may be all the pixels in a row or column of a bitmap that represents the given character, or perhaps even a random subset of pixels from the bitmap. Several of these inputs are created using randomized values within defined parameters, and are attached to the SOM portion of the network, defined by a Kohonen layer (which is a winner-takes-all layer that acts as the classifier) and a Grossberg layer (which converts a single K-node's output into the desired output, usually a coded binary configuration of activated nodes).

However, a classifier is simply not the correct application for the prediction of symptomatic expression when those symptoms are multifaceted and can be expressed in a spectrum of severity. It was for this reason that the selection-pressure approach was applied to BPNNs, which are more nuanced networks, capable of multifactorial outputs which can express activation with continuous precision (any value between 0.0 and 1.0).

The random initialization, grading and addition/removal of inputs are the parts that are relevant to this paper. The introduction to the LAMSTAR chapter in Principles of Artificial Neural Networks 2nd Edition notes the need for applications which are "suited to deal with analytical and non-analytical problems where data are of many vastly different categories and where some categories may be missing, where data are both exact and fuzzy and where vastness of data requires very fast algorithms" [15, p 249]. I believe this statement and the pursuant LAMSTAR theory correctly identifies randomization of inputs and grading of performance as key methods to designing efficient and accurate networks. These methods are meant to be a replacement for intentional design, which in a massively multivariate data environment will ultimately perform less well because intentional design is based on engineer bias and possible misconceptions about which factors are most relevant.

In contrast to the LAMSTAR's real-time assessment and alteration of inputs during training/testing, this project uses Genomes to encode the configuration of inputs and body topology of individual BPNNs and then scores performance for each holistically. The reason for this difference is the fundamental difference between input nodes in the LAMSTAR and the input nodes in medical data sampling. Each input for the LAMSTAR consist of a combination of small binary input points that conceptually might represent an interaction between different parts of the data. Each input for this project, in contrast represents a continuous sample window for a single type and category of diet, exercise, medication, or previous symptom value. Since complex interactions can occur between disparate input types, it would not make sense for inputs to be added, removed, or mutated in real-time during a BPNN's training and assessment. On top of this factor, the variable initialization between different BPNN's body topology may imply that a network might be more or less capable of learning sophisticated interactions among inputs and encoding them in its link weights, making rapid assessment of individual input nodes both difficult and problematic.

3. **Optimizing Body Topology**

Earlier work concerning the optimization of body topology to elicit networks which demonstrate efficiency and accuracy in training can be found in the work of Philipp Kohen[13]. Different configurations of hidden layers and hidden nodes were assessed for their speed in training and accuracy in encoding the XOR problem. Though it has been previously proven that the XOR problem can be solved with a BPNN containing only one hidden layer containing only two hidden nodes[3], Kohen demonstrates that other non-obvious configurations can achieve accuracy faster with a non-minimal configuration.

While attempting to optimize network body topology is not strictly necessary to finding the most relevant inputs among the data, it is very helpful in minimizing the total running time of the 1000 generations without sacrificing accuracy. Because the scoring algorithm contains a factor for network size and training efficiency (see Fig. III.C.2), selection pressure over the generations will trend toward minimizing network complexity. However, if during this period there is a subtle interaction between multiple inputs that achieves higher accuracy by spending more processing time on a slightly larger network body, the better score based on higher accuracy will capture this value, counterbalancing the extra processing cost, and the network will be retained.

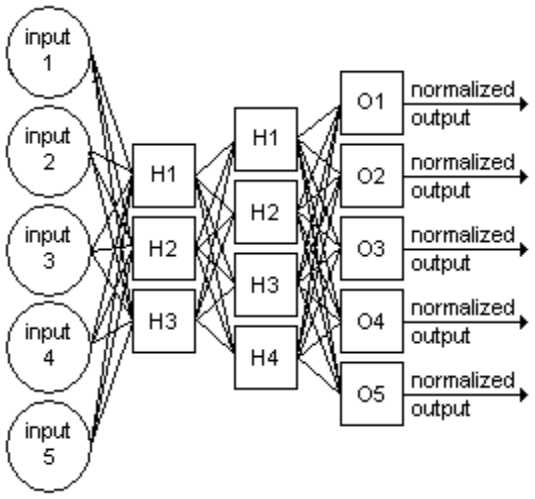
In this way, processing time is minimized over the computation-intensive process of Evolutionary Programming without sacrificing accuracy and relevance of the final results.

II. THEORY AND CONCEPTUAL BACKGROUND

A. **Back Propagation Neural Networks**

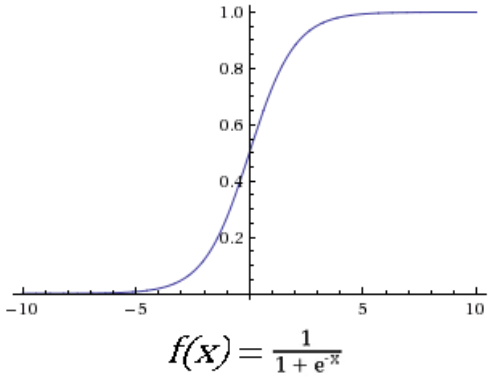
The basic design of a BPNN is meant to represent the operating structure of a human brain (though drastically simplified). There are a number of sensory inputs ("input" 1 through n) which excite a first hidden layer of simulated neurons. Every input is connected to each neuron in the first hidden layer by an activation path, which has an associate link weight, representing the strength of the connection. These inputs are multiplied by a link weight (initially random) to their respective neurons. After the input layer of neurons, there can be any number of hidden layers (H) of neurons between the input layer and the output layer. These layers' neurons are linked only to the layers directly preceding and following (there are no connections that skip a layer). The neurons of a layer are activated by those preceding neurons in the same way that the first layer was activated by the input (modified by link weights). These activation signals are then summed with each neuron, and the neuron applies the same activation function, and so forth, propagating the signal all the way to the output layer[14][17][19].

Fig. II.A.1: Basic BPNN Design



Throughout this process, each node in each layer receives signals propagated from the activation of all nodes in the preceding layer. Within each neuron, the signals (modified by their respective link weights) are summed. This summed value is then applied to the activation function inside the neuron. This function must be linear, because the back-propagation of the error utilizes the derivative. The most popular linear activation function is the sigmoid function[8]. This function transforms activation values from negative infinity to positive infinity to values between 0 and +1 in a way that accentuates differences in original values close to 0:

Fig. II.A.2: Sigmoid Filter Curve



During the training process, a network is presented with a data set consisting of inputs with associated known ‘correct’ output values. At the output layer, the error for each output neuron is determined by calculating the difference between the actual output achieved by forward propagation of the signal through the network, and the correct target output:

Fig. II.A.3: Output Error Formula

$$E = A_{target} - A_{observed}$$

To properly adjust the link weights of the network during training, the error must be calculated for each non-input neuron. The error values derived at the output layer are then

recursively propagated backward through the entire network. This is done for each neuron in each layer (moving backward through the net) by summing the errors (multiplied by the weights) of each neuron it is connected to on the following layer of the network, then multiplying that sum by the derivative of the activation function (with the activation value of that same neuron applied to this derivative):

Fig. II.A.4: Back Propagation Algorithm

$$E_{ij} = (A_{ij}(1\text{-}A_{ij})) * \sum_{k=1}^{n} (W_{ijk} * E_{(i+1)k})$$

Once this error is propagated throughout the entire network, it is used to adjust the weights. The value of the weight adjustment for each link weight from a single neuron (A) to a neuron in the proceeding layer (B) is determined by multiplying the error value at the downstream neuron (B) by the activation value of the current node (A) it is linked to by this link weight, and some arbitrarily assigned learning rate value.

The learning rate (eta) is a scalar value which ensures that the weight adjustments implemented on a single training pass are a small fraction of the calculated error, so as to avoid over-fitting the network's link weights to recently seen data. There is also another variable associated with weight adjustments, known as momentum. The learning momentum (mu) is an additional weight adjustment for each training pass which adds some fraction of the adjustment from the previous training pass to the current adjustment for each node. This helps smooth and accelerate learning when a network's inputs are not ordered in an optimal way[4]. The resultant equation that describes the weight adjustment for one node during a single training pass is:

Fig. II.A.5: Link Weight Adjustment Formula

$$\Delta W_{ijk} = (\eta * A_{ij} * E_{(i+1)k}) + (\mu * \Delta W_{ijk(t-1)})$$

This network is trained iteratively over large datasets. The longer the training runs the more accurate and more sensitive to subtle input differences the network becomes.

The Back Propagation Neural Network is considered one of the slowest of the designs in the field of statistical pattern recognition. Its many neural connections, repetitive multiplication, and summing of activations and weights makes processing time sluggish compared to other neural network designs. Adjustments of weights is also a time-consuming affair, and if the network is too large, the cost of the time spent finding the correct weights to achieve acceptable performance can outweigh the value obtained from a statistical correlative approach to prediction.

However, Back Propagation Networks are often adept at learning and recording subtle correlations between data. If these networks could be designed in an optimal way, much of the drawback of high training time due to overly complex implementation could be avoided. Often, design of a BPNN is a matter of educated guesswork. An engineer familiar with the implementation of these networks and with a good grasp of the complexity of the solution space can make networks that achieve desired accuracy. However, they may be rife with inefficiencies.

Excessive numbers of layers may sometimes unnecessarily increase training time without effectively increasing sensitivity to subtle differences in input data. Bottlenecks created by a layer with a small number of nodes may decrease the accuracy of the network instead of

effectively reducing data complexity. Many of the inputs may be irrelevant to the expected output of the system, uselessly increasing both processing time and the task of data entry[7].

Back Propagating Neural Networks can be an incredibly powerful statistical prediction tool. But their status as a “black box” solution has kept many engineers at bay, not wishing to get themselves into an unending cycle of random tweaking and guesswork.

Yet the very problem of having an unintelligible and complex solution space invokes a solution. If the problem of complex correlative data analysis can be solved using automated statistical methods it seems apparent that the problem of complex (but well bounded) design could be solved by statistical methods as well.

B. **Evolutionary Programming**

One such automation-oriented approach to design is known as Evolutionary Programming. This method borrows heavily from Darwin’s theory of evolution and natural selection. Generally, the theory of Evolutionary Programming seeks to find an optimal solution to a problem by creating a randomized population of potential solutions, applying selection pressure (via a scoring function) to make them “compete” for survival, iterating this process for an arbitrary number of generations until an optimal or satisfactory solution is distilled from the population[9].

Further more subtle and powerful methods of squeezing efficiency out of this theory also mirror biological evolution. A solution’s parameters can be arbitrarily encoded into “genome”, and these genomes can be used to create in the next generation solutions with combinations of

the attributes of the top-performing solutions of the current generation. Perhaps a lucky combination between two top performers will shed the part of the genomes of each which hold some unknown weakness, and will retain the portions of their genomes which hold some known strength.

Another method for searching the solution space is mutation. At every generational step, the top performers in the current generation produce offspring, each with a minor mutation. In this way, the algorithm more efficiently searches the regions of the solution space that are known to hold at least some productive answers.

Evolutionary Programming requires much processing time, and parameters of randomization and mutation must be set by an engineer that is familiar with both the domain of the problem, and with the boundaries of the solution space. However, once these are established, the program can be left to run while the engineer is freed up to tackle tasks that are more amenable to direct human design. They can check back at regular intervals and make adjustments to the randomization scheme or to the scoring algorithm, and trust that the algorithm is doing the most tedious part of the design work; all while producing better results than a team of designers could.

Solution spaces with complex contours and parameters that often interact in counterintuitive (non-common-sense) ways can frustrate even the most insightful designers and engineers. Leaving it to randomization (to find new, unintuitive solutions) and crossbreeding/mutation (to find optimal versions of known solutions) can both produce better solutions than we have today *and* free up our great *human* neural networks to work on more pressing and intellectually stimulating problems.

C. **Synthesis**

This project was designed to be an experiment in constructing an Evolutionary Program that would optimize a Back Propagating Neural Network, with respect to the competing factors of network performance (measured by MSE of output values) and network simplicity (measured by a combination of inter-neuronal connections and training efficiency).

This restriction on network complexity and penalization for low accuracy distils in minimal time a solution BPNN that utilizes a small number of the most relevant inputs while minimizing training time. The focus on minimal inputs also serves to make the final input configurations amenable to human understanding.

As an addendum to this project, a short program was created to interpret the network in terms that are human-readable, as well as determining which factors in the problem space reliably elicit an arbitrary targeted output (in this case, symptom suppression). This information conveys pertinent insight on how the real world system can be influenced through intentional action.

III. METHODS AND ALGORITHMS

A. **Data**

1. **Collection**

Personalized medicine relies heavily on extensive collection of individual, quantifiable metrics regarding a single patient's physical experiences, including medical treatment, dietary intake, exercise regimen, genetic profile, bioinformatics data, symptomatic expression, and many other clinically and observationally traceable factors.

Multivariate fluctuation of these factors was crucial to the success of the project. Identifying a primary cause/treatment is only possible if its application during the observation period is varied. Constant inputs to a system will show no relevance, even if their causal influence is profound. The only way to determine the relative potency of effects of inputs to the system is to observe changes in symptom expression that correlate with changes in medication, diet, and exercise.

For this experiment, data was collected by and about a single subject (the author) for a six month period, beginning at 12am, January 1st and ending at 12am, July 1st, 2012. Quantitative data was recorded concerning the symptomatic expression of the subject's Ulcerative Colitis, and of three specific categories which commonly influence the severity of these symptoms over time. All of these data were recorded in a Microsoft Excel file.

a. **Ulcerative Colitis Symptoms**

Symptoms were separated into eight distinct categories: bowel movements, blood loss, mucus in stool, pus in stool, flatulence, urgency, abdominal cramping, and diarrhea. At each episode during which any symptom was experienced, all symptoms were rated on a scale from 0-10 and recorded, along with the date and time of the episode. This scale is

subjective, where 0 is equivalent to no symptomatic expression in the given category and 10 is equivalent to the highest expression of the given symptom that has ever been experienced. For the term of this experiment, all symptoms were compared to the worst experienced during an 18 month, persistent UC flare up that was experienced several years prior to the present term, and was worse in all respects at one time or another. Therefore, there were no complications in data collection for which the maximum subjective rating of 10 increased in objective severity, necessitating a recalibration of any previously collected symptom ratings during the term of the experiment.

b. **Medication**

Medications were tracked by the trademark name, generic name, or by the name of their active ingredient. Dosages were recorded in milligrams (mg), and associated date and time of each dosage.

All prescription medications were taken under the guidance and consultation of the subject's Primary Care Physician (TODO: credit Spencer) and Gastroenterologist (TODO: credit Kumar). All prescription dosages were regulated based on medical need, and never adjusted exclusively for experimental reasons. Though these dosages changed over time, they were regulated based on the advice of the physicians, and fell within the standard methods of primary and secondary lines of medical intervention.

All over-the-counter medications were taken as needed for symptoms extraneous to UC, and were never taken chronically. One over-the-counter medication was taken with respect to UC treatment. The application of transdermal nicotine patches was based on prior, external research[15], and was taken under the supervision of the patient's gastroenterologist. The patches are a slow release system, and the dosage was recorded as one single event at the time of application.

c. **Dietary Intake**

During the term of the experiment, dietary intake of individual ingredients was recorded in complete detail. All efforts were taken to note all quantities of every edible substance consumed by the patient.

Most meals consumed were prepared by the subject, and therefore exact names and quantities of much of the dietary intake could be recorded exactly. Ingredient quantities were recorded in units appropriate to the given ingredient (oz, tsp, tbsp, cup) based on common usage. When prepackaged food or beverages were consumed, all primary ingredients were recorded, as well as an estimation of the quantities. When this was not possible, such as when the subject ate at a restaurant, estimations were based on menu descriptions and common cooking practices.

d. **Exercise Regimen**

Exercises undertaken by the subject were classified under three categories: strength, cardio, and yoga. The extent of exercise performed by the patient was recorded as a combination of exertion level and duration. Exertion level was ranked on a scale of 1-3, with 1 representing light and 3 representing heavy exertion. Duration of cardio and yoga were recorded in terms of minutes of continuous exertion during one routine. Duration of strength routines were recorded as the total number of repetitions in a session.

2. **Preprocessing**

For each type of event recorded in the Excel file (diet, exercise, medication, symptom), and for each category of that type of event (e.g. exercise: yoga), a time series vector was created and saved as a binary file, with a name composed of its type label (e.g. EXER),

concatenated with a number enumerated in a way that associates it with an entry in a text list of each category (e.g. yoga = 2), and designated as a vector file (i.e. ".vec"). So, for example, the time series vector used to record yoga events is named "EXER2.vec".

The category name list for each type of input was saved in a file composed of the type name (e.g. EXER), concatenated with the category list label (i.e. _CAT), and saved as a text file. So, for example, the list of exercise categories is saved in a text file name "EXER_CAT.txt".

For the purposes of checking for misspellings of dietary ingredients in the master Excel file, the category list for diet was alphabetized before output, so as to facilitate a manual inspection for cases where the same ingredient was spelled differently in disparate entries, because of a typo or data entry under multiple similar descriptions. This sorting was not necessary for medications or exercise types because of the brevity of those lists, and it was not necessary for symptom data because the order was hard coded due to considerations of the neural networks' output configuration.

Later processing considerations for inputs to the neural network necessitated converting each type of input to standard units. Dietary inputs were converted to a standard unit of ounces (oz) for both liquid and solid ingredients. Exercise sessions were converted to an arbitrary exertion unit, calculated by multiplying intensity by duration. Medications were already recorded using the standard unit milligrams (mg). Symptoms were also already recorded in the standard unit used for ranking of symptom expressions.

Though the original time data for any event was recorded including date, hour, and minute, the smallest relevant unit of time for the purposes of this experiment was an hour. For this reason, and to reduced data storage space, all recorded times had the minutes truncated from the data stream when stored in the time series vector files.

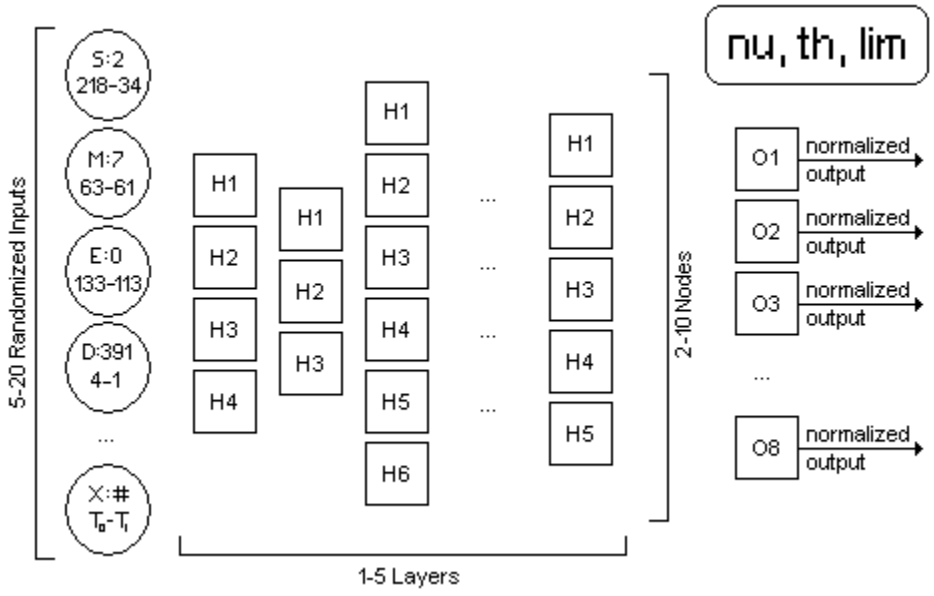
B. **Back Propagation Neural Network**

The goal of this experiment was to attempt to determine whether Evolutionary Programming methods could be fruitfully applied to a standard non-linear Machine Learning theory in order to improve its performance over that which could be implemented by intentional design and manual configuration. To that end, BPNNs were configured with body and input parameters constrained by standard practice. Because of the evolutionary methods applied, the population of BPNNs can only be described by these constraints, and not by explicit numbers.

1. **Body Configuration**

The body of a neural network consists of a number of hidden layers, each with a number of nodes, each of which is linked forward to every node of the following hidden layer. These hidden layers are situated between the input layer and the output layer, and serve to propagate and filter the input signal to produce output activations which represent a prediction.

Fig III.B.1: BPNN Parameters



Though it has been demonstrated that the function for identifying XOR input can be encoded using a minimized configuration of one hidden layer containing two nodes[7][19], other experiments have shown that accuracy can be achieved in fewer epochs with other configurations[13]. For this reason, the body topology of was one of the mutable parameters of the experiment.

For this experiment the numbers of hidden layers of a network were between a minimum of 1 layer a maximum of 5 layers. Each layer had a randomized number of nodes, ranging from a minimum of 2 nodes to a maximum of 10 nodes.

Along with the topology of the body layers and nodes, the learning rate and learning momentum were considered part of the body configuration. The learning rate was constrained to a minimum of 0.0 (no learning) and a maximum of 0.5 (1/2 of the error value). The momentum was constrained to a minimum of 0.0 (no momentum) and a maximum of 1.0 (perfect echo of previous weight change).

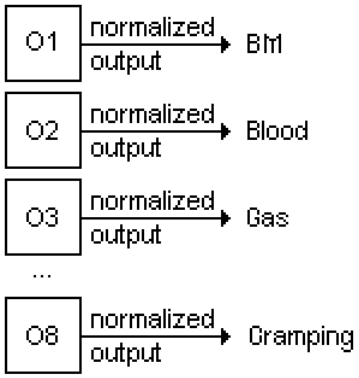
2. **Output Node Configurations**

The purpose of this experiment was to produce a neural network which was both efficient and accurate in predicting the expression of a broad set of disease symptoms. To effectively gauge the accuracy of a network, the prediction nodes needed to be consistent across every instance of the BPNNs. Without this consistency, attempts to measure accuracy of one network against any other network would be irrelevant and ultimately unhelpful. For example, a network with only one predictive node which represents one symptom may well achieve very high accuracy for that one symptom, but the utility of this network would be far

lower than a network that predicted all symptoms with marginally less accuracy. For this reason a consistent configuration of output nodes was chosen.

All BPNNs created in this experiment contained eight discrete prediction nodes. Each node represented the expression value of one of the individual symptom associated with Ulcerative Colitis listed above. For programming ease, each network's prediction nodes represented the symptoms in a consistent order. This order can be seen in the Thesis project code file Includes.h (see Appendix X, section Y).

Fig. III.B.2: Output Layer Configuration



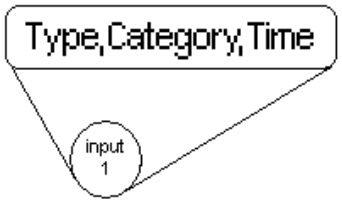
Predictive data steps were divided into 3-hour sections, eight sections per day. Each prediction node, before training, would open its associated data vector file, and compile that data into an interpreted time series vector by summing values of all events in a 3-hour time slice preceding each integer timestamp. Once all time slice values had been calculated, they were normalized by dividing each slice by the largest value from all slices (see OutputNode constructor, Appendix X, section Y).

3. **Input Node Configurations**

The purpose of this experiment was to produce a neural network which was both efficient and accurate in predicting the expression of a broad set of disease symptoms. To

effectively create a network that was efficient while maintaining accuracy, the input nodes needed to represent discrete categories of relevant data, and these categories needed to be experimentally tested for relevance in regard to the problem space. To implement this randomized, competitive search method, flexible input nodes were needed.

Fig. III.B.3: Input Node Parameters



Each input node in this experiment has four relevant parameters. The first two parameters (input_type, input_idx) encode precisely which type and category of data that the input node tracked, and which time series vector it pulled data from. For example, an input node with the settings input_type = 3, input_idx = 4 would track data representing dietary intake of apples.

The second two parameters (front_time, back_time) represent the lead time that a prediction node would use as an offset. This lead time represents the window of time in the associated time series vector that the input node would search prior to a requested prediction event. For example, if the node was tracking medical dosages of Humira (a slow-working, long-lasting medication), a relevant window of time to search for a dosage event prior to the prediction event may be any time between 280 and 60 hours before the prediction. However, if the node was tracking medical dosages of Prednisone (a powerful, but relatively short-lasting medication), the relevant lead time may be closer to 48 - 12 hours preceding the prediction.

The interpreted input vector for each of these input nodes was created by a method similar to what was described above for the output nodes. A time series was created from the

input data in discrete multi-hour chunks by summing the relevant data window, and normalizing based on the highest value of summed equal-size time windows during the entire data set. The main difference between the two methods is that, while output prediction data vectors summed values from the 3 hours preceding each timestamp, an input vector's lead time parameters determined which data was summed for each timestamp. For example, if data were being summed for the vector entry representing 3pm-6pm (timestamp 6pm), and the input node's lead time parameters represented a window between 8 and 2 hours preceding the timestamp, then the input node would sum values from the associated vector file for any events between 10am and 4pm. This sum would be the correct compiled value for the interpreted time series vector for the data time stamped 6pm.

For all non-symptom types of inputs, minimum lead time was 0 hours, and maximum lead time was 336 hours (2 weeks). The lead time window for an input was defined as some continuous portion of time within these limits, beginning at front_time, and ending with back_time.

Special consideration was taken for input nodes which tracked symptoms. It was deemed necessary to have recent symptoms be included as predictive factors, because the overall expression of UC symptoms changes slowly over time, punctuated with acute reactions to treatment or irritation. However, if inputs that represented the most recent 3 hours of a symptom were allowed, a completely accurate but entirely useless network could evolve that simply had 8 inputs (one for each symptom, focusing on the past 3 hours). This network's output could perfectly echo its input, and achieve perfect prediction. But this result would be completely irrelevant to the experiment.

For this reason, any input representing a symptom expression was not allowed to have a lead time within the 3-hour time slice it was predicting. Recent symptoms remain very relevant

for predictions, but being allowed to echo the exact symptom data that was just input must be avoided if the results are to remain useful.

The number of inputs for a single BPNN was constrained to a minimum of 5 input nodes, and a maximum of 20 input nodes. This range appeared to be appropriate, as most final-generation BPNNs contained a number of inputs in the middle of the range.

Because of special considerations concerning time-of-day both for inputs (diet, medication, exercise) and for outputs (symptom expression), special input nodes to encode for the time-of-day were added to every BPNN, regardless of genome. Since the prediction intervals were 3 hours each (8 chunks per day), time-of-day inputs were configured to encode for a 3-bit binary code representing the chunk number. That is, there were 3 input nodes that could each be either full activated (1.0) or fully deactivated (0.0). In this way regular daily cycle effects unrelated to treatment and symptoms could be controlled for by the networks.

Finally, because it would be a tedious task for a user to interpret the input note parameters for human-usable results after the experiment had produced optimized BPNNs, a program was created to compile and present this information in a coherent manner (see III.D).

4. **Training and Prediction**

This experiment focused on determining an accurate and efficient combination of inputs and network body configurations. These were the only parameters that were varied in any experimentally relevant way.

Input and output nodes, prior to training epochs, normalized their input in a way that took into account their relevant data windows and lead times, as described above.

Training of each individual network was performed in the standard method associated with the BPNN theory stated above. No special optimizations or methods were used in a way that would bias the performance of one configuration of a network over other potential configurations. Each network was trained on a data set consisting of a randomized selection of a portion of the available data time slices, and was tested for average Mean Square Error based on the remaining set of the time slices. During separate experiments, different proportions of the total data set were used as training and test subsets. Also, some experiments had a static test set that was randomly selected before Generation 1 and was used for all generations in the experiment, while others randomized the test set (using the same proportions) before each new generation. This was done to test the efficacy of training, testing and ranking the BPNNs' performance based on static or dynamic data set selection. (see Table Y below) It should be noted that the experimental runs below are not listed in numerical order. This is because it was necessary to re-run some earlier trial settings to capture more extensive statistical data over the course of the experiment.

Table III.B.4: Experiment Settings

Run #	Linear Threshold	Threshold Alpha	Threshold Omega	Threshold Squeeze	Static Test Set	Test Set Proportion	Final Average MSE
29	Y	0.0005	0.00005		Y	10%	0.01169
30	Y	0.0005	0.00005		Y	10%	0.011595
31	Y	0.005	0.0005		Y	10%	0.011707
32	Y	0.005	0.0005		Y	10%	0.011234
33	Y	0.05	0.005		Y	10%	0.01173
34	Y	0.05	0.005		Y	10%	0.012568
17	N	0.05		0.95	Y	10%	0.011367
18	N	0.05		0.95	Y	10%	0.011395
25	N	0.05		0.95	Y	20%	0.012356
26	N	0.05		0.95	Y	20%	0.012487
27	N	0.05		0.95	Y	30%	0.012847
28	N	0.05		0.95	Y	30%	0.012684
19	N	0.05		0.95	N	10%	0.011925
20	N	0.05		0.95	N	10%	0.011657
21	N	0.05		0.95	N	20%	0.013019
22	N	0.05		0.95	N	20%	0.012035
23	N	0.05		0.95	N	30%	0.013495
24	N	0.05		0.95	N	30%	0.013786

The network was trained in repetitive epochs until it achieved an MSE score below the desired threshold or reached the arbitrary epoch limit. During some experiments, the threshold tightened as the generations progressed, according to a linear function, and several alpha/omega threshold values were applied over the course of this experiment. During other experiments, the threshold started at an arbitrary alpha value, and was then set in each subsequent generation to a percentage (95%) of the MSE score of the top performing member of the preceding generation. The results of these settings can be seen in the charts below (see VII.A).

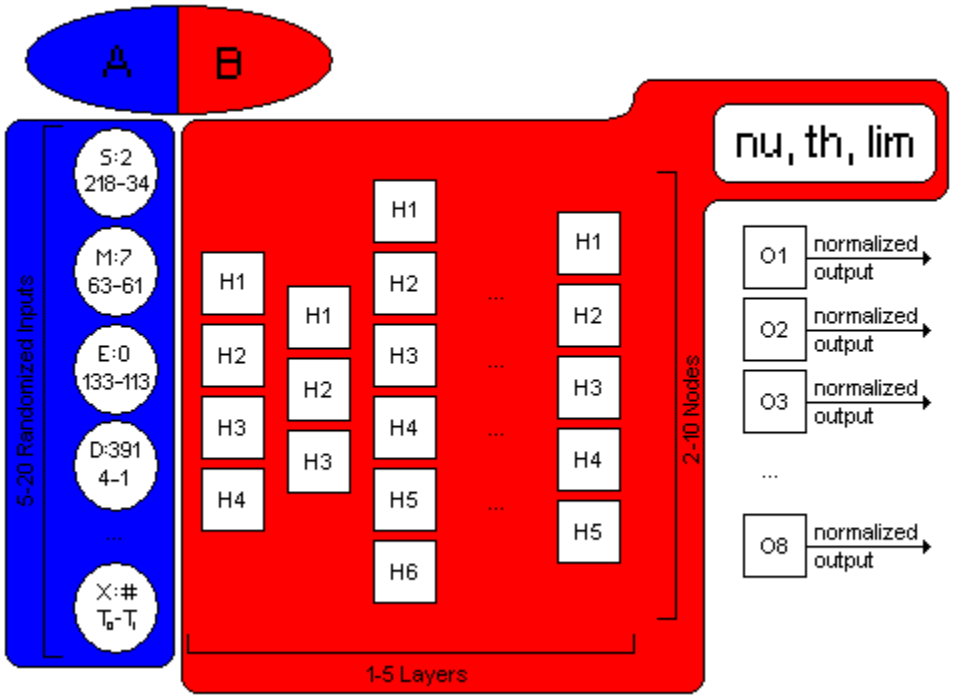
The evaluations of a network's accuracy and training efficiency were available for use in the scoring function, discussed below.

C. **Evolutionary Programming**

Using a method which combines traditional Back Propagation Neural Networks and Evolutionary Programming, the primary factors leading to expression and suppression of the subject's UC symptoms were derived. In short, an Evolutionary Programming approach was used to breed an optimized BPNN for predicting the expression of UC symptoms (outputs), given the factors (inputs) with a given lead time window, as described above.

To achieve this result, an initial population of BPNNs was created, each with attributes randomized according to the parameters described above. These parameters were defined in a class named Genome (see code Appendix X, section Y). Each BPNN was initialized according to the parameters of its genome (inputs, body parameters, learning rate variables), with two sections of the genomic information classified as A/B (see Figure #.#). These initial members were then trained on the input data as described above, and scored based on a multifactor fitness algorithm.

Fig III.C.1: Genome Subsections



1. **Scoring Algorithm**

The algorithm was designed primarily to punish three aspects of poor performance, with some aspects playing a more important role during different phases of the generational progression. First, inaccuracy is the most detrimental to the goal of this experiment. Therefore, averaged Mean Square Error of the prediction nodes over the test set played a major role throughout the generations.

Second, elimination of redundant or irrelevant inputs is salient to the goal for which we implemented this method. Finding the most relevant factors requires not only the discovery of highly relevant input configurations, but also the elimination of less-relevant and completely irrelevant inputs. Finally, for the purposes of speedy and productive generation turnover rate, inefficient BPNN body configurations (with needlessly bulky networks and inefficient learning rates) were undesirable. This factor was the most important when eliminating clearly underperforming members in the early generations, and its influence was decremented relative to accuracy over time. Pressure to reduce the overall link count of a BPNN promoted this corrective pruning of the body configuration to increased overall efficiency and pruning of the inputs to improve accuracy.

Both of these factors (inaccuracy and inefficiency) were meant to be minimized over time. Therefore, the score of a BPNN was calculated by evaluating the inverse of the sum of the two factors combined (see code Appendix), where g = generation number, l = link count, and t = training passes:

Fig. III.C.2: Scoring Algorithm

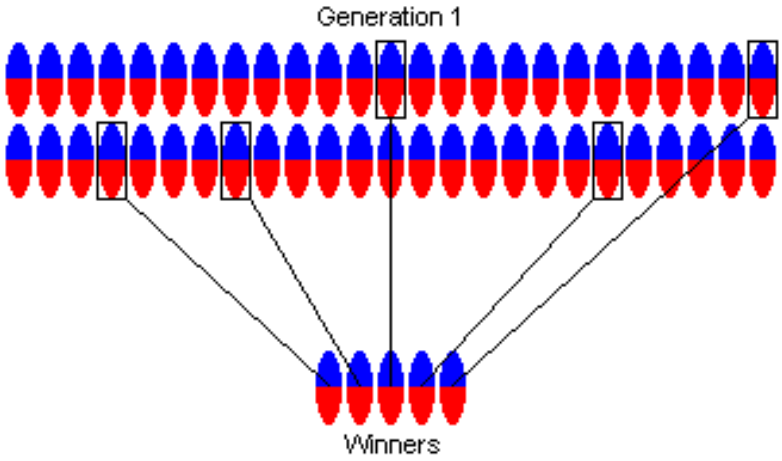
$$S(X) = \frac{1}{(MSE * g^2) + (l * t)/10}$$

2. **Initialization and Generational Propagation**

Each member of the first (randomized) generation was passed through the initialization, training, testing, and scoring procedures. The highest ranking portion (5 of 50) of

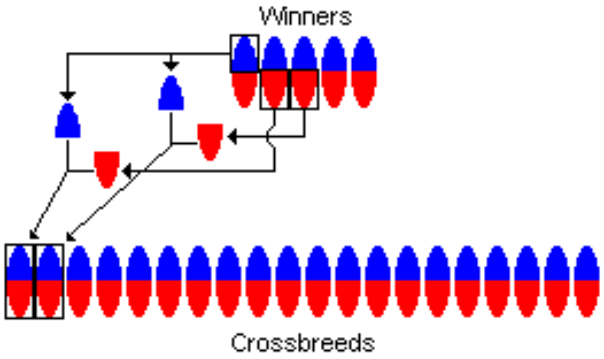
the members was retained and copied on to the next generation, while all other genomes were discarded.

Fig. III.C.3: Selection Method



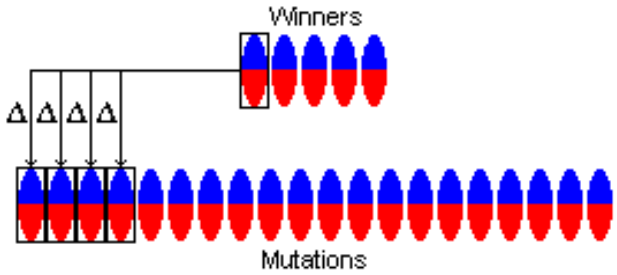
The next 20 members of the second generation were created by crossbreeding attributes from the 5 retained members of the first generation.

Fig. III.C.4: Crossbreeding Method



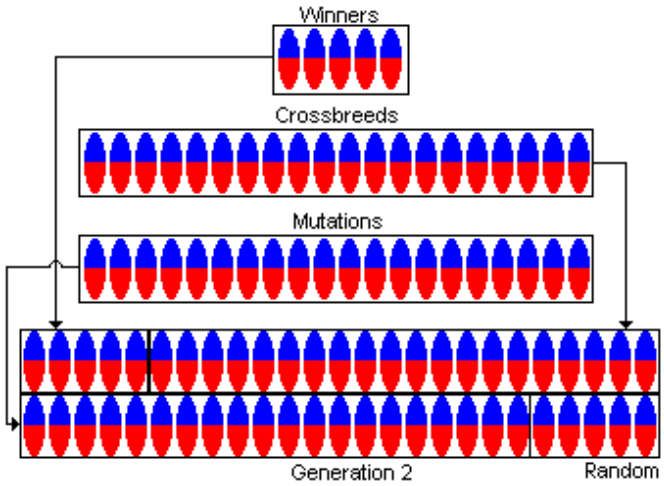
The next 20 members of the second generation were created by implementing minor mutations of the 5 surviving members (4 mutations for each). The crossbreeding and mutation methods are described in the following two subsections.

Fig. III.C.5: Mutation Method



The final 5 members of the second generation were created by the same randomization that was used for every member of the first generation.

Fig. III.C.6: Generation (n+1) Composition



Each of these methods promotes a certain theoretical approach to searching a solution space. A portion of the best performers were retained, because in a stochastic search there is always the chance that the optimal solution has already been found. It must be assured that the best solution, once found, is not discarded.

Crossbreeding is performed because there is a chance that two members of a generation can each contain a different merits and flaws that affect their performance. It is hoped that a chance crossbreeding might result in an offspring that contains the merits, but discards the flaws, resulting in a better overall BPNN.

Mutation is used to produce progeny from each of the top performing BPNNs because it is possible that any BPNN is a small distance away in the solution space from a marginally better performing configuration. This method can be understood under the umbrella of Local Search or Stochastic Annealing[14]. It is hoped that this method will find local maxima in a relatively optimal solution space, in the later generations.

Finally, a portion of each new generation is completely randomized, since there is always a chance that the solution population is stuck within a local maximum, and is missing an entire portion of the solution space where performance is far superior. This method of randomization is far more important in the very early generations than in any other part of the experiment. However, it would be unwise to eliminate it entirely at any point. Therefore, only a small portion of each successive generation was dedicated to freshly randomized members.

3. **Crossbreeding Method**

For the purposes of crossbreeding, Genome parameters were split into two distinct sections (Fig. III.C.1). The first section (A) was comprised of instructions pertaining to the configuration of the inputs of a BPNN. This section contained data describing the number of inputs and configuration data describing the parameters of each of these.

The second section (B) was comprised of configuration parameters for the body of the BPNN. This included the number of hidden layers between the input and output layers, as well as the number of nodes in each respective layer. Also contained in this portion of the Genome was contained the values for the learning rate and its associated learning momentum.

Following the direct transfer of the top ranking members of the current generation to the next generation, a number of next-generation members were created from the Crossbreeding method. This consisted of exhaustive A/B combinations of the top retained members. For this

experiment where 5 members of the current generation were retained, the crossbreeding method produced 20 more members for the following generation, of the form:

Fig. III.C.7: Crossbreeding Permutations

$$\forall G_{gx}\left(\forall G_{gy,\ x!=y}\right): G_{gx\mathrm{A}} + G_{gy\mathrm{B}} \rightarrow G_{(g+1)\mathrm{AB}}$$

Where G_g is a retained high ranking member of the current generation.

4. **Mutation Method**

The mutation methods used were intentionally small in their reconfiguration of aspects of the BPNN. Each retained member of the current generation produced 5 new members of the following generation by random mutation, creating 25 new, slightly altered versions of top performing configurations. These mutations typically added, removed, or randomized a single parameter of the Genome. For example, one mutation would add a single neuron to one of the hidden layers; another mutation would remove an input node; another would create a whole new input node; still another would remove a whole hidden layer. For a full list of mutation types, see Fig. III.C.8 below, or in code (see Appendix X, section Y, Genome::initMutant()).

Table III.C.8: Mutation Types

Value Adjustment	Body Configuration	Input Configuration
RandomSeedReset	HiddenLayerAdd	InputRemove
LearnRateIncrease	HiddenLayerRemove	InputAdd
LearnRateDecrease	HiddenNodeAdd	InputReplace
LearnRateRandom	HiddenNodeRemove	InputRangeFrontIncrease
MomentumIncrease		InputRangeFrontDecrease
MomentumDecrease		InputRangeBackIncrease
MomentumRandom		InputRangeBackDecrease
		InputRangeRandom

5. **Ending the Experiment**

After many hundreds of generations of applying this method, the experiment arrives at a fairly stable (and usually highly uniform) population of high-performing BPNNs. The top scorers have very low MSE over a wide variety of possible inputs. The selection pressure algorithm eliminated input nodes that represent redundant and less-relevant data. These experiments are halted after 1000 generations of propagation and selection. At this point the ultimate goal has been achieved. We are left with a set of input nodes that represent the most relevant factors to the expression and suppression of Ulcerative Colitis. All that is left is making these input nodes human-readable, and deriving useful information from the BPNNs for the purposes of diagnosis and prescription.

6. **Miscellaneous Optimizations**

Early during the coding task, it became apparent that certain configurations of BPNN would become entrenched in the top five (5) retained positions, making the EP search of the problem space much slower and less efficient than it could have been. Retaining top performers was important to deriving an optimized BPNN, but spending extra training cycles making identical networks compete was undesirable, and could be mitigated in two relatively easy ways.

a. **Generation Shuffling**

The first method created was to shuffle the order of the members of a new generation, just after instantiation and before training. Testing and scoring happens sequentially. Without shuffling, it was possible to repeatedly retain five identical members from generation to generation, while ignoring equally scoring members with different configurations. For the purpose of searching the solution space, diversity is always preferable to uniformity. Allowing previous winners to continue solely by inertia was undesirable, and this simple shuffling action helps avoid that pitfall.

b. **Crossbreeding/Mutation Substitution**

The second method created was to substitute the Crossbreeding action with Mutation actions in the case when the top retained members were identical. Crossbreeding five identical members produces twenty five identical members in the following generation. Once again, uniformity is not specifically detrimental, but it does slow down the process of evolution, and wastes the opportunity to explore new local areas of the solution space. Prioritizing Mutation when Crossbreeding is going to prove unhelpful makes the process more efficient and productive (see Appendix X, section Y).

D. **Output Processing**

Once the preceding combination of methods breeds a BPNN that accurately and efficiently predicts the expression and suppression of UC symptoms, given the restricted number of relevant inputs, this Genome configuration is used to determine which of the many factors in the problem space are causative and suppressive (see Appendix X, section Y).

1. **Optimizer**

With only the Genome data, all that can be determined is which input factors are most relevant to the problem space. However, without a further investigation of these factors, we could not determine which of the factors were causative, and which were suppressive of the UC symptoms. To this end, a program was created that constructs a BPNN from the highest-performing Genome of the final generation, sets a target maximized or minimized output (1.0 or 0.0 for all output nodes), then exhaustively test combinations of input (see Appendix X, section Y, RunOptimizer()) to determine which input settings most closely elicited a match for the targeted output (denoted by smallest MSE). For reasons of combinatorial complexity, it was

determined that testing a BPNN with 13 or fewer inputs with all possible permutations of the values 0.0, 0.5, and 1.0 would produce the desired knowledge in acceptable time. This was done in a base-3 odometer fashion. For example, the first 12 settings for a BPNN with 5 input nodes are as follows:

Fig. III.D.1: Optimizer Input Permutations

Input 1	Input 2	Input 3	Input 4	Input 5
0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.5
0.0	0.0	0.0	0.0	1.0
0.0	0.0	0.0	0.5	0.0
0.0	0.0	0.0	0.5	0.5
0.0	0.0	0.0	0.5	1.0
0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0	0.5
0.0	0.0	0.0	1.0	1.0
0.0	0.0	0.5	0.0	0.0
0.0	0.0	0.5	0.0	0.5
0.0	0.0	0.5	0.0	1.0
...	...	...	...	...
1.0	1.0	1.0	1.0	1.0

a. **Diagnostic Setting**

In order to determine which factors in the problem space caused the most severe expression of UC symptoms, the targeted output for the optimized BPNN was set to 1.0 for every output node, equivalent to the “full expression” condition. When the Optimizer finished testing every permutation of the trinary activation values, the combination with the lowest MSE on the targeted output of 1.0 was determined to be the combination of inputs that caused the highest expression of UC symptoms. For this reason, it is called the Diagnostic Setting.

b. **Prescriptive Setting**

Conversely, to determine which factors in the problem space induced the most complete suppression of UC symptoms, the targeted output for the optimized BPNN was set to 0.0 for every output node, equivalent to the “no expression” condition. When the Optimizer finished testing every permutation of the trinary activation values, the combination with the lowest MSE on the targeted output of 0.0 was determined to be the combination of inputs for the best treatment of UC symptoms. For this reason, it is called the Prescriptive Setting.

2. **Human Readability**

Though it would be possible to discover which input types, quantities, timeframes, and activation values were most relevant from a direct reading of the Genome data and Optimizer runtime output, it would be tedious, time consuming, and error prone. For this reason, the Optimizer was created with a portion of code that produces human readable output. This output consists of a list of input type, quantity, and lead time for each input node, in combination with a header that denotes whether the output is related to either the Diagnostic or Prescriptive setting.

These findings can be used to inform personal and medical decisions in service of disease treatment. This derived knowledge is particularly helpful for chronic illnesses with highly personal influencing factors, such as UC and other digestive diseases and syndromes.

The Optimizer output for all 6 experiments was compiled and is shown in the table below, sorted by relative frequency of factor types (see VII.B).

3. **Statistics Graphs**

For the purposes of experiment interpretation, a number of graphs, demonstrating trends in accuracy (MSE), input nodes, BPNN dimensions, and training thresholds over time were needed. A small program to produce comma separated value files for the retained members of each successive generation were produced. Then these files were imported into Microsoft Excel, and generation-series graphs were created from the data (see VII.A).

IV. RESULTS AND DISCUSSION

A. **BPNN and EP Performance**

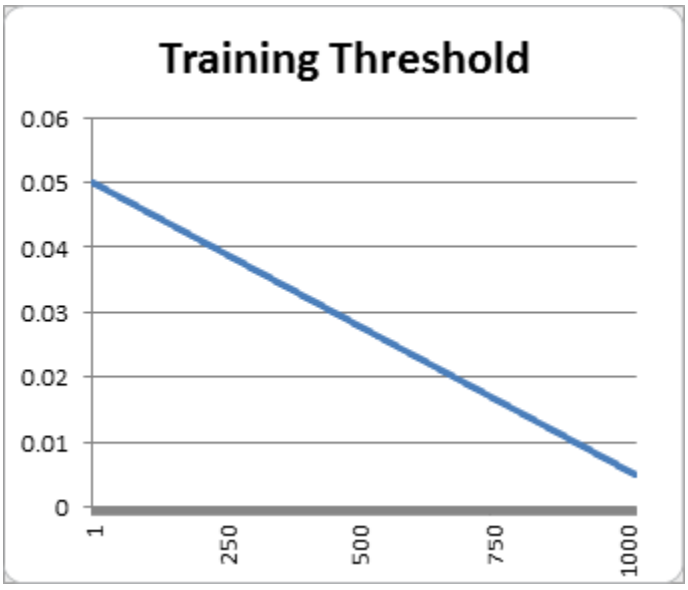
For the purposes of evaluating the performance of the methods described above, an array of statistical graphs was created for each experimental run. These graphs can be viewed in their entirety below (see VII).

Directly below is discussion pertaining to individual graph types and aspects of a single run. For general discussion of each type of graph (subsection IV.A.1 through IV.A.12), the example of experiment “Run 29” will be used. Some discussion of specific aspects of differing experimental settings for individual experimental runs follows the general overviews.

For each graph (not including Training Threshold), the data point (y-axis) at each generation (x-axis) represents the averaged value of the relevant unit for the top five (5) retained members of that generation.

1. **Training Threshold**

Fig. IV.A.1: Training Threshold

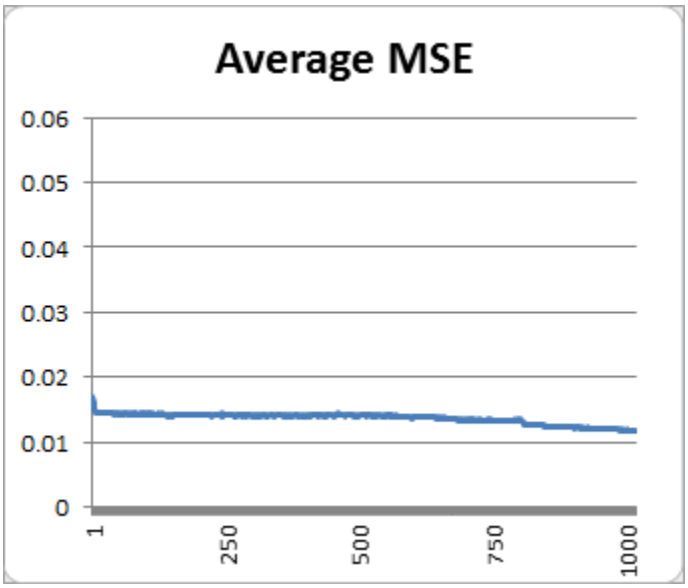


During the course of each experiment, the training threshold (used to determine when to discontinue BPNN training) was varied over time. This variation over time was intended to increase selection pressure among the populations, in order to derive optimal configurations.

In experiments utilizing a linear training threshold squeeze, the Training Threshold graph displays a straight line, sloping downward as the generations progress. In experiments utilizing a dynamic threshold, each generation calculated its threshold by multiplying the preceding generation's average MSE statistic by a value of 0.95. The Training Threshold graphs for these experiments indicate a threshold that very closely mirrors their respective Average MSE graphs.

2. **Average MSE**

Fig. IV.A.2: Average MSE



During the course of each experiment, the average MSE of the top five (5) ranked BPNNs for each generation was recorded. It was not unusual for the many of the fully randomized first-generations' members to have very MSEs (often over 0.5). However, even a

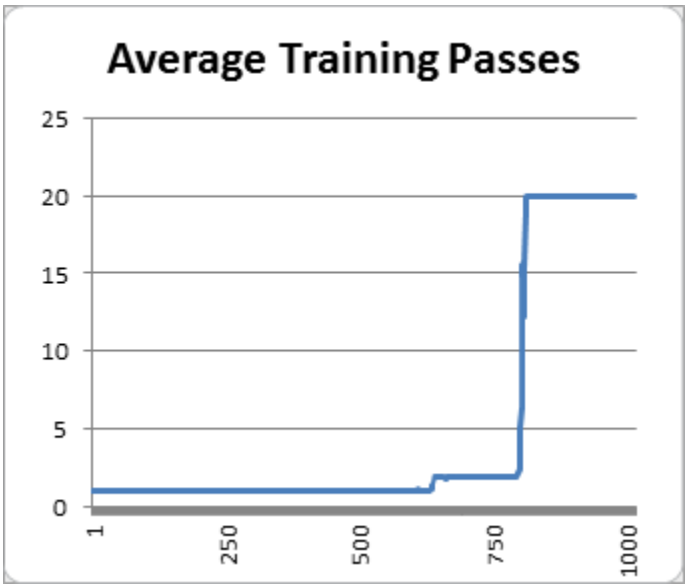
small number of well-performing members to eliminate these poorly competing siblings. Therefore, in every experiment, the average MSE dropped rapidly from its initial value at Generation 1 (typically around 0.02), and then begin a steady decline to its final value at Generation 1000.

It is clear from the steady, gradual decline of the MSE in each experiment that utilized a static test set, especially in Run 33 and Run 34, where the populations spent a significant number of generations without running into the Training Threshold, that the selection pressure applied by the Scoring Algorithm over the population accomplishes the same goal of enforcing measurable accuracy and efficiency in each BPNN. Since this is the purpose of the Training Threshold in standard BPNN experiments not utilizing Evolutionary Programming, it is possible that future BPNN / EP experiments may be able to eliminate arbitrary thresholds entirely.

Runs that utilized a variable test set displayed a much more volatile progression of MSE scores, the reasons for which are discussed in the Average Training Passes section below. However, it should be noted that even these achieved final accuracy that is roughly equivalent to runs with static test sets.

3. **Average Training Passes**

Fig. IV.A.3: Average Training Passes



During the course of each experiment, the average Training Passes of the top five (5) ranked BPNNs for each generation was recorded. This statistic shows one of the most important aspects of computational complexity and experimental design efficiency. Any training process that forces multiple (or maximum) number of training epochs to occur when the Scoring Algorithm and selection pressure could be doing the same work at no additional cost is clearly wasting valuable computation time. This is especially wasteful in the early half of the experiment where it's possible that many of the members of the populations will be large, inefficient, and relatively inaccurate.

The aforementioned arbitrary Training Threshold gives the training sequences permission to cease training early when the MSE is below the threshold. Perhaps the same effect can be achieved by an arbitrary maximum number of training epochs per BPNN that grows as the generations progress.

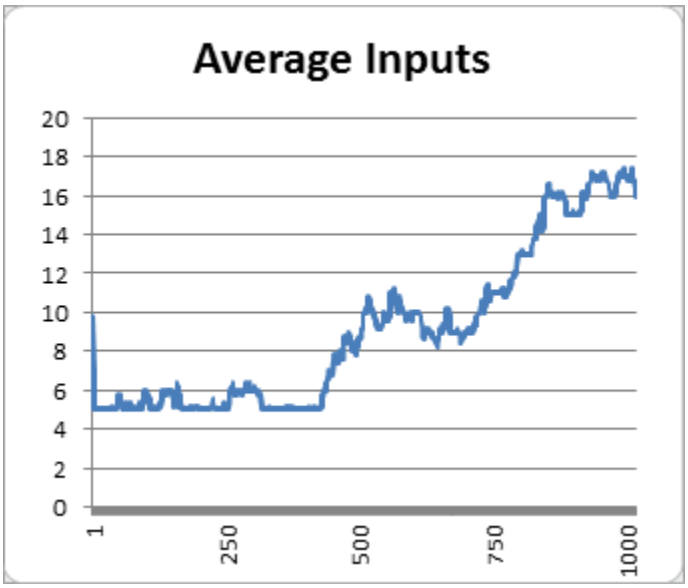
Some interesting effects of different experimental settings can be seen in this graph. Runs that utilize a linear training threshold jump straight up to the maximum number of training epochs, as soon as the average MSE nears the threshold, which can be seen in Run 29-34. Runs that utilize a dynamic threshold and have a static test set (one set generated before Generation 1) likewise jump straight up to the maximum training epochs at Generation 2, as soon as they have the aforementioned 95%-MSE threshold enforced, seen in Run 17, 18, 25-28 (see VII.A).

However, runs that utilize both a dynamic threshold and variable test set (one generated anew at for each subsequent generation) have wildly varying numbers of training epochs, seen in Run 19-24. This is likely because some new test sets will be similar to the preceding generation's test set, allowing all retained BPNN configurations to perform relatively well without extensive training, while other test sets will be dissimilar to the preceding generation's test set, forcing all retained BPNN configurations and their descendants to train extensively to attempt to meet the 95%-MSE threshold.

Throughout all experiments, in both linear and dynamic threshold cases, as well as both static and variable test-set selection cases, the proportion of data entries (10%, 20%, 30%) selected as the test set did not result in any discernable statistical effect on either general performance or final-generation MSE.

4. **Average Input Nodes**

Fig. IV.A.4: Average Input Nodes



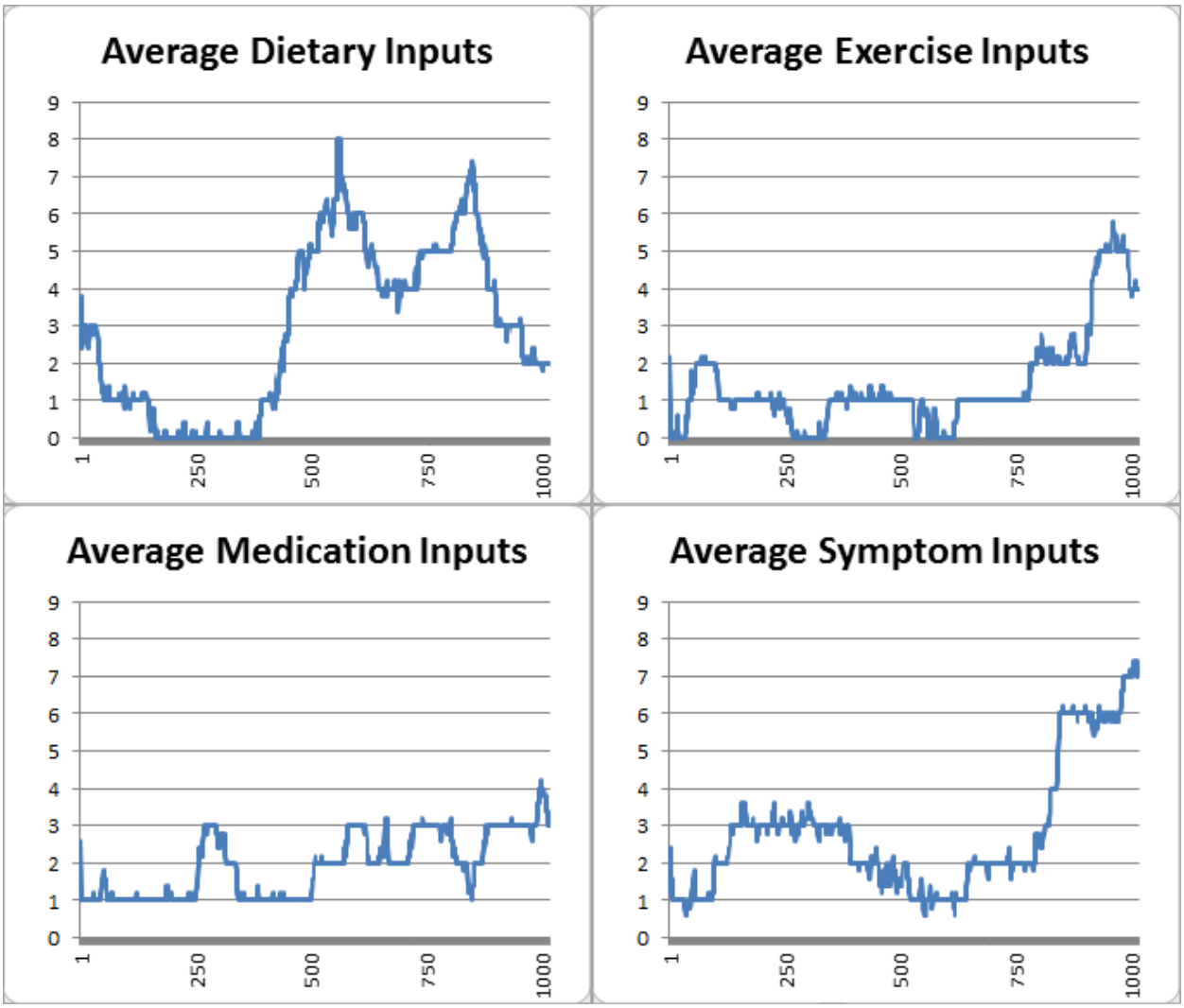
During the course of each experiment, the average number of Input Nodes of the top five (5) ranked BPNNs for each generation was recorded. It is here that evidence of the validity of the core method of the experiment is strongest. The portion of the Scoring Algorithm responsible for enforcing the minimum number of input and hidden nodes is in effect throughout the duration of the experiment. The way that old input nodes are retained and new input nodes are verified as relevant is by justifying their contribution to predictive accuracy.

Every experiment has produced an Average Inputs graph of this general form. Generation 1 begins with a random number of inputs. The selection pressure enforced by the Scoring Algorithm drives out most of the irrelevant inputs, forcing the average down to near the minimum number (5) in the early generations. As selection competition increases as the generations progress, random mutation of existing BPNNs derive children which happen to have more relevant input nodes. When the new input nodes represent factors and prediction windows that are relevant enough to the prediction accuracy of the BPNN to justify their

existence, those nodes are retained. The fact that every Average Inputs graph displays a gradual upward trend after the initial culling, combined with the gradual downward trend of MSE noted earlier demonstrates that the Evolutionary Programming method of random exploration of the solution space (input configurations) for a sufficient number of generation distils the most relevant prediction factors among a massively multivariate data set.

5. **Average Category Input Nodes**

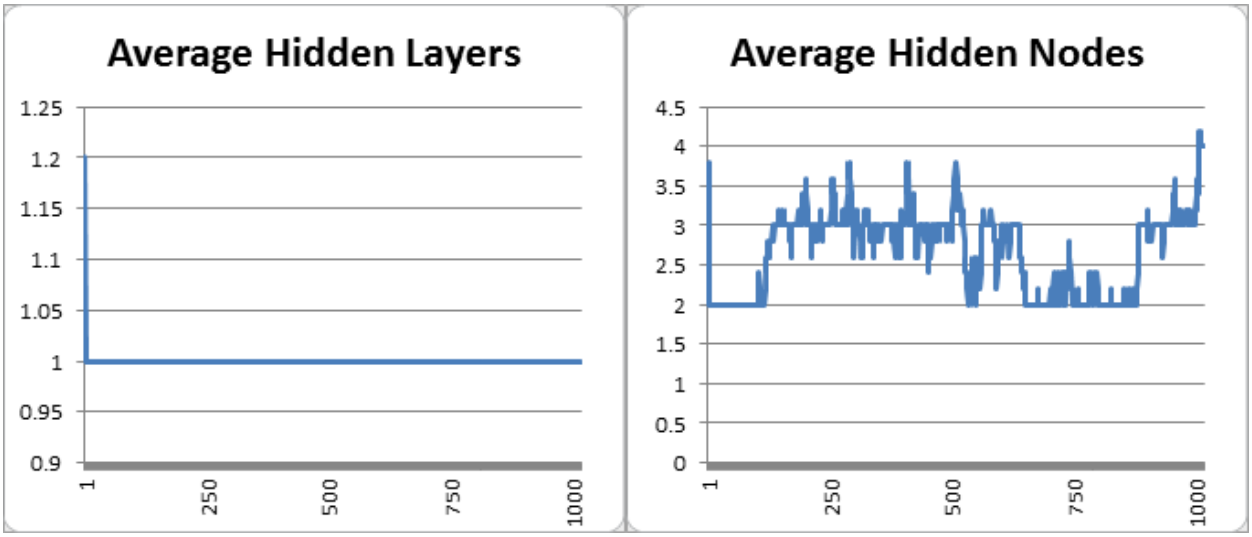
Fig. IV.A.5: Category Input Nodes



During the course of each experiment, the average number of Category Input Nodes of the top five (5) ranked BPNNs for each generation was recorded. To gain insight into the actual problem space at hand (medical treatment of Ulcerative Colitis), input nodes as categorized by general treatment type were also graphed. Further discussion of the relative importance of each result is handled below (see section IV.B and IV.D).

6. **Average Hidden Layers and Nodes**

Fig. IV.A.6: Average Hidden Layers and Nodes



During the course of each experiment, the average number of Hidden Layers and Hidden Nodes of the top five (5) ranked BPNNs for each generation was recorded. These graphs reflect the derived optimal configuration of the body of the BPNN. The black-box nature of BPNN function makes body configuration a particularly difficult concept. It has been noted in previous work, especially concerning logic such as the XOR perceptron.

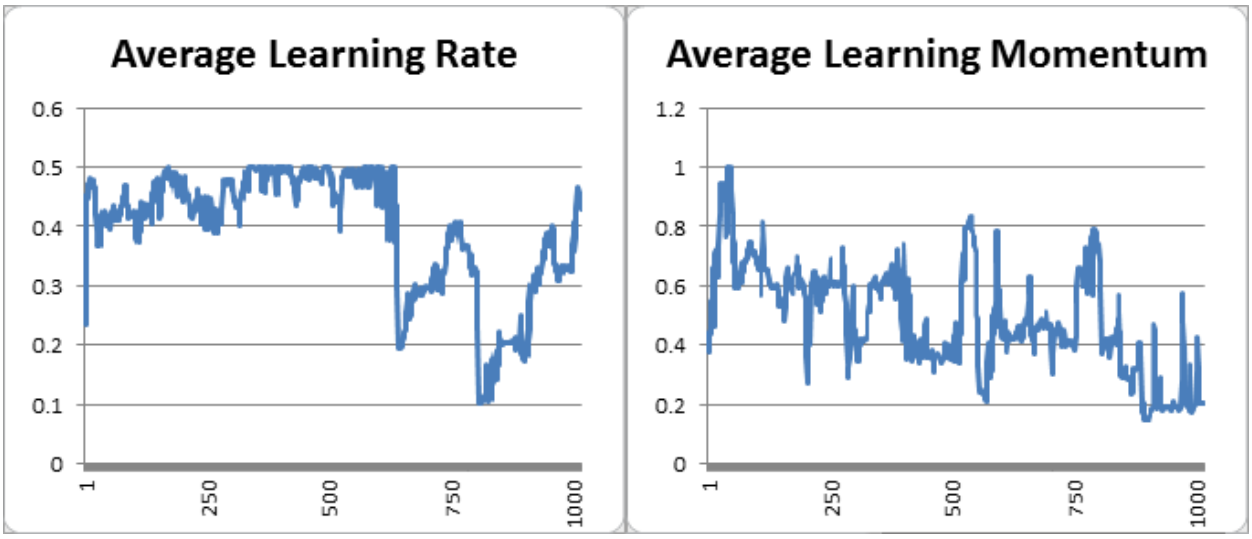
Work has been done to bring EP to bear on the problem of constructing network topology that learns XOR in optimal time[13]. This approach was used to derive BPNNs over the

course of each experiment that successfully encoded the complexity of the relevant input interactions without wasting computation time on unnecessarily bulky configurations.

The result of the Scoring Algorithm's pressure to reduce the number of internal node links (and by proxy, the total number of nodes) was to reduce the number of Hidden Layers in each experiment rapidly to the minimum number (1) and in most experiments to have a slight, but gradual increase in the number of Hidden Nodes toward the end of the experiments, as relevant input nodes were discovered and the predictive complexity and accuracy of the BPNNs increased.

7. **Average Learning Rate and Momentum**

Fig. IV.A.7: Average Learning Rate and Momentum



During the course of each experiment, the average Learning Rate and Learning Momentum of the top five (5) ranked BPNNs for each generation was recorded. Each of these appears to be a random walk. It appears that whatever BPNN lineage was currently dominating the population carried with it random Learning Rate and Momentum values. There are no

discernable patterns in relation to generation, number of inputs, MSE or training threshold. Perhaps the minimum and maximum values for each were within an effective range for both.

B. **Discovered Relevant Inputs**

The following graphs represent the summed instances of each type of input that was present in the final generation, for all experiments. For example, the label “Prednisone” with the value 17 indicates that across all 18 experimental trials, there were a total of 17 input nodes representing ingestion of Prednisone at some window within the given leading time range. These graphs have been separated into Symptom Factor Frequencies and Non-Symptom Factor Frequencies, for ease of interpretation, regarding controllable (Non-Symptom) and non-controllable (Symptom) prediction factors.

Fig. IV.B.1: Symptom Factor Frequencies

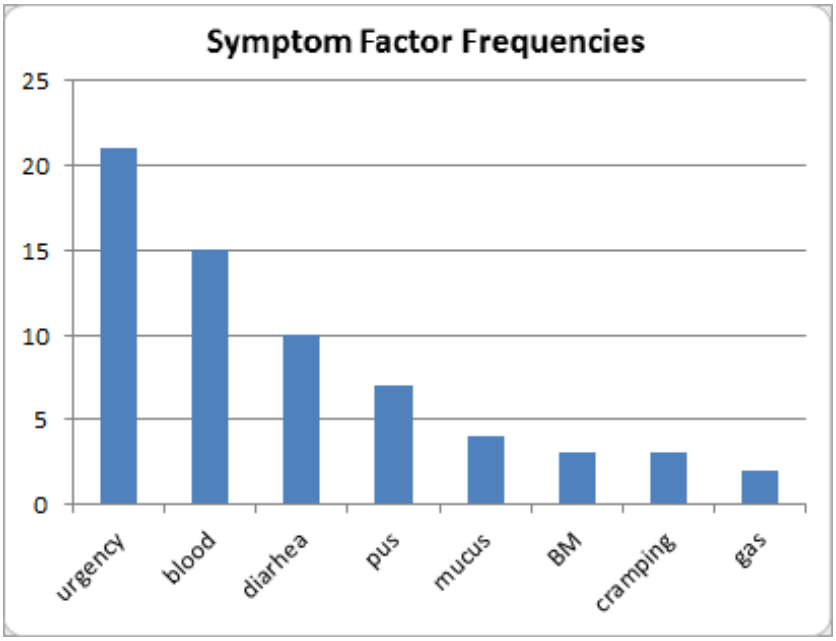
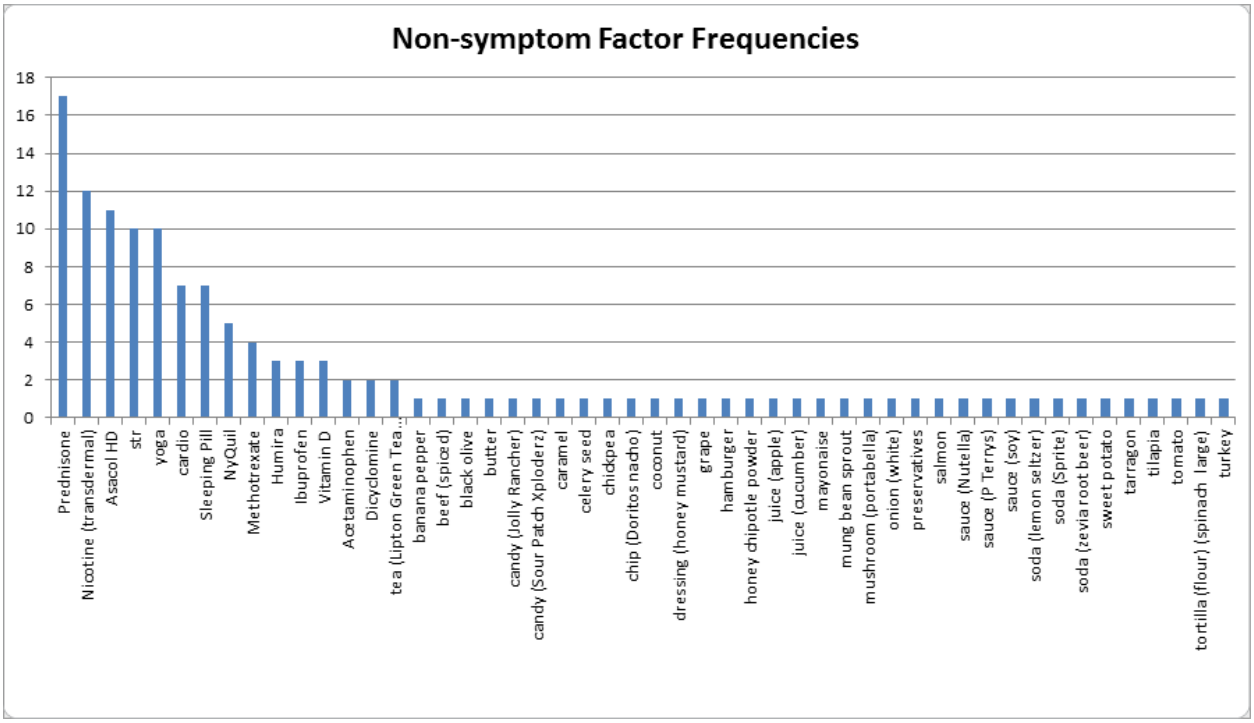


Fig. IV.B.2: Non-Symptom Factor Frequencies



There are two important aspects to these graphs which represent a success in the relevant input seeking method applied in these experiments. First is the fact that there are a relatively large number of Symptom Factors represented across the 18 experiments. Ulcerative Colitis is a disease whose expression and suppression periods can be described in “flares” and “remission”. The onset and fade of symptomatic expression often takes course over periods of weeks or even months. Because of this, there is reason to expect that given prediction increments of 3 hours and input windows of up to two weeks, recent symptomatic expression should be a very strong predictor of current symptomatic expression. This is confirmed not only by the high numbers of Symptom inputs, but also by which symptom inputs are most frequently derived by the EP process. For example, “gas” is a relatively transient symptom, often caused by specific dietary intake, and not often a simple symptom of UC. However, “urgency”, “blood”, “and “diarrhea” are the most pronounced and persistent symptoms of UC and it is reasonable to expect that they would be highly represented among the Relevant Factor Inputs.

Second, among the Non-Symptom Factors, three medications which are known to have medical efficacy, both from personal experience and by medical study are Prednisone, Asacol[1], and Transdermal Nicotine[15]. As with the Symptom Factors, these medical treatment factors were likewise expected to show strong relevance in the Non-Symptom Factors. This graph is another demonstration that the EP method applied achieved the desired result.

However, at this stage in the experiment, though efficient prediction engines (optimized BPNNs) have been produced, the results are still not human-readable. What was needed was a method by which Relevant Factors could be described as either “expressive” or “suppressive” of the disease symptoms.

C. **Optimizer Program**

The Optimizer Program was designed to do exactly this. It used exhaustive combinatorial testing on optimized BPNNs to determine which Relevant Factors were expressive and which were suppressive of UC symptoms. The process is described above (see section X.X). The results are as described by the following graphs.

Fig. IV.C.1: Optimizer Key

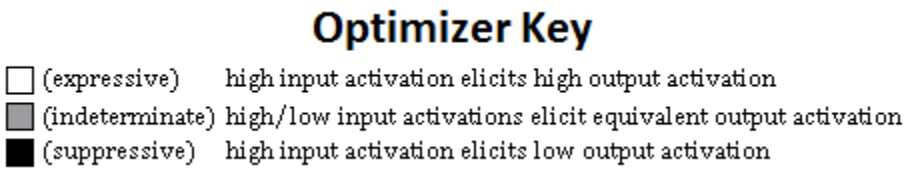
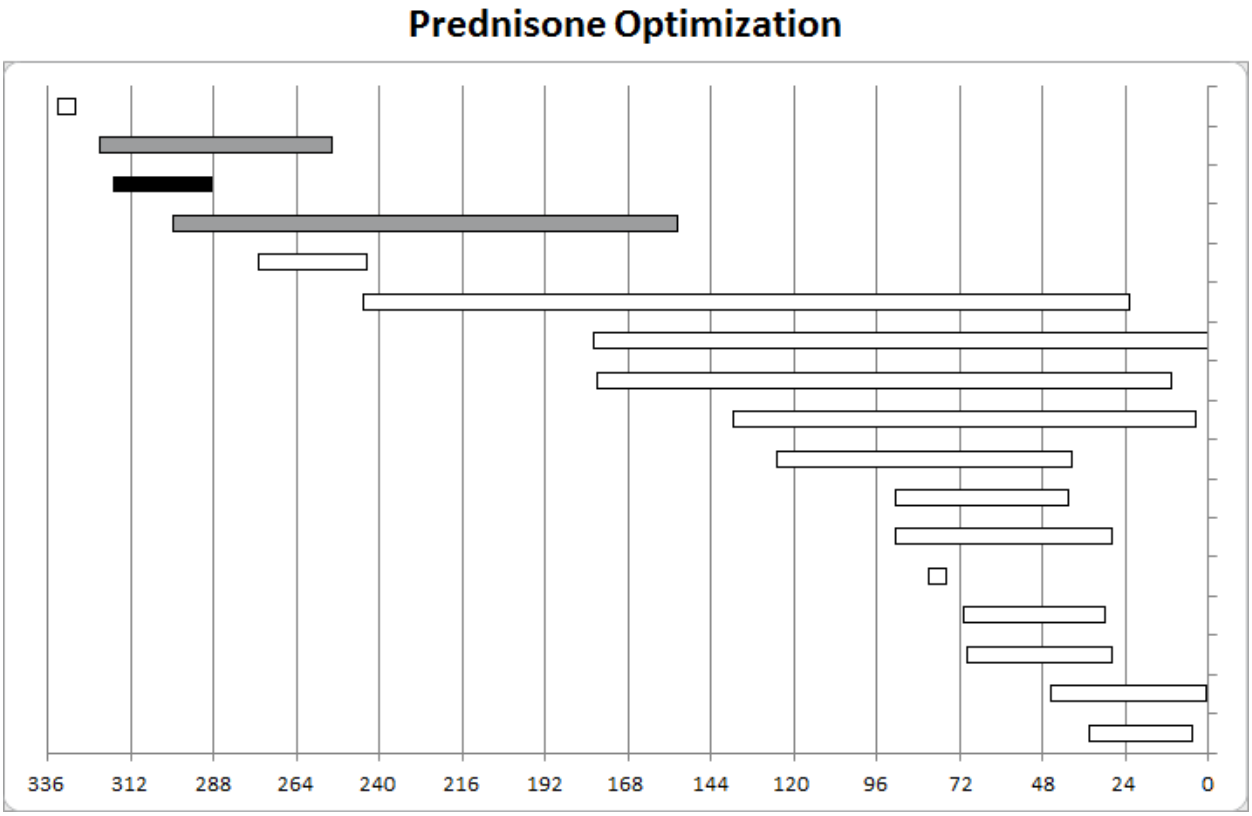


Fig. IV.C.2: Optimization Example



The left-hand extreme of the graph represents the maximum lead time of an input node's sample window (336 hours), while the right-hand extreme represents the minimum lead time (0 hours). Each bar on the graph represents one input node from one of the experimental runs, and its associated sample window preceding a prediction.

The Optimizer Program set target output activations of either 1.0 for all nodes (full expression, "Diagnostic") or 0.0 for all nodes (full suppression, "Prescriptive"), then performed an exhaustive permutation of all inputs, searching for the combination of inputs which produced the lowest MSE for the given target output (either all 1.0 or all 0.0).

Bars that are shaded in white represent inputs for which minimized MSEs for the low target output (0.0) were correlated with high activation of the graphed input node. For example:

if a bar on the Prednisone graph is shaded white, then sampling a HIGH intake of Prednisone within that window before a prediction is likely to correlate with a LOW activation of output nodes.

Bars that are shaded in black represent the opposite correlation. If a bar on a Relevant Factor's graph is shaded black, then sampling a HIGH intake of the factor within that window before a prediction is likely to correlate with a HIGH activation of output nodes.

Bars that are shaded in gray are indeterminate. For these inputs an equal activation of the input node during the associated sample window was observed in both the high target (1.0) and low target (0.0) conditions. It is likely that these nodes were relevant, but less so than some other factor in the same BPNN, and were overridden in the correlation by a more pronounced event, dominated by that other factor.

It is important to remember that these Relevant Factors and Optimizations do not strictly represent causality, but rather consistent correlation over the data set. One more step is necessary to determine true causality. Unfortunately, for medical reasons, this step did not fit within the data collection period for these experiments. Therefore, the final step to determine causality is discussed in the Further Study portion below (see VI).

V. CONCLUSIONS

A. **Primary Goal: Determining Relevant Factors via BPNN and EP**

As discussed earlier in sections III.C.1 and IV.A, the Scoring Algorithm provided constant selection pressure to both prune excess body topography and irrelevant input factors as well as reward prediction accuracy and training efficiency results in final-generation BPNNs that are not only fast, but contain only the most relevant input factors to achieve high accuracy.

In such a massively multi-variant problems space as personal medical data, designing such a network intentionally would result in highly inaccurate predictive capacity. Eliminating irrelevant factors would take intense calculation and interpretation to rule out statistical correlation among large-n dimensional factor interactions. This is not only time prohibitive, but also inefficient, as it applies a brute-force method to a problem that can be optimized within an acceptable margin of error via a smart random-search method that has proven effective, as the Evolutionary Programming method used in these experiments.

Robust prediction engines for vastly complex, unintelligible and irrational problem spaces can be derived by setting reasonable parameters, beginning the evolutionary process, and collecting the final-generation BPNN.

B. **Secondary Goal: Determining Expressive and Suppressive Factors**

Interpreting the configuration of an optimized BPNN is a bit more difficult of a task than simply letting the evolutionary process discover that optimal configuration. Simply reading the parameters of each Relevant Factor input node does not tell us precisely the size and direction of the activation effect of each. Determining these effects takes extra probing. Such probing was done with an Optimizer program (see IV.B).

Output-targeting and exhaustive permutations of input settings elicited a sometimes clear and sometimes indeterminate picture of the impact of each Relevant Factor. Other Optimization programs could potentially be constructed to probe these relationships and provide better insight. One possible way is discussed below (see VI.A).

It is important to note that the networks predictive effects cannot be interpreted strictly as expressive or suppressive causality, but only as correlation between symptomatic expression and prior symptomatic, dietary, medical and exercise events. For example, the Relevant Factor and Optimizer graphs for exercises (strength training, cardio training, and yoga) tempt the viewer to interpret the influence of exercise as expressive of UC symptoms. Consensus among the medical and body-building community, however, point to awareness of the exact opposite effect[6][18].

While it is tempting to interpret consistent correlation as true causation, this must be avoided and steps must be taken to illuminate false correlations. Methods that may be undertaken to achieve this are discussed further below (see VI.B).

C. **Tertiary Goal: Medical Treatment Plan**

Beyond achieving a solid proof-of-concept of the theory described by the title and Section II.C of this thesis, this product of these experiments were intended to be a tool amenable to human-readability and capable of producing insight into the medical condition of Ulcerative Colitis, potentially including suggestions for treatment plans (including diet and exercise) that would produce positive results in the absence of harsh drugs such as Prednisone, Humira, Methotrexate, et al.

While this particular outcome was not achieved, the BPNN/EP method and Optimizer did provide clear medical insight into my personal experience of UC. After data collection was complete, it became obvious that remission and even temporary suppression of my symptoms was totally unsustainable without dependence upon the aforementioned prescription drugs. Neither dietary modification nor exercise was sufficient to mitigate the progression of the disease.

Two months after completion of data collection, I underwent surgical intervention to cure my Ulcerative Colitis. After a year containing three extensive surgeries, I have made a full recovery and am off of the prescription drugs, thus avoiding the harsh side effects that would have gradually compromised many other vital functions.

VI. FURTHER STUDY

A. **Better Optimization Algorithm**

While the Optimization algorithm used in these experiments accurately illuminated the direction and magnitude of the most strongly biased Relevant Factors (such as Prednisone or Blood), these effects often overshadowed the correlative effects of weaker factors. It would be helpful to have an Optimization algorithm that can isolate and discern the exact effects of each individual input's maximum, middle, and minimum activation values.

This method can be designed in the following way. For each input node, and for each specified activation value (1.0, 0.5, and 0.0), all other input nodes within the network are run through the exhaustive permutations of activation values in a way similar to the standard Optimization method described above. MSE against the targeted output values (1.0 or 0.0) are calculated at each pass, as before. However, instead of selecting the lowest MSE configuration as the "optimal" input setting configuration, all MSE scores against the target output are averaged to come up with an overall score for the specific activation value (1.0, 0.5 or 0.0) of the isolated input node. The activation value that achieves the lowest average MSE against the target output values is then marked as the "optimal" setting for that input node to influence the network toward delivering the desired output.

If there is an input representing a very slightly (but reliably) predictive influence toward symptomatic expression (for example, milk), its associated average scores might only reflect a small change in average MSE (for example, target output: 0.0, input:1.0 → avg MSE: 0.454, input 0.5 → avg MSE: 0.439, input 0.0 → avg MSE: 0.411). Though the influence might be small, the direction of the effect would be clear, and would not be overshadowed by much stronger factors, as was the case with the original Optimization algorithm.

There is, however, a major drawback to this method. As it is, the current Optimization algorithm already has a computational complexity of $\mathbf{O(3^n)}$. The new method discussed above would have a computational complexity of $\mathbf{O(n3^n)}$. Considering that the original method, using a fairly typical configuration of 13 inputs had approximately 1.5 million combinations to pass through a final-generation BPNN, and took around 36 hours on a home PC to process, the original Optimization algorithm was already pushing the boundaries of intractability. Multiplying the computational task by **n** inputs for every experiment would have pushed the algorithm well beyond the scope of this thesis.

B. **Eliminating false positives**

In earlier discussion of the problem of closing the gap between reliable correlation and provable causation, it was pointed out that the BPNNs only determine which factors in the given data set frequently preceded particular outputs at prediction events. In an experiment with a closed data set, where the problem space was observed for a prescribed duration, and inputs were varied within the observation period without reference to earlier parts of the data set, the discovery of reliable correlation is the best that can be hoped for.

However, in an experiment with open-ended data collection can truly be used as a tool to determine causation. To do this, the subject must make reference to early interpretations of the influence of relevant factors, and adjust behavior to intentionally attempt to disprove those interpretations.

One clear example of how this might be done arises from the interpretation exercise as an expressive factor for UC in the current data set. Literature on the immunosuppressive effects of intense exercise would suggest that this interpretation is incorrect, though it is clear that the

data set, combined with the Optimizer algorithm suggests otherwise. However, further inspection of the data shows that exercise was only undertaken during a long period of UC flare. It then makes sense that an optimized network would show a correlation between exercise and UC expression.

These spurious correlations might easily lead to a misinterpretation of certain moderate-influence factors as having a bias that is opposite of their actual influence. However, on an open data set, the subject can undertake actions that intend to counteract this effect. This is achieved by partaking in behavior that is opposite of interpreted influence. This action should be performed on all Relevant Factors to counteract coincidental correlations and make the data set more robust.

This will have two potential outcomes. First, the action may reveal the true potency of the Relevant Factor, such as in the case of Prednisone, where taking the medication during a period of UC flare will induce remission, and strengthen the correlation in the continuing data set. Alternately, the action may reveal the true coincidence of the Relevant Factor, such as in the case of exercise, where engaging in yoga, strength training or cardio training during a period of UC remission will elicit no increase in disease symptoms, and the strength of the correlation (and hence false positive) will decrease in the continuing data set.

By this method of intentional, data driven investigation, an understanding of actual causality can be pursued, and we no longer have to rely on educated guesses based on simple correlation.

C. **Better Data Classification**

One confounding issue around dietary factors was that so many food and beverage items likely contain many chemical compounds that cannot be captured simply by naming the ingredients. In these experiments, obviously similar foods, such as green or red bell peppers, were categorized separately because they have two distinct flavors, which indicate distinctly different chemical compositions, despite being from the same vegetable family. In the many cases like this, it is clear that the items have many chemicals in common, as well as some that differ. Neither grouping them together nor keeping them separate during data collection and processing seem like the correct course of action.

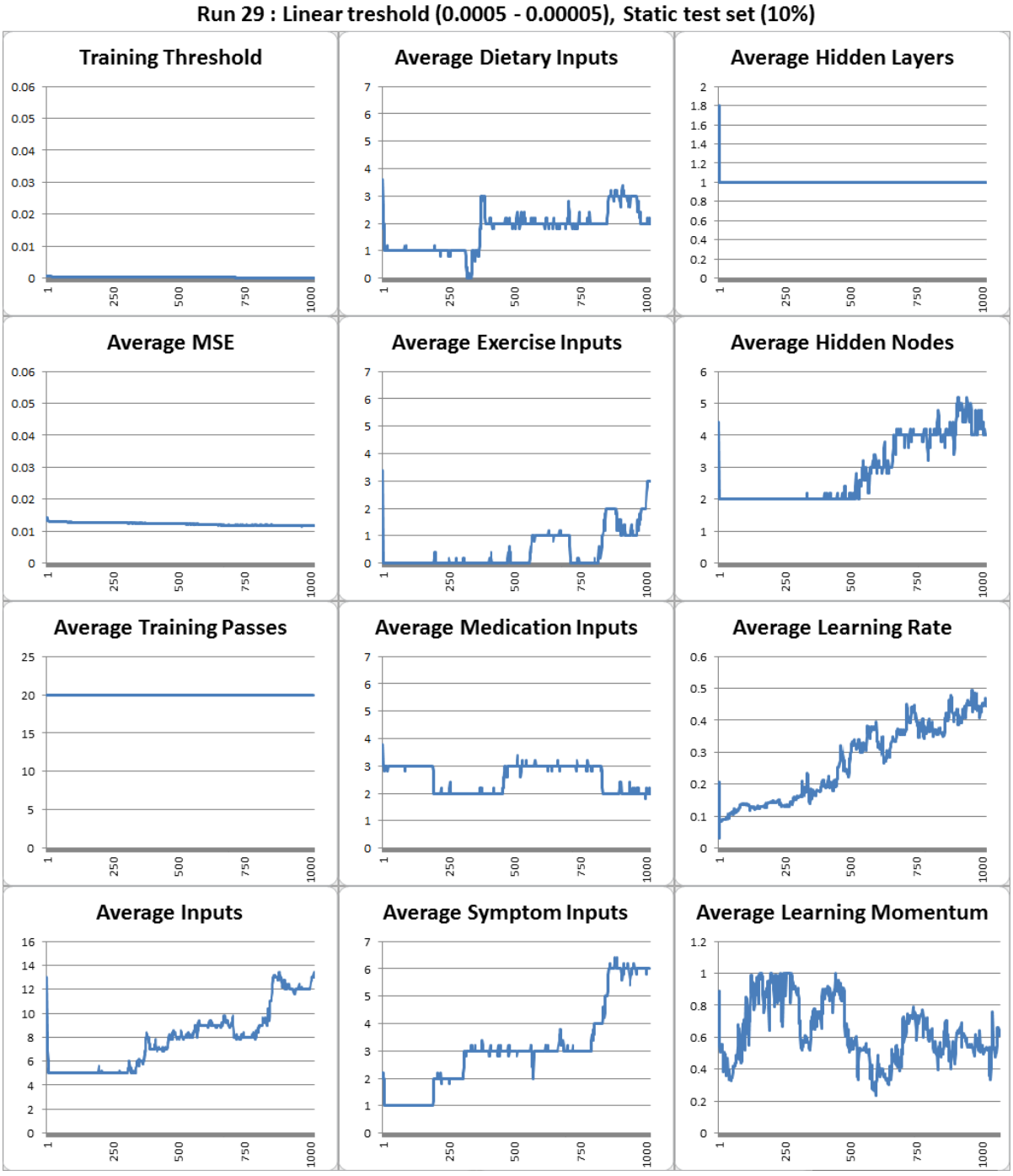
In cases like these, one reasonable solution would be to split every dietary intake ingredient into its active component chemicals. However, this process would require access to an accurate and exhaustive database of said information, which may or may not exist, and may or may not be proprietary if it does. As such, such an undertaking was well beyond the scope of these experiments. If it is indeed possible to attain the information, collating the appropriate ingredient data would be a large undertaking in and of itself. The value of this specificity is unknown, but would certainly be better than simple classification by name.

VII. APPENDICES

APPENDIX A

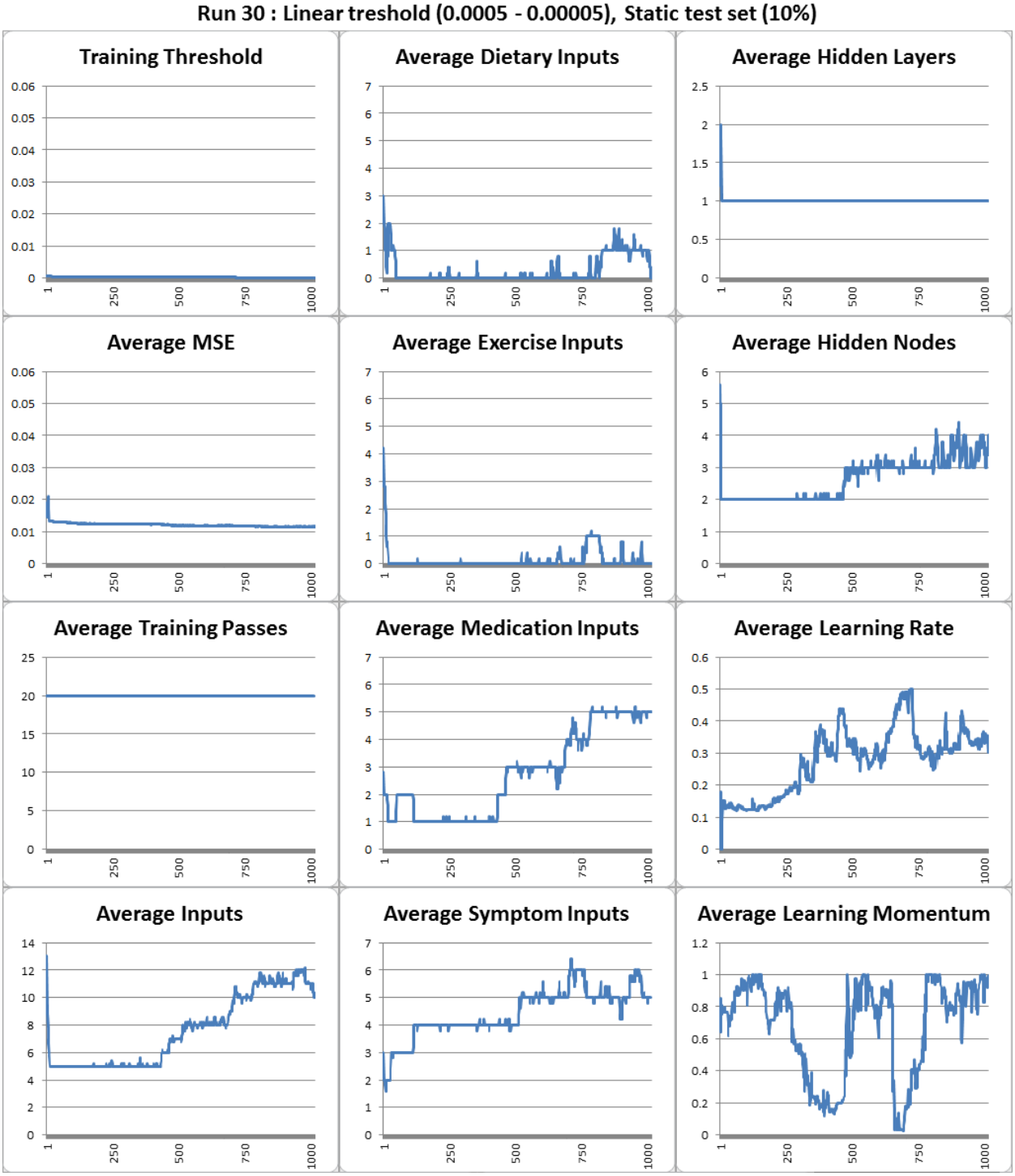
A. **BPNN and EP Performance**

Fig. IX.A.1: Run 29



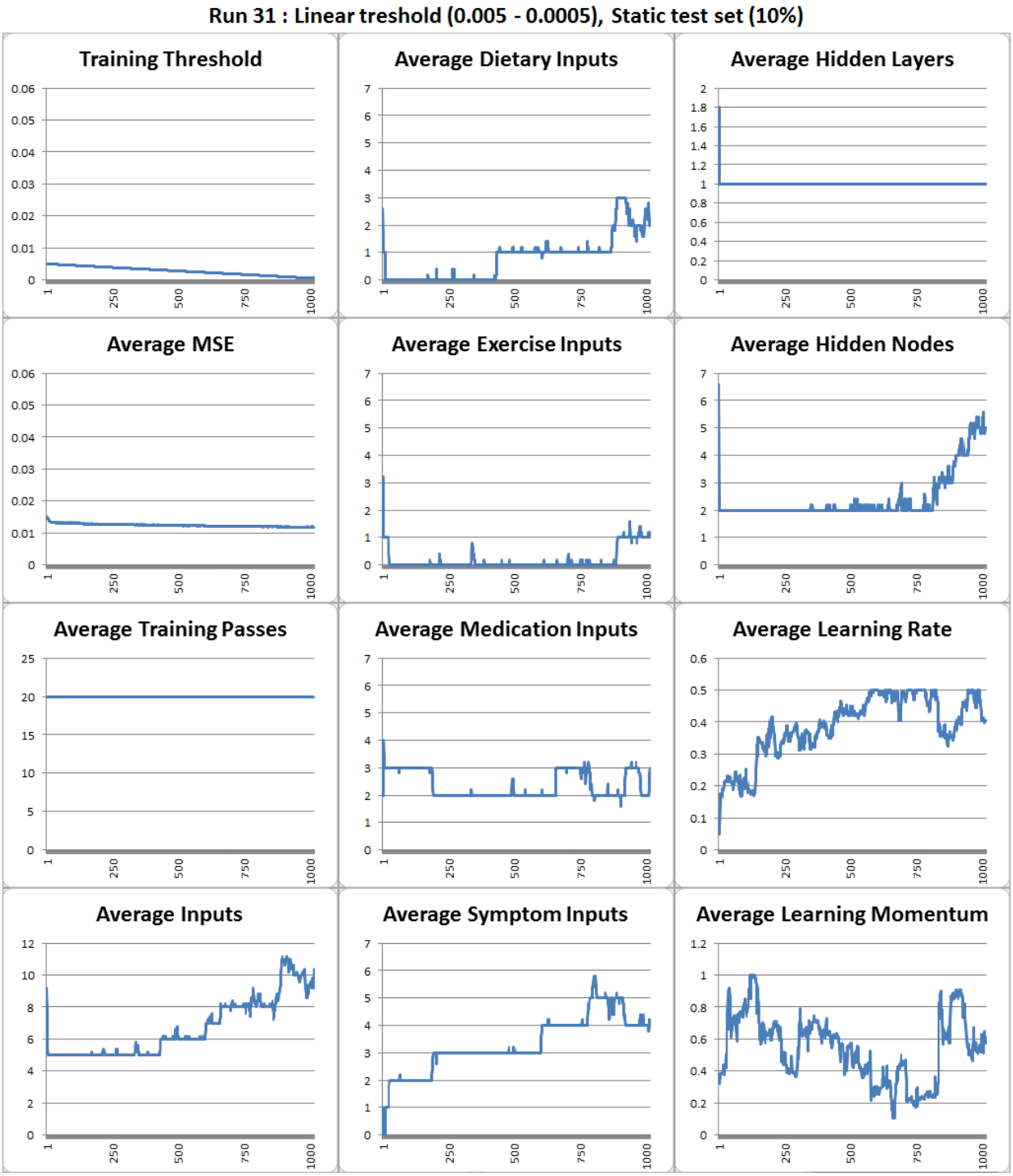
APPENDIX A (continued)

Fig. IX.A.2: Run 30



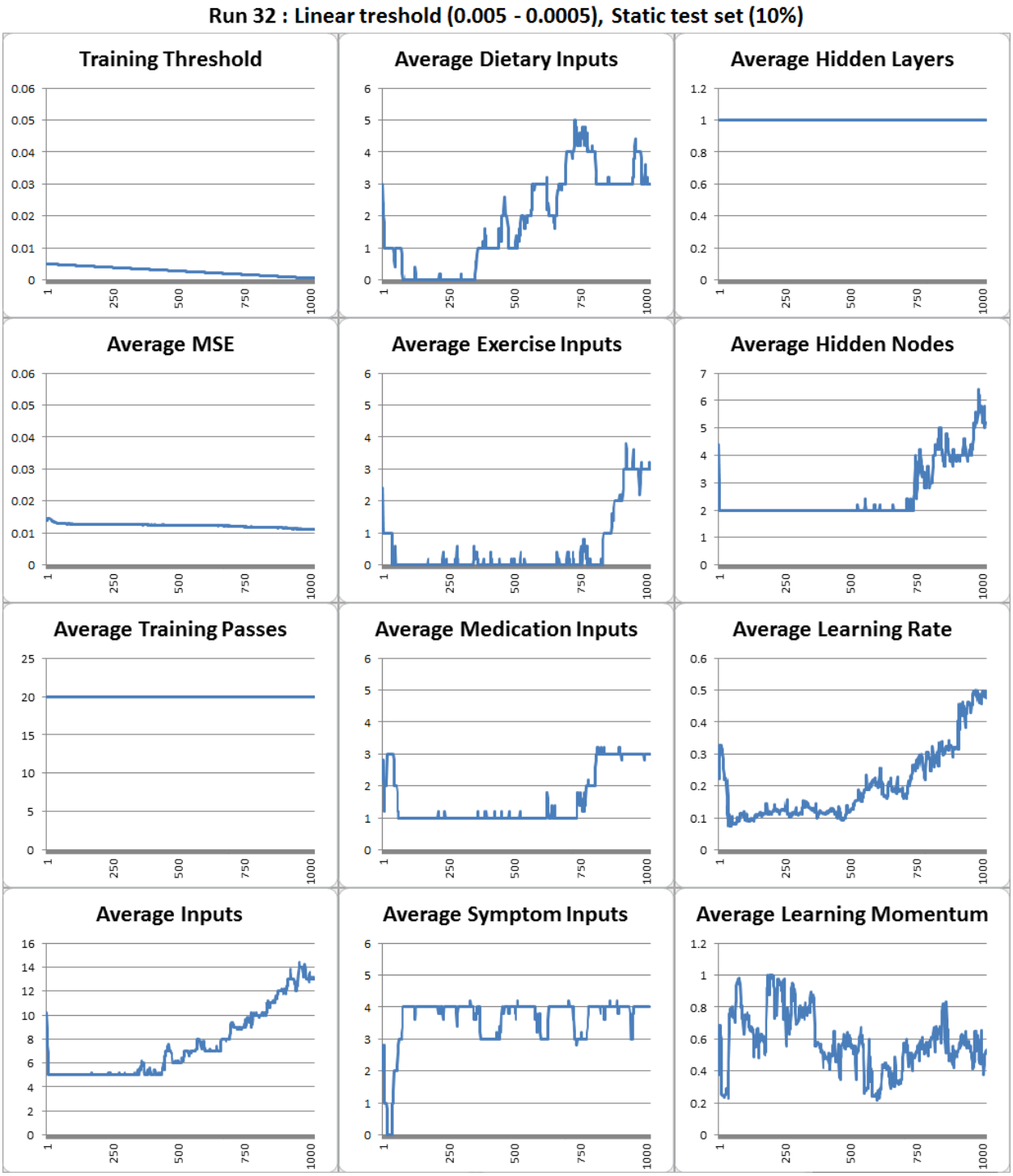
APPENDIX A (continued)

Fig. IX.A.3: Run 31



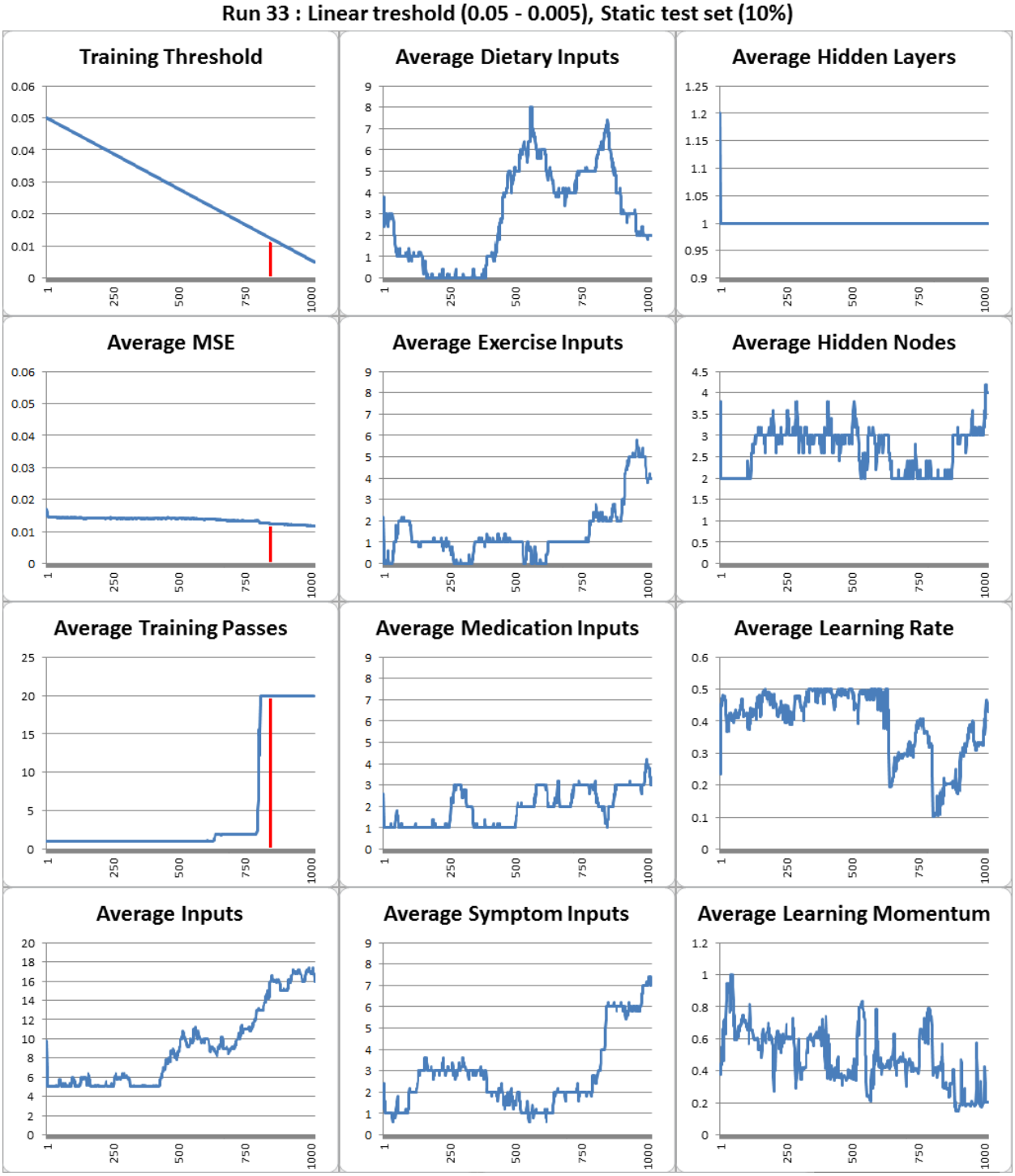
APPENDIX A (continued)

Fig. IX.A.4: Run 32:



APPENDIX A (continued)

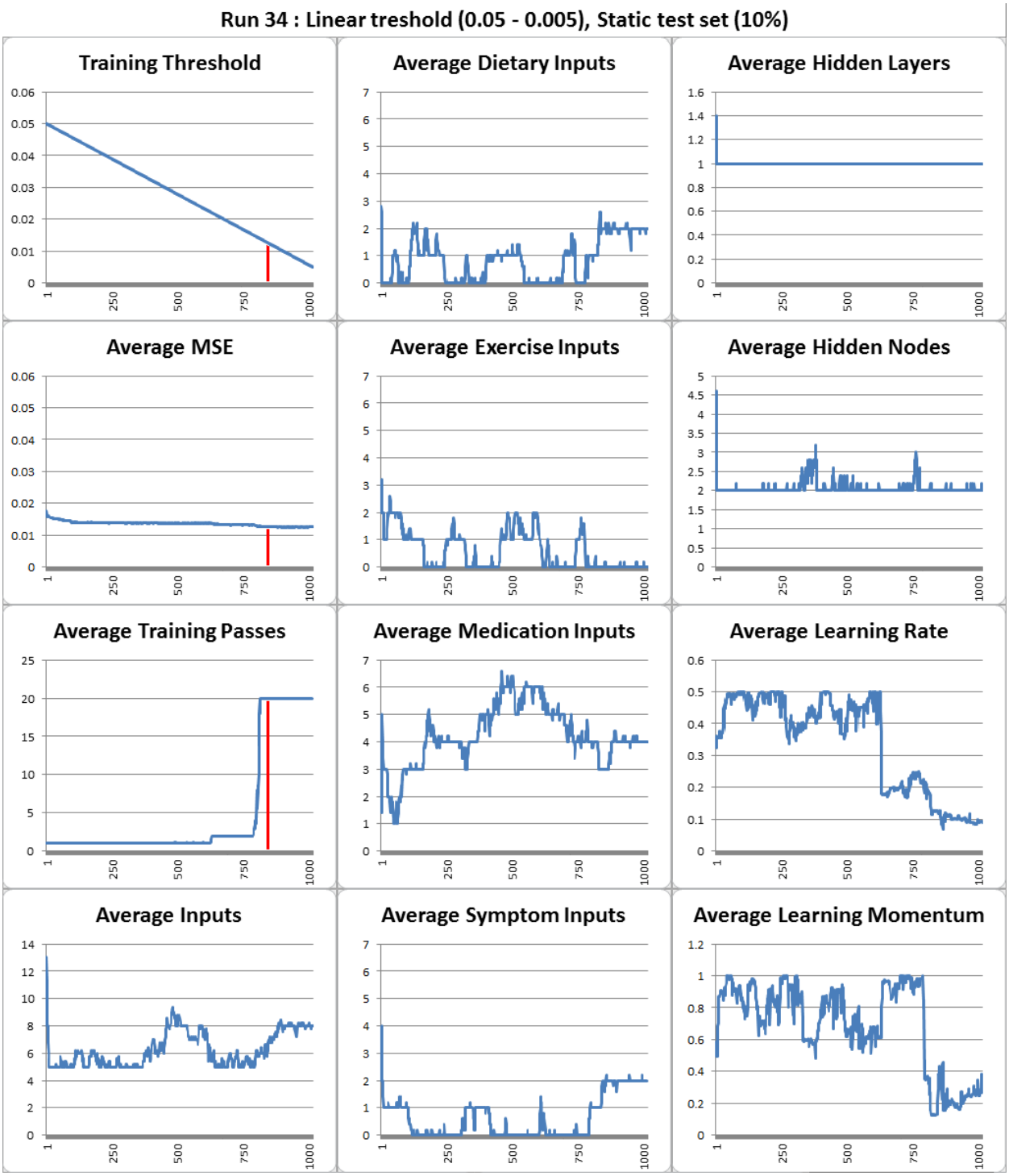
Fig. IX.A.5: Run 33



Note: Red line denotes generation at which training threshold crosses average MSE.

APPENDIX A (continued)

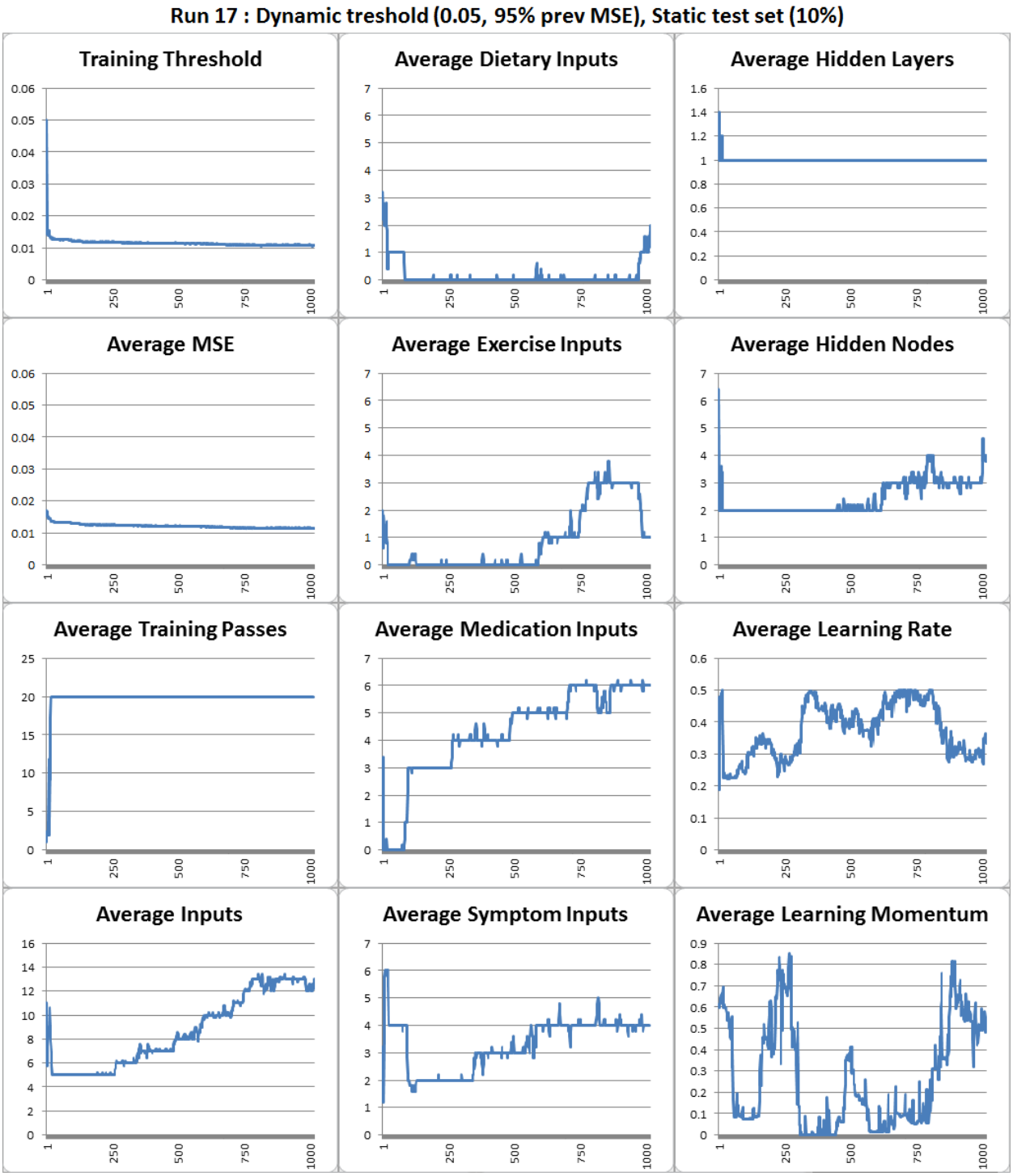
Fig. IX.A.6: Run 34



Note: Red line denotes generation at which training threshold crosses average MSE.

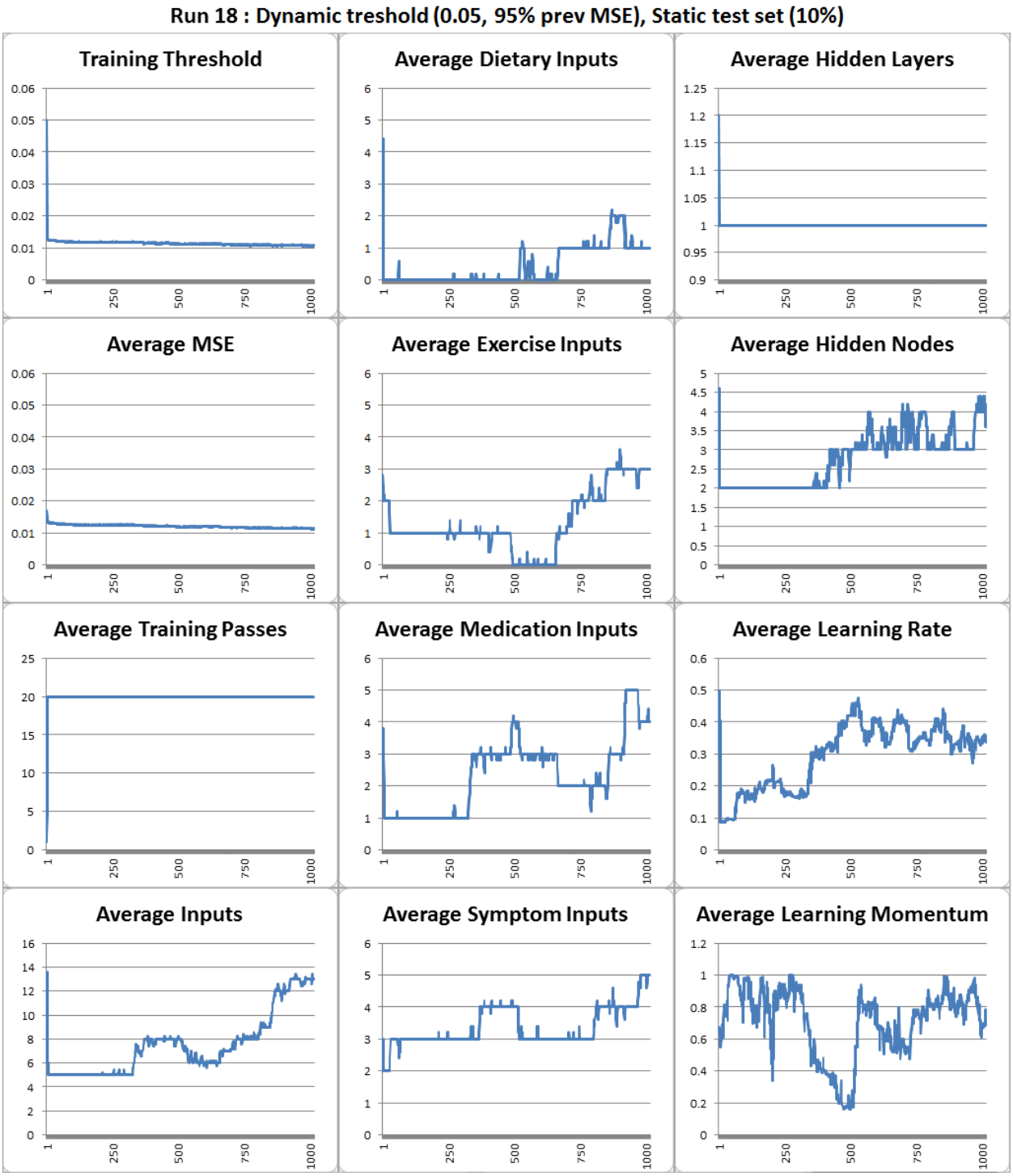
APPENDIX A (continued)

Fig. IX.A.7: Run 17



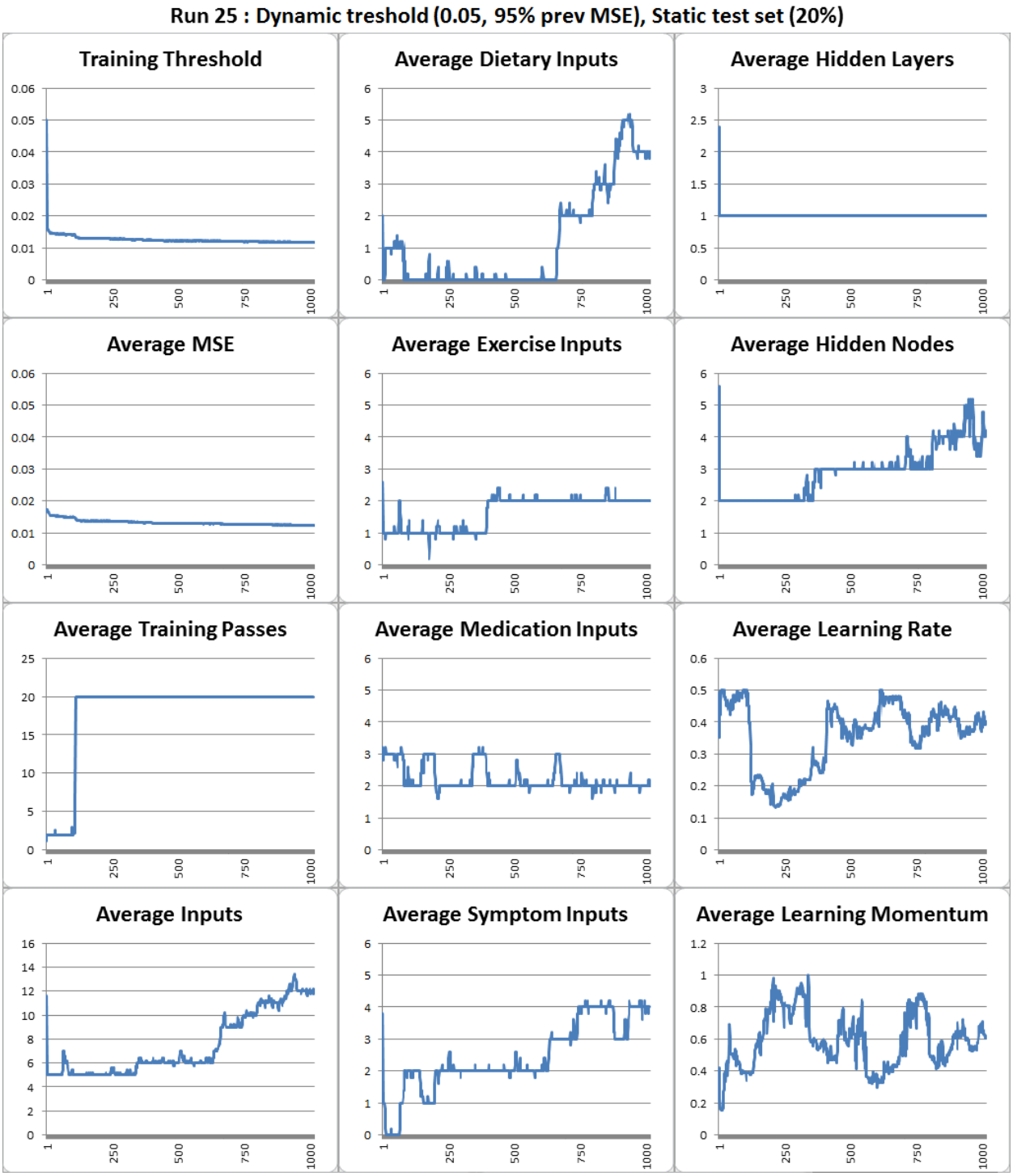
APPENDIX A (continued)

Fig. IX.A.8: Run 18



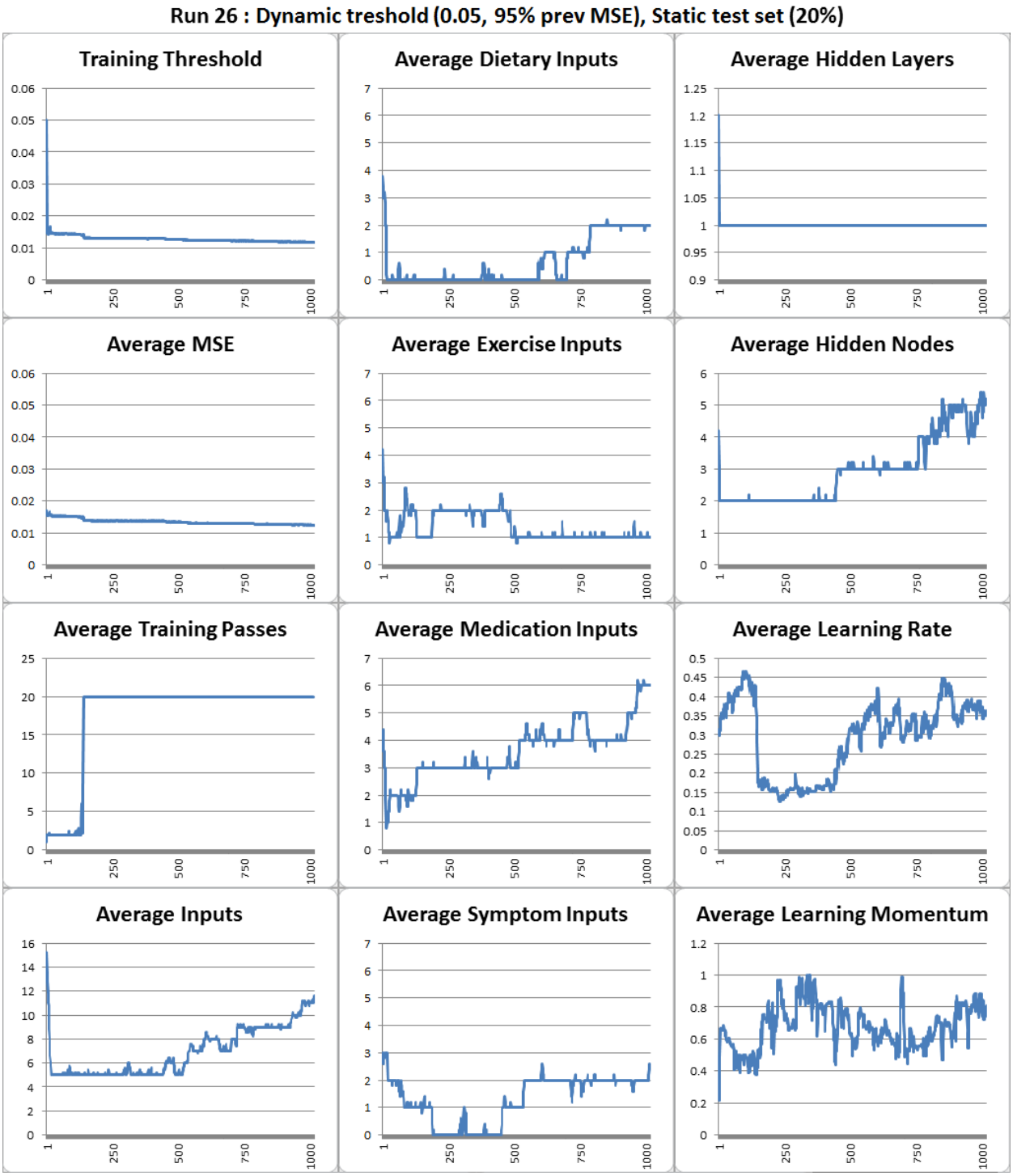
APPENDIX A (continued)

Fig. IX.A.9: Run 25



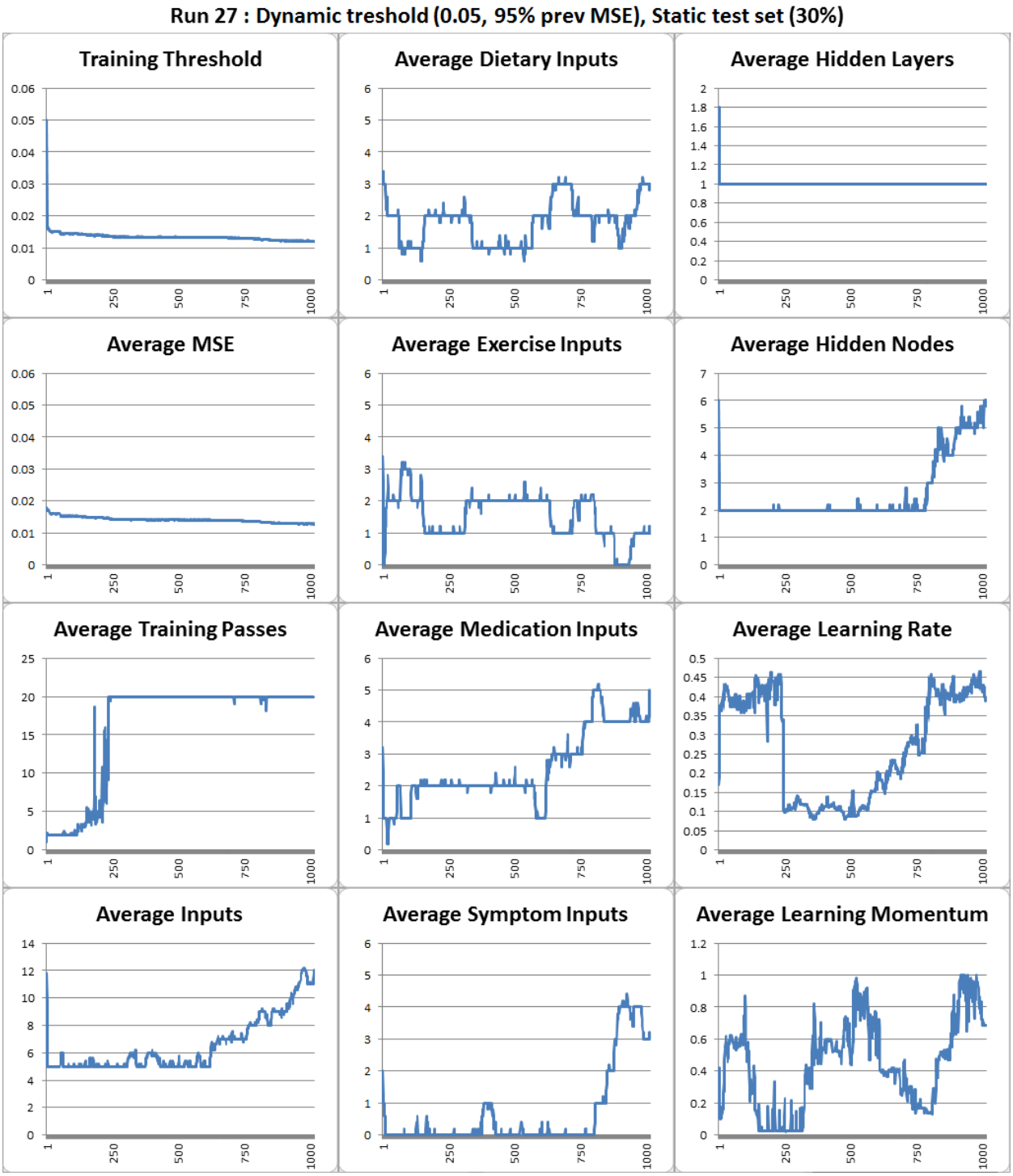
APPENDIX A (continued)

Fig. IX.A.10: Run 26



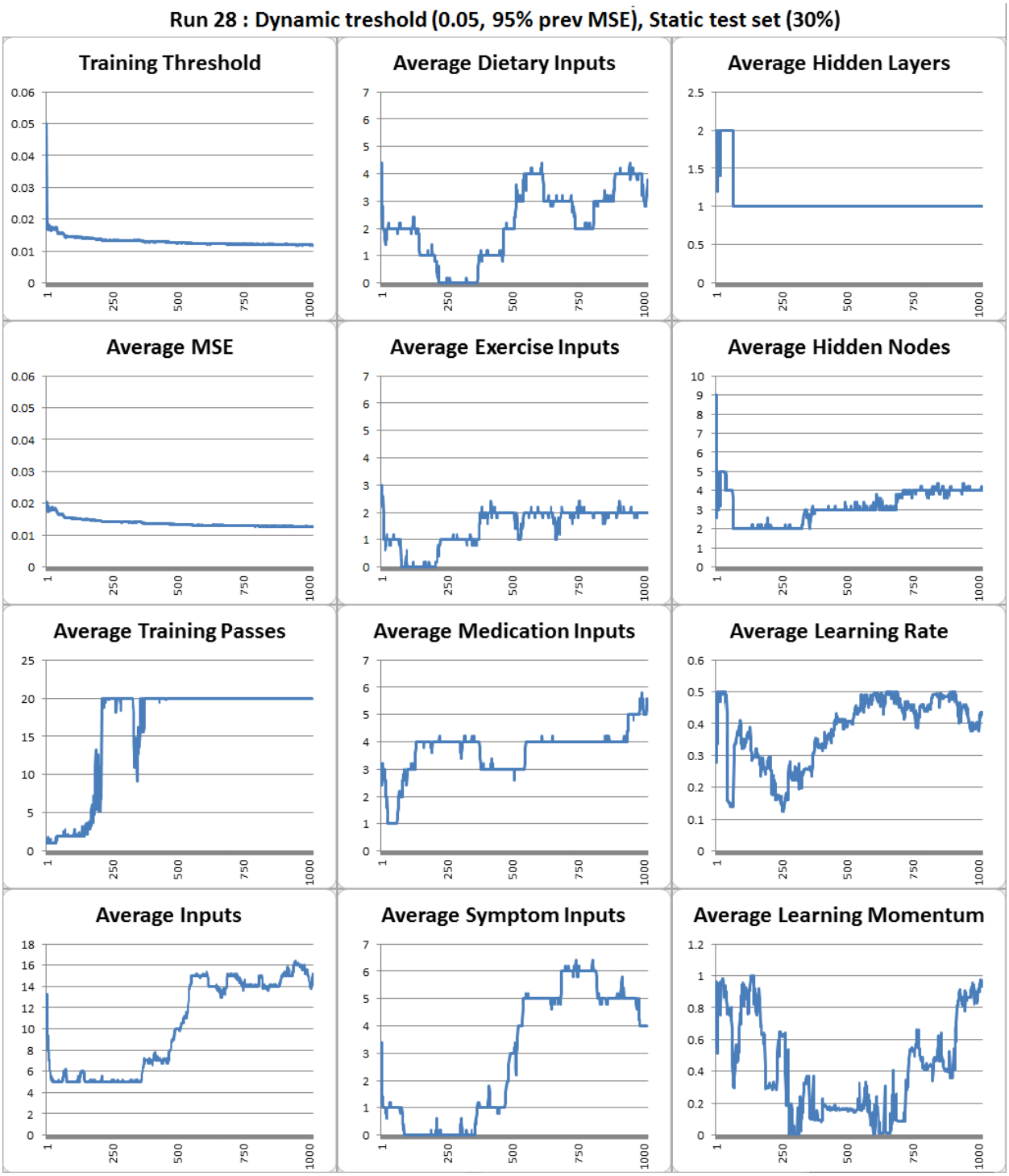
APPENDIX A (continued)

Fig. IX.A.11: Run 27



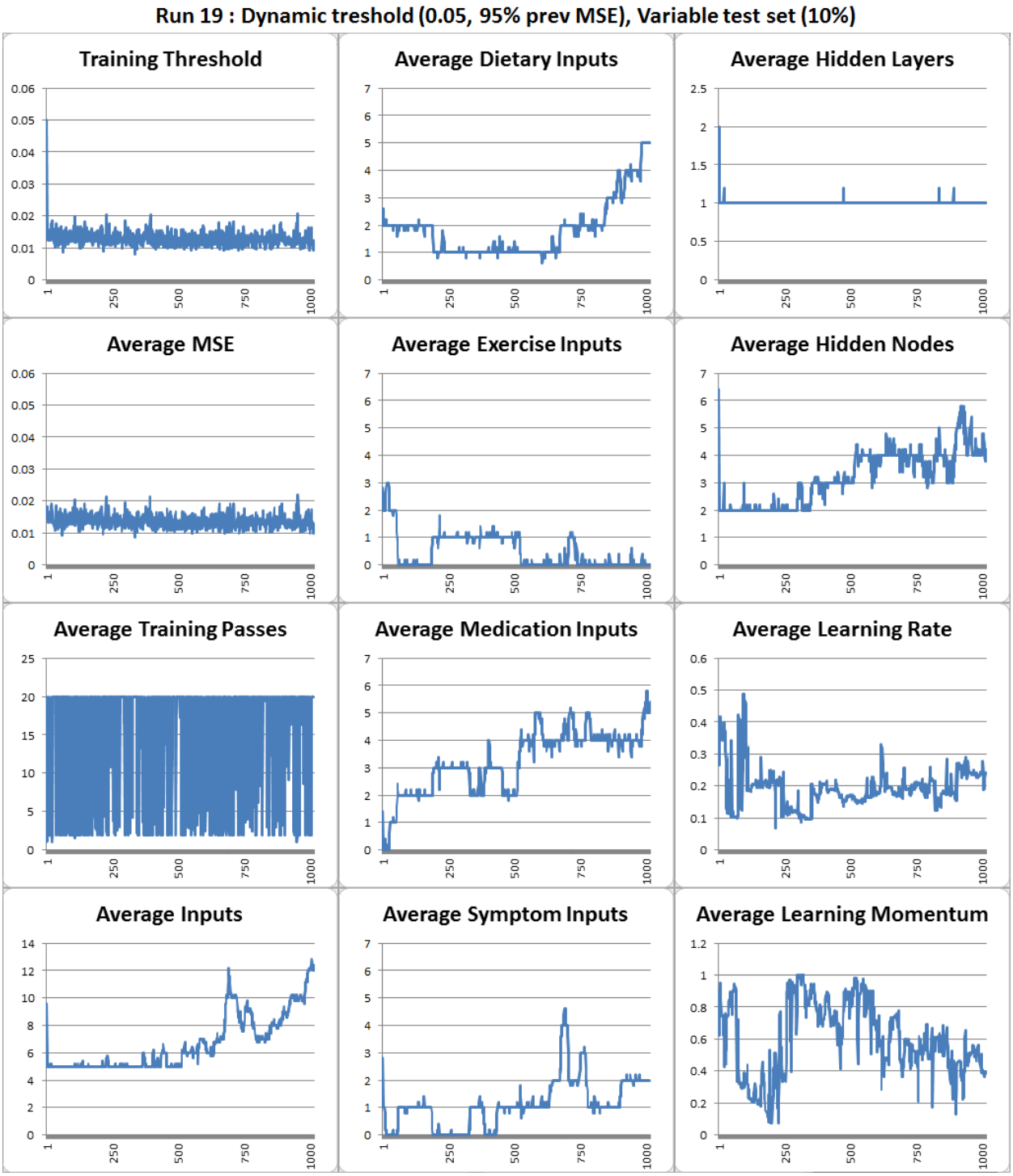
APPENDIX A (continued)

Fig. IX.A.12: Run 28



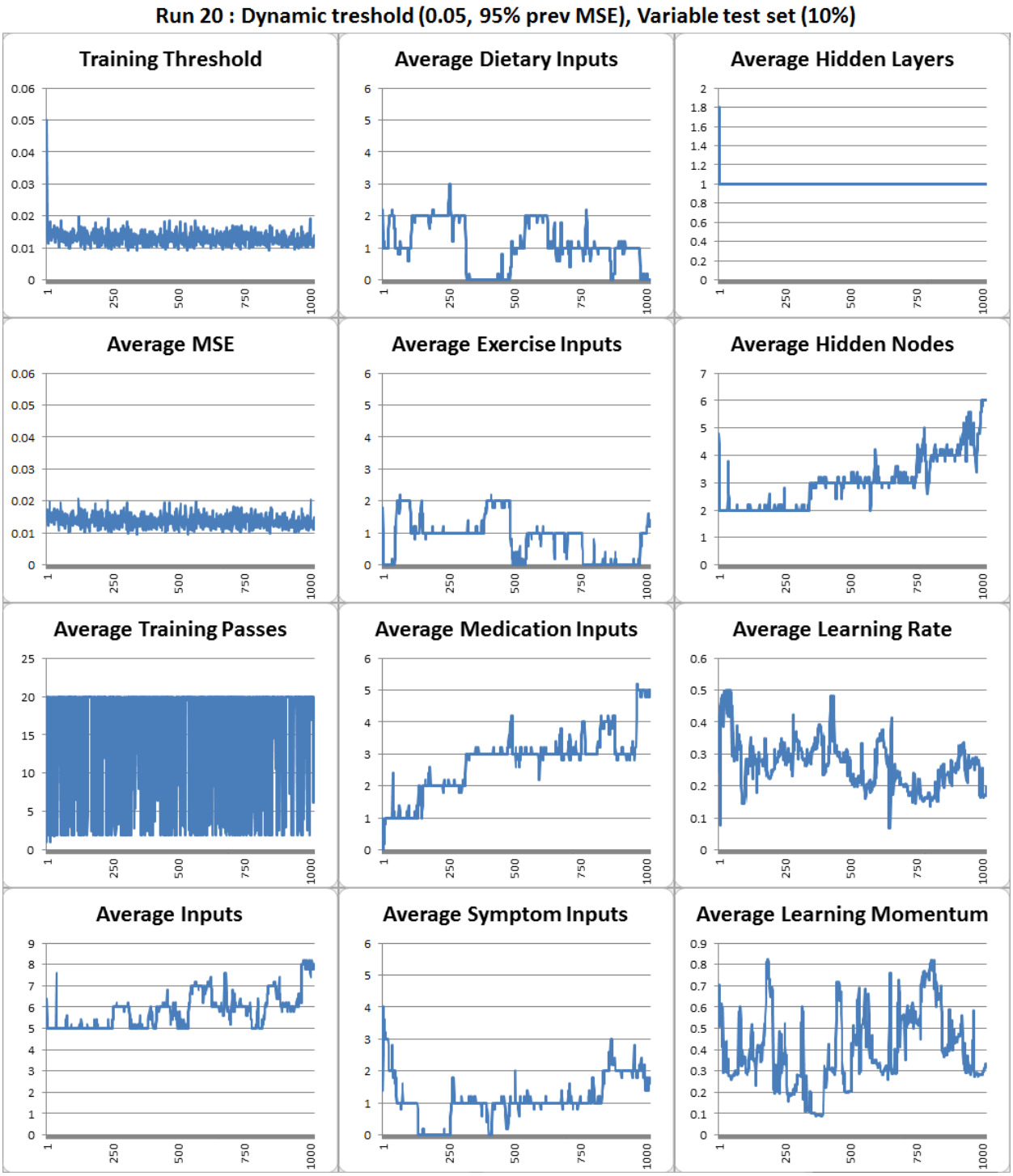
APPENDIX A (continued)

Fig. IX.A.13: Run 19



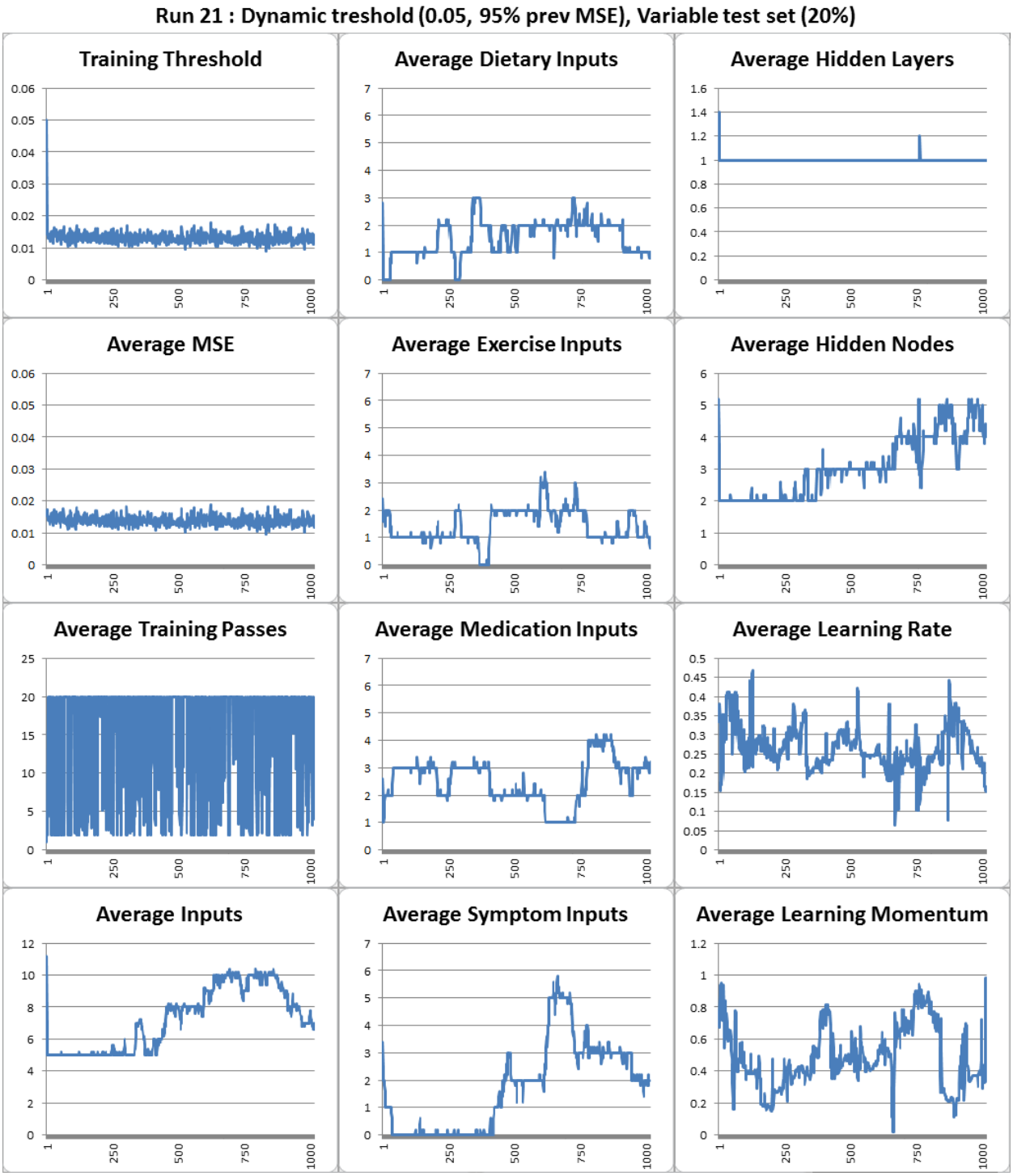
APPENDIX A (continued)

Fig. IX.A.14: Run 20



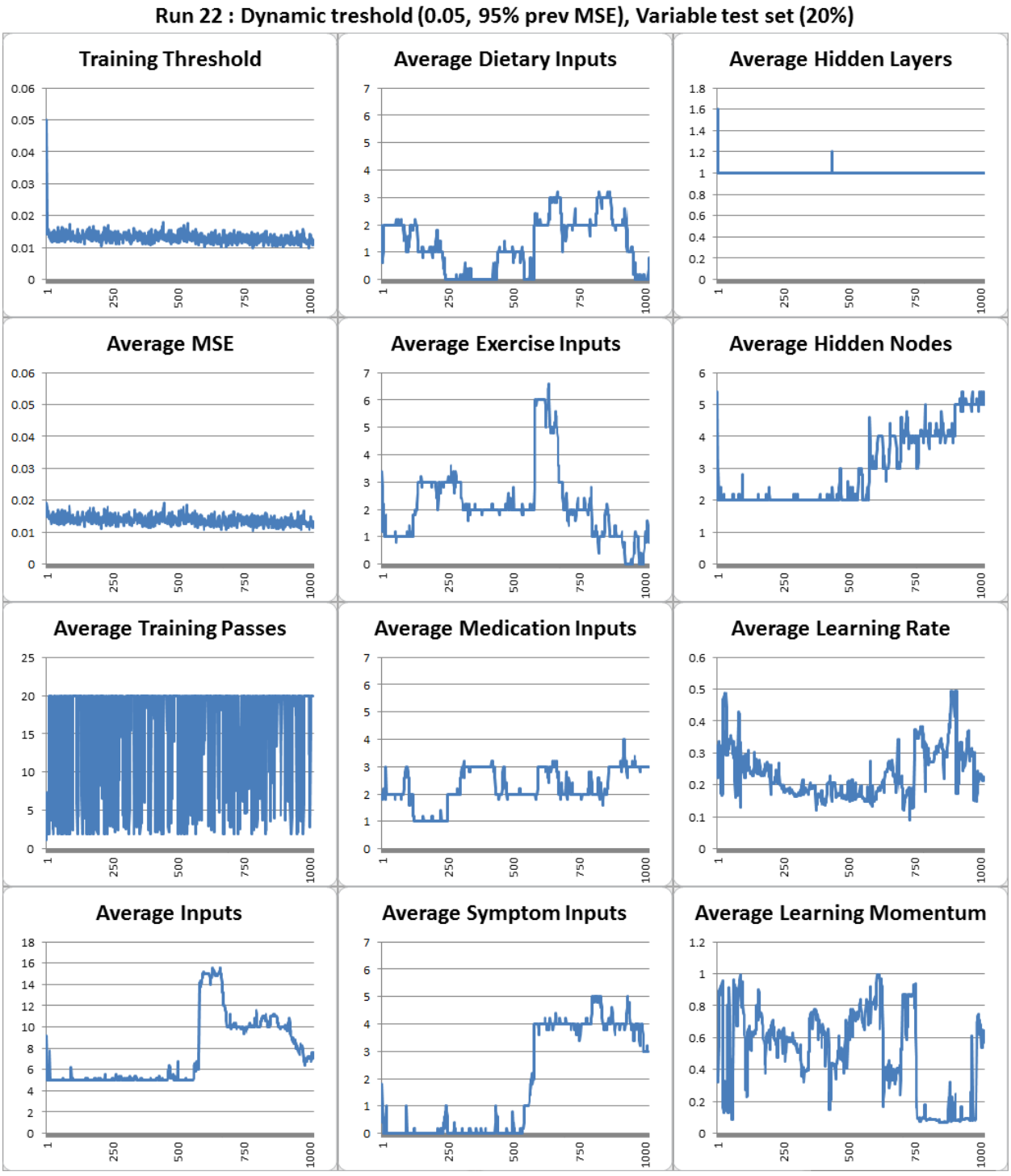
APPENDIX A (continued)

Fig. IX.A.15: Run 21



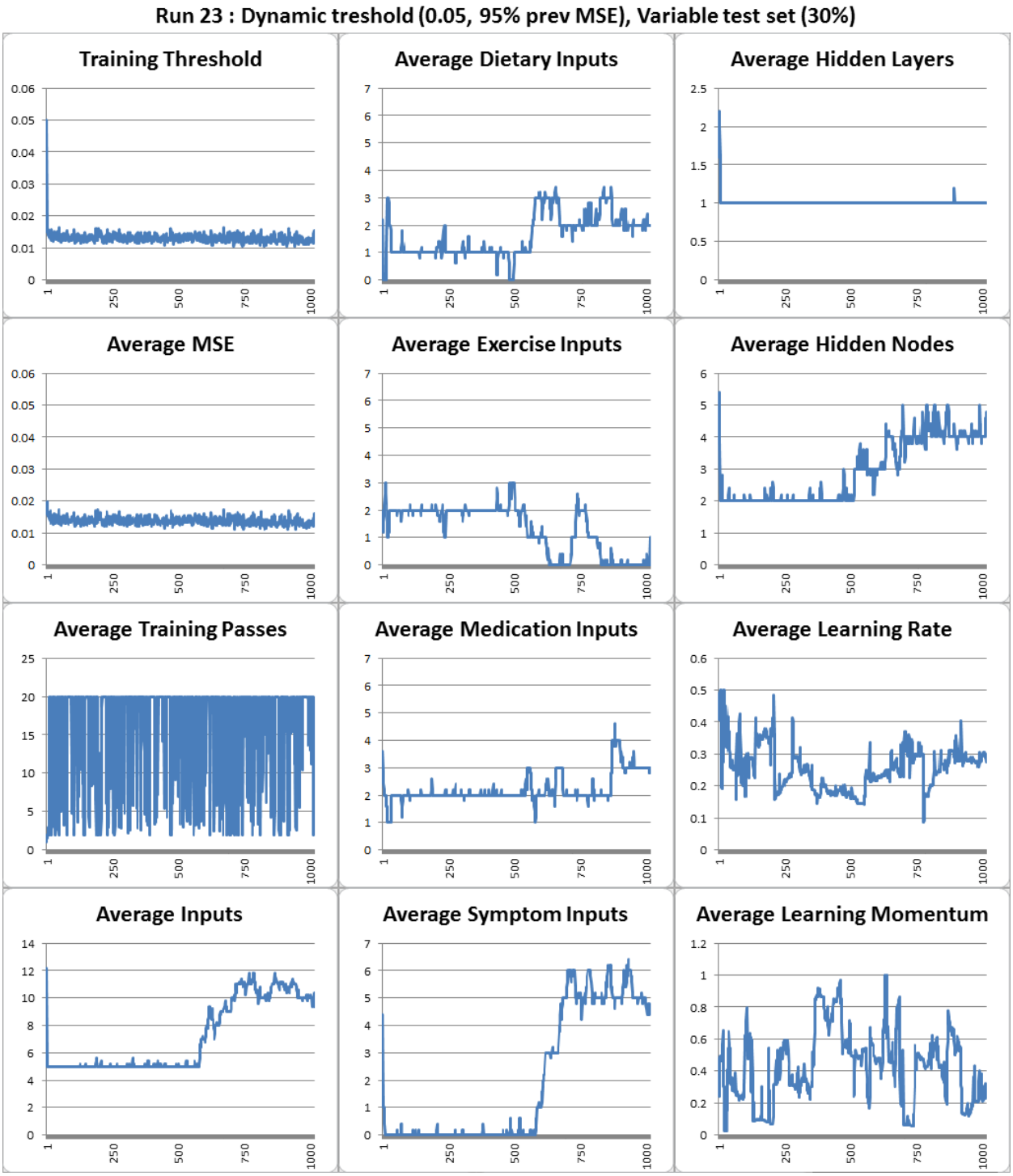
APPENDIX A (continued)

Fig. IX.A.16: Run 22



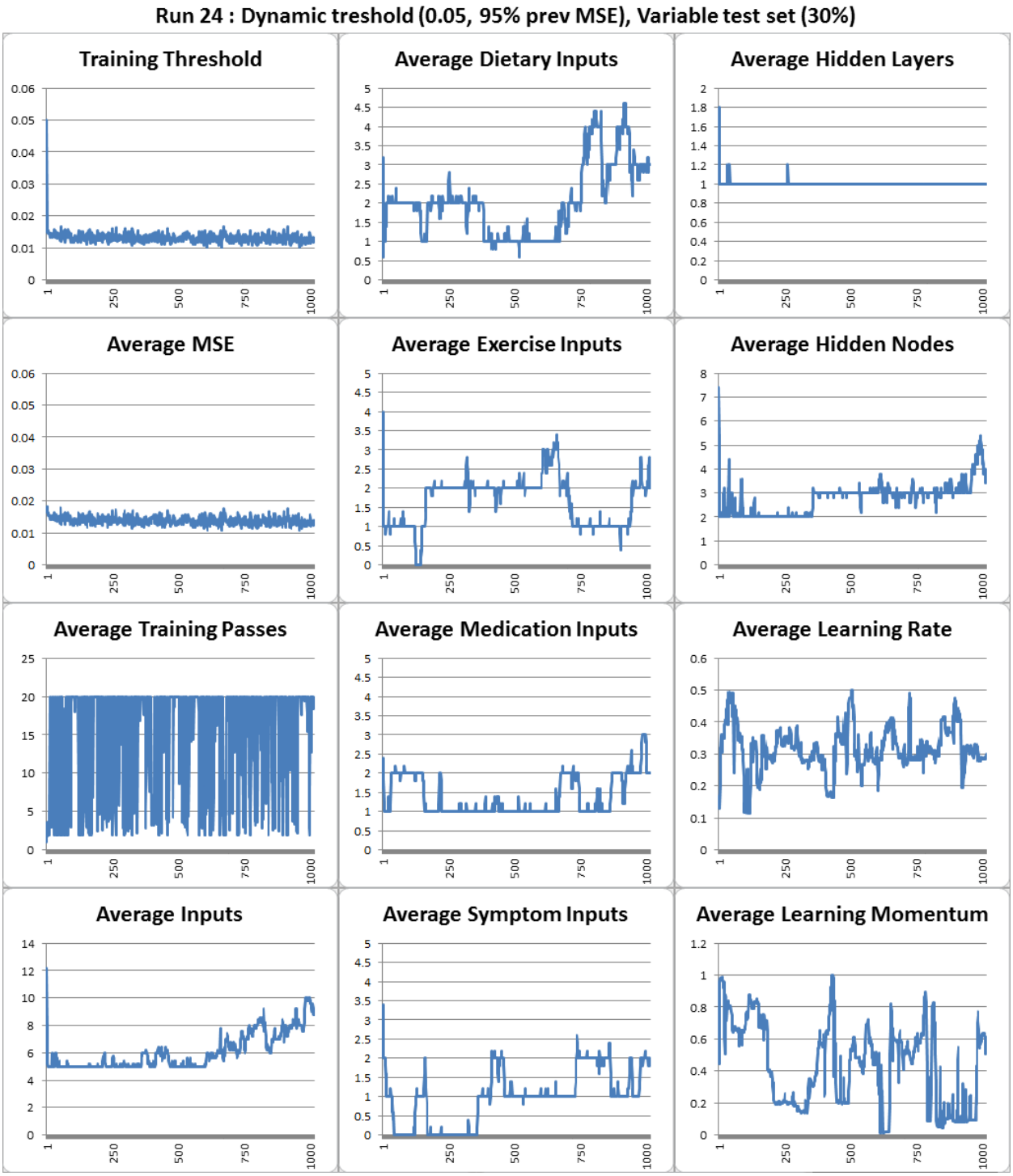
APPENDIX A (continued)

Fig. IX.A.17: Run 23



APPENDIX A (continued)

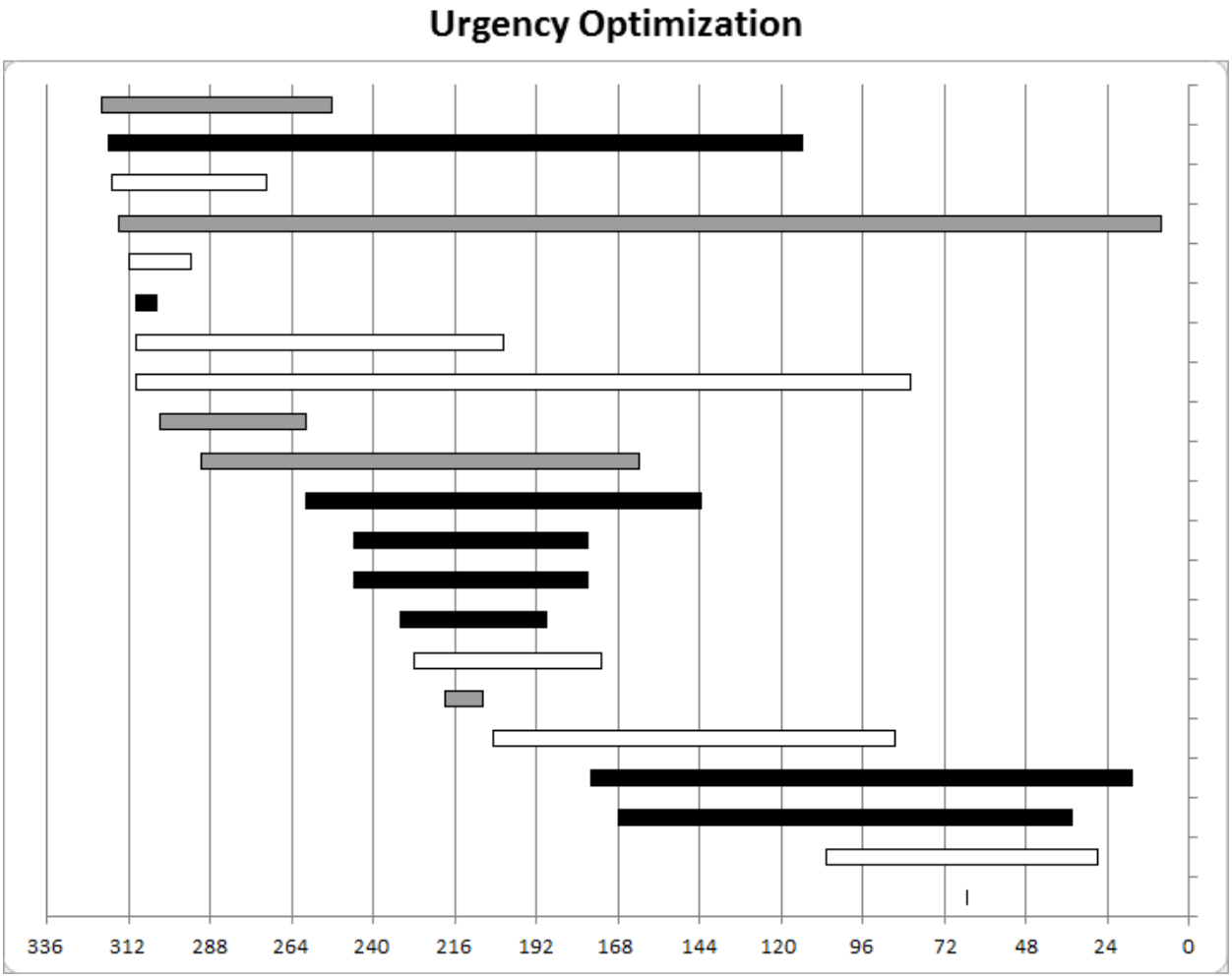
Fig. IX.A.18: Run 24



APPENDIX B

B. **Input-Specific Optimizer Results**

Fig. IX.B.1: Symptom – Urgency



APPENDIX B (continued)

Fig. IX.B.2: Symptom – Blood

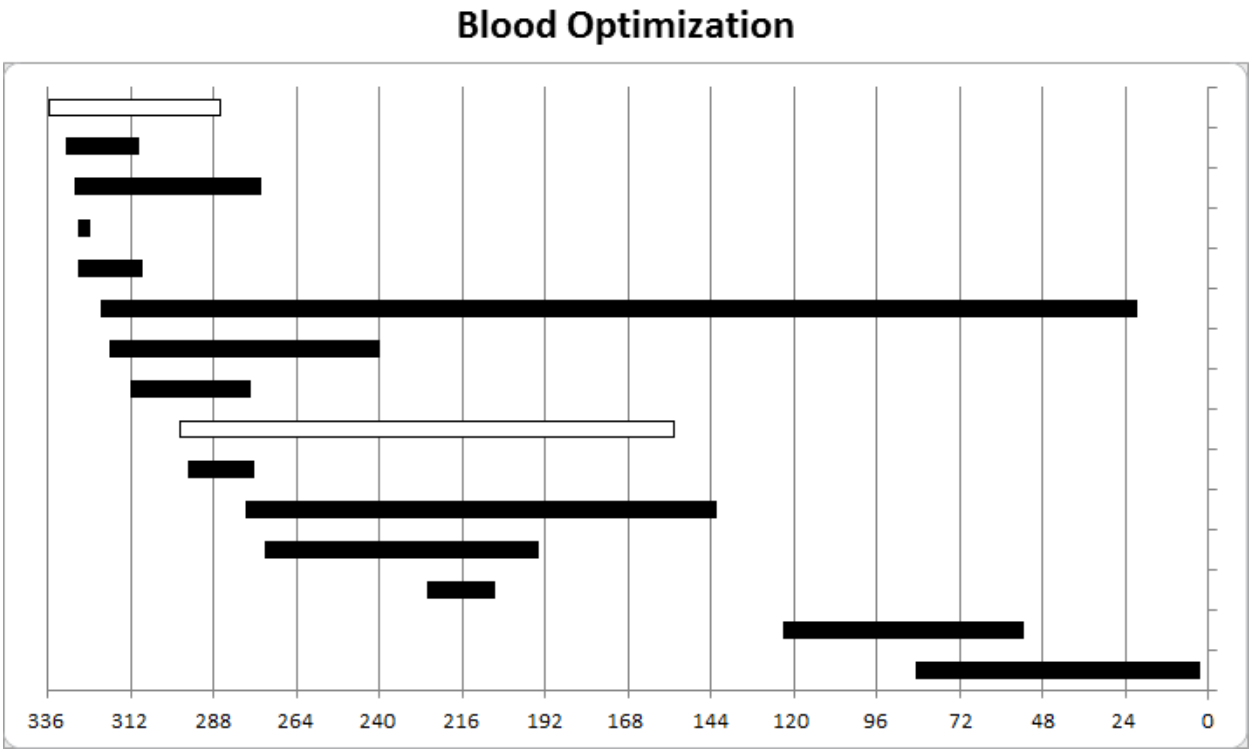
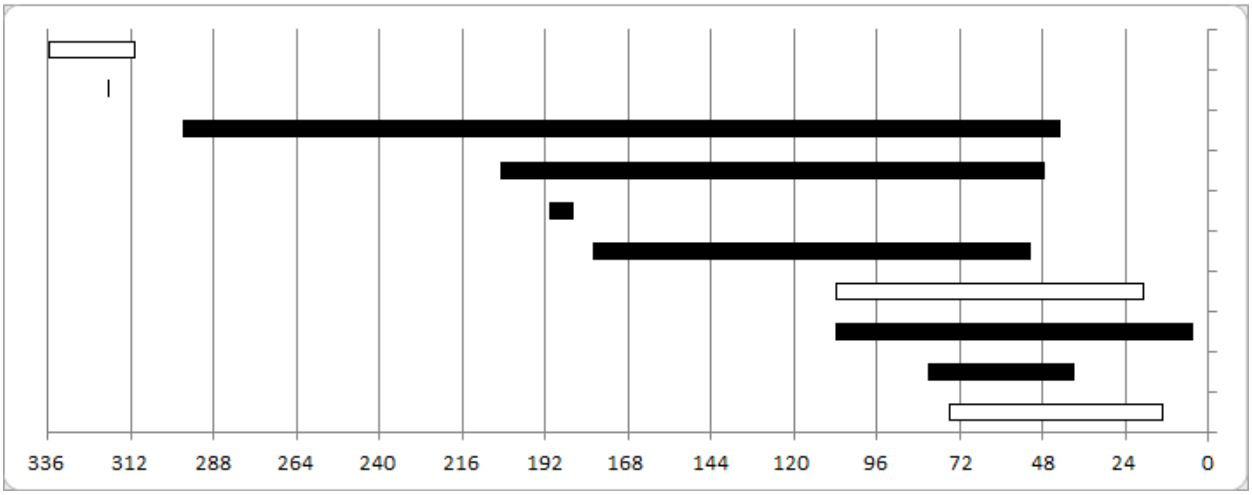


Fig. IX.B.3: Symptom – Diarrhea



APPENDIX B (continued)

Fig. IX.B.4: Symptom – Pus

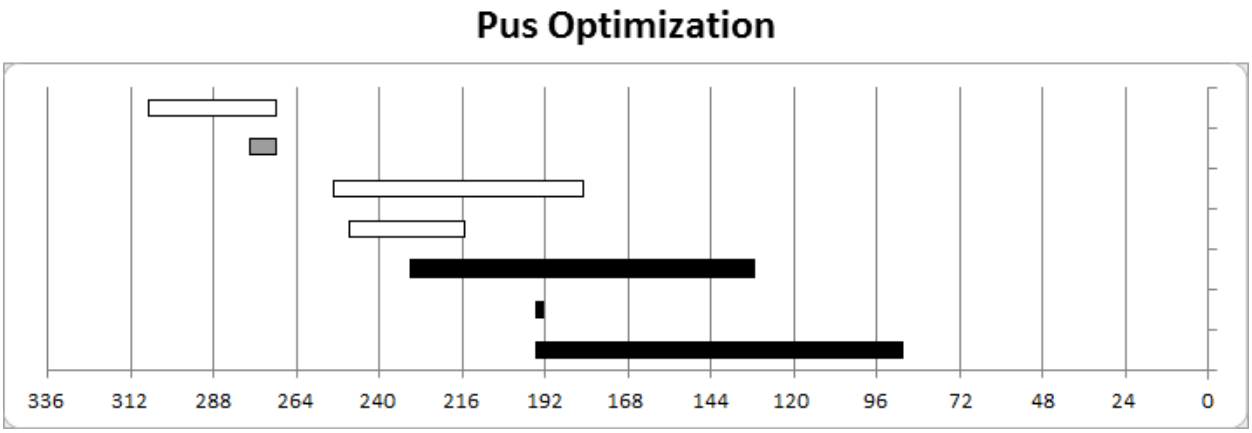


Fig. IX.B.5: Symptom – Mucus

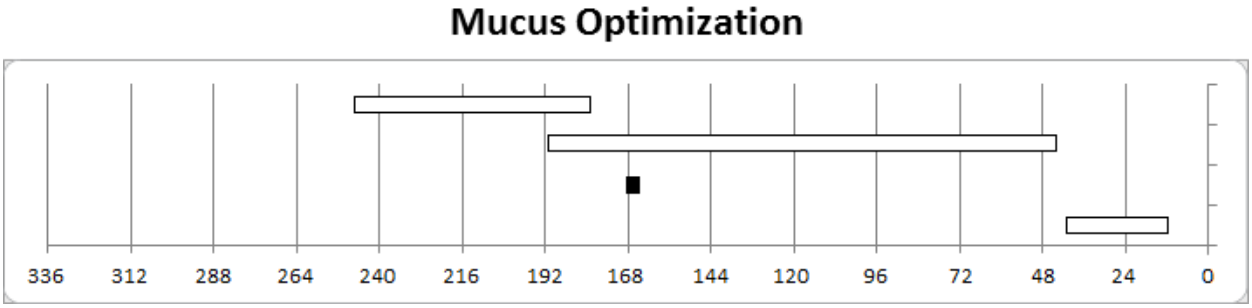
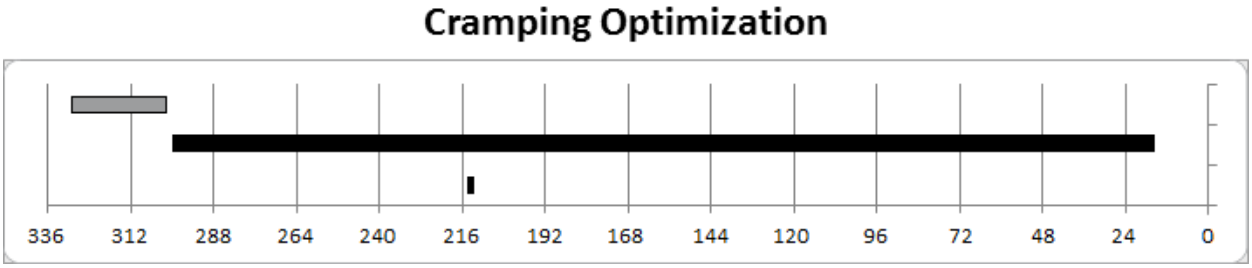


Fig. IX.B.6: Symptom – Abdominal Cramping



APPENDIX B (continued)

Fig. IX.B.7: Symptom – Bowel Movement

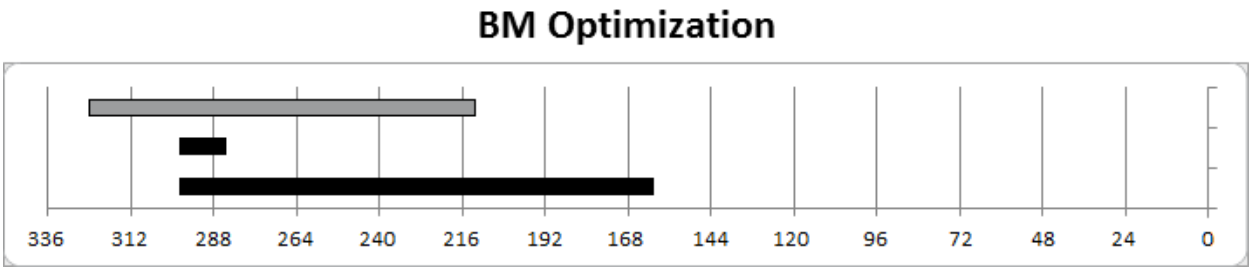
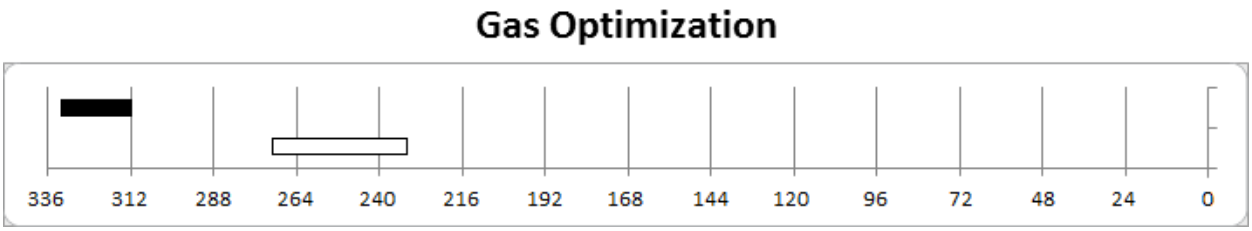
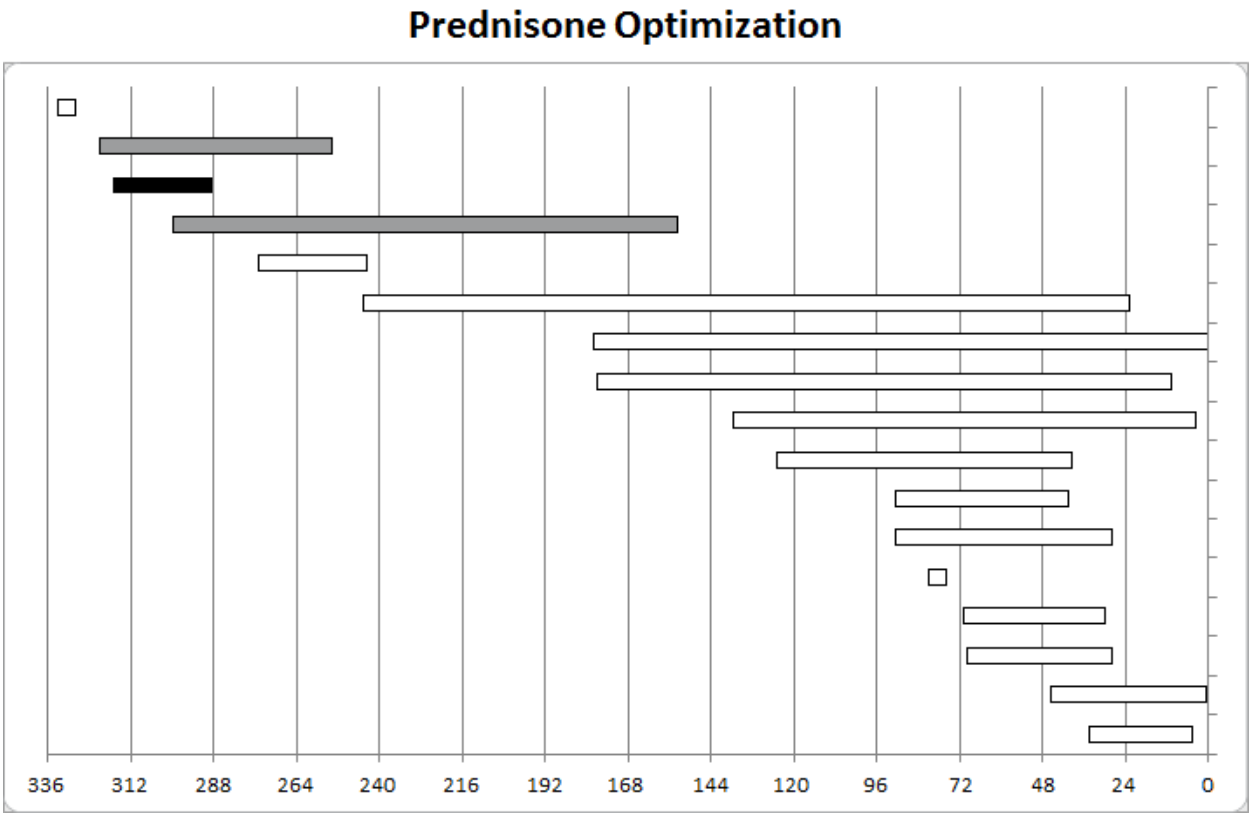


Fig. IX.B.8: Symptom – Gas and Bloating



APPENDIX B (continued)

Fig. IX.B.9: Non-Symptom – Prednisone



APPENDIX B (continued)

Fig. IX.B.10: Non-Symptom –Transdermal Nicotine

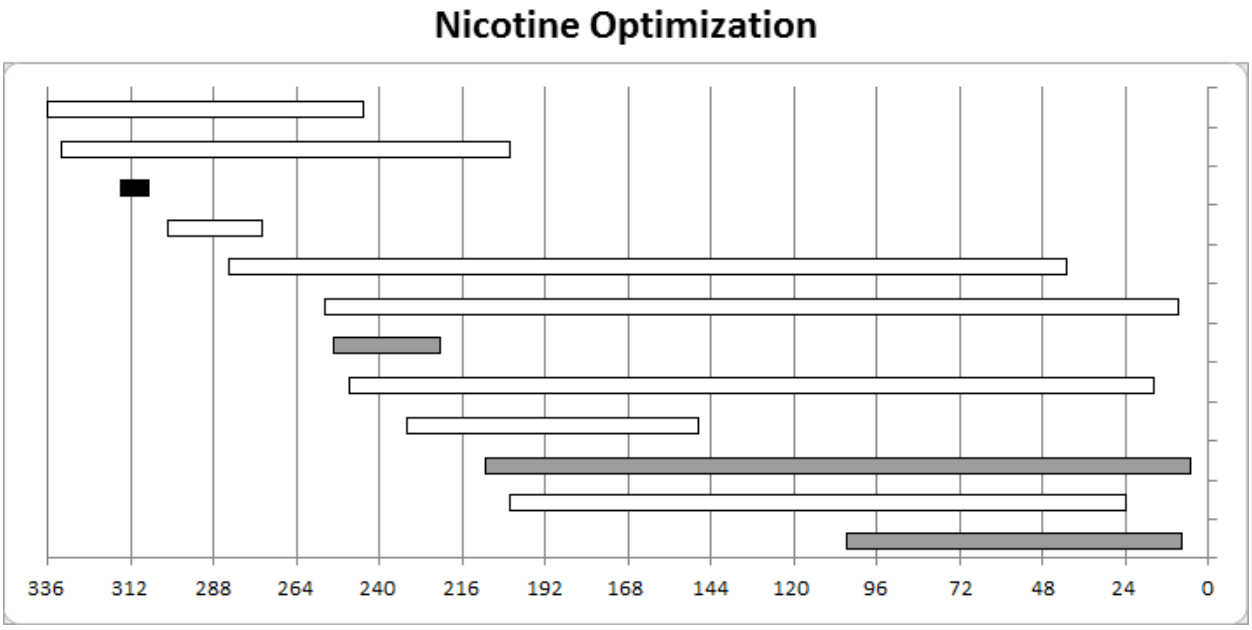
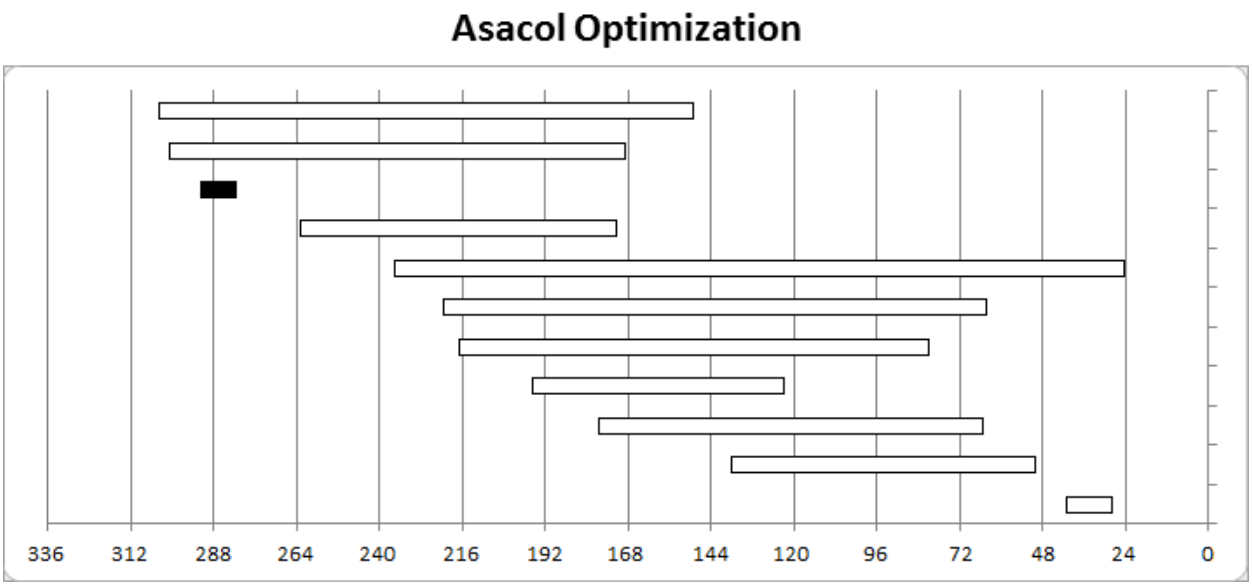


Fig. IX.B.11: Non-Symptom – Asacol



APPENDIX B (continued)

Fig. IX.B.12: Non-Symptom – Yoga

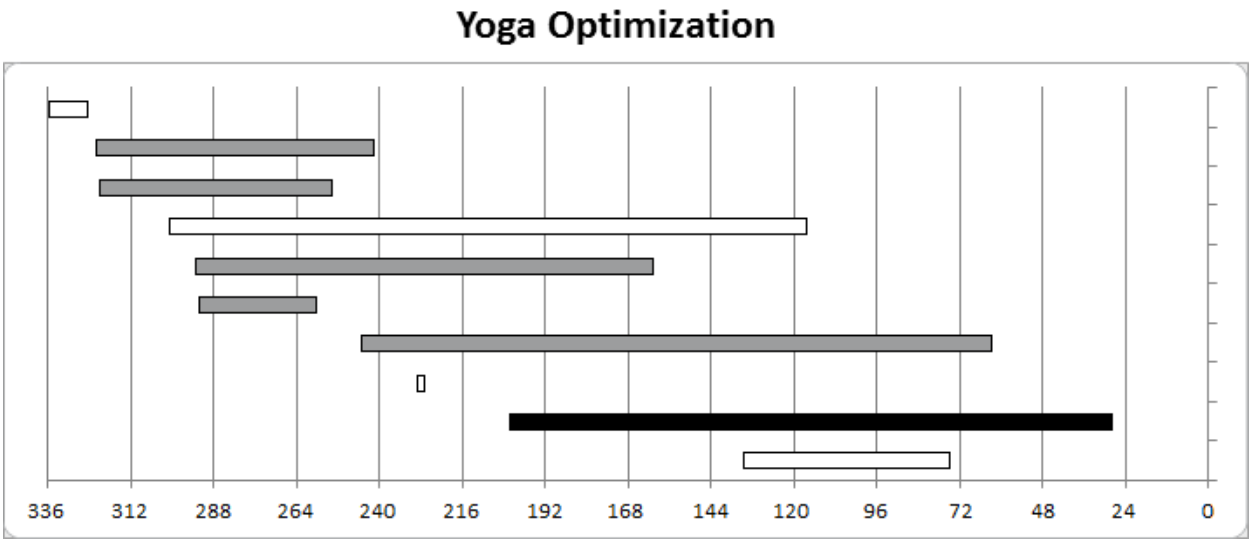
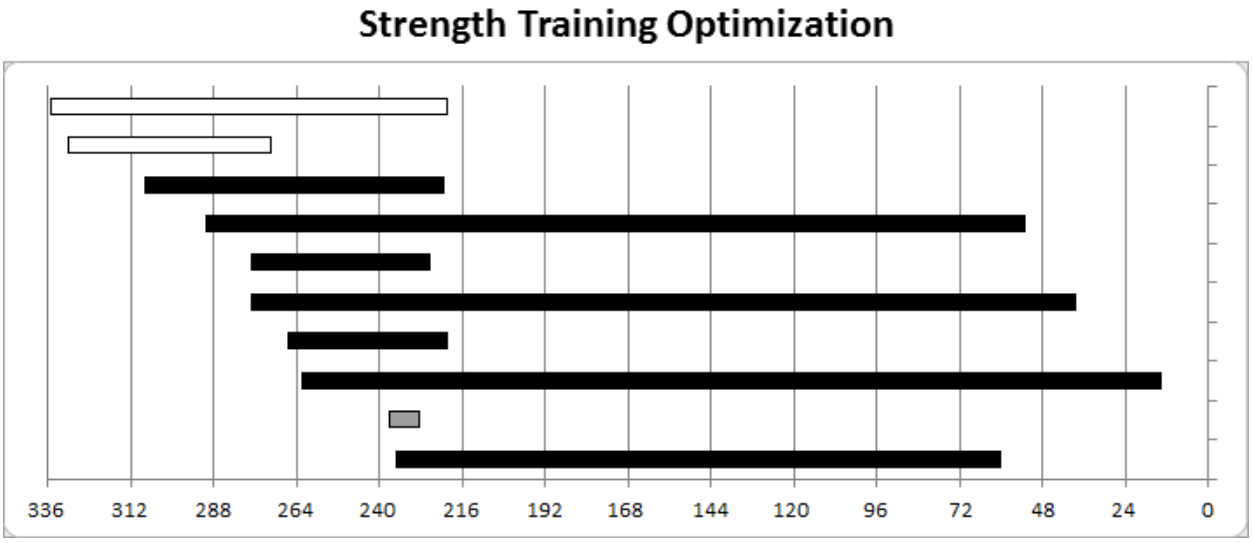


Fig. VIII.B.13: Non-Symptom – Strength Training



APPENDIX B (continued)

Fig. IX.B.14: Non-Symptom – Cardio Training

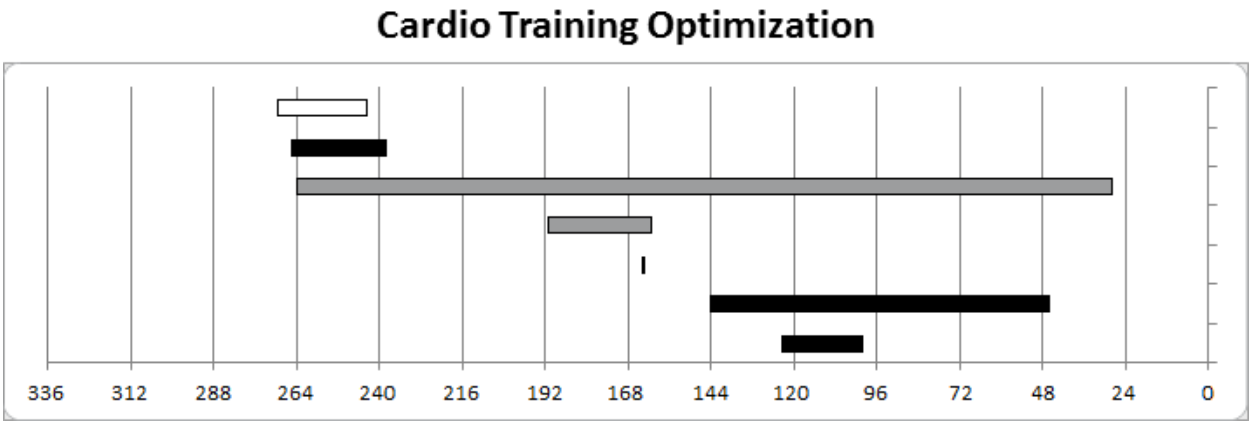


Fig. IX.B.15: Non-Symptom – Sleeping Pill

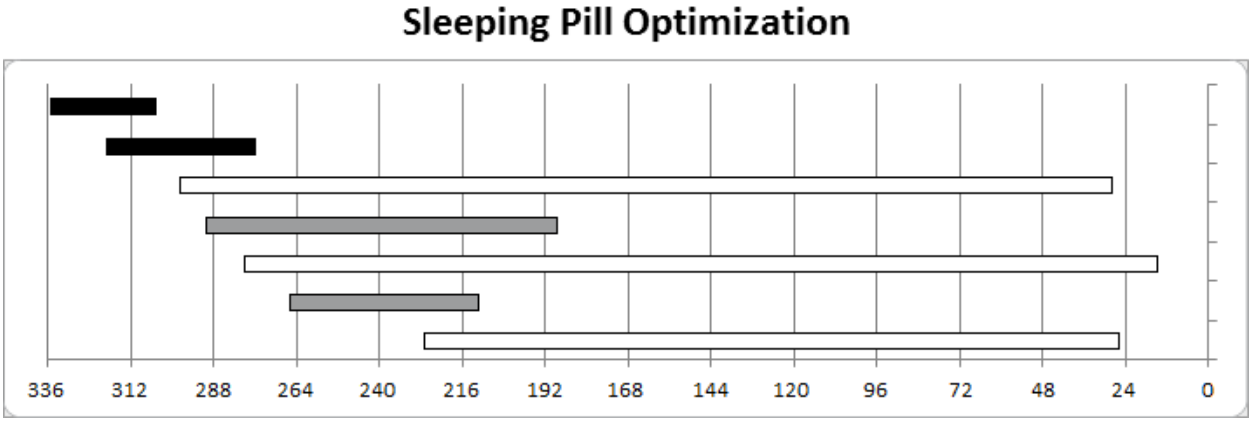
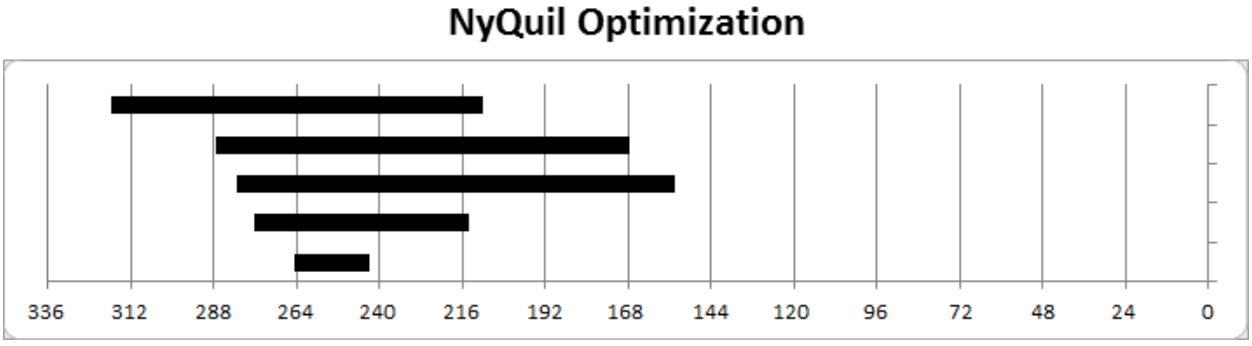


Fig. IX.B.16: Non-Symptom – NyQuil



APPENDIX B (continued)

Fig. IX.B.17: Non-Symptom – Methotrexate

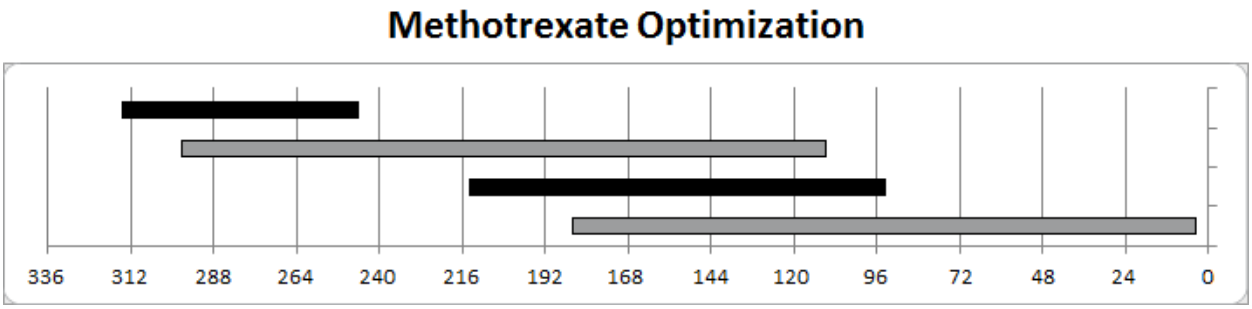


Fig. IX.B.18: Non-Symptom – Vitamin D

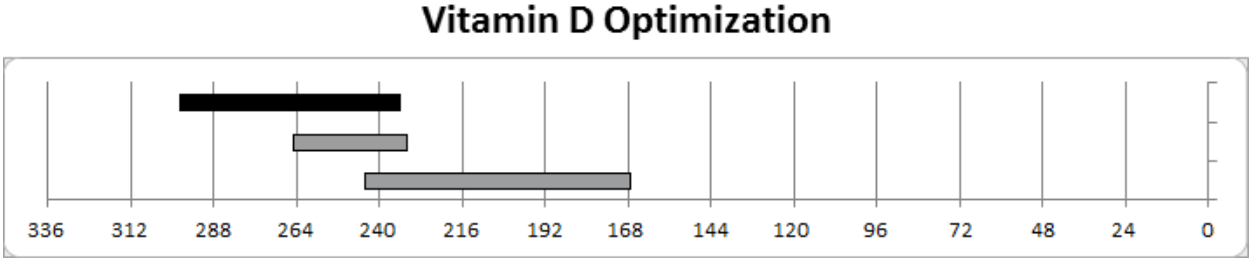
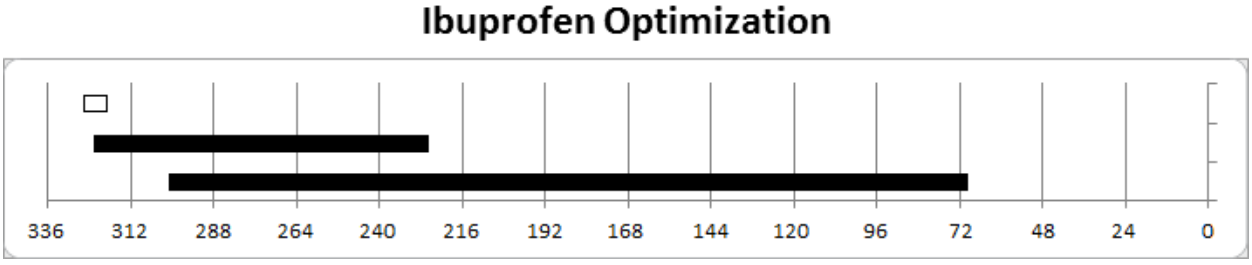


Fig. IX.B.19: Non-Symptom – Ibuprofen



APPENDIX B (continued)

Fig. IX.B.20: Non-Symptom – Humira

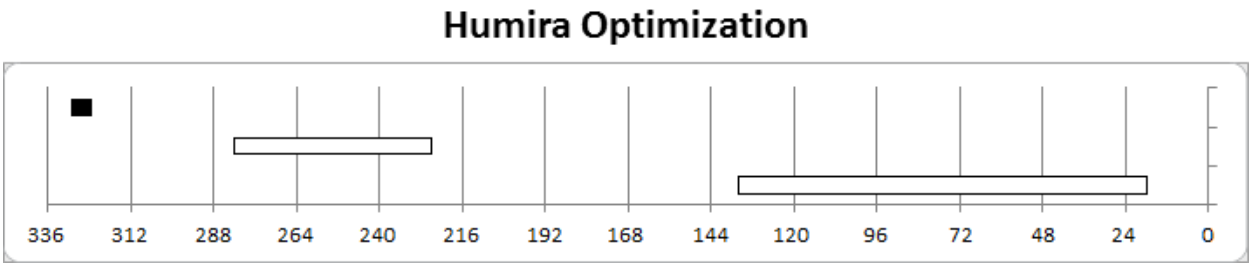


Fig. IX.B.21: Non-Symptom – Green Tea

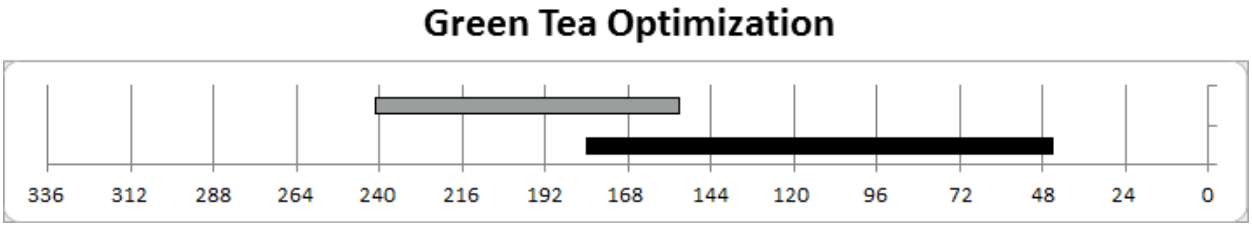


Fig. IX.B.22: Non-Symptom – Dicyclomine

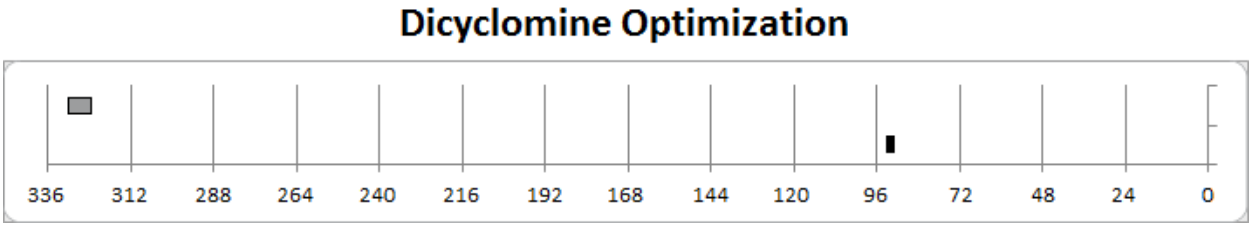
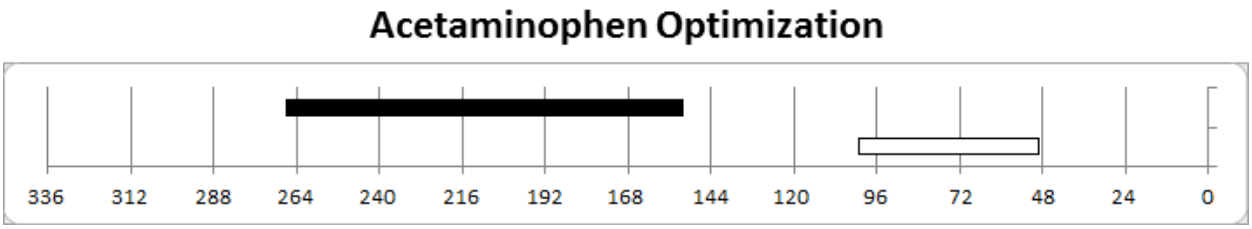


Fig. IX.B.22: Non-Symptom – Acetaminophen



APPENDIX C

Medication Data

	Time	Name	Dose	Date
1	2:09	Sleeping Pill	25	1-Jan
1	5:45	Prednisone	10	
1	5:45	Asacol HD	2400	
1	15:30	Prednisone	10	
1	15:30	Asacol HD	2400	
2	5:25	Prednisone	10	2-Jan
2	5:25	Asacol HD	2400	
2	17:10	Prednisone	10	
2	17:10	Asacol HD	2400	
3	8:30	Prednisone	10	3-Jan
3	8:30	Asacol HD	2400	
3	20:30	Prednisone	10	
3	20:30	Asacol HD	2400	
3	23:59	Sleeping Pill	25	
4	0:15	Sleeping Pill	25	4-Jan
4	8:15	Prednisone	10	
4	8:15	Asacol HD	2400	
4	11:20	Vitamin D	10	
4	11:20	Calcium	600	
4	17:03	Prednisone	10	
4	17:03	Asacol HD	2400	
5	6:20	Prednisone	10	5-Jan
5	6:20	Asacol HD	2400	
5	7:10	Vitamin D	10	
5	7:10	Calcium	600	
5	16:20	Prednisone	10	
5	16:20	Asacol HD	2400	
6	0:15	Sleeping Pill	25	6-Jan
6	6:25	Prednisone	10	
6	6:25	Asacol HD	2400	
6	17:00	Prednisone	10	
6	17:00	Asacol HD	2400	
7	5:45	Prednisone	10	7-Jan
7	18:15	Prednisone	10	
7	18:15	Asacol HD	2400	
8	5:00	Prednisone	10	8-Jan
8	5:00	Asacol HD	2400	

APPENDIX C (continued)

8	10:00	VSL #3	1	
8	13:30	Vitamin D	10	
8	13:30	Calcium	600	
8	22:30	Sleeping Pill	25	
9	6:00	Prednisone	10	9-Jan
9	6:00	Asacol HD	2400	
9	18:50	Prednisone	10	
9	18:50	Asacol HD	2400	
10	6:40	Prednisone	10	10-Jan
10	6:40	Asacol HD	1600	
10	19:10	Prednisone	10	
10	19:10	Asacol HD	2400	
11	6:45	Prednisone	10	11-Jan
11	6:45	Asacol HD	2400	
11	18:20	Prednisone	10	
11	18:20	Asacol HD	2400	
12	8:00	Prednisone	10	12-Jan
12	8:00	Asacol HD	2400	
12	14:20	Humira	40	
12	21:00	Prednisone	10	
12	21:00	Asacol HD	2400	
13	7:15	Prednisone	10	13-Jan
13	7:15	Asacol HD	2400	
13	18:00	Prednisone	10	
13	18:00	Asacol HD	2400	
13	23:45	Sleeping Pill	25	
14	7:05	Prednisone	10	14-Jan
14	7:05	Asacol HD	2400	
14	17:45	Prednisone	10	
14	17:45	Asacol HD	2400	
14	22:10	Sleeping Pill	25	
15	6:30	Prednisone	10	15-Jan
15	6:30	Asacol HD	2400	
15	18:00	Prednisone	10	
15	18:00	Asacol HD	2400	
16	6:25	Prednisone	10	16-Jan
16	6:25	Asacol HD	2400	
16	18:40	Prednisone	10	
16	18:40	Asacol HD	2400	
17	6:00	Prednisone	10	17-Jan

APPENDIX C (continued)

17	6:00	Asacol HD	2400	
17	12:45	Vitamin D	10	
17	12:45	Calcium	600	
17	18:15	Prednisone	10	
17	18:15	Asacol HD	2400	
18	6:15	Prednisone	10	18-Jan
18	6:15	Asacol HD	2400	
18	18:15	Prednisone	10	
18	18:15	Asacol HD	2400	
18	23:10	Sleeping Pill	25	
19	6:50	Prednisone	10	19-Jan
19	6:50	Asacol HD	2400	
19	23:30	Sleeping Pill	25	
20	4:55	Prednisone	10	20-Jan
20	4:55	Asacol HD	2400	
20	18:15	Prednisone	10	
20	18:15	Asacol HD	2400	
20	18:15	Acetaminophen	500	
21	0:50	Acetaminophen	500	21-Jan
21	0:50	Sleeping Pill	25	
21	7:55	Prednisone	10	
21	7:55	Asacol HD	2400	
21	16:25	Vitamin D	10	
21	16:25	Calcium	600	
21	18:30	Prednisone	10	
21	18:30	Asacol HD	2400	
22	1:30	Sleeping Pill	25	22-Jan
22	6:15	Prednisone	10	
22	6:15	Asacol HD	2400	
22	18:00	Prednisone	10	
22	18:00	Asacol HD	2400	
23	5:45	Prednisone	10	23-Jan
23	5:45	Asacol HD	2400	
23	19:00	Prednisone	10	
23	19:00	Asacol HD	2400	
24	0:10	Sleeping Pill	25	24-Jan
24	5:10	Prednisone	10	
24	5:10	Asacol HD	2400	
24	18:00	Prednisone	10	
24	18:00	Asacol HD	2400	

APPENDIX C (continued)

24	18:00	Methotrexate	25	
24	22:45	Sleeping Pill	25	
25	6:00	Prednisone	10	25-Jan
25	6:00	Asacol HD	2400	
25	18:00	Prednisone	10	
25	18:00	Asacol HD	2400	
25	23:00	Sleeping Pill	25	
26	5:30	Prednisone	10	26-Jan
26	5:30	Asacol HD	2400	
26	17:35	Humira	40	
27	7:50	Prednisone	10	27-Jan
27	7:50	Asacol HD	2400	
27	19:45	Prednisone	10	
27	19:45	Asacol HD	2400	
28	6:00	Prednisone	10	28-Jan
28	6:00	Asacol HD	2400	
28	17:45	Prednisone	10	
28	17:45	Asacol HD	2400	
29	6:45	Prednisone	10	29-Jan
29	6:45	Asacol HD	2400	
29	18:00	Prednisone	10	
29	18:00	Asacol HD	2400	
30	0:45	Sleeping Pill	25	30-Jan
30	21:50	Prednisone	10	
30	21:50	Asacol HD	2400	
31	6:00	Prednisone	10	31-Jan
31	6:00	Asacol HD	2400	
31	18:30	Prednisone	10	
31	18:30	Asacol HD	2400	
32	5:40	Prednisone	10	1-Feb
32	5:40	Asacol HD	2400	
32	18:15	Prednisone	10	
32	18:15	Asacol HD	2400	
33	7:00	Prednisone	10	2-Feb
33	7:00	Asacol HD	2400	
34	5:30	Prednisone	10	3-Feb
34	5:30	Asacol HD	2400	
34	18:00	Prednisone	10	
34	18:00	Asacol HD	2400	
35	6:00	Prednisone	10	4-Feb

APPENDIX C (continued)

35	6:00	Asacol HD	2400	
36	6:00	Prednisone	10	5-Feb
36	6:00	Asacol HD	2400	
36	18:45	Prednisone	10	
36	18:45	Asacol HD	2400	
36	23:15	Sleeping Pill	25	
37	5:20	Prednisone	10	6-Feb
37	5:20	Asacol HD	2400	
37	18:30	Prednisone	10	
37	18:30	Asacol HD	2400	
38	5:45	Prednisone	10	7-Feb
38	5:45	Asacol HD	800	
38	18:30	Prednisone	10	
39	6:50	Prednisone	10	8-Feb
39	6:50	Asacol HD	2400	
39	9:40	Humira	40	
39	17:50	Prednisone	10	
39	17:50	Asacol HD	2400	
40	7:25	Prednisone	10	9-Feb
40	7:25	Asacol HD	2400	
40	18:10	Prednisone	10	
40	18:10	Asacol HD	2400	
41	6:15	Prednisone	10	10-Feb
41	6:15	Asacol HD	2400	
41	18:10	Prednisone	10	
41	18:10	Asacol HD	2400	
41	23:50	Sleeping Pill	50	
42	6:05	Prednisone	10	11-Feb
42	6:05	Asacol HD	2400	
42	19:00	Prednisone	10	
42	19:00	Asacol HD	2400	
43	6:00	Prednisone	10	12-Feb
43	6:00	Asacol HD	2400	
43	19:10	Prednisone	10	
43	19:10	Asacol HD	2400	
43	22:00	Sleeping Pill	25	
44	5:35	Prednisone	10	13-Feb
44	5:35	Asacol HD	2400	
44	11:05	Methotrexate	25	
44	12:15	Vitamin D	10	

APPENDIX C (continued)

44	12:15	Calcium	600	
44	18:15	Prednisone	10	
44	18:15	Asacol HD	2400	
45	5:20	Prednisone	10	14-Feb
45	5:20	Asacol HD	2400	
45	14:50	Vitamin D	10	
45	14:50	Calcium	600	
45	18:30	Prednisone	10	
45	18:30	Asacol HD	2400	
46	7:50	Prednisone	10	15-Feb
46	7:50	Asacol HD	2400	
46	18:30	Prednisone	10	
46	18:30	Asacol HD	2400	
47	5:30	Acetaminophen	1000	16-Feb
47	5:30	Prednisone	10	
47	5:30	Asacol HD	2400	
47	18:00	Prednisone	10	
47	18:00	Asacol HD	2400	
48	5:25	Prednisone	10	17-Feb
48	5:25	Asacol HD	2400	
48	11:20	Vitamin D	10	
48	11:20	Calcium	600	
48	18:10	Prednisone	10	
48	18:10	Asacol HD	2400	
49	5:05	Prednisone	10	18-Feb
49	5:05	Asacol HD	2400	
49	18:00	Prednisone	10	
49	18:00	Asacol HD	2400	
50	7:10	Prednisone	10	19-Feb
50	7:10	Asacol HD	2400	
50	9:00	Nicotine (transdermal)	14	
50	18:50	Prednisone	10	
50	18:50	Asacol HD	2400	
51	6:15	Prednisone	10	20-Feb
51	6:15	Asacol HD	2400	
51	18:00	Prednisone	10	
51	18:00	Asacol HD	2400	
52	4:00	Prednisone	10	21-Feb
52	4:00	Asacol HD	2400	

APPENDIX C (continued)

52	12:00	Methotrexate	25	
52	17:15	Prednisone	10	
52	17:15	Asacol HD	2400	
53	6:00	Prednisone	10	22-Feb
53	6:00	Asacol HD	2400	
53	12:45	Humira	40	
54	6:30	Prednisone	10	23-Feb
54	6:30	Asacol HD	2400	
54	18:00	Prednisone	10	
54	18:00	Asacol HD	2400	
55	7:30	Prednisone	10	24-Feb
55	7:30	Asacol HD	2400	
55	18:20	Prednisone	10	
55	18:20	Asacol HD	2400	
56	5:25	Prednisone	10	25-Feb
56	5:25	Asacol HD	2400	
56	17:40	Prednisone	10	
56	17:40	Asacol HD	2400	
57	6:00	Prednisone	10	26-Feb
57	6:00	Asacol HD	2400	
57	22:40	Prednisone	10	
57	22:40	Asacol HD	2400	
58	6:00	Prednisone	10	27-Feb
58	6:00	Asacol HD	2400	
58	9:50	Acetaminophen	500	
58	17:40	Prednisone	10	
58	17:40	Asacol HD	2400	
59	7:40	Prednisone	10	28-Feb
59	7:40	Asacol HD	2400	
59	12:20	Methotrexate	25	
59	17:20	Prednisone	10	
59	17:20	Asacol HD	2400	
60	5:20	Prednisone	10	29-Feb
60	5:20	Asacol HD	2400	
60	18:30	Prednisone	10	
60	18:30	Asacol HD	2400	
61	6:15	Prednisone	10	1-Mar
61	6:15	Asacol HD	2400	
61	7:00	Vitamin D	10	
61	7:00	Calcium	600	

APPENDIX C (continued)

61	18:00	Prednisone	10	
61	18:00	Asacol HD	2400	
62	5:50	Prednisone	10	2-Mar
62	5:50	Asacol HD	2400	
62	17:50	Prednisone	10	
62	17:50	Asacol HD	2400	
62	22:45	Vitamin D	10	
62	22:45	Calcium	600	
63	5:25	Prednisone	10	3-Mar
63	5:25	Asacol HD	2400	
63	18:00	Prednisone	10	
63	18:00	Asacol HD	2400	
64	6:25	Prednisone	10	4-Mar
64	6:25	Asacol HD	2400	
65	6:25	Prednisone	10	5-Mar
65	6:25	Asacol HD	2400	
65	8:50	Prednisone	10	
66	6:30	Prednisone	10	6-Mar
66	6:30	Asacol HD	2400	
66	18:00	Prednisone	10	
66	18:00	Asacol HD	2400	
67	6:45	Prednisone	10	7-Mar
67	6:45	Asacol HD	2400	
67	17:55	Prednisone	10	
67	17:55	Asacol HD	2400	
67	18:30	Methotrexate	25	
67	18:30	Humira	40	
68	5:45	Prednisone	10	8-Mar
68	5:45	Asacol HD	2400	
68	17:00	VSL #3	1	
68	18:00	Prednisone	10	
68	18:00	Asacol HD	2400	
69	5:35	Prednisone	10	9-Mar
69	5:35	Asacol HD	2400	
69	19:00	Prednisone	10	
69	19:00	Asacol HD	2400	
70	6:30	Prednisone	10	10-Mar
70	6:30	Asacol HD	2400	
70	17:50	Prednisone	10	
70	17:50	Asacol HD	2400	

APPENDIX C (continued)

71	7:10	Prednisone	10	11-Mar
71	7:10	Asacol HD	2400	
71	10:30	VSL #3	1	
71	18:25	Prednisone	10	
71	18:25	Asacol HD	2400	
72	5:40	Prednisone	10	12-Mar
72	5:40	Asacol HD	2400	
72	10:20	VSL #3	1	
72	12:45	Vitamin D	10	
72	12:45	Calcium	600	
72	17:55	Prednisone	10	
72	17:55	Asacol HD	2400	
73	5:55	Prednisone	10	13-Mar
73	5:55	Asacol HD	2400	
73	14:20	VSL #3	1	
73	14:55	Methotrexate	25	
73	18:05	Prednisone	10	
73	18:05	Asacol HD	2400	
74	6:30	Prednisone	10	14-Mar
74	6:30	Asacol HD	2400	
74	15:15	Vitamin D	10	
74	15:15	Calcium	600	
74	18:00	Prednisone	10	
74	18:00	Asacol HD	2400	
75	7:10	Prednisone	10	15-Mar
75	7:10	Asacol HD	2400	
75	9:45	VSL #3	1	
75	9:45	Vitamin D	10	
75	9:45	Calcium	600	
75	16:35	Prednisone	10	
75	16:35	Asacol HD	2400	
76	5:30	Prednisone	10	16-Mar
76	5:30	Asacol HD	2400	
76	14:30	Vitamin D	10	
76	14:30	Calcium	600	
76	16:20	VSL #3	1	
76	17:45	Prednisone	10	
76	17:45	Asacol HD	2400	
77	7:00	Prednisone	10	17-Mar
77	7:00	Asacol HD	2400	

APPENDIX C (continued)

77	9:00	Vitamin D	10	
77	9:00	Calcium	600	
77	11:30	VSL #3	1	
77	18:10	Prednisone	10	
77	18:10	Asacol HD	2400	
78	7:00	Prednisone	10	18-Mar
78	7:00	Asacol HD	2400	
78	17:45	Prednisone	10	
78	17:45	Asacol HD	2400	
78	17:45	Vitamin D	10	
78	17:45	Calcium	600	
79	6:30	Prednisone	10	19-Mar
79	6:30	Asacol HD	2400	
79	15:20	VSL #3	1	
79	17:50	Prednisone	10	
79	17:50	Asacol HD	2400	
80	6:45	Prednisone	10	20-Mar
80	6:45	Asacol HD	2400	
80	17:00	VSL #3	1	
80	18:00	Asacol HD	2400	
81	6:30	Prednisone	10	21-Mar
81	6:30	Asacol HD	2400	
81	13:25	VSL #3	1	
81	14:00	Methotrexate	25	
81	14:00	Humira	40	
81	18:00	Acetaminophen	1000	
82	5:30	Acetaminophen	1000	22-Mar
82	5:30	Asacol HD	2400	
82	16:10	Acetaminophen	1000	
82	16:50	VSL #3	1	
82	23:30	Sleeping Pill	25	
83	2:00	NyQuil	2	23-Mar
83	9:15	Acetaminophen	500	
83	11:15	Acetaminophen	500	
83	13:45	Acetaminophen	500	
83	15:20	VSL #3	1	
84	3:45	Acetaminophen	500	24-Mar
84	11:00	Acetaminophen	500	
86	3:35	Sleeping Pill	25	25-Mar
87	9:30	VSL #3	1	26-Mar

APPENDIX C (continued)

88	17:30	Acetaminophen	500	28-Mar
88	22:30	Sleeping Pill	25	
90	14:00	VSL #3	1	30-Mar
90	18:30	Methotrexate	25	
95	12:35	Humira	40	4-Apr
98	20:20	Asacol HD	2400	7-Apr
102	11:10	Humira	40	11-Apr
105	23:30	Methotrexate	25	14-Apr
105	23:30	Asacol HD	2400	
108	4:30	Asacol HD	2400	17-Apr
109	2:55	Asacol HD	2400	18-Apr
109	10:20	Humira	40	
110	5:10	Asacol HD	2400	19-Apr
110	18:00	VSL #3	1	
111	6:25	Asacol HD	2400	20-Apr
112	10:00	Asacol HD	2400	21-Apr
112	20:00	Asacol HD	2400	
113	6:35	Asacol HD	2400	22-Apr
113	13:00	Nicotine (transdermal)	14	
114	4:00	Asacol HD	2400	23-Apr
114	12:40	Nicotine (transdermal)	14	
115	0:55	Asacol HD	2400	24-Apr
115	13:00	Nicotine (transdermal)	14	
116	1:30	Asacol HD	2400	25-Apr
116	13:40	Nicotine (transdermal)	14	
116	22:10	Asacol HD	2400	
117	23:40	Asacol HD	2400	26-Apr
118	9:45	Asacol HD	2400	27-Apr
118	20:25	Nicotine (transdermal)	14	
118	20:25	Asacol HD	2400	
119	9:00	Asacol HD	2400	28-Apr
119	13:50	Nicotine (transdermal)	14	
120	5:30	Asacol HD	2400	29-Apr
120	16:00	Asacol HD	2400	

APPENDIX C (continued)

120	16:00	Nicotine (transdermal)	14	
121	10:10	Asacol HD	2400	30-Apr
121	14:05	Nicotine (transdermal)	14	
121	22:45	Asacol HD	2400	
122	10:10	Asacol HD	2400	1-May
122	13:30	Nicotine (transdermal)	14	
122	14:00	Humira	40	
122	14:00	Methotrexate	25	
122	22:10	Asacol HD	2400	
123	12:25	Nicotine (transdermal)	14	2-May
123	12:25	Asacol HD	2400	
124	8:00	Asacol HD	2400	3-May
124	23:55	Asacol HD	2400	
125	15:30	Nicotine (transdermal)	14	4-May
126	14:50	Nicotine (transdermal)	21	5-May
126	23:00	Asacol HD	2400	
127	9:10	Asacol HD	2400	6-May
127	14:40	Nicotine (transdermal)	21	
128	5:20	Asacol HD	2400	7-May
128	13:25	Nicotine (transdermal)	21	
129	4:20	Asacol HD	2400	8-May
129	13:15	Nicotine (transdermal)	21	
129	14:45	Methotrexate	25	
129	14:45	Humira	40	
129	14:45	Asacol HD	2400	
130	7:20	Asacol HD	2400	9-May
130	13:15	Nicotine (transdermal)	21	
131	13:20	Nicotine (transdermal)	21	10-May
132	12:25	Nicotine (transdermal)	21	11-May
133	8:25	Asacol HD	2400	12-May

APPENDIX C (continued)

133	13:40	Nicotine (transdermal)	14	
134	8:30	Asacol HD	2400	13-May
134	14:00	Nicotine (transdermal)	14	
135	13:45	Nicotine (transdermal)	14	14-May
136	11:30	Humira	40	15-May
136	11:30	Methotrexate	25	
136	13:25	Nicotine (transdermal)	14	
137	13:40	Nicotine (transdermal)	14	16-May
138	13:00	Nicotine (transdermal)	14	17-May
140	15:30	Nicotine (transdermal)	21	19-May
140	15:30	Asacol HD	2400	
141	15:40	Nicotine (transdermal)	21	20-May
141	23:45	Humira	40	
141	23:45	Asacol HD	2400	
142	14:40	Nicotine (transdermal)	14	21-May
143	10:00	Asacol HD	2400	22-May
143	13:40	Nicotine (transdermal)	21	
143	15:20	Methotrexate	25	
144	10:00	Asacol HD	2400	23-May
144	13:45	Nicotine (transdermal)	14	
145	13:45	Nicotine (transdermal)	21	24-May
146	1:15	Sleeping Pill	25	25-May
146	1:15	Acetaminophen	1000	
146	1:15	Nicotine (transdermal)	14	
146	13:45	NyQuil	2	
147	2:20	NyQuil	2	26-May
147	12:30	Ibuprofen	200	
147	13:50	Ibuprofen	400	
147	18:20	Ibuprofen	200	
147	23:30	Ibuprofen	200	

APPENDIX C (continued)

148	11:30	Nicotine (transdermal)	21	27-May
148	21:30	Asacol HD	2400	
149	13:30	Asacol HD	2400	28-May
149	13:30	Ibuprofen	200	
149	19:20	Nicotine (transdermal)	14	
150	1:45	Sleeping Pill	25	29-May
150	15:10	Nicotine (transdermal)	14	
151	15:10	Nicotine (transdermal)	14	30-May
151	15:10	Methotrexate	25	
152	17:20	Ibuprofen	400	31-May
152	22:00	Sleeping Pill	25	
152	22:00	Ibuprofen	200	
153	15:45	Ibuprofen	200	1-Jun
153	15:45	Nicotine (transdermal)	21	
153	21:00	Ibuprofen	400	
155	14:10	Nicotine (transdermal)	14	3-Jun
156	0:15	Sleeping Pill	25	4-Jun
157	10:35	Nicotine (transdermal)	14	5-Jun
157	10:35	Humira	40	
157	23:55	Asacol HD	2400	
157	23:55	Sleeping Pill	25	
158	13:45	Nicotine (transdermal)	21	6-Jun
159	14:30	Nicotine (transdermal)	21	7-Jun
159	23:55	Asacol HD	2400	
160	2:15	Sleeping Pill	25	8-Jun
160	12:30	Nicotine (transdermal)	21	
161	5:30	Sleeping Pill	25	9-Jun
161	12:35	Nicotine (transdermal)	21	
162	0:20	Sleeping Pill	25	10-Jun
162	13:10	Nicotine (transdermal)	21	

APPENDIX C (continued)

162	22:30	Sleeping Pill	25	
163	13:45	Nicotine (transdermal)	21	11-Jun
164	13:00	Nicotine (transdermal)	21	12-Jun
165	13:45	Nicotine (transdermal)	14	13-Jun
166	1:20	Sleeping Pill	25	14-Jun
167	13:20	Nicotine (transdermal)	21	15-Jun
168	12:00	Nicotine (transdermal)	21	16-Jun
169	1:00	Sleeping Pill	25	17-Jun
169	23:10	Sleeping Pill	25	
170	14:00	Nicotine (transdermal)	14	18-Jun
171	13:30	Nicotine (transdermal)	14	19-Jun
171	23:45	Asacol HD	2400	
171	23:45	Sleeping Pill	25	
172	12:45	Nicotine (transdermal)	21	20-Jun
172	23:45	Asacol HD	2400	
173	10:10	Asacol HD	2400	21-Jun
173	13:00	Nicotine (transdermal)	21	
174	13:15	Nicotine (transdermal)	21	22-Jun
174	16:30	Methotrexate	25	
175	13:30	Nicotine (transdermal)	14	23-Jun
176	0:10	Sleeping Pill	25	24-Jun
176	0:10	Ibuprofen	200	
177	13:00	Nicotine (transdermal)	14	25-Jun
178	14:00	Nicotine (transdermal)	14	26-Jun
179	13:30	Nicotine (transdermal)	14	27-Jun
180	1:00	Dicyclomine	10	28-Jun
180	11:30	Dicyclomine	10	
180	11:30	Asacol HD	2400	
180	22:10	Dicyclomine	10	
180	22:10	Asacol HD	2400	

APPENDIX C (continued)

Dietary Data

	Time	Ingredient	Unit	Quantity	Date
1	9:55	banana	cup	0.75	1-Jan
1	13:00	candy (Jolly Rancher)	pc	2	
1	15:00	juice (cucumber)	pc	0.5	
1	15:00	juice (apple)	pc	2	
1	15:00	juice (fennel stalk)	pc	1	
1	15:00	juice (pear)	pc	2	
1	15:00	juice (kiwi)	pc	1	
1	15:50	salmon	oz	3	
1	15:50	fennel	cup	0.5	
1	15:50	marjoram	tbsp	0.25	
1	16:50	banana	cup	0.75	
1	18:30	candy (Nips)	pc	1	
1	23:18	banana	cup	0.75	
2	5:25	H2O	oz	3	2-Jan
2	6:45	banana	cup	0.75	
2	6:45	H2O	oz	5	
2	8:30	pork	oz	4	
2	8:30	tarragon	tbsp	0.25	
2	8:30	honey	tbsp	1	
2	8:40	H2O	oz	8	
2	12:45	banana	cup	0.75	
2	15:00	pork	oz	6	
2	15:00	capsicum (red)	cup	0.5	
2	15:00	capsicum (green)	cup	0.5	
2	15:00	olive oil	tbsp	1	
2	15:00	garlic	tbsp	1	
2	15:00	cumin	tsp	0.5	
2	15:00	coriander	tsp	0.5	
2	15:00	salt	tsp	0.25	
2	17:10	soda (zevia cherry vanilla)	oz	12	
2	17:10	banana	cup	0.75	
2	19:19	candy (Sour Patch Xploderz)	pc	30	
2	21:50	banana	cup	0.75	
2	22:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
3	8:30	H2O	oz	16	3-Jan
3	8:30	banana	cup	0.75	

APPENDIX C (continued)

3	11:30	salmon	oz	5	
3	11:30	dill	tbsp	0.5	
3	11:30	juice (lemon)	tbsp	0.5	
3	11:30	garlic pepper	tbsp	0.25	
3	13:09	soda (zevia cherry vanilla)	oz	12	
3	17:00	avocado	cup	0.5	
3	17:00	onion (white)	cup	0.25	
3	17:00	pork (Torchy's)	oz	6	
3	17:00	cilantro	tbsp	1	
3	17:00	tomatillo salsa	tbsp	1	
3	17:00	tortilla (corn)	pc	2	
3	17:00	H2O	oz	16	
3	18:30	H2O	oz	16	
3	22:00	banana	cup	0.75	
4	8:15	H2O	oz	8	4-Jan
4	8:45	H2O	oz	16	
4	9:35	banana	cup	2	
4	11:20	tuna	oz	5	
4	11:20	nutmeg	tsp	0.25	
4	11:20	coriander	tsp	0.25	
4	11:20	sage	tbsp	0.25	
4	11:20	salt	tsp	0.125	
4	11:20	vinegar (apple cider)	tbsp	2	
4	11:20	soda (zevia root beer)	oz	12	
4	15:00	banana	cup	0.75	
4	17:03	H2O	oz	6	
4	17:25	soda (zevia cherry vanilla)	oz	12	
4	19:30	pork	oz	3	
4	19:30	tarragon	tbsp	0.25	
4	19:30	fennel seed	tbsp	0.5	
4	19:30	jelly (mint)	tbsp	1	
4	20:30	banana	cup	0.75	
4	20:50	gum (Big Red)	pc	1	
5	6:20	H2O	oz	12	5-Jan
5	7:10	banana	cup	1	
5	7:10	H2O	oz	6	
5	11:15	avocado	cup	0.5	
5	11:15	onion (white)	cup	0.25	
5	11:15	pork (Torchy's)	oz	6	
5	11:15	cilantro	tbsp	1	

APPENDIX C (continued)

5	11:15	tomatillo salsa	tbsp	1	
5	11:15	tortilla (corn)	pc	2	
5	11:15	juice (lime)	tbsp	1	
5	11:15	soda (zevia cherry vanilla)	oz	12	
5	15:30	banana	cup	1	
5	15:30	candy (Nips)	pc	1	
5	16:20	H2O	oz	4	
5	20:00	banana	cup	0.75	
5	22:30	salmon	oz	6	
5	22:30	cajun spices	tbsp	1	
5	22:45	candy (Jolly Rancher)	pc	3	
6	6:25	H2O	oz	16	6-Jan
6	9:00	banana	cup	0.75	
6	9:00	H2O	oz	16	
6	11:45	tuna	oz	5	
6	11:45	basil	tbsp	0.5	
6	11:45	mint	tbsp	0.5	
6	11:45	vinegar (red wine)	tbsp	3	
6	11:45	salt	tsp	0.125	
6	11:45	soda (zevia cherry vanilla)	oz	12	
6	11:45	gum (Big Red)	pc	2	
6	14:20	banana	cup	0.75	
6	14:20	soft drink (Gatorade)	oz	15	
6	16:00	couscous	cup	0.5	
6	16:00	lentil	cup	0.05	
6	16:00	sundried tomato	cup	0.05	
6	17:00	H2O	oz	8	
6	19:30	steak	oz	5	
6	19:30	capsicum (red)	cup	0.25	
6	19:30	capsicum (orange)	cup	0.25	
6	19:30	capsicum (green)	cup	0.5	
6	19:30	tomatillo	cup	0.125	
6	19:30	onion (white)	cup	0.25	
6	19:30	cumin	tbsp	0.25	
6	19:30	coriander	tbsp	0.25	
6	19:30	salt	tsp	0.125	
6	19:30	pepper	tsp	0.125	
6	19:30	marjoram	tbsp	0.25	
6	19:30	juice (lime)	tbsp	0.5	
6	19:30	milk chocolate	pc	5	

APPENDIX C (continued)

6	19:30	soda (rhubarb)	oz	12	
6	21:50	milk chocolate	pc	8	
6	22:30	bread (white)	oz	2	
6	22:30	butter	tbsp	0.15	
6	22:30	cheese (cheddar)	oz	0.125	
7	5:45	H2O	oz	6	7-Jan
7	6:50	banana	cup	0.75	
7	6:50	H2O	oz	10	
7	7:40	H2O	oz	12	
7	9:00	pork	oz	6	
7	9:00	tarragon	tbsp	0.5	
7	9:00	salt	tsp	0.125	
7	9:00	jelly (mint)	tbsp	1	
7	9:00	H2O	oz	6	
7	11:00	banana	cup	0.75	
7	12:30	juice (cucumber)	pc	1	
7	12:30	juice (apple)	pc	1	
7	12:30	juice (fennel stalk)	pc	2	
7	12:30	juice (carrot)	pc	1.5	
7	12:30	juice (kale)	pc	2	
7	12:30	juice (kiwi)	pc	1	
7	14:00	avocado	cup	0.5	
7	14:00	onion (white)	cup	0.25	
7	14:00	pork (Torchy's)	oz	6	
7	14:00	cilantro	tbsp	1	
7	14:00	tomatillo salsa	tbsp	1	
7	14:00	tortilla (corn)	pc	2	
7	14:00	chip (corn tortilla)	pc	1	
7	14:00	egg	oz	1	
7	14:00	potato	cup	0.125	
7	14:00	sauce (Torchy's Diablo)	tbsp	0.25	
7	14:00	mahi mahi	oz	0.5	
7	14:00	cheese (queso fresco)	oz	0.125	
7	14:00	juice (apple)	oz	8	
7	16:55	candy (Nips)	pc	4	
7	18:15	milk chocolate	pc	4	
7	18:15	H2O	oz	16	
7	20:30	banana	cup	0.75	
7	20:30	H2O	oz	16	
7	21:30	milk chocolate	pc	4	

APPENDIX C (continued)

7	21:30	tilapia	oz	5	
7	21:30	celery seed	tbsp	0.125	
7	21:30	salt	tsp	0.125	
7	21:30	vinegar (balsamic)	tbsp	0.125	
7	21:30	marjoram	tbsp	0.125	
7	21:30	bread (white)	oz	1	
7	21:30	rosemary	tbsp	0.125	
7	23:00	candy (Nips)	pc	4	
8	5:00	H2O	oz	2	8-Jan
8	6:45	banana	cup	0.75	
8	8:30	pork	oz	6	
8	8:30	tarragon	tbsp	0.75	
8	8:30	jelly (mint)	tbsp	1	
8	8:30	H2O	oz	10	
8	10:00	milk chocolate	pc	5	
8	10:00	banana	cup	0.75	
8	10:00	juice (apple)	oz	6	
8	10:00	strawberry	cup	1.5	
8	12:30	candy (Nips)	pc	1	
8	12:30	soda (orange seltzer)	oz	12	
8	14:30	steak	oz	5	
8	14:30	garlic pepper	tbsp	0.5	
8	14:30	salt	tsp	0.125	
8	14:30	vinegar (red wine)	tbsp	3	
8	14:30	fennel	cup	1.5	
8	14:30	spinach	cup	1.5	
8	14:30	sugar snap peas	cup	1	
8	15:00	banana	cup	0.75	
8	15:45	milk chocolate	pc	3	
8	20:20	carrot	cup	0.25	
8	20:20	cucumber	cup	0.25	
8	20:20	celery	cup	0.125	
8	20:20	raspberry vinagrette dressing	tbsp	2	
8	20:20	sweet potato	cup	0.75	
8	20:20	rosemary	tbsp	0.125	
8	20:20	cumin	tbsp	0.125	
8	20:20	nutmeg	tbsp	0.125	
8	20:20	vinegar (apple cider)	tbsp	0.125	
8	20:20	salmon	oz	5	
8	20:20	brown sugar	tbsp	0.25	

APPENDIX C (continued)

8	20:20	tarragon	tbsp	0.125	
8	20:20	mustard	tsp	0.125	
8	20:20	hard cider	oz	24	
9	6:00	H2O	oz	6	9-Jan
9	8:45	candy (Jolly Rancher)	pc	2	
9	9:50	banana	cup	0.5	
9	9:50	salmon	oz	5	
9	9:50	brown sugar	tbsp	0.25	
9	9:50	tarragon	tbsp	0.125	
9	9:50	mustard	tsp	0.125	
9	9:50	H2O	oz	16	
9	11:30	candy (Nips)	pc	4	
9	13:15	juice (apple)	pc	2	
9	13:15	juice (fennel stalk)	pc	2	
9	13:15	juice (kale)	pc	2	
9	13:15	juice (kiwi)	pc	2	
9	13:15	juice (peach)	pc	1	
9	13:15	juice (carrot)	pc	3	
9	13:30	bread (white)	oz	2	
9	13:30	rosemary	tbsp	0.125	
9	13:30	olive oil	tbsp	0.25	
9	16:15	banana	cup	1.25	
9	16:45	salmon	oz	5	
9	16:45	brown sugar	tbsp	0.25	
9	16:45	tarragon	tbsp	0.125	
9	16:45	mustard	tsp	0.125	
9	16:45	candy (Jolly Rancher)	pc	2	
9	17:15	H2O	oz	16	
9	18:50	H2O	oz	2	
9	19:00	banana	cup	0.5	
9	19:00	pork	oz	6	
9	19:00	jerk seasoning	tbsp	0.5	
9	22:10	green bean	cup	1.25	
9	22:10	marjoram	tbsp	0.5	
9	22:10	salt	tbsp	0.125	
9	22:10	pepper	tbsp	0.125	
9	22:10	butter	tbsp	0.25	
9	22:10	juice (lemon)	tbsp	0.5	
9	21:30	candy (Nips)	pc	2	

APPENDIX C (continued)

10	0:10	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	10-Jan
10	0:10	honey	tbsp	1	
10	0:40	banana	cup	0.75	
10	6:40	H2O	oz	6	
10	8:30	banana	cup	1	
10	8:30	H2O	oz	10	
10	10:45	avocado	cup	0.5	
10	10:45	onion (white)	cup	0.25	
10	10:45	pork (Torchy's)	oz	6	
10	10:45	cilantro	tbsp	1	
10	10:45	tomatillo salsa	tbsp	1	
10	10:45	tortilla (corn)	pc	2	
10	13:15	strawberry	cup	1.25	
10	13:15	sugar	tbsp	1	
10	14:00	candy (Nips)	pc	1	
10	14:00	gum (strawberry)	pc	1	
10	15:50	milk chocolate	pc	5	
10	17:15	banana	cup	1	
10	18:30	bread (white)	oz	1	
10	18:30	rosemary	tbsp	0.125	
10	19:30	pork	oz	8	
10	19:30	tarragon	tbsp	0.5	
10	19:30	salt	tsp	0.125	
10	19:30	jelly (mint)	tbsp	1.5	
10	22:30	banana	cup	0.75	
10	22:30	milk chocolate	pc	4	
10	22:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
10	22:30	honey	tbsp	1	
11	5:20	H2O	oz	16	11-Jan
11	6:45	H2O	oz	2	
11	6:45	banana	cup	0.75	
11	9:10	H2O	oz	10	
11	11:00	banana	cup	1.25	
11	11:00	milk chocolate	pc	6	
11	11:00	H2O	oz	16	
11	12:45	avocado	cup	0.5	
11	12:45	onion (white)	cup	0.25	
11	12:45	pork (Torchy's)	oz	6	

APPENDIX C (continued)

11	12:45	cilantro	tbsp	1	
11	12:45	tomatillo salsa	tbsp	1	
11	12:45	tortilla (corn)	pc	3	
11	14:20	candy (Jolly Rancher)	pc	2	
11	16:30	milk chocolate	pc	5	
11	18:00	banana	cup	1	
11	18:20	H2O	oz	4	
11	18:40	tuna	oz	5	
11	18:40	bread (white)	oz	5	
11	18:40	sweet pickle relish	tbsp	2	
11	18:40	curry powder	tbsp	0.5	
11	18:40	fennel seed	tbsp	0.5	
11	18:40	mayonaise	tbsp	1	
11	18:40	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
11	18:40	honey	tbsp	1	
11	20:10	strawberry	cup	0.75	
11	20:10	sugar	tbsp	0.5	
11	20:10	milk chocolate	pc	6	
11	21:15	roast beef	oz	3	
11	21:15	bread (wheat)	oz	3	
11	21:15	au jus	oz	3	
12	5:05	banana	cup	1.25	12-Jan
12	9:10	salmon	oz	4	
12	9:10	ginger	tbsp	0.5	
12	9:10	salt	tsp	0.125	
12	9:10	honey	tbsp	1	
12	10:10	H2O	oz	16	
12	10:10	candy (Nips)	pc	1	
12	12:20	pork	oz	6	
12	12:20	capsicum (green)	cup	0.5	
12	12:20	capsicum (red)	cup	0.5	
12	12:20	chili powder	tbsp	0.5	
12	12:20	black olive	cup	0.5	
12	12:20	juice (lime)	tbsp	1	
12	12:20	mint	tbsp	1	
12	12:30	H2O	oz	16	
12	13:40	candy (Nips)	pc	4	
12	15:45	H2O	oz	16	
12	17:35	bread (white)	oz	5	

APPENDIX C (continued)

12	17:35	garlic	tbsp	0.75	
12	17:35	butter	tbsp	0.25	
12	20:00	salmon	oz	6	
12	20:00	dill	tbsp	0.25	
12	20:00	lemon pepper	tbsp	0.25	
12	20:00	couscous	cup	1.5	
12	20:00	garlic	tbsp	0.25	
12	20:00	green bean	cup	1	
12	20:00	marjoram	tbsp	0.25	
12	20:00	butter	tbsp	0.25	
12	20:00	juice (lemon)	tbsp	0.25	
12	20:00	H2O	oz	16	
12	20:00	candy (Nips)	pc	3	
12	21:00	H2O	oz	16	
13	3:50	H2O	oz	10	13-Jan
13	7:15	H2O	oz	2	
13	7:15	banana	cup	1	
13	8:00	milk chocolate	pc	5	
13	8:10	H2O	oz	4	
13	10:10	milk chocolate	pc	8	
13	12:10	pork	oz	6	
13	12:10	brown sugar	tbsp	2	
13	12:10	basil	tbsp	0.5	
13	12:10	mint	tbsp	0.5	
13	12:10	salt	tsp	0.125	
13	13:05	banana	cup	1	
13	13:50	milk chocolate	pc	6	
13	13:50	H2O	oz	16	
13	15:10	avocado	cup	0.5	
13	15:10	onion (white)	cup	0.25	
13	15:10	pork (Torchy's)	oz	6	
13	15:10	cilantro	tbsp	1	
13	15:10	tomatillo salsa	tbsp	1	
13	15:10	tortilla (corn)	pc	2	
13	15:10	juice (lime)	tbsp	1	
13	15:10	cheese (queso fresco)	tbsp	1	
13	15:10	H2O	oz	16	
13	17:10	candy (Nips)	pc	3	
13	18:00	H2O	oz	8	
13	18:30	strawberry	cup	1.5	

APPENDIX C (continued)

13	18:30	sugar	tbsp	2	
13	19:00	banana	cup	1	
13	19:00	H2O	oz	8	
13	22:00	banana	cup	1	
13	22:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
13	22:00	honey	tbsp	1	
14	7:05	H2O	oz	4	14-Jan
14	9:00	banana	cup	1	
14	9:00	milk chocolate	pc	6	
14	11:00	avocado	cup	0.5	
14	11:00	onion (white)	cup	0.25	
14	11:00	pork (Torchy's)	oz	6	
14	11:00	cilantro	tbsp	1	
14	11:00	tomatillo salsa	tbsp	1	
14	11:00	tortilla (corn)	pc	2	
14	11:00	H2O	oz	16	
14	12:05	candy (Nips)	pc	2	
14	14:00	milk chocolate	pc	5	
14	14:00	candy (Nips)	pc	2	
14	14:00	banana	cup	1	
14	17:45	H2O	oz	2	
14	18:40	milk chocolate	pc	5	
14	19:30	tea (green)	oz	18	
14	19:30	bread (corn)	oz	6	
14	19:30	jalapeno	tbsp	0.5	
14	19:30	butter	tbsp	0.25	
14	19:30	dough (rice)	oz	1	
14	19:30	mushroom (white)	cup	0.125	
14	19:30	cabbage (green)	cup	0.125	
14	19:30	ginger	tbsp	0.25	
14	19:30	sauce (soy)	tbsp	0.5	
14	19:30	arugala	cup	0.25	
14	19:30	pork	oz	1	
14	19:30	corn	cup	0.125	
14	19:30	kidney bean	cup	0.125	
14	19:30	bacon	oz	1	
14	19:30	shrimp	oz	3	
14	19:30	crab	oz	2	
14	19:30	bread (wheat)	oz	2	

APPENDIX C (continued)

14	21:00	cake (white)	oz	3	
14	21:00	chocolate fudge	oz	2	
15	0:10	banana	cup	0.75	15-Jan
15	0:10	H2O	oz	14	
15	6:30	H2O	oz	2	
15	8:15	banana	cup	0.75	
15	8:15	H2O	oz	16	
15	9:45	avocado	cup	0.5	
15	9:45	onion (white)	cup	0.25	
15	9:45	pork (Torchy's)	oz	6	
15	9:45	cilantro	tbsp	1	
15	9:45	tortilla (corn)	pc	3	
15	10:20	milk chocolate	pc	8	
15	12:00	banana	cup	0.75	
15	14:30	milk chocolate	pc	6	
15	16:00	pork	oz	8	
15	16:00	ginger	tbsp	1	
15	16:00	honey	tbsp	1	
15	16:00	juice (lime)	tbsp	0.5	
15	16:00	salt	tsp	0.125	
15	16:00	H2O	oz	16	
15	18:00	cake (chocolate)	oz	4	
15	18:00	chocolate fudge	oz	6	
15	18:00	raspberry	cup	0.125	
15	18:00	sugar	tbsp	1	
15	18:30	juice (apple)	pc	2	
15	18:30	juice (kiwi)	pc	2	
15	18:30	juice (fennel stalk)	pc	2	
15	18:30	juice (carrot)	pc	2	
15	18:30	juice (strawberry)	pc	10	
15	21:50	milk chocolate	pc	4	
16	1:40	bread (wheat)	oz	5	16-Jan
16	1:40	mushroom (white)	cup	0.25	
16	1:40	capsicum (red)	cup	0.25	
16	1:40	capsicum (yellow)	cup	0.25	
16	1:40	onion (white)	tbsp	2	
16	1:40	artichoke	cup	0.25	
16	1:40	tomato paste	cup	0.25	
16	1:40	cheese (mozzarella)	cup	0.5	
16	6:25	H2O	oz	3	

APPENDIX C (continued)

16	9:20	banana	cup	0.75	
16	9:20	milk chocolate	pc	6	
16	9:20	H2O	oz	10	
16	11:40	avocado	cup	0.5	
16	11:40	onion (white)	cup	0.25	
16	11:40	pork (Torchy's)	oz	6	
16	11:40	cilantro	tbsp	1	
16	11:40	tomatillo salsa	tbsp	1	
16	11:40	tortilla (corn)	pc	4	
16	12:45	milk chocolate	pc	3	
16	14:15	tea (green)	oz	8	
16	14:15	sugar	tbsp	1	
16	15:30	banana	cup	0.75	
16	15:30	milk chocolate	pc	5	
16	15:40	salmon	oz	3	
16	15:40	mint	tbsp	0.5	
16	15:40	rosemary	tbsp	0.5	
16	15:40	brown sugar	tbsp	1	
16	15:40	vinegar (red wine)	tbsp	1	
16	15:40	tea (green)	oz	8	
16	15:40	sugar	tbsp	1	
16	17:00	milk chocolate	pc	8	
16	18:40	H2O	oz	2	
16	18:40	strawberry	cup	2	
16	18:40	sugar	tbsp	3	
16	18:40	banana	cup	0.5	
16	20:15	milk chocolate	pc	3	
16	22:45	banana	cup	0.5	
16	22:45	candy (Starburst)	pc	8	
17	6:00	H2O	oz	2	17-Jan
17	6:45	banana	cup	0.5	
17	6:45	bread (white)	oz	4	
17	6:45	rosemary	tbsp	0.25	
17	6:45	olive oil	tbsp	0.125	
17	6:45	H2O	oz	6	
17	9:00	candy (Starburst)	pc	4	
17	10:45	pork	oz	8	
17	10:45	tarragon	tbsp	0.5	
17	10:45	jelly (mint)	tbsp	2	
17	10:45	tea (green)	oz	8	

APPENDIX C (continued)

17	11:45	banana	cup	1	
17	11:45	milk chocolate	pc	6	
17	12:45	H2O	oz	16	
17	14:10	banana	cup	0.5	
17	15:15	bread (white)	oz	3	
17	16:45	milk chocolate	pc	6	
17	18:15	milk chocolate	pc	4	
17	18:15	H2O	oz	16	
17	18:15	avocado	cup	0.25	
17	18:15	onion (white)	cup	0.125	
17	18:15	pork (Torchy's)	oz	3	
17	18:15	cilantro	tbsp	0.5	
17	18:15	tomatillo salsa	tbsp	1	
17	18:15	tortilla (corn)	pc	1	
17	22:10	juice (apple)	pc	1.5	
17	22:10	juice (carrot)	pc	2	
17	22:10	juice (kiwi)	pc	1.5	
17	22:10	juice (strawberry)	pc	7	
17	22:10	juice (fennel stalk)	pc	1	
18	6:25	banana	cup	0.5	18-Jan
18	6:25	H2O	oz	4	
18	10:45	banana	cup	0.75	
18	10:45	milk chocolate	pc	8	
18	10:45	H2O	oz	16	
18	10:45	avocado	cup	0.25	
18	10:45	onion (white)	cup	0.125	
18	10:45	pork (Torchy's)	oz	3	
18	10:45	cilantro	tbsp	0.5	
18	10:45	tomatillo salsa	tbsp	1	
18	10:45	tortilla (corn)	pc	2	
18	11:45	strawberry	cup	1.5	
18	11:45	sugar	tbsp	1.5	
18	12:45	milk chocolate	pc	6	
18	14:30	banana	cup	0.75	
18	14:30	tea (green)	oz	8	
18	14:30	honey	tbsp	1	
18	16:50	pork	oz	8	
18	16:50	sugar snap peas	cup	0.75	
18	16:50	capsicum (red)	cup	0.75	
18	16:50	capsicum (green)	cup	0.75	

APPENDIX C (continued)

18	16:50	cumin	tbsp	0.25	
18	16:50	coriander	tbsp	0.25	
18	16:50	rosemary	tbsp	0.25	
18	16:50	salt	tsp	0.25	
18	16:50	fennel	cup	0.75	
18	18:15	H2O	oz	6	
18	19:00	milk chocolate	pc	4	
18	20:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	16	
18	20:00	milk chocolate	pc	6	
18	23:10	banana	cup	0.5	
19	6:50	H2O	oz	14	19-Jan
19	7:45	banana	cup	0.75	
19	8:15	milk chocolate	pc	6	
19	8:30	H2O	oz	16	
19	9:50	milk chocolate	pc	5	
19	11:30	pork	oz	6	
19	11:30	garlic	tbsp	0.5	
19	11:30	jelly (mint)	tbsp	1	
19	12:30	candy (Jolly Rancher)	pc	2	
19	14:45	avocado	cup	0.5	
19	14:45	onion (white)	cup	0.25	
19	14:45	pork (Torchy's)	oz	6	
19	14:45	cilantro	tbsp	1	
19	14:45	tomatillo salsa	tbsp	2	
19	14:45	tortilla (corn)	pc	4	
19	14:45	juice (lime)	tbsp	1	
19	14:45	soft drink (Vitamin Water XXX)	oz	4	
19	14:45	H2O	oz	10	
19	14:45	milk chocolate	pc	8	
19	19:30	chicken	oz	1	
19	19:30	sauce (chili garlic)	tbsp	1	
19	19:30	dressing (honey mustard)	tbsp	2	
19	19:30	cucumber	cup	0.125	
19	19:30	lettuce (romaine)	cup	1	
19	19:30	carrot	cup	0.125	
19	19:30	cabbage (green)	cup	0.125	
19	19:30	steak	oz	10	
19	19:30	butter	tbsp	1	
19	19:30	potato	cup	0.5	

APPENDIX C (continued)

19	19:30	garlic	tbsp	1	
19	19:30	green bean	cup	0.75	
19	19:30	apple	cup	0.25	
19	19:30	bread (wheat)	oz	3	
19	19:30	ice cream (vanilla)	cup	0.125	
19	19:30	soda (Pepsi)	oz	30	
19	19:30	rum	oz	6	
19	19:30	juice (lemon)	tbsp	1	
19	19:30	sugar	tbsp	1	
19	19:30	hard cider	oz	12	
20	8:10	banana	cup	0.75	20-Jan
20	8:10	H2O	oz	12	
20	9:20	banana	cup	0.75	
20	9:20	H2O	oz	14	
20	9:20	milk chocolate	pc	12	
20	10:50	avocado	cup	0.25	
20	10:50	onion (white)	cup	0.125	
20	10:50	pork (Torchy's)	oz	3	
20	10:50	cilantro	tbsp	0.5	
20	10:50	tomatillo salsa	tbsp	1	
20	10:50	tortilla (corn)	pc	1	
20	10:50	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
20	10:50	honey	tbsp	1	
20	12:15	milk chocolate	pc	5	
20	15:20	salmon	oz	7	
20	15:20	mint	tbsp	1	
20	15:20	honey	tbsp	2	
20	15:20	juice (lemon)	tbsp	1	
20	15:20	salt	tbsp	0.125	
20	15:20	pepper	tsp	0.125	
20	15:20	H2O	oz	14	
20	18:15	H2O	oz	16	
20	19:30	candy (Jolly Rancher)	pc	2	
20	21:30	fava bean	cup	0.125	
20	21:30	cracked wheat	cup	0.125	
20	21:30	chickpea	cup	0.25	
20	21:30	lettuce (romaine)	cup	0.5	
20	21:30	basmati rice	cup	1.5	
20	21:30	bread (pita)	oz	5	

APPENDIX C (continued)

20	21:30	beef (spiced)	oz	5	
20	21:30	garlic	tbsp	2	
20	21:30	olive oil	tbsp	2	
21	0:50	H2O	oz	6	21-Jan
21	4:50	H2O	oz	16	
21	7:55	H2O	oz	6	
21	9:10	banana	cup	0.75	
21	9:35	coffee	oz	8	
21	9:35	sugar	tbsp	1	
21	11:30	avocado	cup	0.5	
21	11:30	onion (white)	cup	0.25	
21	11:30	pork (Torchy's)	oz	6	
21	11:30	cilantro	tbsp	1	
21	11:30	tomatillo salsa	tbsp	2	
21	11:30	tortilla (corn)	pc	4	
21	11:30	juice (apple)	oz	8	
21	11:30	milk chocolate	pc	4	
21	12:20	H2O	oz	16	
21	12:20	milk chocolate	pc	5	
21	14:50	candy (Nips)	pc	2	
21	16:25	banana	cup	0.75	
21	16:25	milk chocolate	pc	8	
21	16:25	soft drink (Gatorade)	oz	20	
21	18:30	banana	cup	1	
21	19:20	milk chocolate	pc	6	
21	20:30	salmon	oz	8	
21	20:30	capsicum (red)	cup	0.5	
21	20:30	artichoke	cup	0.75	
21	20:30	bread (white)	oz	6	
21	20:30	cheese (parmasean)	oz	2	
21	20:30	couscous	cup	1	
21	20:30	mushroom (white)	cup	0.125	
21	20:30	hard cider	oz	24	
21	22:00	milk chocolate	pc	12	
21	22:00	banana	cup	0.25	
22	6:15	H2O	oz	4	22-Jan
22	8:30	H2O	oz	8	
22	9:20	milk chocolate	pc	4	
22	11:45	avocado	cup	0.25	
22	11:45	onion (white)	cup	0.125	

APPENDIX C (continued)

22	11:45	pork (Torchy's)	oz	3	
22	11:45	cilantro	tbsp	0.5	
22	11:45	tomatillo salsa	tbsp	1	
22	11:45	tortilla (corn)	pc	2	
22	11:45	H2O	oz	16	
22	11:45	strawberry	cup	1	
22	11:45	sugar	tbsp	3	
22	12:20	milk chocolate	pc	5	
22	12:20	candy (Nips)	pc	2	
22	14:40	pork	oz	8	
22	14:40	jerk seasoning	tbsp	0.5	
22	14:40	parsley	tbsp	0.25	
22	15:20	banana	cup	1	
22	15:20	milk chocolate	pc	6	
22	18:00	H2O	oz	2	
22	18:00	banana	cup	1	
22	18:00	green bean	cup	0.75	
22	18:00	broccoli	cup	1	
22	18:00	basil	tbsp	1.5	
22	18:00	garlic	tbsp	0.5	
22	18:00	pepper	tsp	0.25	
22	18:00	butter	tbsp	0.25	
22	18:00	juice (lime)	tbsp	0.5	
22	18:00	salt	tsp	0.125	
22	18:00	dried papaya	oz	2	
22	21:45	milk chocolate	pc	8	
22	21:45	banana	cup	0.25	
23	4:00	banana	cup	0.75	23-Jan
23	4:00	H2O	oz	8	
23	5:45	H2O	oz	8	
23	8:45	banana	cup	0.75	
23	10:00	milk chocolate	pc	10	
23	10:30	H2O	oz	16	
23	12:10	fennel	cup	1.5	
23	12:10	pork	oz	6	
23	12:10	tomatillo	cup	0.25	
23	12:10	spinach	cup	2	
23	12:10	basil	tbsp	2	
23	12:10	mint	tbsp	0.5	
23	12:10	salt	tsp	0.125	

APPENDIX C (continued)

23	12:10	vinegar (balsamic)	tbsp	2	
23	12:30	milk chocolate	pc	8	
23	15:00	banana	cup	0.75	
23	15:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
23	15:00	honey	tbsp	1	
23	16:00	candy (Nips)	pc	2	
23	17:45	banana	cup	0.75	
23	17:45	milk chocolate	pc	8	
23	21:30	roast beef	oz	4	
23	21:30	bread (wheat)	oz	4	
23	21:30	au jus	oz	4	
23	23:00	tea (green)	oz	8	
23	23:00	honey	tbsp	1	
23	23:00	cherry	cup	1	
24	0:10	H2O	oz	10	24-Jan
24	5:10	H2O	oz	2	
24	9:45	banana	cup	0.75	
24	9:45	milk chocolate	pc	6	
24	9:45	candy (Nips)	pc	3	
24	10:30	H2O	oz	16	
24	11:30	avocado	cup	0.5	
24	11:30	onion (white)	cup	0.25	
24	11:30	pork (Torchy's)	oz	6	
24	11:30	cilantro	tbsp	1	
24	11:30	tomatillo salsa	tbsp	2	
24	11:30	tortilla (corn)	pc	4	
24	11:30	juice (apple)	oz	8	
24	12:30	milk chocolate	pc	6	
24	14:45	banana	cup	0.75	
24	14:45	milk chocolate	pc	6	
24	16:00	tea (green)	oz	8	
24	16:00	honey	tbsp	1	
24	17:00	pork	oz	6	
24	17:00	sage	tbsp	0.5	
24	17:00	olive oil	tbsp	1	
24	17:00	jelly (mint)	tbsp	1	
24	17:30	milk chocolate	pc	3	
24	17:30	banana	cup	0.75	
24	18:00	H2O	oz	8	

APPENDIX C (continued)

24	19:45	avocado	cup	0.25	
24	19:45	onion (white)	cup	0.125	
24	19:45	pork (Torchy's)	oz	3	
24	19:45	cilantro	tbsp	0.5	
24	19:45	tomatillo salsa	tbsp	1	
24	19:45	tortilla (corn)	pc	2	
24	20:30	onion (white)	cup	0.25	
24	20:30	capsicum (red)	cup	0.25	
24	20:30	capsicum (green)	cup	0.25	
24	20:30	canola oil	tbsp	1	
24	20:30	egg	oz	6	
24	20:30	pepper	tsp	0.25	
24	20:30	tea (green)	oz	8	
24	22:20	H2O	oz	16	
24	22:20	milk chocolate	pc	3	
25	8:00	H2O	oz	16	25-Jan
25	9:45	banana	cup	0.75	
25	9:45	milk chocolate	pc	8	
25	10:45	banana	cup	0.5	
25	10:45	milk chocolate	pc	4	
25	11:45	avocado	cup	0.25	
25	11:45	onion (white)	cup	0.125	
25	11:45	pork (Torchy's)	oz	3	
25	11:45	cilantro	tbsp	0.5	
25	11:45	tomatillo salsa	tbsp	1	
25	11:45	tortilla (corn)	pc	2	
25	11:45	milk chocolate	pc	4	
25	11:45	H2O	oz	16	
25	14:00	tea (green)	oz	8	
25	14:00	candy (Nips)	pc	4	
25	16:40	banana	cup	0.75	
25	16:40	milk chocolate	pc	6	
25	18:00	tea (green)	oz	4	
25	18:00	honey	tbsp	0.5	
25	18:50	banana	cup	0.5	
25	19:30	dough (rice)	oz	1	
25	19:30	cabbage (green)	cup	0.125	
25	19:30	potato	cup	0.125	
25	19:30	sauce (soy)	tbsp	1	
25	19:30	shrimp	oz	2	

APPENDIX C (continued)

25	19:30	rice noodle	oz	5	
25	19:30	beef broth	oz	4	
25	19:30	soy bean	cup	0.25	
25	19:30	green bean	cup	0.25	
25	19:30	onion (green)	cup	0.125	
25	19:30	egg	oz	3	
25	21:30	tea (green)	oz	12	
25	21:30	honey	tbsp	1.5	
25	21:30	candy (Jolly Rancher)	pc	4	
25	21:30	candy (Nips)	pc	2	
25	21:30	cake (white)	oz	2	
25	23:00	milk chocolate	pc	6	
26	5:30	H2O	oz	2	26-Jan
26	6:30	banana	cup	0.125	
26	6:30	milk chocolate	pc	8	
26	6:30	H2O	oz	16	
26	8:00	banana	cup	0.75	
26	8:40	milk chocolate	pc	4	
26	9:35	soft drink (Vitamin Water XXX)	oz	16	
26	13:45	soft drink (Vitamin Water XXX)	oz	16	
26	13:45	avocado	cup	0.5	
26	13:45	onion (white)	cup	0.25	
26	13:45	pork (Torchy's)	oz	6	
26	13:45	cilantro	tbsp	1	
26	13:45	tomatillo salsa	tbsp	2	
26	13:45	tortilla (corn)	pc	4	
26	17:35	banana	cup	0.75	
26	19:15	banana	cup	0.5	
26	19:15	milk chocolate	pc	5	
26	19:15	H2O	oz	16	
26	19:15	avocado	cup	0.25	
26	19:15	onion (white)	cup	0.125	
26	19:15	pork (Torchy's)	oz	3	
26	19:15	cilantro	tbsp	0.5	
26	19:15	tomatillo salsa	tbsp	1	
26	19:15	tortilla (corn)	pc	2	
26	22:20	potato	cup	0.25	
26	22:20	canola oil	tbsp	2	
26	22:20	salt	tsp	0.125	
26	22:20	bread (wheat)	oz	1	

APPENDIX C (continued)

26	22:20	lettuce (romaine)	cup	0.125	
26	22:20	tomato	cup	0.125	
26	22:20	veggie burger	oz	0.125	
26	22:20	cheese (cheddar)	oz	0.5	
26	22:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
27	7:50	H2O	oz	6	27-Jan
27	10:45	avocado	cup	0.25	
27	10:45	onion (white)	cup	0.125	
27	10:45	pork (Torchy's)	oz	3	
27	10:45	cilantro	tbsp	0.5	
27	10:45	tomatillo salsa	tbsp	1	
27	10:45	tortilla (corn)	pc	2	
27	12:00	banana	cup	0.5	
27	14:40	banana	cup	0.5	
27	16:30	banana	cup	0.5	
27	16:30	H2O	oz	16	
27	17:45	avocado	cup	0.25	
27	17:45	onion (white)	cup	0.125	
27	17:45	pork (Torchy's)	oz	3	
27	17:45	cilantro	tbsp	0.5	
27	17:45	tomatillo salsa	tbsp	1	
27	17:45	tortilla (corn)	pc	2	
27	19:30	potato	cup	1	
27	19:30	canola oil	tbsp	2	
27	19:30	salt	tsp	0.25	
27	19:30	bread (wheat)	oz	2	
27	19:30	lettuce (romaine)	cup	0.25	
27	19:30	tomato	cup	0.25	
27	19:30	hamburger	oz	3	
27	19:30	sauce (P Terry's)	tbsp	1	
27	19:30	tea (black)	oz	24	
27	20:30	green bean	cup	2	
27	20:30	carrot	cup	0.25	
27	20:30	garlic	tbsp	0.5	
27	20:30	olive oil	tbsp	0.5	
27	20:30	canola oil	tbsp	0.5	
27	20:30	salt	tsp	0.125	
27	20:30	pepper	tsp	0.25	
27	20:30	wild rice	cup	0.5	

APPENDIX C (continued)

27	20:30	bulgar wheat	cup	0.25	
27	20:30	basmati rice	cup	0.5	
27	20:30	squash (butternut)	cup	0.25	
27	20:30	turkey	oz	2	
27	20:30	raisin	cup	0.25	
27	20:30	mint	tbsp	0.5	
27	20:30	candy (Jolly Rancher)	pc	2	
28	6:00	H2O	oz	8	28-Jan
28	9:25	banana	cup	1.25	
28	9:25	H2O	oz	8	
28	10:30	steel cut oatmeal	cup	1	
28	10:30	strawberry	cup	0.125	
28	10:30	almond	cup	0.125	
28	10:30	egg	cup	0.125	
28	10:30	spinach	cup	0.125	
28	10:30	sweet potato	cup	0.125	
28	12:30	bread (wheat)	oz	4	
28	12:30	cheese (mozzarella)	oz	3	
28	12:30	cheese (feta)	oz	2	
28	12:30	tomato sauce	cup	0.125	
28	12:30	olive oil	tbsp	0.5	
28	12:30	spinach	cup	0.125	
28	12:30	black olive	cup	0.125	
28	12:30	sundried tomato	cup	0.25	
28	15:00	H2O	oz	8	
28	15:45	candy (Sour Patch Xploderz)	pc	10	
28	16:30	candy (Sour Patch Xploderz)	pc	10	
28	17:45	H2O	oz	16	
28	19:45	candy (Sour Patch Xploderz)	pc	5	
28	20:00	beef	oz	8	
28	20:00	wine (red)	oz	4	
28	20:00	dijon mustard	tbsp	1	
28	20:00	thyme	tbsp	0.25	
28	20:00	sage	tbsp	0.125	
28	20:00	mushroom (portabella)	cup	0.25	
28	20:00	salt	tsp	0.125	
28	20:00	pepper	tbsp	0.25	
28	20:00	squash (butternut)	cup	0.75	
28	20:00	potato	cup	0.25	
28	20:00	celery	cup	0.125	

APPENDIX C (continued)

28	20:00	carrot	cup	0.25	
28	20:00	mint	tbsp	0.125	
28	20:00	tarragon	tbsp	0.125	
28	20:00	vegitable broth	cup	0.75	
28	20:00	clove	tbsp	0.125	
28	20:00	brussel sprout	cup	1.5	
28	20:00	juice (lemon)	tbsp	0.75	
28	20:00	lemon zest	tbsp	0.125	
28	20:00	wine (white)	tbsp	2	
28	20:00	garlic	tbsp	0.5	
28	20:00	olive oil	tbsp	1	
28	20:00	poppy seed	tbsp	0.125	
29	0:10	cake (almond flour)	oz	6	29-Jan
29	0:10	milk chocolate	pc	12	
29	6:45	H2O	oz	2	
29	11:05	banana	cup	1.25	
29	11:05	H2O	oz	6	
29	11:55	coffee	oz	8	
29	11:55	sugar	tbsp	1	
29	12:00	candy (Sour Patch Xploderz)	pc	10	
29	12:30	egg	oz	4	
29	12:30	capsicum (red)	cup	0.5	
29	12:30	capsicum (green)	cup	0.5	
29	12:30	tomatillo	cup	0.5	
29	12:30	salt	tsp	0.125	
29	12:30	tarragon	tbsp	0.25	
29	14:30	avocado	cup	0.25	
29	14:30	onion (white)	cup	0.125	
29	14:30	pork (Torchy's)	oz	3	
29	14:30	cilantro	tbsp	0.5	
29	14:30	tomatillo salsa	tbsp	1	
29	14:30	tortilla (corn)	pc	2	
29	14:30	H2O	oz	16	
29	15:20	coffee	oz	8	
29	15:20	H2O	oz	16	
29	16:30	H2O	oz	16	
29	18:00	pork	oz	6	
29	18:00	rosemary	tbsp	0.25	
29	18:00	jelly (mint)	tbsp	1	
29	18:00	soft drink (Vitamin Water XXX)	oz	16	

APPENDIX C (continued)

29	19:30	bread (white)	oz	5	
29	19:30	squash (butternut)	cup	0.5	
29	19:30	potato	cup	0.125	
29	19:30	celery	cup	0.05	
29	19:30	carrot	cup	0.1	
29	19:30	mint	tbsp	0.05	
29	19:30	tarragon	tbsp	0.05	
29	19:30	vegitable broth	cup	0.125	
29	19:30	clove	tbsp	0.05	
29	21:00	banana	cup	0.5	
29	22:15	potato	cup	1	
29	22:15	canola oil	tbsp	2	
29	22:15	salt	tsp	0.25	
29	22:15	bread (wheat)	oz	2	
29	22:15	lettuce (romaine)	cup	0.25	
29	22:15	tomato	cup	0.25	
29	22:15	hamburger	oz	3	
29	22:15	sauce (P Terry's)	tbsp	1	
29	22:15	tea (black)	oz	24	
30	3:20	banana	cup	0.75	30-Jan
30	8:20	banana	cup	1.5	
30	8:40	coffee	oz	8	
30	8:40	sugar	tbsp	1	
30	9:00	banana	cup	0.5	
30	9:00	strawberry	cup	1.5	
30	9:00	juice (apple)	oz	8	
30	10:20	steel cut oatmeal	cup	2	
30	10:20	strawberry	cup	0.125	
30	10:20	almond	cup	0.05	
30	11:30	H2O	oz	6	
30	13:00	avocado	cup	0.5	
30	13:00	onion (white)	cup	0.25	
30	13:00	pork (Torchy's)	oz	6	
30	13:00	cilantro	tbsp	1	
30	13:00	tomatillo salsa	tbsp	2	
30	13:00	tortilla (corn)	pc	4	
30	13:00	milk chocolate	pc	10	
30	13:00	banana	cup	0.25	
30	13:00	H2O	oz	32	
30	14:25	milk chocolate	pc	10	

APPENDIX C (continued)

30	14:25	banana	cup	0.25	
30	16:00	avocado	cup	0.25	
30	16:00	onion (white)	cup	0.125	
30	16:00	pork (Torchy's)	oz	3	
30	16:00	cilantro	tbsp	0.5	
30	16:00	tomatillo salsa	tbsp	1	
30	16:00	tortilla (corn)	pc	2	
30	16:00	milk chocolate	pc	10	
30	23:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
30	23:30	sugar	tbsp	2	
31	8:30	banana	cup	0.5	31-Jan
31	8:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
31	8:30	sugar	tbsp	1	
31	9:30	avocado	cup	0.25	
31	9:30	onion (white)	cup	0.125	
31	9:30	pork (Torchy's)	oz	3	
31	9:30	cilantro	tbsp	0.5	
31	9:30	tomatillo salsa	tbsp	1	
31	9:30	tortilla (corn)	pc	2	
31	9:30	H2O	oz	16	
31	9:30	milk chocolate	pc	10	
31	9:30	banana	cup	0.25	
31	11:50	pork	oz	6	
31	11:50	capsicum (red)	cup	0.5	
31	11:50	capsicum (green)	cup	0.75	
31	11:50	onion (green)	cup	0.125	
31	11:50	cumin	tbsp	0.5	
31	11:50	coriander	tbsp	0.25	
31	11:50	salt	tsp	0.125	
31	11:50	pepper	tsp	0.25	
31	11:50	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
31	11:50	sugar	tbsp	1	
31	12:40	milk chocolate	pc	10	
31	14:35	milk chocolate	pc	5	
31	16:30	juice (apple)	pc	1	
31	16:30	juice (kiwi)	pc	1	
31	16:30	juice (celery)	pc	1	
31	16:30	juice (cucumber)	pc	1	

APPENDIX C (continued)

31	16:30	juice (fennel stalk)	pc	1	
31	16:30	juice (carrot)	pc	2	
31	17:10	milk chocolate	pc	8	
31	17:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
31	17:45	banana	cup	0.5	
31	20:40	tilapia	oz	8	
31	20:40	garlic	tbsp	0.5	
31	20:40	olive oil	tbsp	0.5	
31	20:40	dill	tbsp	0.5	
31	20:40	salt	tsp	0.125	
31	20:40	cumin	tbsp	0.125	
31	20:40	cinnamon	tbsp	0.05	
31	20:40	pepper	tbsp	0.125	
31	20:40	tortilla (corn)	pc	3	
31	20:40	fennel	cup	0.25	
31	20:40	cabbage (green)	cup	0.5	
31	20:40	sour cream	cup	0.1	
31	20:40	honey chipotle powder	tbsp	0.5	
31	20:40	H2O	oz	10	
31	20:40	milk chocolate	pc	8	
31	20:40	banana	cup	0.25	
31	20:40	juice (lime)	tbsp	0.5	
32	9:30	avocado	cup	0.25	1-Feb
32	9:30	onion (white)	cup	0.125	
32	9:30	pork (Torchy's)	oz	3	
32	9:30	cilantro	tbsp	0.5	
32	9:30	tomatillo salsa	tbsp	1	
32	9:30	tortilla (corn)	pc	2	
32	9:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
32	9:30	sugar	tbsp	1	
32	10:10	milk chocolate	pc	15	
32	11:25	banana	cup	0.5	
32	12:20	avocado	cup	0.25	
32	12:20	onion (white)	cup	0.125	
32	12:20	pork (Torchy's)	oz	3	
32	12:20	cilantro	tbsp	0.5	
32	12:20	tomatillo salsa	tbsp	2.5	
32	12:20	tortilla (corn)	pc	2	

APPENDIX C (continued)

32	13:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
32	13:45	honey	tbsp	1	
32	14:10	milk chocolate	pc	5	
32	15:20	banana	cup	0.75	
32	16:15	milk chocolate	pc	3	
32	16:45	tilapia	oz	8	
32	16:45	garlic	tbsp	0.5	
32	16:45	olive oil	tbsp	0.5	
32	16:45	salt	tsp	0.125	
32	16:45	cumin	tbsp	0.125	
32	16:45	cinnamon	tbsp	0.05	
32	16:45	pepper	tbsp	0.125	
32	16:45	tortilla (corn)	pc	2	
32	16:45	fennel	cup	0.25	
32	16:45	cabbage (green)	cup	0.5	
32	16:45	sour cream	cup	0.1	
32	16:45	honey chipotle powder	tbsp	0.5	
32	16:45	juice (lime)	tbsp	0.5	
32	16:45	H2O	oz	16	
32	18:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
32	18:15	honey	tbsp	1	
32	20:30	salmon	oz	6	
32	20:30	cumin	tbsp	0.125	
32	20:30	honey chipotle powder	tbsp	0.25	
32	20:30	brown sugar	tbsp	1	
32	20:30	salt	tsp	0.125	
32	20:30	garlic	tbsp	0.5	
32	20:30	jelly (triple berry)	tbsp	2	
32	20:30	juice (lime)	tbsp	0.25	
32	20:30	couscous	cup	1	
32	20:30	curry powder	tbsp	0.5	
32	20:30	H2O	oz	16	
32	20:30	asparagus	cup	1	
32	20:30	lemon pepper	tbsp	0.25	
32	21:30	candy (Jolly Rancher)	pc	2	
32	22:20	banana	cup	0.75	
33	8:20	banana	cup	0.75	2-Feb
33	8:20	milk chocolate	pc	10	

APPENDIX C (continued)

33	8:20	coffee	oz	8	
33	11:15	milk chocolate	pc	10	
33	12:20	pork	oz	6	
33	12:20	tarragon	tbsp	0.25	
33	12:20	jelly (mint)	tbsp	1	
33	12:20	salt	tsp	0.125	
33	14:30	milk chocolate	pc	5	
33	17:40	banana	cup	1.5	
33	20:00	tilapia	oz	8	
33	20:00	garlic	tbsp	0.5	
33	20:00	olive oil	tbsp	0.5	
33	20:00	salt	tsp	0.125	
33	20:00	cumin	tbsp	0.125	
33	20:00	cinnamon	tbsp	0.05	
33	20:00	pepper	tbsp	0.125	
33	20:00	tortilla (corn)	pc	4	
33	20:00	fennel	cup	0.25	
33	20:00	cabbage (green)	cup	0.5	
33	20:00	sour cream	cup	0.1	
33	20:00	honey chipotle powder	tbsp	0.5	
33	20:00	juice (lime)	tbsp	0.5	
33	22:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
33	22:30	honey	tbsp	1	
34	0:15	banana	cup	0.5	3-Feb
34	0:15	H2O	oz	8	
34	9:15	H2O	oz	8	
34	9:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	6	
34	9:15	banana	cup	1.25	
34	11:00	avocado	cup	0.5	
34	11:00	onion (white)	cup	0.25	
34	11:00	pork (Torchy's)	oz	6	
34	11:00	cilantro	tbsp	1	
34	11:00	tomatillo salsa	tbsp	4	
34	11:00	tortilla (corn)	pc	4	
34	11:00	milk chocolate	pc	18	
34	12:50	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
34	12:50	milk chocolate	pc	5	
34	12:50	juice (apple)	pc	2	

APPENDIX C (continued)

34	12:50	juice (kiwi)	pc	1	
34	12:50	juice (cucumber)	pc	1	
34	12:50	juice (fennel stalk)	pc	1	
34	16:00	avocado	cup	0.25	
34	16:00	onion (white)	cup	0.125	
34	16:00	pork (Torchy's)	oz	3	
34	16:00	cilantro	tbsp	0.5	
34	16:00	tomatillo salsa	tbsp	3	
34	16:00	tortilla (corn)	pc	2	
34	16:00	milk chocolate	pc	5	
34	18:00	H2O	oz	4	
34	18:00	milk chocolate	pc	8	
34	19:00	coffee	oz	4	
34	20:15	egg	oz	4	
34	20:15	ham	oz	3	
34	20:15	capsicum (red)	cup	0.5	
34	20:15	capsicum (green)	cup	0.5	
34	20:15	honey	tbsp	1	
34	20:15	salt	tsp	0.125	
34	20:15	butter	tbsp	0.5	
34	21:30	banana	cup	0.75	
34	22:00	H2O	oz	16	
34	22:30	banana	cup	0.75	
35	8:15	avocado	cup	0.25	4-Feb
35	8:15	onion (white)	cup	0.125	
35	8:15	pork (Torchy's)	oz	3	
35	8:15	cilantro	tbsp	0.5	
35	8:15	tomatillo salsa	tbsp	2	
35	8:15	tortilla (corn)	pc	2	
35	8:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
35	8:15	honey	tbsp	1	
35	8:15	milk chocolate	pc	10	
35	9:45	H2O	oz	20	
35	10:50	avocado	cup	0.25	
35	10:50	onion (white)	cup	0.125	
35	10:50	pork (Torchy's)	oz	3	
35	10:50	cilantro	tbsp	0.5	
35	10:50	tomatillo salsa	tbsp	2	
35	10:50	tortilla (corn)	pc	2	

APPENDIX C (continued)

35	10:50	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
35	10:50	milk chocolate	pc	8	
35	14:00	egg	oz	4	
35	14:00	ham	oz	3	
35	14:00	capsicum (red)	cup	0.5	
35	14:00	capsicum (green)	cup	0.5	
35	14:00	onion (green)	cup	0.125	
35	14:00	salt	tsp	0.125	
35	14:00	pepper	tsp	0.125	
35	14:00	bread (corn)	oz	4	
35	14:00	butter	tbsp	0.25	
35	14:00	honey	tbsp	1	
35	14:00	olive oil	tbsp	0.25	
35	14:00	coffee	oz	8	
35	14:00	sugar	tbsp	0.5	
35	15:30	H2O	oz	16	
35	18:20	milk chocolate	oz	8	
35	20:10	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
35	20:10	honey	tbsp	2	
35	20:10	bread (corn)	oz	2	
35	20:10	butter	tbsp	0.125	
35	21:20	milk chocolate	pc	10	
35	21:20	banana	cup	0.25	
36	8:00	H2O	oz	14	5-Feb
36	8:00	milk chocolate	pc	16	
36	8:00	banana	cup	0.25	
36	9:20	banana	cup	0.75	
36	10:10	milk chocolate	pc	4	
36	10:10	H2O	oz	16	
36	11:20	avocado	cup	0.25	
36	11:20	onion (white)	cup	0.125	
36	11:20	pork (Torchy's)	oz	3	
36	11:20	cilantro	tbsp	0.5	
36	11:20	tomatillo salsa	tbsp	2	
36	11:20	tortilla (corn)	pc	2	
36	11:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
36	11:20	honey	oz	1	
36	13:20	banana	cup	0.75	

APPENDIX C (continued)

36	15:30	potato	cup	1	
36	15:30	canola oil	tbsp	2	
36	15:30	salt	tsp	0.25	
36	15:30	bread (wheat)	oz	2	
36	15:30	lettuce (romaine)	cup	0.25	
36	15:30	tomato	cup	0.25	
36	15:30	hamburger	oz	3	
36	15:30	sauce (P Terry's)	tbsp	1	
36	15:30	tea (black)	oz	24	
36	16:10	banana	cup	0.75	
36	16:10	milk chocolate	pc	15	
36	19:45	pork	oz	6	
36	19:45	green salsa	tbsp	6	
36	19:45	green chili	tbsp	3	
36	19:45	onion (white)	cup	0.1	
36	19:45	cilantro	tbsp	3	
36	19:45	avocado	cup	0.75	
36	19:45	tortilla (corn)	pc	3	
36	20:00	milk chocolate	pc	15	
36	20:00	H2O	oz	16	
36	22:00	banana	cup	0.75	
36	23:15	H2O	oz	16	
37	5:20	H2O	oz	16	6-Feb
37	5:20	banana	cup	0.75	
37	8:20	H2O	oz	12	
37	9:20	pork	oz	4	
37	9:20	green salsa	tbsp	4	
37	9:20	green chili	tbsp	2	
37	9:20	onion (white)	cup	0.5	
37	9:20	garlic	tbsp	0.5	
37	9:20	cilantro	tbsp	2	
37	9:20	avocado	cup	0.5	
37	9:20	tortilla (corn)	pc	4	
37	9:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
37	9:20	honey	tbsp	1	
37	9:50	milk chocolate	pc	8	
37	11:15	H2O	oz	8	
37	11:30	banana	cup	0.75	
37	13:00	coffee	oz	8	

APPENDIX C (continued)

37	13:00	sugar	tbsp	1	
37	14:20	banana	cup	0.75	
37	15:15	pork	oz	6	
37	15:15	tarragon	tbsp	0.5	
37	15:15	honey	tbsp	1	
37	15:15	jelly (mint)	tbsp	1	
37	16:10	H2O	oz	16	
37	17:10	banana	cup	0.5	
37	17:40	pork	oz	6	
37	17:40	green salsa	tbsp	6	
37	17:40	green chili	tbsp	3	
37	17:40	onion (white)	tbsp	0.75	
37	17:40	garlic	tbsp	0.75	
37	17:40	cilantro	tbsp	2	
37	17:40	avocado	cup	0.5	
37	17:40	tortilla (corn)	pc	3	
37	18:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
37	18:30	honey	tbsp	1	
37	20:00	corned beef	oz	5	
37	20:00	bread (white)	oz	4	
37	20:00	dijon mustard	tbsp	3	
37	20:00	black olive	cup	0.25	
37	20:00	capsicum (green)	cup	0.25	
37	20:00	capsicum (red)	cup	0.25	
38	3:00	H2O	oz	16	7-Feb
38	5:45	H2O	oz	8	
38	5:45	banana	cup	0.5	
38	6:30	banana	cup	0.5	
38	7:20	H2O	oz	8	
38	8:45	banana	cup	0.5	
38	10:45	coffee	oz	4	
38	10:45	avocado	cup	0.5	
38	10:45	onion (white)	cup	0.25	
38	10:45	pork (Torchy's)	oz	6	
38	10:45	cilantro	tbsp	1	
38	10:45	tomatillo salsa	tbsp	2	
38	10:45	tortilla (corn)	pc	4	
38	10:45	juice (lime)	tbsp	1	
38	10:45	milk chocolate	pc	10	

APPENDIX C (continued)

38	10:45	raisin	cup	0.25	
38	13:00	cake (white)	oz	3	
38	13:00	juice (lemon)	tbsp	1	
38	13:00	poppy seed	tbsp	0.5	
38	13:00	sugar	tbsp	3	
38	13:00	coffee	oz	8	
38	15:00	milk chocolate	pc	5	
38	15:00	raisin	cup	0.125	
38	15:15	onion (white)	cup	0.125	
38	15:15	pork (Torchy's)	oz	3	
38	15:15	cilantro	tbsp	0.5	
38	15:15	tomatillo salsa	tbsp	2	
38	15:15	tortilla (corn)	pc	2	
38	15:15	banana	cup	0.5	
38	15:15	H2O	oz	16	
38	16:25	H2O	oz	16	
38	17:35	milk chocolate	pc	3	
38	17:35	raisin	cup	0.1	
38	17:50	coffee	oz	3	
38	18:30	banana	cup	0.5	
38	18:40	coffee	oz	5	
38	18:40	sugar	tbsp	1	
38	19:40	milk chocolate	pc	5	
38	19:40	raisin	cup	0.125	
38	20:30	onion (white)	cup	0.125	
38	20:30	pork (Torchy's)	oz	3	
38	20:30	cilantro	tbsp	0.5	
38	20:30	tomatillo salsa	tbsp	2	
38	20:30	tortilla (corn)	pc	2	
38	20:30	H2O	oz	16	
38	20:50	milk chocolate	pc	10	
38	20:50	raisin	cup	0.25	
38	22:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
38	22:00	honey	tbsp	1	
38	22:20	banana	cup	0.5	
38	23:00	candy (Jolly Rancher)	pc	2	
39	6:50	H2O	oz	8	8-Feb
39	7:55	banana	cup	0.5	
39	9:00	pork	oz	4	

APPENDIX C (continued)

39	9:00	green salsa	tbsp	4
39	9:00	green chili	tbsp	2
39	9:00	onion (white)	cup	0.1
39	9:00	cilantro	tbsp	2
39	9:00	avocado	cup	0.5
39	9:00	tortilla (corn)	pc	2
39	9:00	milk chocolate	pc	8
39	9:00	raisin	cup	0.25
39	10:15	coffee	oz	3
39	10:45	coffee	oz	5
39	10:45	sugar	tbsp	1
39	12:40	milk chocolate	pc	9
39	12:40	raisin	cup	0.25
39	14:20	banana	cup	1
39	15:00	milk chocolate	pc	5
39	15:00	raisin	cup	0.125
39	16:35	H2O	oz	16
39	16:35	avocado	cup	0.5
39	16:35	onion (white)	cup	0.125
39	16:35	pork (Torchy's)	oz	3
39	16:35	cilantro	tbsp	0.5
39	16:35	tomatillo salsa	tbsp	1
39	16:35	tortilla (corn)	pc	2
39	17:50	H2O	oz	16
39	19:30	salmon	oz	5
39	19:30	jelly (triple berry)	tbsp	2
39	19:30	juice (lime)	tbsp	0.5
39	19:30	honey chipotle powder	tbsp	0.5
39	19:30	brown sugar	tbsp	1
39	19:30	cumin	tbsp	0.25
39	19:30	asparagus	cup	1.25
39	19:30	garlic pepper	tbsp	0.25
39	19:30	salt	tsp	0.125
39	19:30	banana	cup	0.5
39	20:10	milk chocolate	pc	4
39	20:10	raisin	cup	0.125
39	20:10	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8
39	20:10	honey	tbsp	1
39	20:30	milk chocolate	pc	8

APPENDIX C (continued)

39	20:30	raisin	cup	0.25	
39	22:10	H2O	oz	16	
39	22:10	banana	cup	0.5	
39	23:20	milk chocolate	pc	4	
39	23:20	raisin	cup	0.125	
40	7:45	banana	cup	0.5	9-Feb
40	7:45	H2O	oz	16	
40	7:45	coffee	oz	8	
40	7:45	sugar	tbsp	1	
40	8:20	milk chocolate	pc	5	
40	8:20	raisin	cup	0.125	
40	9:35	avocado	cup	0.5	
40	9:35	onion (white)	cup	0.125	
40	9:35	pork (Torchy's)	oz	3	
40	9:35	cilantro	tbsp	0.5	
40	9:35	tomatillo salsa	tbsp	2	
40	9:35	tortilla (corn)	pc	2	
40	9:35	strawberry	cup	1.5	
40	9:35	sugar	tbsp	1.5	
40	12:10	banana	cup	0.5	
40	12:10	coffee	oz	8	
40	12:10	sugar	tbsp	0.5	
40	12:10	pork	oz	5	
40	12:10	rosemary	tbsp	0.3	
40	12:10	sage	tbsp	0.2	
40	12:10	jelly (mint)	tbsp	1	
40	12:10	H2O	oz	16	
40	13:00	milk chocolate	pc	5	
40	13:00	raisin	cup	0.25	
40	14:25	milk chocolate	pc	4	
40	14:25	raisin	cup	0.25	
40	15:30	coffee	oz	8	
40	15:30	sugar	tbsp	1	
40	15:30	H2O	oz	16	
40	16:30	milk chocolate	pc	3	
40	16:30	raisin	cup	0.125	
40	17:30	milk chocolate	pc	3	
40	17:30	raisin	cup	0.125	
40	18:10	bread (soda)	oz	3	
40	18:10	sugar	tbsp	2	

APPENDIX C (continued)

40	18:10	cranberry	cup	0.125	
40	18:10	orange	cup	0.05	
40	18:10	H2O	oz	16	
40	20:20	tilapia	oz	3	
40	20:20	garlic	tbsp	0.375	
40	20:20	cumin	tbsp	0.15	
40	20:20	cinnamon	tbsp	0.15	
40	20:20	olive oil	tbsp	0.375	
40	20:20	pepper	tsp	0.15	
40	20:20	salt	tsp	0.15	
40	20:20	juice (lime)	tbsp	0.15	
40	20:20	fennel	cup	0.75	
40	20:20	cabbage (green)	cup	0.75	
40	20:20	sour cream	cup	0.15	
40	20:20	honey chipotle powder	tbsp	0.75	
40	20:20	tortilla (corn)	pc	3	
40	20:20	cilantro	tbsp	0.375	
40	20:20	avocado	cup	0.5	
40	20:20	tomato	cup	0.125	
40	20:20	cilantro	tbsp	0.25	
40	20:20	onion (white)	cup	0.125	
41	6:15	H2O	oz	2	10-Feb
41	6:50	banana	cup	0.75	
41	7:30	milk chocolate	pc	6	
41	7:30	raisin	cup	0.25	
41	9:55	banana	cup	1.5	
41	9:55	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
41	9:55	honey	tbsp	1	
41	10:20	milk chocolate	pc	10	
41	10:20	raisin	cup	0.25	
41	12:15	H2O	oz	16	
41	12:15	avocado	cup	0.5	
41	12:15	onion (white)	cup	0.25	
41	12:15	pork (Torchy's)	oz	6	
41	12:15	cilantro	tbsp	1	
41	12:15	tomatillo salsa	tbsp	2	
41	12:15	tortilla (corn)	pc	4	
41	12:15	soda (Mt Dew Code Red)	oz	20	
41	14:00	H2O	oz	10	

APPENDIX C (continued)

41	14:00	avocado	cup	0.25
41	14:00	onion (white)	cup	0.125
41	14:00	pork (Torchy's)	oz	3
41	14:00	cilantro	tbsp	0.5
41	14:00	tomatillo salsa	tbsp	1
41	14:00	tortilla (corn)	pc	2
41	14:15	dark chocolate	pc	3
41	14:15	cranberry	cup	0.125
41	14:45	dark chocolate	pc	4
41	14:45	cranberry	cup	0.125
41	16:00	bread (soda)	oz	4
41	16:00	sugar	tbsp	3
41	16:00	cranberry	cup	0.125
41	16:00	orange	cup	0.05
41	16:00	butter	tbsp	0.25
41	16:00	H2O	oz	20
41	16:00	banana	cup	1.25
41	16:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8
41	16:00	honey	tbsp	1
41	18:10	pork	oz	3
41	18:10	rosemary	tbsp	0.125
41	18:10	sage	tbsp	0.125
41	18:10	jelly (mint)	tbsp	0.5
41	18:10	salt	tsp	0.125
41	18:10	strawberry	cup	1
41	18:10	sugar	tbsp	1
41	18:10	H2O	oz	10
41	21:20	asparagus	cup	1
41	21:20	butter	tbsp	1
41	21:20	garlic pepper	tbsp	0.25
41	21:20	salt	tsp	0.125
41	22:20	dark chocolate	pc	5
41	22:20	cranberry	cup	0.125
41	22:30	cucumber	cup	0.5
41	22:30	seaweed	cup	0.25
41	22:30	vinegar (rice wine)	tbsp	3
41	22:30	sugar	tbsp	1
41	22:30	sesame seed	tbsp	0.5
41	22:30	snow crab	oz	2

APPENDIX C (continued)

41	23:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
41	23:30	honey	tbsp	1	
41	23:30	dark chocolate	pc	3	
41	23:30	cranberry	cup	0.125	
41	23:30	soda (La Croix coconut)	oz	12	
42	6:05	H2O	oz	4	11-Feb
42	9:25	avocado	cup	0.25	
42	9:25	onion (white)	cup	0.125	
42	9:25	pork (Torchy's)	oz	3	
42	9:25	cilantro	tbsp	0.5	
42	9:25	tomatillo salsa	tbsp	1	
42	9:25	tortilla (corn)	pc	2	
42	9:25	banana	cup	0.75	
42	9:25	H2O	oz	20	
42	10:20	dark chocolate	pc	3	
42	10:20	cranberry	cup	0.125	
42	12:45	soda (La Croix coconut)	oz	12	
42	12:45	pork	oz	5	
42	12:45	onion (white)	cup	0.25	
42	12:45	capsicum (red)	cup	0.25	
42	12:45	capsicum (green)	cup	0.25	
42	12:45	spinach	cup	0.25	
42	12:45	cumin	tbsp	0.25	
42	12:45	coriander	tbsp	0.25	
42	12:45	avocado	cup	0.5	
42	12:45	salt	tsp	0.125	
42	12:45	tortilla (corn)	pc	3	
42	14:50	juice (cucumber)	pc	1	
42	14:50	juice (plum)	pc	2	
42	14:50	juice (peach)	pc	1	
42	14:50	juice (strawberry)	pc	10	
42	15:50	avocado	cup	0.25	
42	15:50	onion (white)	cup	0.125	
42	15:50	pork (Torchy's)	oz	3	
42	15:50	cilantro	tbsp	0.5	
42	15:50	tomatillo salsa	tbsp	1	
42	15:50	tortilla (corn)	pc	2	
42	15:50	dark chocolate	pc	4	
42	15:50	cranberry	cup	0.125	

APPENDIX C (continued)

42	16:30	dark chocolate	pc	6	
42	16:30	cranberry	cup	0.25	
42	18:30	dark chocolate	pc	4	
42	18:30	cranberry	cup	0.125	
42	19:00	H2O	oz	2	
42	19:50	banana	cup	0.75	
42	21:00	tea (Honest Tea honey green)	oz	16	
42	21:00	salmon	oz	8	
42	21:00	olive oil	tbsp	1	
42	21:00	dill	tbsp	1	
42	21:00	fennel	cup	0.75	
42	21:00	cabbage (green)	cup	0.5	
42	21:00	sour cream	cup	1.25	
42	21:00	honey chipotle powder	tbsp	1	
42	21:00	tortilla (corn)	pc	3	
42	21:30	dark chocolate	pc	4	
42	21:30	cranberry	cup	0.125	
43	6:00	H2O	oz	4	12-Feb
43	7:00	banana	cup	1	
43	7:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
43	7:00	honey	tbsp	1	
43	8:25	avocado	cup	0.25	
43	8:25	onion (white)	cup	0.125	
43	8:25	pork (Torchy's)	oz	3	
43	8:25	cilantro	tbsp	0.5	
43	8:25	tomatillo salsa	tbsp	3	
43	8:25	tortilla (corn)	pc	2	
43	8:25	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
43	8:25	honey	tbsp	1	
43	8:50	dark chocolate	pc	4	
43	8:50	cranberry	cup	0.125	
43	12:30	egg	oz	6	
43	12:30	capsicum (red)	cup	0.4	
43	12:30	capsicum (green)	cup	0.4	
43	12:30	ham	oz	2	
43	12:30	spinach	cup	0.25	
43	12:30	rosemary	tbsp	0.125	
43	12:30	tomatillo salsa	tbsp	3	

APPENDIX C (continued)

43	12:30	tortilla (corn)	pc	3	
43	12:30	onion (green)	cup	0.125	
43	12:30	coffee	oz	4	
43	12:30	sugar	tbsp	0.5	
43	15:20	coffee	oz	4	
43	15:20	sugar	tbsp	0.5	
43	15:20	H2O	oz	16	
43	15:40	dark chocolate	pc	6	
43	15:40	cranberry	cup	0.125	
43	21:00	tea (Lipton Green Tea - decaf)	oz	8	
43	21:00	honey	tbsp	1.5	
43	21:15	banana	cup	1	
43	22:00	H2O	oz	4	
44	5:35	H2O	oz	4	13-Feb
44	5:50	banana	cup	1	
44	9:40	banana	cup	1	
44	9:40	tea (Lipton Green Tea - decaf)	oz	8	
44	9:40	honey	tbsp	1	
44	12:00	pork	oz	5	
44	12:00	thyme	tbsp	0.25	
44	12:00	jelly (mint)	tbsp	1	
44	12:00	soda (La Croix coconut)	oz	12	
44	13:30	banana	cup	2.5	
44	13:30	strawberry	cup	1.5	
44	13:30	honey	tbsp	2	
44	13:30	H2O	oz	16	
44	15:45	avocado	cup	0.5	
44	15:45	onion (white)	cup	0.25	
44	15:45	pork (Torchy's)	oz	6	
44	15:45	cilantro	tbsp	1	
44	15:45	tomatillo salsa	tbsp	2	
44	15:45	tortilla (corn)	pc	4	
44	15:45	juice (lime)	tbsp	0.5	
44	15:45	H2O	oz	16	
44	18:15	asparagus	cup	1.5	
44	18:15	salt	tsp	0.125	
44	18:15	pepper	tsp	0.125	
44	18:15	soda (La Croix coconut)	oz	12	
44	18:15	butter	tbsp	0.25	
44	19:10	banana	cup	1	

APPENDIX C (continued)

44	19:10	candy (Jolly Rancher)	pc	2	
44	19:10	tea (Lipton Green Tea - decaf)	oz	8	
44	19:10	honey	tbsp	1.5	
44	21:25	banana	cup	1	
44	22:20	tea (black)	oz	16	
44	22:20	potato	cup	0.25	
44	22:20	canola oil	tbsp	0.5	
44	22:20	salt	tsp	0.1	
44	22:20	candy (Jolly Rancher)	pc	2	
45	5:20	H2O	oz	10	14-Feb
45	5:20	banana	cup	0.5	
45	8:20	banana	cup	0.5	
45	8:20	H2O	oz	8	
45	9:30	avocado	cup	0.25	
45	9:30	onion (white)	cup	0.125	
45	9:30	pork (Torchy's)	oz	3	
45	9:30	cilantro	tbsp	0.5	
45	9:30	tomatillo salsa	tbsp	2	
45	9:30	tortilla (corn)	pc	2	
45	9:30	juice (lime)	tbsp	0.5	
45	9:30	tea (Lipton Green Tea - decaf)	oz	9	
45	9:30	honey	tbsp	1	
45	12:10	banana	cup	1	
45	12:10	avocado	cup	0.25	
45	12:10	onion (white)	cup	0.125	
45	12:10	pork (Torchy's)	oz	3	
45	12:10	cilantro	tbsp	0.5	
45	12:10	tomatillo salsa	tbsp	2	
45	12:10	tortilla (corn)	pc	2	
45	13:20	gum (Doublemint)	pc	2	
45	13:40	H2O	oz	14	
45	14:50	juice (Odwala Blueberry B)	oz	12	
45	14:50	banana	cup	1	
45	17:15	banana	cup	1	
45	17:15	soda (La Croix coconut)	oz	12	
45	18:30	avocado	cup	0.25	
45	18:30	onion (white)	cup	0.125	
45	18:30	pork (Torchy's)	oz	3	
45	18:30	cilantro	tbsp	0.5	
45	18:30	tomatillo salsa	tbsp	2	

APPENDIX C (continued)

45	18:30	tortilla (corn)	pc	2	
45	18:30	juice (lime)	tbsp	0.5	
45	18:30	tea (Lipton Green Tea - decaf)	oz	12	
45	18:30	honey	tbsp	2	
45	20:30	carrot	cup	0.75	
45	20:30	celery	cup	0.25	
45	20:30	hummus	cup	0.5	
45	20:30	chip (corn tortilla)	cup	0.5	
45	20:30	avocado	cup	1	
45	20:30	tomato	cup	0.25	
45	20:30	onion (white)	cup	0.125	
45	20:30	garlic pepper	tbsp	0.25	
45	20:30	strawberry	cup	1.5	
45	20:30	milk chocolate	pc	15	
45	23:00	mint chocolate	pc	6	
45	23:00	wafer (chocolate)	oz	2	
46	4:30	banana	cup	1.25	15-Feb
46	7:50	H2O	oz	6	
46	9:00	banana	cup	1.25	
46	9:00	H2O	oz	32	
46	9:00	avocado	cup	0.25	
46	9:00	onion (white)	cup	0.125	
46	9:00	pork (Torchy's)	oz	3	
46	9:00	cilantro	tbsp	0.5	
46	9:00	tomatillo salsa	tbsp	2	
46	9:00	tortilla (corn)	pc	2	
46	10:00	mint chocolate	pc	6	
46	10:00	wafer (chocolate)	oz	2	
46	12:15	pork	oz	4	
46	12:15	capsicum (green)	cup	0.5	
46	12:15	fennel	cup	0.5	
46	12:15	tomatillo	cup	0.25	
46	12:15	cumin	tbsp	0.5	
46	12:15	coriander	tbsp	0.5	
46	12:15	salt	tsp	0.125	
46	12:15	tortilla (corn)	pc	5	
46	12:15	soda (La Croix coconut)	oz	12	
46	12:15	mint chocolate	pc	6	
46	12:15	wafer (chocolate)	oz	2	
46	14:00	milk chocolate	pc	8	

APPENDIX C (continued)

46	14:00	H2O	oz	16	
46	15:45	milk chocolate	pc	6	
46	15:45	banana	cup	1.25	
46	18:30	juice (cucumber)	pc	1	
46	18:30	juice (peach)	pc	1	
46	18:30	juice (plum)	pc	2	
46	18:30	juice (kiwi)	pc	1	
46	18:30	juice (carrot)	pc	2	
46	19:30	tea (Lipton Green Tea - decaf)	oz	9	
46	19:30	honey	tbsp	1	
46	22:00	banana	cup	1.25	
46	22:20	ham	oz	3	
46	22:20	capsicum (green)	cup	0.25	
46	22:20	capsicum (red)	cup	0.25	
46	22:20	spinach	cup	0.125	
46	22:20	rosemary	tbsp	0.125	
46	22:20	tea (Lipton Green Tea - decaf)	oz	9	
46	22:20	honey	tbsp	0.75	
47	3:00	H2O	oz	16	16-Feb
47	5:30	H2O	oz	6	
47	8:25	banana	cup	1.25	
47	8:25	milk chocolate	pc	8	
47	8:25	tea (Lipton Green Tea - decaf)	oz	9	
47	8:25	honey	tbsp	1	
47	9:45	tea (Lipton Green Tea - decaf)	oz	9	
47	9:45	honey	tbsp	0.5	
47	12:00	avocado	cup	0.5	
47	12:00	onion (white)	cup	0.25	
47	12:00	beef (Torchy's)	oz	3	
47	12:00	pork (Torchy's)	oz	3	
47	12:00	cilantro	tbsp	1	
47	12:00	tomatillo salsa	tbsp	4	
47	12:00	tortilla (corn)	pc	4	
47	12:00	H2O	oz	8	
47	12:00	mint chocolate	pc	12	
47	12:00	wafer (chocolate)	oz	4	
47	13:15	banana	cup	1.5	
47	13:15	strawberry	cup	1.5	
47	13:15	honey	tbsp	2	
47	13:15	soda (La Croix coconut)	oz	10	

APPENDIX C (continued)

47	16:20	avocado	cup		0.25	
47	16:20	onion (white)	cup		0.125	
47	16:20	pork (Torchy's)	oz		3	
47	16:20	cilantro	tbsp		0.5	
47	16:20	tomatillo salsa	tbsp	w		
47	16:20	tortilla (corn)	pc		2	
47	16:20	tea (Lipton Green Tea - decaf)	oz		9	
47	16:20	honey	tbsp		0.75	
47	17:20	pork	oz		4	
47	17:20	capsicum (green)	cup		0.5	
47	17:20	fennel	cup		0.5	
47	17:20	tomatillo	cup		0.25	
47	17:20	cumin	tbsp		0.5	
47	17:20	coriander	tbsp		0.5	
47	17:20	salt	tsp		0.125	
47	17:20	tortilla (corn)	pc		3	
47	17:20	avocado	cup		0.75	
47	17:20	H2O	oz		8	
47	18:00	H2O	oz		8	
47	19:20	tea (Lipton Green Tea - decaf)	oz		8	
47	19:20	honey	tbsp		0.5	
47	20:45	banana	cup		0.75	
47	20:50	H2O	oz		16	
47	20:50	tortilla (corn)	pc		4	
47	20:50	pork	oz		8	
47	20:50	onion (white)	cup		0.25	
47	20:50	cilantro	tbsp		0.5	
47	20:50	green chili	cup		0.25	
47	20:50	green salsa	tbsp		4	
48	0:20	H2O	oz		16	17-Feb
48	5:25	banana	cup		0.75	
48	5:25	H2O	oz		4	
48	9:10	avocado	cup		0.25	
48	9:10	onion (white)	cup		0.125	
48	9:10	pork (Torchy's)	oz		3	
48	9:10	cilantro	tbsp		0.5	
48	9:10	tomatillo salsa	tbsp		1	
48	9:10	tortilla (corn)	pc		2	
48	9:10	tea (Lipton Green Tea - decaf)	oz		10	
48	9:10	honey	tbsp		1	

APPENDIX C (continued)

48	10:45	tea (Lipton Green Tea - decaf)	oz	10	
48	10:45	honey	tbsp	1	
48	11:20	H2O	oz	2	
48	12:30	banana	cup	1.5	
48	14:30	milk chocolate	pc	6	
48	14:30	tea (Lipton Green Tea - decaf)	oz	10	
48	14:30	honey	tbsp	0.75	
48	14:55	bread (corn)	oz	6	
48	14:55	butter	tbsp	0.5	
48	14:55	honey	tbsp	0.5	
48	16:55	bread (corn)	oz	3	
48	16:55	butter	tbsp	0.5	
48	16:55	honey	tbsp	0.5	
48	16:55	H2O	oz	16	
48	18:10	H2O	oz	16	
48	18:20	avocado	cup	0.25	
48	18:20	onion (white)	cup	0.125	
48	18:20	pork (Torchy's)	oz	3	
48	18:20	cilantro	tbsp	0.5	
48	18:20	tomatillo salsa	tbsp	1	
48	18:20	tortilla (corn)	pc	2	
48	19:00	bread (corn)	oz	1	
48	19:00	pork	oz	4	
48	19:00	thyme	tbsp	0.25	
48	19:00	jelly (mint)	tbsp	1	
48	19:00	H2O	oz	10	
48	19:00	salt	tsp	0.125	
48	20:40	banana	cup	0.75	
48	22:00	tea (Lipton Green Tea - decaf)	oz	20	
48	22:00	honey	tbsp	2	
48	22:00	bread (corn)	oz	3	
48	22:00	butter	tbsp	0.25	
48	22:00	dark chocolate	pc	2	
49	5:05	H2O	oz	3	18-Feb
49	10:45	pork	oz	5	
49	10:45	capsicum (green)	cup	0.5	
49	10:45	asparagus	cup	0.75	
49	10:45	sage	tbsp	0.25	
49	10:45	vanilla	tbsp	0.5	
49	10:45	salt	tsp	0.125	

APPENDIX C (continued)

49	10:45	pepper	tsp	0.125	
49	10:45	banana	cup	0.5	
49	11:15	H2O	oz	14	
49	14:30	avocado	cup	0.25	
49	14:30	onion (white)	cup	0.125	
49	14:30	pork (Torchy's)	oz	3	
49	14:30	cilantro	tbsp	0.5	
49	14:30	tomatillo salsa	tbsp	1	
49	14:30	tortilla (corn)	pc	2	
49	14:30	juice (lime)	tbsp	0.5	
49	14:30	tea (Lipton Green Tea - decaf)	oz	10	
49	14:30	honey	tbsp	0.75	
49	16:00	avocado	cup	0.25	
49	16:00	onion (white)	cup	0.125	
49	16:00	pork (Torchy's)	oz	3	
49	16:00	cilantro	tbsp	0.5	
49	16:00	tomatillo salsa	tbsp	1	
49	16:00	tortilla (corn)	pc	2	
49	16:00	juice (lime)	tbsp	0.5	
49	16:00	milk chocolate	pc	2	
49	18:50	banana	cup	1.25	
49	18:50	milk chocolate	pc	3	
49	20:20	strawberry	cup	0.75	
49	20:20	sugar	tbsp	1	
49	20:20	tea (Lipton Green Tea - decaf)	oz	4	
49	21:30	tea (Lipton Green Tea - decaf)	oz	6	
49	21:30	honey	tbsp	0.75	
49	22:00	avocado	cup	0.25	
49	22:00	onion (white)	cup	0.125	
49	22:00	pork (Torchy's)	oz	3	
49	22:00	cilantro	tbsp	0.5	
49	22:00	tomatillo salsa	tbsp	1	
49	22:00	tortilla (corn)	pc	2	
49	22:00	juice (lime)	tbsp	0.5	
50	7:10	H2O	oz	8	19-Feb
50	12:45	banana	cup	1.5	
50	13:25	gum (Doublemint)	pc	2	
50	16:20	juice (Odwala Blueberry B)	oz	12	
50	18:50	H2O	oz	3	
50	19:55	honey mustard	tbsp	2	

APPENDIX C (continued)

50	19:55	lettuce (romain)	cup	2	
50	19:55	carrot	cup	0.125	
50	19:55	cabbage (red)	cup	0.125	
50	19:55	shrimp	oz	3	
50	19:55	pasta (rice)	cup	2.5	
50	19:55	juice (lemon)	tbsp	1	
50	19:55	butter	tbsp	2	
50	19:55	cilantro	tbsp	0.125	
50	19:55	tea (Lipton Green Tea - decaf)	oz	10	
50	19:55	honey	tbsp	1	
50	19:55	salt	tsp	0.25	
50	22:05	banana	cup	0.75	
50	22:05	milk chocolate	pc	2	
50	22:05	caramel	pc	4	
50	23:15	H2O	oz	16	
50	23:15	milk chocolate	pc	4	
50	23:15	caramel	pc	4	
51	6:15	H2O	oz	10	20-Feb
51	7:00	banana	cup	0.75	
51	7:50	tea (Lipton Green Tea - decaf)	oz	10	
51	7:50	honey	tbsp	1	
51	10:45	avocado	cup	0.25	
51	10:45	onion (white)	cup	0.125	
51	10:45	pork (Torchy's)	oz	3	
51	10:45	cilantro	tbsp	0.5	
51	10:45	tomatillo salsa	tbsp	2	
51	10:45	tortilla (corn)	pc	2	
51	10:45	tea (Lipton Green Tea - decaf)	oz	10	
51	10:45	honey	tbsp	1	
51	10:45	milk chocolate	pc	2	
51	10:45	caramel	pc	4	
51	12:00	milk chocolate	pc	2	
51	12:00	caramel	pc	4	
51	13:50	milk chocolate	pc	2	
51	13:50	caramel	pc	4	
51	13:50	avocado	cup	0.25	
51	13:50	onion (white)	cup	0.125	
51	13:50	pork (Torchy's)	oz	3	
51	13:50	cilantro	tbsp	0.5	
51	13:50	tomatillo salsa	tbsp	1	

APPENDIX C (continued)

51	13:50	tortilla (corn)	pc	2	
51	13:50	tea (Lipton Green Tea - decaf)	oz	10	
51	13:50	honey	tbsp	1	
51	14:40	candy (Starburst)	pc	8	
51	15:30	banana	cup	0.75	
51	16:30	gum (Doublemint)	pc	2	
51	17:00	bread (pita)	oz	6	
51	17:00	steak	oz	4	
51	17:00	capsicum (green)	cup	0.25	
51	17:00	onion (white)	cup	0.25	
51	17:00	canola oil	tbsp	2	
51	17:00	H2O	oz	16	
51	18:00	potato	cup	0.125	
51	18:00	canola oil	tbsp	0.5	
51	18:00	salt	tsp	0.25	
51	18:00	vinegar (white)	tbsp	1	
51	18:00	candy (Starburst)	pc	4	
51	19:00	asparagus	cup	1	
51	19:00	lemon pepper	tbsp	0.5	
51	19:00	salt	tsp	0.125	
51	19:00	tea (Lipton Green Tea - decaf)	oz	10	
51	19:00	honey	tbsp	1	
51	19:00	H2O	oz	16	
51	21:05	banana	cup	0.75	
51	21:05	carrot	cup	1.5	
51	21:05	chickpea	cup	0.5	
51	21:05	capsicum (red)	cup	0.125	
51	21:05	garlic	tbsp	1	
51	21:05	tahini	tbsp	0.5	
51	21:05	juice (lemon)	tbsp	0.25	
52	4:00	banana	cup	0.75	21-Feb
52	8:40	H2O	oz	10	
52	9:25	avocado	cup	0.25	
52	9:25	onion (white)	cup	0.125	
52	9:25	pork (Torchy's)	oz	3	
52	9:25	cilantro	tbsp	0.5	
52	9:25	tomatillo salsa	tbsp	1	
52	9:25	tortilla (corn)	pc	2	
52	9:25	candy (Starburst)	pc	7	
52	11:40	pork	oz	5	

APPENDIX C (continued)

52	11:40	thyme	tbsp	0.125	
52	11:40	jelly (mint)	tbsp	0.5	
52	11:40	tea (Lipton Green Tea - decaf)	oz	8	
52	11:40	honey	tbsp	0.5	
52	13:45	banana	cup	0.75	
52	13:45	candy (Jolly Rancher)	pc	6	
52	15:20	banana	cup	0.75	
52	15:20	soda (IBC Cream Soda)	oz	12	
52	18:30	banana	cup	0.75	
52	20:30	tilapia	oz	3	
52	20:30	garlic	tbsp	0.375	
52	20:30	cumin	tbsp	0.15	
52	20:30	cinnamon	tbsp	0.15	
52	20:30	olive oil	tbsp	0.375	
52	20:30	pepper	tsp	0.15	
52	20:30	salt	tsp	0.15	
52	20:30	juice (lime)	tbsp	0.15	
52	20:30	fennel	cup	0.75	
52	20:30	cabbage (green)	cup	0.75	
52	20:30	sour cream	cup	0.15	
52	20:30	honey chipotle powder	tbsp	0.75	
52	20:30	tortilla (corn)	pc	3	
52	20:30	soda (La Croix coconut)	oz	12	
52	22:00	tea (Lipton Green Tea Honey / Lemon - decaf)	pc	10	
52	22:00	honey	tbsp	1	
52	23:50	banana	cup	0.5	
52	23:50	candy (Jolly Rancher)	pc	2	
53	6:00	H2O	oz	8	22-Feb
53	7:00	banana	cup	0.5	
53	9:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
53	9:15	honey	tbsp	0.75	
53	10:20	banana	cup	0.5	
53	12:00	pork	oz	4	
53	12:00	capsicum (red)	cup	0.5	
53	12:00	capsicum (green)	cup	0.5	
53	12:00	mint	tbsp	0.5	
53	12:00	tomatillo	cup	0.25	
53	12:00	tortilla (corn)	pc	3	

APPENDIX C (continued)

53	12:00	honey	tbsp	1	
53	12:00	green chili	cup	0.125	
53	12:00	salt	tsp	0.125	
53	12:00	soda (La Croix coconut)	oz	12	
53	15:00	soda (La Croix coconut)	oz	12	
53	15:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
53	15:00	honey	tbsp	1	
53	16:10	banana	cup	0.5	
53	17:30	soda (IBC Cream Soda)	oz	12	
53	19:00	salmon	oz	8	
53	19:00	asparagus	cup	1	
53	19:00	salt	tsp	0.125	
53	19:00	garlic pepper	tbsp	0.25	
53	19:00	butter	tbsp	0.25	
53	19:00	olive oil	tbsp	0.25	
53	19:00	juice (lemon)	tbsp	0.5	
53	21:10	green bean	cup	1	
53	21:10	carrot	cup	0.125	
53	21:10	garlic	tbsp	0.5	
53	21:10	salt	tsp	0.125	
53	21:10	pepper	tsp	0.125	
53	21:10	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
53	21:10	honey	tbsp	0.75	
54	6:30	H2O	oz	6	23-Feb
54	7:30	banana	cup	0.75	
54	9:45	banana	cup	0.75	
54	9:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
54	9:45	honey	tbsp	1	
54	11:30	H2O	oz	10	
54	12:15	avocado	cup	0.5	
54	12:15	onion (white)	cup	0.25	
54	12:15	pork (Torchy's)	oz	6	
54	12:15	cilantro	tbsp	1	
54	12:15	tomatillo salsa	tbsp	4	
54	12:15	tortilla (corn)	pc	4	
54	12:15	soda (IBC Cream Soda)	oz	12	
54	13:45	banana	cup	0.75	
54	13:45	bread (corn)	oz	3	

APPENDIX C (continued)

54	13:45	butter	tbsp	0.25	
54	13:45	honey	tbsp	0.75	
54	14:45	banana	cup	0.5	
54	14:45	juice (apple)	oz	8	
54	14:45	strawberry	cup	1	
54	15:50	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
54	15:50	honey	tbsp	0.5	
54	19:30	gum (Doublemint)	pc	3	
54	20:20	H2O	oz	16	
54	20:20	bread (corn)	oz	6	
54	20:20	butter	tbsp	0.5	
54	20:20	honey	tbsp	1	
54	21:20	dark chocolate	pc	8	
54	21:20	cranberry	cup	0.125	
54	23:30	banana	cup	0.5	
54	23:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	5	
55	4:00	H2O	oz	10	24-Feb
55	7:30	H2O	oz	4	
55	7:40	banana	cup	0.5	
55	9:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	5	
55	9:45	honey	tbsp	0.5	
55	9:45	avocado	cup	0.25	
55	9:45	onion (white)	cup	0.125	
55	9:45	pork (Torchy's)	oz	3	
55	9:45	cilantro	tbsp	0.5	
55	9:45	tomatillo salsa	tbsp	1	
55	9:45	tortilla (corn)	pc	2	
55	9:45	H2O	oz	16	
55	12:20	soda (Virgil's black cherry cream soda)	oz	12	
55	12:20	pork	oz	5	
55	12:20	capsicum (green)	cup	0.5	
55	12:20	capsicum (red)	cup	0.5	
55	12:20	cumin	tbsp	0.5	
55	12:20	coriander	tbsp	0.25	
55	12:20	tortilla (corn)	pc	3	
55	12:20	avocado	cup	0.25	
55	12:20	salt	tsp	0.125	
55	13:20	dark chocolate	pc	10	

APPENDIX C (continued)

55	13:20	cranberry	cup	0.25	
55	13:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
55	13:20	honey	tbsp	0.5	
55	15:30	dark chocolate	pc	8	
55	15:30	cranberry	cup	0.25	
55	16:40	avocado	cup	0.25	
55	16:40	onion (white)	cup	0.125	
55	16:40	pork (Torchy's)	oz	3	
55	16:40	cilantro	tbsp	0.5	
55	16:40	tomatillo salsa	tbsp	1	
55	16:40	tortilla (corn)	pc	2	
55	16:40	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
55	16:40	honey	tbsp	0.5	
55	16:40	banana	cup	0.5	
55	17:20	dark chocolate	pc	5	
55	17:20	cranberry	cup	0.125	
55	17:20	H2O	oz	16	
55	18:20	H2O	oz	16	
55	18:20	dark chocolate	pc	6	
55	18:20	cranberry	cup	0.125	
55	19:10	banana	cup	0.5	
55	22:30	avocado	cup	0.25	
55	22:30	onion (white)	cup	0.125	
55	22:30	pork (Torchy's)	oz	3	
55	22:30	cilantro	tbsp	0.5	
55	22:30	tomatillo salsa	tbsp	1	
55	22:30	tortilla (corn)	pc	2	
55	22:30	dark chocolate	pc	6	
55	22:30	cranberry	cup	0.125	
55	22:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	20	
55	22:30	honey	tbsp	1.5	
55	22:30	banana	cup	0.5	
56	5:25	banana	cup	0.5	25-Feb
56	8:45	banana	cup	0.5	
56	8:45	soda (La Croix coconut)	oz	12	
56	11:30	dark chocolate	pc	10	
56	11:30	cranberry	cup	0.25	
56	11:50	tilapia	oz	3	

APPENDIX C (continued)

56	11:50	garlic	tbsp	0.375
56	11:50	cumin	tbsp	0.15
56	11:50	cinnamon	tbsp	0.15
56	11:50	olive oil	tbsp	0.375
56	11:50	pepper	tsp	0.15
56	11:50	salt	tsp	0.15
56	11:50	juice (lime)	tbsp	0.15
56	11:50	fennel	cup	0.75
56	11:50	cabbage (green)	cup	0.75
56	11:50	sour cream	cup	0.15
56	11:50	honey chipotle powder	tbsp	0.75
56	11:50	tortilla (corn)	pc	3
56	11:50	avocado	cup	0.5
56	11:50	soda (IBC Cream Soda)	oz	12
56	12:45	gum (strawberry orange)	pc	2
56	13:20	banana	cup	0.5
56	13:20	H2O	oz	16
56	16:00	dark chocolate	pc	8
56	16:00	cranberry	cup	0.25
56	16:00	banana	cup	0.5
56	16:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10
56	16:00	honey	tbsp	0.5
56	17:40	dark chocolate	pc	5
56	17:40	cranberry	cup	0.125
56	17:40	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10
56	17:40	honey	tbsp	1
56	19:00	pork	oz	5
56	19:00	onion (white)	cup	0.25
56	19:00	capsicum (green)	cup	0.25
56	19:00	capsicum (red)	cup	0.25
56	19:00	cumin	tbsp	0.5
56	19:00	coriander	tbsp	0.25
56	19:00	turmeric	tbsp	0.25
56	19:00	salt	tsp	0.125
56	19:00	avocado	cup	0.25
56	19:00	olive oil	tbsp	0.75
56	19:00	tortilla (corn)	pc	2
56	19:00	soda (La Croix coconut)	oz	12

APPENDIX C (continued)

56	21:30	banana	cup	0.75	
56	21:30	H2O	oz	16	
56	23:05	asparagus	cup	0.75	
56	23:05	butter	tbsp	0.5	
56	23:05	garlic pepper	tbsp	0.75	
56	23:05	salt	tsp	0.125	
56	23:05	juice (lemon)	tbsp	0.5	
56	23:05	dark chocolate	pc	8	
56	23:05	cranberry	cup	0.125	
56	23:05	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
56	23:05	honey	tbsp	0.75	
57	6:00	H2O	oz	10	26-Feb
57	7:10	banana	cup	0.75	
57	8:15	pork	oz	6	
57	8:15	rosemary	tbsp	0.5	
57	8:15	jelly (mint)	tbsp	1	
57	8:15	salt	tsp	0.125	
57	11:20	avocado	cup	0.5	
57	11:20	onion (white)	cup	0.25	
57	11:20	pork (Torchy's)	oz	6	
57	11:20	cilantro	tbsp	1	
57	11:20	tomatillo salsa	tbsp	2	
57	11:20	tortilla (corn)	pc	4	
57	11:20	soda (La Croix coconut)	oz	12	
57	12:50	banana	cup	0.5	
57	14:50	avocado	cup	0.25	
57	14:50	onion (white)	cup	0.125	
57	14:50	pork (Torchy's)	oz	3	
57	14:50	cilantro	tbsp	0.5	
57	14:50	tomatillo salsa	tbsp	1	
57	14:50	tortilla (corn)	pc	2	
57	14:50	soda (La Croix coconut)	oz	12	
57	14:50	dark chocolate	pc	12	
57	14:50	cranberry	cup	0.25	
57	15:55	green bean	cup	1.5	
57	15:55	carrot	cup	0.125	
57	15:55	salt	tsp	0.125	
57	15:55	pepper	tsp	0.125	
57	15:55	olive oil	tbsp	1	

APPENDIX C (continued)

57	17:00	soda (IBC Cream Soda)	oz	12	
57	17:45	candy (Jolly Rancher)	pc	2	
57	19:00	banana	cup	1.75	
57	19:00	H2O	oz	16	
57	22:40	banana	cup	0.75	
57	22:40	milk chocolate	pc	5	
57	22:40	raisin	cup	0.125	
57	22:40	H2O	oz	16	
58	6:00	H2O	oz	4	27-Feb
58	7:40	banana	cup	0.75	
58	9:50	avocado	cup	0.25	
58	9:50	onion (white)	cup	0.125	
58	9:50	pork (Torchy's)	oz	3	
58	9:50	cilantro	tbsp	0.5	
58	9:50	tomatillo salsa	tbsp	1	
58	9:50	tortilla (corn)	pc	2	
58	9:50	soda (La Croix coconut)	oz	12	
58	12:45	pork	oz	6	
58	12:45	capsicum (red)	cup	0.4	
58	12:45	capsicum (orange)	cup	0.4	
58	12:45	honey	tbsp	2	
58	12:45	sage	tbsp	0.5	
58	12:45	salt	tsp	0.125	
58	12:45	vinegar (balsamic)	tbsp	2	
58	12:45	avocado	cup	0.75	
58	12:45	tortilla (corn)	pc	3	
58	12:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
58	12:45	milk chocolate	pc	10	
58	12:45	raisin	cup	0.25	
58	14:40	banana	cup	0.75	
58	14:40	soda (La Croix coconut)	oz	12	
58	14:40	milk chocolate	pc	10	
58	14:40	raisin	cup	0.25	
58	16:20	milk chocolate	pc	8	
58	16:20	raisin	cup	0.125	
58	16:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
58	16:20	honey	tbsp	0.75	
58	17:40	H2O	oz	16	

APPENDIX C (continued)

58	17:40	avocado	cup	0.25	
58	17:40	onion (white)	cup	0.125	
58	17:40	pork (Torchy's)	oz	3	
58	17:40	cilantro	tbsp	0.5	
58	17:40	tomatillo salsa	tbsp	1	
58	17:40	tortilla (corn)	pc	2	
58	19:55	milk chocolate	pc	10	
58	19:55	raisin	cup	0.25	
58	19:55	banana	cup	1.75	
58	20:40	lettuce (romain)	cup	2	
58	20:40	chicken	oz	3	
58	20:40	bread (white)	oz	2	
58	20:40	olive oil	tbsp	1	
58	20:40	garlic	tbsp	1	
58	20:40	dressing (Caesar)	tbsp	2	
58	20:40	cheese (parmasean)	oz	1	
58	22:35	gum (strawberry orange)	pc	2	
59	0:10	banana	cup	0.75	28-Feb
59	2:25	H2O	oz	6	
59	7:40	H2O	oz	6	
59	9:40	banana	cup	1.5	
59	9:40	H2O	oz	16	
59	12:00	banana	cup	0.75	
59	12:00	salmon	oz	5	
59	12:00	honey	tbsp	1	
59	12:00	cinnamon	tbsp	0.125	
59	12:00	ground mustard	tbsp	0.125	
59	12:00	salt	tsp	0.125	
59	12:20	milk chocolate	pc	8	
59	12:20	raisin	cup	0.25	
59	13:20	juice (cucumber)	pc	1	
59	13:20	juice (carrot)	pc	3	
59	13:20	juice (strawberry)	pc	15	
59	14:50	H2O	oz	20	
59	14:50	milk chocolate	pc	12	
59	14:50	raisin	cup	0.4	
59	17:20	onion (white)	cup	0.125	
59	17:20	pork (Torchy's)	oz	3	
59	17:20	cilantro	tbsp	0.5	
59	17:20	tomatillo salsa	tbsp	1	

APPENDIX C (continued)

59	17:20	tortilla (corn)	pc	2	
59	17:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
59	17:20	honey	tbsp	0.5	
59	17:20	milk chocolate	pc	5	
59	17:20	raisin	cup	0.125	
59	17:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
59	17:20	honey	tbsp	0.5	
59	20:00	ham	oz	2	
59	20:00	capsicum (red)	cup	0.25	
59	20:00	capsicum (yellow)	cup	0.25	
59	20:00	onion (white)	cup	0.125	
59	20:00	spinach	cup	0.125	
59	20:00	avocado	cup	0.5	
59	20:00	tortilla (corn)	pc	2	
59	20:00	oregano	tbsp	0.25	
59	20:00	salt	tbsp	0.125	
59	20:00	pepper	tsp	0.125	
59	20:00	soda (La Croix coconut)	oz	12	
59	22:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
59	22:00	honey	tbsp	0.5	
59	22:00	milk chocolate	pc	6	
59	22:00	raisin	cup	0.125	
59	22:00	banana	cup	0.75	
59	23:25	banana	cup	0.75	
60	5:00	H2O	oz	8	29-Feb
60	5:20	banana	cup	0.75	
60	5:20	H2O	oz	8	
60	9:00	bread (corn)	oz	3	
60	9:00	butter	tbsp	0.25	
60	9:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
60	9:00	honey	tbsp	1.5	
60	9:00	H2O	oz	16	
60	11:20	bread (corn)	oz	2	
60	11:20	butter	tbsp	0.25	
60	11:20	banana	cup	0.75	
60	12:45	tortilla (corn)	pc	4	
60	12:45	capsicum (green)	cup	0.25	

APPENDIX C (continued)

60	12:45	capsicum (yellow)	cup	0.25	
60	12:45	capsicum (orange)	cup	0.25	
60	12:45	sugar snap pea	cup	0.5	
60	12:45	ham	oz	3	
60	12:45	basil	tbsp	0.25	
60	12:45	ginger	tbsp	0.25	
60	12:45	vinegar (red wine)	tbsp	1	
60	12:45	avocado	cup	1	
60	12:45	soda (La Croix cran-rasp)	oz	12	
60	15:45	banana	cup	0.75	
60	15:45	tea (Lipton Green Tea - decaf)	oz	10	
60	15:45	honey	tbsp	0.5	
60	15:45	milk chocolate	pc	5	
60	15:45	raisin	cup	0.125	
60	16:55	bread (corn)	oz	2	
60	16:55	butter	tbsp	0.4	
60	16:55	honey	tbsp	0.5	
60	16:55	H2O	oz	16	
60	19:30	banana	cup	0.75	
60	21:20	bread (wheat)	oz	5	
60	21:20	corned beef	oz	5	
60	21:20	sour kraut	cup	0.75	
60	21:20	dressing (Thousand Island)	tbsp	3	
60	21:20	tea (Lipton Green Tea - decaf)	oz	10	
60	21:20	honey	tbsp	0.5	
60	23:10	soda (La Croix cran-rasp)	oz	12	
61	2:00	banana	cup	0.75	1-Mar
61	6:15	H2O	oz	2	
61	6:15	banana	cup	0.75	
61	10:00	banana	cup	1	
61	11:10	bread (wheat)	oz	5	
61	11:10	corned beef	oz	5	
61	11:10	sour kraut	cup	0.75	
61	11:10	dressing (Thousand Island)	tbsp	3	
61	13:00	green bean	cup	2	
61	13:00	butter	tbsp	0.25	
61	13:00	salt	tsp	0.125	
61	13:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
61	13:00	honey	tbsp	0.5	

APPENDIX C (continued)

61	14:50	soda (La Croix cran-rasp)	oz	12	
61	16:50	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
61	16:50	honey	tbsp	0.5	
61	18:00	H2O	oz	4	
61	19:00	banana	cup	1.5	
61	21:00	tilapia	oz	2	
61	21:00	salmon	oz	4	
61	21:00	garlic	tbsp	0.375	
61	21:00	cumin	tbsp	0.15	
61	21:00	cinnamon	tbsp	0.25	
61	21:00	olive oil	tbsp	0.375	
61	21:00	pepper	tsp	0.15	
61	21:00	salt	tsp	0.15	
61	21:00	juice (lime)	tbsp	0.15	
61	21:00	fennel	cup	0.75	
61	21:00	cabbage (green)	cup	0.75	
61	21:00	sour cream	cup	0.15	
61	21:00	honey chipotle powder	tbsp	0.75	
61	21:00	tortilla (corn)	pc	3	
61	23:00	sherbet (mango)	cup	1.5	
62	5:50	H2O	oz	2	2-Mar
62	8:25	H2O	oz	10	
62	9:15	banana	cup	1	
62	10:50	banana	cup	1	
62	10:50	milk chocolate	pc	5	
62	10:50	raisin	cup	0.125	
62	11:10	H2O	oz	16	
62	12:45	onion (white)	cup	0.25	
62	12:45	capsicum (green)	cup	0.5	
62	12:45	capsicum (yellow)	cup	0.25	
62	12:45	pork	oz	5	
62	12:45	olive oil	tbsp	0.5	
62	12:45	cumin	tbsp	1	
62	12:45	coriander	tbsp	0.75	
62	12:45	salt	tsp	0.125	
62	12:45	tortilla (corn)	pc	3	
62	12:45	soda (La Croix cran-rasp)	oz	12	
62	14:10	milk chocolate	pc	6	
62	14:10	raisin	cup	0.125	

APPENDIX C (continued)

62	16:40	banana	cup	1.75	
62	19:00	H2O	oz	12	
62	19:50	chickpea	oz	9	
62	19:50	olive oil	oz	1	
62	19:50	coriander	tbsp	0.25	
62	19:50	parsley	tbsp	0.25	
62	19:50	garlic	tbsp	0.25	
62	19:50	hummus	cup	0.5	
62	19:50	beef	oz	5	
62	19:50	mint	tbsp	0.25	
62	19:50	garlic	tbsp	2	
62	19:50	soda (Dr. Pepper)	oz	20	
62	21:00	H2O	oz	12	
62	22:45	banana	cup	0.75	
62	22:45	H2O	oz	16	
63	5:25	H2O	oz	4	3-Mar
63	5:30	banana	cup	0.75	
63	5:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
63	5:30	honey	tbsp	1	
63	7:20	banana	cup	0.5	
63	9:45	H2O	oz	14	
63	9:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
63	9:45	honey	tbsp	0.5	
63	11:35	banana	cup	0.75	
63	11:35	soda (La Croix cran-rasp)	oz	12	
63	14:40	pork	oz	8	
63	14:40	capsicum (green)	cup	0.25	
63	14:40	capsicum (yellow)	cup	0.25	
63	14:40	capsicum (red)	cup	0.25	
63	14:40	onion (white)	cup	0.25	
63	14:40	tortilla (corn)	pc	3	
63	14:40	cumin	tbsp	0.25	
63	14:40	coriander	tbsp	0.25	
63	14:40	salt	tsp	0.125	
63	14:40	olive oil	tbsp	0.5	
63	14:40	H2O	oz	8	
63	17:00	candy (Jolly Rancher)	pc	3	
63	21:15	king snapper	oz	5	

APPENDIX C (continued)

63	21:15	garlic	tbsp	0.375	
63	21:15	cumin	tbsp	0.15	
63	21:15	cinnamon	tbsp	0.15	
63	21:15	olive oil	tbsp	0.375	
63	21:15	pepper	tsp	0.15	
63	21:15	salt	tsp	0.15	
63	21:15	juice (lime)	tbsp	0.15	
63	21:15	fennel	cup	0.75	
63	21:15	cabbage (green)	cup	0.75	
63	21:15	sour cream	cup	0.15	
63	21:15	honey chipotle powder	tbsp	0.75	
63	21:15	tortilla (corn)	pc	3	
63	21:15	soda (La Croix cran-rasp)	oz	12	
63	22:10	candy (Andes Mint)	pc	4	
63	23:15	H2O	oz	6	
64	0:15	candy (Andes Mint)	pc	2	4-Mar
64	0:15	banana	cup	0.75	
64	6:25	H2O	oz	2	
64	8:45	banana	cup	1.5	
64	10:05	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
64	10:05	honey	tbsp	0.5	
64	12:30	bread (white)	oz	3	
64	12:30	cinnamon	tbsp	0.125	
64	12:30	butter	tbsp	0.5	
64	12:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
64	13:30	pork	oz	8	
64	13:30	capsicum (green)	cup	0.25	
64	13:30	capsicum (yellow)	cup	0.25	
64	13:30	capsicum (red)	cup	0.25	
64	13:30	onion (white)	cup	0.25	
64	13:30	tortilla (corn)	pc	3	
64	13:30	cumin	tbsp	0.25	
64	13:30	coriander	tbsp	0.25	
64	13:30	salt	tsp	0.125	
64	13:30	olive oil	tbsp	0.5	
64	16:00	banana	cup	0.5	
64	16:00	bread (white)	oz	3	
64	16:00	butter	tbsp	0.75	

APPENDIX C (continued)

64	20:30	couscous	cup	1	
64	20:30	garlic pepper	tbsp	1	
64	20:30	green bean	cup	1.5	
64	20:30	butter	tbsp	1	
64	20:30	juice (lemon)	tbsp	0.5	
64	20:30	olive oil	tbsp	1	
64	20:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
64	20:30	honey	tbsp	0.5	
64	22:40	couscous	cup	0.5	
64	22:40	garlic	tbsp	0.5	
64	22:40	olive oil	tbsp	0.5	
65	6:25	banana	cup	0.75	5-Mar
65	6:25	H2O	oz	8	
65	9:15	banana	cup	0.75	
65	11:30	pork	oz	5	
65	11:30	capsicum (green)	cup	0.2	
65	11:30	capsicum (yellow)	cup	0.2	
65	11:30	capsicum (red)	cup	0.2	
65	11:30	onion (white)	cup	0.2	
65	11:30	tortilla (corn)	pc	3	
65	11:30	cumin	tbsp	0.2	
65	11:30	coriander	tbsp	0.2	
65	11:30	salt	tsp	0.1	
65	11:30	olive oil	tbsp	0.4	
65	11:30	soda (La Croix cran-rasp)	oz	12	
65	15:00	banana	cup	1.5	
65	16:10	king snapper	oz	5	
65	16:10	garlic	tbsp	0.375	
65	16:10	cumin	tbsp	0.15	
65	16:10	cinnamon	tbsp	0.15	
65	16:10	olive oil	tbsp	0.375	
65	16:10	pepper	tsp	0.15	
65	16:10	salt	tsp	0.15	
65	16:10	juice (lime)	tbsp	0.15	
65	16:10	fennel	cup	0.75	
65	16:10	cabbage (green)	cup	0.75	
65	16:10	sour cream	cup	0.15	
65	16:10	honey chipotle powder	tbsp	0.75	
65	16:10	tortilla (corn)	pc	2	

APPENDIX C (continued)

65	16:10	soda (La Croix cran-rasp)	oz	12	
65	18:15	juice (carrot)	pc	3	
65	18:15	juice (apple)	pc	1	
65	18:15	juice (spinach)	cup	2	
65	20:45	asparagus	cup	2	
65	20:45	butter	tbsp	0.5	
65	20:45	coriander	tbsp	0.125	
65	20:45	garlic pepper	tbsp	0.25	
65	20:45	salt	tsp	0.125	
65	20:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
65	20:45	honey	tbsp	0.25	
65	21:25	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
65	21:25	honey	tbsp	0.25	
66	6:30	H2O	oz	8	6-Mar
66	9:00	banana	cup	0.75	
66	9:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
66	9:00	honey	tbsp	0.25	
66	11:05	banana	cup	0.75	
66	12:35	pork	oz	6	
66	12:35	milk chocolate	pc	10	
66	12:35	caramel	pc	6	
66	12:35	cookie (white)	oz	4	
66	12:35	soda (La Croix cran-rasp)	oz	12	
66	12:35	salt	tbsp	0.125	
66	12:35	jelly (mint)	tbsp	0.5	
66	14:00	milk chocolate	pc	4	
66	16:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
66	16:00	honey	tbsp	1.25	
66	17:00	banana	cup	0.75	
66	18:00	pork	oz	8	
66	18:00	capsicum (green)	cup	0.4	
66	18:00	capsicum (red)	cup	0.4	
66	18:00	wine (red)	cup	0.25	
66	18:00	honey	tbsp	2	
66	18:00	thyme	tbsp	0.5	
66	18:00	sage	tbsp	0.5	
66	18:00	salt	tsp	0.125	

APPENDIX C (continued)

66	18:00	soda (La Croix cran-rasp)	oz	12	
66	18:45	milk chocolate	pc	3	
66	20:10	milk chocolate	pc	3	
66	21:00	steak	oz	5	
66	21:00	capsicum (red)	cup	1	
66	21:00	onion (red)	cup	0.25	
66	21:00	lettuce (romain)	cup	0.5	
66	21:00	olive oil	tbsp	2	
66	21:00	garlic	tbsp	1	
66	21:00	parsley	tbsp	1	
66	21:00	vinegar (white wine)	tbsp	2	
66	21:00	oregano	tbsp	0.5	
66	21:00	salt	tsp	0.125	
66	21:00	broccoli	cup	0.25	
66	21:00	carrot	cup	0.25	
66	21:00	tortilla (flour) (spinach - large)	pc	1	
66	21:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
66	21:00	honey	tbsp	1	
66	22:30	milk chocolate	pc	2	
66	22:30	H2O	oz	16	
67	6:45	H2O	oz	2	7-Mar
67	6:45	banana	cup	0.5	
67	7:30	milk chocolate	pc	4	
67	7:30	H2O	oz	14	
67	8:55	banana	cup	0.75	
67	8:55	milk chocolate	pc	2	
67	11:15	banana	cup	0.75	
67	11:15	milk chocolate	pc	2	
67	12:00	H2O	oz	16	
67	14:50	hamburger	oz	4	
67	14:50	lettuce (iceberg)	cup	0.5	
67	14:50	onion (red)	cup	0.25	
67	14:50	mayonaise	tbsp	1	
67	14:50	pickle (dill)	cup	0.125	
67	14:50	bread (white)	oz	4	
67	14:50	pepper	tbsp	0.5	
67	14:50	salt	tsp	0.25	
67	14:50	potato	cup	1	
67	14:50	canola oil	tbsp	4	

APPENDIX C (continued)

67	14:50	soda (Sprite)	oz	24	
67	17:30	dark chocolate	pc	6	
67	17:55	soda (La Croix cran-rasp)	oz	12	
67	17:55	H2O	oz	10	
67	21:00	pasta (rice)	cup	3	
67	21:00	olive oil	tbsp	2	
67	21:00	parsley	tbsp	2	
67	21:00	spinach	cup	0.5	
67	21:00	garlic	tbsp	1	
67	21:00	pepper	tsp	0.25	
67	21:00	juice (lemon)	tbsp	1	
67	21:00	salt	tsp	0.125	
67	21:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
67	21:00	honey	tbsp	0.75	
67	21:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
67	21:45	honey	tbsp	0.75	
67	21:45	dark chocolate	pc	4	
67	22:10	banana	cup	1	
67	23:15	dark chocolate	pc	2	
67	23:15	H2O	oz	8	
68	5:45	banana	cup	1	8-Mar
68	5:45	H2O	oz	8	
68	9:20	banana	cup	1	
68	9:20	dark chocolate	pc	2	
68	9:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
68	9:20	honey	tbsp	1	
68	10:50	onion (white)	cup	0.125	
68	10:50	pork (Torchy's)	oz	3	
68	10:50	cilantro	tbsp	0.5	
68	10:50	tomatillo salsa	tbsp	2	
68	10:50	tortilla (corn)	pc	2	
68	10:50	H2O	oz	16	
68	12:10	pork	oz	6	
68	12:10	fennel	cup	0.75	
68	12:10	capsicum (green)	cup	0.5	
68	12:10	capsicum (red)	cup	0.5	
68	12:10	sugar snap pea	cup	0.5	
68	12:10	basil	tbsp	1	

APPENDIX C (continued)

68	12:10	salt	tsp	0.125	
68	12:10	olive oil	tbsp	1	
68	12:10	dark chocolate	pc	2	
68	13:10	H2O	oz	16	
68	13:10	dark chocolate	pc	2	
68	15:30	onion (white)	cup	0.125	
68	15:30	pork (Torchy's)	oz	3	
68	15:30	cilantro	tbsp	0.5	
68	15:30	tomatillo salsa	tbsp	1	
68	15:30	tortilla (corn)	pc	2	
68	17:00	banana	cup	1	
68	17:00	juice (apple)	oz	4	
68	17:00	strawberry	cup	0.75	
68	18:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
68	18:00	honey	tbsp	0.5	
68	21:10	banana	cup	1	
68	21:20	dark chocolate	pc	2	
69	5:35	H2O	oz	2	9-Mar
69	6:00	banana	cup	1	
69	9:30	banana	cup	1	
69	9:30	onion (white)	cup	0.125	
69	9:30	pork (Torchy's)	oz	3	
69	9:30	cilantro	tbsp	0.5	
69	9:30	tomatillo salsa	tbsp	1	
69	9:30	tortilla (corn)	pc	2	
69	9:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
69	9:30	honey	tbsp	0.5	
69	10:40	banana	cup	1	
69	10:40	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
69	10:40	honey	tbsp	0.5	
69	12:00	pork	oz	6	
69	12:00	capsicum (red)	cup	0.25	
69	12:00	capsicum (green)	cup	0.25	
69	12:00	onion (white)	cup	0.25	
69	12:00	avocado	cup	0.75	
69	12:00	garlic pepper	tbsp	0.5	
69	12:00	cumin	tbsp	1	
69	12:00	tomatillo	cup	0.25	

APPENDIX C (continued)

69	12:00	coriander	tbsp	0.25	
69	12:00	tortilla (corn)	pc	3	
69	12:00	juice (cucumber)	pc	0.5	
69	12:00	juice (pear)	pc	1	
69	12:00	juice (kiwi)	pc	1	
69	12:00	juice (carrot)	pc	3	
69	12:00	juice (spinach)	cup	2	
69	14:00	banana	cup	1	
69	14:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
69	14:00	honey	tbsp	0.5	
69	15:00	candy (Jolly Rancher)	pc	4	
69	16:50	banana	cup	1	
69	17:20	banana	cup	1	
69	17:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
69	17:20	honey	tbsp	0.5	
69	19:00	avocado	cup	0.25	
69	19:00	onion (white)	cup	0.125	
69	19:00	pork (Torchy's)	oz	3	
69	19:00	cilantro	tbsp	0.5	
69	19:00	tomatillo salsa	tbsp	1	
69	19:00	tortilla (corn)	pc	2	
69	22:00	banana	cup	1	
70	10:00	banana	cup	1	10-Mar
70	10:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
70	10:00	honey	tbsp	0.5	
70	12:40	candy (Jolly Rancher)	pc	4	
70	14:00	banana	cup	1	
70	14:00	H2O	oz	16	
70	15:20	pork	oz	6	
70	15:20	capsicum (green)	cup	0.5	
70	15:20	capsicum (red)	cup	0.5	
70	15:20	fennel	cup	0.5	
70	15:20	mung bean sprout	cup	1	
70	15:20	mushroom (shitake)	cup	0.5	
70	15:20	ginger	tbsp	0.5	
70	15:20	salt	tsp	0.125	
70	15:20	vinegar (red wine)	tbsp	2	

APPENDIX C (continued)

70	15:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
70	15:20	honey	tbsp	0.75	
70	17:50	banana	cup	1	
70	17:50	H2O	oz	16	
70	21:30	banana	cup	1	
70	21:30	king snapper	oz	3	
70	21:30	tortilla (corn)	pc	2	
70	21:30	fennel	cup	0.5	
70	21:30	cabbage (green)	cup	0.5	
70	21:30	honey chipotle powder	tbsp	0.5	
70	21:30	olive oil	tbsp	0.25	
70	21:30	garlic	tbsp	0.25	
70	21:30	cumin	tbsp	0.1	
70	21:30	cinnamon	tbsp	0.1	
70	21:30	sour cream	cup	0.1	
70	21:30	pepper	tsp	0.1	
70	21:30	salt	tsp	0.1	
70	21:30	avocado	cup	0.5	
70	22:50	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
70	22:50	honey	tbsp	0.75	
71	7:10	H2O	oz	8	11-Mar
71	9:25	H2O	oz	8	
71	9:45	banana	cup	1	
71	10:30	juice (kiwi)	pc	1	
71	10:30	juice (pear)	pc	1	
71	10:30	juice (fennel stalk)	pc	2	
71	10:30	juice (cucumber)	pc	0.5	
71	10:30	juice (strawberry)	pc	10	
71	12:00	H2O	oz	16	
71	12:55	milk chocolate	pc	4	
71	12:55	caramel	pc	6	
71	12:55	cookie (white)	oz	4	
71	12:55	dark chocolate	pc	2	
71	12:55	coconut	oz	2	
71	12:55	sugar	tbsp	3	
71	14:10	milk chocolate	pc	2	
71	14:10	wafer (vanilla)	oz	4	
71	14:10	H2O	oz	16	

APPENDIX C (continued)

71	15:25	milk chocolate	pc	2	
71	15:25	wafer (vanilla)	oz	4	
71	15:25	banana	cup	1	
71	17:20	milk chocolate	pc	1	
71	17:20	wafer (vanilla)	oz	2	
71	17:20	onion (white)	cup	0.125	
71	17:20	pork (Torchy's)	oz	3	
71	17:20	cilantro	tbsp	0.5	
71	17:20	tomatillo salsa	tbsp	1	
71	17:20	tortilla (corn)	pc	2	
71	18:25	soda (La Croix berry)	oz	12	
71	19:30	banana	cup	1	
71	19:30	milk chocolate	pc	2	
71	19:30	wafer (vanilla)	oz	4	
71	19:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
71	19:30	honey	tbsp	0.75	
71	21:30	green bean	cup	1.5	
71	21:30	garlic pepper	tbsp	0.5	
71	21:30	salt	tsp	0.125	
71	21:30	butter	tbsp	0.5	
71	23:30	banana	cup	1	
71	23:30	milk chocolate	pc	1	
71	23:30	wafer (vanilla)	oz	2	
72	5:40	banana	cup	1	12-Mar
72	10:20	banana	cup	1	
72	10:20	juice (pomegranate blackberry)	oz	4	
72	10:20	peach	cup	0.25	
72	10:20	grape	cup	0.25	
72	10:20	strawberry	cup	0.25	
72	10:20	milk chocolate	pc	1	
72	10:20	wafer (vanilla)	oz	2	
72	11:50	pork	oz	8	
72	11:50	tomatillo	cup	0.25	
72	11:50	mushroom (shitake)	cup	0.25	
72	11:50	mung bean sprout	cup	0.75	
72	11:50	turmeric	tbsp	0.75	
72	11:50	garlic pepper	tbsp	0.75	
72	11:50	salt	tsp	0.125	
72	11:50	soda (La Croix berry)	oz	12	

APPENDIX C (continued)

72	12:45	milk chocolate	pc	1	
72	12:45	wafer (vanilla)	oz	2	
72	12:45	H2O	oz	16	
72	14:20	milk chocolate	pc	1	
72	14:20	wafer (vanilla)	oz	2	
72	14:20	banana	cup	1	
72	16:00	wafer (vanilla)	oz	4	
72	16:00	banana	cup	2	
72	16:00	juice (carrot)	pc	2	
72	16:00	juice (cucumber)	pc	0.5	
72	16:00	juice (kale)	pc	1.5	
72	16:00	juice (apple)	pc	1	
72	16:00	juice (pear)	pc	1	
72	16:00	juice (kiwi)	pc	1	
72	17:15	banana	cup	1	
72	17:55	H2O	oz	16	
72	18:50	banana	cup	1	
72	18:50	strawberry	cup	1.25	
72	18:50	honey	tbsp	1	
72	18:50	soda (La Croix berry)	oz	12	
72	21:10	egg	oz	4	
72	21:10	tomatillo	cup	0.25	
72	21:10	onion (white)	cup	0.125	
72	21:10	capsicum (green)	cup	0.25	
72	21:10	basil	tbsp	0.5	
72	21:10	thyme	tbsp	0.25	
72	21:10	salt	tsp	0.125	
72	21:10	pepper	tsp	0.125	
72	21:10	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
72	21:10	honey	tbsp	0.25	
72	23:30	potato	cup	1	
72	23:30	vinegar (white)	tbsp	2	
72	23:30	sunflower oil	tbsp	2	
72	23:30	salt	tsp	0.25	
72	23:30	candy (Jolly Rancher)	pc	2	
73	9:20	banana	cup	1	13-Mar
73	10:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
73	10:30	honey	tbsp	0.5	

APPENDIX C (continued)

73	11:50	pork	oz	8	
73	11:50	capsicum (red)	cup	0.25	
73	11:50	capsicum (green)	cup	0.25	
73	11:50	fennel	cup	0.5	
73	11:50	avocado	cup	0.75	
73	11:50	coriander	tbsp	0.25	
73	11:50	cumin	tbsp	0.5	
73	11:50	salt	tsp	0.125	
73	11:50	garlic pepper	tbsp	0.5	
73	11:50	tortilla (corn)	pc	3	
73	11:50	soda (La Croix berry)	oz	12	
73	14:20	banana	cup	1	
73	14:20	juice (pomegranate blackberry)	oz	4	
73	14:20	strawberry	oz	6	
73	16:20	milk chocolate	pc	4	
73	16:20	caramel	pc	6	
73	16:20	cookie (white)	oz	4	
73	14:55	soda (La Croix berry)	oz	12	
73	18:05	milk chocolate	pc	2	
73	18:05	sugar	tbsp	4	
73	18:05	butterscotch	tbsp	0.5	
73	18:05	H2O	oz	16	
73	19:20	banana	cup	1	
73	20:00	dark chocolate	pc	1	
73	20:00	mint	tbsp	0.125	
73	20:00	sugar	tbsp	4	
73	20:10	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
73	20:10	honey	tbsp	0.5	
73	20:25	bread (white)	oz	4	
73	20:25	salami	oz	2	
73	20:25	pepperoni	oz	2	
73	20:25	spinach	cup	0.5	
73	20:25	black olive	cup	0.125	
73	20:25	capsicum (green)	cup	0.125	
73	23:30	banana	cup	1	
74	7:00	banana	cup	1	14-Mar
74	11:10	banana	cup	1	
74	11:50	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	

APPENDIX C (continued)

74	11:50	honey	tbsp	0.5	
74	11:50	H2O	oz	16	
74	13:20	pork	oz	8	
74	13:20	capsicum (green)	cup	0.25	
74	13:20	capsicum (red)	cup	0.5	
74	13:20	fennel	cup	0.5	
74	13:20	mushroom (shitake)	cup	0.25	
74	13:20	turmeric	tbsp	0.5	
74	13:20	garlic pepper	tbsp	0.5	
74	13:20	salt	tsp	0.125	
74	13:20	soda (La Croix berry)	oz	12	
74	15:15	banana	cup	1	
74	15:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
74	15:15	honey	tbsp	0.5	
74	16:25	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
74	16:25	honey	tbsp	0.5	
74	18:20	juice (kale)	pc	2	
74	18:20	juice (cucumber)	pc	1	
74	18:20	juice (apple)	pc	1	
74	18:20	juice (kiwi)	pc	1	
74	18:20	juice (pear)	pc	1	
74	18:20	juice (carrot)	pc	3	
74	21:00	salmon	oz	5	
74	21:00	ground mustard	tbsp	0.05	
74	21:00	cinnamon	tbsp	0.125	
74	21:00	brown sugar	tbsp	1	
74	21:00	asparagus	cup	1.5	
74	21:00	butter	tbsp	0.125	
74	21:00	lemon pepper	tbsp	0.125	
74	21:00	salt	tsp	0.125	
75	7:10	banana	cup	1	15-Mar
75	9:45	banana	cup	1	
75	9:45	juice (pomegranate blackberry)	oz	4	
75	9:45	strawberry	cup	0.5	
75	9:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
75	9:45	honey	tbsp	1	
75	11:00	salmon	oz	4	
75	11:00	ground mustard	tbsp	0.05	

APPENDIX C (continued)

75	11:00	cinnamon	tbsp	0.125	
75	11:00	brown sugar	tbsp	0.25	
75	11:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
75	11:00	honey	tbsp	0.5	
75	13:00	salmon	oz	4	
75	13:00	ground mustard	tbsp	0.05	
75	13:00	cinnamon	tbsp	0.125	
75	13:00	brown sugar	tbsp	0.25	
75	13:25	H2O	oz	16	
75	13:25	pork	oz	4	
75	13:25	corn meal	oz	8	
75	13:25	coriander	tbsp	0.5	
75	13:25	cumin	tbsp	0.5	
75	13:25	salt	tbsp	0.125	
75	13:25	apple	cup	0.125	
75	13:25	raisin	cup	0.125	
75	13:25	sugar	tbsp	2	
75	15:25	banana	cup	1	
75	15:25	milk chocolate	pc	4	
75	15:25	caramel	pc	6	
75	15:25	cookie (white)	oz	4	
75	16:35	soda (La Croix berry)	oz	12	
75	18:00	milk chocolate	pc	1	
75	18:00	wafer (vanilla)	oz	2	
75	18:00	banana	cup	1	
75	20:50	juice (lime)	tbsp	0.5	
75	20:50	onion (white)	cup	0.25	
75	20:50	pork (Torchy's)	oz	6	
75	20:50	cilantro	tbsp	1	
75	20:50	tomatillo salsa	tbsp	2	
75	20:50	tortilla (corn)	pc	4	
75	20:50	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
75	20:50	honey	tbsp	1	
75	22:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
75	22:00	honey	tbsp	1	
76	5:30	banana	cup	0.75	16-Mar
76	9:10	banana	cup	0.75	
76	10:10	milk chocolate	pc	2	

APPENDIX C (continued)

76	10:10	wafer (vanilla)	oz	4	
76	10:10	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
76	10:10	honey	tbsp	0.5	
76	11:45	pork	oz	8	
76	11:45	capsicum (red)	cup	0.25	
76	11:45	capsicum (green)	cup	0.25	
76	11:45	onion (white)	cup	0.125	
76	11:45	tomatillo	cup	0.125	
76	11:45	cumin	tbsp	0.25	
76	11:45	coriander	tbsp	0.25	
76	11:45	garlic pepper	tbsp	0.5	
76	11:45	salt	tsp	0.125	
76	11:45	soda (La Croix berry)	oz	12	
76	14:20	banana	cup	0.75	
76	14:30	H2O	oz	16	
76	16:20	banana	cup	0.75	
76	16:20	juice (pomegranate blackberry)	oz	4	
76	16:20	strawberry	cup	0.5	
76	17:45	salmon	oz	3	
76	17:45	ground mustard	tbsp	0.125	
76	17:45	cinnamon	tbsp	0.125	
76	17:45	brown sugar	tbsp	0.25	
76	17:45	H2O	oz	16	
76	20:00	bread (wheat)	oz	6	
76	20:00	ham	oz	4	
76	20:00	turkey	oz	4	
76	20:00	bacon	oz	2	
76	20:00	honey mustard	tbsp	2	
76	20:00	capsicum (green)	cup	0.25	
76	20:00	black olive	cup	0.125	
76	20:00	soda (La Croix berry)	oz	12	
76	23:10	rum	oz	3	
76	23:10	soda (Coca Cola)	oz	12	
77	7:00	banana	cup	0.75	17-Mar
77	7:00	H2O	oz	16	
77	9:00	banana	cup	0.75	
77	9:00	H2O	oz	16	
77	9:00	gum (Doublemint)	pc	2	
77	11:30	banana	cup	1	

APPENDIX C (continued)

77	11:30	strawberry	cup	0.5	
77	11:30	juice (pomegranate blackberry)	oz	4	
77	15:00	pork	oz	6	
77	15:00	capsicum (red)	cup	0.25	
77	15:00	capsicum (green)	cup	0.25	
77	15:00	tomatillo	cup	0.25	
77	15:00	sugar snap pea	cup	0.5	
77	15:00	ginger	tbsp	0.5	
77	15:00	parsley	tbsp	1	
77	15:00	salt	tsp	0.125	
77	15:00	vinegar (rice wine)	tbsp	1	
77	15:00	soda (La Croix berry)	oz	12	
77	18:10	H2O	oz	10	
77	21:45	banana	cup	1.25	
77	22:00	avocado	cup	0.5	
77	22:00	tomato	cup	0.125	
77	22:00	onion (white)	cup	0.125	
77	22:00	juice (lime)	tbsp	0.5	
77	22:00	garlic pepper	tbsp	0.25	
77	22:00	salt	tsp	0.125	
77	22:00	chip (corn tortilla)	oz	3	
77	22:15	gum (Doublemint)	pc	2	
77	22:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
77	22:15	honey	tbsp	0.75	
77	22:15	candy (Sour Patch Kids)	pc	15	
77	23:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
77	23:20	honey	tbsp	0.75	
78	7:00	H2O	oz	6	18-Mar
78	8:15	banana	cup	0.5	
78	9:00	banana	cup	0.5	
78	9:00	H2O	oz	16	
78	11:00	candy (Sour Patch Kids)	pc	10	
78	11:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
78	11:00	honey	tbsp	0.5	
78	12:30	pork	oz	8	
78	12:30	capsicum (red)	cup	0.25	
78	12:30	capsicum (green)	cup	0.25	
78	12:30	tomatillo	cup	0.25	

APPENDIX C (continued)

78	12:30	cumin	tbsp	0.5	
78	12:30	coriander	tbsp	0.25	
78	12:30	turmeric	tbsp	0.5	
78	12:30	soda (La Croix berry)	oz	12	
78	12:30	salt	tsp	0.125	
78	12:30	tortilla (corn)	pc	3	
78	15:20	mint chocolate	pc	3	
78	15:20	coconut ice cream	oz	6	
78	17:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
78	17:00	honey	tbsp	0.5	
78	19:00	juice (carrot)	pc	2	
78	19:00	juice (apple)	pc	1	
78	19:00	juice (kiwi)	pc	1	
78	19:00	juice (kale)	pc	1	
78	19:00	juice (peach)	pc	1	
78	19:00	banana	cup	1	
78	22:00	brown sugar	tbsp	2	
78	22:00	sugar	tbsp	1	
78	22:00	bread (white)	oz	1	
78	22:00	butter	tbsp	1	
78	22:00	milk chocolate	pc	4	
79	3:50	banana	cup	0.5	19-Mar
79	6:30	H2O	oz	6	
79	6:30	banana	cup	0.5	
79	11:25	pork	oz	8	
79	11:25	fennel	cup	0.5	
79	11:25	capsicum (red)	cup	0.5	
79	11:25	capsicum (green)	cup	0.5	
79	11:25	basil	tbsp	0.5	
79	11:25	oregano	tbsp	0.25	
79	11:25	salt	tsp	0.125	
79	15:20	banana	cup	1.25	
79	15:20	juice (pomegranate blackberry)	oz	6	
79	15:20	raspberry	cup	0.25	
79	15:20	blackberry	cup	0.25	
79	15:20	salmon	oz	3	
79	15:20	ground mustard	tbsp	0.125	
79	15:20	cinnamon	tbsp	0.125	
79	15:20	brown sugar	tbsp	0.5	

APPENDIX C (continued)

79	15:50	H2O	oz	3	
79	18:00	bagel (white)	oz	6	
79	18:00	blueberry	cup	0.125	
79	18:00	raspberry	cup	0.125	
79	18:00	banana	cup	1	
79	18:00	peanut butter	tbsp	3	
79	18:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
79	18:00	honey	tbsp	1.5	
79	18:35	H2O	oz	16	
79	22:15	gum (Doublemint)	pc	2	
79	22:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
79	22:15	honey	tbsp	0.75	
79	22:15	asparagus	cup	2	
79	22:15	butter	tbsp	0.75	
79	22:15	garlic pepper	tbsp	0.75	
79	22:15	salt	tsp	0.125	
79	22:15	banana	cup	1	
79	22:15	H2O	oz	16	
79	23:45	H2O	oz	6	
80	6:45	banana	cup	1	20-Mar
80	9:45	banana	cup	1	
80	10:45	H2O	oz	8	
80	11:50	banana	cup	1	
80	11:50	milk chocolate	pc	4	
80	11:50	nougat	oz	3	
80	12:10	pork	oz	6	
80	12:10	capsicum (red)	cup	0.5	
80	12:10	capsicum (green)	cup	0.5	
80	12:10	fennel	cup	0.5	
80	12:10	cumin	tbsp	0.75	
80	12:10	coriander	tbsp	0.25	
80	12:10	garlic pepper	tbsp	0.75	
80	12:10	salt	tsp	0.125	
80	12:10	soda (La Croix coconut)	oz	12	
80	13:25	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
80	13:25	honey	tbsp	1	
80	14:35	soda (La Croix coconut)	oz	12	
80	17:00	banana	cup	1	

APPENDIX C (continued)

80	17:00	juice (pomegranate blackberry)	oz	4	
80	17:00	peach	cup	1	
80	18:00	H2O	oz	4	
80	20:00	bagel (white)	oz	6	
80	20:00	blueberry	cup	0.125	
80	20:00	butter	tbsp	0.75	
80	20:00	H2O	oz	16	
80	20:10	candy (Jolly Rancher)	pc	2	
81	6:30	H2O	oz	10	21-Mar
81	7:10	banana	cup	1	
81	9:55	banana	cup	1	
81	9:55	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
81	9:55	honey	tbsp	0.75	
81	12:20	pork	oz	6	
81	12:20	capsicum (red)	cup	0.5	
81	12:20	capsicum (green)	cup	0.5	
81	12:20	onion (white)	cup	0.25	
81	12:20	mushroom (portabella)	cup	0.25	
81	12:20	wine (red)	tbsp	4	
81	12:20	sage	tbsp	0.5	
81	12:20	thyme	tbsp	0.5	
81	12:20	salt	tsp	0.125	
81	12:20	soda (La Croix coconut)	oz	12	
81	13:25	juice (peach)	pc	2	
81	13:25	juice (fennel stalk)	pc	1	
81	13:25	juice (carrot)	pc	2	
81	13:25	juice (kale)	pc	1.5	
81	15:45	grape juice	oz	8	
81	15:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
81	15:45	honey	tbsp	0.75	
81	16:35	banana	cup	1	
81	16:35	H2O	oz	16	
81	18:10	bagel (white)	oz	6	
81	18:10	blueberry	cup	0.125	
81	18:10	raspberry	cup	0.125	
81	18:10	peanut butter	tbsp	2	
81	18:10	jelly (triple berry)	tbsp	3	
81	18:20	soda (La Croix coconut)	oz	12	

APPENDIX C (continued)

81	19:45	banana	cup	1	
81	21:45	pepperoni	oz	1	
81	21:45	asparagus	cup	1.5	
81	21:45	olive oil	tbsp	1	
81	21:45	salt	tsp	0.125	
81	21:45	banana	cup	0.5	
81	22:15	grape juice	oz	6	
82	6:15	banana	cup	1	22-Mar
82	9:25	banana	cup	1	
82	10:45	pork	oz	6	
82	10:45	garlic	tbsp	0.5	
82	10:45	salt	tsp	0.125	
82	10:45	pepper	tbsp	0.125	
82	10:45	honey	tbsp	1	
82	11:00	H2O	oz	8	
82	11:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	10	
82	11:00	honey	tbsp	0.5	
82	13:00	pork	oz	6	
82	13:00	capsicum (green)	cup	0.75	
82	13:00	capsicum (red)	cup	0.75	
82	13:00	fennel	cup	0.5	
82	13:00	salt	tsp	0.125	
82	13:00	cumin	tbsp	0.25	
82	13:00	coriander	tbsp	0.25	
82	13:00	garlic pepper	tbsp	0.5	
82	13:00	grape juice	oz	4	
82	16:10	H2O	oz	8	
82	16:10	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
82	16:10	honey	tbsp	0.5	
82	16:50	banana	cup	1	
82	16:50	juice (pomegranate blackberry)	oz	4	
82	16:50	cherry	cup	0.5	
82	19:45	H2O	oz	16	
82	20:30	banana	cup	0.5	
82	20:30	candy (Skittles sour)	pc	4	
82	20:30	soda (La Croix coconut)	oz	12	
82	22:50	banana	cup	0.5	
83	7:40	banana	cup	0.5	23-Mar

APPENDIX C (continued)

83	7:40	H2O	oz	6	
83	11:15	banana	cup	0.5	
83	11:15	H2O	oz	10	
83	14:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	9	
83	14:30	honey	tbsp	0.5	
83	15:20	banana	cup	0.5	
83	15:20	juice (pomegranate blackberry)	oz	4	
83	15:20	raspberry	cup	0.25	
83	15:20	blackberry	cup	0.25	
83	15:20	blueberry	cup	0.125	
83	21:40	H2O	oz	10	
83	23:30	banana	cup	0.5	
84	2:45	banana	cup	0.5	24-Mar
84	2:45	H2O	oz	8	
84	2:45	bagel (white)	oz	3	
84	4:00	H2O	oz	16	
84	9:45	juice (carrot)	pc	2	
84	9:45	juice (pear)	pc	1	
84	9:45	juice (apple)	pc	1	
84	9:45	juice (kale)	pc	1.5	
84	11:00	banana	cup	0.5	
84	15:30	banana	cup	0.5	
84	15:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
84	15:30	honey	tbsp	0.5	
84	16:55	pork	oz	6	
84	16:55	capsicum (red)	cup	0.5	
84	16:55	capsicum (green)	cup	0.25	
84	16:55	turmeric	tbsp	0.5	
84	16:55	garlic	tbsp	0.5	
84	16:55	salt	tsp	0.125	
84	16:55	H2O	oz	10	
84	19:30	banana	cup	0.5	
84	20:30	bagel (white)	oz	3	
84	20:30	cinnamon	tbsp	0.125	
84	20:30	brown sugar	tbsp	0.25	
84	20:30	peanut butter	tbsp	1	
84	20:30	jelly (triple berry)	tbsp	2	
84	20:30	soda (La Croix coconut)	oz	12	

APPENDIX C (continued)

84	21:30	bagel (white)	oz	3	
84	21:30	cinnamon	tbsp	0.125	
84	21:30	brown sugar	tbsp	0.25	
84	21:30	peanut butter	tbsp	1	
84	21:30	jelly (triple berry)	tbsp	2	
85	9:00	banana	cup	0.5	25-Mar
85	9:10	H2O	oz	8	
85	11:10	banana	cup	0.5	
85	11:10	soda (La Croix coconut)	oz	12	
85	12:40	bagel (white)	oz	6	
85	12:40	cinnamon	tbsp	0.25	
85	12:40	brown sugar	tbsp	0.5	
85	14:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
85	14:45	honey	tbsp	0.5	
85	15:30	hamburger	oz	4	
85	15:30	bread (wheat)	oz	4	
85	15:30	onion (white)	cup	0.125	
85	15:30	mushroom (white)	cup	0.125	
85	15:30	lettuce (romaine)	cup	0.125	
85	15:30	tomato	cup	0.125	
85	15:30	bacon	oz	2	
85	15:30	ketchup	tbsp	2	
85	15:30	mustard	tbsp	0.5	
85	15:30	mayonaise	tbsp	0.5	
85	15:30	sweet potato	cup	0.25	
85	15:30	canola oil	tbsp	2	
85	15:30	salt	tsp	0.25	
85	15:30	candy (Sour Patch Kids)	pc	15	
85	23:30	bagel (white)	oz	6	
85	23:30	cinnamon	tbsp	0.25	
85	23:30	brown sugar	tbsp	0.5	
85	23:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
85	23:30	honey	tbsp	0.5	
86	8:30	banana	cup	0.5	26-Mar
86	8:45	H2O	oz	10	
86	9:30	strawberry	cup	1	
86	9:30	juice (pomegranate blackberry)	oz	8	
86	11:25	candy (Sour Patch Kids)	pc	12	

APPENDIX C (continued)

86	11:25	H2O	oz	16	
86	12:30	pork	oz	6	
86	12:30	onion (white)	cup	0.125	
86	12:30	capsicum (green)	cup	0.25	
86	12:30	fennel	cup	0.25	
86	12:30	salt	tsp	0.125	
86	12:30	garlic pepper	tbsp	0.25	
86	13:45	candy (Sour Patch Kids)	pc	6	
86	15:40	bagel (white)	oz	3	
86	15:40	cinnamon	tbsp	0.25	
86	15:40	brown sugar	tbsp	0.5	
86	15:40	butter	tbsp	0.5	
86	21:15	bread (soda)	oz	5	
86	21:15	squash (butternut)	cup	0.5	
86	21:15	H2O	oz	16	
86	21:15	soy milk	oz	4	
86	21:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
86	21:15	honey	tbsp	0.5	
87	0:30	bagel (white)	oz	3	27-Mar
87	0:30	cinnamon	tbsp	0.25	
87	0:30	brown sugar	tbsp	0.5	
87	0:30	butter	tbsp	0.5	
87	8:00	bagel (white)	oz	3	
87	8:00	cinnamon	tbsp	0.25	
87	8:00	brown sugar	tbsp	0.5	
87	8:00	butter	tbsp	0.25	
87	10:00	bagel (white)	oz	6	
87	10:00	cinnamon	tbsp	0.5	
87	10:00	brown sugar	tbsp	1	
87	10:00	butter	tbsp	0.5	
87	11:45	soda (La Croix coconut)	oz	12	
87	15:15	hamburger	oz	4	
87	15:15	bread (wheat)	oz	4	
87	15:15	onion (white)	cup	0.125	
87	15:15	lettuce (romaine)	cup	0.125	
87	15:15	bacon	oz	2	
87	15:15	ketchup	tbsp	2	
87	15:15	mustard	tbsp	0.5	
87	15:15	mayonaise	tbsp	0.5	

APPENDIX C (continued)

87	15:15	potato	cup	2	
87	15:15	soda (La Croix coconut)	oz	12	
87	15:15	canola oil	tbsp	4	
87	16:30	candy (Sour Patch Kids)	pc	16	
87	16:30	H2O	oz	16	
87	20:00	shrimp	oz	3	
87	20:00	asparagus	cup	0.25	
87	20:00	pasta (wheat)	cup	1.5	
87	20:00	tomato	cup	0.125	
87	20:00	cream	oz	2	
87	20:00	garlic	tbsp	0.125	
87	20:00	candy (Jolly Rancher)	pc	1	
87	23:00	bagel (white)	oz	6	
87	23:00	cinnamon	tbsp	0.5	
87	23:00	brown sugar	tbsp	1	
87	23:00	butter	tbsp	0.5	
88	10:45	bagel (white)	oz	6	28-Mar
88	10:45	cinnamon	tbsp	0.5	
88	10:45	brown sugar	tbsp	1	
88	10:45	peanut butter	tbsp	2	
88	10:45	jelly (triple berry)	tbsp	3	
88	10:45	soda (La Croix coconut)	oz	12	
88	10:45	candy (Sour Patch Kids)	pc	8	
88	14:30	bread (wheat)	oz	8	
88	14:30	pepperoni	oz	6	
88	14:30	salami	oz	6	
88	14:30	capsicum (green)	cup	0.125	
88	14:30	black olive	cup	1	
88	14:30	banana pepper	cup	1	
88	14:30	soda (Barq's root beer)	oz	16	
88	14:30	potato	cup	0.125	
88	14:30	sauce (BBQ)	tbsp	0.5	
88	20:30	squash (butternut)	cup	1	
88	20:30	bread (soda)	oz	4	
88	20:30	butter	tbsp	1	
88	20:30	H2O	oz	20	
88	20:30	soy milk	oz	4	
88	20:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
88	20:30	honey	tbsp	0.75	

APPENDIX C (continued)

89	5:10	H2O	oz	16	29-Mar
89	9:25	bagel (white)	oz	6	
89	9:25	cinnamon	tbsp	0.5	
89	9:25	brown sugar	tbsp	1	
89	9:25	peanut butter	tbsp	2	
89	9:25	jelly (triple berry)	tbsp	3	
89	13:15	H2O	oz	8	
89	13:15	bread (wheat)	oz	4	
89	13:15	hamburger	oz	4	
89	13:15	lettuce (romaine)	cup	0.125	
89	13:15	bacon	oz	1	
89	13:15	mushroom (white)	cup	0.25	
89	13:15	ketchup	tbsp	1	
89	13:15	mayonaise	tbsp	1	
89	13:15	mustard	tbsp	0.5	
89	13:15	potato	cup	0.25	
89	13:15	canola oil	tbsp	1	
89	13:15	salt	tbsp	0.25	
89	13:15	H2O	oz	16	
89	16:00	bagel (wheat)	oz	6	
89	16:00	peanut butter	tbsp	2	
89	16:00	jelly (triple berry)	tbsp	3	
89	20:25	H2O	oz	16	
89	21:10	pork	oz	6	
89	21:10	capsicum (red)	cup	0.25	
89	21:10	capsicum (green)	cup	0.25	
89	21:10	onion (white)	cup	0.25	
89	21:10	cumin	tbsp	0.5	
89	21:10	coriander	tbsp	0.25	
89	21:10	garlic pepper	tbsp	0.5	
89	21:10	avocado	cup	0.25	
89	21:10	tortilla (corn)	pc	3	
89	22:30	bread (sourdough)	oz	6	
89	22:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
89	22:30	honey	tbsp	2	
90	9:45	bagel (white)	oz	6	30-Mar
90	9:45	cinnamon	tbsp	0.5	
90	9:45	brown sugar	tbsp	1	
90	9:45	butter	tbsp	0.5	

APPENDIX C (continued)

90	9:45	H2O	oz	16	
90	12:45	bread (sourdough)	oz	6	
90	12:45	butter	tbsp	0.5	
90	12:45	jelly (triple berry)	tbsp	1	
90	12:45	honey	tbsp	0.75	
90	12:45	H2O	oz	16	
90	14:00	juice (cucumber)	pc	1	
90	14:00	juice (fennel stalk)	pc	2	
90	14:00	juice (apple)	pc	1	
90	14:00	juice (carrot)	pc	1	
90	14:00	juice (kale)	pc	2	
90	15:30	bagel (wheat)	oz	6	
90	15:30	jelly (triple berry)	tbsp	2	
90	15:30	peanut butter	tbsp	2	
90	18:30	H2O	oz	10	
90	21:45	bread (sourdough)	oz	6	
90	21:45	butter	tbsp	0.75	
90	21:45	jelly (triple berry)	tbsp	2	
90	22:30	milk chocolate	pc	2	
90	22:30	caramel	pc	3	
90	22:30	cookie (white)	oz	2	
91	7:10	H2O	oz	16	31-Mar
91	10:30	bagel (white)	oz	6	
91	10:30	cinnamon	tbsp	0.5	
91	10:30	brown sugar	tbsp	1	
91	10:30	jelly (triple berry)	tbsp	2	
91	10:30	peanut butter	tbsp	2	
91	10:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
91	10:30	honey	tbsp	0.5	
91	15:10	bread (sourdough)	oz	6	
91	15:10	honey	tbsp	1	
91	15:10	jelly (triple berry)	tbsp	1.5	
91	15:10	butter	tbsp	0.75	
91	15:10	H2O	oz	8	
91	19:30	bagel (wheat)	oz	6	
91	19:30	butter	tbsp	0.5	
91	19:30	honey	tbsp	0.5	
91	19:30	H2O	oz	16	

APPENDIX C (continued)

91	22:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
91	22:30	honey	tbsp	0.5	
92	4:15	H2O	oz	14	1-Apr
92	8:20	bagel (white)	oz	3	
92	8:20	cinnamon	tbsp	0.25	
92	8:20	brown sugar	tbsp	0.5	
92	13:00	pork	oz	6	
92	13:00	cinnamon	tbsp	0.25	
92	13:00	honey	tbsp	0.25	
92	13:00	H2O	oz	16	
92	16:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
92	16:00	honey	tbsp	0.75	
92	16:00	bread (sourdough)	oz	4	
92	16:00	butter	tbsp	0.5	
92	16:00	squash (butternut)	cup	2	
92	16:00	H2O	oz	8	
92	16:00	soy milk	oz	8	
92	16:00	salt	tsp	0.125	
92	16:00	milk chocolate	pc	2	
92	16:00	caramel	pc	3	
92	16:00	cookie (white)	oz	2	
92	16:00	gum (Big Red)	pc	1	
92	21:00	bread (wheat)	oz	4	
92	21:00	onion (white)	cup	0.125	
92	21:00	mushroom (white)	cup	0.125	
92	21:00	lettuce (romaine)	cup	0.125	
92	21:00	tomato	cup	0.125	
92	21:00	bacon	oz	2	
92	21:00	ketchup	tbsp	2	
92	21:00	mustard	tbsp	0.5	
92	21:00	mayonaise	tbsp	0.5	
92	21:00	sweet potato	cup	0.5	
92	21:00	canola oil	tbsp	1.5	
92	21:00	salt	tsp	0.125	
92	21:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
92	21:00	honey	tbsp	0.75	
92	22:20	H2O	oz	16	
92	22:20	gum (Big Red)	pc	1	

APPENDIX C (continued)

93	11:00	bread (sourdough)	oz	8	2-Apr
93	11:00	peanut butter	tbsp	2	
93	11:00	preserves (cherry)	tbsp	3	
93	11:00	H2O	oz	16	
93	15:00	milk chocolate	pc	2	
93	15:00	wafer (vanilla)	oz	4	
93	15:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
93	15:00	honey	tbsp	0.75	
93	19:15	pork	oz	8	
93	19:15	cinnamon	tbsp	0.5	
93	19:15	salt	tsp	0.125	
93	19:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
93	19:15	honey	tbsp	2	
93	23:10	bagel (white)	oz	6	
93	23:10	butter	tbsp	0.5	
93	23:10	honey	tbsp	1	
93	23:10	H2O	oz	16	
93	23:10	milk chocolate	pc	1	
93	23:10	wafer (vanilla)	oz	2	
94	9:10	H2O	oz	8	3-Apr
94	9:35	bread (sourdough)	oz	6	
94	9:35	preserves (cherry)	tbsp	3	
94	9:35	honey	tbsp	0.5	
94	9:35	butter	tbsp	0.75	
94	9:35	H2O	oz	16	
94	12:00	bread (wheat)	oz	4	
94	12:00	onion (white)	cup	0.125	
94	12:00	mushroom (white)	cup	0.125	
94	12:00	lettuce (romaine)	cup	0.125	
94	12:00	tomato	cup	0.125	
94	12:00	bacon	oz	2	
94	12:00	ketchup	tbsp	2	
94	12:00	mustard	tbsp	0.5	
94	12:00	mayonaise	tbsp	0.5	
94	12:00	potato	cup	0.25	
94	12:00	canola oil	tbsp	2	
94	12:00	salt	tsp	0.25	
94	12:00	soda (La Croix coconut)	oz	12	

APPENDIX C (continued)

94	14:50	milk chocolate	pc	1	
94	14:50	wafer (vanilla)	oz	2	
94	15:55	soda (La Croix coconut)	oz	12	
94	18:40	milk chocolate	pc	1	
94	18:40	wafer (vanilla)	oz	2	
94	18:40	H2O	oz	16	
94	18:40	bread (sourdough)	oz	8	
94	18:40	butter	tbsp	0.5	
94	18:40	honey	tbsp	1	
94	20:45	bread (white)	oz	3	
94	20:45	roast beef	oz	3	
94	20:45	spinach	cup	0.125	
94	20:45	banana pepper	cup	0.25	
94	20:45	soda (La Croix coconut)	oz	4	
94	23:30	bagel (white)	oz	3	
94	23:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
95	9:40	bread (white)	oz	3	4-Apr
95	9:40	roast beef	oz	3	
95	9:40	spinach	cup	0.125	
95	9:40	banana pepper	cup	0.25	
95	9:40	soda (La Croix coconut)	oz	8	
95	11:45	bagel (white)	oz	6	
95	11:45	butter	tbsp	0.5	
95	11:45	honey	tbsp	0.5	
95	11:45	H2O	oz	8	
95	14:00	milk chocolate	pc	1	
95	14:00	wafer (vanilla)	oz	2	
95	18:00	bread (sourdough)	oz	8	
95	18:00	butter	tbsp	0.75	
95	18:00	preserves (cherry)	tbsp	3	
95	18:00	soda (La Croix coconut)	oz	12	
95	22:00	bagel (white)	oz	3	
95	22:00	peanut butter	tbsp	1	
95	22:00	honey	tbsp	0.5	
95	22:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
96	9:45	soda (La Croix coconut)	oz	12	5-Apr
96	9:45	bread (sourdough)	oz	8	
96	9:45	butter	tbsp	0.5	

APPENDIX C (continued)

96	9:45	honey	tbsp	0.5	
96	13:00	avocado	cup	0.5	
96	13:00	onion (white)	cup	0.25	
96	13:00	pork (Torchy's)	oz	6	
96	13:00	cilantro	tbsp	1	
96	13:00	tomatillo salsa	tbsp	2	
96	13:00	H2O	oz	16	
96	14:50	H2O	oz	8	
96	16:15	milk chocolate	pc	3	
96	16:15	wafer (vanilla)	oz	2	
96	16:15	caramel	pc	3	
96	16:15	cookie (white)	oz	2	
96	16:40	milk chocolate	pc	3	
96	16:40	wafer (vanilla)	oz	2	
96	16:40	caramel	pc	3	
96	16:40	cookie (white)	oz	2	
96	20:45	capsicum (green)	cup	0.5	
96	20:45	capsicum (red)	cup	0.5	
96	20:45	olive oil	tbsp	2	
96	20:45	vinegar (red wine)	tbsp	2	
96	20:45	basil	tbsp	0.5	
96	20:45	thyme	tbsp	0.25	
96	20:45	H2O	oz	8	
96	22:10	milk chocolate	pc	3	
96	22:10	wafer (vanilla)	oz	2	
96	22:10	caramel	pc	3	
96	22:10	cookie (white)	oz	2	
97	5:30	H2O	oz	8	6-Apr
97	8:25	H2O	oz	8	
97	9:30	bread (sourdough)	oz	6	
97	9:30	butter	tbsp	0.25	
97	9:30	honey	tbsp	0.75	
97	9:30	soda (La Croix coconut)	oz	12	
97	12:30	hamburger	oz	4	
97	12:30	bread (wheat)	oz	4	
97	12:30	onion (white)	cup	0.125	
97	12:30	mushroom (white)	cup	0.125	
97	12:30	lettuce (romaine)	cup	0.125	
97	12:30	bacon	oz	2	
97	12:30	ketchup	tbsp	2	

APPENDIX C (continued)

97	12:30	mustard	tbsp	0.5	
97	12:30	mayonaise	tbsp	0.5	
97	12:30	sweet potato	cup	1	
97	12:30	canola oil	tbsp	3	
97	12:30	salt	tsp	0.25	
97	12:30	soda (La Croix coconut)	oz	12	
97	15:30	bagel (white)	oz	6	
97	15:30	butter	tbsp	0.25	
97	15:30	honey	tbsp	1	
97	17:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	9	
97	17:20	honey	tbsp	0.5	
97	23:25	banana	cup	0.5	
98	0:05	candy (Jolly Rancher)	pc	1	7-Apr
98	0:10	H2O	oz	8	
98	2:25	bread (sourdough)	oz	5	
98	2:25	butter	tbsp	0.5	
98	2:25	H2O	oz	6	
98	8:45	H2O	oz	10	
98	9:00	bagel (white)	oz	6	
98	9:00	butter	tbsp	0.25	
98	9:00	honey	tbsp	1	
98	9:00	H2O	oz	16	
98	12:00	avocado	cup	0.25	
98	12:00	onion (white)	cup	0.125	
98	12:00	pork (Torchy's)	oz	3	
98	12:00	cilantro	tbsp	0.5	
98	12:00	tomatillo salsa	tbsp	1	
98	12:00	tortilla (corn)	pc	1	
98	14:45	corn meal	oz	9	
98	14:45	pork	oz	3	
98	14:45	beef	oz	3	
98	14:45	apple	cup	0.125	
98	14:45	raisin	cup	0.125	
98	14:45	cinnamon	tbsp	0.125	
98	14:45	sugar	tbsp	1	
98	14:45	cumin	tbsp	0.25	
98	14:45	coriander	tbsp	0.25	
98	14:45	jalapeno	tbsp	0.5	
98	14:45	salt	tbsp	0.25	

APPENDIX C (continued)

98	14:45	canola oil	tbsp	3	
98	14:45	juice (Bolthouse Farms Berry Boost)	oz	8	
98	14:45	soda (La Croix coconut)	oz	12	
98	15:45	chip (Doritos nacho)	oz	5	
98	20:00	bagel (white)	oz	6	
98	20:00	cinnamon	tbsp	0.5	
98	20:00	brown sugar	tbsp	1	
98	20:00	butter	tbsp	0.5	
98	20:00	honey	tbsp	0.75	
98	20:00	H2O	oz	10	
99	2:00	bread (white)	oz	8	8-Apr
99	2:00	honey	tbsp	0.5	
99	2:00	preserves (cherry)	tbsp	2	
99	2:00	butter	tbsp	0.75	
99	12:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
99	12:00	honey	tbsp	0.5	
99	12:05	avocado	cup	0.25	
99	12:05	onion (white)	cup	0.125	
99	12:05	pork (Torchy's)	oz	3	
99	12:05	cilantro	tbsp	0.5	
99	12:05	tomatillo salsa	tbsp	1	
99	12:05	H2O	oz	16	
99	14:00	bread (white)	oz	8	
99	14:00	preserves (cherry)	tbsp	2	
99	14:00	peanut butter	tbsp	2	
99	14:00	soda (La Croix coconut)	oz	12	
99	18:00	pork	oz	5	
99	18:00	capsicum (red)	cup	0.25	
99	18:00	capsicum (green)	cup	0.5	
99	18:00	cumin	tbsp	0.5	
99	18:00	coriander	tbsp	0.25	
99	18:00	onion (white)	cup	0.25	
99	18:00	garlic pepper	tbsp	0.5	
99	18:00	olive oil	tbsp	1	
99	19:20	bread (white)	oz	8	
99	19:20	preserves (cherry)	tbsp	1	
99	19:20	peanut butter	tbsp	2	
99	19:20	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	

APPENDIX C (continued)

99	19:20	honey	tbsp	0.5	
99	23:00	bagel (white)	oz	6	
99	23:00	cinnamon	tbsp	0.5	
99	23:00	brown sugar	tbsp	1	
99	23:00	butter	tbsp	0.5	
100	6:45	H2O	oz	6	9-Apr
100	9:35	bread (white)	oz	8	
100	9:35	butter	tbsp	0.5	
100	9:35	honey	tbsp	0.75	
100	10:40	H2O	oz	4	
100	12:00	corn meal	oz	6	
100	12:00	pork	oz	3	
100	12:00	beef	oz	3	
100	12:00	cumin	tbsp	0.5	
100	12:00	coriander	tbsp	0.25	
100	12:00	jalapeno	tbsp	0.5	
100	12:00	salt	tbsp	0.25	
100	12:00	canola oil	tbsp	2	
100	14:20	soda (La Croix coconut)	oz	4	
100	16:30	bread (white)	oz	8	
100	16:30	preserves (cherry)	tbsp	2	
100	16:30	peanut butter	tbsp	2	
100	17:30	milk chocolate	pc	3	
100	17:30	caramel	pc	3	
100	17:30	cookie (white)	oz	2	
100	17:30	wafer (vanilla)	oz	2	
100	22:10	flour (rice)	cup	1.5	
100	22:10	capsicum (red)	cup	0.25	
100	22:10	mushroom (shitake)	cup	0.25	
100	22:10	artichoke	cup	0.5	
101	7:15	H2O	oz	14	10-Apr
101	9:10	bagel (white)	oz	6	
101	9:10	cinnamon	tbsp	0.5	
101	9:10	brown sugar	tbsp	1	
101	9:10	butter	tbsp	0.75	
101	9:40	H2O	oz	16	
101	11:50	pork	oz	16	
101	11:50	cumin	tbsp	0.25	
101	11:50	coriander	tbsp	0.25	
101	11:50	garlic pepper	tbsp	0.5	

APPENDIX C (continued)

101	11:50	olive oil	tbsp	1	
101	13:30	H2O	oz	4	
101	13:30	caramel	oz	3	
101	13:30	canola oil	tbsp	1	
101	13:30	sugar	tbsp	2	
101	13:30	apple	cup	0.125	
101	13:30	raisin	cup	0.125	
101	17:00	bread (white)	oz	4	
101	17:00	preserves (cherry)	tbsp	1	
101	17:00	peanut butter	tbsp	2	
101	18:25	H2O	oz	12	
101	20:40	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
101	20:40	honey	tbsp	0.5	
101	21:40	bread (white)	oz	6	
101	21:40	steak	oz	6	
101	21:40	capsicum (green)	cup	0.125	
101	21:40	capsicum (red)	cup	0.125	
101	21:40	onion (white)	cup	0.125	
101	21:40	mushroom (white)	cup	0.125	
101	21:40	chicken	oz	4	
101	21:40	sauce (hotwing)	tbsp	1	
101	21:40	H2O	oz	12	
101	21:40	potato	cup	0.25	
101	21:40	canola oil	tbsp	0.5	
101	21:40	salt	tsp	0.25	
102	6:10	bagel (white)	oz	3	11-Apr
102	6:10	cinnamon	tbsp	0.25	
102	6:10	brown sugar	tbsp	0.5	
102	6:10	H2O	oz	4	
102	9:00	bagel (white)	oz	6	
102	9:00	cinnamon	tbsp	0.5	
102	9:00	brown sugar	tbsp	1	
102	9:00	butter	tbsp	0.5	
102	9:00	H2O	oz	8	
102	10:40	H2O	oz	8	
102	12:15	egg	oz	6	
102	12:15	capsicum (red)	cup	0.5	
102	12:15	capsicum (green)	cup	0.5	
102	12:15	basil	tbsp	0.5	

APPENDIX C (continued)

102	12:15	salt	tsp	0.125	
102	12:15	pepper	tsp	0.125	
102	12:15	tortilla (flour)	oz	6	
102	13:30	H2O	oz	12	
102	20:35	bread (white)	oz	6	
102	20:35	butter	tbsp	0.5	
102	20:35	jelly (raspberry)	tbsp	3	
102	23:15	bread (white)	oz	6	
102	23:15	butter	tbsp	0.75	
102	23:15	jelly (raspberry)	tbsp	3	
103	6:20	bread (white)	oz	4	12-Apr
103	6:20	garlic	tbsp	1	
103	9:20	bread (white)	oz	6	
103	9:20	garlic	tbsp	1.5	
103	9:20	H2O	oz	8	
103	10:40	H2O	oz	8	
103	11:40	pork	oz	6	
103	11:40	cinnamon	tbsp	0.5	
103	11:40	honey	tbsp	1	
103	14:40	sweet potato	cup	1.5	
103	14:40	butter	tbsp	1	
103	14:40	brown sugar	tbsp	3	
103	17:10	bread (white)	oz	6	
103	17:10	jelly (raspberry)	tbsp	2	
103	21:00	bread (white)	oz	8	
103	21:00	garlic	tbsp	2	
103	21:00	H2O	oz	8	
103	21:40	H2O	oz	24	
104	7:05	bread (white)	oz	6	13-Apr
104	7:05	preserves (apricot)	tbsp	2	
104	7:05	jelly (raspberry)	tbsp	1	
104	7:05	H2O	oz	10	
104	9:30	bagel (white)	oz	3	
104	9:30	cinnamon	tbsp	0.25	
104	9:30	brown sugar	tbsp	0.5	
104	9:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
104	9:30	honey	tbsp	0.5	
104	12:30	hamburger	oz	4	
104	12:30	bread (wheat)	oz	4	

APPENDIX C (continued)

104	12:30	onion (white)	cup	0.125	
104	12:30	lettuce (romaine)	cup	0.125	
104	12:30	tomato	cup	0.125	
104	12:30	bacon	oz	2	
104	12:30	ketchup	tbsp	2	
104	12:30	mustard	tbsp	0.5	
104	12:30	mayonaise	tbsp	0.5	
104	12:30	potato	cup	0.5	
104	12:30	canola oil	tbsp	2	
104	12:30	salt	tsp	0.5	
104	16:10	soda (La Croix coconut)	oz	12	
104	20:00	bagel (white)	oz	3	
104	20:00	cinnamon	tbsp	0.25	
104	20:00	brown sugar	tbsp	0.5	
104	20:00	butter	tbsp	0.5	
104	20:40	H2O	oz	8	
104	21:10	rice	oz	3	
104	21:10	salmon	oz	2	
104	21:10	avocado	cup	0.125	
104	23:20	rice	oz	3	
104	23:20	salmon	oz	2	
104	23:20	avocado	cup	0.125	
105	9:00	bagel (white)	oz	6	14-Apr
105	9:00	cinnamon	tbsp	0.5	
105	9:00	brown sugar	tbsp	1	
105	9:00	H2O	oz	16	
105	12:30	arugala	cup	2	
105	12:30	salmon	oz	3	
105	12:30	potato	cup	0.5	
105	12:30	green bean	cup	0.25	
105	12:30	black olive	cup	0.125	
105	12:30	shallot	tbsp	2	
105	12:30	vinegar (red wine)	tbsp	2	
105	12:30	olive oil	tbsp	2	
105	12:30	salt	tsp	0.125	
105	12:30	pepper	tsp	0.125	
105	12:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
105	12:30	honey	tbsp	0.5	
105	12:30	garlic	tbsp	0.5	

APPENDIX C (continued)

105	16:30	pasta (wheat)	cup	0.5	
105	16:30	cheese (parmasean)	tbsp	0.5	
105	16:30	capsicum (red)	cup	0.25	
105	16:30	steak	oz	4	
105	16:30	lettuce (romaine)	cup	0.75	
105	16:30	garlic	tbsp	0.5	
105	16:30	oregano	tbsp	0.5	
105	16:30	parsley	tbsp	0.5	
105	16:30	olive oil	tbsp	0.5	
105	16:30	onion (white)	cup	0.125	
105	16:30	tortilla (flour)	oz	3	
105	16:30	soda (La Croix coconut)	oz	4	
105	20:40	bagel (white)	oz	6	
105	20:40	cinnamon	tbsp	0.5	
105	20:40	brown sugar	tbsp	1	
105	20:40	H2O	oz	4	
105	21:50	H2O	oz	16	
106	5:30	H2O	oz	6	15-Apr
106	5:50	H2O	oz	10	
106	11:05	bagel (white)	oz	6	
106	11:05	cinnamon	tbsp	0.5	
106	11:05	brown sugar	tbsp	1	
106	11:05	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
106	11:05	honey	tbsp	0.5	
106	15:50	pork	oz	6	
106	15:50	thyme	tbsp	0.25	
106	15:50	rosemary	tbsp	0.25	
106	15:50	honey	tbsp	0.75	
106	22:30	bread (white)	oz	6	
106	22:30	preserves (apricot)	tbsp	3	
107	10:00	bagel (white)	oz	6	16-Apr
107	10:00	cinnamon	tbsp	0.5	
107	10:00	brown sugar	tbsp	1	
107	10:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
107	10:00	honey	tbsp	0.5	
107	10:30	H2O	oz	16	
107	13:00	pork	oz	10	
107	13:00	capsicum (green)	cup	0.75	

APPENDIX C (continued)

107	13:00	capsicum (red)	cup	0.75	
107	13:00	cumin	tbsp	0.5	
107	13:00	coriander	tbsp	0.5	
107	13:00	vinegar (red wine)	tbsp	2	
107	13:00	salt	tsp	0.125	
107	13:00	garlic pepper	tbsp	0.5	
107	13:00	olive oil	tbsp	1	
107	19:10	bread (white)	oz	6	
107	19:10	honey	tbsp	1	
107	20:10	bread (white)	oz	6	
107	20:10	honey	tbsp	1	
107	20:40	H2O	oz	16	
108	4:30	H2O	oz	16	17-Apr
108	9:50	bagel (white)	oz	6	
108	9:50	cinnamon	tbsp	0.5	
108	9:50	brown sugar	tbsp	1	
108	9:50	H2O	oz	16	
108	12:00	chicken	oz	4	
108	12:00	italian spices	tbsp	0.5	
108	12:00	crushed red pepper	tbsp	0.25	
108	14:30	H2O	oz	16	
108	14:30	gum (watermelon)	pc	1	
108	15:45	bread (white)	oz	8	
108	15:45	garlic	tbsp	2	
108	15:45	H2O	oz	16	
108	19:30	bread (white)	oz	6	
108	19:30	honey	tbsp	0.75	
108	21:20	H2O	oz	16	
108	21:20	avocado	cup	0.5	
109	2:55	H2O	oz	8	18-Apr
109	9:45	bagel (white)	oz	6	
109	9:45	cinnamon	tbsp	0.5	
109	9:45	brown sugar	tbsp	1	
109	9:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
109	9:45	honey	tbsp	0.5	
109	12:00	salmon	oz	5	
109	12:00	lemon pepper	tbsp	0.25	
109	12:00	parsley	tbsp	0.125	
109	12:00	olive oil	tbsp	0.5	

APPENDIX C (continued)

109	12:00	H2O	oz	16	
109	13:05	salmon	oz	5	
109	13:05	lemon pepper	tbsp	0.25	
109	13:05	parsley	tbsp	0.125	
109	13:05	olive oil	tbsp	0.5	
109	13:05	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	6	
109	18:30	bread (white)	oz	6	
109	18:30	jelly (raspberry)	tbsp	3	
109	21:30	neon sour gummy worms	pc	9	
110	5:10	H2O	oz	4	19-Apr
110	8:45	H2O	oz	4	
110	9:30	cake (white)	oz	3	
110	9:30	juice (lemon)	tbsp	1	
110	9:30	poppy seed	tbsp	0.5	
110	9:30	sugar	tbsp	3	
110	9:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	8	
110	9:30	honey	tbsp	0.5	
110	9:30	strawberry	cup	1.5	
110	12:00	H2O	oz	8	
110	12:30	beef	oz	6	
110	12:30	capsicum (green)	cup	1.5	
110	12:30	capsicum (red)	cup	0.25	
110	12:30	salt	tsp	0.125	
110	12:30	pepper	tsp	0.125	
110	12:30	onion (white)	cup	0.125	
110	12:30	olive oil	tbsp	1.5	
110	12:30	neon sour gummy worms	pc	6	
110	12:30	H2O	oz	16	
110	12:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
110	12:30	honey	tbsp	0.5	
110	15:30	neon sour gummy worms	pc	6	
110	15:30	H2O	oz	16	
110	18:00	banana	cup	1	
110	18:00	juice (pomegranate blackberry)	oz	4	
110	18:00	strawberry	cup	1	
110	20:30	tilapia	oz	5	
110	20:30	vinegar (balsamic)	tbsp	0.5	
110	20:30	sage	tbsp	0.25	

APPENDIX C (continued)

110	20:30	celery seed	tbsp	0.25	
110	20:30	quinoa	cup	1	
110	20:30	olive oil	tbsp	2	
110	20:30	parsley	tbsp	0.25	
110	20:30	oregano	tbsp	0.25	
110	20:30	juice (lemon)	tbsp	2	
110	20:30	brussel sprout	cup	1	
110	20:30	wine (white)	tbsp	1	
110	20:30	pepper	tsp	0.25	
110	20:30	salt	tsp	0.125	
111	6:25	H2O	oz	6	20-Apr
111	10:00	bread (wheat)	oz	8	
111	10:00	margerine	tbsp	0.25	
111	10:00	honey	tbsp	0.75	
111	11:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
111	11:30	honey	tbsp	0.5	
111	11:30	neon sour gummy worms	pc	6	
111	12:00	pork	oz	6	
111	12:00	italian spices	tbsp	1	
111	12:00	bread (wheat)	oz	6	
111	12:00	honey dijon mustard	tbsp	1.5	
111	12:00	H2O	oz	16	
111	20:30	salmon	oz	8	
111	20:30	potato	cup	0.75	
111	20:30	asparagus	cup	1	
111	20:30	salt	tsp	0.25	
111	20:30	pepper	tsp	0.5	
111	20:30	margerine	tbsp	1	
111	20:30	lemon pepper	tbsp	0.5	
111	20:30	basil	tbsp	0.5	
111	20:30	garlic	tbsp	0.25	
111	20:30	olive oil	tbsp	1	
111	20:30	juice (lime)	tbsp	0.5	
111	20:30	parsley	tbsp	0.25	
111	20:30	H2O	oz	16	
112	0:45	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	21-Apr
112	0:45	honey	tbsp	0.25	
112	1:45	bread (wheat)	oz	8	

APPENDIX C (continued)

112	1:45	honey	tbsp	0.75	
112	1:45	margerine	tbsp	0.25	
112	10:00	H2O	oz	2	
112	10:30	cake (white)	oz	3	
112	10:30	orange	cup	0.125	
112	10:30	pineapple	cup	0.125	
112	10:30	sugar	tbsp	3	
112	10:30	strawberry	cup	0.75	
112	12:45	neon sour gummy worms	pc	6	
112	13:50	H2O	oz	40	
112	15:45	bread (white)	oz	8	
112	15:45	turkey	oz	6	
112	15:45	mayonaise	tbsp	2	
112	15:45	sauce (cranberry)	tbsp	4	
112	15:45	chip (corn tortilla)	cup	2	
112	15:45	tomato	cup	0.125	
112	15:45	onion (white)	tbsp	2	
112	15:45	cilantro	tbsp	0.5	
112	15:45	H2O	oz	12	
112	15:45	lettuce (romaine)	cup	0.5	
112	19:45	H2O	oz	12	
112	23:00	steak	oz	5	
112	23:00	capsicum (green)	cup	0.75	
112	23:00	capsicum (red)	cup	0.5	
112	23:00	onion (white)	cup	0.25	
112	23:00	mushroom (white)	cup	0.25	
112	23:00	garlic pepper	tbsp	0.5	
112	23:00	salt	tsp	0.125	
112	23:00	H2O	oz	16	
112	23:50	sugar	tbsp	3	
112	23:50	graham cracker	oz	2	
112	23:50	milk chocolate	pc	3	
113	12:00	peach	cup	1	22-Apr
113	12:00	blackberry	cup	0.75	
113	12:00	juice (pomegranate blackberry)	oz	8	
113	13:00	H2O	oz	12	
113	19:00	ground turkey	oz	5	
113	19:00	bread (wheat)	oz	4	
113	19:00	dressing (honey mustard)	tbsp	2	
113	19:00	mushroom (white)	cup	0.25	

APPENDIX C (continued)

113	19:00	lettuce (romaine)	cup	0.25	
113	19:00	bacon	oz	2	
113	19:00	sweet potato	cup	0.25	
113	19:00	canola oil	tbsp	1	
113	19:00	H2O	oz	16	
114	0:15	bread (wheat)	oz	8	23-Apr
114	0:15	margerine	tbsp	0.25	
114	0:15	honey	tbsp	0.5	
114	4:00	H2O	oz	8	
114	10:45	tilapia	oz	3	
114	10:45	fennel	cup	0.75	
114	10:45	cabbage (green)	cup	0.75	
114	10:45	honey chipotle powder	tbsp	0.75	
114	10:45	olive oil	tbsp	0.375	
114	10:45	garlic	tbsp	0.375	
114	10:45	cumin	tbsp	0.15	
114	10:45	cinnamon	tbsp	0.15	
114	10:45	sour cream	cup	0.15	
114	10:45	pepper	tsp	0.15	
114	10:45	salt	tsp	0.15	
114	10:45	juice (lime)	tbsp	0.15	
114	10:45	H2O	oz	8	
114	11:15	H2O	oz	16	
114	13:25	juice (kale)	pc	2	
114	13:25	juice (pear)	pc	1	
114	13:25	juice (cucumber)	pc	0.5	
114	13:25	juice (carrot)	pc	3	
114	17:15	pork	oz	6	
114	17:15	italian spices	tbsp	1	
114	17:15	bread (wheat)	oz	3	
114	17:15	honey dijon mustard	tbsp	2	
114	22:30	milk chocolate	pc	3	
114	22:45	strawberry	cup	1.5	
114	22:45	sugar	tbsp	1	
114	22:45	H2O	oz	8	
115	0:55	H2O	oz	2	24-Apr
115	5:05	H2O	oz	6	
115	9:00	H2O	oz	6	
115	10:00	bread (wheat)	oz	8	
115	10:00	margerine	tbsp	0.25	

APPENDIX C (continued)

115	10:00	honey	tbsp	0.75	
115	10:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
115	12:15	steak	oz	6	
115	12:15	capsicum (green)	cup	1	
115	12:15	onion (white)	tbsp	3	
115	12:15	garlic pepper	tbsp	1	
115	12:15	salt	tsp	0.125	
115	12:15	H2O	oz	16	
115	13:50	milk chocolate	pc	2	
115	13:50	H2O	oz	16	
115	15:30	graham cracker	oz	3	
115	19:25	pasta (wheat)	cup	2	
115	19:25	olive oil	tbsp	2	
115	19:25	basil	tbsp	0.5	
115	19:25	garlic	tbsp	2	
115	19:25	pepper	tsp	1	
115	19:25	salt	tsp	0.125	
115	19:25	parsley	tbsp	0.25	
115	19:25	capsicum (red)	cup	0.5	
115	19:25	juice (lime)	tbsp	0.5	
115	19:25	cheese (parmasean)	tbsp	1	
115	19:25	oregano	tbsp	0.25	
116	1:30	H2O	oz	6	25-Apr
116	9:20	H2O	oz	4	
116	9:45	bread (wheat)	oz	8	
116	9:45	margerine	tbsp	0.25	
116	9:45	jelly (raspberry)	tbsp	2	
116	9:45	preserves (apricot)	tbsp	2	
116	9:45	H2O	oz	12	
116	10:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
116	10:30	honey	tbsp	0.75	
116	12:00	tilapia	oz	3	
116	12:00	fennel	cup	0.75	
116	12:00	cabbage (green)	cup	0.75	
116	12:00	honey chipotle powder	tbsp	0.75	
116	12:00	olive oil	tbsp	0.375	
116	12:00	garlic	tbsp	0.375	
116	12:00	cumin	tbsp	0.15	

APPENDIX C (continued)

116	12:00	cinnamon	tbsp	0.15	
116	12:00	sour cream	cup	0.15	
116	12:00	pepper	tsp	0.15	
116	12:00	salt	tsp	0.15	
116	12:00	juice (lime)	tbsp	0.15	
116	12:00	H2O	oz	16	
116	13:40	graham cracker	oz	6	
116	13:40	H2O	oz	16	
116	21:40	bread (white)	oz	4	
116	21:40	turkey	oz	3	
116	21:40	mayonaise	tbsp	1	
116	21:40	sauce (cranberry)	tbsp	2	
116	21:40	chip (corn tortilla)	cup	1	
116	21:40	tomato	cup	0.1	
116	21:40	cilantro	tbsp	0.25	
116	21:40	onion (white)	tbsp	1	
116	21:40	H2O	oz	2	
116	21:40	lettuce (romaine)	cup	0.25	
117	3:10	banana	cup	0.75	26-Apr
117	3:10	H2O	oz	6	
117	8:20	H2O	oz	6	
117	8:50	H2O	oz	8	
117	9:10	bread (wheat)	oz	6	
117	9:10	margerine	tbsp	0.25	
117	9:10	honey	tbsp	1	
117	11:30	bread (white)	oz	4	
117	11:30	turkey	oz	3	
117	11:30	mayonaise	tbsp	1	
117	11:30	sauce (cranberry)	tbsp	2	
117	11:30	chip (corn tortilla)	cup	1	
117	11:30	tomato	cup	0.1	
117	11:30	cilantro	tbsp	0.25	
117	11:30	onion (white)	tbsp	1	
117	11:30	lettuce (romaine)	cup	0.25	
117	12:30	H2O	oz	16	
117	19:45	salmon	oz	10	
117	19:45	garlic pepper	tbsp	1	
117	19:45	coriander	tbsp	0.25	
117	19:45	salt	tsp	0.125	
117	19:45	olive oil	tbsp	0.5	

APPENDIX C (continued)

117	19:45	sweet potato	cup	1	
117	19:45	honey chipotle powder	tbsp	1	
117	19:45	honey dijon mustard	tbsp	0.5	
118	9:45	H2O	oz	2	27-Apr
118	10:00	H2O	oz	16	
118	10:00	bread (white)	oz	6	
118	10:00	margerine	tbsp	0.25	
118	10:00	honey	tbsp	0.5	
118	10:00	cinnamon	tbsp	0.25	
118	10:00	sugar	tbsp	0.125	
118	12:00	capsicum (green)	cup	1.5	
118	12:00	beef	oz	4	
118	12:00	tomato	cup	0.25	
118	12:00	quinoa	cup	0.125	
118	12:00	oregano	tbsp	0.25	
118	12:00	vinegar (balsamic)	tbsp	2	
118	12:00	garlic pepper	tbsp	0.25	
118	12:00	H2O	oz	16	
118	12:00	cake (white)	oz	3	
118	12:00	juice (lemon)	tbsp	0.25	
118	12:00	sugar	tbsp	3	
118	12:00	egg	oz	0.25	
118	12:00	poppy seed	tbsp	0.25	
118	15:20	roast beef	oz	6	
118	15:20	bread (wheat)	oz	8	
118	15:20	honey dijon mustard	tbsp	2	
118	15:20	au jus	oz	6	
118	19:35	cinnamon	tbsp	1	
118	19:35	sugar	tbsp	1	
118	19:35	bread (white)	oz	6	
118	19:35	margerine	tbsp	0.125	
118	20:15	cinnamon	tbsp	1	
118	20:15	sugar	tbsp	1	
118	20:15	bread (white)	oz	6	
118	20:15	margerine	tbsp	0.125	
118	20:25	H2O	oz	16	
119	0:55	salmon	oz	6	28-Apr
119	0:55	lemon pepper	tbsp	0.5	
119	0:55	basil	tbsp	0.25	
119	0:55	H2O	oz	6	

APPENDIX C (continued)

119	9:00	H2O	oz	4	
119	10:45	bread (potato)	oz	6	
119	10:45	margerine	tbsp	0.25	
119	10:45	honey	tbsp	1	
119	11:45	salmon	oz	4	
119	11:45	lemon pepper	tbsp	0.5	
119	11:45	basil	tbsp	0.25	
119	11:45	H2O	oz	16	
119	14:30	salmon	oz	8	
119	14:30	lemon pepper	tbsp	1	
119	14:30	basil	tbsp	0.5	
119	17:00	steak	oz	6	
119	17:00	capsicum (green)	cup	0.5	
119	17:00	salt	tsp	0.125	
119	18:30	roast beef	oz	4	
119	18:30	bread (white)	oz	3	
119	18:30	onion (white)	cup	0.25	
119	18:30	pickle (dill)	cup	0.125	
119	18:30	au jus	oz	3	
119	18:30	H2O	oz	16	
119	21:00	bread (white)	oz	3	
119	21:00	hamburger	oz	3	
119	21:00	bacon	oz	2	
119	21:00	onion (white)	cup	0.125	
119	21:00	honey dijon mustard	tbsp	1	
119	21:00	H2O	oz	16	
120	5:30	H2O	oz	4	29-Apr
120	11:30	bread (potato)	oz	8	
120	11:30	margerine	tbsp	1.5	
120	11:30	jelly (raspberry)	tbsp	2	
120	11:30	preserves (apricot)	tbsp	2	
120	13:00	roast beef	oz	4	
120	13:00	bread (white)	oz	3	
120	13:00	onion (white)	tbsp	0.25	
120	13:00	au jus	oz	3	
120	13:00	milk chocolate	pc	1	
120	13:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
120	13:00	honey	tbsp	0.5	
120	16:00	H2O	oz	4	

APPENDIX C (continued)

120	16:00	milk chocolate	pc	1	
120	17:00	bread (white)	oz	3	
120	17:00	hamburger	oz	3	
120	17:00	tomato	tbsp	0.125	
120	17:00	mango	tbsp	0.125	
120	17:00	peach	tbsp	0.125	
120	17:00	onion (white)	tbsp	0.125	
120	17:00	H2O	oz	16	
120	21:00	juice (cucumber)	pc	1	
120	21:00	juice (pear)	pc	1	
120	21:00	juice (kale)	pc	2	
120	21:00	juice (carrot)	pc	2	
120	21:00	juice (celery)	pc	2	
120	21:00	juice (fennel stalk)	pc	1	
120	22:30	chip (corn tortilla)	cup	1.5	
120	22:30	salmon	oz	8	
120	22:30	honey chipotle powder	tbsp	0.5	
120	22:30	H2O	oz	16	
121	9:20	bread (potato)	oz	5	30-Apr
121	9:20	margerine	tbsp	0.25	
121	9:20	honey	tbsp	0.5	
121	9:20	H2O	oz	4	
121	10:20	bread (potato)	oz	3	
121	10:20	margerine	tbsp	0.125	
121	10:20	honey	tbsp	0.25	
121	11:20	hamburger	oz	3	
121	11:20	avocado	cup	0.5	
121	11:20	capsicum (green)	cup	0.25	
121	11:20	bacon	oz	2	
121	11:20	tomato	cup	0.125	
121	11:20	honey dijon mustard	tbsp	0.5	
121	17:30	salmon	oz	4	
121	17:30	honey chipotle powder	tbsp	0.25	
121	17:30	honey	tbsp	0.25	
121	17:30	quinoa	cup	0.25	
121	22:45	H2O	oz	2	
121	23:35	bagel (white)	oz	3	
121	23:35	cinnamon	tbsp	0.125	
121	23:35	brown sugar	tbsp	0.25	
122	0:15	bread (white)	oz	3	1-May

APPENDIX C (continued)

122	0:15	cinnamon	tbsp	0.25	
122	0:15	H2O	oz	8	
122	10:10	bread (potato)	oz	8	
122	10:10	margerine	tbsp	0.25	
122	10:10	honey	tbsp	1.5	
122	10:10	H2O	oz	24	
122	12:30	sweet potato	cup	1.5	
122	12:30	olive oil	tbsp	1	
122	12:30	honey chipotle powder	tbsp	0.5	
122	12:30	honey dijon mustard	tbsp	1	
122	12:30	capsicum (green)	cup	1	
122	12:30	hamburger	oz	8	
122	12:30	tomato	cup	0.25	
122	12:30	onion (white)	cup	0.25	
122	12:30	pepper	tbsp	0.25	
122	12:30	salt	tsp	0.125	
122	12:30	vinegar (balsamic)	tbsp	1	
122	12:30	H2O	oz	16	
122	12:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
122	12:30	honey	tbsp	0.5	
122	18:45	pasta (wheat)	cup	1.5	
122	18:45	chicken	oz	6	
122	18:45	tomato	cup	0.125	
122	18:45	olive oil	tbsp	3	
122	18:45	juice (lemon)	tbsp	0.5	
122	18:45	pepper	tbsp	0.125	
123	3:35	H2O	oz	6	2-May
123	10:30	bread (white)	oz	12	
123	10:30	cinnamon	tbsp	1	
123	10:30	H2O	oz	16	
123	12:25	H2O	oz	16	
123	12:25	pork	oz	5	
123	12:25	capsicum (green)	cup	0.25	
123	12:25	cumin	tbsp	0.25	
123	12:25	garlic pepper	tbsp	0.25	
123	12:25	coriander	tbsp	0.25	
123	14:00	pasta (wheat)	cup	1	
123	14:00	chicken	oz	2	
123	14:00	olive oil	tbsp	3	

APPENDIX C (continued)

123	14:00	juice (lemon)	tbsp	0.5	
123	14:00	pepper	tbsp	0.125	
123	14:00	H2O	oz	16	
123	19:20	bagel (white)	oz	8	
123	19:20	cinnamon	tbsp	0.5	
123	19:20	raisin	cup	0.125	
124	8:00	H2O	oz	16	3-May
124	10:05	bread (sourdough)	oz	8	
124	10:05	margerine	tbsp	0.125	
124	10:05	honey	tbsp	0.5	
124	10:05	jelly (raspberry)	tbsp	1	
124	10:05	bacon	oz	2	
124	10:05	H2O	oz	16	
124	13:00	pork	oz	6	
124	13:00	H2O	oz	16	
124	15:45	bagel (white)	oz	8	
124	15:45	cinnamon	tbsp	0.5	
124	15:45	raisin	tbsp	0.125	
124	15:45	margerine	tbsp	0.25	
124	17:30	salmon	oz	5	
124	17:30	lemon pepper	tbsp	0.5	
124	17:30	basil	tbsp	0.25	
124	23:55	H2O	oz	4	
125	9:45	bread (white)	oz	8	4-May
125	9:45	strawberry	cup	0.125	
125	9:45	banana	cup	0.125	
125	9:45	sugar	tbsp	2	
125	9:45	margerine	tbsp	0.25	
125	9:45	bacon	oz	2	
125	13:00	pork	oz	6	
125	13:00	black olive	cup	0.25	
125	13:00	quinoa	cup	0.25	
125	13:00	salt	tsp	0.125	
125	16:00	bread (white)	oz	8	
125	16:00	strawberry	cup	0.125	
125	16:00	banana	cup	0.125	
125	16:00	sugar	tbsp	2	
125	16:00	margerine	tbsp	0.25	
125	18:00	hard cider	oz	12	
125	21:00	chicken	oz	4	

APPENDIX C (continued)

125	21:00	sauce (BBQ)	tbsp	2	
125	21:30	pork	oz	3	
125	21:30	tomato	cup	0.125	
125	21:30	onion (white)	cup	0.05	
125	21:30	capsicum (red)	cup	0.05	
125	21:30	capsicum (green)	cup	0.05	
125	21:30	bread (white)	oz	3	
125	21:30	italian spices	tbsp	0.325	
126	10:15	bread (white)	oz	8	5-May
126	10:15	strawberry	cup	0.125	
126	10:15	banana	cup	0.125	
126	10:15	sugar	tbsp	2	
126	10:15	margerine	tbsp	0.25	
126	10:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
126	10:15	honey	tbsp	0.5	
126	13:30	tomato	cup	0.25	
126	13:30	pork	oz	4	
126	13:30	bread (white)	oz	4	
126	13:30	italian spices	tbsp	0.5	
126	13:30	onion (white)	cup	0.125	
126	13:30	capsicum (red)	cup	0.05	
126	13:30	capsicum (green)	cup	0.05	
126	13:30	mushroom (white)	cup	0.2	
126	13:30	candy (Sour Patch Kids)	pc	20	
126	13:40	gum (watermelon)	pc	1	
126	18:45	bread (white)	oz	3	
126	18:45	onion (white)	cup	0.125	
126	18:45	roast beef	oz	3	
126	18:45	au jus	oz	3	
126	18:45	cheese (swiss)	oz	2	
126	20:30	H2O	oz	12	
126	21:00	H2O	oz	12	
126	21:40	bread (white)	oz	3	
126	21:40	onion (white)	cup	0.125	
126	21:40	roast beef	oz	3	
126	21:40	au jus	oz	3	
126	22:00	candy (Sour Patch Kids)	pc	6	
126	23:00	H2O	oz	12	
127	9:10	H2O	oz	4	6-May

APPENDIX C (continued)

127	10:20	bread (white)	oz	8	
127	10:20	strawberry	cup	0.125	
127	10:20	banana	cup	0.125	
127	10:20	sugar	tbsp	2	
127	10:20	margerine	tbsp	0.25	
127	15:30	bread (wheat)	oz	8	
127	15:30	chicken	oz	6	
127	15:30	bacon	oz	4	
127	15:30	alfalfa sprout	cup	0.25	
127	15:30	avocado	cup	0.5	
127	15:30	dressing (honey mustard)	tbsp	4	
127	15:30	H2O	oz	12	
127	20:30	bagel (white)	oz	8	
127	20:30	cinnamon	tbsp	0.25	
127	20:30	raisin	cup	0.125	
127	20:30	H2O	oz	12	
128	5:20	H2O	oz	2	7-May
128	10:10	bread (white)	oz	6	
128	10:10	cinnamon	tbsp	0.25	
128	10:10	sugar	tbsp	1	
128	10:10	margerine	tbsp	0.125	
128	10:10	H2O	oz	16	
128	11:25	salmon	oz	4	
128	11:25	lemon pepper	tbsp	0.125	
128	11:25	basil	tbsp	0.125	
128	13:25	bread (white)	oz	8	
128	13:25	pork	oz	8	
128	13:25	onion (white)	cup	0.125	
128	13:25	capsicum (red)	cup	0.125	
128	13:25	mushroom (white)	cup	0.125	
128	13:25	italian spices	tbsp	2	
128	13:25	tomato	cup	0.25	
128	13:25	H2O	oz	16	
128	20:25	bread (wheat)	oz	4	
128	20:25	steak	oz	3	
128	20:25	spinach	cup	0.25	
128	20:25	cucumber	cup	0.125	
128	20:25	capsicum (green)	cup	0.125	
128	20:25	dressing (honey mustard)	tbsp	0.5	
128	20:25	flour (white)	cup	0.5	

APPENDIX C (continued)

128	20:25	butter	tbsp	1	
128	20:25	sugar	tbsp	2	
129	4:20	H2O	oz	8	8-May
129	5:20	bread (sourdough)	oz	4	
129	5:20	sauce (Nutella)	tbsp	1	
129	8:00	H2O	oz	8	
129	10:00	bread (sourdough)	oz	4	
129	10:00	margerine	tbsp	0.5	
129	10:00	honey	tbsp	1	
129	13:15	bread (wheat)	oz	6	
129	13:15	capsicum (green)	cup	0.125	
129	13:15	spinach	cup	0.125	
129	13:15	steak	oz	4	
129	13:15	dressing (honey mustard)	tbsp	1	
129	13:15	H2O	oz	16	
129	14:45	H2O	oz	4	
129	16:30	strawberry	cup	1	
129	20:45	bagel (white)	oz	3	
129	20:45	cinnamon	tbsp	0.125	
129	20:45	raisin	cup	0.125	
129	21:15	beef	oz	3	
129	21:15	onion (white)	cup	0.125	
129	21:15	garlic	tbsp	0.5	
129	21:15	lettuce (romaine)	cup	0.5	
129	21:15	tomato	cup	0.125	
129	21:15	bread (pita)	oz	3	
129	21:15	eggplant	cup	0.25	
129	21:15	olive oil	tbsp	0.5	
129	21:15	juice (lemon)	tbsp	0.5	
130	7:20	H2O	oz	8	9-May
130	10:45	bagel (white)	oz	12	
130	10:45	cinnamon	tbsp	0.5	
130	10:45	brown sugar	tbsp	1	
130	10:45	margerine	tbsp	0.5	
130	11:45	beef	oz	3	
130	11:45	onion (white)	cup	0.125	
130	11:45	garlic	tbsp	1	
130	11:45	lettuce (romaine)	cup	0.125	
130	11:45	tomato	cup	0.05	
130	11:45	bread (pita)	oz	3	

APPENDIX C (continued)

130	11:45	eggplant	cup	0.25	
130	11:45	olive oil	tbsp	0.5	
130	11:45	juice (lemon)	tbsp	0.5	
130	11:45	H2O	oz	12	
130	14:45	H2O	oz	12	
130	20:10	ground turkey	oz	2	
130	20:10	bread (wheat)	oz	2	
130	20:10	lettuce (romaine)	cup	0.125	
130	20:10	bacon	oz	1	
130	20:10	mushroom (white)	cup	0.125	
130	20:10	mayonaise	tbsp	0.5	
130	20:10	ketchup	tbsp	0.5	
130	20:10	potato	cup	0.125	
130	20:10	canola oil	tbsp	0.5	
130	20:10	salt	tbsp	0.125	
131	10:00	bread (white)	oz	6	10-May
131	10:00	cinnamon	tbsp	0.125	
131	10:00	sugar	tbsp	0.5	
131	10:00	sauce (Nutella)	tbsp	1	
131	10:00	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
131	10:00	honey	tbsp	0.5	
131	11:45	ground turkey	oz	3	
131	11:45	bacon	oz	1	
131	11:45	lettuce (romaine)	cup	0.125	
131	11:45	bread (wheat)	oz	4	
131	11:45	mushroom (white)	cup	0.125	
131	11:45	potato	cup	0.125	
131	11:45	canola oil	tbsp	1	
131	11:45	salt	tsp	0.125	
131	11:45	mayonaise	tbsp	0.5	
131	11:45	ketchup	tbsp	0.25	
131	15:00	chip (corn tortilla)	cup	2	
131	15:00	tomato	cup	0.5	
131	15:00	onion (white)	cup	0.125	
131	15:00	jalapeno	cup	0.05	
131	17:30	pork	oz	3	
131	17:30	beef	oz	3	
131	17:30	corn meal	oz	6	
131	17:30	cumin	tbsp	1	

APPENDIX C (continued)

131	17:30	chili powder	tbsp	0.05	
131	23:30	couscous	cup	1	
131	23:30	lentil	cup	0.05	
131	23:30	tomato	cup	0.05	
131	23:30	garlic	tbsp	0.125	
131	23:30	olive oil	tbsp	0.25	
132	9:45	bread (white)	oz	6	11-May
132	9:45	margerine	tbsp	0.125	
132	9:45	sauce (Nutella)	tbsp	1	
132	9:45	jelly (raspberry)	tbsp	1	
132	12:30	pork	oz	3	
132	12:30	beef	oz	3	
132	12:30	corn meal	oz	6	
132	12:30	cumin	tbsp	1	
132	12:30	chili powder	tbsp	0.05	
132	14:10	strawberry	cup	0.25	
132	15:00	pork	oz	6	
132	15:00	bread (wheat)	oz	4	
132	15:00	honey dijon mustard	tbsp	1	
132	15:00	italian spices	tbsp	1	
132	15:00	H2O	oz	12	
132	21:30	pork	oz	6	
132	21:30	nectarine	cup	1	
132	21:30	onion (white)	cup	0.25	
132	21:30	salt	tsp	0.125	
132	21:30	pepper	tsp	0.25	
132	21:30	juice (lemon)	tbsp	0.5	
132	21:30	dijon mustard	tbsp	0.5	
132	21:30	coconut oil	tbsp	2	
132	23:30	hard cider	oz	20	
133	8:25	H2O	oz	2	
133	10:00	cake (white)	oz	6	
133	10:00	cinnamon	tbsp	1	
133	10:00	sugar	tbsp	2	
133	13:50	pork	oz	6	
133	13:50	bread (wheat)	oz	4	
133	13:50	honey dijon mustard	tbsp	1	
133	13:50	italian spices	tbsp	1	
133	14:30	candy (Jello)	cup	0.75	
133	14:30	rum	oz	3	

APPENDIX C (continued)

133	14:30	vodka	oz	3	
133	20:45	pork	oz	6	
133	20:45	cajun spices	tbsp	1	
133	20:45	chip (corn tortilla)	cup	1.5	
134	0:30	bread (white)	oz	6	13-May
134	0:30	sauce (Nutella)	tbsp	1	
134	0:30	jelly (raspberry)	tbsp	1	
134	0:30	preserves (apricot)	tbsp	1	
134	8:30	H2O	oz	4	
134	10:30	egg	oz	5	
134	10:30	pork	oz	4	
134	10:30	pear	cup	1	
134	10:30	salt	tsp	0.125	
134	10:30	pepper	tsp	0.125	
134	10:30	wine (red)	tbsp	2	
134	15:50	chip (corn tortilla)	cup	1	
134	15:50	tomato	cup	0.5	
134	15:50	onion (white)	cup	0.125	
134	15:50	jalapeno	cup	0.05	
134	15:50	pork	oz	6	
134	15:50	bread (wheat)	oz	4	
134	15:50	honey dijon mustard	tbsp	1	
134	15:50	italian spices	tbsp	1	
134	21:00	salmon	oz	8	
134	21:00	garlic	tbsp	1	
134	21:00	paprika	tbsp	0.125	
134	21:00	salt	tsp	0.125	
134	21:00	pepper	tsp	0.25	
134	21:00	asparagus	cup	0.75	
134	21:00	juice (lemon)	tbsp	0.5	
134	21:00	tomato	cup	0.5	
134	21:00	grape juice	oz	8	
135	1:30	nectarine	cup	0.75	14-May
135	10:15	salmon	oz	5	
135	10:15	garlic	tbsp	0.5	
135	10:15	paprika	tbsp	0.05	
135	10:15	salt	tsp	0.125	
135	10:15	pepper	tsp	0.125	
135	10:15	juice (lemon)	tbsp	0.25	
135	11:45	pork	oz	6	

APPENDIX C (continued)

135	11:45	bread (wheat)	oz	4	
135	11:45	honey dijon mustard	tbsp	1	
135	11:45	H2O	oz	4	
135	11:45	grape juice	oz	4	
135	11:45	italian spices	tbsp	1	
135	13:00	pork	oz	6	
135	13:00	italian spices	tbsp	1	
135	13:00	honey dijon mustard	tbsp	0.5	
135	17:30	cake (white)	oz	6	
135	17:30	cinnamon	tbsp	1	
135	17:30	sugar	tbsp	2	
135	21:00	steak	oz	8	
135	21:00	mushroom (portabella)	cup	0.75	
135	21:00	onion (green)	cup	0.125	
135	21:00	wine (red)	cup	0.25	
135	21:00	olive oil	tbsp	2	
135	21:00	salt	tsp	0.125	
135	21:00	pepper	tsp	0.125	
135	21:00	sweet potato	cup	1	
135	21:00	juice (lime)	tbsp	0.5	
135	21:00	coconut milk	cup	0.125	
135	21:00	thyme	tbsp	0.5	
135	21:00	couscous	cup	0.5	
135	21:00	lentil	cup	0.05	
135	21:00	tomato	cup	0.05	
135	21:00	H2O	oz	8	
135	21:00	grape juice	oz	8	
136	6:15	banana	cup	0.5	15-May
136	10:15	salmon	oz	8	
136	10:15	salt	tsp	0.125	
136	10:15	pepper	tsp	0.125	
136	10:15	paprika	tbsp	0.05	
136	10:15	juice (lemon)	tbsp	0.5	
136	10:15	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
136	10:15	garlic	tbsp	0.5	
136	10:15	honey	tbsp	0.5	
136	11:00	banana	cup	0.125	
136	11:00	pineapple	cup	0.25	
136	11:00	mango	cup	0.25	

APPENDIX C (continued)

136	11:00	strawberry	cup	0.25	
136	11:00	blackberry	cup	0.125	
136	11:00	kiwi	cup	0.125	
136	12:45	bread (white)	oz	8	
136	12:45	pork	oz	3	
136	12:45	italian spices	tbsp	0.5	
136	12:45	cheese (provalone)	oz	6	
136	12:45	tomato	cup	0.25	
136	12:45	capsicum (green)	cup	0.125	
136	12:45	onion (white)	cup	0.125	
136	12:45	grape juice	oz	6	
136	12:45	H2O	oz	6	
136	15:40	chicken	oz	4	
136	15:40	sauce (BBQ)	tbsp	2	
136	15:40	grape juice	oz	6	
136	15:40	H2O	oz	6	
136	19:00	banana	cup	0.125	
136	19:00	pineapple	cup	0.25	
136	19:00	mango	cup	0.25	
136	19:00	strawberry	cup	0.25	
136	19:00	blackberry	cup	0.125	
136	19:00	kiwi	cup	0.125	
136	20:00	chicken broth	cup	1.5	
136	20:00	egg	oz	3	
136	20:00	chive	tbsp	1	
136	20:00	ginger	tbsp	0.125	
137	10:20	pork	oz	6	16-May
137	10:20	nectarine	cup	0.5	
137	10:20	mint	tbsp	0.5	
137	10:20	onion (white)	cup	0.125	
137	10:20	juice (lemon)	tbsp	1	
137	10:20	salt	tsp	0.125	
137	10:20	dijon mustard	tbsp	0.5	
137	10:20	H2O	oz	12	
137	12:30	candy (Jolly Rancher)	pc	2	
137	15:30	chicken broth	cup	1.5	
137	15:30	egg	oz	3	
137	15:30	chive	tbsp	1	
137	15:30	ginger	tbsp	0.125	
137	15:30	H2O	oz	6	

APPENDIX C (continued)

137	15:30	grape juice	oz	6	
137	16:30	pork	oz	6	
137	16:30	artichoke	cup	0.25	
137	16:30	tomato	cup	0.25	
137	16:30	garlic	tbsp	0.125	
137	16:30	olive oil	tbsp	1.5	
137	19:30	banana	cup	0.125	
137	19:30	pineapple	cup	0.25	
137	19:30	mango	cup	0.25	
137	19:30	strawberry	cup	0.25	
137	19:30	blackberry	cup	0.125	
137	19:30	kiwi	cup	0.125	
137	21:45	turkey	oz	4	
137	21:45	tomato	cup	0.25	
137	21:45	lettuce (romaine)	cup	0.25	
137	21:45	bread (white)	oz	6	
137	21:45	cheese (parmasean)	tbsp	1	
138	10:45	pork	oz	6	17-May
138	10:45	artichoke	cup	0.25	
138	10:45	tomato	cup	0.25	
138	10:45	garlic	tbsp	0.125	
138	10:45	olive oil	tbsp	1.5	
138	10:45	banana	cup	0.125	
138	10:45	pineapple	cup	0.25	
138	10:45	mango	cup	0.25	
138	10:45	strawberry	cup	0.25	
138	10:45	blackberry	cup	0.125	
138	10:45	kiwi	cup	0.125	
138	10:45	H2O	oz	24	
138	12:50	H2O	oz	16	
138	15:00	cake (white)	oz	3	
138	15:00	juice (lemon)	tbsp	1	
138	15:00	poppy seed	tbsp	0.5	
138	15:00	sugar	tbsp	3	
138	15:00	H2O	oz	16	
138	18:00	pork	oz	6	
138	18:00	artichoke	cup	0.25	
138	18:00	tomato	cup	0.25	
138	18:00	garlic	tbsp	0.125	
138	18:00	olive oil	tbsp	1.5	

APPENDIX C (continued)

138	23:15	chicken	oz	4	
138	23:15	sauce (BBQ)	tbsp	2	
138	23:15	avocado	cup	0.25	
138	23:15	milk chocolate	pc	1	
138	23:15	almond butter	tbsp	0.5	
138	23:15	honey	tbsp	0.5	
138	23:15	cinnamon	tbsp	0.125	
138	23:15	nutmeg	tbsp	0.125	
138	23:15	H2O	oz	8	
139	10:25	banana	cup	1	18-May
139	10:25	H2O	oz	6	
139	12:00	pork	oz	8	
139	12:00	bread (white)	oz	6	
139	12:00	honey dijon mustard	tbsp	1	
139	12:00	italian spices	tbsp	1	
139	16:00	cake (white)	oz	2	
139	16:00	cake (chocolate)	oz	2	
139	16:00	milk chocolate	pc	1	
139	16:00	juice (lemon)	tbsp	0.25	
139	16:00	sugar	tbsp	2	
139	19:30	cake (white)	oz	2	
139	19:30	cake (chocolate)	oz	2	
139	19:30	milk chocolate	pc	1	
139	19:30	juice (lemon)	tbsp	0.25	
139	19:30	sugar	tbsp	2	
139	20:00	squash (acorn)	cup	1	
139	20:00	apple	cup	0.75	
139	20:00	brown sugar	tbsp	3	
139	20:00	coconut oil	tbsp	1	
139	20:00	olive oil	tbsp	2	
139	20:00	steak	oz	5	
139	20:00	garlic pepper	tbsp	1	
139	20:00	honey dijon mustard	tbsp	0.5	
139	20:00	wine (red)	tbsp	1	
139	20:00	salt	tsp	0.125	
139	20:00	cranberry juice (lime)	oz	8	
139	20:00	H2O	oz	8	
139	22:00	cake (white)	oz	2	
139	22:00	cake (chocolate)	oz	2	
139	22:00	milk chocolate	pc	1	

APPENDIX C (continued)

139	22:00	juice (lemon)	tbsp	0.25	
139	22:00	sugar	tbsp	2	
140	10:45	pork	oz	6	19-May
140	10:45	artichoke	cup	0.25	
140	10:45	tomato	cup	0.25	
140	10:45	garlic	tbsp	0.125	
140	10:45	olive oil	tbsp	1.5	
140	10:45	cranberry juice (lime)	oz	8	
140	14:30	H2O	oz	8	
140	14:30	pork	oz	8	
140	14:30	bread (white)	oz	6	
140	14:30	honey dijon mustard	tbsp	1	
140	14:30	italian spices	tbsp	1	
140	21:00	tuna	oz	4	
140	21:00	garlic	tbsp	1	
140	21:00	salt	tsp	0.125	
140	21:00	juice (lime)	tbsp	1	
140	21:00	coconut oil	tbsp	1	
140	21:00	squash (zucchini)	cup	0.25	
140	21:00	tomato	cup	0.5	
140	21:00	onion (white)	cup	0.5	
140	21:00	capsicum (green)	cup	0.25	
140	21:00	capsicum (red)	cup	0.25	
140	21:00	mushroom (white)	cup	0.5	
140	21:00	cranberry juice (lime)	oz	8	
140	21:00	H2O	oz	8	
140	21:00	hard cider	oz	22	
141	5:00	cake (white)	oz	2	20-May
141	5:00	cake (chocolate)	oz	2	
141	5:00	milk chocolate	pc	1	
141	5:00	juice (lemon)	tbsp	0.25	
141	5:00	sugar	tbsp	2	
141	15:00	pork	oz	4	
141	15:00	italian spices	tbsp	0.5	
141	15:00	bread (white)	oz	6	
141	15:00	honey dijon mustard	tbsp	1	
141	15:00	pineapple	cup	0.5	
141	15:00	mango	cup	0.25	
141	15:00	blueberry	cup	0.125	
141	15:00	strawberry	cup	0.25	

APPENDIX C (continued)

141	15:00	kiwi	cup	0.125	
141	15:00	banana	cup	0.125	
141	15:00	scalop	oz	3	
141	15:00	scalion	cup	0.125	
141	15:00	tomato	cup	0.125	
141	15:00	coconut milk	cup	0.125	
141	15:00	coconut oil	tbsp	0.5	
141	15:00	garlic	tbsp	0.5	
141	15:00	oregano	tbsp	0.125	
141	15:00	H2O	oz	12	
141	17:30	cranberry juice	oz	6	
141	17:30	H2O	oz	6	
141	20:30	hamburger	oz	5	
141	20:30	capsicum (red)	cup	0.125	
141	20:30	bread (wheat)	oz	6	
141	20:30	lettuce (romaine)	cup	0.25	
141	20:30	bacon	oz	2	
141	20:30	mushroom (white)	cup	0.125	
141	20:30	sauce (BBQ)	tbsp	2	
141	23:45	H2O	oz	12	
142	10:55	pork	oz	6	21-May
142	10:55	artichoke	cup	0.25	
142	10:55	tomato	cup	0.25	
142	10:55	garlic	tbsp	0.125	
142	10:55	olive oil	tbsp	1.5	
142	10:55	cranberry juice (lime)	oz	6	
142	10:55	H2O	oz	10	
142	14:00	pineapple	cup	0.5	
142	14:00	mango	cup	0.25	
142	14:00	blueberry	cup	0.125	
142	14:00	strawberry	cup	0.25	
142	14:00	kiwi	cup	0.125	
142	14:00	banana	cup	0.125	
142	16:30	pork	oz	4	
142	16:30	italian spices	tbsp	1	
142	21:00	chicken	oz	10	
142	21:00	potato	cup	0.5	
142	21:00	sauce (BBQ)	tbsp	2	
142	21:00	ketchup	tbsp	0.5	
142	21:00	H2O	oz	16	

APPENDIX C (continued)

143	10:00	H2O	oz	16	22-May
143	10:00	hamburger	oz	5	
143	10:00	bread (wheat)	oz	5	
143	10:00	sauce (BBQ)	tbsp	1	
143	10:00	lettuce (romaine)	cup	0.25	
143	10:00	tomato	cup	0.25	
143	14:10	pineapple	cup	0.5	
143	14:10	mango	cup	0.25	
143	14:10	blueberry	cup	0.125	
143	14:10	strawberry	cup	0.25	
143	14:10	kiwi	cup	0.125	
143	14:10	banana	cup	0.125	
143	21:50	scalop	oz	3	
143	21:50	scalion	cup	0.125	
143	21:50	tomato	cup	0.125	
143	21:50	coconut milk	cup	0.125	
143	21:50	coconut oil	tbsp	0.5	
143	21:50	garlic	tbsp	0.5	
143	21:50	oregano	tbsp	0.125	
143	21:50	cranberry juice (lime)	oz	4	
143	21:50	H2O	oz	4	
144	10:00	hamburger	oz	5	23-May
144	10:00	bread (wheat)	oz	5	
144	10:00	sauce (BBQ)	tbsp	1	
144	10:00	lettuce (romaine)	cup	0.25	
144	10:00	tomato	cup	0.25	
144	10:00	cranberry juice (lime)	oz	12	
144	10:00	H2O	oz	12	
144	15:40	pork	oz	4	
144	15:40	italian spices	tbsp	0.5	
144	15:40	H2O	oz	16	
144	21:30	squash (spaghetti)	cup	4	
144	21:30	tomato	cup	1.5	
144	21:30	italian spices	tbsp	1	
144	21:30	onion (white)	cup	0.25	
144	21:30	garlic	tbsp	1	
144	21:30	capsicum (red)	cup	0.25	
144	21:30	capsicum (green)	cup	0.25	
144	21:30	H2O	oz	8	
145	1:45	H2O	oz	4	24-May

APPENDIX C (continued)

145	8:25	banana	cup	0.75	
145	10:00	pork	oz	6	
145	10:00	artichoke	cup	0.25	
145	10:00	tomato	cup	0.25	
145	10:00	garlic	tbsp	0.125	
145	10:00	olive oil	tbsp	1.5	
145	10:00	H2O	oz	16	
145	12:20	juice (apple)	pc	1	
145	12:20	juice (carrot)	pc	3	
145	12:20	juice (kale)	pc	2	
145	12:20	juice (strawberry)	pc	5	
145	14:30	pork	oz	8	
145	14:30	italian spices	tbsp	1	
145	14:30	bread (white)	oz	6	
145	14:30	honey dijon mustard	tbsp	2	
145	14:30	H2O	oz	12	
145	19:30	H2O	oz	16	
145	22:45	bread (pita)	oz	4	
145	22:45	chickpea	cup	0.25	
145	22:45	garlic	tbsp	1	
145	22:45	olive oil	tbsp	3	
145	22:45	cumin	tbsp	0.5	
145	22:45	H2O	oz	16	
146	1:15	H2O	oz	4	25-May
146	11:00	banana	cup	0.75	
146	11:00	H2O	oz	16	
146	17:20	pork	oz	6	
146	17:20	bread (white)	oz	6	
146	17:20	italian spices	tbsp	1	
146	17:20	honey dijon mustard	tbsp	1	
146	23:30	cake (white)	oz	6	
146	23:30	sugar	tbsp	3	
146	23:30	blueberry	cup	0.125	
146	23:30	H2O	oz	16	
147	2:20	juice (apple)	pc	2	26-May
147	2:20	juice (carrot)	pc	4	
147	12:30	sweet potato	cup	2	
147	12:30	juice (lime)	tbsp	0.25	
147	12:30	coconut milk	cup	0.25	
147	12:30	thyme	tbsp	1	

APPENDIX C (continued)

147	12:30	H2O	oz	16	
147	13:45	bread (white)	oz	8	
147	13:45	turkey	oz	6	
147	13:45	mayonaise	tbsp	2	
147	13:45	sauce (cranberry)	tbsp	4	
147	13:45	chip (corn tortilla)	cup	2	
147	13:45	tomato	cup	0.125	
147	13:45	onion (white)	tbsp	2	
147	13:45	cilantro	tbsp	0.5	
147	13:45	H2O	oz	16	
147	13:50	H2O	oz	16	
147	18:20	H2O	oz	16	
147	21:30	eggplant	cup	0.5	
147	21:30	garlic	tbsp	0.25	
147	21:30	olive oil	tbsp	1	
147	21:30	steak	oz	8	
147	21:30	onion (white)	cup	0.25	
147	21:30	capsicum (green)	cup	0.5	
147	21:30	capsicum (red)	cup	0.25	
147	21:30	mushroom (white)	cup	0.25	
147	21:30	tomato	cup	0.25	
147	21:30	lettuce (romaine)	cup	0.25	
147	21:30	bread (pita)	oz	4	
147	21:30	H2O	oz	16	
147	21:30	candy (Starburst)	pc	8	
147	23:30	H2O	oz	16	
148	3:00	cake (white)	oz	3	27-May
148	3:00	sugar	tbsp	1.5	
148	3:00	blueberry	cup	0.125	
148	11:30	tea (Lipton Green Tea Honey / Lemon - decaf)	oz	12	
148	11:30	honey	tbsp	1	
148	12:30	bagel (white)	oz	6	
148	12:30	blueberry	cup	0.125	
148	12:30	bacon	oz	3	
148	12:30	margerine	tbsp	0.5	
148	12:30	H2O	oz	12	
148	14:30	salmon	oz	4	
148	14:30	potato	cup	0.125	
148	14:30	black olive	cup	0.125	

APPENDIX C (continued)

148	14:30	shallot	cup	0.05	
148	14:30	garlic	tbsp	0.5	
148	14:30	juice (lemon)	tbsp	0.5	
148	14:30	olive oil	tbsp	1	
148	14:30	basil	tbsp	0.25	
148	14:30	tomato	cup	0.125	
148	14:30	cake (white)	oz	3	
148	14:30	sugar	tbsp	1.5	
148	14:30	blueberry	cup	0.125	
148	21:30	bread (wheat)	oz	4	
148	21:30	hamburger	oz	4	
148	21:30	tomato	cup	0.25	
148	21:30	lettuce (romaine)	cup	0.125	
148	21:30	candy (Jello)	cup	0.5	
148	21:30	sauce (BBQ)	tbsp	1	
148	21:30	H2O	oz	12	
149	2:05	candy (Starburst)	pc	6	28-May
149	11:45	cake (white)	oz	3	
149	11:45	sugar	tbsp	1.5	
149	11:45	blueberry	cup	0.125	
149	11:45	H2O	oz	20	
149	13:30	bread (wheat)	oz	4	
149	13:30	hamburger	oz	4	
149	13:30	tomato	cup	0.25	
149	13:30	lettuce (romaine)	cup	0.125	
149	13:30	bacon	oz	2	
149	18:00	salmon	oz	4	
149	18:00	potato	cup	0.125	
149	18:00	black olive	cup	0.125	
149	18:00	shallot	cup	0.05	
149	18:00	garlic	tbsp	0.5	
149	18:00	juice (lemon)	tbsp	0.5	
149	18:00	olive oil	tbsp	1	
149	18:00	basil	tbsp	0.25	
149	18:00	tomato	cup	0.125	
149	18:00	eggplant	cup	0.5	
149	18:00	garlic	tbsp	0.25	
149	18:00	olive oil	tbsp	1	
149	18:00	bread (pita)	oz	4	
149	18:00	H2O	oz	16	

APPENDIX C (continued)

149	22:30	pasta (wheat)	cup	2	
149	22:30	margerine	tbsp	1.5	
149	22:30	basil	tbsp	0.125	
149	22:30	oregano	tbsp	0.125	
149	22:30	salt	tsp	0.125	
149	22:30	pepper	tsp	0.125	
149	22:30	H2O	oz	6	
150	1:45	H2O	oz	4	29-May
150	10:30	sweet potato	cup	1.5	
150	10:30	juice (lime)	tbsp	0.75	
150	10:30	coconut milk	cup	0.25	
150	10:30	thyme	tbsp	0.75	
150	10:30	H2O	oz	4	
150	14:20	bread (wheat)	oz	6	
150	14:20	oats	cup	0.125	
150	14:20	honey	tbsp	0.125	
150	14:20	salami	oz	3	
150	14:20	pepperoni	oz	3	
150	14:20	lettuce (romaine)	cup	0.25	
150	14:20	tomato	cup	0.25	
150	14:20	black olive	cup	0.25	
150	14:20	pickle (dill)	cup	0.125	
150	14:20	oregano	tbsp	0.5	
150	14:20	chip (Sun Chips original)	cup	1.5	
150	14:20	flour (white)	cup	0.25	
150	14:20	sugar	tbsp	2	
150	14:20	butter	tbsp	1	
150	14:20	soda (Barq's root beer)	oz	20	
150	15:10	H2O	oz	16	
150	23:30	bread (wheat)	oz	6	
150	23:30	oats	cup	0.125	
150	23:30	honey	tbsp	0.125	
150	23:30	salami	oz	3	
150	23:30	pepperoni	oz	3	
150	23:30	lettuce (romaine)	cup	0.25	
150	23:30	tomato	cup	0.25	
150	23:30	black olive	cup	0.25	
150	23:30	pickle (dill)	cup	0.125	
150	23:30	oregano	tbsp	0.5	
151	9:40	H2O	oz	10	30-May

APPENDIX C (continued)

151	10:20	H2O	oz	16	
151	11:45	pork	oz	5	
151	11:45	italian spices	tbsp	1	
151	11:45	bread (white)	oz	5	
151	11:45	honey dijon mustard	tbsp	1	
151	11:45	candy (Jello)	cup	0.75	
151	11:45	H2O	oz	16	
151	20:00	asparagus	cup	1.5	
151	20:00	mushroom (white)	cup	0.5	
151	20:00	margerine	tbsp	0.5	
151	20:00	garlic pepper	tbsp	0.5	
151	20:00	juice (lemon)	tbsp	1	
151	20:00	vinegar (balsamic)	tbsp	0.5	
151	20:00	oregano	tbsp	0.25	
151	20:00	tilapia	oz	6	
151	20:00	onion (white)	cup	0.75	
151	20:00	capsicum (red)	cup	0.5	
151	20:00	egg	oz	1	
151	20:00	mayonaise	tbsp	1	
151	20:00	dijon mustard	tbsp	1	
151	20:00	soda (lemon seltzer)	oz	4	
152	10:00	cereal (Oat Oh's)	cup	1.5	31-May
152	10:00	H2O	oz	16	
152	12:45	pork	oz	5	
152	12:45	italian spices	tbsp	1	
152	12:45	bread (white)	oz	6	
152	12:45	honey dijon mustard	tbsp	1	
152	12:45	candy (Jello)	cup	0.75	
152	12:45	H2O	oz	16	
152	17:20	H2O	oz	16	
152	19:50	tortilla (corn)	oz	4	
152	19:50	tortilla (flour)	oz	3	
152	19:50	kidney bean	cup	0.25	
152	19:50	lettuce (romaine)	cup	1	
152	19:50	tomato	cup	0.125	
152	19:50	steak	oz	3	
152	19:50	ground beef	oz	3	
152	19:50	cumin	tbsp	0.125	
152	19:50	coriander	tbsp	0.125	
152	19:50	H2O	oz	16	

APPENDIX C (continued)

152	22:00	H2O	oz	4	
153	10:30	cereal (Peanutbutter Puffs)	cup	1.5	1-Jun
153	10:30	H2O	oz	16	
153	15:45	H2O	oz	16	
153	16:30	cereal (Peanutbutter Puffs)	cup	1.5	
153	21:00	eggplant	cup	0.5	
153	21:00	garlic	tbsp	0.25	
153	21:00	olive oil	tbsp	1	
153	21:00	steak	oz	8	
153	21:00	onion (white)	cup	0.25	
153	21:00	capsicum (green)	cup	0.5	
153	21:00	capsicum (red)	cup	0.25	
153	21:00	mushroom (white)	cup	0.25	
153	21:00	bread (pita)	oz	4	
153	21:00	H2O	oz	16	
154	11:30	cereal (Peanutbutter Puffs)	cup	1.5	2-Jun
154	11:30	H2O	oz	16	
154	13:25	pork	oz	5	
154	13:25	italian spices	tbsp	1	
154	13:25	bread (white)	oz	6	
154	13:25	honey dijon mustard	tbsp	1	
154	13:25	H2O	oz	16	
154	18:30	shrimp	oz	3	
154	18:30	asparagus	cup	0.25	
154	18:30	squash (zucchini)	cup	0.25	
154	18:30	pasta (wheat)	cup	0.75	
154	18:30	garlic	tbsp	0.5	
154	18:30	oregano	tbsp	0.25	
154	18:30	butter	tbsp	0.5	
154	18:30	salt	tsp	0.125	
154	18:30	pepper	tsp	0.25	
154	22:00	turkey	oz	4	
154	22:00	potato	cup	0.75	
154	22:00	green bean	cup	0.75	
154	22:00	carrot	cup	0.25	
154	22:00	bread (wheat)	oz	3	
154	22:00	celery	cup	0.125	
154	22:00	italian spices	tbsp	2	
154	22:00	H2O	oz	16	
154	22:00	oats	cup	0.25	

APPENDIX C (continued)

154	22:00	blueberry	cup	0.125	
155	11:10	salmon	oz	4	3-Jun
155	11:10	potato	cup	0.125	
155	11:10	black olive	cup	0.125	
155	11:10	shallot	cup	0.05	
155	11:10	garlic	tbsp	0.5	
155	11:10	juice (lemon)	tbsp	0.5	
155	11:10	olive oil	tbsp	1	
155	11:10	basil	tbsp	0.25	
155	17:40	cereal (Oat Oh's)	cup	1.5	
155	20:30	hamburger	oz	6	
155	20:30	bread (wheat)	oz	4	
155	20:30	sauce (BBQ)	tbsp	1	
155	20:30	lettuce (romaine)	cup	0.125	
155	20:30	bacon	oz	2	
155	20:30	avocado	cup	0.25	
155	20:30	H2O	oz	12	
155	23:45	chicken	oz	3	
155	23:45	potato	cup	0.25	
155	23:45	asparagus	cup	0.125	
155	23:45	vinegar (balsamic)	tbsp	3	
155	23:45	H2O	oz	24	
156	10:30	cereal (Oat Oh's)	cup	1.5	4-Jun
156	10:30	H2O	oz	16	
156	12:20	hamburger	oz	6	
156	12:20	bread (wheat)	oz	4	
156	12:20	lettuce (romaine)	cup	0.25	
156	12:20	bacon	oz	2	
156	12:20	sauce (BBQ)	tbsp	1	
156	16:30	cereal (Oat Oh's)	cup	1.5	
156	16:30	H2O	oz	16	
156	20:00	beef	oz	10	
156	20:00	onion (white)	cup	0.25	
156	20:00	cactus	cup	0.25	
156	20:00	tortilla (flour)	oz	10	
156	20:00	cilantro	tbsp	1	
156	20:00	olive oil	tbsp	4	
156	20:00	chip (corn tortilla)	cup	0.125	
156	20:00	jalapeno	tbsp	1	
156	20:00	H2O	oz	16	

APPENDIX C (continued)

157	10:45	cereal (Oat Oh's)	cup	1.5	5-Jun
157	13:10	shrimp	oz	3	
157	13:10	asparagus	cup	0.25	
157	13:10	squash (zucchini)	cup	0.25	
157	13:10	pasta (wheat)	cup	0.75	
157	13:10	garlic	tbsp	0.5	
157	13:10	oregano	tbsp	0.25	
157	13:10	butter	tbsp	0.5	
157	13:10	salt	tsp	0.125	
157	13:10	pepper	tsp	0.25	
157	13:10	H2O	oz	12	
157	15:10	cereal (Oat Oh's)	cup	1.5	
157	15:45	hamburger	oz	6	
157	15:45	bread (wheat)	oz	4	
157	15:45	bacon	oz	3	
157	15:45	sauce (BBQ)	tbsp	2	
157	15:45	H2O	oz	12	
157	20:50	steak	oz	4	
157	20:50	mushroom (white)	cup	0.125	
157	20:50	capsicum (red)	cup	0.125	
157	20:50	onion (white)	cup	0.125	
157	20:50	bread (white)	oz	5	
157	20:50	butter	tbsp	1	
157	20:50	garlic	tbsp	0.25	
157	20:50	H2O	oz	12	
158	5:15	banana	cup	1	6-Jun
158	10:20	bread (white)	oz	6	
158	10:20	strawberry	cup	0.125	
158	10:20	banana	cup	0.125	
158	12:30	bread (white)	oz	6	
158	12:30	pork	oz	5	
158	12:30	italian spices	tbsp	1	
158	12:30	honey dijon mustard	tbsp	1	
158	13:45	H2O	oz	12	
158	16:20	cereal (Oat Oh's)	cup	1.5	
158	21:50	tuna	oz	5	
158	21:50	vinegar (red wine)	tbsp	2	
158	21:50	turmeric	tbsp	0.5	
158	21:50	salt	tsp	0.125	
158	21:50	brussel sprout	cup	0.75	

APPENDIX C (continued)

158	21:50	juice (lemon)	tbsp	0.5	
158	21:50	wine (white)	tbsp	0.5	
158	21:50	olive oil	tbsp	1	
158	21:50	pepper	tsp	0.25	
158	21:50	couscous	cup	1	
158	21:50	italian spices	tbsp	1	
158	21:50	H2O	oz	12	
159	10:10	cereal (Oat Oh's)	cup	3	7-Jun
159	10:10	H2O	oz	16	
159	13:10	strawberry	cup	1	
159	13:10	peach	cup	0.25	
159	13:10	mango	cup	0.25	
159	13:10	blueberry	cup	0.25	
159	15:30	H2O	oz	16	
159	16:10	chicken	oz	3	
159	16:10	potato	cup	0.25	
159	16:10	asparagus	cup	0.125	
159	16:10	vinegar (balsamic)	tbsp	3	
159	16:10	tomato	cup	0.125	
159	19:30	bread (white)	oz	8	
159	19:30	roast beef	oz	6	
159	19:30	onion (white)	cup	0.125	
159	19:30	au jus	oz	3	
159	19:30	H2O	oz	8	
159	19:30	potato	cup	0.25	
159	19:30	sauce (BBQ)	tbsp	1	
159	19:30	canola oil	tbsp	1	
159	23:55	H2O	oz	8	
159	23:55	bread (white)	oz	3	
159	23:55	banana	cup	0.125	
159	23:55	strawberry	cup	0.125	
160	2:15	H2O	oz	2	8-Jun
160	10:10	bread (white)	oz	6	
160	10:10	strawberry	cup	0.125	
160	10:10	banana	cup	0.125	
160	10:10	H2O	oz	16	
160	10:10	margerine	tbsp	1	
160	14:20	shrimp	oz	3	
160	14:20	asparagus	cup	0.25	
160	14:20	squash (zucchini)	cup	0.25	

APPENDIX C (continued)

160	14:20	pasta (wheat)	cup	0.75	
160	14:20	garlic	tbsp	0.5	
160	14:20	oregano	tbsp	0.25	
160	14:20	butter	tbsp	0.5	
160	14:20	salt	tsp	0.125	
160	14:20	pepper	tsp	0.25	
160	19:00	bread (white)	oz	3	
160	19:00	butter	tbsp	0.5	
160	19:00	garlic	tbsp	0.5	
160	19:00	H2O	oz	16	
160	21:45	ground beef	oz	5	
160	21:45	tomato	cup	0.125	
160	21:45	onion (white)	cup	0.05	
160	21:45	jalapeno	tbsp	0.25	
160	21:45	lettuce (romaine)	cup	0.25	
160	21:45	bacon	oz	2	
160	21:45	sauce (BBQ)	tbsp	1	
160	21:45	bread (wheat)	oz	4	
160	21:45	H2O	oz	16	
161	5:30	bread (white)	oz	2	9-Jun
161	5:30	banana	cup	0.125	
161	5:30	strawberry	cup	0.125	
161	12:00	H2O	oz	4	
161	12:20	steak	oz	4	
161	12:20	mushroom (white)	cup	0.125	
161	12:20	capsicum (red)	cup	0.125	
161	12:20	onion (white)	cup	0.125	
161	12:20	bread (white)	oz	3	
161	14:30	pork	oz	5	
161	14:30	italian spices	tbsp	1	
161	14:30	bread (white)	oz	6	
161	14:30	honey dijon mustard	tbsp	1	
161	14:30	H2O	oz	16	
161	21:40	cheese (mozzarella)	oz	8	
161	21:40	tomato	cup	0.5	
161	21:40	bread (wheat)	oz	8	
161	21:40	pork	oz	3	
161	21:40	italian spices	tbsp	1	
161	21:40	beer	oz	4	
162	0:20	H2O	oz	4	10-Jun

APPENDIX C (continued)

162	11:00	H2O	oz	12	
162	11:00	bread (white)	oz	3	
162	11:00	strawberry	cup	0.05	
162	11:00	banana	cup	0.05	
162	11:00	sugar	tbsp	1.5	
162	13:30	cheese (mozzarella)	oz	8	
162	13:30	tomato	cup	0.5	
162	13:30	bread (wheat)	oz	8	
162	13:30	pork	oz	3	
162	13:30	italian spices	tbsp	1	
162	13:30	H2O	oz	16	
162	21:00	turkey	oz	3	
162	21:00	lettuce (romaine)	cup	0.125	
162	21:00	sauce (cranberry)	tbsp	2	
162	21:00	mayonaise	tbsp	1	
162	21:00	chip (corn tortilla)	cup	1	
162	21:00	tomato	cup	0.125	
162	21:00	onion (white)	cup	0.05	
162	21:00	jalapeno	tbsp	0.05	
162	21:00	soda (Sprite)	oz	12	
162	22:30	H2O	oz	12	
163	10:15	bread (white)	oz	3	11-Jun
163	10:15	strawberry	cup	0.05	
163	10:15	banana	cup	0.05	
163	10:15	sugar	tbsp	2	
163	10:15	H2O	oz	16	
163	10:45	cake (white)	oz	2	
163	10:45	blueberry	cup	0.75	
163	10:45	H2O	oz	16	
163	12:10	pinto bean	cup	0.5	
163	12:10	tortilla (flour)	oz	4	
163	12:10	tortilla (corn)	oz	2	
163	12:10	ground beef	oz	8	
163	12:10	lettuce (romaine)	cup	0.5	
163	12:10	cheese (cheddar)	oz	2	
163	12:10	mexican spices	tbsp	1	
163	21:40	pork	oz	4	
163	21:40	leek	cup	0.75	
163	21:40	capsicum (red)	cup	0.25	
163	21:40	turmeric	tbsp	0.5	

APPENDIX C (continued)

163	21:40	garlic pepper	tbsp	0.5	
163	22:45	candy (Mike + Ike)	pc	15	
163	22:45	H2O	oz	16	
164	8:30	bread (white)	oz	3	12-Jun
164	8:30	strawberry	cup	0.05	
164	8:30	banana	cup	0.05	
164	8:30	sugar	tbsp	2	
164	8:30	H2O	oz	12	
164	10:10	cake (white)	oz	4	
164	10:10	cranberry	cup	0.125	
164	10:10	H2O	oz	12	
164	12:30	bread (white)	oz	3	
164	12:30	pork	oz	6	
164	12:30	italian spices	tbsp	1	
164	12:30	chili powder	tbsp	0.25	
164	12:30	H2O	oz	12	
164	13:00	H2O	oz	12	
164	19:00	turkey	oz	3	
164	19:00	bread (wheat)	oz	4	
164	19:00	italian spices	tbsp	1	
164	19:00	corn starch	tbsp	0.5	
165	0:30	cereal (Chocolate Squares)	cup	1.5	13-Jun
165	10:20	turkey	oz	3	
165	10:20	sauce (cranberry)	cup	0.125	
165	10:20	lettuce (romaine)	cup	0.125	
165	10:20	mayonaise	tbsp	2	
165	10:20	bread (white)	cup	1.5	
165	10:20	chip (corn tortilla)	cup	0.5	
165	10:20	tomato	cup	0.1	
165	10:20	onion (white)	cup	0.1	
165	10:20	cilantro	tbsp	0.5	
165	10:20	H2O	oz	12	
165	12:30	cereal (Chocolate Squares)	cup	1.5	
165	12:30	H2O	oz	12	
165	16:20	broccoli	cup	0.25	
165	16:20	potato	cup	0.5	
165	16:20	steak	oz	3	
165	16:20	corn starch	tbsp	0.5	
165	16:20	H2O	oz	12	
165	21:00	bread (wheat)	oz	6	

APPENDIX C (continued)

165	21:00	hamburger	oz	6	
165	21:00	tomato	cup	0.1	
165	21:00	onion (white)	cup	0.1	
165	21:00	cilantro	tbsp	0.5	
165	21:00	H2O	oz	12	
166	1:20	H2O	oz	6	14-Jun
166	10:20	bread (white)	oz	6	
166	10:20	strawberry	cup	0.05	
166	10:20	banana	cup	0.05	
166	10:20	sugar	tbsp	1	
166	12:20	bread (white)	oz	6	
166	12:20	pork	oz	6	
166	12:20	italian spices	tbsp	1	
166	12:20	chili powder	tbsp	0.25	
166	12:20	tomato	cup	0.05	
166	12:20	onion (white)	cup	0.05	
166	12:20	jalapeno	tbsp	0.25	
166	12:20	cilantro	tbsp	0.5	
166	12:20	H2O	oz	12	
166	19:30	shrimp	oz	6	
166	19:30	asparagus	cup	0.5	
166	19:30	pasta (wheat)	cup	3	
166	19:30	tomato	cup	0.25	
166	19:30	cream	oz	4	
166	19:30	garlic	tbsp	0.25	
166	19:30	H2O	oz	16	
167	12:45	hamburger	oz	5	15-Jun
167	12:45	bread (wheat)	oz	6	
167	12:45	avocado	cup	0.25	
167	12:45	bacon	oz	1.5	
167	12:45	tomato	cup	0.125	
167	12:45	onion (white)	tbsp	2	
167	12:45	cilantro	tbsp	0.5	
167	18:00	cereal (Chocolate Squares)	cup	1.5	
167	20:00	hamburger	oz	5	
167	20:00	bread (wheat)	oz	6	
167	20:00	avocado	cup	0.25	
167	20:00	bacon	oz	0.5	
167	20:00	tomato	cup	0.25	
167	20:00	onion (white)	cup	0.125	

APPENDIX C (continued)

167	20:00	jalapeno	tbsp	0.5	
168	13:10	H2O	oz	16	16-Jun
168	13:10	bread (white)	oz	6	
168	13:10	pork	oz	6	
168	13:10	honey dijon mustard	tbsp	1	
168	13:10	italian spices	tbsp	1	
168	13:10	chili powder	tbsp	0.2	
168	19:45	chip (corn tortilla)	cup	3	
168	19:45	tomato	cup	0.75	
168	19:45	onion (white)	cup	0.25	
168	19:45	jalapeno	tbsp	1	
168	19:45	H2O	oz	16	
168	21:10	steak	oz	8	
168	21:10	capsicum (red)	cup	0.25	
168	21:10	onion (red)	cup	0.25	
168	21:10	pepper	tbsp	1	
168	21:10	garlic	tbsp	0.5	
169	10:30	cereal (Chocolate Squares)	cup	1.5	17-Jun
169	10:30	H2O	oz	16	
169	14:45	turkey	oz	6	
169	14:45	potato	cup	1	
169	14:45	green bean	cup	0.125	
169	14:45	corn	cup	0.125	
169	14:45	preservatives	tbsp	0.5	
169	20:45	pork	oz	2	
169	20:45	green salsa	tbsp	2	
169	20:45	green chili	tbsp	1	
169	20:45	onion (white)	cup	0.05	
169	20:45	cilantro	tbsp	1	
169	20:45	tortilla (corn)	pc	1	
169	20:45	H2O	oz	16	
169	22:15	H2O	oz	16	
170	10:30	pork	oz	2	18-Jun
170	10:30	green salsa	tbsp	2	
170	10:30	green chili	tbsp	1	
170	10:30	onion (white)	cup	0.05	
170	10:30	cilantro	tbsp	1	
170	10:30	tortilla (corn)	pc	1	
170	10:30	H2O	oz	16	
170	14:20	pork	oz	2	

APPENDIX C (continued)

170	14:20	green salsa	tbsp	2	
170	14:20	green chili	tbsp	1	
170	14:20	onion (white)	cup	0.05	
170	14:20	cilantro	tbsp	1	
170	14:20	avocado	cup	0.25	
170	14:20	tortilla (corn)	pc	1	
170	14:20	H2O	oz	16	
170	18:30	H2O	oz	16	
170	19:30	pork	oz	4	
170	19:30	italian spices	tbsp	1	
170	19:30	pasta (wheat)	cup	1.5	
170	19:30	tomato	cup	0.75	
170	19:30	capsicum (red)	cup	0.25	
170	19:30	H2O	oz	16	
171	10:10	cereal (Golden Grahams)	cup	1.5	19-Jun
171	10:10	H2O	oz	16	
171	13:30	pork	oz	4	
171	13:30	italian spices	tbsp	1	
171	13:30	pasta (wheat)	cup	1.5	
171	13:30	tomato	cup	0.75	
171	13:30	capsicum (red)	cup	0.25	
171	13:30	H2O	oz	16	
171	14:30	fruit snacks	oz	6	
171	20:15	H2O	oz	16	
171	21:30	couscous	cup	1.5	
171	21:30	basil	tbsp	0.25	
171	21:30	oregano	tbsp	0.25	
171	21:30	sauce (soy)	tbsp	1	
171	21:30	ginger	tbsp	0.25	
171	21:30	steak	oz	6	
171	21:30	onion (red)	cup	0.25	
171	21:30	capsicum (green)	cup	0.75	
171	21:30	H2O	oz	16	
171	21:30	mushroom (white)	cup	0.25	
171	23:45	H2O	oz	12	
172	10:30	cereal (Golden Grahams)	cup	1.5	20-Jun
172	10:30	H2O	oz	16	
172	14:20	pork	oz	4	
172	14:20	blueberry	cup	0.125	
172	14:20	salt	tsp	0.125	

APPENDIX C (continued)

172	14:20	H2O	oz	16	
172	15:30	H2O	oz	16	
172	16:40	couscous	cup	1.5	
172	16:40	italian spices	tbsp	0.25	
172	16:40	candy (Jello)	cup	0.75	
172	20:00	steak	oz	3	
172	20:00	potato	cup	0.5	
172	20:00	capsicum (yellow)	cup	0.25	
172	20:00	mushroom (white)	cup	0.125	
172	20:00	preservatives	tbsp	1	
172	22:10	chicken	oz	3	
172	22:10	sauce (hotwing)	tbsp	0.5	
172	22:10	H2O	oz	16	
172	23:45	cereal (Chocolate Squares)	cup	1.5	
172	23:45	H2O	oz	16	
173	10:10	cereal (Golden Grahams)	cup	1.5	21-Jun
173	10:10	H2O	oz	16	
173	14:20	hamburger	oz	5	
173	14:20	bread (wheat)	oz	6	
173	14:20	bacon	oz	2	
173	14:20	mushroom (white)	cup	0.125	
173	14:20	sugar	tbsp	2	
173	14:20	pepper	tsp	2	
173	14:20	ketchup	tbsp	1	
173	14:20	mayonaise	tbsp	1	
173	14:20	mustard	tbsp	0.5	
173	14:20	avocado	cup	0.5	
173	14:20	chip (corn tortilla)	cup	1.5	
173	14:20	lettuce (romaine)	cup	0.5	
173	14:20	tomato	cup	0.125	
173	14:20	onion (white)	tbsp	2	
173	14:20	cilantro	tbsp	0.5	
173	23:15	H2O	oz	2	
174	10:15	bread (white)	oz	6	22-Jun
174	10:15	strawberry	cup	0.05	
174	10:15	banana	cup	0.05	
174	10:15	margerine	tbsp	0.25	
174	10:15	H2O	oz	16	
174	12:30	pork	oz	4	
174	12:30	honey dijon mustard	tbsp	1	

APPENDIX C (continued)

174	12:30	tomato	cup	0.125	
174	12:30	onion (white)	tbsp	2	
174	12:30	cilantro	tbsp	0.5	
174	12:30	bread (white)	oz	6	
174	12:30	italian spices	tbsp	1	
174	21:00	couscous	cup	0.25	
174	21:00	italian spices	tbsp	0.5	
174	21:00	H2O	oz	12	
174	21:00	cereal (Chocolate Squares)	cup	1.5	
175	12:30	H2O	oz	16	23-Jun
175	14:30	bread (wheat)	oz	4	
175	14:30	margerine	tbsp	0.75	
175	14:30	jelly (raspberry)	tbsp	1	
175	14:30	jelly (apricot)	tbsp	1	
175	19:45	pasta (wheat)	cup	2	
175	19:45	butter	tbsp	0.5	
175	19:45	salt	tsp	0.25	
175	19:45	pepper	tbsp	0.125	
175	19:45	garlic pepper	tbsp	0.25	
175	19:45	H2O	oz	12	
176	0:10	H2O	oz	4	24-Jun
176	11:30	bread (white)	oz	6	
176	11:30	strawberry	cup	0.05	
176	11:30	banana	cup	0.05	
176	11:30	H2O	oz	12	
176	13:45	couscous	cup	1.5	
176	13:45	H2O	oz	16	
176	13:45	olive oil	tbsp	0.5	
176	13:45	basil	tbsp	0.5	
176	19:10	potato	cup	0.75	
176	19:10	butter	tbsp	0.5	
176	19:10	salt	tsp	0.125	
176	19:10	pepper	tsp	0.125	
176	19:10	H2O	oz	12	
176	20:30	tilapia	oz	2	
176	20:30	tortilla (corn)	oz	4	
176	20:30	fennel	cup	0.5	
176	20:30	cabbage (green)	cup	0.5	
176	20:30	honey chipotle powder	tbsp	0.5	
176	20:30	olive oil	tbsp	0.25	

APPENDIX C (continued)

176	20:30	garlic	tbsp	0.25	
176	20:30	cumin	tbsp	0.1	
176	20:30	cinnamon	tbsp	0.1	
176	20:30	sour cream	cup	0.1	
176	20:30	pepper	tsp	0.1	
176	20:30	salt	tsp	0.1	
176	20:30	juice (lime)	tbsp	0.1	
176	20:30	H2O	oz	12	
177	10:15	bread (wheat)	oz	4	25-Jun
177	10:15	margerine	tbsp	0.5	
177	10:15	jelly (raspberry)	tbsp	1.5	
177	10:15	jelly (apricot)	tbsp	1.5	
177	10:15	H2O	oz	12	
177	14:30	pasta (wheat)	cup	2	
177	14:30	butter	tbsp	0.5	
177	14:30	salt	tsp	0.25	
177	14:30	pepper	tbsp	0.125	
177	14:30	garlic pepper	tbsp	0.25	
177	14:30	H2O	oz	12	
177	17:50	pork	oz	5	
177	17:50	honey dijon mustard	tbsp	1	
177	17:50	italian spices	tbsp	1	
177	17:50	bread (white)	oz	5	
177	22:30	couscous	cup	0.25	
177	22:30	italian spices	tbsp	0.5	
177	23:10	cereal (Golden Grahams)	cup	1.5	
177	23:10	H2O	oz	12	
178	10:20	bread (white)	oz	6	26-Jun
178	10:20	strawberry	cup	0.05	
178	10:20	banana	cup	0.05	
178	10:20	sugar	tbsp	1	
178	15:30	bread (white)	oz	6	
178	15:30	margerine	tbsp	0.5	
178	15:30	jelly (raspberry)	tbsp	1.5	
178	15:30	jelly (apricot)	tbsp	1.5	
178	20:00	potato	cup	1	
178	20:00	sour cream	tbsp	1.5	
178	20:00	salt	tsp	0.125	
178	20:00	pepper	tsp	0.125	
178	22:30	tuna	oz	5	

APPENDIX C (continued)

178	22:30	olive oil	tbsp	1	
178	22:30	honey dijon mustard	tbsp	1.5	
178	22:30	sugar snap pea	cup	0.5	
178	22:30	italian spices	tbsp	0.5	
178	22:30	juice (lemon)	tbsp	0.5	
178	22:30	salt	tsp	0.125	
179	2:30	bread (white)	oz	5	27-Jun
179	2:30	H2O	oz	16	
179	10:20	tilapia	oz	2	
179	10:20	tortilla (corn)	oz	4	
179	10:20	fennel	cup	0.5	
179	10:20	cabbage (green)	cup	0.5	
179	10:20	honey chipotle powder	tbsp	0.5	
179	10:20	olive oil	tbsp	0.25	
179	10:20	garlic	tbsp	0.25	
179	10:20	cumin	tbsp	0.1	
179	10:20	cinnamon	tbsp	0.1	
179	10:20	sour cream	cup	0.1	
179	10:20	pepper	tsp	0.1	
179	10:20	salt	tsp	0.1	
179	10:20	juice (lime)	tbsp	0.1	
179	13:30	H2O	oz	16	
179	14:30	bread (wheat)	oz	6	
179	14:30	peanut butter	tbsp	1	
179	14:30	jelly (raspberry)	tbsp	1.5	
179	14:50	chicken	oz	4	
179	14:50	flour (white)	cup	0.75	
179	14:50	italian spices	tbsp	2	
179	14:50	pork	oz	4	
179	14:50	tomato	cup	0.5	
179	14:50	salt	tsp	0.25	
179	14:50	pepper	tsp	0.25	
179	14:50	sauce (BBQ)	tbsp	1	
179	17:00	H2O	oz	16	
179	17:45	H2O	oz	16	
179	23:30	pork	oz	5	
179	23:30	capsicum (green)	cup	0.5	
179	23:30	leek	cup	0.25	
179	23:30	cumin	tbsp	0.25	
179	23:30	coriander	tbsp	0.25	

APPENDIX C (continued)

179	23:30	turmeric	tbsp	0.125	
179	23:30	cherry	cup	0.5	
179	23:30	H2O	oz	16	
180	11:30	lettuce (romaine)	cup	1.5	28-Jun
180	11:30	bread (white)	oz	2	
180	11:30	italian spices	tbsp	0.5	
180	11:30	dressing (Caesar)	tbsp	2	
180	11:30	chicken	oz	4	
180	11:30	flour (white)	cup	0.75	
180	11:30	italian spices	tbsp	2	
180	11:30	pork	oz	4	
180	11:30	tomato	cup	0.5	
180	11:30	salt	tsp	0.25	
180	11:30	pepper	tsp	0.25	
180	11:30	sauce (BBQ)	tbsp	1	
180	11:30	H2O	oz	16	
180	14:45	H2O	oz	16	
180	21:45	cereal (Cinnamon Toast Crunch)	cup	1.5	
180	21:45	H2O	oz	12	
180	22:10	bread (white)	oz	5	
180	22:10	salmon	oz	4	
180	22:10	bacon	oz	2	
180	22:10	sauce (BBQ)	tbsp	1	
180	22:10	tomato	cup	0.25	
180	22:10	H2O	oz	16	

Exercise Data

	Time	Type	Intensity	Quantity	Date
2	9:50	str	2	55	2-Jan
4	8:45	str	2	75	4-Jan
4	21:20	str	3	55	
5	7:10	str	3	65	5-Jan
6	10:05	str	3	60	6-Jan
7	6:50	str	2	80	7-Jan
8	8:30	str	3	85	8-Jan
8	4:10	str	1	50	
8	4:10	yoga	1	5	
9	14:10	str	2	65	9-Jan
9	14:10	cardio	2	5	
11	9:10	str	3	90	11-Jan
12	15:45	str	3	90	12-Jan
12	15:45	cardio	3	5	
13	15:10	cardio	1	30	13-Jan
15	16:00	str	3	80	15-Jan
16	13:40	str	2	10	16-Jan
16	13:40	cardio	3	10	
16	14:30	cardio	3	5	
17	12:45	str	3	95	17-Jan
17	18:15	str	2	20	
19	8:30	str	3	90	19-Jan
21	10:05	str	2	75	21-Jan
22	8:30	str	3	50	22-Jan
23	10:30	str	3	80	23-Jan
23	17:45	str	2	50	
26	9:35	str	3	55	26-Jan
26	9:35	str	2	60	
26	17:25	str	3	50	
26	20:30	str	2	200	
27	13:30	str	2	200	27-Jan
29	15:20	str	3	135	29-Jan
30	15:30	str	2	115	30-Jan
32	10:10	str	2	100	1-Feb
34	10:00	str	3	70	3-Feb
34	14:10	yoga	2	10	
35	9:45	str	3	90	4-Feb
35	15:30	str	2	90	

APPENDIX C (continued)

36	10:10	str	2	50	5-Feb
36	10:50	str	2	50	
36	18:00	str	3	45	
37	11:30	str	3	60	6-Feb
37	12:30	str	3	60	
38	12:50	str	2	70	7-Feb
38	14:00	str	2	110	
38	15:15	cardio	2	5	
39	10:15	str	2	15	8-Feb
39	10:30	str	2	25	
39	16:30	cardio	3	3	
39	17:50	cardio	3	5	
40	7:25	str	3	75	9-Feb
40	11:10	str	2	80	
40	13:00	str	2	40	
40	15:30	cardio	3	3	
40	22:00	str	3	50	
41	9:55	cardio	3	2	10-Feb
41	14:00	str	2	120	
41	15:30	str	2	40	
41	19:45	str	3	100	
42	12:30	str	1	100	12-Feb
42	17:50	str	3	30	
42	18:15	str	1	100	
44	10:30	str	3	85	14-Feb
44	17:15	str	3	100	
44	10:45	str	3	50	
44	10:45	yoga	2	10	
47	10:45	str	3	65	17-Feb
47	10:45	yoga	2	10	
47	11:20	str	3	60	
48	11:15	str	2	95	18-Feb
48	20:30	yoga	1	45	
49	13:25	str	2	50	19-Feb
49	21:10	str	2	55	
50	12:00	str	1	200	20-Feb
50	12:00	cardio	1	60	
55	17:20	str	3	100	24-Feb
55	18:00	str	3	80	
55	18:30	str	3	70	

APPENDIX C (continued)

55	19:10	str	3	30	
55	21:00	str	2	35	
56	21:30	str	3	50	25-Feb
56	22:00	str	3	60	
56	23:05	cardio	3	2	
57	7:10	str	3	50	26-Feb
57	15:55	str	3	30	
57	17:00	str	3	80	
58	9:40	str	3	60	27-Feb
58	14:55	str	3	60	
60	5:20	str	2	80	29-Feb
60	13:00	str	2	40	
60	20:45	str	2	30	
61	7:00	str	2	75	1-Mar
61	12:45	str	3	110	
61	15:20	str	2	60	
61	15:50	str	3	25	
61	19:15	str	2	50	
62	11:10	str	3	60	2-Mar
62	21:00	str	2	130	
62	21:30	str	2	150	
62	22:45	str	2	150	
63	0:25	str	3	80	3-Mar
63	18:00	str	3	60	
63	18:20	str	3	45	
63	23:15	str	2	50	
64	17:10	str	2	45	4-Mar
64	22:00	str	3	105	
65	18:00	str	3	70	5-Mar
65	23:35	yoga	2	40	
66	11:15	cardio	3	5	6-Mar
67	19:40	str	2	160	7-Mar
67	22:45	str	2	30	
68	12:40	cardio	3	5	8-Mar
68	13:05	cardio	3	5	
70	13:10	str	2	180	10-Mar
70	14:00	str	2	60	
70	14:25	str	2	60	
70	22:50	str	3	60	
70	23:30	str	3	55	

APPENDIX C (continued)

71	12:00	str	3	45	11-Mar
71	12:30	str	2	80	
71	14:10	str	3	120	
71	16:10	str	3	80	
71	22:30	str	3	50	
72	11:15	str	3	50	12-Mar
72	12:45	str	2	90	
72	15:40	str	3	70	
73	14:15	str	3	80	13-Mar
75	19:30	yoga	2	40	15-Mar
79	13:00	str	3	70	19-Mar
79	15:50	str	2	200	
79	17:55	str	3	40	
79	18:35	str	3	50	
79	20:10	str	2	100	
80	13:25	str	2	100	20-Mar
80	14:35	str	3	90	
80	16:20	str	2	90	
81	7:30	yoga	2	40	21-Mar
81	19:55	str	3	80	
89	13:45	str	3	45	29-Mar
90	12:00	str	3	45	31-Mar
90	16:30	str	3	50	
91	22:20	str	3	80	1-Apr
92	17:15	str	3	45	2-Apr
92	20:10	str	2	80	
96	11:55	str	3	50	5-Apr
97	11:50	cardio	1	30	6-Apr
101	11:00	str	3	40	10-Apr
101	13:25	str	3	30	
101	19:10	str	3	35	
102	10:30	str	3	40	11-Apr
102	15:30	str	3	45	
103	21:30	str	3	40	12-Apr
103	21:30	cardio	3	20	
104	20:40	cardio	3	20	13-Apr
104	20:40	str	3	40	
105	21:50	cardio	3	20	14-Apr
105	21:50	str	3	40	
107	14:30	cardio	3	4	16-Apr

APPENDIX C (continued)

107	20:40	cardio	3	20	
107	20:40	str	3	40	
108	20:40	cardio	3	20	17-Apr
108	20:40	str	3	40	
110	12:00	cardio	3	25	19-Apr
110	15:30	cardio	2	30	
112	13:50	cardio	3	45	21-Apr
113	13:00	cardio	3	20	22-Apr
114	22:40	str	2	75	23-Apr
114	23:10	str	2	75	
114	23:35	str	2	60	
118	19:45	cardio	3	20	27-Apr
119	13:50	yoga	1	25	28-Apr
120	15:30	yoga	1	25	29-Apr
123	17:00	yoga	2	25	2-May
138	12:50	str	3	60	17-May
141	16:40	str	3	75	20-May
141	21:35	str	3	60	
149	14:30	cardio	2	10	28-May
149	19:20	str	2	60	
149	20:20	str	3	75	
149	21:05	str	3	60	
170	18:30	cardio	1	20	18-Jun
171	20:15	cardio	1	15	19-Jun
173	18:20	cardio	2	10	21-Jun
173	18:55	str	2	20	
173	20:30	str	3	30	

Symptom Data

	Time	BM	blood	mucus	pus	gas	urgency	cramping	diarhea	Date
1	5:45	1	1	0	5	8	5	0	0	1-Jan
1	7:00	1	2	2	1	4	2	2	2	
1	9:45	1	2	1	1	1	1	0	1	
1	17:30	0	0	0	0	3	0	0	0	
1	18:30	0	0	0	0	4	0	0	0	
1	19:45	0	0	2	0	3	0	0	0	
1	20:55	0	0	2	0	3	0	0	0	
2	5:20	2	5	3	8	5	5	0	3	2-Jan
2	6:30	0	5	3	1	1	2	0	0	
2	6:50	1	7	1	0	1	1	3	4	
2	9:20	1	7	1	0	1	3	4	5	
3	5:30	1	2	0	8	5	0	0	0	3-Jan
3	8:20	1	6	3	0	6	0	0	3	
3	9:20	1	3	2	0	1	0	0	2	
3	19:52	0	0	0	0	2	0	0	0	
3	23:20	0	0	0	0	3	0	0	0	
4	0:10	0	0	0	0	1	0	0	0	4-Jan
4	4:44	0	0	0	0	3	0	0	0	
4	4:45	0	0	0	0	6	0	0	0	
4	8:15	2	2	1	2	1	1	1	0	
4	8:45	1	3	3	2	0	1	1	0	
4	13:00	1	2	2	2	2	1	1	0	
4	16:24	0	0	0	0	2	0	0	0	
4	19:00	0	0	0	0	1	0	0	0	
4	19:40	0	0	0	0	2	0	0	0	
4	20:20	0	0	0	0	1	0	0	0	
4	20:45	0	0	0	0	2	0	0	0	
5	0:10	0	0	0	0	3	0	0	0	5-Jan
5	4:15	0	0	0	0	3	0	0	0	
5	6:20	1	2	2	2	6	1	0	0	
5	7:10	3	2	2	1	3	0	1	1	
5	9:00	1	3	2	0	1	0	2	2	
5	10:50	0	2	3	0	2	0	0	0	
5	15:30	0	0	0	0	1	0	0	0	
5	17:23	1	1	1	1	1	0	0	0	
5	19:45	0	0	0	0	2	0	0	0	
5	21:30	0	0	0	0	3	0	0	0	
6	6:25	1	0	1	1	1	0	0	0	6-Jan

APPENDIX C (continued)

6	9:10	3	1	0	0	1	0	0	0	
6	10:00	1	2	3	1	0	0	1	0	
7	1:45	0	0	0	0	2	0	0	0	7-Jan
7	5:45	2	0	2	1	5	0	0	0	
7	7:40	2	1	1	1	1	2	0	0	
7	13:30	1	1	2	2	1	0	0	0	
7	20:40	0	0	0	0	1	0	0	0	
7	23:00	0	0	0	0	2	0	0	0	
8	5:00	2	1	1	1	5	1	1	0	8-Jan
8	9:45	2	0	1	1	2	0	1	1	
8	16:10	0	0	0	0	1	0	0	0	
8	17:35	0	0	0	2	2	0	0	0	
8	18:15	0	1	1	3	2	0	0	0	
8	19:20	0	0	0	2	2	0	0	0	
8	21:30	0	0	0	0	2	0	0	0	
9	6:00	3	3	0	7	0	0	0	0	9-Jan
9	8:35	4	2	0	0	0	1	0	1	
9	19:10	0	0	0	4	0	0	0	0	
9	19:25	0	0	0	0	3	0	0	0	
10	6:40	2	1	2	3	4	1	1	0	10-Jan
10	8:30	3	2	1	1	3	0	0	0	
10	11:30	1	1	1	0	1	1	1	0	
11	5:20	3	0	1	0	3	0	1	0	11-Jan
11	8:00	2	1	2	1	2	0	1	0	
11	11:15	1	1	1	1	0	1	1	0	
12	3:15	0	0	0	0	3	0	0	0	12-Jan
12	4:50	1	1	2	2	4	0	0	0	
12	8:00	2	1	1	0	0	0	0	0	
12	12:30	0	0	0	0	1	0	0	0	
13	3:50	2	2	1	6	5	0	2	1	13-Jan
13	7:20	2	4	2	0	2	1	1	3	
13	8:10	1	2	3	0	1	0	1	2	
13	23:30	0	0	0	7	0	0	0	0	
14	1:30	0	0	2	4	2	0	0	0	14-Jan
14	4:45	2	1	2	1	1	0	2	1	
14	7:05	3	2	1	0	2	0	1	2	
14	12:25	1	0	1	0	0	0	0	1	
15	6:30	3	1	1	0	3	0	0	0	15-Jan
15	9:20	0	0	0	0	2	0	0	0	
15	10:15	2	2	2	1	0	0	1	0	

APPENDIX C (continued)

15	12:55	5	1	0	1	1	1	0	0	
16	6:25	1	1	3	0	2	0	0	0	16-Jan
16	8:40	2	0	1	0	1	0	0	0	
16	12:45	0	0	0	0	2	0	0	0	
16	13:25	0	0	0	0	3	0	0	0	
16	14:15	0	0	0	0	3	0	0	0	
16	14:55	0	0	0	0	5	0	0	0	
16	15:50	3	1	1	0	2	0	1	0	
16	20:15	0	0	0	0	6	0	0	0	
16	23:30	0	0	0	0	8	0	0	0	
16	23:50	0	0	0	0	2	0	0	0	
17	6:00	2	1	2	2	0	0	0	0	17-Jan
17	7:15	2	1	0	1	1	0	0	0	
17	8:45	1	1	1	1	1	0	0	0	
17	10:00	1	1	1	0	0	1	0	0	
17	22:15	0	0	0	0	3	0	0	0	
18	2:55	2	0	2	1	0	0	0	1	18-Jan
18	6:25	2	0	0	0	0	1	0	1	
18	10:45	0	0	0	0	2	0	0	0	
18	12:45	0	0	0	0	3	0	0	0	
18	13:20	0	1	1	3	2	0	0	0	
18	14:50	1	3	3	2	3	0	0	0	
18	19:00	0	0	0	0	3	0	0	0	
18	22:00	0	0	0	0	2	0	0	0	
18	23:10	0	0	0	0	2	0	0	0	
19	6:40	0	0	0	0	4	0	0	0	19-Jan
19	6:50	3	0	0	0	2	0	1	0	
19	9:10	2	0	0	0	0	1	0	1	
19	12:30	2	0	1	1	0	1	0	0	
19	22:45	0	0	0	0	3	0	0	0	
19	23:55	0	0	0	0	2	0	0	0	
20	4:55	1	3	0	3	4	0	0	0	20-Jan
20	8:10	1	2	1	1	5	0	0	0	
20	15:40	0	0	0	0	2	0	0	0	
20	20:00	0	0	0	0	3	0	0	0	
21	4:50	3	2	0	5	3	0	0	0	21-Jan
21	7:55	2	1	1	1	0	1	0	0	
21	10:30	2	1	1	1	0	0	0	0	
21	12:10	0	0	0	0	3	0	0	0	
21	15:25	0	0	0	0	2	0	0	0	

APPENDIX C (continued)

21	15:55	3	2	0	1	0	0	0	2	
22	0:30	0	0	0	0	4	0	0	0	22-Jan
22	6:15	2	2	1	4	3	0	0	0	
22	8:20	1	3	1	2	2	0	0	0	
22	11:30	2	1	1	0	3	4	0	0	
22	13:40	0	0	0	0	2	0	0	0	
22	16:00	0	0	0	0	2	0	0	0	
22	20:10	0	0	0	0	2	0	0	0	
23	3:00	4	2	0	2	1	1	0	0	23-Jan
23	5:45	3	3	0	0	0	2	0	1	
23	8:45	2	0	0	1	3	1	0	3	
23	17:45	0	0	0	0	2	0	0	0	
23	18:30	0	0	0	0	2	0	0	0	
23	19:00	1	1	0	3	1	0	0	0	
23	21:30	0	0	0	0	2	0	0	0	
24	8:00	3	2	1	0	0	0	0	0	24-Jan
24	11:00	2	0	1	0	0	0	0	0	
24	16:00	0	0	0	0	2	0	0	0	
24	16:30	0	0	0	0	3	0	0	0	
24	18:00	0	0	0	0	3	0	0	0	
24	19:00	1	1	0	3	5	0	0	0	
24	22:20	0	0	0	0	1	0	0	0	
25	3:00	0	0	0	0	3	0	0	0	25-Jan
25	8:00	2	1	2	0	4	0	0	0	
26	5:20	3	0	1	5	0	1	0	2	26-Jan
26	7:30	0	0	0	0	2	0	0	0	
26	7:40	3	4	1	1	0	2	1	3	
26	8:45	2	2	2	1	3	2	0	3	
26	9:25	1	3	3	0	1	4	0	3	
26	10:20	0	0	0	1	2	0	0	0	
26	11:25	2	2	2	1	5	2	0	0	
26	13:00	0	0	0	1	5	0	0	0	
26	13:40	0	0	0	0	4	0	0	0	
26	14:30	0	0	0	0	3	0	0	0	
27	4:30	3	0	1	2	0	1	3	2	27-Jan
27	7:30	2	0	0	0	0	0	1	1	
27	16:30	0	0	0	1	3	0	0	0	
27	18:10	1	1	2	2	3	0	0	0	
28	3:30	0	0	0	0	2	0	0	0	28-Jan
28	6:00	3	1	1	1	0	1	0	1	

APPENDIX C (continued)

28	9:25	3	0	0	0	3	0	0	0	
28	13:30	2	1	0	0	4	1	0	0	
28	16:00	0	0	0	1	5	0	0	0	
28	16:30	0	0	0	0	3	0	0	0	
28	16:45	0	0	0	0	2	0	1	0	
28	16:55	0	0	0	0	4	0	0	0	
28	17:35	3	1	0	2	5	3	0	3	
28	18:20	0	0	0	1	5	0	1	0	
28	19:00	0	0	0	1	5	4	0	0	
28	19:45	0	0	0	1	3	1	0	0	
28	22:30	0	0	0	2	4	2	0	0	
28	22:35	0	0	0	0	5	0	0	0	
29	1:00	0	0	0	1	3	0	0	0	29-Jan
29	1:30	0	0	0	0	3	0	0	0	
29	3:10	0	0	0	2	4	0	0	0	
29	6:45	2	1	0	2	2	0	0	0	
29	10:45	4	2	2	4	7	1	0	2	
29	16:10	0	0	0	0	1	0	0	0	
29	23:00	0	0	0	3	4	0	0	0	
30	0:45	0	1	0	4	3	0	0	0	30-Jan
30	1:30	0	0	0	0	3	0	0	0	
30	8:00	4	3	1	2	3	1	0	1	
30	11:30	4	3	0	1	0	3	0	2	
30	14:50	2	0	3	2	3	3	0	0	
30	16:30	0	0	0	1	3	0	0	0	
30	17:00	0	0	0	0	5	0	0	0	
30	20:30	0	0	0	0	4	0	0	0	
30	21:30	0	0	0	1	6	0	0	0	
31	1:00	3	1	0	3	2	2	0	1	31-Jan
31	5:45	4	1	0	1	4	0	2	2	
31	6:10	4	0	0	0	1	3	1	2	
31	6:15	4	0	0	0	2	3	0	3	
31	17:45	0	0	0	0	2	0	0	0	
31	20:20	0	0	0	0	1	0	0	0	
31	22:30	0	0	0	2	1	0	0	0	
32	5:40	0	0	0	3	2	0	0	0	1-Feb
32	8:00	2	1	1	3	1	0	1	0	
32	10:00	2	2	1	0	0	0	2	0	
32	18:35	1	1	2	4	1	4	0	0	
33	7:00	3	0	1	3	4	0	0	0	2-Feb

APPENDIX C (continued)

33	8:00	3	1	1	1	0	0	0	0	
33	9:20	2	2	0	1	0	2	0	0	
33	10:30	3	2	1	1	3	3	0	0	
34	5:30	3	0	1	2	3	0	0	0	3-Feb
34	9:15	1	0	1	1	2	0	0	0	
34	9:45	2	1	0	1	1	0	0	0	
34	13:50	1	2	0	1	2	0	1	0	
34	18:00	0	0	0	0	2	0	0	0	
34	18:10	1	0	2	0	4	1	0	0	
34	19:45	0	0	0	0	1	0	0	0	
34	21:30	0	0	0	0	2	0	0	0	
34	22:00	2	0	0	1	0	2	0	1	
35	7:20	3	0	0	0	0	0	0	0	4-Feb
35	14:10	0	0	0	0	1	0	0	0	
35	15:30	0	0	0	0	1	0	0	0	
35	17:30	2	0	2	0	1	0	0	0	
35	20:30	0	0	0	0	4	0	0	0	
35	23:15	0	0	0	0	2	0	0	0	
36	6:00	3	1	0	1	3	0	0	0	5-Feb
36	9:00	2	1	2	0	1	0	0	0	
36	17:30	2	1	0	3	1	1	0	0	
36	19:45	0	0	0	0	4	0	0	0	
36	21:15	0	0	0	0	2	0	0	0	
36	22:45	0	0	0	1	2	0	0	0	
37	4:45	1	0	0	1	6	0	0	0	6-Feb
37	5:20	3	0	0	1	4	0	0	0	
37	5:40	0	0	0	0	1	0	0	0	
37	6:20	0	0	0	0	1	0	0	0	
37	7:30	0	0	0	0	1	0	0	0	
37	8:20	3	1	0	1	0	0	0	1	
37	11:15	4	1	2	0	1	1	0	0	
37	11:20	3	0	0	0	0	2	0	2	
37	13:45	1	1	1	1	1	0	0	2	
37	20:30	0	0	0	0	2	0	0	0	
38	3:00	0	0	0	0	4	0	0	0	7-Feb
38	4:15	0	0	0	0	3	0	0	0	
38	5:45	0	0	0	0	3	0	0	0	
38	6:00	0	0	0	0	3	0	0	0	
38	7:20	4	1	1	3	0	0	0	0	
38	9:10	2	0	0	0	1	2	0	0	

APPENDIX C (continued)

38	12:10	2	0	1	1	1	0	0	0	
38	12:20	2	1	0	0	0	2	0	0	
38	16:25	1	0	1	1	1	0	0	0	
38	22:00	0	0	0	0	2	0	0	0	
38	22:20	0	0	0	0	4	0	0	0	
39	0:40	2	1	0	2	6	0	0	0	8-Feb
39	7:00	2	1	0	0	1	2	0	0	
39	8:10	3	3	0	1	0	0	0	0	
39	10:40	1	2	1	2	0	0	0	0	
39	19:50	0	0	0	0	2	0	0	0	
39	20:20	2	0	0	3	2	1	0	0	
40	7:00	4	3	0	2	0	1	0	0	9-Feb
40	8:40	3	2	0	1	0	0	1	0	
40	13:35	0	0	0	0	2	0	0	0	
40	16:50	2	3	1	0	0	0	1	0	
40	22:30	0	0	0	0	2	0	0	0	
41	6:15	3	0	0	0	1	0	0	0	10-Feb
41	8:05	0	0	0	0	3	0	0	0	
41	9:40	0	0	0	0	4	0	0	0	
41	11:00	3	0	0	0	1	0	0	0	
41	13:00	0	0	0	0	2	0	0	0	
41	13:30	3	0	0	0	1	0	1	0	
41	15:45	0	0	0	0	1	0	0	0	
41	16:40	0	0	0	0	2	0	0	0	
41	17:05	2	1	1	0	2	0	0	0	
41	19:45	0	0	0	0	4	0	0	0	
41	20:30	0	0	0	2	3	0	0	0	
41	21:10	0	0	0	1	2	0	0	0	
41	22:05	0	0	0	0	3	0	0	0	
41	23:30	0	0	0	0	4	0	0	0	
42	6:00	5	0	0	2	1	0	0	3	11-Feb
42	7:45	3	2	0	1	0	0	0	0	
42	13:45	3	1	0	1	0	0	0	0	
43	6:00	3	3	0	2	1	0	0	0	12-Feb
43	16:30	0	0	0	1	2	0	0	0	
43	16:35	3	2	0	3	1	3	0	0	
44	5:25	3	2	0	5	1	1	0	0	13-Feb
44	9:30	6	1	1	1	0	1	0	4	
44	10:05	0	0	2	3	0	1	0	0	
44	10:10	0	1	2	2	0	1	0	0	

APPENDIX C (continued)

44	11:00	3	3	2	1	0	4	0	2	
44	19:10	0	0	1	2	1	0	0	0	
45	3:00	0	0	0	0	3	0	0	0	14-Feb
45	5:20	5	1	0	3	2	1	1	2	
45	8:20	3	2	1	2	0	0	0	0	
45	10:15	2	2	2	2	0	0	0	0	
45	13:50	0	0	0	1	0	0	0	0	
45	18:00	0	0	0	1	1	0	0	0	
45	21:00	0	0	0	2	1	0	0	0	
45	21:30	0	0	0	0	3	0	0	0	
46	4:00	3	2	0	3	0	1	1	2	15-Feb
46	7:50	3	3	1	2	2	0	0	0	
46	11:35	2	2	1	2	1	0	0	0	
46	18:40	2	2	1	3	3	0	0	0	
46	23:10	3	0	0	0	3	0	2	3	
47	8:10	2	1	1	1	1	0	0	1	16-Feb
47	12:00	0	0	0	0	2	0	0	0	
47	14:20	2	4	0	2	1	2	0	0	
47	18:25	0	0	0	0	4	0	0	0	
47	18:45	0	0	0	0	2	0	0	0	
47	19:20	0	0	0	2	4	0	0	0	
47	22:10	0	0	0	3	2	0	0	0	
47	22:55	0	0	0	1	3	0	0	0	
47	23:05	0	0	0	1	2	0	0	0	
48	5:00	2	3	1	3	1	0	0	0	17-Feb
48	8:20	3	2	0	1	0	0	1	1	
48	13:10	0	0	0	0	2	0	0	0	
48	18:10	2	3	2	4	0	0	0	1	
48	23:15	0	0	0	0	3	0	0	0	
49	5:00	3	2	0	4	0	1	0	2	18-Feb
49	8:30	3	2	2	3	1	0	0	0	
49	17:20	2	3	0	4	0	0	0	3	
50	2:50	0	0	0	0	3	0	0	0	19-Feb
50	3:30	4	2	1	4	2	0	0	3	
50	7:10	5	4	0	3	2	3	0	0	
50	16:10	0	0	0	0	1	0	0	0	
50	20:45	0	0	0	4	2	3	0	0	
50	22:30	0	0	0	3	1	0	0	0	
51	0:15	0	0	0	4	1	0	0	0	20-Feb
51	6:10	3	7	0	4	1	1	0	0	

APPENDIX C (continued)

51	6:45	2	3	0	2	0	2	0	0	
51	7:30	0	5	0	2	0	3	0	0	
51	14:40	1	4	0	3	2	0	0	0	
51	23:30	0	0	0	0	2	0	0	0	
52	4:00	3	5	1	2	0	0	0	0	21-Feb
52	7:20	4	3	0	2	1	0	1	1	
52	8:40	2	3	2	3	0	0	0	1	
52	9:50	2	3	2	2	0	2	0	0	
52	11:20	4	3	3	3	0	0	0	0	
53	6:00	3	6	1	4	2	1	0	0	22-Feb
53	8:40	3	6	0	4	1	0	0	0	
53	14:00	0	1	1	5	1	0	0	0	
53	21:15	0	0	0	0	3	2	0	0	
53	22:45	0	0	0	4	2	0	0	0	
54	3:45	3	4	2	5	5	0	2	3	23-Feb
54	6:25	2	5	3	3	8	0	0	3	
54	7:30	3	3	2	2	2	0	0	2	
54	15:00	1	2	2	4	2	0	0	0	
54	15:30	1	5	0	3	1	0	0	0	
54	16:10	0	5	0	3	1	0	2	4	
54	20:40	0	0	0	0	1	0	0	0	
54	21:30	0	0	0	0	3	0	0	0	
54	23:00	0	1	0	4	2	0	0	0	
55	4:00	3	4	2	4	4	1	1	2	24-Feb
55	7:30	2	1	0	2	0	0	1	2	
55	7:40	2	1	0	2	0	3	0	0	
56	5:15	3	4	0	5	3	5	0	1	25-Feb
56	11:20	3	3	0	3	0	0	0	0	
56	14:40	2	4	0	3	1	0	0	0	
56	14:55	1	4	0	1	1	0	0	0	
56	19:20	1	3	2	2	4	0	0	0	
56	20:30	0	0	0	1	4	0	0	0	
56	21:45	0	0	0	1	4	0	0	0	
56	22:05	0	0	0	0	2	0	0	0	
56	23:05	0	1	0	1	3	0	0	0	
57	6:00	5	5	1	1	1	0	0	1	26-Feb
57	9:00	2	3	1	3	0	3	0	0	
57	23:30	0	0	1	3	0	0	0	0	
58	3:00	0	0	0	2	2	0	0	0	27-Feb
58	6:00	5	3	0	3	1	0	4	2	

APPENDIX C (continued)

58	7:40	3	3	1	2	0	0	2	3	
58	16:45	0	0	0	3	1	0	0	0	
58	18:15	0	0	0	3	0	0	0	0	
58	19:20	1	3	0	2	3	0	1	0	
58	22:20	0	0	0	0	3	0	0	0	
59	2:25	2	3	1	3	4	0	1	1	28-Feb
59	7:40	4	4	0	2	2	0	0	0	
59	10:25	1	4	0	2	1	0	0	0	
59	13:50	3	5	0	2	2	4	1	3	
60	5:00	3	4	0	2	0	0	1	1	29-Feb
60	8:30	3	3	1	2	0	0	0	0	
60	17:40	0	0	0	0	4	0	0	0	
60	19:50	1	1	0	5	2	3	0	0	
60	21:30	2	2	0	2	3	2	4	2	
60	23:15	2	3	2	1	0	2	2	0	
61	5:00	0	0	0	0	5	0	0	0	1-Mar
61	6:00	5	4	1	3	4	0	3	2	
61	9:45	2	3	1	2	1	0	0	0	
61	17:50	0	0	0	4	4	0	0	0	
61	18:30	5	2	3	3	6	0	1	1	
61	19:45	2	3	0	2	0	2	0	0	
62	5:35	3	2	1	2	2	0	0	0	2-Mar
62	8:25	3	3	0	3	0	0	0	2	
62	9:30	2	3	3	2	0	3	0	1	
62	17:40	0	0	0	3	3	0	0	0	
62	19:00	0	0	0	0	2	0	0	0	
62	22:45	0	0	0	3	3	0	0	0	
62	23:15	0	0	0	0	3	0	0	0	
63	3:45	1	2	1	6	4	0	0	0	3-Mar
63	4:50	4	5	0	4	0	0	1	0	
63	5:10	4	2	1	0	3	7	0	6	
63	5:25	3	0	0	0	3	7	6	0	
63	5:55	2	2	0	1	0	3	0	5	
63	7:00	1	4	3	2	6	9	0	0	
63	7:45	0	3	2	3	6	0	0	5	
63	8:00	0	0	0	0	4	0	0	0	
63	8:20	0	0	0	0	4	0	0	0	
63	10:45	0	0	0	0	4	0	0	0	
63	11:50	0	0	0	0	3	0	0	0	
63	12:10	0	0	0	2	1	0	0	0	

APPENDIX C (continued)

63	12:55	0	0	0	1	1	0	0	0	
63	13:00	2	1	2	0	0	10	0	5	
63	15:00	0	0	0	0	3	0	0	0	
63	16:30	3	2	0	1	0	6	0	6	
63	22:10	0	0	0	0	1	0	0	0	
64	1:40	0	0	0	4	2	0	0	0	4-Mar
64	6:25	2	4	0	6	4	0	0	0	
64	9:05	3	5	0	4	0	3	0	1	
64	15:00	1	2	0	2	3	2	0	0	
64	23:00	0	0	0	0	3	0	0	0	
64	23:20	0	0	0	0	1	0	0	0	
64	23:50	0	0	0	0	3	0	0	0	
65	2:55	0	0	0	4	1	0	0	0	5-Mar
65	4:45	0	0	0	3	4	0	0	0	
65	6:20	0	0	0	0	4	0	0	0	
65	6:30	3	5	0	3	3	0	0	0	
65	7:25	1	2	2	4	3	0	0	0	
65	8:45	0	0	0	0	2	0	0	0	
65	8:50	1	3	1	1	3	1	0	0	
65	9:40	1	3	0	1	3	2	0	0	
65	9:50	2	2	1	0	4	3	0	0	
65	10:30	0	0	0	0	4	1	0	0	
65	10:40	2	3	0	2	6	0	0	0	
65	11:40	0	0	0	0	5	2	0	0	
65	11:50	0	0	0	0	2	0	0	0	
65	11:55	0	0	0	0	5	0	0	0	
65	12:10	3	2	0	0	3	2	0	0	
65	13:10	0	0	0	0	5	0	0	0	
65	13:35	0	0	0	2	4	1	0	0	
65	14:20	1	0	0	2	4	3	0	0	
65	15:00	0	0	0	0	3	0	1	0	
65	18:00	0	0	0	0	2	0	1	0	
65	18:10	0	0	0	2	4	0	0	0	
65	18:45	0	0	0	0	4	0	0	0	
65	19:40	0	0	0	0	6	0	0	0	
65	21:15	0	0	0	0	2	0	0	0	
65	21:20	0	0	0	0	2	0	0	0	
65	22:05	0	0	0	1	0	0	0	0	
65	23:10	0	0	0	3	0	0	0	0	
65	23:30	0	0	0	4	0	0	0	0	

APPENDIX C (continued)

66	2:00	0	0	0	4	3	0	0	0	6-Mar
66	4:25	2	3	0	4	1	0	0	2	
66	6:25	3	3	0	3	2	0	0	4	
66	6:30	3	3	0	1	2	0	0	5	
66	9:50	2	2	3	4	0	0	0	0	
67	3:00	0	0	0	3	2	0	0	0	7-Mar
67	6:20	0	0	0	0	8	0	0	0	
67	6:30	2	6	0	5	2	0	0	2	
67	7:50	1	5	0	1	0	3	0	1	
67	9:55	0	0	0	0	2	0	0	0	
67	11:00	0	0	0	0	3	0	0	0	
67	12:10	1	5	0	3	2	0	1	1	
67	12:20	2	2	0	0	0	3	2	1	
68	5:00	2	6	0	7	4	0	0	0	8-Mar
68	5:45	0	2	0	2	0	0	0	0	
68	8:20	0	1	0	0	4	0	0	0	
68	8:30	2	4	1	4	3	0	0	0	
68	9:45	1	3	0	1	4	3	0	0	
68	9:55	1	2	1	2	4	3	0	0	
68	10:45	1	2	0	2	3	2	0	0	
68	11:30	2	3	0	2	5	3	0	0	
68	12:35	1	3	3	2	6	3	0	0	
68	14:20	2	3	2	3	0	2	0	0	
68	15:25	1	3	2	2	0	3	0	2	
68	18:00	0	0	0	0	4	0	0	0	
68	18:15	0	0	0	0	4	0	0	0	
68	21:10	0	0	0	0	1	0	0	0	
68	22:40	2	5	1	3	0	0	0	4	
69	0:10	0	0	0	0	3	0	0	0	9-Mar
69	5:25	2	4	0	3	0	1	0	0	
69	6:10	4	5	3	2	0	0	0	2	
69	17:30	1	1	0	5	4	3	0	0	
69	19:10	3	4	0	2	0	1	0	0	
69	20:10	2	4	0	2	0	1	4	4	
70	6:15	5	7	0	4	0	0	0	3	10-Mar
70	18:10	0	3	0	6	3	0	0	0	
70	22:30	0	1	0	3	2	0	0	0	
70	23:10	2	4	0	2	1	0	0	0	
71	7:10	4	4	0	1	1	0	0	1	11-Mar
71	9:25	3	3	0	1	2	0	0	0	

APPENDIX C (continued)

71	17:55	1	1	0	3	2	3	0	0	
71	19:55	0	2	0	2	4	0	0	0	
71	21:45	0	0	0	0	2	0	0	0	
71	21:50	0	1	0	3	2	0	0	0	
71	22:00	0	1	0	0	4	0	0	0	
71	23:30	0	0	0	0	4	0	0	0	
72	5:20	4	4	0	3	2	0	0	0	12-Mar
72	5:40	3	2	0	0	1	2	0	0	
72	10:30	3	3	3	2	0	0	0	1	
72	14:30	0	0	0	0	3	0	0	0	
72	14:35	1	2	0	2	1	3	0	0	
72	19:20	0	0	0	0	2	0	0	0	
72	21:10	0	0	0	0	3	0	0	0	
72	21:15	0	1	0	1	1	0	0	0	
72	21:20	0	0	0	0	4	0	0	0	
72	21:25	0	0	0	0	3	0	0	0	
72	21:45	0	1	0	3	2	0	0	0	
72	22:00	0	0	0	0	4	0	0	0	
72	22:30	0	0	0	0	2	0	0	0	
73	0:30	0	0	0	0	3	0	0	0	13-Mar
73	2:00	4	2	1	2	0	0	0	0	
73	5:55	2	3	0	2	1	0	0	0	
73	9:20	4	2	2	2	3	0	0	0	
73	9:50	2	2	2	1	3	0	0	2	
73	15:25	1	0	1	3	1	0	0	0	
73	17:45	0	6	2	2	3	0	2	0	
73	19:25	0	0	0	0	4	0	1	0	
73	19:50	0	3	0	1	3	0	0	0	
73	20:45	0	0	0	0	4	0	0	0	
73	21:20	0	0	0	0	2	0	0	0	
73	21:40	0	0	0	0	4	0	0	0	
73	22:10	0	0	0	0	4	0	0	0	
73	22:40	0	0	0	0	2	0	0	0	
73	23:00	0	3	2	2	4	0	0	0	
74	2:15	0	2	0	3	5	0	0	0	14-Mar
74	2:40	3	4	0	1	2	0	0	0	
74	6:25	6	5	2	2	2	2	0	1	
74	10:55	3	3	2	2	3	2	0	1	
74	20:00	0	0	0	0	4	0	0	0	
74	21:10	0	0	0	3	3	0	0	0	

APPENDIX C (continued)

74	22:10	0	0	0	4	3	2	0	0	
74	23:00	0	0	0	1	3	0	0	0	
75	0:10	0	0	0	0	4	0	0	0	15-Mar
75	3:05	3	5	1	3	0	0	1	2	
75	7:10	3	4	0	3	6	0	0	0	
75	14:20	0	0	0	0	2	0	0	0	
75	15:40	0	0	0	3	1	0	0	0	
75	16:30	0	0	0	3	0	0	0	0	
75	19:30	0	0	0	1	2	0	0	0	
76	2:00	0	3	1	4	3	4	0	0	16-Mar
76	8:30	4	6	2	5	0	2	0	0	
76	14:25	1	1	0	2	4	0	0	0	
76	16:25	0	0	0	0	2	0	4	0	
76	17:55	2	3	1	2	3	1	2	3	
76	20:10	0	0	0	0	2	0	0	0	
76	20:20	0	0	0	0	4	0	0	0	
76	20:25	0	0	0	1	2	0	0	0	
76	21:20	0	0	0	1	4	0	0	0	
77	2:00	0	0	0	3	3	0	0	0	17-Mar
77	5:00	0	0	0	3	2	0	0	0	
77	7:00	5	4	1	2	3	0	1	3	
77	10:45	1	3	1	2	4	0	0	2	
77	13:00	0	0	0	0	5	0	0	0	
77	14:20	1	2	1	1	1	1	0	0	
77	15:10	0	0	0	0	6	0	0	0	
77	16:05	4	4	1	2	8	2	0	2	
77	18:10	0	1	0	2	5	0	0	0	
77	23:15	0	0	0	0	3	0	0	0	
77	2:30	0	0	0	4	2	0	0	0	18-Mar
78	4:00	0	0	0	4	2	0	0	0	
78	7:00	4	5	2	3	5	1	0	1	
78	9:00	2	3	2	2	0	1	0	4	
78	10:45	3	1	3	2	0	3	0	2	
78	15:00	0	0	0	0	4	0	0	0	
78	15:10	0	0	0	0	4	0	0	0	
78	15:30	1	1	3	1	6	2	0	2	
79	3:00	5	3	2	2	4	0	0	1	19-Mar
79	3:55	0	3	1	3	4	6	0	2	
79	4:45	0	0	0	0	4	0	0	0	
79	5:10	0	0	0	0	4	0	0	0	

APPENDIX C (continued)

79	6:00	0	0	0	0	4	0	0	0	
79	6:30	3	4	2	2	4	0	0	4	
79	7:00	0	0	0	0	6	0	0	0	
79	7:10	0	0	0	0	3	0	0	0	
79	7:20	0	0	0	0	4	0	0	0	
79	11:55	0	0	0	1	2	0	0	0	
79	12:00	0	0	0	3	2	0	0	0	
79	12:15	0	0	0	1	3	0	0	0	
79	19:45	0	0	0	0	3	0	0	0	
79	22:15	0	0	0	3	2	0	0	0	
79	23:45	0	0	0	2	4	0	0	0	
80	0:10	0	0	0	0	3	0	0	0	20-Mar
80	0:15	0	0	0	0	2	0	0	0	
80	4:10	3	1	1	3	3	0	4	0	
80	6:45	2	4	3	3	0	0	0	2	
80	7:45	2	5	0	3	4	0	3	3	
80	15:25	0	0	0	0	3	0	0	0	
80	17:30	0	0	0	4	2	5	0	0	
80	18:00	0	0	0	1	3	0	0	0	
80	19:00	0	0	0	2	3	0	0	0	
80	20:10	0	0	0	0	2	0	0	0	
80	22:00	1	1	0	8	4	10	0	1	
81	7:10	0	0	0	6	2	10	0	0	21-Mar
81	7:40	0	0	0	4	2	0	0	0	
81	7:50	0	0	2	3	1	0	0	0	
81	9:40	2	3	2	6	2	3	2	1	
81	11:00	2	3	3	6	5	0	2	0	
81	11:10	3	2	2	1	6	4	2	0	
81	14:10	2	2	2	1	3	1	2	0	
81	18:20	0	0	0	0	3	0	0	0	
82	5:15	4	8	0	8	2	0	0	1	22-Mar
82	14:20	0	0	0	5	2	0	0	0	
82	19:50	0	3	0	6	2	0	0	0	
82	19:55	2	3	0	1	3	0	0	2	
82	23:00	2	9	2	4	3	1	5	2	
83	4:00	5	3	0	3	0	3	2	6	23-Mar
83	7:40	2	3	2	3	0	0	0	0	
83	9:15	1	2	4	3	0	1	0	3	
83	15:15	2	3	2	2	0	2	0	5	
83	21:40	3	4	3	3	0	2	0	5	

APPENDIX C (continued)

84	2:10	0	0	0	0	4	0	0	0	24-Mar
84	3:45	2	4	0	5	2	0	0	1	
84	4:10	1	3	0	3	2	2	3	2	
84	9:40	0	0	0	0	3	0	0	0	
84	9:50	2	3	0	5	0	3	0	0	
84	14:00	0	0	0	4	5	0	0	0	
84	16:45	0	0	0	0	3	0	0	0	
84	17:10	0	1	0	4	3	0	0	0	
84	18:30	0	0	0	3	2	5	0	0	
84	19:45	0	0	0	3	2	0	0	0	
84	20:45	0	0	0	4	2	1	0	0	
85	3:15	0	0	0	2	3	0	0	0	25-Mar
85	3:35	0	0	0	3	1	0	0	0	
85	3:40	0	0	0	0	5	0	0	0	
85	7:20	0	0	0	2	3	0	0	0	
85	8:45	0	1	0	5	2	0	0	0	
85	8:55	2	2	0	3	0	3	0	0	
85	9:05	1	4	0	1	0	4	2	0	
85	15:45	0	0	0	0	2	0	0	0	
85	19:15	0	0	0	3	1	0	0	0	
85	22:30	0	0	0	4	2	0	0	0	
85	23:35	0	0	0	4	1	0	0	0	
86	0:30	0	0	0	2	0	0	0	0	26-Mar
86	0:35	0	0	0	1	0	0	0	0	
86	0:40	0	0	0	0	3	0	0	0	
86	7:20	0	0	0	6	2	10	0	0	
86	8:45	3	3	2	3	4	2	0	1	
86	10:00	0	0	0	0	0	0	4	0	
86	10:30	0	3	3	2	2	0	0	0	
86	11:25	0	0	0	0	0	0	5	0	
86	12:10	3	4	2	2	1	3	0	0	
86	15:35	1	2	0	1	2	0	0	0	
86	17:10	0	0	0	0	3	0	0	0	
86	20:00	1	3	1	2	2	0	1	0	
87	6:30	0	0	0	3	2	0	0	0	27-Mar
87	6:35	2	3	0	0	1	0	1	0	
87	9:00	0	3	4	2	3	0	0	1	
87	23:30	0	0	0	8	3	8	0	0	
88	5:30	0	0	0	3	1	0	0	0	28-Mar
88	8:00	2	7	0	2	3	0	4	0	

APPENDIX C (continued)

88	10:00	3	4	0	5	0	3	1	0	
89	4:25	0	0	0	5	2	0	0	0	29-Mar
89	5:05	1	6	0	2	6	0	2	2	
89	5:10	1	5	0	1	2	2	6	2	
89	5:25	2	0	0	0	1	3	8	4	
89	8:40	2	4	0	0	0	3	4	2	
89	8:45	1	2	0	0	0	4	2	0	
90	5:00	4	7	0	5	3	8	5	2	30-Mar
90	5:40	2	5	3	0	2	7	3	5	
90	10:00	2	4	0	3	0	10	2	0	
90	18:30	4	2	2	5	0	8	0	5	
90	18:40	0	4	4	0	0	0	8	0	
91	6:55	0	1	1	5	3	0	4	0	31-Mar
91	7:00	0	0	0	0	2	0	6	0	
91	7:05	0	0	0	0	4	0	8	0	
91	7:10	2	4	3	1	3	0	0	0	
91	7:20	1	4	2	1	1	0	5	0	
91	17:00	0	4	0	6	1	0	8	0	
91	17:20	0	0	0	0	0	0	6	0	
91	20:45	0	3	0	3	2	0	10	0	
91	20:55	0	0	0	0	0	0	10	0	
91	22:20	0	0	0	0	0	0	7	0	
91	22:45	0	0	0	0	0	0	5	0	
92	2:40	3	6	3	4	3	0	8	0	1-Apr
92	2:55	0	3	2	1	0	0	8	0	
92	3:55	3	2	2	1	3	0	6	0	
92	4:15	2	5	1	0	2	0	8	2	
92	4:25	1	2	0	0	2	0	9	0	
92	6:00	0	0	0	0	0	0	8	0	
92	8:00	1	0	1	0	1	0	8	2	
92	8:15	1	3	2	0	1	0	9	0	
92	10:05	1	1	2	3	0	2	6	0	
92	10:20	0	0	0	0	0	0	8	0	
92	10:40	0	0	0	0	0	0	8	0	
92	11:30	0	0	0	0	0	0	8	0	
92	11:50	0	4	2	3	0	0	10	0	
92	12:30	0	0	0	0	0	0	6	0	
92	13:00	0	0	0	0	0	0	8	0	
92	14:50	0	2	2	1	1	0	10	0	
92	14:55	0	0	0	0	0	0	8	0	

APPENDIX C (continued)

92	18:00	0	0	0	0	0	0	9	0	
92	19:00	0	0	0	0	0	0	6	0	
92	21:45	0	4	0	5	3	0	8	0	
92	22:10	0	0	0	0	0	0	5	0	
92	22:20	0	0	0	0	0	0	6	0	
92	22:40	0	0	0	0	1	0	8	0	
92	23:30	0	0	0	0	0	0	8	0	
92	23:50	0	0	0	0	2	0	8	0	
93	1:30	2	3	1	4	2	3	10	0	2-Apr
93	6:40	0	3	1	4	4	2	8	0	
93	8:00	3	2	2	1	2	0	8	0	
93	9:30	3	1	2	2	0	0	8	1	
93	10:20	0	0	0	0	0	0	7	0	
93	11:00	0	0	0	0	0	0	6	0	
93	11:20	0	0	0	0	0	0	9	0	
93	11:30	2	4	2	1	0	3	10	0	
93	14:00	0	0	0	0	0	0	2	0	
93	19:30	0	0	0	0	0	0	6	0	
93	19:35	2	3	3	5	0	2	8	0	
93	20:05	3	3	2	2	0	4	6	0	
93	20:25	2	2	2	1	0	2	6	4	
94	8:10	1	4	2	7	1	1	7	1	3-Apr
94	9:10	0	3	2	2	0	0	4	0	
94	9:20	0	0	0	0	0	0	4	0	
94	10:10	0	0	0	0	0	0	2	0	
94	10:30	0	4	2	3	1	2	8	0	
94	11:30	2	4	2	1	1	4	6	0	
94	12:55	0	4	2	3	0	3	8	0	
94	17:15	0	2	2	4	0	0	8	0	
94	17:50	0	0	0	0	0	0	4	0	
94	21:25	2	2	2	2	0	4	6	1	
94	22:00	3	2	3	1	0	2	4	5	
94	23:45	0	0	0	0	0	0	7	0	
95	2:00	0	1	0	3	1	0	4	0	4-Apr
95	6:55	2	3	2	4	0	1	0	2	
95	8:40	1	2	4	2	0	0	0	2	
95	9:15	1	1	3	1	0	3	0	3	
95	9:50	1	1	3	1	0	3	4	3	
95	12:50	2	1	1	1	0	4	0	4	
95	19:00	3	2	1	3	0	2	0	2	

APPENDIX C (continued)

95	19:10	3	3	1	1	0	8	3	4	
95	20:00	2	2	2	1	2	2	0	4	
95	21:30	0	2	2	3	0	0	8	0	
95	22:15	0	0	0	0	0	0	6	0	
96	5:40	1	2	3	3	2	0	0	3	5-Apr
96	6:35	2	2	2	1	4	2	0	3	
96	7:00	2	4	2	1	3	2	8	4	
96	7:10	1	2	2	1	3	2	8	4	
96	9:55	1	1	2	1	0	6	2	4	
96	20:30	0	2	2	4	2	2	4	0	
96	22:00	0	0	0	0	0	0	7	0	
96	23:10	0	0	0	0	0	0	6	0	
96	23:40	2	4	1	4	0	2	3	1	
97	4:55	2	0	0	3	2	3	2	3	6-Apr
97	5:10	1	0	0	0	0	3	1	2	
97	5:20	2	2	0	0	2	3	1	1	
97	5:30	2	3	0	0	1	2	1	1	
97	5:55	2	3	0	0	1	2	2	1	
97	8:25	2	3	2	0	1	3	1	2	
97	11:45	0	1	3	3	2	0	0	0	
98	0:05	0	2	0	5	0	4	0	4	7-Apr
98	0:10	0	2	3	5	0	3	0	0	
98	2:25	0	2	3	5	0	1	0	0	
98	6:30	1	3	1	5	0	0	0	3	
98	8:45	2	3	1	2	0	1	1	2	
98	9:00	0	0	0	0	0	0	3	0	
98	9:40	2	3	2	2	0	0	1	0	
98	11:00	2	1	2	1	0	1	1	0	
98	16:45	0	1	0	5	0	3	3	0	
98	19:30	2	4	0	3	3	2	1	0	
98	20:05	2	4	0	1	1	3	2	0	
98	20:20	2	2	0	0	2	8	1	4	
99	0:35	3	1	1	1	3	8	1	0	8-Apr
99	0:45	3	3	0	1	1	6	4	0	
99	5:00	1	3	0	2	2	3	0	5	
99	7:40	1	0	0	1	2	2	0	1	
99	8:50	0	1	1	0	3	0	0	0	
99	9:10	1	3	3	1	2	10	0	0	
99	9:15	2	4	0	2	3	6	0	0	
99	11:00	3	1	2	1	3	4	0	0	

APPENDIX C (continued)

99	12:00	0	2	2	5	0	1	0	0	
99	21:00	0	0	0	7	2	1	0	0	
99	21:50	2	0	0	2	3	2	2	2	
100	2:40	0	0	2	4	2	0	0	0	9-Apr
100	2:50	2	1	2	1	0	3	3	0	
100	6:45	2	3	1	3	0	2	0	0	
100	7:40	1	4	2	1	3	1	3	0	
100	8:10	2	3	0	1	2	0	3	0	
100	9:20	1	2	4	2	3	2	2	3	
100	10:00	1	4	0	2	0	3	2	1	
100	11:00	1	1	7	2	0	2	2	2	
100	14:30	0	4	5	4	2	2	3	2	
100	19:45	2	4	3	2	1	2	2	0	
100	20:00	0	2	0	0	0	0	6	0	
101	4:45	0	4	2	6	1	3	2	2	10-Apr
101	5:00	1	5	0	4	1	8	3	0	
101	7:15	3	5	0	5	2	5	3	6	
101	7:25	2	5	0	0	0	6	0	8	
101	8:30	1	4	0	2	0	4	0	2	
101	8:40	2	5	2	0	1	4	0	3	
101	9:15	2	3	6	1	1	2	2	3	
101	9:50	2	4	5	1	2	8	1	0	
101	18:30	0	0	0	7	2	4	0	0	
101	19:10	0	0	0	0	0	0	3	0	
101	19:40	0	4	3	6	2	2	2	0	
101	19:50	0	4	2	2	1	3	4	0	
101	20:40	0	0	0	0	0	0	4	0	
101	23:45	0	5	2	3	0	2	2	2	
102	3:15	0	2	0	5	4	2	1	0	11-Apr
102	4:05	0	3	2	3	0	2	2	1	
102	5:35	2	2	2	1	2	2	1	2	
102	7:10	2	2	0	2	1	2	2	3	
102	8:30	2	2	2	3	0	1	1	1	
102	8:50	2	2	0	1	0	1	0	1	
102	9:30	0	2	0	3	1	1	2	0	
102	10:00	0	4	4	3	1	4	2	1	
102	11:15	2	3	1	1	2	0	1	0	
102	18:15	3	1	2	5	1	3	0	3	
102	18:25	2	4	2	1	1	5	0	4	
102	18:45	3	3	3	0	1	2	1	4	

APPENDIX C (continued)

102	19:15	2	4	2	0	0	3	0	5	
102	20:45	1	4	1	5	0	6	0	8	
102	21:35	1	1	1	3	0	4	2	6	
103	3:15	1	3	1	4	0	3	1	8	12-Apr
103	6:20	1	1	0	5	4	2	1	3	
103	7:10	3	2	0	2	5	2	0	3	
103	10:40	0	1	0	5	2	5	0	4	
103	10:50	0	1	3	1	5	8	0	0	
103	12:35	1	4	4	2	3	6	0	4	
103	15:40	0	2	2	3	7	9	0	0	
103	17:20	0	2	3	1	3	9	1	0	
103	20:10	1	3	2	1	4	6	1	0	
103	21:05	2	3	0	0	2	4	3	3	
103	21:10	2	4	0	0	1	8	6	4	
103	23:15	2	4	1	2	2	3	0	0	
104	4:10	2	2	0	1	2	2	0	2	13-Apr
104	5:35	1	2	0	1	2	1	4	0	
104	6:15	2	1	3	1	1	2	2	2	
104	7:15	2	1	2	1	2	3	3	0	
104	8:45	0	3	3	2	0	0	0	0	
104	9:25	0	2	3	0	2	0	0	0	
104	14:40	0	1	2	4	2	2	0	0	
104	16:20	0	0	0	0	0	0	6	0	
104	18:30	0	0	0	0	0	0	4	0	
104	22:35	1	4	2	5	3	2	0	0	
104	23:25	2	5	0	1	2	0	8	1	
105	4:05	4	3	0	0	0	2	0	3	14-Apr
105	5:35	2	1	0	1	0	2	0	4	
105	8:45	2	2	2	1	0	4	0	3	
105	9:05	1	4	2	0	0	3	3	1	
105	13:30	0	0	0	0	0	0	4	0	
105	14:30	0	0	0	0	0	0	4	0	
105	16:20	0	0	0	4	0	2	2	0	
105	16:40	3	1	1	2	0	2	0	3	
105	19:20	1	2	0	4	1	4	2	3	
105	19:40	1	4	0	0	2	8	2	4	
105	21:10	2	4	1	0	0	4	6	6	
105	21:30	0	7	1	0	0	7	8	6	
105	23:40	3	6	2	0	0	6	6	6	
106	0:45	1	6	1	3	1	2	2	3	15-Apr

APPENDIX C (continued)

106	2:40	5	5	1	1	4	5	8	0	
106	5:30	2	4	0	1	3	4	2	0	
106	5:50	2	2	0	1	2	4	0	0	
106	8:45	0	4	5	4	1	2	0	0	
106	10:55	2	1	1	1	1	1	0	0	
106	16:45	0	0	0	6	1	10	0	0	
106	18:15	0	2	3	3	1	4	4	0	
106	19:20	6	6	0	2	4	8	6	6	
107	1:00	2	4	3	1	0	2	2	3	16-Apr
107	7:10	2	5	2	4	0	9	2	4	
107	7:30	1	6	3	3	1	7	2	4	
107	9:25	1	4	2	3	1	6	3	0	
107	12:10	0	4	2	4	0	2	4	0	
107	20:30	0	2	2	4	3	3	1	0	
108	1:50	2	4	0	3	0	4	4	3	17-Apr
108	3:45	3	3	0	2	0	2	0	2	
108	4:05	2	4	0	3	0	4	0	4	
108	4:30	1	5	2	3	0	4	0	2	
108	4:45	0	4	0	2	1	2	6	0	
108	5:00	1	3	2	0	2	2	6	0	
108	8:25	0	0	2	6	2	3	0	0	
108	9:35	0	3	0	6	0	3	0	1	
108	10:00	0	6	2	3	2	5	2	1	
108	11:00	0	0	0	0	0	0	6	0	
108	16:15	0	1	0	4	2	3	0	0	
108	19:50	0	2	0	4	1	3	0	0	
108	20:50	0	5	0	3	2	3	2	3	
109	2:20	2	10	2	3	2	7	10	1	18-Apr
109	2:25	0	5	0	0	0	0	6	0	
109	2:35	1	2	0	0	2	4	6	0	
109	2:45	1	4	0	0	1	2	4	0	
109	2:55	1	4	3	0	2	0	8	2	
109	4:55	2	5	3	4	2	6	4	2	
109	6:10	4	1	4	2	2	4	0	4	
109	7:30	0	2	0	3	2	2	0	0	
109	9:55	0	4	8	3	2	4	0	0	
109	20:30	0	0	0	6	2	1	0	0	
109	21:30	0	1	0	4	1	2	0	0	
110	0:55	1	1	3	6	0	9	4	3	19-Apr
110	1:00	0	1	3	4	1	0	4	0	

APPENDIX C (continued)

110	2:45	2	1	2	0	4	8	0	4	
110	4:30	2	0	0	0	0	6	4	4	
110	4:45	2	2	0	1	3	4	6	2	
110	5:10	2	1	0	0	2	2	4	2	
110	8:45	2	4	0	3	2	2	2	1	
110	9:50	2	0	5	3	0	3	2	2	
110	10:10	1	4	6	3	3	9	4	3	
110	15:20	0	0	0	3	2	4	0	0	
110	18:10	2	1	4	1	2	5	5	0	
110	18:25	2	1	0	2	0	3	6	1	
110	20:40	2	4	0	0	0	6	8	3	
110	21:00	3	3	0	0	0	8	6	5	
110	22:00	3	2	0	0	0	7	4	8	
110	22:15	2	3	0	0	0	9	5	8	
111	6:25	3	4	0	5	4	3	4	6	20-Apr
111	6:55	0	2	4	3	0	3	5	0	
111	7:10	1	3	2	1	1	5	0	0	
111	7:25	0	1	2	2	0	0	4	0	
111	8:35	3	1	0	0	0	4	4	5	
111	9:15	1	5	2	1	0	4	6	0	
111	14:30	0	0	0	0	0	0	3	0	
111	15:10	0	0	0	0	0	0	3	0	
111	16:35	0	1	1	3	2	3	0	0	
111	21:00	3	1	0	0	0	2	4	1	
111	21:20	0	3	2	3	1	2	6	0	
111	21:40	0	3	0	2	2	8	6	0	
111	22:10	1	1	0	0	1	3	6	0	
112	1:00	1	1	2	5	1	0	6	4	21-Apr
112	1:30	0	1	3	1	2	0	4	0	
112	8:00	0	0	0	3	4	4	0	0	
112	8:05	3	2	0	2	0	5	2	3	
112	10:00	3	4	0	3	2	0	2	3	
112	11:00	1	3	2	0	2	0	4	0	
112	11:20	0	0	0	0	0	0	4	0	
112	11:30	0	0	0	0	0	0	4	0	
112	13:00	0	0	0	0	0	0	6	0	
112	19:35	0	1	1	7	1	10	0	0	
112	20:00	0	0	0	3	2	3	0	0	
112	22:45	2	2	1	0	2	4	6	0	
112	23:40	2	3	0	2	0	4	3	0	

APPENDIX C (continued)

113	0:10	2	3	0	1	2	5	8	6	22-Apr
113	6:35	2	2	0	3	3	3	0	3	
113	8:05	2	2	0	1	2	3	0	1	
113	8:15	3	4	0	0	2	5	6	3	
113	8:35	2	2	0	0	3	5	6	1	
113	11:30	2	4	0	2	0	6	0	0	
113	14:40	2	2	0	2	0	9	0	2	
113	15:20	1	2	1	3	0	7	0	2	
113	16:00	2	2	2	2	0	6	0	3	
113	18:30	2	2	0	1	0	2	4	0	
113	19:00	0	0	0	0	0	0	5	0	
113	20:00	0	0	0	0	0	0	4	0	
113	20:15	0	0	0	0	0	0	4	0	
114	2:10	0	3	0	4	0	2	0	3	23-Apr
114	3:50	1	3	0	1	7	4	0	1	
114	4:00	2	3	0	0	3	4	4	1	
114	6:10	1	2	0	3	5	4	2	1	
114	8:50	3	3	1	1	4	3	0	1	
114	9:45	1	1	3	1	1	2	0	0	
114	10:10	2	2	1	0	2	4	1	0	
114	17:40	3	4	0	1	0	4	4	2	
114	17:55	0	5	2	2	1	5	6	0	
115	3:00	0	0	0	4	3	6	0	0	24-Apr
115	4:05	3	3	1	2	5	7	0	2	
115	4:15	2	2	0	0	0	8	2	0	
115	4:40	2	2	0	0	0	8	2	1	
115	5:05	0	0	0	0	0	0	5	0	
115	9:00	3	5	1	0	0	6	2	3	
115	10:30	0	0	0	0	0	0	6	0	
115	11:00	0	0	0	0	0	0	6	0	
115	12:00	0	0	0	0	0	0	4	0	
115	18:30	0	1	0	7	0	3	0	0	
115	19:30	0	1	0	6	3	7	0	0	
115	21:55	3	2	0	3	2	7	0	2	
115	22:45	1	5	2	3	2	6	4	0	
115	22:55	0	5	0	0	3	0	6	0	
115	23:25	2	3	2	0	2	0	7	0	
116	1:30	2	4	0	2	3	0	7	0	25-Apr
116	6:10	2	5	0	3	2	0	2	0	
116	8:50	2	4	3	2	3	2	1	2	

APPENDIX C (continued)

116	9:25	2	3	0	2	3	9	0	2	
116	10:25	0	5	0	3	2	2	0	8	
116	17:20	0	2	2	2	3	9	0	0	
116	19:30	0	3	1	3	3	7	0	0	
116	20:55	2	3	0	1	4	6	4	0	
116	21:20	0	2	2	3	3	6	0	0	
116	21:50	2	5	1	1	2	8	6	0	
116	22:15	2	3	0	2	1	8	8	5	
117	2:45	2	4	1	5	3	7	0	1	26-Apr
117	4:15	0	1	2	4	3	4	0	0	
117	5:45	0	0	0	0	3	4	2	0	
117	6:25	1	0	0	3	4	4	1	0	
117	8:00	3	4	0	2	0	4	0	0	
117	8:20	2	3	2	0	0	5	4	3	
117	8:50	1	2	0	3	2	6	4	3	
117	10:30	0	1	0	4	2	3	1	0	
117	19:55	0	1	1	6	2	5	0	0	
117	20:10	1	2	0	2	2	6	1	0	
117	20:55	1	3	0	3	1	4	2	0	
117	23:40	2	4	0	2	1	5	5	3	
118	3:55	2	1	0	2	0	7	2	3	27-Apr
118	4:10	1	4	0	1	2	9	4	2	
118	6:25	4	3	2	2	0	3	3	4	
118	9:45	2	0	2	0	0	2	0	2	
118	19:30	0	0	0	4	2	3	0	0	
118	20:25	0	0	0	0	0	0	4	0	
118	22:30	1	2	0	2	3	6	1	3	
118	22:35	0	4	2	3	3	7	0	0	
118	22:45	0	4	2	3	1	6	0	0	
118	23:00	0	4	2	2	1	5	2	0	
118	23:10	0	4	0	1	4	9	2	6	
118	23:20	1	6	2	1	3	9	6	0	
119	4:15	4	3	2	3	8	9	0	4	28-Apr
119	9:00	0	0	1	3	1	0	0	0	
119	10:00	2	4	0	2	1	1	0	0	
119	10:15	1	4	2	2	1	2	0	0	
119	10:30	2	4	1	0	2	2	0	0	
120	5:30	0	0	0	3	0	0	0	0	29-Apr
120	5:45	0	1	0	3	2	2	0	0	
120	7:05	1	2	0	6	1	9	0	3	

APPENDIX C (continued)

120	7:15	2	4	0	3	0	6	0	3	
120	7:25	2	2	0	0	2	6	4	0	
120	10:30	3	3	1	4	1	2	0	0	
120	10:45	2	2	4	0	2	9	0	0	
120	22:50	0	0	0	4	0	6	0	0	
120	23:25	0	0	0	3	1	2	0	0	
121	2:10	0	0	0	0	0	0	3	0	30-Apr
121	2:45	2	1	0	2	1	1	0	0	
121	7:45	0	3	2	3	1	1	0	3	
121	10:10	2	4	0	5	0	5	0	4	
121	10:20	0	2	0	3	1	6	2	0	
121	10:55	2	3	0	1	0	2	3	0	
121	22:40	0	0	0	7	0	3	0	0	
121	22:50	2	4	0	6	1	10	0	0	
121	23:40	2	6	0	2	0	4	6	2	
121	23:50	0	5	0	2	0	6	2	1	
122	0:10	0	8	2	1	2	4	1	5	1-May
122	9:00	2	4	0	6	1	3	2	3	
122	9:30	2	3	0	1	0	3	0	3	
122	14:20	2	3	3	7	0	10	0	0	
122	14:50	1	4	0	2	1	4	0	0	
122	16:20	3	2	0	1	0	2	0	0	
122	19:10	3	4	2	3	3	4	2	0	
122	22:10	2	3	0	1	1	2	4	0	
123	3:20	2	2	0	3	0	2	0	1	2-May
123	3:35	1	2	0	0	2	3	1	0	
123	5:50	4	4	0	0	1	2	0	2	
123	9:30	1	0	0	3	1	4	4	0	
123	10:15	2	4	1	1	2	4	0	0	
123	20:50	1	0	0	4	2	3	0	2	
124	4:05	0	0	0	3	0	0	0	0	3-May
124	4:10	2	0	1	2	4	7	0	1	
124	4:20	2	2	0	1	3	7	0	3	
124	8:00	2	3	0	2	5	2	0	4	
124	9:55	2	1	3	1	4	6	0	3	
125	0:25	0	0	0	5	0	4	0	0	4-May
125	0:35	0	0	0	6	3	10	0	0	
125	1:00	1	1	1	5	2	9	0	0	
125	5:30	4	3	0	4	3	3	0	3	
125	5:50	2	1	0	0	0	6	1	3	

APPENDIX C (continued)

125	9:30	1	1	0	4	0	1	0	0	
125	20:00	1	0	0	5	3	9	0	0	
125	20:55	1	1	0	4	3	6	0	0	
125	21:20	2	2	0	1	2	3	0	2	
126	3:00	0	0	0	4	0	4	0	0	5-May
126	4:00	0	0	0	4	2	6	0	0	
126	5:40	3	0	0	2	3	5	0	4	
126	9:50	2	0	0	2	3	8	0	3	
126	10:00	2	0	1	1	2	8	0	3	
126	19:10	1	0	0	4	2	6	0	0	
126	21:20	0	0	0	4	3	5	0	0	
126	23:00	0	0	0	4	4	5	0	0	
127	1:35	0	0	0	2	4	6	0	0	6-May
127	1:40	0	0	0	5	3	10	0	0	
127	3:50	2	1	0	2	4	6	0	2	
127	6:10	2	2	0	2	1	6	0	3	
127	8:25	0	0	0	3	5	3	0	0	
127	9:10	4	2	0	1	4	4	0	3	
127	22:50	1	1	0	5	5	8	0	0	
127	22:55	3	2	0	3	4	8	0	2	
128	2:30	0	0	0	2	3	7	0	0	7-May
128	5:20	4	5	0	3	5	6	0	6	
128	9:10	1	2	0	2	2	3	0	2	
128	9:40	0	3	2	3	1	3	0	0	
128	10:25	2	3	0	1	3	5	0	1	
128	21:00	1	0	0	4	2	2	0	0	
129	0:20	0	0	0	4	3	2	0	0	8-May
129	4:00	3	4	0	5	3	4	0	3	
129	4:20	1	1	0	0	6	7	1	3	
129	8:00	4	4	0	2	3	8	0	7	
129	13:00	3	4	0	3	0	2	0	4	
129	15:10	2	3	0	0	3	7	0	4	
129	15:20	0	5	0	3	0	6	2	2	
129	22:00	0	0	0	2	3	3	0	0	
129	22:45	2	0	0	3	1	6	0	1	
130	5:55	0	0	0	3	0	2	0	0	9-May
130	7:00	2	0	0	2	1	3	0	0	
130	7:20	1	2	3	3	3	0	0	0	
130	7:25	3	3	0	0	3	4	3	0	
130	7:40	0	0	0	0	3	0	1	0	

APPENDIX C (continued)

130	9:50	2	3	0	4	0	6	0	2	
130	12:15	2	2	1	0	2	9	0	1	
130	20:00	0	0	0	4	3	4	0	0	
130	20:25	3	2	0	0	2	6	2	1	
130	23:20	1	1	0	2	1	6	0	3	
131	2:30	0	0	0	4	2	0	0	0	10-May
131	5:05	2	0	0	5	5	9	0	5	
131	5:15	2	2	0	0	5	4	0	0	
131	7:10	2	2	0	2	4	3	0	0	
131	9:10	2	0	0	0	0	3	0	2	
131	11:55	2	2	1	1	2	3	0	3	
131	19:00	2	1	0	1	2	5	1	0	
131	19:50	3	2	0	1	0	6	0	2	
131	20:00	0	2	2	1	1	2	0	0	
131	23:40	0	3	0	1	4	9	0	0	
132	2:20	3	0	0	2	0	4	0	3	11-May
132	5:35	3	0	0	2	4	9	0	2	
132	5:40	3	2	0	0	4	8	0	2	
132	10:00	2	0	0	4	3	10	0	2	
132	11:15	2	0	0	1	5	3	0	2	
132	16:00	2	0	0	1	3	3	0	0	
132	22:30	2	0	0	3	2	6	0	5	
133	8:15	0	3	0	5	0	2	0	2	12-May
133	10:05	3	4	3	3	2	3	0	0	
133	10:15	0	1	0	1	2	7	0	0	
133	13:20	2	0	0	2	2	6	0	3	
133	17:30	1	0	0	3	1	3	0	0	
133	21:45	2	0	0	1	2	4	0	0	
133	23:10	0	0	0	3	3	3	0	0	
134	5:15	1	0	0	2	3	3	0	0	13-May
134	8:30	5	3	0	1	4	2	2	0	
134	11:30	2	0	0	1	2	4	2	0	
134	20:30	0	0	0	4	2	3	0	0	
134	21:30	0	0	0	4	2	6	0	0	
135	1:45	0	0	0	3	3	8	0	0	14-May
135	1:50	0	0	0	3	3	7	0	0	
135	5:25	0	0	0	5	1	6	0	0	
135	5:35	4	3	0	3	4	7	0	2	
135	5:55	3	2	0	0	1	8	1	0	
135	7:30	5	4	0	2	3	6	2	4	

APPENDIX C (continued)

135	9:45	0	3	0	1	0	2	0	2	
135	14:30	0	0	0	0	0	0	2	0	
135	15:30	0	0	0	0	0	0	2	0	
136	2:05	0	0	0	5	0	0	0	0	15-May
136	2:15	2	3	0	3	6	7	0	0	
136	5:25	0	0	0	4	2	6	0	0	
136	5:30	3	3	0	1	6	6	2	0	
136	5:40	3	5	0	0	1	5	2	0	
136	8:50	3	4	0	2	4	8	0	2	
136	9:00	2	1	0	0	2	7	3	2	
136	12:50	2	2	0	2	2	8	0	0	
136	22:45	2	0	0	7	3	6	0	0	
137	4:15	3	3	0	4	1	3	2	4	16-May
137	7:35	4	0	0	4	0	3	0	3	
138	5:10	0	0	0	5	0	4	0	0	17-May
138	5:15	2	0	0	7	2	10	0	0	
138	5:25	0	3	0	3	1	4	4	0	
138	6:20	4	3	0	3	1	3	2	0	
138	10:00	4	3	0	2	2	3	0	2	
138	23:45	0	0	0	8	1	1	0	0	
139	9:00	1	1	0	5	1	2	0	0	18-May
139	9:10	2	3	0	0	4	6	0	0	
139	11:15	3	5	0	2	0	4	3	2	
139	23:45	0	0	0	6	0	3	0	0	
140	3:15	0	0	0	3	1	4	0	0	19-May
140	6:15	1	0	0	3	2	5	0	0	
140	6:20	3	3	0	0	0	8	0	0	
140	6:25	5	4	0	0	2	9	2	3	
140	10:30	4	4	0	1	0	1	0	0	
141	2:00	2	1	0	3	1	3	0	1	20-May
141	2:45	2	1	0	2	1	3	0	3	
141	7:00	2	2	0	2	4	4	4	4	
141	7:15	1	2	0	0	0	8	4	2	
141	11:45	3	0	0	2	0	2	0	2	
142	8:00	1	0	0	6	1	6	0	1	21-May
142	8:15	2	0	0	4	2	6	0	2	
142	9:55	5	5	0	3	0	4	2	5	
142	19:15	3	0	0	5	1	4	2	2	
142	19:45	0	5	0	1	1	8	4	1	
142	21:30	2	4	0	0	2	6	4	4	

APPENDIX C (continued)

143	2:15	1	0	0	4	1	3	2	0	22-May
143	8:15	0	0	0	3	1	2	0	0	
143	9:25	2	6	0	0	0	4	2	4	
143	9:35	0	6	0	0	0	8	1	4	
143	9:45	0	5	0	0	0	8	2	0	
143	16:00	0	0	0	0	0	0	8	0	
143	16:10	0	0	0	0	0	0	2	0	
143	16:30	0	0	0	0	0	0	8	0	
143	17:55	0	0	0	0	0	0	8	0	
143	19:25	3	5	2	4	0	7	6	3	
143	19:35	1	7	0	1	0	6	8	2	
144	3:45	0	4	0	6	2	3	2	0	23-May
144	8:05	2	6	0	8	4	3	1	3	
144	10:15	2	6	0	3	0	6	2	3	
144	23:45	2	3	0	5	2	9	4	3	
144	23:55	3	6	0	2	3	7	2	3	
145	1:45	3	5	1	1	0	4	2	0	24-May
145	4:15	3	3	0	2	0	4	1	5	
145	4:25	2	3	0	0	1	6	0	6	
145	7:05	3	1	0	1	1	8	0	3	
145	11:05	1	3	0	0	0	3	10	0	
145	13:20	2	1	0	3	0	2	2	0	
145	23:20	0	0	0	0	0	0	8	0	
146	1:15	0	0	0	0	0	0	10	0	25-May
146	6:15	3	1	0	2	3	9	0	2	
146	6:25	1	3	0	1	0	6	0	3	
146	7:35	2	1	0	0	0	4	0	2	
146	7:45	3	3	0	0	0	6	0	3	
146	9:45	2	2	0	0	3	7	0	2	
146	12:00	2	2	0	0	4	6	0	2	
146	13:45	2	3	0	0	4	9	0	4	
146	15:45	2	3	0	0	4	5	0	2	
146	16:00	3	2	0	0	2	7	0	4	
146	17:50	3	1	0	2	0	9	0	5	
147	2:25	1	1	0	2	0	8	0	7	26-May
147	10:45	2	1	0	3	2	3	0	5	
147	13:50	1	1	0	2	0	5	0	8	
147	18:20	0	1	0	3	3	2	0	0	
148	3:00	1	1	0	5	0	4	0	0	27-May
148	5:50	2	0	0	3	1	8	0	5	

APPENDIX C (continued)

148	6:00	2	4	0	1	2	9	0	4	
148	10:30	2	3	0	3	1	7	0	5	
148	11:35	2	3	0	2	1	7	0	5	
148	11:55	0	3	0	4	1	6	1	1	
148	21:00	2	0	0	7	2	5	0	2	
148	21:45	1	2	0	2	1	6	0	3	
149	9:20	0	0	0	8	0	3	0	0	28-May
149	9:30	0	0	0	2	3	7	0	0	
149	11:30	1	2	0	2	0	2	0	0	
149	11:45	1	3	0	3	1	2	0	0	
149	20:05	0	0	0	7	0	3	0	0	
149	21:20	2	0	0	7	1	6	2	2	
149	21:25	0	4	0	0	3	7	4	0	
150	1:30	3	4	0	2	2	5	2	3	29-May
150	1:45	2	7	0	0	1	7	4	6	
150	9:30	0	1	0	5	0	2	0	0	
150	11:05	2	2	0	4	2	10	0	0	
150	11:15	2	4	0	0	0	7	0	1	
150	11:35	1	3	0	0	1	7	2	0	
150	13:45	0	1	0	1	0	0	0	0	
150	17:00	2	1	0	4	2	4	0	0	
151	0:15	2	2	0	3	2	6	0	1	30-May
151	0:45	1	4	0	0	3	7	4	0	
151	9:40	3	10	0	0	1	3	5	0	
151	10:20	1	4	0	0	2	1	1	0	
151	16:20	2	4	0	3	0	3	0	0	
151	22:00	2	2	0	5	3	4	2	2	
152	4:30	1	2	0	3	1	2	0	1	31-May
152	8:10	3	2	0	2	3	4	0	2	
152	8:25	2	3	0	1	3	8	0	4	
152	9:40	2	2	0	3	4	6	2	5	
152	9:50	1	1	0	1	3	5	1	0	
152	15:10	3	0	0	5	2	7	1	2	
153	5:15	0	1	0	8	3	10	0	0	1-Jun
153	8:20	2	2	0	3	0	3	0	0	
153	13:30	0	2	0	5	0	4	2	1	
153	19:10	0	2	0	6	1	5	2	2	
153	21:45	0	1	0	2	0	2	2	0	
154	4:00	0	1	0	3	1	3	2	0	2-Jun
154	8:15	2	2	0	5	2	4	1	3	

APPENDIX C (continued)

154	8:20	3	4	0	3	0	8	0	2	
154	9:45	2	5	0	3	0	8	3	3	
154	10:15	1	4	0	2	0	7	5	2	
154	12:30	0	5	0	2	0	8	5	2	
154	12:45	0	4	0	1	0	9	4	0	
154	13:10	2	2	0	0	3	6	2	2	
154	16:10	0	0	0	0	0	0	5	0	
154	19:00	0	2	0	7	0	3	3	0	
154	22:30	0	2	0	7	0	4	4	2	
154	22:50	0	1	0	7	0	6	3	1	
155	5:30	1	1	0	6	3	5	2	2	3-Jun
155	9:10	2	3	0	4	2	7	0	0	
155	10:30	3	1	0	3	3	6	0	0	
155	10:50	2	2	0	3	2	5	0	3	
155	11:45	2	2	0	2	0	8	3	2	
155	12:20	2	0	0	3	2	6	3	0	
155	16:00	2	2	0	2	2	5	0	3	
155	20:30	0	0	0	0	0	0	2	0	
156	1:15	0	1	0	7	0	7	3	0	4-Jun
156	1:20	2	2	0	0	2	8	3	3	
156	2:00	2	6	0	6	2	8	4	5	
156	2:10	0	3	0	0	1	3	4	0	
156	6:05	2	3	0	2	0	7	2	3	
156	6:15	3	3	2	3	0	9	4	6	
156	7:10	1	2	0	3	0	4	2	2	
156	9:50	3	3	0	2	0	2	1	1	
156	10:20	0	3	2	0	0	6	1	2	
156	13:45	0	0	0	0	0	0	2	0	
156	17:45	0	0	0	0	0	0	3	0	
156	21:00	3	3	0	0	1	5	3	4	
156	21:25	3	6	3	0	0	6	6	7	
156	21:50	0	4	3	0	0	7	7	8	
156	22:20	0	0	0	0	0	0	2	0	
157	2:30	1	3	0	3	1	3	0	2	5-Jun
157	4:45	2	2	0	1	3	4	2	4	
157	6:20	2	2	0	1	2	4	5	5	
157	7:40	2	2	0	1	1	5	3	6	
157	8:00	2	3	0	1	1	6	2	6	
157	8:40	1	2	0	1	1	4	3	6	
157	9:00	1	1	0	0	1	4	4	7	

APPENDIX C (continued)

157	9:15	0	0	0	1	0	2	6	0	
157	9:50	1	1	0	2	1	3	6	6	
157	12:30	1	2	0	3	0	4	0	4	
157	15:55	2	2	3	3	0	9	1	1	
157	20:45	0	1	0	3	0	2	0	0	
157	22:00	1	2	3	2	0	7	1	0	
157	22:05	1	3	2	3	0	9	3	0	
157	23:30	0	0	0	0	0	0	2	0	
158	0:15	0	0	0	3	0	3	4	2	6-Jun
158	4:45	2	2	0	2	3	6	1	3	
158	7:15	3	4	0	2	3	9	0	4	
158	9:10	2	2	0	1	3	7	0	6	
158	9:30	2	3	0	1	3	7	0	7	
158	9:50	1	1	0	1	3	6	0	7	
158	10:45	2	2	0	2	2	6	2	4	
158	17:20	0	1	0	8	1	3	0	0	
158	19:45	0	1	0	8	2	4	0	0	
158	22:30	0	2	0	5	3	4	1	0	
158	22:45	1	3	0	2	2	6	0	0	
158	23:10	2	4	0	2	2	6	2	0	
158	23:25	0	6	0	1	2	7	4	0	
159	2:45	2	3	0	3	2	3	0	3	7-Jun
159	4:10	2	3	0	3	2	3	0	3	
159	4:45	2	3	0	3	2	3	0	3	
159	7:10	2	3	0	3	2	3	0	3	
159	8:00	2	1	0	3	2	3	0	3	
159	8:10	0	2	0	0	3	0	2	0	
159	8:40	2	2	0	0	0	3	2	4	
159	17:30	0	2	0	3	2	5	2	0	
159	18:45	2	2	0	0	3	6	1	6	
159	20:10	3	2	0	2	2	8	3	8	
159	21:45	2	1	0	2	1	8	2	8	
159	22:30	2	2	0	1	1	8	4	8	
160	1:30	0	0	0	0	0	0	4	0	8-Jun
160	2:10	2	3	0	2	2	6	3	2	
160	4:15	2	2	0	3	0	3	2	2	
160	9:45	2	2	0	2	0	4	0	3	
160	16:30	2	2	0	2	3	5	2	4	
160	17:15	2	1	0	3	2	6	0	5	
160	19:10	3	3	3	2	3	8	3	6	

APPENDIX C (continued)

160	20:45	1	4	3	3	2	4	4	1	
160	22:30	2	4	2	2	3	4	2	5	
161	2:45	1	3	0	3	0	2	0	1	9-Jun
161	12:00	3	3	0	2	0	3	1	5	
161	12:10	2	5	0	1	0	5	2	6	
161	12:30	0	0	0	0	0	0	3	0	
161	18:10	0	1	0	5	0	4	2	1	
161	18:40	0	1	0	5	0	4	0	1	
161	19:30	1	2	0	4	0	4	2	3	
161	22:40	0	4	3	1	3	5	4	0	
161	22:55	0	3	2	1	2	5	4	0	
162	4:30	2	5	0	0	2	6	4	4	10-Jun
162	10:45	2	4	3	0	3	4	4	3	
162	11:20	0	4	2	4	2	0	4	0	
162	14:30	0	0	0	0	0	0	4	0	
162	15:40	0	0	0	0	0	0	2	0	
162	18:00	1	2	1	2	1	4	4	3	
162	21:30	3	3	0	2	3	9	0	2	
162	21:45	0	4	0	3	1	7	4	0	
163	7:15	3	2	0	6	0	4	2	4	11-Jun
163	10:00	3	4	0	3	2	6	0	5	
163	12:20	3	3	2	3	0	4	2	4	
163	20:00	2	4	3	2	2	6	4	6	
163	21:10	3	5	4	2	3	7	5	8	
163	21:20	0	5	2	2	1	9	6	8	
164	0:45	3	2	2	2	3	8	0	4	12-Jun
164	7:50	2	3	0	7	2	8	2	6	
164	8:00	2	2	0	5	0	6	0	4	
164	10:45	2	3	3	2	2	6	1	0	
164	13:45	2	3	0	3	2	4	0	4	
164	16:10	2	4	0	0	2	9	2	3	
164	23:30	0	1	0	9	1	2	0	0	
165	1:10	2	0	0	3	3	4	0	0	13-Jun
165	5:10	1	3	0	4	2	6	0	4	
165	9:10	2	3	0	5	1	2	0	0	
165	17:30	0	0	0	0	0	0	4	0	
165	17:50	0	0	0	0	0	0	4	0	
165	18:40	0	0	0	0	0	0	2	0	
165	19:30	0	0	0	0	0	0	4	0	
165	20:20	0	1	0	3	1	0	2	0	

APPENDIX C (continued)

165	20:35	4	4	0	2	0	4	4	8	
165	23:30	0	0	0	3	2	4	2	0	
166	1:00	3	2	0	3	0	6	4	5	14-Jun
166	4:15	1	1	0	7	1	6	0	4	
166	6:10	2	1	0	3	1	5	0	6	
166	7:10	1	2	0	3	1	4	0	7	
166	9:10	2	1	0	2	0	5	0	8	
166	11:10	1	3	0	1	0	7	0	7	
166	12:45	3	2	0	0	1	6	0	5	
166	14:20	1	1	0	0	1	7	0	6	
166	16:20	1	1	0	2	2	8	0	5	
167	2:15	2	2	0	4	2	5	2	4	15-Jun
167	7:15	2	1	0	3	1	6	0	0	
167	9:30	2	2	0	3	2	4	0	1	
167	11:30	1	0	0	3	1	4	0	2	
167	19:45	2	1	0	3	0	4	2	1	
167	22:00	2	1	0	2	0	6	4	3	
168	1:10	2	6	0	0	3	6	6	7	16-Jun
168	1:20	1	5	0	0	2	6	6	6	
168	7:35	2	3	0	0	1	5	2	4	
168	12:30	0	4	0	2	0	6	2	4	
168	18:00	0	0	0	0	0	0	4	0	
168	19:00	0	3	0	6	0	4	6	3	
168	20:10	1	4	0	2	1	6	4	0	
169	1:00	2	3	0	0	0	4	2	3	17-Jun
169	4:45	2	2	0	2	0	8	2	6	
169	5:15	2	0	0	2	0	8	2	5	
169	9:30	2	3	0	4	0	6	0	3	
169	10:45	0	0	0	0	0	0	4	0	
169	10:55	0	2	0	5	0	6	4	6	
169	18:30	2	3	0	6	0	8	4	6	
169	19:30	0	1	0	4	0	6	2	8	
169	21:00	0	2	0	4	2	3	4	2	
169	22:00	0	3	0	3	0	3	0	0	
169	22:15	0	0	0	0	0	0	6	0	
170	7:45	2	0	0	4	3	6	0	4	18-Jun
170	10:45	2	2	0	4	0	6	4	6	
170	12:30	2	3	0	3	3	6	0	7	
170	14:30	1	3	3	4	2	0	4	6	
170	14:40	0	3	2	2	1	0	6	4	

APPENDIX C (continued)

170	18:30	0	0	0	0	0	0	4	0	
170	20:10	1	4	0	5	0	6	6	7	
170	22:10	1	5	2	3	0	4	4	8	
171	7:10	2	0	2	7	3	9	0	8	19-Jun
171	10:00	1	5	3	3	0	6	3	4	
171	10:20	0	0	0	0	0	0	4	0	
171	11:30	1	3	0	4	0	7	6	5	
171	18:10	2	3	1	3	4	6	0	6	
171	22:15	2	3	2	0	0	6	4	0	
172	7:00	2	3	0	0	1	6	2	6	20-Jun
172	7:30	1	2	0	0	2	6	2	7	
172	10:00	2	2	2	3	2	0	4	6	
172	11:45	1	3	2	3	0	6	0	4	
172	15:30	0	0	0	0	0	0	2	0	
172	18:30	0	0	0	5	2	2	0	0	
172	20:45	1	3	0	6	2	3	2	6	
173	1:45	0	2	0	6	2	6	0	4	21-Jun
173	2:00	2	3	0	3	2	7	0	6	
173	2:30	2	5	0	2	2	8	0	9	
173	7:30	2	5	1	0	1	6	0	8	
173	7:50	1	4	0	0	1	8	0	8	
173	10:20	1	3	0	5	1	6	0	4	
173	13:25	0	3	0	6	2	6	4	3	
173	14:00	0	5	0	1	0	5	5	5	
173	20:00	2	2	0	6	2	4	0	4	
173	22:10	0	4	4	0	2	8	4	2	
173	23:15	0	0	0	0	0	0	6	0	
174	0:10	2	6	0	0	2	6	6	7	22-Jun
174	0:20	0	8	0	0	1	7	7	8	
174	2:20	1	8	0	0	2	7	7	8	
174	2:25	0	1	0	0	0	6	7	1	
174	3:15	1	5	0	0	2	4	8	3	
174	7:45	2	4	0	4	0	7	0	4	
174	9:15	1	5	0	5	0	6	4	3	
174	10:45	1	4	0	6	0	4	3	4	
174	11:45	1	4	3	4	0	6	0	6	
174	13:30	0	4	0	4	2	8	4	4	
174	15:45	0	3	2	3	1	6	0	3	
174	18:00	1	0	0	7	1	5	0	2	
174	23:00	1	0	0	6	1	0	4	6	

APPENDIX C (continued)

175	3:30	1	0	0	6	1	7	4	6	23-Jun
175	7:30	1	2	0	6	2	8	4	8	
175	10:45	2	1	0	4	3	6	3	3	
175	15:15	1	3	0	6	0	8	5	6	
175	17:30	1	1	3	5	0	7	0	7	
175	20:10	2	1	0	7	0	7	4	7	
175	22:30	1	2	2	6	0	6	2	6	
176	4:15	1	0	0	4	0	6	3	6	24-Jun
176	7:30	2	1	2	6	0	7	0	7	
176	9:15	1	1	0	7	0	8	2	6	
176	11:45	2	0	0	4	1	6	4	7	
176	14:20	1	1	4	4	0	8	0	6	
176	22:45	3	1	2	4	1	5	0	8	
177	8:10	1	2	0	6	0	5	0	6	25-Jun
177	9:10	3	1	3	4	0	6	0	8	
177	10:00	2	1	0	5	2	8	0	7	
177	13:15	2	0	0	3	0	6	3	7	
177	13:45	2	2	7	2	0	7	2	0	
177	18:45	1	1	1	3	0	6	0	7	
177	23:45	2	0	0	3	2	6	0	6	
178	3:30	1	2	2	3	1	7	0	8	26-Jun
178	7:45	1	0	0	2	1	6	0	7	
178	10:15	2	0	0	3	1	3	2	8	
178	14:30	0	0	6	1	1	6	2	4	
178	20:30	3	0	0	2	0	6	4	6	
178	23:45	1	0	0	3	1	6	0	7	
179	7:30	1	1	0	0	2	6	0	8	27-Jun
179	9:20	2	1	0	0	1	6	0	7	
179	11:30	1	1	0	0	1	4	0	5	
179	23:00	2	0	0	2	1	4	0	4	
180	0:45	3	2	0	3	1	9	0	7	28-Jun
180	1:00	1	5	0	1	3	9	2	8	
180	4:35	2	2	0	3	1	7	0	8	
180	7:20	3	1	0	0	2	9	0	10	
180	7:30	1	0	0	0	2	7	0	9	
180	10:10	1	1	0	3	1	6	0	8	
180	10:30	2	2	0	0	2	8	3	6	
180	14:45	1	0	0	5	1	7	0	6	
180	19:30	2	0	1	4	3	9	0	7	

APPENDIX D

ProcessData.cpp

```
/**
 * \file ProcessData.cpp
 *
 * Processes all data files into BPNN
 * data vectors
 *
 * ---------------------------------------
 * Jonathan Lessner. 2012
 */


#include "Includes.h"

using namespace std;

void ProcessSymptomFile(string file_name);
void ProcessExerciseFile(string file_name);
void ProcessMedicationFile(string file_name);
void ProcessDietFile(string file_name);
void SelectTestIndexes(string file_name);

//---------------------------------------------------------------------------

void main(int argc, char * argv[])
{
	string inputLine;
	bool from_bat;

	if(argc < 2)	from_bat = false;
	else		from_bat = true;

	if(from_bat)
	{
		inputLine = (string) argv[1];
	}
	else
	{
		// Read user input
		cout << "Enter data file type,
(s)ymptom/(m)edication/(d)iet/(e)xercise, or select (t)estcases: ";
		getline(cin, inputLine);
	}

	// Select file process type
	if(inputLine == "s" || inputLine == "symptom")
	{
		if(from_bat)
		{
			inputLine = (string) argv[2];
		}
		else
```

APPENDIX D (continued)

```
{
            // Read user input
            cout << "Enter symptom file name" << endl;
            getline(cin, inputLine);
      }

      ProcessSymptomFile(inputLine);
}
else if(inputLine == "m" || inputLine == "medication")
{
      if(from_bat)
      {
            inputLine = (string) argv[2];
      }
      else
      {
            // Read user input
            cout << "Enter medication file name" << endl;
            getline(cin, inputLine);
      }

      ProcessMedicationFile(inputLine);
}
else if(inputLine == "d" || inputLine == "diet")
{
      if(from_bat)
      {
            inputLine = (string) argv[2];
      }
      else
      {
            // Read user input
            cout << "Enter diet file name" << endl;
            getline(cin, inputLine);
      }

      ProcessDietFile(inputLine);
}
else if(inputLine == "e" || inputLine == "exercise")
{
      if(from_bat)
      {
            inputLine = (string) argv[2];
      }
      else
      {
            // Read user input
            cout << "Enter exercise file name" << endl;
            getline(cin, inputLine);
      }

      ProcessExerciseFile(inputLine);
}
else if(inputLine == "t" || inputLine == "testcase")
```

APPENDIX D (continued)

```
	{
		if(from_bat)
		{
			inputLine = (string) argv[2];
		}
		else
		{
			// Read user input
			cout << "Enter symptom file name" << endl;
			getline(cin, inputLine);
		}

		SelectTestIndexes(inputLine);
	}
	else
	{
		cout << "Invalid Command. (try
(s)ymptom/(m)edication/(d)iet/(e)xercise)" << endl;
	}

	// Read next line and restart processing loop
	cout << "Processing finished... Hit <enter> to close window" << endl;
	getline(cin, inputLine);
}

#define DEBUG_PRINT true
void debugOut(stringstream print_line)
{
	if(DEBUG_PRINT)
		cout << print_line;
}

#define SYMPTOM_COLUMNS 10
#define NUM_SYMPTOMS    8
void ProcessSymptomFile(string file_name)
{
	cout << "Processing Symptom File: " << file_name << endl;

	string inputLine;
	std::vector<int> inputVector;
	std::vector<std::vector<DataEntry>> sympEntries;

	// Open input file
	ifstream inputFile(file_name.c_str());

	if(!inputFile.is_open())
	{
		cout << "File open failed!" << endl;
	}

	// Prepare symptom vectors
	for(int i = 0; i < NUM_SYMPTOMS; ++i)
	{
		std::vector<DataEntry> empty_symp_vector;
```

APPENDIX D (continued)

```
			sympEntries.push_back(empty_symp_vector);
		}

		int tries = 0;
//		while(tries < 1)
		while(true)
		{
			inputVector.clear();
			getline(inputFile, inputLine);
			//cout << inputLine.size() << endl;
			//cout << inputLine << endl;

			if(inputFile.eof())
				break;

			int char_cnt = 0;
			for(int column = 0; column < SYMPTOM_COLUMNS; ++column)
			{
				stringstream inputColumn;
				bool done = false, pre_semi = true;
				while(!done)
				{
					if(inputLine[char_cnt] >= '0' && inputLine[char_cnt]
<= '9')
					{
						if(pre_semi)
							inputColumn << inputLine[char_cnt];
					}
					else if(inputLine[char_cnt] == '\t')
					{
						char_cnt++;
						break;
					}
					else if(inputLine[char_cnt] == ':')
						pre_semi = false;
					else if(inputLine[char_cnt] == '\n')
						done = true;
					char_cnt++;
				}
				inputVector.push_back(atoi(inputColumn.str().c_str()));
				//cout << inputVector.back() << endl;
			}
			//for(unsigned int i = 0; i < inputVector.size(); ++i)
				//cout << inputVector[i] << "\t" << endl;

			// Calculate entry time
			int entry_time = ((inputVector[0]-1) * 24) + inputVector[1];
			//cout << entry_time << endl;

			// Add to vector for each (non-0) symptom
			for(int symp_idx = 0; symp_idx < NUM_SYMPTOMS; ++symp_idx)
			{
				if(inputVector[symp_idx + 2] > 0)
				{
```

APPENDIX D (continued)

```
                        DataEntry temp_symp_entry;
                        temp_symp_entry.time = entry_time;
                        temp_symp_entry.value = (double) inputVector[symp_idx
+ 2];

                        //cout << "S#: " << symp_idx << ", T: " <<
temp_symp_entry.time << ", V: " << temp_symp_entry.value << endl;
                        sympEntries[symp_idx].push_back(temp_symp_entry);
                  }
            }

            ++tries;
      }
      inputFile.close();

      // Output exercise data vectors to files
      for(int symp_idx = 0; symp_idx < NUM_SYMPTOMS; ++symp_idx)
      {
            stringstream outfile_name;
            outfile_name << "../Data/output/SYMP" << symp_idx << ".vec";
            //cout << outfile_name.str() << endl;
            ofstream outputFile(outfile_name.str().c_str(),
ios_base::binary);

            //cout << "S#: " << symp_idx << "\t\n";
            for(unsigned int i = 0; i < sympEntries[symp_idx].size(); ++i)
            {
                  //cout << "\tT: " << sympEntries[symp_idx][i].time << ", V:
" << sympEntries[symp_idx][i].value << endl;
                  outputFile.write((char*) &sympEntries[symp_idx][i],
sizeof(DataEntry));
            }
            outputFile.close();
      }

/*
      // Read test
//    unsigned int i = 0;
      for(unsigned int i = 0; i < NUM_SYMPTOMS; ++i)
      {
            stringstream testfile_name;
            testfile_name << "../Data/output/SYMP" << i << ".vec";
            ifstream inputTestFile(testfile_name.str().c_str(),
ios_base::binary);
            std::vector<DataEntry> read_test_vec;
            while(true)
            {
                  DataEntry test_symp_entry;
                  inputTestFile.read((char*) &test_symp_entry,
sizeof(DataEntry));

                  if(inputTestFile.eof())
                        break;
```

APPENDIX D (continued)

```
                  read_test_vec.push_back(test_symp_entry);

                  //cout << "E#: " << read_test_vec.size() << ", T: " <<
read_test_vec.back().time << ", V: " << read_test_vec.back().value << endl;
            }
            inputTestFile.close();
      }
*/
}

#define EXERCISE_COLUMNS      5
void ProcessExerciseFile(string file_name)
{
      cout << "Processing Exercise File: " << file_name << endl;

      std::vector<string> exerName;
      string inputLine;
      std::vector<int> inputVector;
      std::vector<std::vector<DataEntry>> exerEntries;

      // Open input file
      ifstream inputFile(file_name.c_str());

      if(!inputFile.is_open())
      {
            cout << "File open failed!" << endl;
      }

      int tries = 0;
//    while(tries < 1)
      while(true)
      {
            inputVector.clear();
            getline(inputFile, inputLine);
            //cout << inputLine.size() << endl;
            //cout << inputLine << endl;

            if(inputFile.eof())
                  break;

            int char_cnt = 0;
            string exercise_type;
            for(int column = 0; column < EXERCISE_COLUMNS; ++column)
            {
                  stringstream inputColumn;
                  bool done = false, pre_semi = true;
                  while(!done)
                  {
                        if((inputLine[char_cnt] >= '0' && inputLine[char_cnt]
<= '9') ||
                           (inputLine[char_cnt] >= 'A' && inputLine[char_cnt]
<= 'Z') ||
                           (inputLine[char_cnt] >= 'a' && inputLine[char_cnt]
<= 'z'))
```

APPENDIX D (continued)

```
                        {
                              if(pre_semi)
                                    inputColumn << inputLine[char_cnt];
                        }
                        else if(inputLine[char_cnt] == '\t')
                        {
                              char_cnt++;
                              break;
                        }
                        else if(inputLine[char_cnt] == ':')
                              pre_semi = false;
                        else if(inputLine[char_cnt] == '\n')
                              done = true;
                        char_cnt++;
                  }

                  if(column == 2)
                        exercise_type = inputColumn.str();
                  else

      inputVector.push_back(atoi(inputColumn.str().c_str()));
            }

            //cout << exercise_type << endl;
            //for(unsigned int i = 0; i < inputVector.size(); ++i)
                  //cout << inputVector[i] << "\t" << endl;

            // Find vector idx for type of exercise (or push a new one)
            unsigned int vector_idx;
            bool push_new = true;
            for(unsigned int i = 0; i < exerName.size(); ++i)
            {
                  if(exercise_type == exerName[i])
                  {
                        push_new = false;
                        vector_idx = i;
                  }
            }

            if(push_new)
            {
                  vector_idx = exerName.size();

                  exerName.push_back(exercise_type);
                  std::vector<DataEntry> empty_exer_vector;
                  exerEntries.push_back(empty_exer_vector);
            }

            DataEntry temp_exer_entry;

            // Calculate entry time
            temp_exer_entry.time = ((inputVector[0]-1) * 24) +
inputVector[1];
```

APPENDIX D (continued)

```
            // Calculate value (intensity * reps-or-mins)
            temp_exer_entry.value = (double) (inputVector[2] *
inputVector[3]);

            // Add the entry to the appropraite vector
            //cout << "V#: " << vector_idx << ", E: " << exerName[vector_idx]
<< ", T: " << temp_exer_entry.time << ", V: " << temp_exer_entry.value <<
endl;
            exerEntries[vector_idx].push_back(temp_exer_entry);

            ++tries;
      }
      inputFile.close();

      // Output symptom data vectors to files
      ofstream outputCatFile("../Data/output/EXER_CAT.txt");
      for(unsigned int exer_type = 0; exer_type < exerName.size();
++exer_type)
      {
            // Write the name out to the category file
            outputCatFile << exerName[exer_type] << endl;

            //cout << exerName[exer_type] << ":" << endl;
            stringstream outfile_name;
            outfile_name << "../Data/output/EXER" << exer_type << ".vec";
            //cout << outfile_name.str() << endl;
            ofstream outputFile(outfile_name.str().c_str(),
ios_base::binary);

            for(unsigned int exer_entry = 0; exer_entry <
exerEntries[exer_type].size(); ++exer_entry)
            {
                  //cout << "\tT: " <<
exerEntries[exer_type][exer_entry].time << ", V: " <<
exerEntries[exer_type][exer_entry].value << endl;
                  outputFile.write((char*)
&exerEntries[exer_type][exer_entry], sizeof(DataEntry));
            }
            outputFile.close();
      }
      outputCatFile.close();

/*
      // Read test
      ifstream inputCatFile("../Data/output/EXER_CAT.txt");
      std::vector<string> categoryNames;
      while(true)
      {
            string testLine;
            getline(inputCatFile, testLine);
            //cout << testLine << endl;

            if(inputCatFile.eof())
                  break;
```

APPENDIX D (continued)

```
			categoryNames.push_back(testLine);
		}
		//cout << categoryNames.size() << endl;

		for(unsigned int exer_type = 0; exer_type < categoryNames.size();
++exer_type)
		{
			stringstream testfile_name;
			testfile_name << "../Data/output/EXER" << exer_type << ".vec";
			ifstream inputTestFile(testfile_name.str().c_str(),
ios_base::binary);
			std::vector<DataEntry> read_test_vec;

			cout << categoryNames[exer_type] << ":" << endl;
			while(true)
			{
				DataEntry test_exer_entry;
				inputTestFile.read((char*) &test_exer_entry,
sizeof(DataEntry));

				if(inputTestFile.eof())
					break;

				read_test_vec.push_back(test_exer_entry);

				cout << "\tT: " << read_test_vec.back().time << ", V: " <<
read_test_vec.back().value << endl;
			}
			inputTestFile.close();
		}
*/
}


#define MEDICATION_COLUMNS	4
void ProcessMedicationFile(string file_name)
{
		cout << "Processing Medication File: " << file_name << endl;

		std::vector<string> mediName;
		string inputLine;
		std::vector<double> inputVector;
		std::vector<std::vector<DataEntry>> mediEntries;

		// Open input file
		ifstream inputFile(file_name.c_str());

		if(!inputFile.is_open())
		{
			cout << "File open failed!" << endl;
		}

		int tries = 0;
```

APPENDIX D (continued)

```
//    while(tries < 1)
      while(true)
      {
            inputVector.clear();
            getline(inputFile, inputLine);
            //cout << inputLine.size() << endl;
            //cout << inputLine << endl;

            if(inputFile.eof())
                  break;

            int char_cnt = 0;
            string medication_type;
            for(int column = 0; column < MEDICATION_COLUMNS; ++column)
            {
                  stringstream inputColumn;
                  bool done = false, pre_semi = true;
                  while(!done)
                  {
                        if((inputLine[char_cnt] >= '0' && inputLine[char_cnt]
<= '9') || (inputLine[char_cnt] == '.') ||
                           (inputLine[char_cnt] >= 'A' && inputLine[char_cnt]
<= 'Z') ||
                           (inputLine[char_cnt] >= 'a' && inputLine[char_cnt]
<= 'z') ||
                           (inputLine[char_cnt] == ' ' || inputLine[char_cnt]
== '(' || inputLine[char_cnt] == ')'))
                        {
                              if(pre_semi)
                                    inputColumn << inputLine[char_cnt];
                        }
                        else if(inputLine[char_cnt] == '\t')
                        {
                              char_cnt++;
                              break;
                        }
                        else if(inputLine[char_cnt] == ':')
                              pre_semi = false;
                        else if(inputLine[char_cnt] == '\n')
                              done = true;
                        char_cnt++;
                  }

                  if(column == 2)
                        medication_type = inputColumn.str();
                  else

      inputVector.push_back(atof(inputColumn.str().c_str()));
            }

            //cout << medication_type << endl;
            //for(unsigned int i = 0; i < inputVector.size(); ++i)
                  //cout << inputVector[i] << "\t" << endl;
```

APPENDIX D (continued)

```
            // Find vector idx for type of medication (or push a new one)
            unsigned int vector_idx;
            bool push_new = true;
            for(unsigned int i = 0; i < mediName.size(); ++i)
            {
                  if(medication_type == mediName[i])
                  {
                        push_new = false;
                        vector_idx = i;
                  }
            }

            if(push_new)
            {
                  vector_idx = mediName.size();

                  mediName.push_back(medication_type);
                  std::vector<DataEntry> empty_medi_vector;
                  mediEntries.push_back(empty_medi_vector);
            }

            DataEntry temp_medi_entry;

            // Calculate entry time
            temp_medi_entry.time = (int) ((inputVector[0]-1) * 24) +
inputVector[1];

            // Calculate dosage value
            temp_medi_entry.value = inputVector[2];

            // Add the entry to the appropraite vector
            //cout << "V#: " << vector_idx << ", E: " << mediName[vector_idx]
<< ", T: " << temp_medi_entry.time << ", V: " << temp_medi_entry.value <<
endl;
            mediEntries[vector_idx].push_back(temp_medi_entry);

            ++tries;
      }
      inputFile.close();

      // Output medication data vectors to files
      stringstream medi_cat_filename;
      ofstream outputCatFile("../Data/output/MEDI_CAT.txt");
      for(unsigned int medi_type = 0; medi_type < mediName.size();
++medi_type)
      {
            // Write the name out to the category file
            outputCatFile << mediName[medi_type] << endl;

            //cout << mediName[medi_type] << ":" << endl;

            stringstream outfile_name;
            outfile_name << "../Data/output/MEDI";
            if(medi_type < 10) outfile_name << "0";
```

APPENDIX D (continued)

```
            outfile_name << medi_type << ".vec";
            //cout << outfile_name.str() << endl;

            ofstream outputFile(outfile_name.str().c_str(),
ios_base::binary);

            for(unsigned int medi_entry = 0; medi_entry <
mediEntries[medi_type].size(); ++medi_entry)
            {
                  //cout << "\tT: " <<
mediEntries[medi_type][medi_entry].time << ", V: " <<
mediEntries[medi_type][medi_entry].value << endl;
                  outputFile.write((char*)
&mediEntries[medi_type][medi_entry], sizeof(DataEntry));
            }

            outputFile.close();

      }
      outputCatFile.close();

/*
      // Read test
      ifstream inputCatFile("../Data/output/MEDI_CAT.txt");
      std::vector<string> categoryNames;
      while(true)
      {
            string testLine;
            getline(inputCatFile, testLine);
            //cout << testLine << endl;

            if(inputCatFile.eof())
                  break;

            categoryNames.push_back(testLine);
      }
      //cout << categoryNames.size() << endl;

      for(unsigned int medi_type = 0; medi_type < categoryNames.size();
++medi_type)
      {
            stringstream testfile_name;
            testfile_name << "../Data/output/MEDI";
            if(medi_type < 10) testfile_name << "0";
            testfile_name << medi_type << ".vec";
            ifstream inputTestFile(testfile_name.str().c_str(),
ios_base::binary);
            std::vector<DataEntry> read_test_vec;

            cout << categoryNames[medi_type] << ":" << endl;

            while(true)
            {
                  DataEntry test_medi_entry;
```

APPENDIX D (continued)

```
                  inputTestFile.read((char*) &test_medi_entry,
sizeof(DataEntry));

                  if(inputTestFile.eof())
                        break;

                  read_test_vec.push_back(test_medi_entry);

                  cout << "\tT: " << read_test_vec.back().time << ", V: " <<
read_test_vec.back().value << endl;
            }
            inputTestFile.close();
      }
*/
}

#define DIET_COLUMNS    5
void ProcessDietFile(string file_name)
{
      cout << "Processing Diet File: " << file_name << endl;

      std::vector<string> dietName;
      string inputLine;
      std::vector<double> inputVector;
      std::vector<std::vector<DataEntry>> dietEntries;

      // Open input file
      ifstream inputFile(file_name.c_str());

      if(!inputFile.is_open())
      {
            cout << "File open failed!" << endl;
      }

      int tries = 0;
//    while(tries < 1)
      while(true)
      {
            inputVector.clear();
            getline(inputFile, inputLine);
            //cout << inputLine.size() << endl;
            //cout << inputLine << endl;

            if(inputFile.eof())
                  break;

            int char_cnt = 0;
            string diet_type, unit_type;
            for(int column = 0; column < DIET_COLUMNS; ++column)
            {
                  stringstream inputColumn;
                  bool done = false, pre_semi = true;
                  while(!done)
                  {
```

APPENDIX D (continued)

```
                        if((inputLine[char_cnt] >= '0' && inputLine[char_cnt]
<= '9') || (inputLine[char_cnt] == '.') ||
                           (inputLine[char_cnt] >= 'A' && inputLine[char_cnt]
<= 'Z') ||
                           (inputLine[char_cnt] >= 'a' && inputLine[char_cnt]
<= 'z') ||
                           (inputLine[char_cnt] == ' ' || inputLine[char_cnt]
== '(' || inputLine[char_cnt] == ')'))
                        {
                              if(pre_semi)
                                    inputColumn << inputLine[char_cnt];
                        }
                        else if(inputLine[char_cnt] == '\t')
                        {
                              char_cnt++;
                              break;
                        }
                        else if(inputLine[char_cnt] == ':')
                              pre_semi = false;
                        else if(inputLine[char_cnt] == '\n')
                              done = true;
                        char_cnt++;
                  }

                  if(column == 2)
                        diet_type = inputColumn.str();
                  else if(column == 3)
                        unit_type = inputColumn.str();
                  else

      inputVector.push_back(atof(inputColumn.str().c_str()));
            }

//          cout << diet_type << endl;
//          cout << unit_type << endl;
//          for(unsigned int i = 0; i < inputVector.size(); ++i)
//                cout << inputVector[i] << "\t" << endl;

            // Find vector idx for type of diet (or push a new one)
            unsigned int vector_idx;
            bool push_new = true;
            for(unsigned int i = 0; i < dietName.size(); ++i)
            {
                  if(diet_type == dietName[i])
                  {
                        push_new = false;
                        vector_idx = i;
                  }
            }

            if(push_new)
            {
                  vector_idx = dietName.size();
```

APPENDIX D (continued)

```
                  dietName.push_back(diet_type);
                  std::vector<DataEntry> empty_diet_vector;
                  dietEntries.push_back(empty_diet_vector);
            }

            DataEntry temp_diet_entry;

            // Calculate entry time
            temp_diet_entry.time = (int) ((inputVector[0]-1) * 24) +
inputVector[1];

            // Calculate quantity value (with conversions - to ounces)
            if(unit_type == "oz")               // "no conversion"
                  temp_diet_entry.value = inputVector[2];
            else if(unit_type == "cup")         // "8x conversion"
                  temp_diet_entry.value = inputVector[2] * 8;
            else if(unit_type == "tbsp")  // "0.5x conversion"
                  temp_diet_entry.value = inputVector[2] * 0.5;
            else if(unit_type == "tsp")   // "0.167x conversion"
                  temp_diet_entry.value = inputVector[2] * 0.167;
            else if(unit_type == "pc")    // "no conversion" (special cases,
such as juiced fruit or candy)
                  temp_diet_entry.value = inputVector[2];
            else
            {
                  cout << "unhandled unit type: " << unit_type << ", at line:
" << tries << endl;
                  temp_diet_entry.value = 0;
            }

            // Add the entry to the appropraite vector
            //cout << "V#: " << vector_idx << ", E: " << dietName[vector_idx]
<< ", T: " << temp_diet_entry.time << ", V: " << temp_diet_entry.value <<
endl;
            dietEntries[vector_idx].push_back(temp_diet_entry);

            ++tries;
      }
      inputFile.close();

      // Sort food names (alphabetically), for verification of no
misspelling)
//    std::vector<string> alphaName;
      std::vector<unsigned int> dietIdx;
      std::vector<bool> used_name;
      for(unsigned int i = 0; i < dietName.size(); ++i)
            used_name.push_back(false);
      while(dietIdx.size() < dietName.size())
      {
            unsigned lowest_idx = 0;
            while(used_name[lowest_idx])
                  ++lowest_idx;

            for(unsigned int i = 0; i < dietName.size(); ++i)
```

APPENDIX D (continued)

```
			{
				if(!used_name[i])
				{
					if(dietName[i] < dietName[lowest_idx])
						lowest_idx = i;
				}
			}

//			alphaName.push_back(dietName[lowest_idx]);
			dietIdx.push_back(lowest_idx);
			used_name[lowest_idx] = true;
		}
//		for(unsigned int i = 0; i < dietIdx.size(); ++i)
//			cout << "slot: " << i << ", idx: " << dietIdx[i] << ", name: " <<
dietName[dietIdx[i]] << endl;



		// Output diet data vectors to files
		stringstream diet_cat_filename;
		ofstream outputCatFile("../Data/output/DIET_CAT.txt");
		for(unsigned int diet_type = 0; diet_type < dietName.size();
++diet_type)
		{

			// Write the name out to the category file
			outputCatFile << dietName[dietIdx[diet_type]] << endl;

			if(diet_type <= 4)
				cout << dietName[dietIdx[diet_type]] << ":" << endl;

			stringstream outfile_name;
			outfile_name << "../Data/output/DIET";
			if(diet_type < 100) outfile_name << "0";
			if(diet_type < 10) outfile_name << "0";
			outfile_name << diet_type << ".vec";
			//cout << outfile_name.str() << endl;

			ofstream outputFile(outfile_name.str().c_str(),
ios_base::binary);

			for(unsigned int diet_entry = 0; diet_entry <
dietEntries[dietIdx[diet_type]].size(); ++diet_entry)
			{
				if(diet_type <= 4)
					cout << "\tT: " <<
dietEntries[dietIdx[diet_type]][diet_entry].time << ", V: " <<
dietEntries[dietIdx[diet_type]][diet_entry].value << endl;
				outputFile.write((char*)
&(dietEntries[dietIdx[diet_type]][diet_entry]), sizeof(DataEntry));
			}

			outputFile.close();
```

APPENDIX D (continued)

```
	}
	outputCatFile.close();

/*
	// Read test
	ifstream inputCatFile("../Data/output/DIET_CAT.txt");
	std::vector<string> categoryNames;
	while(true)
	{
		string testLine;
		getline(inputCatFile, testLine);
		//cout << testLine << endl;

		if(inputCatFile.eof())
			break;

		categoryNames.push_back(testLine);
	}
	//cout << categoryNames.size() << endl;

	for(unsigned int diet_type = 0; diet_type < categoryNames.size();
++diet_type)
	{
		stringstream testfile_name;
		testfile_name << "../Data/output/DIET";
		if(diet_type < 100) testfile_name << "0";
		if(diet_type < 10) testfile_name << "0";
		testfile_name << diet_type << ".vec";
		ifstream inputTestFile(testfile_name.str().c_str(),
ios_base::binary);
		std::vector<DataEntry> read_test_vec;

		if(diet_type <= 4)
			cout << categoryNames[diet_type] << ":" << endl;

		while(true)
		{
			DataEntry test_diet_entry;
			inputTestFile.read((char*) &test_diet_entry,
sizeof(DataEntry));

			if(inputTestFile.eof())
				break;

			read_test_vec.push_back(test_diet_entry);

			if(diet_type <= 4)
				cout << testfile_name.str() << "\tT: " <<
read_test_vec.back().time << ", V: " << read_test_vec.back().value << endl;
		}
		inputTestFile.close();
	}
*/
}
```

APPENDIX D (continued)

```
void SelectTestIndexes(string file_name)
{
	cout << "Selecting Test Indexes:" << endl;

	string inputLine;
	int max_day = 0;
	int max_idx, num_test_idxs;
	std::vector<int> test_idx;
	//std::vector<std::vector<DataEntry>> sympEntries;

	// Open input file
	ifstream inputFile(file_name.c_str());

	if(!inputFile.is_open())
	{
		cout << "File open failed!" << endl;
	}

	int tries = 0;
//	while(tries < 1)
	while(true)
	{
		getline(inputFile, inputLine);
		//cout << inputLine.size() << endl;
		//cout << inputLine << endl;

		if(inputFile.eof())
			break;

		int char_cnt = 0;
		for(int column = 0; column < SYMPTOM_COLUMNS; ++column)
		{
			stringstream inputColumn;
			bool done = false, pre_semi = true;
			while(!done)
			{
				if(inputLine[char_cnt] >= '0' && inputLine[char_cnt]
<= '9')
				{
					if(pre_semi)
						inputColumn << inputLine[char_cnt];
				}
				else if(inputLine[char_cnt] == '\t')
				{
					char_cnt++;
					break;
				}
				else if(inputLine[char_cnt] == ':')
					pre_semi = false;
				else if(inputLine[char_cnt] == '\n')
					done = true;
				char_cnt++;
			}
```

APPENDIX D (continued)

```
                  // add hour index only
                  if(column == 0)
                  {
                        if(atoi(inputColumn.str().c_str()) > max_day)
                              max_day = atoi(inputColumn.str().c_str());
                  }
            }
            ++tries;
      }
      inputFile.close();

      // Find total number of indexes (each data day is split into eight 3-
hour chunks)
      max_idx = max_day * CHUNKS_PER_DAY;

      // Output max index to file
      stringstream outfile_maxidx_name;
      outfile_maxidx_name << "../Data/output/MAXINDEX.vec";
      cout << outfile_maxidx_name.str() << endl;
      ofstream outputMaxIdxFile(outfile_maxidx_name.str().c_str(),
ios_base::binary);
      outputMaxIdxFile.write((char*) &max_idx, sizeof(int));
      outputMaxIdxFile.close();

      // Select 10% of the chunks to be set at random as the "test set"
      num_test_idxs = (int) ((double)max_idx * TEST_PERCENTAGE);
      test_idx.clear();
//#define TEST_RANDOM_SEED    1
      time_t seconds;
      seconds = time (NULL);
      srand((unsigned int)seconds);

      cout << "max_day: " << max_day << endl << "max_idx: " << max_idx <<
endl << "num_test_idxs: " << num_test_idxs << endl;
      cout << "seconds: " << seconds << endl;


      while(test_idx.size() < num_test_idxs)
//    while(test_idx.size() < num_test_idxs * 100)
      {
            int random_idx = (rand() % max_idx) + 1;  // chose a random idx
from 1 to max_idx

            // Check if it's already used
            bool used = false;
            for(unsigned int i = 0; i < test_idx.size(); ++i)
            {
                  if(random_idx == test_idx[i])
                  {
                        //cout << "collision at testslot: " << i << ", rand:
" << random_idx << ", entry: " << test_idx[i] << endl;
                        used = true;
                        break;
                  }
```

APPENDIX D (continued)

```
			}

			// Push on to test_idx list if unused
			if(!used)
				test_idx.push_back(random_idx);
		}

		for(unsigned int i = 0; i < test_idx.size(); ++i)
			cout << i << ":\t" << test_idx[i] << endl;

		// Output selected indexes in numerical order
		stringstream outfile_name;
		outfile_name << "../Data/output/TESTCASES.vec";
		cout << outfile_name.str() << endl;
		ofstream outputFile(outfile_name.str().c_str(), ios_base::binary);


		int num_output = 0;
		int last_used_idx = 0;
		while(num_output < num_test_idxs)
		{
			int next_lowest_idx = max_idx;
			for(unsigned int i = 0; i < test_idx.size(); ++i)
			{
				if(test_idx[i] > last_used_idx && test_idx[i] <
next_lowest_idx)
					next_lowest_idx = test_idx[i];
			}

			cout << "output_num: " << (num_output+1) << ",\tidx: " <<
next_lowest_idx << endl;
			outputFile.write((char*) &next_lowest_idx, sizeof(int));

			last_used_idx = next_lowest_idx;
			++num_output;
		}

		outputFile.close();

/*
		// Read test
//		unsigned int i = 0;
		for(unsigned int i = 0; i < NUM_SYMPTOMS; ++i)
		{
			ifstream inputTestFile(outfile_name.str().c_str(),
ios_base::binary);
			std::vector<int> read_test_vec;
			while(true)
			{
				int test_idx_entry;
				inputTestFile.read((char*) &test_idx_entry, sizeof(int));

				if(inputTestFile.eof())
					break;
```

APPENDIX D (continued)

```
                read_test_vec.push_back(test_idx_entry);

                cout << "E#: " << read_test_vec.size() << ", IDX: " <<
read_test_vec.back() << endl;
            }
            inputTestFile.close();
      }
*/
}
```

APPENDIX D (continued)

B. **Includes.h**

```
#include <stdio.h>
#include <time.h>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <fstream>
#include <cstdlib>
#include <unordered_map>

// Data structures
typedef struct
{
	int time;
	double value;
} DataEntry;

// Defines
#define CHUNKS_PER_DAY  8
#define TEST_PERCENTAGE 0.3f

// NOTE: End of ProcessData project
```

APPENDIX D (continued)

RunThesis.cpp

```
/**
 * \file RunTheis.cpp
 *
 * Main entry point for running the entire
 * thesis project simulation
 *
 * ---------------------------------------
 * Jonathan Lessner. 2012
 */

#include "./BPNN/BPNN.hpp"
//#include "./BPNN/InputNode.hpp"
#include "Includes.h"

using namespace std;

void RunTest();
void RunProgram(unsigned int init_generation);
bool TopRetainedIdentical(std::vector<Genome> test_generation);
void RunOptimizer(int exp_num, bool maximize);
void LoadNamesFromFile(std::string file_name, std::vector<string>
&name_vector);
void SelectTestIndexes();

//-------------------------------------------------------------------------

void main(int argc, char * argv[])
{
	string inputLine;

	//RunTest();
	//return;

	// Read user input
	cout << "Enter desired function, (r)un/(s)tats/(p)rintgen/(o)ptimizer:
";
	getline(cin, inputLine);

	// Select file process type
	if(inputLine == "r" || inputLine == "run")
	{
		// Read user input
		cout << "Enter last saved generation (enter 0 to begin fresh): ";
		getline(cin, inputLine);

		unsigned int gen_start = atoi(inputLine.c_str());
		if(gen_start == 0)
			cout << endl << "Starting fresh with randomized starting
generation." << endl;
		else
```

APPENDIX D (continued)

```
                  cout << endl << "Starting with saved victors from
generation # " << gen_start << "." << endl;

            cout << "Stop processing at any time by closing window." << endl;
            cout << "Processing stops automatically at generation " <<
MAX_GENERATION << "." << endl;
            cout << endl << "Hit <Enter> to begin processing." << endl;

            getline(cin, inputLine);

            RunProgram(gen_start);
      }
      else if(inputLine == "s" || inputLine == "stats")
      {
            // Read user input
            cout << "Enter experiment run # to generate stats for: ";
            getline(cin, inputLine);
            unsigned int exp_num = atoi(inputLine.c_str());
            std::vector<double> threshold, avg_MSE, avg_hl, avg_nodes,
avg_inputs, avg_symp_inputs, avg_exer_inputs, avg_medi_inputs,
avg_diet_inputs, avg_learn_rate, avg_learn_momentum, avg_training_passes;
            double denom = (double)NUM_RETAINED_MEMBERS;

            // Initialize output string
            stringstream stats_string;
            stats_string << setprecision(5);

            // Tracker for dynamic threshold tracking ...
if(!LINEAR_THRESHOLD)
            double last_best_MSE;

            // For each generation...
            for(unsigned int generation_num = 1; generation_num <
MAX_GENERATION+1; ++generation_num)
            {
                  double tot_MSE = 0.0f,
                           tot_hl = 0.0f,
                           tot_nodes = 0.0f,
                           tot_inputs = 0.0f,
                           tot_symp_inputs = 0.0f,
                           tot_exer_inputs = 0.0f,
                           tot_medi_inputs = 0.0f,
                           tot_diet_inputs = 0.0f,
                           tot_learn_rate = 0.0f,
                           tot_learn_momentum = 0.0f,
                           tot_training_passes = 0.0f;


                  // Load the saved top-5 for the submitted
experiment/generation
                  for(unsigned int gene_num = 0; gene_num <
NUM_RETAINED_MEMBERS; ++gene_num)
                  {
                        bool load_continue = true;
```

APPENDIX D (continued)

```
                        Genome member_genome;
                        load_continue =
member_genome.LoadFile(generation_num,gene_num+1,exp_num);

                        if(!load_continue)
                        {
                              cout << "Failed to load generation " <<
generation_num << ", member " << gene_num << ". Please try again." << endl;
                              return;
                        }
                        else
                        {
                              //member_genome.Print();
                              // Record total stats
                              tot_MSE += member_genome.getMSE();
                              tot_hl += (double)member_genome.getHLayers();
                              for(unsigned int i = 0; i <
member_genome.getHLayers(); ++i)
                                    tot_nodes += member_genome.getHLNodes(i);
                              tot_inputs += member_genome.getNumInputs();
                              for(unsigned int i = 0; i <
member_genome.getNumInputs(); ++i)
                              {
                                    if(member_genome.getInput(i).input_type
== InputType::Symptom)              tot_symp_inputs += 1.0f;
                                    else
if(member_genome.getInput(i).input_type == InputType::Exercise)
      tot_exer_inputs += 1.0f;
                                    else
if(member_genome.getInput(i).input_type == InputType::Medication)
      tot_medi_inputs += 1.0f;
                                    else
if(member_genome.getInput(i).input_type == InputType::Diet)
      tot_diet_inputs += 1.0f;
                              }
                              tot_learn_rate += member_genome.getLearnRate();
                              tot_learn_momentum +=
member_genome.getMomentum();
                              if(exp_num >= 14)
                                    tot_training_passes +=
member_genome.getTP();


                              if(!LINEAR_THRESHOLD)
                              {
                                    if(gene_num == 0)
                                    {
                                          // Calculate this generation's
training threshold
                                          if(generation_num == 1)

      threshold.push_back(MSE_TRAINING_THRESHOLD_ALPHA);
                                          else
```

APPENDIX D (continued)

```
      threshold.push_back(last_best_MSE * MSE_TRAINING_THRESHOLD_SQUEEZE);
                                          last_best_MSE =
member_genome.getMSE();
                                    }
                              }
                        }
                  }

                  if(LINEAR_THRESHOLD)
                  {
                        // Calculate this generation's training threshold
                        double mse_training_threshold_alpha,
mse_training_threshold_omega;
                        if(exp_num == 9 || exp_num == 10)
                        {
                              mse_training_threshold_alpha = 0.0005f;
                              mse_training_threshold_omega = 0.00005f;
                        }
                        else if(exp_num == 11 || exp_num == 12)
                        {
                              mse_training_threshold_alpha = 0.005f;
                              mse_training_threshold_omega = 0.0005f;
                        }
                        else if(exp_num == 13 || exp_num == 14)
                        {
                              mse_training_threshold_alpha = 0.05f;
                              mse_training_threshold_omega = 0.005f;
                        }
                        else  // default
                        {
                              mse_training_threshold_alpha = 0.05f;
                              mse_training_threshold_omega = 0.005f;
                        }

                        double range = mse_training_threshold_omega -
mse_training_threshold_alpha;
                        double threshold_percent = ((double) generation_num-
1) / ((double)MAX_GENERATION-1);
                        double training_threshold =
mse_training_threshold_alpha + range * threshold_percent;
                        threshold.push_back(training_threshold);
                        //cout << "Training Threshold:\t" <<
training_threshold << endl;
                  }


                  // Divide totals by num_retained per generation, and record
time series for each
                  avg_MSE.push_back(tot_MSE/denom);
                  avg_hl.push_back(tot_hl/denom);
                  avg_nodes.push_back(tot_nodes/denom);
                  avg_inputs.push_back(tot_inputs/denom);
                  avg_symp_inputs.push_back(tot_symp_inputs/denom);
                  avg_exer_inputs.push_back(tot_exer_inputs/denom);
```

APPENDIX D (continued)

```
                  avg_medi_inputs.push_back(tot_medi_inputs/denom);
                  avg_diet_inputs.push_back(tot_diet_inputs/denom);
                  avg_learn_rate.push_back(tot_learn_rate/denom);
                  avg_learn_momentum.push_back(tot_learn_momentum/denom);
                  avg_training_passes.push_back(tot_training_passes/denom);
            }


            // Output to CSV file
            for(unsigned int i = 0; i < MAX_GENERATION; ++i)
                  stats_string << threshold[i] << "," << avg_MSE[i] << "," <<
avg_hl[i] << "," << avg_nodes[i] << "," << avg_inputs[i] << "," <<
avg_symp_inputs[i] << "," << avg_exer_inputs[i] << "," << avg_medi_inputs[i]
<< "," << avg_diet_inputs[i] << "," << avg_learn_rate[i] << "," <<
avg_learn_momentum[i] << "," << avg_training_passes[i] << "\n";
            stringstream stats_file_name;
            stats_file_name << "..\\Genomes\\Run";
            if(exp_num < 10)
                  stats_file_name << "0";
            stats_file_name << exp_num;
            stats_file_name << "\\0 - Stats.txt";
            cout << "Filename: " << stats_file_name.str() << endl;

            // Open output file for writing
            ofstream outputStatsFile(stats_file_name.str().c_str());

            // Output report to file
            outputStatsFile << stats_string.str();
            outputStatsFile.close();

            // Print report to screen
            cout << endl << endl << endl;
            cout << stats_string.str();
      }
      else if(inputLine == "p" || inputLine == "printgen")
      {
            // Read user input
            cout << "Enter generation to print: ";
            getline(cin, inputLine);
            unsigned int gen_print = atoi(inputLine.c_str());

            // Load the saved top-5 for the submitted generation
            for(unsigned int gene_num = 0; gene_num < NUM_RETAINED_MEMBERS;
++gene_num)
            {
                  bool load_continue = true;
                  Genome member_genome;
                  load_continue =
member_genome.LoadFile(gen_print,gene_num+1);
                  if(!load_continue)
                  {
                        cout << "Failed to load generation " << gen_print <<
". Please try again." << endl;
                        return;
```

APPENDIX D (continued)

```
                  }
                  member_genome.Print();
            }
      }
      else if(inputLine == "o" || inputLine == "optimizer")
      {
            // Read user input
            cout << "Enter experiment run # to use: ";
            getline(cin, inputLine);
            unsigned int exp_num = atoi(inputLine.c_str());

            cout << "Choose (d)agnoistic/(p)rescriptive: ";
            getline(cin, inputLine);

            bool maximize;
            if(inputLine == "d" || inputLine == "dagnoistic")
                  maximize = true;
            else //if(inputLine == "p" || inputLine == "prescriptive")
                  maximize = false;

            RunOptimizer(exp_num,maximize);
/*
            // Load the saved top-5 for the submitted generation
            for(unsigned int gene_num = 0; gene_num < NUM_RETAINED_MEMBERS;
++gene_num)
            {
                  bool load_continue = true;
                  Genome member_genome;
                  load_continue =
member_genome.LoadFile(gen_print,gene_num+1);
                  if(!load_continue)
                  {
                        cout << "Failed to load generation " << gen_print <<
". Please try again." << endl;
                        return;
                  }
                  member_genome.Print();
            }
*/
      }

      cout << endl << endl << "Hit <enter> to finish" << endl;
      getline(cin, inputLine);

      return;
}

//--------------------------------------------------------------------------
-------------
// This is the place to run any code test at top level, so I don't have to
deal with any
// preload/savefile overhead
//--------------------------------------------------------------------------
-------------
```

APPENDIX D (continued)

```
void RunTest()
{
	string inputLine;

	double range;
	range = MSE_TRAINING_THRESHOLD_OMEGA - MSE_TRAINING_THRESHOLD_ALPHA;
	cout << range << endl;
	for(unsigned int gen = 1; gen <= MAX_GENERATION; ++gen)
	{
		double percent = ((double) gen-1) / ((double)MAX_GENERATION-1);
		double threshold = MSE_TRAINING_THRESHOLD_ALPHA + range *
percent;

		cout << gen << ":\t" << percent << ":\t" << threshold << endl;




	}



/*
	std::vector<Genome> this_generation, next_generation;

	// Load the saved top-5 for the submitted generation
	for(unsigned int gene_num = 0; gene_num < 5; ++gene_num)
	{
		bool load_continue = true;
		Genome member_genome;
		load_continue = member_genome.LoadFile(1,gene_num+1);
		this_generation.push_back(member_genome);
	}

	for(unsigned int i = 0; i < 5; ++i)
		this_generation[i].Print();

	cout << TopRetainedIdentical(this_generation) << endl;
*/
/*
	for(unsigned int gene_num = 0; gene_num < 5; ++gene_num)
	{
		BPNN member_BPNN;
		Genome member_genome;

		member_genome.InitRandom(1,gene_num+1);

		//cout << "\tscore: " << member_genome.getScore() << endl;
		this_generation.push_back(member_genome);
	}

	for(unsigned int gene_num = 0; gene_num < this_generation.size();
++gene_num)
```

APPENDIX D (continued)

```
	{
		cout << endl;
		this_generation[gene_num].Print();
		this_generation[gene_num].SaveFile(gene_num+1);
	}
	this_generation.clear();
		cout << endl;
		cout << endl;
	for(unsigned int gene_num = 0; gene_num < 5; ++gene_num)
	{
		Genome member_genome;

		cout << endl;
		member_genome.LoadFile(1,gene_num+1);
		member_genome.Print();
		this_generation.push_back(member_genome);
	}
*/
/*
	Genome new_genome;
	new_genome.InitMutant(0,0,this_generation[0]);

	for(unsigned int A = 0; A < 5; ++A)
	{
		for(unsigned int B = 0; B < 5; ++B)
		{
			if(A != B)
			{
				Genome new_genome;

	new_genome.InitHybrid(1,0,this_generation[A],this_generation[B]);
			}
		}
	}
*/
	cout << endl << endl << "Hit <enter> to finish" << endl;
	getline(cin, inputLine);
}

void RunProgram(unsigned int init_generation)
{
	string inputLine;
	std::vector<Genome> this_generation, next_generation;

	// Set random seed
	srand(time (NULL));

	//------------------------------------------------------------------------
------------
	// Step 0 - Generate first generation, or load selected generation from
save files
	//------------------------------------------------------------------------
------------
```

APPENDIX D (continued)

```
	if(init_generation == 0)
	{
		// Create the FIRST generation, completely from scratch
(spontaneous generation)
		for(unsigned int gene_num = 0; gene_num < NUM_GEN_MEMBERS;
++gene_num)
		{
			Genome member_genome;

			member_genome.InitRandom(1,gene_num+1);
			this_generation.push_back(member_genome);
		}
	}
	else
	{
		// Load the saved top-5 for the submitted generation
		for(unsigned int gene_num = 0; gene_num < NUM_RETAINED_MEMBERS;
++gene_num)
		{
			bool load_continue = true;
			Genome member_genome;
			load_continue =
member_genome.LoadFile(init_generation,gene_num+1);
			if(!load_continue)
			{
				cout << "Failed to load generation " <<
init_generation << ". Please try again." << endl;
				return;
			}
			this_generation.push_back(member_genome);
		}
	}

	while(true)
	{
		double training_threshold;

		//-----------------------------------------------------------------
------------------
		// Step 1 - Generate next generation (if not a fresh start)
		//-----------------------------------------------------------------
------------------
		if(this_generation.size() == NUM_RETAINED_MEMBERS)
		{
			unsigned int generation_num =
this_generation[0].getGenNum() + 1;

			if(generation_num > MAX_GENERATION)
				return;

			// Set this generation's training threshold based on last
generation's best scorer
			// New code (relative threshold tightening)
			if(!LINEAR_THRESHOLD)
```

APPENDIX D (continued)

```
                        training_threshold = this_generation[0].getMSE() *
MSE_TRAINING_THRESHOLD_SQUEEZE;

                  // Update the generation number and the member number of
retained members
                  for(unsigned int gene_num = 0; gene_num <
NUM_RETAINED_MEMBERS; ++gene_num)
                  {
                        this_generation[gene_num].setGenNum(generation_num);
                        this_generation[gene_num].setMemNum(gene_num+1);
                  }

                  // Reseed rand generator so that mutations don't repeat
over generations
                  srand(time (NULL));

                  if(TopRetainedIdentical(this_generation))
                  {
                        cout << "Mutations++" << endl;

                        // If the top retained members are all identical,
forego crossbreeding, and make members 2-25 mutations of member 1
                        next_generation.clear();
                        next_generation.push_back(this_generation[0]);
                        this_generation.clear();
                        this_generation.push_back(next_generation[0]);

                        cout << "mutating" << endl;
                        for(unsigned int mut_num = 1; mut_num < 25;
++mut_num)
                        {
                              Genome new_genome;

      new_genome.InitMutant(generation_num,mut_num+1,this_generation[0]);
                              this_generation.push_back(new_genome);
                        }
                  }
                  else
                  {
                        cout << "Crossbreeds++" << endl;


                        // Create the next (20) of the generation members by
GA (crossbreeding)
                        cout << "crossbreeding" << endl;
                        for(unsigned int A = 0; A < NUM_RETAINED_MEMBERS;
++A)
                        {
                              for(unsigned int B = 0; B <
NUM_RETAINED_MEMBERS; ++B)
                              {
                                    if(A != B)
                                    {
                                          Genome new_genome;
```

APPENDIX D (continued)

```
                                          unsigned int member_num =
(A*(NUM_RETAINED_MEMBERS-1)) + NUM_RETAINED_MEMBERS + 1;
                                          if(B < A)    member_num += B;
                                          else         member_num += B-1;

      new_genome.InitHybrid(generation_num,member_num,this_generation[A],this
_generation[B]);

      this_generation.push_back(new_genome);
                                    }
                              }
                        }
                  }

                  // Create the next (20) of the generation members by GA
(mutation)
                  cout << "mutating" << endl;
                  for(unsigned int A = 0; A < NUM_RETAINED_MEMBERS; ++A)
                  {
                        for(unsigned int mut_num = 0; mut_num <
(NUM_RETAINED_MEMBERS-1); ++mut_num)
                        {
                              Genome new_genome;
                              unsigned int member_num =
(A*(NUM_RETAINED_MEMBERS-1)) + mut_num +
(NUM_RETAINED_MEMBERS*NUM_RETAINED_MEMBERS) + 1;

      new_genome.InitMutant(generation_num,member_num,this_generation[A]);
                              this_generation.push_back(new_genome);
                        }
                  }

                  // Create the last (5) of the generation members by GA
(spontaneous generation)
                  cout << "randomizing" << endl;
                  for(unsigned int rand_num = 0; rand_num <
NUM_RETAINED_MEMBERS; ++rand_num)
                  {
                        Genome new_genome;
                        unsigned int member_num = NUM_GEN_MEMBERS -
NUM_RETAINED_MEMBERS + rand_num + 1;
                        new_genome.InitRandom(generation_num, member_num);
                        this_generation.push_back(new_genome);
                  }

                  // Shuffle generation indexes, to avoid Top 5 entrenchment
(in later generations)
                  std::vector<bool> genome_used;
                  next_generation.clear();      // using "next_generation"
vector as a convenient place for shuffling
                  for(unsigned int i = 0; i < this_generation.size(); ++i)
                  {
                        next_generation.push_back(this_generation[i]);
                        genome_used.push_back(false);
```

APPENDIX D (continued)

```
                  }

                  this_generation.clear();
                  for(unsigned int i = 0; i < next_generation.size(); ++i)
                  {
                        //cout << "Selecting genome #" << (i+1) << ":" <<
endl;

                        unsigned int random_range = next_generation.size() -
i;
                        unsigned int index_selected = rand() % random_range;
                        unsigned int index_found = 0;

                        //cout << "\trandom_range:\t" << random_range << endl
<< "\tindex_selected:\t" << index_selected << endl;

                        while(true)
                        {
                              if(genome_used[index_found])
                                    ++index_found;
                              else if(index_selected > 0)
                              {
                                    ++index_found;
                                    --index_selected;
                              }
                              else
                                    break;
                        }

                        //for(unsigned int j = 0; j < alphabet.size(); ++j)
                              //cout << j << ": " << alphabet[j] << ", " <<
letter_used[j] << endl;

                        //cout << "\tCorrelating letter:\t" <<
alphabet[index_found] << endl;


      this_generation.push_back(next_generation[index_found]);
                        genome_used[index_found] = true;

                        //cout << endl << endl;
                  }

                  // Reset the random seed for each member (fix for
entrenched starting weights)
                  for(unsigned int i = 0; i < this_generation.size(); ++i)
                        this_generation[i].setRandSeed((unsigned int)
rand());
            }
            else
                  // Set first generation's training threshold (arbitrary)
                  training_threshold = MSE_TRAINING_THRESHOLD_ALPHA;

            // Create new test vector cases if not static test-set
```

APPENDIX D (continued)

```
            if(!STATIC_TEST_SET)
                  SelectTestIndexes();


            //-----------------------------------------------------------------
------------------
            // Step 2 - Instantiate, train, test, score each member of the
current generation
            //-----------------------------------------------------------------
------------------

            // Old code (linear threshold tightening)
            if(LINEAR_THRESHOLD)
            {
                  unsigned int generation_num =
this_generation[0].getGenNum();
                  double range = MSE_TRAINING_THRESHOLD_OMEGA -
MSE_TRAINING_THRESHOLD_ALPHA;
                  double threshold_percent = ((double) generation_num-1) /
((double)MAX_GENERATION-1);
//                double training_threshold = MSE_TRAINING_THRESHOLD_ALPHA +
range * threshold_percent;
                  training_threshold = MSE_TRAINING_THRESHOLD_ALPHA + range *
threshold_percent;
            }

            cout << "Training Threshold:\t" << training_threshold << endl;
            for(unsigned int gene_num = 0; gene_num < this_generation.size();
++gene_num)
            {
                  BPNN member_BPNN;

                  member_BPNN.InitNetwork(&this_generation[gene_num]);
                  member_BPNN.RunTrainingSession(training_threshold);
                  member_BPNN.RunTest();

                  //this_generation[gene_num].Print();
                  //cout << "Member: " << gene_num << "\tscore: " <<
this_generation[gene_num].getScore() << endl;
            }

            //-----------------------------------------------------------------
------------------
            // Step 3 - Rank the member genomes by score
            //-----------------------------------------------------------------
------------------

            //  Set up saved registry
            std::vector<bool> saved;
            for(unsigned int gene_num = 0; gene_num < NUM_GEN_MEMBERS;
++gene_num)
                  saved.push_back(false);

            //  Save the top NUM_RETAINED_MEMBERS
```

APPENDIX D (continued)

```
            next_generation.clear();
            for(unsigned int num_retained = 0; num_retained <
NUM_RETAINED_MEMBERS; ++num_retained)
            {
                  // Find first entry that hasn't already been saved
                  unsigned int top_entry = 0;
                  for(unsigned int gene_num = 0; gene_num < NUM_GEN_MEMBERS;
++gene_num)
                  {
                        if(!saved[top_entry])
                              break;
                        else
                              ++top_entry;
                  }

                  // Find the highest unsaved score
                  for(unsigned int gene_num = 0; gene_num < NUM_GEN_MEMBERS;
++gene_num)
                  {
                        //cout << "Member: " << gene_num << "\tscore: " <<
this_generation[gene_num].getScore() << endl;

                        if(!saved[gene_num] &&
(this_generation[gene_num].getScore() >
this_generation[top_entry].getScore()))
                        {
                              top_entry = gene_num;
                              //cout << "new top entry: " << top_entry <<
"\ttop score: " << this_generation[top_entry].getScore() << endl;
                        }
                  }

                  // Save the found top unsaved entry, save it into the next
generation, and mark it as saved
                  next_generation.push_back(this_generation[top_entry]);
                  saved[top_entry] = true;
            }

            // Print out top selected members
            cout << "Next generation first 5 entries:" << endl;
            for(unsigned int gene_num = 0; gene_num < next_generation.size();
++gene_num)
                  cout << "\t" << gene_num << ", score: " <<
next_generation[gene_num].getScore() << ", MSE: " <<
next_generation[gene_num].getMSE() << endl;


            // Purge this generation, move saved members to start new
generation, and write each saved genome to file
            this_generation.clear();
            for(unsigned int num_retained = 0; num_retained <
NUM_RETAINED_MEMBERS; ++num_retained)
            {
                  this_generation.push_back(next_generation[num_retained]);
```

APPENDIX D (continued)

```
                  this_generation[num_retained].Print();
                  this_generation[num_retained].SaveFile(num_retained+1);
            }
      }

      cout << endl << endl << "Hit <enter> to finish" << endl;
      getline(cin, inputLine);
}

bool TopRetainedIdentical(std::vector<Genome> test_generation)
{
      // Compare member 0 to each other member
      for(unsigned int i = 1; i < test_generation.size(); ++i)
      {
            //if(test_generation[0].getRandSeed() !=
test_generation[i].getRandSeed())   return false;
            if(test_generation[0].getLearnRate() !=
test_generation[i].getLearnRate())  return false;
            if(test_generation[0].getMomentum() !=
test_generation[i].getMomentum())   return false;

            if(test_generation[0].getHLayers() !=
test_generation[i].getHLayers())          return false;
            for(unsigned int j = 0; j < test_generation[0].getHLayers(); ++j)
                  if(test_generation[0].getHLNodes(j) !=
test_generation[i].getHLNodes(j))return false;

            if(test_generation[0].getNumInputs() !=
test_generation[i].getNumInputs())  return false;
            for(unsigned int j = 0; j < test_generation[0].getNumInputs();
++j)
            {
                  if(test_generation[0].getInput(j).input_type !=
test_generation[i].getInput(j).input_type)return false;
                  if(test_generation[0].getInput(j).input_idx !=
test_generation[i].getInput(j).input_idx)return false;
                  if(test_generation[0].getInput(j).data_front !=
test_generation[i].getInput(j).data_front)return false;
                  if(test_generation[0].getInput(j).data_back !=
test_generation[i].getInput(j).data_back)return false;
            }
      }

      return true;
}



//-------------------------------------------------------------------------
--------------------------------
// Optimizer program
//
// For each retained genome (top 5) in final generation (1000) of selected
experimental run (run_num):
```

APPENDIX D (continued)

```
//    - Initializes and trains BPNN from retained genome
//    - Exhaustively tests binary activation of all input nodes, and searches
for which activation/deactivation
//      pattern results in maximum (diagnostic) or minimum (prescriptive)
expression of output (symptoms)
//
// Args: int exp_num - experiment number from which to retrieve winning
genomes
//           bool maximize - true: maximize output activation (run diagnostic
function)
//                               - false: minimize output expression (run
prescriptive function)
//-------------------------------------------------------------------------
---------------------------------

void RunOptimizer(int exp_num, bool maximize)
{
      cout << "Using experiment #" << exp_num << endl;
      if(maximize)      cout << "Diagnostic calculation:\n\tSearching for
maximized output firing\n\t(symbolizes maximum symptomatic expression)" <<
endl;
      else              cout << "Prescriptive calculation:\n\tSearching for
minimized output firing\n\t(symbolizes minimum symptomatic expression)" <<
endl;

      std::vector<Genome> optimizer_members;

      // Load all symptom/exercise/medication/diet names
      std::vector<string> symp_name;
      std::vector<string> exer_name;
      std::vector<string> medi_name;
      std::vector<string> diet_name;

      for(unsigned int i = 0; i < 8; ++i)
            symp_name.push_back(SYMP_NAME[i]);

      LoadNamesFromFile("../Data/EXER_CAT.txt", exer_name);
      LoadNamesFromFile("../Data/MEDI_CAT.txt", medi_name);
      LoadNamesFromFile("../Data/DIET_CAT.txt", diet_name);
/*
      // Print tests
      cout << "Symptoms: " << endl;
      for(unsigned int i = 0; i < symp_name.size(); ++i)
            cout << "\t" << symp_name[i] << endl;
      cout << "Exercise Types: " << endl;
      for(unsigned int i = 0; i < exer_name.size(); ++i)
            cout << "\t" <<  exer_name[i] << endl;
      cout << "Medication: " << endl;
      for(unsigned int i = 0; i < medi_name.size(); ++i)
            cout << "\t" <<  medi_name[i] << endl;
      cout << "Diet Ingredients: " << endl;
      for(unsigned int i = 0; i < diet_name.size(); ++i)
            cout << "\t" <<  diet_name[i] << endl;
*/
```

APPENDIX D (continued)

```
	// Load the saved top-5 for the submitted experiment/generation
	for(unsigned int gene_num = 0; gene_num < NUM_RETAINED_MEMBERS;
++gene_num)
	{
		bool load_continue = true;
		Genome member_genome;
		load_continue =
member_genome.LoadFile(1000,gene_num+1,exp_num,(exp_num>14));

		if(!load_continue)
		{
			cout << "Failed to load member " << gene_num << ". Please
try again." << endl;
			optimizer_members.clear();
			return;
		}
		else
		{
			member_genome.Print();
			optimizer_members.push_back(member_genome);
		}
	}

	double training_threshold;
	if(exp_num < 14)
	{
		unsigned int generation_num = 1000;
		double range = MSE_TRAINING_THRESHOLD_OMEGA -
MSE_TRAINING_THRESHOLD_ALPHA;
		double threshold_percent = ((double) generation_num-1) /
((double)MAX_GENERATION-1);
//		double training_threshold = MSE_TRAINING_THRESHOLD_ALPHA + range
* threshold_percent;
		training_threshold = MSE_TRAINING_THRESHOLD_ALPHA + range *
threshold_percent;
	}
	else
		training_threshold = optimizer_members[0].getMSE() *
MSE_TRAINING_THRESHOLD_SQUEEZE;
//	for(unsigned int gene_num = 0; gene_num < optimizer_members.size();
++gene_num)

	for(unsigned int gene_num = 0; gene_num < 1; ++gene_num)
	{
		//-------------------------------------------------------------
------------------
		// Step 1 - Build and train BPNN
		//-------------------------------------------------------------
------------------
		BPNN member_BPNN;

		member_BPNN.InitNetwork(&optimizer_members[gene_num]);
		member_BPNN.RunTrainingSession(training_threshold);
```

APPENDIX D (continued)

```
            std::vector<double> target_output_vector;
            for(unsigned int i = 0; i < SYMP_OUTPUTS; ++i)
            {
                  if(maximize)      target_output_vector.push_back(1.0f);
                  else              target_output_vector.push_back(0.0f);
            }


            //-----------------------------------------------------------------
------------------
            // Step 2 - Create exhaustive trinary (0/.5/1) input Activations
and find MSEs for each
            //-----------------------------------------------------------------
------------------
            // New way
            OptimizerSequence lowest;
            lowest.MSE = 1.0f;
            // Old way
//          std::vector<OptimizerSequence> sequence_list;

            unsigned int num_time_inputs, num_data_inputs;
            num_time_inputs = 3;
            num_data_inputs = member_BPNN.getNumInputNodes() -
num_time_inputs;
            //std::cout << "num_time_inputs: " << num_time_inputs << ",
num_data_inputs: " << num_data_inputs << std::endl;

            // NOTE: for each permutation of the data inputs, the MSE scores
will be averaged from ALL of the possible permutations of time-of-day
            //          (e.g. if there are 5 data inputs and 3 time-of-day
inputs, and the optimizer is currently testing permutation 0/1/0/0/1, the
            //             subloop will find the MSE for ALL time-of-day
inputs (0/0/0 through 1/1/1), then the data loop will average the MSEs for
those
            //             tests, and the avg MSE for this data permutation
will be stored in the OptimizerSequence vector)


            // Cycle through the data
            std::vector<double> data_cycle;
            for(unsigned int i = 0; i < num_data_inputs; ++i)
                  data_cycle.push_back(0);
            unsigned int num_data_steps = 1;
            for(unsigned int i = 0; i < num_data_inputs; ++i)
                  //num_data_steps *= 2;        // binary input testing (0/1)
                  num_data_steps *= 3;          // trinary input testing
(0/.5/1)

            // For debugging only
//          double current_lowest_MSE = 1.0f;

            cout << "num data steps: " << num_data_steps << endl;
```

APPENDIX D (continued)

```
		for(unsigned int data_step = 0; data_step < num_data_steps;
++data_step)
		{
			//if(false)
			if(data_step%10000 == 0)
			{
				// Print current data permutation
				cout << std::fixed;
				cout << "data step: " << data_step << ", sequence: ";
				for(unsigned int i = 0; i < data_cycle.size(); ++i)
					cout << setprecision(2) << data_cycle[i] << "
";
				//cout << endl;
			}

			// Cycle through times
			double avg_MSE, total_MSE = 0;
			std::vector<int> time_cycle;
			for(unsigned int i = 0; i < num_time_inputs; ++i)
				time_cycle.push_back(0);
			unsigned int num_time_steps = 1 << num_time_inputs;
			//std::cout << "num time steps: " << num_time_steps <<
std::endl;
			for(unsigned int time_step = 0; time_step < num_time_steps;
++time_step)
			{
				/*
				// Print current time permutation
				std::cout << "\ttime step: " << time_step <<
std::endl << "\t";
				for(unsigned int i = 0; i < time_cycle.size(); ++i)
					std::cout << time_cycle[i] << " ";
				std::cout << std::endl;
				*/

				// Create input vector by combining the time_cycle
and data_cycle
				std::vector<double> input_vector;
				for(unsigned int i = 0; i < num_time_inputs; ++i)
					input_vector.push_back(time_cycle[i]);
				for(unsigned int i = 0; i < num_data_inputs; ++i)
					input_vector.push_back(data_cycle[i]);
				//for(unsigned int i = 0; i < input_vector.size();
++i)
				//	std::cout << "\t" << input_vector[i];
				//std::cout << std::endl;

				// Do processing for this time permutation
				member_BPNN.setOptimizerInputs(input_vector);
				member_BPNN.RunPropagation();

				// Get SEs, and calculate MSE
				member_BPNN.calcOptimizerMSE(target_output_vector);
				//cout << "MSE: " << member_BPNN.getMSE() << endl;
```

APPENDIX D (continued)

```
                        total_MSE += member_BPNN.getMSE();


                        // Increment counter on the time cycle for next
permutation
                        if(time_step < num_time_steps-1)
                        {
                              int current_time_idx = (int)time_cycle.size() -
1;
                              time_cycle[current_time_idx] += 1;
                              while(time_cycle[current_time_idx] > 1 &&
current_time_idx >= 0)
                              {
                                    time_cycle[current_time_idx] = 0;
                                    --current_time_idx;
                                    if(current_time_idx >= 0)
                                          time_cycle[current_time_idx] += 1;
                                    //std::cout << "current_time_idx: " <<
current_time_idx << std::endl;
                              }
                        }
                  }

                  // Average the MSEs and record the permutation and avg MSE
in the OptimizerSequence vector
                  avg_MSE = total_MSE / num_time_steps;
                  //if(false)
                  if(data_step%10000 == 0)
                        cout << setprecision(6) << ", Avg MSE: " << avg_MSE
<< endl;
                  // Old way
/*
                  OptimizerSequence new_sequence;
                  new_sequence.activation_list = data_cycle;
                  new_sequence.MSE = avg_MSE;
                  sequence_list.push_back(new_sequence);
*/
                  // New way
//                if(avg_MSE < current_lowest_MSE)
                  if(avg_MSE < lowest.MSE)
                  {

                        lowest.activation_list = data_cycle;
                        lowest.MSE = avg_MSE;

//                      current_lowest_MSE = avg_MSE;
                        cout << "New lowest MSE: " << avg_MSE << " at data
step " << data_step << endl;
                  }

                  // Increment counter on the data cycle for next permutation
                  if(data_step < num_data_steps-1)
                  {
                        int current_data_idx = (int)data_cycle.size() - 1;
```

APPENDIX D (continued)

```
                        //data_cycle[current_data_idx] += 1;      // binary
input testing (0/1)
                        data_cycle[current_data_idx] += 0.5;      // trinary
input testing (0/.5/1)
                        while(data_cycle[current_data_idx] > 1 &&
current_data_idx >= 0)
                        {
                              data_cycle[current_data_idx] = 0;
                              --current_data_idx;
                              if(current_data_idx >= 0)
                                    //data_cycle[current_data_idx] += 1;
      // binary input testing (0/1)
                                    data_cycle[current_data_idx] += 0.5;
      // trinary input testing (0/.5/1)
                              //cout << "current_data_idx: " <<
current_data_idx << endl;
                        }
                  }

/*
                  unsigned int time_steps = 1 << num_time_inputs;
                  std::cout << "time steps: " << time_steps << std::endl;
*/
            }

            //-----------------------------------------------------------------
------------------
            // Step 4 - Find lowest-avg-MSE sequence (the one that most
closely matches the target output)
            //-----------------------------------------------------------------
------------------
            // Old way
/*
            unsigned int low_MSE_idx = 0;
            for(unsigned int i = 1; i < sequence_list.size(); ++i)
            {
                  if(sequence_list[i].MSE < sequence_list[low_MSE_idx].MSE)
                        low_MSE_idx = i;
            }
            cout << "Lowest-MSE idx: " << sequence_list[low_MSE_idx].MSE <<
", idx: " << low_MSE_idx << endl;
*/
            // New way
            // no need to do anything here.  Lowest pattern and MSE is
tracked in real-time above


            //-----------------------------------------------------------------
------------------
            // Step 3 - Output the report to a file and then print to screen
            //-----------------------------------------------------------------
------------------

            stringstream report;
```

APPENDIX D (continued)

```
            report << "-------------------------------------------------------
------------------------" << endl;
            if(maximize)
                  report << "                                Diagnostic Report
" << endl
                        << "-------------------------------------------------
---------------------------" << endl
                          << "Sypmtoms are maximally expressed (undesirable)
when:" << endl;
            else
                  report << "                               Prescriptive Report
" << endl
                        << "-------------------------------------------------
---------------------------" << endl
                          << "Sypmtoms are minimally expressed (desirable)
when:" << endl;
            //optimizer_members[gene_num].Print();
            for(unsigned int input_num = num_time_inputs; input_num <
member_BPNN.getNumInputNodes(); ++input_num)
            {
                  report << "\t";
                  // Type
                  if(member_BPNN.getInputType(input_num) ==
InputType::Symptom)                  report <<
symp_name[member_BPNN.getInputIdx(input_num)];
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Exercise)           report <<
exer_name[member_BPNN.getInputIdx(input_num)];
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Medication)  report <<
medi_name[member_BPNN.getInputIdx(input_num)];
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Diet)               report <<
diet_name[member_BPNN.getInputIdx(input_num)];

                  report << " is aproximately ";

                  // Quantity
                  // Old way
//                  report <<
(sequence_list[low_MSE_idx].activation_list[input_num - num_time_inputs] *
member_BPNN.getMaxInput(input_num));
                  // New Way
                  report << (lowest.activation_list[input_num -
num_time_inputs] * member_BPNN.getMaxInput(input_num));

                  // Units
                  if(member_BPNN.getInputType(input_num) ==
InputType::Symptom)                  report << " on the symptom rating scale";
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Exercise)           report << " minutes*intensity";
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Medication)  report << " mgs";
```

APPENDIX D (continued)

```
                else if(member_BPNN.getInputType(input_num) ==
InputType::Diet)              report << " oz";

                // Express in per-day terms
                // Old way
//              report << " (aprox. " <<
(((sequence_list[low_MSE_idx].activation_list[input_num - num_time_inputs] *
member_BPNN.getMaxInput(input_num)) / (member_BPNN.getFrontRange(input_num) -
member_BPNN.getBackRange(input_num))) * 24);
                // New way
                report << " (aprox. " <<
(((lowest.activation_list[input_num - num_time_inputs] *
member_BPNN.getMaxInput(input_num)) / (member_BPNN.getFrontRange(input_num) -
member_BPNN.getBackRange(input_num))) * 24);
                report << " per day)";

                // Expression/Exertion/Dosage/Ingestion time range
                report << " between " <<
member_BPNN.getFrontRange(input_num) << " and " <<
member_BPNN.getBackRange(input_num) << " hours before target time.";


                report << endl;
        }

        stringstream report_file_name;
        report_file_name << "..\\Genomes\\Run";
        if(exp_num < 10)
                report_file_name << "0";
        report_file_name << exp_num;
        report_file_name << "\\";
        report_file_name << gene_num;
        if(maximize)      report_file_name << " - Diagnostic.txt";
        else              report_file_name << " - Prescriptive.txt";
        cout << "Filename: " << report_file_name.str() << endl;

        // Open output file for writing
        ofstream outputReportFile(report_file_name.str().c_str());

        // Output report to file
        outputReportFile << report.str();
        outputReportFile.close();

        // Print report to screen
        cout << endl << endl << endl;
        cout << report.str();
/*
        // Print report
        cout << endl << endl << endl;
        cout << "------------------------------------------------------------
----------------------" << endl;
        if(maximize)
                cout << "                                Diagnostic Report
" << endl
```

APPENDIX D (continued)

```
                        << "-------------------------------------------------
--------------------------" << endl
                          << "Sypmtoms are maximally expressed (undesirable)
when:" << endl;
            else
                  cout << "                           Prescriptive Report
" << endl
                        << "-------------------------------------------------
--------------------------" << endl
                          << "Sypmtoms are minimally expressed (desirable)
when:" << endl;
            //optimizer_members[gene_num].Print();
            for(unsigned int input_num = num_time_inputs; input_num <
member_BPNN.getNumInputNodes(); ++input_num)
            {
                  cout << "\t";
                  // Type
                  if(member_BPNN.getInputType(input_num) ==
InputType::Symptom)                cout <<
symp_name[member_BPNN.getInputIdx(input_num)];
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Exercise)          cout <<
exer_name[member_BPNN.getInputIdx(input_num)];
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Medication)  cout <<
medi_name[member_BPNN.getInputIdx(input_num)];
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Diet)              cout <<
diet_name[member_BPNN.getInputIdx(input_num)];

                  cout << " is aproximately ";

                  // Quantity
                  cout <<
(sequence_list[low_MSE_idx].activation_list[input_num - num_time_inputs] *
member_BPNN.getMaxInput(input_num));

                  // Units
                  if(member_BPNN.getInputType(input_num) ==
InputType::Symptom)                cout << " on the symptom rating scale";
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Exercise)          cout << " minutes*intensity";
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Medication)  cout << " mgs";
                  else if(member_BPNN.getInputType(input_num) ==
InputType::Diet)              cout << " oz";

                  // Expression/Exertion/Dosage/Ingestion time range
                  cout << " between " << member_BPNN.getFrontRange(input_num)
<< " and " << member_BPNN.getBackRange(input_num) << " hours before target
time.";

                  cout << endl;
            }
```

APPENDIX D (continued)

```
*/

            // Clear the sequence list

            // Old way
//          sequence_list.clear();
      }


      // Purge memory and exit
      optimizer_members.clear();
};

//---------------------------------------------------------------------------
--------------------------------
// LoadNamesFromFile
//---------------------------------------------------------------------------
--------------------------------

void LoadNamesFromFile(std::string file_name, std::vector<string>
&name_vector)
{
      ifstream inputNameFile(file_name.c_str());

      if(!inputNameFile.is_open())
      {
            cout << "File open failed!" << endl;
            return;
      }

      while(true)
      {
            string inputLine;
            getline(inputNameFile, inputLine);
            //cout << inputLine << endl;

            if(inputNameFile.eof())
                  break;

            name_vector.push_back(inputLine);
      }
      inputNameFile.close();
      //std::cout << name_vector.size() << std::endl;
}

void SelectTestIndexes(/*string file_name*/)
{
      cout << "Selecting Test Indexes:" << endl;

      // Load max index from input file
      int max_idx, num_test_idxs;
      ifstream inputFile("..\\Data\\MAXINDEX.vec");

      if(!inputFile.is_open())
```

APPENDIX D (continued)

```
	{
		cout << "File open failed!" << endl;
	}

	inputFile.read((char*) &max_idx, sizeof(int));

	inputFile.close();


	// Select 10% of the chunks to be set at random as the "test set"
	std::vector<int> test_idx;
	num_test_idxs = (int) ((double)max_idx * TEST_PERCENTAGE);
	test_idx.clear();
	time_t seconds;
	seconds = time (NULL);
	srand((unsigned int)seconds);

	//cout << "max_idx: " << max_idx << endl << "num_test_idxs: " <<
num_test_idxs << endl;
	//cout << "seconds: " << seconds << endl;


	while(test_idx.size() < num_test_idxs)
	{
		int random_idx = (rand() % max_idx) + 1;  // chose a random idx
from 1 to max_idx

		// Check if it's already used
		bool used = false;
		for(unsigned int i = 0; i < test_idx.size(); ++i)
		{
			if(random_idx == test_idx[i])
			{
				//cout << "collision at testslot: " << i << ", rand:
" << random_idx << ", entry: " << test_idx[i] << endl;
				used = true;
				break;
			}
		}

		// Push on to test_idx list if unused
		if(!used)
			test_idx.push_back(random_idx);
	}

	//for(unsigned int i = 0; i < test_idx.size(); ++i)
		//cout << i << ":\t" << test_idx[i] << endl;

	// Output selected indexes in numerical order
	stringstream outfile_name;
	outfile_name << "../Data/TESTCASES_D.vec";
	//cout << outfile_name.str() << endl;
	ofstream outputFile(outfile_name.str().c_str(), ios_base::binary);
```

APPENDIX D (continued)

```
	int num_output = 0;
	int last_used_idx = 0;
	while(num_output < num_test_idxs)
	{
		int next_lowest_idx = max_idx;
		for(unsigned int i = 0; i < test_idx.size(); ++i)
		{
			if(test_idx[i] > last_used_idx && test_idx[i] <
next_lowest_idx)
				next_lowest_idx = test_idx[i];
		}

		//cout << "output_num: " << (num_output+1) << ",\tidx: " <<
next_lowest_idx << endl;
		outputFile.write((char*) &next_lowest_idx, sizeof(int));

		last_used_idx = next_lowest_idx;
		++num_output;
	}

	outputFile.close();

/*
	// Read test
//	unsigned int i = 0;
	ifstream inputTestFile(outfile_name.str().c_str(), ios_base::binary);
	std::vector<int> read_test_vec;
	while(true)
	{
		int test_idx_entry;
		inputTestFile.read((char*) &test_idx_entry, sizeof(int));

		if(inputTestFile.eof())
			break;

		read_test_vec.push_back(test_idx_entry);

		cout << "E#: " << read_test_vec.size() << ", IDX: " <<
read_test_vec.back() << endl;
	}
	inputTestFile.close();
*/
}
```

APPENDIX D (continued)

Includes.h

```
#ifndef INCLUDES_INCLUDED
#define INCLUDES_INCLUDED

#include <stdio.h>
#include <math.h>
#include <time.h>
#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>
#include <vector>
#include <fstream>
#include <cstdlib>
#include <unordered_map>

// Enums
namespace InputType
{
	enum Type
	{
		Symptom,
		Exercise,
		Medication,
		Diet,
		TimeOfDay,
	} ;
}

namespace MutationType
{
	enum Type
	{
		RandomSeedReset,
		LearnRateIncrease,
		LearnRateDecrease,
		LearnRateRandom,
		MomentumIncrease,
		MomentumDecrease,
		MomentumRandom,

		HiddenLayerAdd,
		HiddenLayerRemove,
		HiddenNodeAdd,
		HiddenNodeRemove,

		InputRemove,	// Remove is enumed before Add because of data
processing reasons (see Genome::InitMutant())
		InputAdd,
		InputReplace,
		InputRangeFrontIncrease,	// Moves further back in time
		InputRangeFrontDecrease,	// Moves closer forward in time
```

APPENDIX D (continued)

```
            InputRangeBackIncrease,       // Moves further back in time
            InputRangeBackDecrease,       // Moves closer forward in time
            InputRangeRandom,

            NUM_MUTATION_TYPES,
      };
}

// Data structures
typedef struct
{
      int time;
      double value;
} DataEntry;

typedef struct
{
      InputType::Type input_type;
      unsigned int input_idx, data_front, data_back; // sample range is
indicated in # of hours preceding sample time index: (sample_hour -
data_front, sample_hour - data_back)
} InputChromosome;

typedef struct
{
      std::vector<double> activation_list;
      double MSE;
} OptimizerSequence;

static const std::string SYMP_NAME[8] =
{
      "BM",
      "blood",
      "mucus",
      "pus",
      "gas",
      "urgency",
      "cramping",
      "diarhea"
};
#endif



// Defines
#define CHUNK_SIZE                3
#define CHUNKS_PER_DAY        (24/CHUNK_SIZE)
#define MAX_GENERATION        1000

#define     SYMP_OUTPUTS 8

#define NUM_GEN_MEMBERS             50
#define NUM_RETAINED_MEMBERS  5
```

APPENDIX D (continued)

```
#define MAX_LEAD_TIME                (2*7*24)    // 2 weeks

#define MSE_TRAINING_THRESHOLD      0.0005f

/*
#define MSE_TRAINING_THRESHOLD_ALPHA      0.0005f
#define MSE_TRAINING_THRESHOLD_OMEGA      0.00005f
*/


#define LINEAR_THRESHOLD      true
#define MSE_TRAINING_THRESHOLD_ALPHA      0.05f
#define MSE_TRAINING_THRESHOLD_SQUEEZE    0.95f
#define MSE_TRAINING_THRESHOLD_OMEGA      0.005f

#define STATIC_TEST_SET       true
#define TEST_PERCENTAGE 0.1f
#define TEST_SET_PERCENT      10.0f
//#define TEST_PERCENTAGE     0.2f
//#define TEST_SET_PERCENT    20.0f
//#define TEST_PERCENTAGE           0.3f
//#define TEST_SET_PERCENT    30.0f
```

APPENDIX D (continued)

Genome.hpp

```
/*****************************************
* Genome.hpp
*
* Genome for initializing a BPNN
*
* Author: Jon Lessner 2012
*****************************************/

#ifndef GENOME_HPP_INCLUDED
#define GENOME_HPP_INCLUDED

#include "Genome.hpp"
#include "../Includes.h"

class Genome
{

      public:

            Genome();

            ~Genome();

            void InitRandom(unsigned int gen_num, unsigned int mem_num);
            void InitHybrid(unsigned int gen_num, unsigned int mem_num,
Genome gene_A, Genome gene_B);
            void InitMutant(unsigned int gen_num, unsigned int mem_num,
Genome gene_A);


            void SaveFile(unsigned int slot_num, bool with_training_passes =
true);
            bool LoadFile(unsigned int gen_num, unsigned int mem_num,
unsigned int experiment_num = -1, bool with_training_passes = true);
            void Print();

            unsigned int getGenNum()                  { return
generation_num; }
            void setGenNum(unsigned int gen_num){ generation_num = gen_num;}
            unsigned int getMemNum()                  { return member_num; }
            void setMemNum(unsigned int mem_num){ member_num = mem_num; }
            unsigned int getRandSeed()                { return rand_seed; }
            void setRandSeed(unsigned int seed) { rand_seed = seed; }
            double getLearnRate()                     { return learn_rate; }
            double getMomentum()                      { return momentum; }
            unsigned int getHLayers()                 { return
hl_nodes.size(); }
            int getHLNodes(unsigned int layer)  { return hl_nodes[layer]; }
            unsigned int getNumInputs()               { return
input_node.size(); }
```

APPENDIX D (continued)

```
            InputChromosome getInput(unsigned int index)    { return
input_node[index]; }

            void setMSE(double gen_MSE)               { avg_MSE = gen_MSE; }
            double getMSE()                                 { return avg_MSE;
}
            void setScore(double gen_score)           { score = gen_score; }
            double getScore()                         { return score; }
            unsigned int getTP()                      { return
training_passes; }
            void setTP(unsigned int TP)               { training_passes = TP;
}

      protected:
            unsigned int generation_num, member_num;
            //time_t rand_seed;
            unsigned int rand_seed, training_passes;
            double learn_rate, momentum;
            double avg_MSE, score;
            std::vector<int> hl_nodes;

            std::vector<InputChromosome> input_node;

      }; // end class Genome

#endif // end GENOME_HPP_INCLUDED
```

APPENDIX D (continued)

Genome.cpp

```
/**
* Genome.cpp
*
* Implementation of the Genome class.
* Genome for initializing a BPNN
*
* Author: Jon Lessner 2012
*/

#include "Genome.hpp"

using namespace std;

// Constants used for managing minimum and maximum configuration parameters
#define MIN_HLAYERS     1
#define MAX_HLAYERS     5
#define MIN_HLNODES     2
#define MAX_HLNODES     10

#define MIN_LEARNRATE   0.0f
#define MAX_LEARNRATE   0.5f
#define MIN_MOMENTUM    0.0f
#define MAX_MOMENTUM    1.0f

#define MAX_INPUTS            20
#define MIN_INPUTS            5

//------------------------------------------------------------------------

Genome::Genome()
//:more_inits(INIT_VAL)
{

}

//------------------------------------------------------------------------

Genome::~Genome()
{
      //cout << "Genome deleted: " << endl;
}

//------------------------------------------------------------------------

void Genome::InitRandom(unsigned int gen_num, unsigned int mem_num)
{
      generation_num = gen_num;
      member_num = mem_num;
      score = -1.0f;
```

APPENDIX D (continued)

```
	//cout << "Initializing random Genome:\tGen: " << generation_num << ",
Mem: " << member_num << endl;


	//-------------------------------------------------------------------
--
	// Genome portion A - body and constants
	//-------------------------------------------------------------------
--

	// Set the random seed, learning rate and momentum rate
	//rand_seed = time (NULL);
	rand_seed = rand();
	srand(rand_seed);
	learn_rate = (double(rand()) / double(RAND_MAX)) * (MAX_LEARNRATE -
MIN_LEARNRATE) + MIN_LEARNRATE;
	momentum = (double(rand()) / double(RAND_MAX)) * (MAX_MOMENTUM -
MIN_MOMENTUM) + MIN_MOMENTUM;

	// Configure the body of the BPNN
	hl_nodes.clear();
	int num_layers = (rand() % (MAX_HLAYERS - MIN_HLAYERS + 1)) +
MIN_HLAYERS;
	for(int i = 0; i < num_layers; ++i)
		hl_nodes.push_back((rand() % (MAX_HLNODES - MIN_HLNODES + 1)) +
MIN_HLNODES);


	//-------------------------------------------------------------------
--
	// Genome portion B - inputs
	//-------------------------------------------------------------------
--

	// Fetch and calculate max data time (hour)
/*
	stringstream test_idx_file_name;
	test_idx_file_name << "..\\Data\\TESTCASES.vec";
	ifstream inputTestFile(test_idx_file_name.str().c_str(),
ios_base::binary);
	std::vector<int> test_idx;
	while(true)
	{
		int test_idx_entry;
		inputTestFile.read((char*) &test_idx_entry, sizeof(int));

		if(inputTestFile.eof())
			break;

		test_idx.push_back(test_idx_entry);
	}
	inputTestFile.close();
	unsigned int max_training_idx = (unsigned int) (test_idx.size() *
(100.0f/TEST_SET_PERCENT));
```

APPENDIX D (continued)

```
	unsigned int max_data_range = max_training_idx * CHUNK_SIZE;
*/
	unsigned int max_data_range = MAX_LEAD_TIME;
	//cout << "Max data range: " << max_data_hour << endl;

	// Fetch max type indexes
	std::vector<unsigned int> cat_num;
	for(unsigned int i = 0; i < 4; ++i) cat_num.push_back(0);
	for(unsigned int i = 0; i < 4; ++i)
	{
		if(i == 0)
		{
			cat_num[i] = 8; // we don't need to count the number of
symptoms. there are always 8
			continue;
		}

		ifstream inputCatFile;
		     if(i == 1)   inputCatFile.open("..\\Data\\EXER_CAT.txt");
		else if(i == 2)   inputCatFile.open("..\\Data\\MEDI_CAT.txt");
		else if(i == 3) inputCatFile.open("..\\Data\\DIET_CAT.txt");

		while(true)
		{
			string testLine;
			getline(inputCatFile, testLine);
			//cout << testLine << endl;

			if(inputCatFile.eof())
				break;

			cat_num[i] += 1;
		}
		inputCatFile.close();
	}
	//cout << "Num symptoms: " << cat_num[0] << endl;
	//cout << "Num exercises: " << cat_num[1] << endl;
	//cout << "Num medications: " << cat_num[2] << endl;
	//cout << "Num diets: " << cat_num[3] << endl;

	// Select the variable inputs
	unsigned int num_inputs = (rand() % (MAX_INPUTS - MIN_INPUTS + 1)) +
MIN_INPUTS;
	for(unsigned int i = 0; i < num_inputs; ++i)
	{
		// Select a random type of signal input node
		InputChromosome new_input;

		// Choose the data type (symp/exer/medi/diet)
		InputType::Type data_type = (InputType::Type) (rand() %
InputType::TimeOfDay);
		new_input.input_type = data_type;

		// Choose a category index
```

APPENDIX D (continued)

```
            new_input.input_idx = rand() % cat_num[(unsigned int)
new_input.input_type];

            // Choose data range
            if(data_type == InputType::Symptom)
            {
                  new_input.data_back = (rand() % (max_data_range -
CHUNK_SIZE)) + CHUNK_SIZE; // minimum offset of 3 hours before sample
                  new_input.data_front = (rand() % (max_data_range -
new_input.data_back + 1)) + new_input.data_back; // minimum offset of 3 hours
before sample
            }
            else
            {
                  new_input.data_back = rand() % max_data_range;
                  new_input.data_front = (rand() % (max_data_range -
new_input.data_back + 1)) + new_input.data_back;
            }

            // Add genome to the list
            input_node.push_back(new_input);
      }
}

//---------------------------------------------------------------------

void Genome::InitHybrid(unsigned int gen_num, unsigned int mem_num, Genome
gene_A, Genome gene_B)
{
      generation_num = gen_num;
      member_num = mem_num;
      score = -1.0f;
      //cout << "Initializing hybrid Genome:\tGen: " << generation_num << ",
Mem: " << member_num << endl;

      /*
      cout << "Genome A:" << endl;
      gene_A.Print();
      cout << endl << endl << "Genome B:" << endl;
      gene_B.Print();
      */

      //---------------------------------------------------------------------
--
      // Genome portion A - body and constants
      //---------------------------------------------------------------------
--

      // Set the random seed, learning rate and momentum rate
      rand_seed = gene_A.getRandSeed();
      learn_rate = gene_A.getLearnRate();
      momentum = gene_A.getMomentum();

      // Configure the body of the BPNN
```

APPENDIX D (continued)

```
	hl_nodes.clear();
	for(unsigned int i = 0; i < gene_A.getHLayers(); ++i)
		hl_nodes.push_back(gene_A.getHLNodes(i));


	//-------------------------------------------------------------------
--
	// Genome portion B - inputs
	//-------------------------------------------------------------------
--

	// Select the variable inputs
	for(unsigned int i = 0; i < gene_B.getNumInputs(); ++i)
	{
		// Select a random type of signal input node
		InputChromosome new_input;//, gene_input;

		// Fetch the data type (symp/exer/medi/diet)
		new_input.input_type = gene_B.getInput(i).input_type;

		// Fetch category index
		new_input.input_idx = gene_B.getInput(i).input_idx;

		// Fetch data range
		new_input.data_front = gene_B.getInput(i).data_front;
		new_input.data_back = gene_B.getInput(i).data_back;

		// Add genome to the list
		input_node.push_back(new_input);
	}

	//cout << endl << endl << "Hybrid:" << endl;
	//Print();
}

//---------------------------------------------------------------------

void Genome::InitMutant(unsigned int gen_num, unsigned int mem_num, Genome
gene_A)
{
	generation_num = gen_num;
	member_num = mem_num;
	score = -1.0f;
	//cout << "Initializing mutant Genome:\tGen: " << generation_num << ",
Mem: " << member_num << endl;

	//cout << "Genome A:" << endl;
	//gene_A.Print();

	// First, copy the parent genome

	// Set the random seed, learning rate and momentum rate
	rand_seed = gene_A.getRandSeed();
	learn_rate = gene_A.getLearnRate();
```

APPENDIX D (continued)

```
	momentum = gene_A.getMomentum();

	// Configure the body of the BPNN
	hl_nodes.clear();
	for(unsigned int i = 0; i < gene_A.getHLayers(); ++i)
		hl_nodes.push_back(gene_A.getHLNodes(i));

	// Select the variable inputs
	for(unsigned int i = 0; i < gene_A.getNumInputs(); ++i)
	{
		// Select a random type of signal input node
		InputChromosome new_input;//, gene_input;

		// Fetch the data type (symp/exer/medi/diet)
		new_input.input_type = gene_A.getInput(i).input_type;

		// Fetch category index
		new_input.input_idx = gene_A.getInput(i).input_idx;

		// Fetch data range
		new_input.data_front = gene_A.getInput(i).data_front;
		new_input.data_back = gene_A.getInput(i).data_back;

		// Add genome to the list
		input_node.push_back(new_input);
	}

	bool try_again;
	do
	{
		try_again = false;

		MutationType::Type mut_type = (MutationType::Type) (rand() %
(unsigned int)MutationType::NUM_MUTATION_TYPES);
		//cout << "Mutation Type: " << mut_type << endl;

		// Rand seed mutation
		if(mut_type == MutationType::RandomSeedReset)
			rand_seed = rand();

		// Learn Rate mutations
		else if(mut_type == MutationType::LearnRateIncrease)
		{
			if(learn_rate == MAX_LEARNRATE)
				try_again = true;
			else
			{
				learn_rate *= 1.1f;
				if(learn_rate > MAX_LEARNRATE)
					learn_rate = MAX_LEARNRATE;
			}
		}
		else if(mut_type == MutationType::LearnRateDecrease)
		{
```

APPENDIX D (continued)

```
                  learn_rate *= 0.9f;
                  if(learn_rate < MIN_LEARNRATE)
                        learn_rate = MIN_LEARNRATE;
            }
            else if(mut_type == MutationType::LearnRateRandom)
                  learn_rate = (double(rand()) / double(RAND_MAX)) *
(MAX_LEARNRATE - MIN_LEARNRATE) + MIN_LEARNRATE;

            // Momentum mutations
            else if(mut_type == MutationType::MomentumIncrease)
            {
                  if(momentum == MAX_MOMENTUM)
                        try_again = true;
                  else
                  {
                        momentum *= 1.1f;
                        if(momentum > MAX_MOMENTUM)
                              momentum = MAX_MOMENTUM;
                  }
            }
            else if(mut_type == MutationType::MomentumDecrease)
            {
                  momentum *= 0.9f;
                  if(momentum < MIN_MOMENTUM)
                        momentum = MIN_MOMENTUM;
            }
            else if(mut_type == MutationType::MomentumRandom)
                  momentum = (double(rand()) / double(RAND_MAX)) *
(MAX_MOMENTUM - MIN_MOMENTUM) + MIN_MOMENTUM;

            // Hidden Layer mutations
            else if(mut_type == MutationType::HiddenLayerAdd)
            {
                  if(gene_A.getHLayers() < MAX_HLAYERS)
                  {
                        unsigned int insert_layer_idx = rand() %
(gene_A.getHLayers()+1);
                        //cout << "Selected insertion point: " <<
insert_layer_idx << endl;
                        hl_nodes.clear();
                        for(unsigned int layer = 0; layer <
gene_A.getHLayers()+1; ++layer)
                        {
                              if(layer < insert_layer_idx)

      hl_nodes.push_back(gene_A.getHLNodes(layer));
                              else if(layer == insert_layer_idx)
                                    hl_nodes.push_back((rand() % (MAX_HLNODES
- MIN_HLNODES)) + MIN_HLNODES);
                              else

      hl_nodes.push_back(gene_A.getHLNodes(layer-1));
                        }
                  }
```

APPENDIX D (continued)

```
                else
                        try_again = true;
        }
        else if(mut_type == MutationType::HiddenLayerRemove)
        {
                if(gene_A.getHLayers() > MIN_HLAYERS)
                {
                        unsigned int remove_layer_idx = rand() %
gene_A.getHLayers();
                        //cout << "Selected removal point: " <<
remove_layer_idx << endl;
                        hl_nodes.clear();
                        for(unsigned int layer = 0; layer <
gene_A.getHLayers(); ++layer)
                        {
                                if(layer != remove_layer_idx)

      hl_nodes.push_back(gene_A.getHLNodes(layer));
                        }
                }
                else
                        try_again = true;
        }
        else if(mut_type == MutationType::HiddenNodeAdd)
        {
                unsigned int insert_node_idx = rand() % getHLayers();
                //cout << "Selected insertion point: " << insert_node_idx
<< endl;
                if(hl_nodes[insert_node_idx] >= MAX_HLNODES)
                        try_again = true;
                else
                        hl_nodes[insert_node_idx] += 1;
                //if(hl_nodes[insert_node_idx] > MAX_HLNODES)
                        //hl_nodes[insert_node_idx] = MAX_HLNODES;


        }
        else if(mut_type == MutationType::HiddenNodeRemove)
        {
                unsigned int remove_node_idx = rand() % getHLayers();
                //cout << "Selected removal point: " << remove_node_idx <<
endl;
                if(hl_nodes[remove_node_idx] <= MIN_HLNODES)
                        try_again = true;
                else
                        hl_nodes[remove_node_idx] -= 1;
                //if(hl_nodes[remove_node_idx] < MIN_HLNODES)
                        //hl_nodes[remove_node_idx] = MIN_HLNODES;
        }

        // Input Node mutations
        else if(mut_type == MutationType::InputRemove)
        {
                if(getNumInputs() > MIN_INPUTS)
```

APPENDIX D (continued)

```
                  {
                        unsigned int remove_input_idx = rand() %
getNumInputs();
                        //cout << "Selected removal input: " <<
remove_input_idx << endl;
                        input_node.clear();
                        for(unsigned int input = 0; input <
gene_A.getNumInputs(); ++input)
                        {
                              if(input != remove_input_idx)

      input_node.push_back(gene_A.getInput(input));
                        }
                  }
                  else
                        try_again = true;
            }


            else
            {
                  // Fetch and calculate max data time (hour)
/*
                  stringstream test_idx_file_name;
                  test_idx_file_name << "..\\Data\\TESTCASES.vec";
                  ifstream inputTestFile(test_idx_file_name.str().c_str(),
ios_base::binary);
                  std::vector<int> test_idx;
                  while(true)
                  {
                        int test_idx_entry;
                        inputTestFile.read((char*) &test_idx_entry,
sizeof(int));

                        if(inputTestFile.eof())
                              break;

                        test_idx.push_back(test_idx_entry);
                  }
                  inputTestFile.close();
                  unsigned int max_training_idx = (unsigned int)
(test_idx.size() * (100.0f/TEST_SET_PERCENT));
                  unsigned int max_data_range = max_training_idx *
CHUNK_SIZE;
*/
                  unsigned int max_data_range = MAX_LEAD_TIME;
                  //cout << "Max data range: " << max_data_hour << endl;

                  // Fetch max type indexes
                  std::vector<unsigned int> cat_num;
                  for(unsigned int i = 0; i < 4; ++i) cat_num.push_back(0);
                  for(unsigned int i = 0; i < 4; ++i)
                  {
                        if(i == 0)
```

APPENDIX D (continued)

```
                        {
                              cat_num[i] = 8; // we don't need to count the
number of symptoms. there are always 8
                              continue;
                        }

                        ifstream inputCatFile;
                               if(i == 1)
      inputCatFile.open("..\\Data\\EXER_CAT.txt");
                        else if(i == 2)
      inputCatFile.open("..\\Data\\MEDI_CAT.txt");
                        else if(i == 3)
inputCatFile.open("..\\Data\\DIET_CAT.txt");

                        while(true)
                        {
                              string testLine;
                              getline(inputCatFile, testLine);
                              //cout << testLine << endl;

                              if(inputCatFile.eof())
                                    break;

                              cat_num[i] += 1;
                        }
                        inputCatFile.close();
                  }
                  //cout << "Num symptoms: " << cat_num[0] << endl;
                  //cout << "Num exercises: " << cat_num[1] << endl;
                  //cout << "Num medications: " << cat_num[2] << endl;
                  //cout << "Num diets: " << cat_num[3] << endl;

                  // Select the variable inputs
                  //unsigned int num_inputs = rand() % MAX_INPUTS;
                  if(mut_type == MutationType::InputAdd)
                  {
                        if(input_node.size() < MAX_INPUTS)
                        {
                              // Select a random type of signal input node
                              InputChromosome new_input;

                              // Choose the data type (symp/exer/medi/diet)
                              InputType::Type data_type = (InputType::Type)
(rand() % InputType::TimeOfDay);
                              new_input.input_type = data_type;

                              // Choose a category index
                              new_input.input_idx = rand() %
cat_num[(unsigned int) new_input.input_type];

                              // Choose data range
                              if(data_type == InputType::Symptom)
                              {
```

APPENDIX D (continued)

```
                                    new_input.data_back = (rand() %
(max_data_range - CHUNK_SIZE)) + CHUNK_SIZE; // minimum offset of 3 hours
before sample
                                    new_input.data_front = (rand() %
(max_data_range- new_input.data_back)) + new_input.data_back; // minimum
offset of 3 hours before sample
                              }
                              else
                              {
                                    new_input.data_back = rand() %
max_data_range;
                                    new_input.data_front = (rand() %
(max_data_range - new_input.data_back)) + new_input.data_back;
                              }

                              // Add input to the list
                              input_node.push_back(new_input);
                        }
                        else
                              try_again = true;
                  }
                  else if(mut_type == MutationType::InputReplace)
                  {
                        // Select a random type of signal input node
                        InputChromosome new_input;

                        // Choose the data type (symp/exer/medi/diet)
                        InputType::Type data_type = (InputType::Type) (rand()
% InputType::TimeOfDay);
                        new_input.input_type = data_type;

                        // Choose a category index
                        new_input.input_idx = rand() % cat_num[(unsigned int)
new_input.input_type];

                        // Choose data range
                        if(data_type == InputType::Symptom)
                        {
                              new_input.data_back = (rand() % (max_data_range
- CHUNK_SIZE)) + CHUNK_SIZE; // minimum offset of 3 hours before sample
                              new_input.data_front = (rand() %
(max_data_range - new_input.data_back)) + new_input.data_back; // minimum
offset of 3 hours before sample
                        }
                        else
                        {
                              new_input.data_back = rand() % max_data_range;
                              new_input.data_front = (rand() %
(max_data_range - new_input.data_back)) + new_input.data_back;
                        }

                        // Add remove old input, and add new input to the
list
```

APPENDIX D (continued)

```
                        unsigned int replace_input_idx = rand() %
getNumInputs();
                        //cout << "Selected removal input: " <<
remove_input_idx << endl;
                        input_node.clear();
                        for(unsigned int input = 0; input <
gene_A.getNumInputs(); ++input)
                        {
                              if(input != replace_input_idx)

      input_node.push_back(gene_A.getInput(input));
                              else
                                    input_node.push_back(new_input);
                        }
                  }
                  else if(getNumInputs() != 0)
                  {
                        unsigned int mutate_input_idx = rand() %
getNumInputs();
                        //cout << "Selected mutate input: " <<
mutate_input_idx << endl;

                        if(mut_type == MutationType::InputRangeFrontIncrease)
      // <-- (hour ,
                        {
                              if(input_node[mutate_input_idx].data_front >=
max_data_range)
                                    try_again = true;
                              else
                              {
                                    if(rand() % 2)
      input_node[mutate_input_idx].data_front = (unsigned int) ((double)
input_node[mutate_input_idx].data_front * 1.1f);
                                    else
      input_node[mutate_input_idx].data_front =
input_node[mutate_input_idx].data_front + 1;

      if(input_node[mutate_input_idx].data_front > max_data_range)

      input_node[mutate_input_idx].data_front = max_data_range;
                              }
                        }
                        if(mut_type == MutationType::InputRangeFrontDecrease)
      // (hour, -->
                        {
                              if(input_node[mutate_input_idx].data_front <=
input_node[mutate_input_idx].data_back)
                                    try_again = true;
                              else
                              {
                                    if(rand() % 2)
      input_node[mutate_input_idx].data_front = (unsigned int) ((double)
input_node[mutate_input_idx].data_front * 0.9f);
```

APPENDIX D (continued)

```
                                    else
      input_node[mutate_input_idx].data_front =
input_node[mutate_input_idx].data_front - 1;

      if(input_node[mutate_input_idx].data_front <
input_node[mutate_input_idx].data_back)

      input_node[mutate_input_idx].data_front =
input_node[mutate_input_idx].data_back;
                              }
                        }
                        if(mut_type == MutationType::InputRangeBackIncrease)
      // <-- ,hour)
                        {
                              if(input_node[mutate_input_idx].data_back >=
input_node[mutate_input_idx].data_front)
                                    try_again = true;
                              else
                              {
                                    if(rand() % 2)
      input_node[mutate_input_idx].data_back = (unsigned int) ((double)
input_node[mutate_input_idx].data_back * 1.1f);
                                    else
      input_node[mutate_input_idx].data_back =
input_node[mutate_input_idx].data_back + 1;
                                    if(input_node[mutate_input_idx].data_back
> input_node[mutate_input_idx].data_front)

      input_node[mutate_input_idx].data_back =
input_node[mutate_input_idx].data_front;
                              }
                        }
                        if(mut_type == MutationType::InputRangeBackDecrease)
      // ,hour) -->
                        {
                              unsigned int min_back;
                              if(input_node[mutate_input_idx].input_type ==
InputType::Symptom)     min_back = CHUNK_SIZE;
                              else
                                          min_back = 0;

                              if(input_node[mutate_input_idx].data_back <=
min_back)
                                    try_again = true;
                              else
                              {
                                    if(rand() % 2)

      input_node[mutate_input_idx].data_back = (unsigned int) ((double)
input_node[mutate_input_idx].data_back * 0.9f);
                                    else
if(input_node[mutate_input_idx].data_back > min_back)
```

APPENDIX D (continued)

```
	input_node[mutate_input_idx].data_back =
input_node[mutate_input_idx].data_back - 1;
                                    if(input_node[mutate_input_idx].data_back
< min_back)

	input_node[mutate_input_idx].data_back = min_back;
                                }
                        }
                        if(mut_type == MutationType::InputRangeRandom)
                        {
                                // Choose data range
                                if(input_node[mutate_input_idx].input_type ==
InputType::Symptom)
                                {
                                        //cout << "type == symptom" << endl;
                                        //cout << "max_data_range: " <<
max_data_range << endl;
                                        input_node[mutate_input_idx].data_back =
(rand() % (max_data_range - CHUNK_SIZE)) + CHUNK_SIZE; // minimum offset of 3
hours before sample
                                        //cout << "data_back: " <<
input_node[mutate_input_idx].data_back << endl;
                                        input_node[mutate_input_idx].data_front =
(rand() % (max_data_range - input_node[mutate_input_idx].data_back + 1)) +
input_node[mutate_input_idx].data_back; // minimum offset of 3 hours before
sample
                                        //cout << "data_front: " <<
input_node[mutate_input_idx].data_front << endl;
                                }
                                else
                                {
                                        //cout << "type != symptom" << endl;
                                        input_node[mutate_input_idx].data_back =
rand() % max_data_range;
                                        input_node[mutate_input_idx].data_front =
(rand() % (max_data_range - input_node[mutate_input_idx].data_back + 1)) +
input_node[mutate_input_idx].data_back;
                                }
                        }
                }
        }
	}while(try_again);


	//cout << "Mutated genome: " << endl;
	//Print();
}

//------------------------------------------------------------------

void Genome::SaveFile(unsigned int slot_num, bool with_training_passes/* =
true*/)
{
	// Create filename
```

APPENDIX D (continued)

```
	stringstream gene_file_name;
	gene_file_name << "..\\Genomes\\Gene_";
	if(generation_num < 1000)	gene_file_name << "0";
	if(generation_num < 100)	gene_file_name << "0";
	if(generation_num < 10)		gene_file_name << "0";
	gene_file_name << generation_num << "-";
	if(slot_num < 10)		gene_file_name << "0";
	gene_file_name << slot_num << ".gen";
	cout << "Filename: " << gene_file_name.str() << endl;

	// Open output file for writing (binary)
	ofstream outputGenomeFile(gene_file_name.str().c_str(),
ios_base::binary);

	// Write ID stuff
	outputGenomeFile.write((char*) &generation_num, sizeof(unsigned int));
	outputGenomeFile.write((char*) &member_num, sizeof(unsigned int));

	// Write Section A - variables and body config
	outputGenomeFile.write((char*) &rand_seed, sizeof(unsigned int));
	outputGenomeFile.write((char*) &learn_rate, sizeof(double));
	outputGenomeFile.write((char*) &momentum, sizeof(double));
	outputGenomeFile.write((char*) &avg_MSE, sizeof(double));
	outputGenomeFile.write((char*) &score, sizeof(double));
	if(with_training_passes)
		outputGenomeFile.write((char*) &training_passes, sizeof(unsigned
int));

	unsigned int num_layers = hl_nodes.size();
	outputGenomeFile.write((char*) &num_layers, sizeof(unsigned int));
	for(unsigned int layer = 0; layer < num_layers; ++layer)
		outputGenomeFile.write((char*) &hl_nodes[layer], sizeof(int));

	// Write Section B - inputs
	unsigned int num_inputs = input_node.size();
	outputGenomeFile.write((char*) &num_inputs, sizeof(unsigned int));
	for(unsigned int input = 0; input < num_inputs; ++input)
	{
		outputGenomeFile.write((char*) &input_node[input].input_type,
sizeof(InputType::Type));
		outputGenomeFile.write((char*) &input_node[input].input_idx,
sizeof(unsigned int));
		outputGenomeFile.write((char*) &input_node[input].data_front,
sizeof(unsigned int));
		outputGenomeFile.write((char*) &input_node[input].data_back,
sizeof(unsigned int));
	}

	// Close output file
	outputGenomeFile.close();
}

//---------------------------------------------------------------------
```

APPENDIX D (continued)

```
bool Genome::LoadFile(unsigned int gen_num, unsigned int mem_num, unsigned
int experiment_num/* = -1*/, bool with_training_passes/* = true*/)
{
	// Record generation and member #
	generation_num = gen_num;
	member_num = mem_num;

	// Create filename
	stringstream gene_file_name;
	gene_file_name << "..\\Genomes";
	if(experiment_num != -1)
	{
		gene_file_name << "\\Run";
		if(experiment_num < 10)
			gene_file_name << "0";
		gene_file_name << experiment_num;
	}
	gene_file_name << "\\Gene_";
	if(generation_num < 1000)	gene_file_name << "0";
	if(generation_num < 100)	gene_file_name << "0";
	if(generation_num < 10)		gene_file_name << "0";
	gene_file_name << generation_num << "-";
	if(member_num < 10)			gene_file_name << "0";
	gene_file_name << member_num << ".gen";
	cout << "Filename: " << gene_file_name.str() << endl;

	// Open output file for writing (binary)
	ifstream intputGenomeFile(gene_file_name.str().c_str(),
ios_base::binary);

	if(!intputGenomeFile)
		return false;

	// Read ID stuff
	intputGenomeFile.read((char*) &generation_num, sizeof(unsigned int));
	intputGenomeFile.read((char*) &member_num, sizeof(unsigned int));

	// Read Section A - variables and body config
	intputGenomeFile.read((char*) &rand_seed, sizeof(unsigned int));
	intputGenomeFile.read((char*) &learn_rate, sizeof(double));
	intputGenomeFile.read((char*) &momentum, sizeof(double));
	intputGenomeFile.read((char*) &avg_MSE, sizeof(double));
	intputGenomeFile.read((char*) &score, sizeof(double));
	if(with_training_passes)
		intputGenomeFile.read((char*) &training_passes, sizeof(unsigned
int));

	unsigned int num_layers;
	intputGenomeFile.read((char*) &num_layers, sizeof(unsigned int));
	hl_nodes.clear();
	for(unsigned int layer = 0; layer < num_layers; ++layer)
	{
		int hl_layer_size;
		intputGenomeFile.read((char*) &hl_layer_size, sizeof(int));
```

APPENDIX D (continued)

```
			hl_nodes.push_back(hl_layer_size);
	}

	// Read Section B - inputs
	unsigned int num_inputs;
	intputGenomeFile.read((char*) &num_inputs, sizeof(unsigned int));
	input_node.clear();
	for(unsigned int input = 0; input < num_inputs; ++input)
	{
			InputChromosome member_input;

			intputGenomeFile.read((char*) &member_input.input_type,
sizeof(InputType::Type));
			intputGenomeFile.read((char*) &member_input.input_idx,
sizeof(unsigned int));
			intputGenomeFile.read((char*) &member_input.data_front,
sizeof(unsigned int));
			intputGenomeFile.read((char*) &member_input.data_back,
sizeof(unsigned int));
			input_node.push_back(member_input);
	}

	// Close output file
	intputGenomeFile.close();

	return true;
}


//---------------------------------------------------------------------

void Genome::Print()
{
	cout << "Print Genome:\tGen: " << generation_num << ", Mem: " <<
member_num << endl;

	//---------------------------------------------------------------------
--
	// Genome portion A - body and constants
	//---------------------------------------------------------------------
--

	// Print the random seed, learning rate and momentum rate
	cout << "\tRandom Seed: " << rand_seed << endl;
	cout << "\tLearning Rate: " << learn_rate << endl;
	cout << "\tAdjustment Momentum: " << momentum << endl;
	cout << "\tMSE: " << avg_MSE << endl;
	cout << "\tScore: " << score << endl;
	cout << "\tTraining Passes: " << getTP() << endl;

	// Print the body configuration of the BPNN
	cout << "\tHidden layers: ";
	for(unsigned int i = 0; i < hl_nodes.size(); ++i)
			cout << hl_nodes[i] << " ";
```

APPENDIX D (continued)

```
	cout << endl;

	//-------------------------------------------------------------------
--
	// Genome portion B - inputs
	//-------------------------------------------------------------------
--

	// Print the inputs
	unsigned int num_inputs = rand() % MAX_INPUTS;
	for(unsigned int i = 0; i < input_node.size(); ++i)
	{
		// Print input
		cout << "\tInput # " << i << ":";
		cout << "\ttype: ";
		if(input_node[i].input_type == InputType::Symptom)
	cout << "Symp";
		else if(input_node[i].input_type == InputType::Exercise)
	cout << "Exer";
		else if(input_node[i].input_type == InputType::Medication)
	cout << "Medi";
		else if(input_node[i].input_type == InputType::Diet)
	cout << "Diet";
		cout << "\tidx: " << input_node[i].input_idx;
		cout << "\trange: (" << input_node[i].data_front << "," <<
input_node[i].data_back << ")" << endl;
	}
}
```

APPENDIX D (continued)

BPNNNode.hpp

```
/*****************************************
* BPNNNode.hpp
*
* Basic BPNN node
*
* Author: Jon Lessner 2012
*****************************************/

#ifndef BPNNNODE_HPP_INCLUDED
#define BPNNNODE_HPP_INCLUDED

#include "../Includes.h"

class BPNNNode
{

      public:

            BPNNNode();

            //BPNNNode(int value);  // test implementation

            ~BPNNNode();

            void PushUplink(BPNNNode* parent_node);
            void PushDownlink(BPNNNode* child_node, double weight);

            BPNNNode* GetUplink(unsigned int index);
            BPNNNode* GetDownlink(unsigned int index);

            unsigned int GetUplinkCnt() {return Uplink.size();}
            unsigned int GetDownlinkCnt() {return Downlink.size();}
            double GetDownlinkWeight(unsigned int link_index) {return
DownlinkWeight[link_index];}

            //void PushWeightAdjustment(int node_index, double adjustment);
// obsolete

            void ResetActivation() {Activation = 0.0f;}
            double getActivationValue() {return Activation;}

            virtual void CalcError(int time_index);
            void ResetError() {Error = 0.0f;}
            double getErrorValue() {return Error;}

            virtual     void AdjustWeight(double learn_rate, double
momentum);

            void FireNeuron();
            void AddSignal(double signal_value) {Activation += signal_value;}
            void SigmaFilter();
```

APPENDIX D (continued)

```
            double dSigmaFilter();

            int getTestValue() {return test_value;}

            void RunTest();
      protected:

            std::vector<BPNNNode*> Uplink, Downlink; // Uplink = connections
to previous layer nodes, Downlink = connections to following layer nodes
            std::vector<double> DownlinkWeight;
            std::vector<double> SavedAdjustment;

            double Activation;
            double Error;

            int test_value;

      private:

            BPNNNode* BPNNNode::FindParentDownlink(unsigned int parent_idx);
            unsigned int BPNNNode::FindParentDownlinkIdx(unsigned int
parent_idx);

      }; // end class BPNNNode

#endif // end BPNNNODE_HPP_INCLUDED
```

APPENDIX D (continued)

BPNNNode.cpp

```
/**
* BPNNNode.cpp
*
* Implementation of the BPNNNode class.
* Basic BPNN node
*
* Author: Jon Lessner 2012
*/

#include "BPNNNode.hpp"

using namespace std;

#define WEIGHT_MAX      1.0f
#define WEIGHT_MIN      -1.0f

//---------------------------------------------------------------------

BPNNNode::BPNNNode()
//:more_inits(INIT_VAL)
{
      //cout << "BPNNNode created" << endl;
//    SavedAdjustment = 0.0f;
}

/*
//---------------------------------------------------------------------

BPNNNode::BPNNNode(int value) // test implementation
//:more_inits(INIT_VAL)
{
      //cout << "BPNNNode created: " << value << endl;
      test_value = value;
      //SavedAdjustment = 0.0f;
}
*/

//---------------------------------------------------------------------

BPNNNode::~BPNNNode()
{
      //cout << "BPNNNode deleted: " << test_value << endl;
}

//---------------------------------------------------------------------

void BPNNNode::PushUplink(BPNNNode* parent_node)
{
      Uplink.push_back(parent_node);
```

APPENDIX D (continued)

```
//    cout << "node with idx " << getTestValue() << " uplinked to node with
idx " << Uplink.back()->getTestValue() << endl;
}

//------------------------------------------------------------------------

void BPNNNode::PushDownlink(BPNNNode* child_node, double weight)
{
      Downlink.push_back(child_node);
      DownlinkWeight.push_back(weight);
      SavedAdjustment.push_back(0.0f);
}

//------------------------------------------------------------------------

BPNNNode* BPNNNode::GetUplink(unsigned int index)
{
      if(index >= Uplink.size())
      {
            cout << "index (" << index << ") out of range: (0-" <<
(Uplink.size()-1) << ")" << endl;
            return NULL;
      }

      return Uplink[index];
}

//------------------------------------------------------------------------

BPNNNode* BPNNNode::GetDownlink(unsigned int index)
{
      if(index >= Downlink.size())
      {
            cout << "index (" << index << ") out of range: (0-" <<
(Downlink.size()-1) << ")" << endl;
            return NULL;
      }

      return Downlink[index];
}

//------------------------------------------------------------------------

void BPNNNode::FireNeuron()
{
      for(unsigned int i = 0; i < GetDownlinkCnt(); ++i)
            Downlink[i]->AddSignal(Activation * DownlinkWeight[i]);
//          Downlink[i]->AddSignal(Activation);
}

//------------------------------------------------------------------------

void BPNNNode::SigmaFilter()
{
```

APPENDIX D (continued)

```
	//cout << "Sigma(" << Activation << "): ";
	Activation = (1.0f / (1.0f + exp(-Activation)));
	//cout << Activation << endl;
}

//---------------------------------------------------------------------

double BPNNNode::dSigmaFilter()
{
	//cout << "Sigma(" << Activation << "): ";
	return (Activation * (1 - Activation));
	//cout << Activation << endl;
}

/* obsolete
//---------------------------------------------------------------------

void BPNNNode::PushWeightAdjustment(int node_index, double adjustment)
{
//	for(unsigned int i = 0; i < GetDownlinkCnt(); ++i)
//		cout << "Dowlink idx: " << i << ", old weight: " <<
DownlinkWeight[i] << endl;
	// Correct the downlink weight
//	cout << "adjustment idx: " << node_index << ", adjustment value: " <<
adjustment << endl;
	DownlinkWeight[node_index] += adjustment;
	if(DownlinkWeight[node_index] > WEIGHT_MAX)
		DownlinkWeight[node_index] = WEIGHT_MAX;
	if(DownlinkWeight[node_index] < WEIGHT_MIN)
		DownlinkWeight[node_index] = WEIGHT_MIN;
//	for(unsigned int i = 0; i < GetDownlinkCnt(); ++i)
//		cout << "Dowlink idx: " << i << ", new weight: " <<
DownlinkWeight[i] << endl;
//	cout << endl;

	// Propogate the the adjustment back
	for(unsigned int i = 0; i < GetUplinkCnt(); ++i)
		GetUplink(i)->PushWeightAdjustment(FindParentDownlinkIdx(i),
adjustment);
}
*/
//---------------------------------------------------------------------

void BPNNNode::CalcError(int time_index)
{
	// All generic error calcs will ignore the time_index, and pull error
up from downstream net nodes
	double err = 0;
	for(unsigned int link = 0; link < GetDownlinkCnt(); ++link)
		err = err + GetDownlinkWeight(link) * GetDownlink(link)-
>getErrorValue();
	Error = dSigmaFilter() * err; // (derivitive of Sigmoid) * Error
}
```

APPENDIX D (continued)

```
//-------------------------------------------------------------------------

void BPNNNode::AdjustWeight(double learn_rate, double momentum)
{
	for(unsigned int i = 0; i < GetDownlinkCnt(); ++i)
	{
		double del_w = learn_rate * GetDownlink(i)->getErrorValue() *
Activation;
		//cout << " del_w: " << del_w << endl;

		//cout << "\t\tdownlink #" << i << ": " << DownlinkWeight[i];
		DownlinkWeight[i] += del_w + (momentum * SavedAdjustment[i]);
		//cout << ":" << DownlinkWeight[i] << endl;
		SavedAdjustment[i] = del_w;
	}
}

//-------------------------------------------------------------------------

void BPNNNode::RunTest()
{
//	for(unsigned int i = 0; i < GetUplinkCnt(); ++i)
//		GetUplink(i)->PushWeightAdjustment(FindParentDownlinkIdx(i),
0.05f);
}

//-------------------------------------------------------------------------

BPNNNode* BPNNNode::FindParentDownlink(unsigned int parent_idx)
{
	for(unsigned int i = 0; i < GetUplink(parent_idx)->GetDownlinkCnt();
++i)
	{
		if(GetUplink(parent_idx)->GetDownlink(i) == this)
			return GetUplink(parent_idx)->GetDownlink(i);
	}

	return NULL;
}

//-------------------------------------------------------------------------

unsigned int BPNNNode::FindParentDownlinkIdx(unsigned int parent_idx)
{
	for(unsigned int i = 0; i < GetUplink(parent_idx)->GetDownlinkCnt();
++i)
	{
		if(GetUplink(parent_idx)->GetDownlink(i) == this)
			return i;
	}

	return 0;
}
```

APPENDIX D (continued)

OutputNode.hpp

```
/*******************************************
* OutputNode.hpp
*
* Basic BPNN prediction node
*
* Author: Jon Lessner 2012
*******************************************/

#ifndef OUTPUTNODE_HPP_INCLUDED
#define OUTPUTNODE_HPP_INCLUDED

#include "BPNNNode.hpp"
#include "../Includes.h"

class OutputNode : public BPNNNode
{

      public:

            OutputNode();

//          OutputNode(int value);  // test implementation
            OutputNode(unsigned int outnode_idx);

            ~OutputNode();

            double GetSympOutputVector(int time_index);
            void CalcError(int time_index);
            double getRawErrorValue() {return RawError;}

            void CalcOptimizerError(double target_activation);

      protected:

            std::vector<double> symp_output;

            unsigned int symp_idx;

            double RawError;  // place to store (Target-Activation) for MSE
calc

      private:

      }; // end class OutputNode

#endif // end OUTPUTNODE_HPP_INCLUDED
```

APPENDIX D (continued)

OutputNode.cpp

```
/**
* OutputNode.cpp
*
* Implementation of the OutputNode class.
* Basic BPNN perception node
*
* Author: Jon Lessner 2012
*/

#include "OutputNode.hpp"

using namespace std;

//---------------------------------------------------------------------------

OutputNode::OutputNode()
:BPNNNode()
//,more_inits(INIT_VAL)
{
	//cout << "OutputNode created" << endl;
}


//---------------------------------------------------------------------------

OutputNode::OutputNode(unsigned int outnode_idx)
:BPNNNode()
//,more_inits(INIT_VAL)
{
	//cout << "OutputNode created: " << value << endl;

	// Open input file
	stringstream file_name;
	file_name.clear();
	symp_idx = outnode_idx;
	file_name << "..\\Data\\SYMP" << symp_idx << ".vec";
	//cout << "\tsymp_idx: " << symp_idx << endl;
	//cout << file_name.str() << endl;

	ifstream inputFile(file_name.str().c_str(), ios_base::binary);

	if(!inputFile.is_open())
	{
		cout << "File open failed!" << endl;
		return;
	}

	std::vector<DataEntry> read_vec;

	// Read data from file
```

APPENDIX D (continued)

```
	while(true)
	{
		DataEntry data_entry;
		inputFile.read((char*) &data_entry, sizeof(DataEntry));

		if(inputFile.eof())
			break;

		read_vec.push_back(data_entry);

		//cout << "E#: " << read_vec.size() << ", T: " <<
read_vec.back().time << ", V: " << read_vec.back().value << endl;
	}

	// Set trackers
	int max_hour = (int) read_vec.back().time;
	unsigned int front_idx = 0, back_idx = 0;
	int cur_hour = 0;
	double max_total = 0;

	//cout << "max_hour: " << max_hour << endl;
	for(int cur_hour = 0; cur_hour < (max_hour+1); ++cur_hour)
	{
		// Move forward  back marker
		while((back_idx < (read_vec.size()-1)) &&
			(cur_hour >= (int) read_vec[back_idx].time))		//
make sure  "back" marker is inclusive --> ,hour)
			++back_idx;

		// Move forward front marker
		while((front_idx < (read_vec.size()-1)) &&
			((cur_hour-CHUNK_SIZE) > (int) read_vec[front_idx].time))
	// make sure "front" marker is inclusive --> (hour, --- and that it
starts at only the previous (CHUNK_SIZE) hours
//			(cur_hour > (int) read_vec[front_idx].time))
		// make sure "front" marker is inclusive --> (hour,
			++front_idx;

		// Tally everything in range
		double total_value = 0;
		for(unsigned int range_idx = front_idx; range_idx < (back_idx+1);
++range_idx)
		{
			//if(range_idx == front_idx)
				//cout << cur_hour << "\t(" << front_idx << ":" <<
range_idx << ":" << back_idx << ")" << endl;
			if((range_idx != back_idx) || (cur_hour >= (int)
read_vec[back_idx].time))
				total_value += read_vec[range_idx].value;
		}

		//cout << "total: " << total_value << endl;
		symp_output.push_back(total_value);
```

APPENDIX D (continued)

```
            // Keep track of the largest total for normalization
            if(total_value > max_total)
            {
                  max_total = total_value;
                  //cout << "cur_hour: " << cur_hour << ", total: " <<
total_value << endl;
            }
      }

      // Normalize the vector
      //cout << "Max Total: " << max_total << endl;
      for(unsigned int time_idx = 0; time_idx < symp_output.size();
++time_idx)
      {
            //if(symp_idx==0 && symp_output[time_idx]!=0)
                  //cout << time_idx << ":\t(" << symp_output[time_idx] <<
"\t:\t";
            symp_output[time_idx] = symp_output[time_idx]/max_total;
            //if(symp_idx==0 && symp_output[time_idx]!=0)
                  //cout << symp_output[time_idx] << ")" << endl;
      }

      // Close input file
      inputFile.close();
}

//---------------------------------------------------------------------

double OutputNode::GetSympOutputVector(int time_index)
{
      if(time_index < (int) symp_output.size())
            return symp_output[time_index];
      else
            return 0.0f;
}

//---------------------------------------------------------------------

void OutputNode::CalcError(int time_index)
{
      //cout << GetSympOutputVector(time_index) << " - " << Activation;
/*
      Error = GetSympOutputVector(time_index) - Activation;
      Error = (Activation * (1 - Activation)) * Error; // (derivitive of
Sigmoid) * Error
*/
      RawError = GetSympOutputVector(time_index) - Activation;
      Error = dSigmaFilter() * RawError; // (derivitive of Sigmoid) *
RawError
}

//---------------------------------------------------------------------

void OutputNode::CalcOptimizerError(double target_activation)
```

APPENDIX D (continued)

```
{
	RawError = target_activation - Activation;
/*
	Error = target_activation - Activation;
	Error = (Activation * (1 - Activation)) * Error; // (derivitive of
Sigmoid) * Error
*/
}

//---------------------------------------------------------------------

OutputNode::~OutputNode()
{
	//cout << "InputNode deleted: " << test_value << endl;
}
```

APPENDIX D (continued)

InputNode.hpp

```
/*****************************************
* InputNode.hpp
*
* Basic BPNN perception node
*
* Author: Jon Lessner 2012
*****************************************/

#ifndef INPUTNODE_HPP_INCLUDED
#define INPUTNODE_HPP_INCLUDED

#include "BPNNNode.hpp"
#include "../Includes.h"

class InputNode : public BPNNNode
{
      public:

            InputNode();

//          InputNode(int value);   // test implementation
            InputNode(InputChromosome chromo);  // TODO: input nodes should
be created only with a chromosome (type, range, etc)

            ~InputNode();

            void ReadInputVector(int time_index);
            void SetInputVector(double optimizer_activation)      {
Activation = optimizer_activation; }      // This method is for use only by
the optimizer, to explicitly set Activation values to 1.0/0.0
            InputType::Type getInputType() { return input_type; }
            unsigned int getInputIdx() { return input_idx; }
            int getFrontRange() { return front_range; }
            int getBackRange() { return back_range; }
            double getMaxInput() { return max_input; }

      protected:

            double max_input;
            std::vector<double> input;

            InputType::Type input_type;
            unsigned int input_idx;
            int front_range, back_range;


      private:

      }; // end class InputNode
#endif // end INPUTNODE_HPP_INCLUDED
```

APPENDIX D (continued)

InputNode.cpp

```
/**
* InputNode.cpp
*
* Implementation of the InputNode class.
* Basic BPNN perception node
*
* Author: Jon Lessner 2012
*/

#include "InputNode.hpp"

using namespace std;

//------------------------------------------------------------------------

InputNode::InputNode()
:BPNNNode()
//,more_inits(INIT_VAL)
{
	//cout << "InputNode created" << endl;
}


//------------------------------------------------------------------------

//InputNode::InputNode(int value)   // test implementation
InputNode::InputNode(InputChromosome chromo)    // TODO: input nodes should
be created only with a chromosome (type, range, etc)
:BPNNNode()
//,more_inits(INIT_VAL)
{
	//cout << "InputNode created: " << endl;

	stringstream file_name;

	// Use chromosome to determine type, idx, range
	input_type = chromo.input_type;
	input_idx = chromo.input_idx;
	front_range = chromo.data_front;
	back_range = chromo.data_back;

	/*
	cout << "\tinput_type: " << input_type << "\n\tinput_idx: " <<
input_idx;
	if(input_type != InputType::TimeOfDay)
		cout << "\n\ttime_range: (" << -front_range << "," << -back_range
<< ")";
	cout << endl;
	*/
```

APPENDIX D (continued)

```
	file_name.clear();
	file_name << "..\\Data\\";
	if(input_type == InputType::Symptom)
		file_name << "SYMP" << input_idx << ".vec";
	else if(input_type == InputType::Exercise)
		file_name << "EXER" << input_idx << ".vec";
	else if(input_type == InputType::Medication)
		file_name << "MEDI" << ((input_idx < 10) ? "0":"") << input_idx
<< ".vec";
	else if(input_type == InputType::Diet)
		file_name << "DIET" << ((input_idx < 100) ? "0":"") <<
((input_idx < 10) ? "0":"") << input_idx << ".vec";
	else if(input_type == InputType::TimeOfDay)
	{
		return;
	}
	else
	{
		cout << "Failure to find input type: " << (unsigned int)
input_type << endl;
		return;
	}


	//cout << file_name.str() << endl;

	// Open input file
	ifstream inputFile(file_name.str().c_str(), ios_base::binary);

	if(!inputFile.is_open())
	{
		cout << "File open failed!" << endl;
		return;
	}

	std::vector<DataEntry> read_vec;

	// Read data from file
	while(true)
	{
		DataEntry data_entry;
		inputFile.read((char*) &data_entry, sizeof(DataEntry));

		if(inputFile.eof())
			break;

		read_vec.push_back(data_entry);

		//if(input_type == InputType::Medication)
		//cout << "E#: " << read_vec.size() << ", T: " <<
read_vec.back().time << ", V: " << read_vec.back().value << endl;
	}

	// Set trackers
```

APPENDIX D (continued)

```
	int max_hour = (int) read_vec.back().time + front_range;
	unsigned int front_idx = 0, back_idx = 0;
	int cur_hour = 0;
	double max_total = 0;

	//if(input_type == InputType::Medication)
	//cout << "max_hour: " << max_hour << endl;
	for(int cur_hour = 0; cur_hour < (max_hour+1); ++cur_hour)
	{
		// Move forward  back marker
		while((back_idx < (read_vec.size()-1)) &&
			((cur_hour-back_range) >= (int) read_vec[back_idx].time))
	// make sure "back" marker is inclusive --> ,hour)
			++back_idx;

		// Move forward front marker
		while((front_idx < (read_vec.size()-1)) &&
			((cur_hour-front_range) > (int)
read_vec[front_idx].time))    // make sure "front" marker is inclusive -->
(hour,
			++front_idx;

		// Tally everything in range
		double total_value = 0;
		for(unsigned int range_idx = front_idx; range_idx < (back_idx+1);
++range_idx)
		{
			//cout << cur_hour << "\t(" << front_idx << ":" <<
range_idx << ":" << back_idx << ")" << endl;
			if((range_idx != back_idx) || ((cur_hour-back_range) >=
(int) read_vec[back_idx].time))
				total_value += read_vec[range_idx].value;
		}

		//if(input_type == InputType::Symptom/* && cur_hour == 2289*/)
		//cout << "total: " << total_value << ", fi: " << front_idx << ",
bi: " << back_idx << endl;
		input.push_back(total_value);

		// Keep track of the largest total for normalization
		if(total_value > max_total)
		{
			max_total = total_value;
			//if(input_type == InputType::Medication)
			//cout << "cur_hour: " << cur_hour << ", total: " <<
total_value << endl;
		}
	}

	// Save the max value (for later use in the optimizer only)
	max_input = max_total;

	// Normalize the vector
	//if(input_type == InputType::Medication)
```

APPENDIX D (continued)

```
	//cout << "Max Total: " << max_total << endl;
	for(unsigned int time_idx = 0; time_idx < input.size(); ++time_idx)
	{
		//if(input_type == InputType::Exercise)
		//cout << time_idx << ":\t(" << input[time_idx] << "\t:\t";
		input[time_idx] = input[time_idx]/max_total;
		//if(input_type == InputType::Exercise)
		//cout << input[time_idx] << ")" << endl;
	}

	// Close input file
	inputFile.close();
}

//------------------------------------------------------------------------

InputNode::~InputNode()
{
	//cout << "InputNode deleted: " << test_value << endl;
}

//------------------------------------------------------------------------

void InputNode::ReadInputVector(int time_index)
{
	if(input_type == InputType::TimeOfDay)
	{
		// This will set the first 3-node section of the inputs to
represent time-of-day (which 1/8 of the day it falls into: 000-111, where 000
= 12am-3am and 111 = 9pm-12am, etc)
		if(input_idx == 0)	Activation = (double) ((time_index/12) %
2);
		if(input_idx == 1)	Activation = (double) ((time_index/6) %
2);
		if(input_idx == 2)	Activation = (double) ((time_index/3) %
2);
	}
	else
	{
		if(time_index < (int) input.size())
			Activation = input[time_index];
		else
			Activation = 0.0f;
	}
}
```

APPENDIX D (continued)

BPNN.hpp

```
/*******************************************
* BPNN.hpp
*
* Basic BPNN
*
* Author: Jon Lessner 2012
*******************************************/

#ifndef BPNN_HPP_INCLUDED
#define BPNN_HPP_INCLUDED

#include "InputNode.hpp"
#include "OutputNode.hpp"
#include "BPNNNode.hpp"
#include "Genome.hpp"
#include "../Includes.h"

class BPNN
{

      public:

            BPNN();

            BPNN(int value);  // test implementation

            ~BPNN();

            //void InitNetwork();
            void InitNetwork(Genome* init_genome);

            void RunTrainingSession(double training_threshold);

            void RunTest();

            // Methods for setting explicit inputs and output targets
(optimizer)
            unsigned int getNumInputNodes()           { return
input_layer.size(); }
            void setOptimizerInputs(std::vector<double> inputs);
            void RunPropagation()   { RunPropagation(-1); CalcMSE(); } //
this is the special no-argument method, to be used only by the optimizer
            void calcOptimizerMSE(std::vector<double> outputs);
            double getMSE()   { return MSE; }
            InputType::Type getInputType(unsigned int input_idx) { return
input_layer[input_idx].getInputType(); }
            unsigned int getInputIdx(unsigned int input_idx) { return
input_layer[input_idx].getInputIdx(); }
            int getFrontRange(unsigned int input_idx) { return
input_layer[input_idx].getFrontRange(); }
```

APPENDIX D (continued)

```
            int getBackRange(unsigned int input_idx) { return
input_layer[input_idx].getBackRange(); }
            double getMaxInput(unsigned int input_idx) { return
input_layer[input_idx].getMaxInput(); }

      protected:

            Genome *genome;

            std::vector<InputNode> input_layer;
            std::vector<std::vector<BPNNNode>> hidden_layer;
            std::vector<OutputNode> output_layer;
            double learn_rate;
            double momentum;
            std::vector<int> test_idx;
            double MSE, avg_MSE, last_avg_MSE, MSE_change;
            unsigned int link_cnt, training_passes;

            void RunPropagation(int time_index);
            void CalcError(int time_index);
            void AdjustWeights();
            void CalcMSE();
            void ClearActivation();

            void RunTrainingPass();

            void RunTestPass();

      private:

      }; // end class InputNode

#endif // end BPNN_HPP_INCLUDED
```

APPENDIX D (continued)

BPNN.cpp

```
/**
* BPNN.cpp
*
* Implementation of the BPNN class.
* Basic BPNN
*
* Author: Jon Lessner 2012
*/

#include "BPNN.hpp"

using namespace std;

//------------------------------------------------------------------------

BPNN::BPNN()
//:more_inits(INIT_VAL)
{


}


//------------------------------------------------------------------------

BPNN::BPNN(int value)   // test implementation
//:,more_inits(INIT_VAL)
{

}

//------------------------------------------------------------------------

BPNN::~BPNN()
{
      //cout << "BPNN deleted: " << endl;
}

//------------------------------------------------------------------------

void BPNN::InitNetwork(Genome* init_genome)
{
      // TODO: Init BPNN values from genome

      genome = init_genome;

      int node_cnt = 0;

      cout << "initializing BPNN: ";
```

APPENDIX D (continued)

```
	cout << "\tGen: " << genome->getGenNum() << "\tMem: " << genome->getMemNum() << endl;

	// Create the input layer
	input_layer.clear();

	// Create the 3 "time of day" input nodes (3-bit binary, based on 8 3-hour chunks per day) (ex: chunk 3 = 6-9am = 010)
	for(unsigned int i = 0; i < 3; ++i)
	{
		InputChromosome time_input;
		time_input.input_type = InputType::TimeOfDay;
		time_input.input_idx = i;
		InputNode time_node(time_input);

		input_layer.push_back(time_node);
		++node_cnt;
	}

	// Create the rest of the inputs from the chromosomes in the genome
	for(unsigned int i = 0; i < genome->getNumInputs(); ++i)
//	for(unsigned int i = 0; i < 4; ++i)
	{
/*
		InputChromosome test_input;
		test_input.input_type = (InputType::Type) i;
		test_input.input_idx = i;
		test_input.data_front = 1;
		test_input.data_back = 1;
		InputNode data_node(test_input);
*/
		InputNode data_node(genome->getInput(i));
		input_layer.push_back(data_node);
		++node_cnt;
	}

	// Create the hidden layers
	for(unsigned int i = 0; i < hidden_layer.size(); ++i)
		hidden_layer[i].clear();
	hidden_layer.clear();
	for(unsigned int k = 0; k < genome->getHLayers(); ++k)
//	for(unsigned int k = 0; k < 2; ++k)
	{
		//cout << "Creating Hidden Layer #" << k << " with " << genome->getHLNodes(k) << " nodes:" << endl;
		std::vector<BPNNNode> temp_layer;
		for(int i = 0; i < genome->getHLNodes(k); ++i)
//		for(int i = 0; i < 2; ++i)
		{
			//cout << "\tCreating node #" << i << endl;
			BPNNNode layer_node;
			temp_layer.push_back(layer_node);
			++node_cnt;
		}
```

APPENDIX D (continued)

```
            hidden_layer.push_back(temp_layer);
      }

//#define   SYMP_OUTPUTS 8
      // Create the output layer
      output_layer.clear();
      for(unsigned int i = 0; i < SYMP_OUTPUTS; ++i)
      {
            OutputNode output_node(i);
            output_layer.push_back(output_node);
            ++node_cnt;
      }


      link_cnt = 0;
      // Hook up input layer to first hidden layer
      srand((unsigned int) genome->getRandSeed());
      for(unsigned int i = 0; i < input_layer.size(); ++i)
      {
            for(unsigned int j = 0; j < hidden_layer[0].size(); ++j)
            {
                  double weight = ((double(rand()) / double(RAND_MAX)) *
2.0f) -1.0f;      // randomized, and normalized in range -1.0 to 1.0
                  input_layer[i].PushDownlink(&(hidden_layer[0][j]), weight);
            // link node in inputlayer forward to a node in first hiddenlayer
                  hidden_layer[0][j].PushUplink(&(input_layer[i]));
                  // link the hiddenlayer node backward to the inputlayer one
                  ++link_cnt;
            }
      }

      // Hook up the hidden layers
      for(unsigned int k = 0; k < hidden_layer.size()-1; ++k)
      {
            for(unsigned int i = 0; i < hidden_layer[k].size(); ++i)
            {
                  for(unsigned int j = 0; j < hidden_layer[k+1].size(); ++j)
                  {
                        double weight = ((double(rand()) / double(RAND_MAX))
* 2.0f) -1.0f;     // randomized, and normalized in range -1.0 to 1.0

      hidden_layer[k][i].PushDownlink(&(hidden_layer[k+1][j]), weight); //
link node in layer k forward to a node in layer (k+1)

      hidden_layer[k+1][j].PushUplink(&(hidden_layer[k][i]));
      // link the second node backward to the first one
                        ++link_cnt;
                  }
            }
      }

      // Hook up the output layer
      for(unsigned int i = 0; i < hidden_layer.back().size(); ++i)
      {
```

APPENDIX D (continued)

```
            for(unsigned int j = 0; j < output_layer.size(); ++j)
            {
                  double weight = ((double(rand()) / double(RAND_MAX)) *
2.0f) -1.0f;      // randomized, and normalized in range -1.0 to 1.0
                  hidden_layer.back()[i].PushDownlink(&(output_layer[j]),
weight);    // link node in final hiddenlayer forward to a node in
outputlayer
                  output_layer[j].PushUplink(&(hidden_layer.back()[i]));
                  // link the outputlayer node backward to the hiddenlayer
one
                  ++link_cnt;
            }
      }

      // Get learning rate and momentum from genome
      learn_rate = genome->getLearnRate();
      momentum = genome->getMomentum();

      // Load test index selection from file
      test_idx.clear();
      stringstream test_idx_file_name;
      if(STATIC_TEST_SET)
            test_idx_file_name << "..\\Data\\TESTCASES_S.vec";
      else
            test_idx_file_name << "..\\Data\\TESTCASES_D.vec";
      ifstream inputTestFile(test_idx_file_name.str().c_str(),
ios_base::binary);
      while(true)
      {
            int test_idx_entry;
            inputTestFile.read((char*) &test_idx_entry, sizeof(int));

            if(inputTestFile.eof())
                  break;

            test_idx.push_back(test_idx_entry);

            //cout << "E#: " << test_idx.size() << ", IDX: " <<
test_idx.back() << endl;
      }
      inputTestFile.close();

      // Init training MSE tracker
      last_avg_MSE = 0.0f;
}

//-------------------------------------------------------------------

void BPNN::RunTrainingSession(double training_threshold)
{
      // Run a number of training passes
#define MAX_TRAINING_PASSES   20
      bool done = false;
      training_passes = 0;
```

APPENDIX D (continued)

```
	while(!done && training_passes < MAX_TRAINING_PASSES)
	{
		RunTrainingPass();
/*
		// Quick training
		if(MSE_change > 0.0f)
			done = true;
*/
		// Slow training
//		if(avg_MSE < MSE_TRAINING_THRESHOLD)
		if(avg_MSE < training_threshold)
			done = true;

		++training_passes;
	}

	genome->setTP(training_passes);
}

//---------------------------------------------------------------------

void BPNN::RunTest()
{
	RunTestPass();

	// Score
/*
	cout << "Scoring: " << endl;
	cout << "\tavg_MSE: " << avg_MSE << endl;
	cout << "\tgeneration: " << genome->getGenNum() << endl;
	cout << "\tlink_cnt: " << link_cnt << endl;
	cout << "\ttraining_passes: " << training_passes << endl;
	cout << (double)(avg_MSE * (genome->getGenNum() * genome->getGenNum()))
+ (((double)(link_cnt * training_passes)) / 10.0f) << endl;
	cout << 1.0f / ((double)(avg_MSE * (genome->getGenNum() * genome-
>getGenNum())) + (((double)(link_cnt * training_passes)) / 10.0f)) << endl;
*/
	// Calc score from the formula:
	// 1 / ((error * gen^2) + (links * trainings)/10)
	genome->setMSE(avg_MSE);
	genome->setScore(1.0f / ((double)(avg_MSE * (genome->getGenNum() *
genome->getGenNum())) + (((double)(link_cnt * training_passes)) / 10.0f)));
}

//---------------------------------------------------------------------

void BPNN::setOptimizerInputs(std::vector<double> inputs)
{
	for(unsigned int i = 0; i < inputs.size(); ++i)
		input_layer[i].SetInputVector(inputs[i]);
}

//---------------------------------------------------------------------
```

APPENDIX D (continued)

```
void BPNN::calcOptimizerMSE(std::vector<double> outputs)
{
	for(unsigned int i = 0; i < outputs.size(); ++i)
		output_layer[i].CalcOptimizerError(outputs[i]);
	CalcMSE();
}


//------------------------------------------------------------------------

void BPNN::RunPropagation(int time_index)
{
	//cout << "Running propagation for time index: " << time_index << endl;

	if(time_index != -1)	// If this is NOT an optimizer run (-1),
initialize the input Activation values from the datastream vectors
	{
		// Zero Activation values
		ClearActivation();

		// Initialize input nodes from datastream vectors
		for(unsigned int i = 0; i < input_layer.size(); ++i)
			input_layer[i].ReadInputVector(time_index);
	}

	// Propagate signal from input to first hidden layer
	for(unsigned int i = 0; i < input_layer.size(); ++i)
		input_layer[i].FireNeuron();

	// Propagate signal through hidden layers, and to output layer
	for(unsigned int k = 0; k < hidden_layer.size(); ++k)
	{
		for(unsigned int i = 0; i < hidden_layer[k].size(); ++i)
		{
			// Apply Sigma filter
			hidden_layer[k][i].SigmaFilter();
			// Propagate signal to next layer
			hidden_layer[k][i].FireNeuron();
		}
	}

	// Sigma filter output layer
	for(unsigned int i = 0; i < output_layer.size(); ++i)
		output_layer[i].SigmaFilter();
}

//------------------------------------------------------------------------

void BPNN::CalcError(int time_index)
{
	//cout << "CalcError: " << endl;
	// Calc difference between target output and actual activation for
output layer
	for(unsigned int node = 0; node < output_layer.size(); ++node)
```

APPENDIX D (continued)

```
	{
		output_layer[node].CalcError(time_index);
		//cout << "\terror for node #: " << node << " = " <<
output_layer[node].getErrorValue() << endl;
	}

	// Backprop error to all hidden layer (starting at last, and stepping
backward to first)
	for(int layer = (int)hidden_layer.size()-1; layer >= 0; --layer)
	{
		//cout << "hidden layer: " << layer << endl;

		for(unsigned int node = 0; node < hidden_layer[layer].size();
++node)
		{
			hidden_layer[layer][node].CalcError(time_index);
			//cout << "\terror for node #: " << node << " = " <<
hidden_layer[layer][node].getErrorValue() << endl;
		}
	}
/*
	//cout << "input layer: " << endl;
	for(unsigned int node = 0; node < input_layer.size(); ++node)
	{
		input_layer[node].CalcError(time_index);
		//cout << "\terror for node #: " << node << " = " <<
input_layer[node].getErrorValue() << endl;
	}
*/
}

//------------------------------------------------------------------------

void BPNN::AdjustWeights()
{
	// Adjust hidden layer(s) weights
	for(int layer = (int)hidden_layer.size()-1; layer >= 0; --layer)
	{
		//cout << "hidden layer #" << layer << ": " << endl;

		for(unsigned int node = 0; node < hidden_layer[layer].size();
++node)
		{
		//cout << "\tadjustment for node #" << node << ": ";
			hidden_layer[layer][node].AdjustWeight(learn_rate,
momentum);
		}
	}

	// Adjust input layer weights
	//cout << "input layer: " << endl;
	for(unsigned int node = 0; node < input_layer.size(); ++node)
	{
		//cout << "\tadjustment for node #" << node << ": ";
```

APPENDIX D (continued)

```
			input_layer[node].AdjustWeight(learn_rate, momentum);
		}

}

//-------------------------------------------------------------------

void BPNN::CalcMSE()
{
		MSE = 0.0f;
		for(unsigned int i = 0; i < output_layer.size(); ++i)
		{
			double node_error = output_layer[i].getRawErrorValue();
			//double node_error = output_layer[i].getErrorValue();
			MSE += node_error * node_error;
		}
		MSE /= output_layer.size();
}

//-------------------------------------------------------------------

void BPNN::ClearActivation()
{
		for(unsigned int i = 0; i < input_layer.size(); ++i)
			input_layer[i].ResetActivation();
		for(unsigned int k = 0; k < hidden_layer.size(); ++k)
		{
			for(unsigned int i = 0; i < hidden_layer[k].size(); ++i)
				hidden_layer[k][i].ResetActivation();
		}
		for(unsigned int i = 0; i < output_layer.size(); ++i)
			output_layer[i].ResetActivation();
}

//-------------------------------------------------------------------

void BPNN::RunTrainingPass()
{
		unsigned int max_training_idx = (unsigned int) (test_idx.size() *
(100.0f/TEST_SET_PERCENT));
		unsigned int cur_test_idx_ptr = 0;

		//for(unsigned int i = 0; i < test_idx.size(); ++i)
			//cout << "E#: " << i << ", IDX: " << test_idx[i] << endl;

		//cout << "max_training_idx: " << max_training_idx << endl;

		double total_MSE = 0.0f;
		unsigned int training_cnt = 0;
		// Run training for index 1 to MAX (not 0 to MAX-1, because it makes
the math easier)
		for(unsigned int cur_training_idx = 1; cur_training_idx <=
max_training_idx; ++cur_training_idx)
		{
```

APPENDIX D (continued)

```
            bool run_prop = true;

            if((cur_test_idx_ptr < test_idx.size()) &&
               (test_idx[cur_test_idx_ptr] == cur_training_idx))
            {
                  //cout << "Skipping index #" << cur_training_idx << "
because it's a test case" << endl;
                  ++cur_test_idx_ptr;
                  run_prop = false;
            }

//#define TRAINING_CAP  5
#define TRAINING_CAP    max_training_idx
            if(cur_training_idx > TRAINING_CAP)
                  run_prop = false;

            if(run_prop)
            {
                  unsigned int time_stamp = (cur_training_idx * CHUNK_SIZE) -
1; // ex: Index #1 would convert to time_stamp 2, encompasing output from
hour 0 to 2

                  RunPropagation(time_stamp);

                  CalcError(time_stamp);

                  AdjustWeights();

                  CalcMSE();

                  total_MSE += MSE;
                  ++training_cnt;
/*
                  if(cur_training_idx >= 680 && cur_training_idx <= 720)
                  {

                        for(unsigned int i = 0; i < input_layer.size(); ++i)
                              cout << input_layer[i].getActivationValue() <<
endl;

                        for(unsigned int i = 0; i < output_layer.size(); ++i)
                              //cout << output_layer[i].getErrorValue() <<
endl;
                              cout <<
output_layer[i].GetSympOutputVector(time_stamp) << " - " <<
output_layer[i].getActivationValue() << " ->\t" <<
output_layer[i].getErrorValue() << endl;
                  }

                  if(cur_training_idx == TRAINING_CAP)
                        cout << "cur_training_idx: " << cur_training_idx <<
",\ttime_stamp: " << time_stamp << "\tMSE: " << MSE << endl;
*/
            }
```

APPENDIX D (continued)

```
	}

	avg_MSE = total_MSE / training_cnt;
	//cout << "Avg MSE: " << avg_MSE;

	if(last_avg_MSE != 0.0f)
	{
		MSE_change = avg_MSE - last_avg_MSE;
		//cout << "\tchange: " << MSE_change << endl;
	}
	//else
		//cout << endl;
	last_avg_MSE = avg_MSE;
}

//-------------------------------------------------------------------

void BPNN::RunTestPass()
{
	unsigned int max_training_idx = (unsigned int) (test_idx.size() *
(100.0f/TEST_SET_PERCENT));
	unsigned int cur_test_idx_ptr = 0;

	//for(unsigned int i = 0; i < test_idx.size(); ++i)
		//cout << "E#: " << i << ", IDX: " << test_idx[i] << endl;

	//cout << "max_training_idx: " << max_training_idx << endl;

	double total_MSE = 0.0f;
	//unsigned int training_cnt = 0;

	// Run test indexes
	for(unsigned int cur_test_idx = 0; cur_test_idx < test_idx.size();
++cur_test_idx)
	{
		unsigned int time_stamp = (test_idx[cur_test_idx] * CHUNK_SIZE) -
1; // ex: Index #1 would convert to time_stamp 5, encompasing output from
hour 0 to 5

		RunPropagation(time_stamp);

		CalcError(time_stamp);

		//AdjustWeights();

		CalcMSE();

		total_MSE += MSE;
		//++training_cnt;
	}

	avg_MSE = total_MSE / (double) test_idx.size();
	//cout << "Test Avg MSE: " << avg_MSE;
}
```

APPENDIX D (continued)

```
/*
	// Test prints
	cout << "input layer: " << endl;
	for(unsigned int i = 0; i < input_layer.size(); ++i)
	{
		cout << "node " << i << ": " << endl;
		for(unsigned int j = 0; j < input_layer[i].GetDownlinkCnt(); ++j)
			cout << "\tweight " << j << ": " <<
input_layer[i].GetDownlinkWeight(j) << endl;
	}
	cout << endl << endl;

	for(unsigned int k = 0; k < hidden_layer.size(); ++k)
	{
		cout << "hidden layer " << k << ": " << endl;
		for(unsigned int i = 0; i < hidden_layer[k].size(); ++i)
		{
			cout << "node " << i << ": " << endl;
			for(unsigned int j = 0; j <
hidden_layer[k][i].GetDownlinkCnt(); ++j)
				cout << "\tweight " << j << ": " <<
hidden_layer[k][i].GetDownlinkWeight(j) << endl;
		}
		cout << endl << endl;
	}
*/
```

VII. CITED LITERATURE

1. Beck, David E.: Medical Treatment of Ulcerative Colitis. *Clinical Colon Rectal Surgery*. Feb 2004.
   Web: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2780076/

2. BoxCar 2D: Computation Intelligence Car Evolution Using Box2D Physics (v3.2).
   Web: http://boxcar2d.com/

3. Bradley, Peter: The XOR Problem and Solution. *Consortium on Cognitive Science Instruction*.
   Web:
   http://www.mind.ilstu.edu/curriculum/artificial_neural_net/xor_problem_and_solution.php

4. Cheshire Engineering Corportation: Basic Concepts for Neural Networks.
   Web: http://www.cheshireeng.com/Neuralyst/nnbg.htm

5. Genetics Home Reference: Ulcerative colitis. Oct 2012.
   Web: http://ghr.nlm.nih.gov/condition/ulcerative-colitis

6. Gleeson, Michael, Lancaster, Graeme I., Bishop Nicolette C.: Nutritional Strategies to Minimise Exercise-Induced Immunosuppression in Athletes*. Canadian Journal of Applied Physiology*. 2001.
   Web: http://www.nrcresearchpress.com/doi/abs/10.1139/h2001-039#.U5JUPfmwK8E

7. Graupe, Daniel: Principles of Artificial Neural Networks (2nd Edition), *World Scientific Publishing*, 2007, pp.59-94, 249-280

8. Hsiung, Samuel: Multilayer Feedforward Network and the Backpropagation Algorithm. *generation5.org*
   Web: http://www.generation5.org/content/1999/nn_bp.asp

9. Johnson, Steven: “Emerging Techonology: Are you ready for computers that speed up the process of evolution and teach themselves to think?”, *Discover Magazine*, Aug 2003.
   Web: http://discovermagazine.com/2003/aug/feattech

10. Kaznatcheev, Artem: Machine Learning and Prediction Without Understanding. *Theory, Evolution, and Games Group*, (wordpress.com).
    Web: http://egtheory.wordpress.com/2013/06/05/prediction-vs-understanding/

11. Kelly, Ciaran P., Pothoulakis, Charalabos, LaMont, Thomas J.: Clostridium difficile Colitis. *The New England Journal of Medicine*. 1994.
    Web: http://www.nejm.org/doi/pdf/10.1056/NEJM199401273300406

12. Kirkpatrick, S.,Gelatt, C.D. Jr., Vecchi, M.P. : "Optimization by Simulated Annealing", *Science*, 1983, v.220, No.4598, pp.671-680.
    Web: http://mkweb.bcgsc.ca/papers/kirkpatrick-simulatedannealing.pdf

13. Kohen, Philipp: Combining Genetic Algorithms and Neural Networks. Masters thesis, University of Tennessee, Knoxville pp.52-56
Web: http://homepages.inf.ed.ac.uk/pkoehn/publications/gann94.pdf

14. Kutza, Karsten: Neural Networks Source. *Japan Singapore AI Centre*.
Web: http://www.ip-atlas.com/pub/nap/nn-src/

15. McGraith, J., McDonald, J.W., McDonald, J.K.: Transdermal Nicotine for Induction of Remission in Ulcerative Colitis. *National Center for Biotechnology Information*, (PubMed.gov).
Web: http://www.ncbi.nlm.nih.gov/pubmed/15495126

16. Montana, David J. and Davis, Lawrence: Training Feedforward Neural Networks Using Genetic Algorithms. *BBN Systems and Technologies Corp*.
Web: http://ijcai.org/Past%20Proceedings/IJCAI-89-VOL1/PDF/122.pdf

17. Norvig, P. and Russel, S.: Artificial Intelligence: A Modern Approach (2nd Edition), *Pearson Education*, 2003, pp.563-587

18. Pedersen, B.K. and Toft, A.D.: Effects of exercise on lymphocytes and cytokines. *British Journal of Sports Medicine*, Mar. 9, 2000.
Web: http://pubmedcentralcanada.ca/pmcc/articles/PMC1724218/

19. Schemenauer, Neil: Rudimentary BPNN implementation.
Web: http://arctrix.com/nas/python/bpnn.py

VIII. VITA

NAME: Jonathan Paul Lessner

EDUCATION: B.S., Computer Science (Theoretical), Northern Illinois University, DeKalb, Illinois, 2004

M.S., Computer Science, University of Illinois at Chicago, Chicago, Illinois, 2014